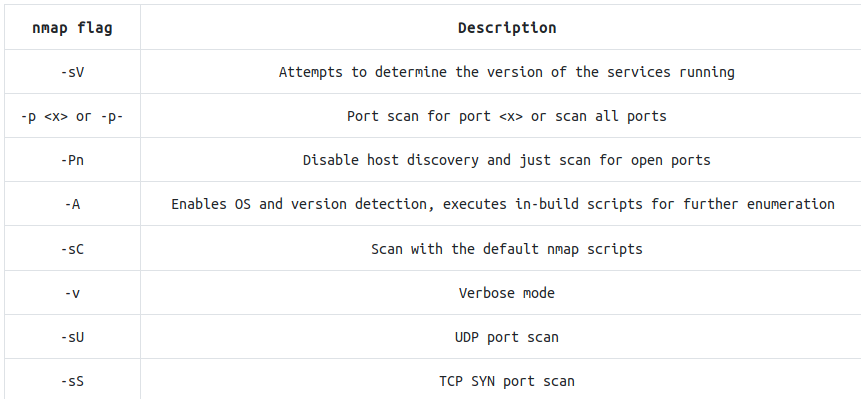
# TryHackMe Write UP for Vulnersity Room

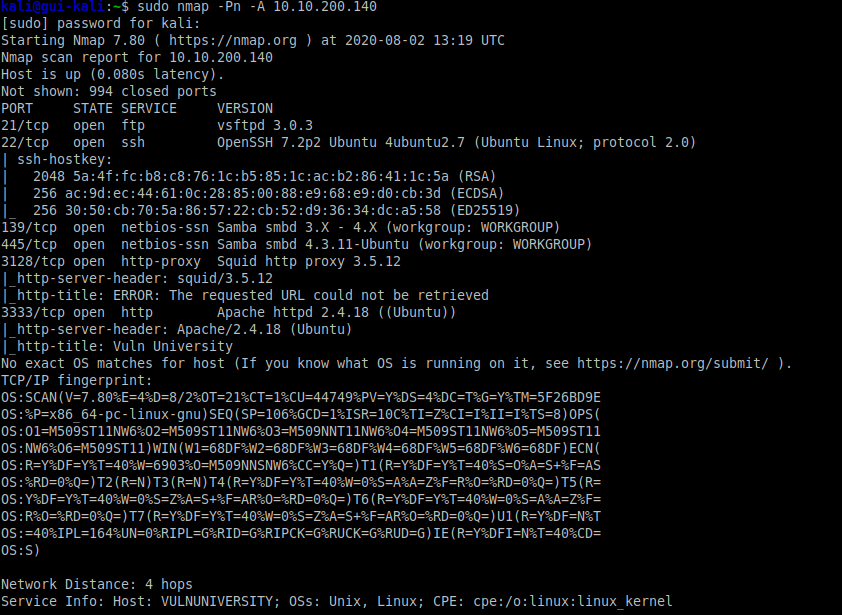
Reference room:

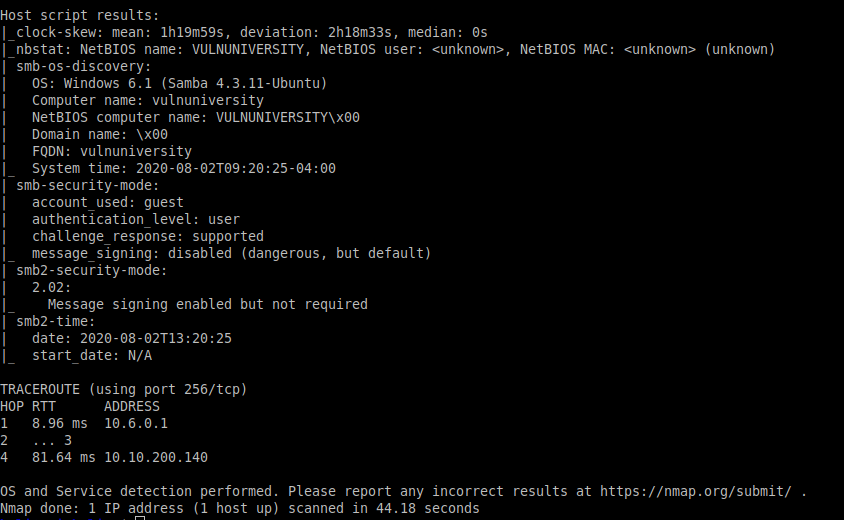
<https://tryhackme.com/room/vulnversity>

# Reconnaissance



Enumerate target, by default scan top 1000 ports. For the flag option see above.

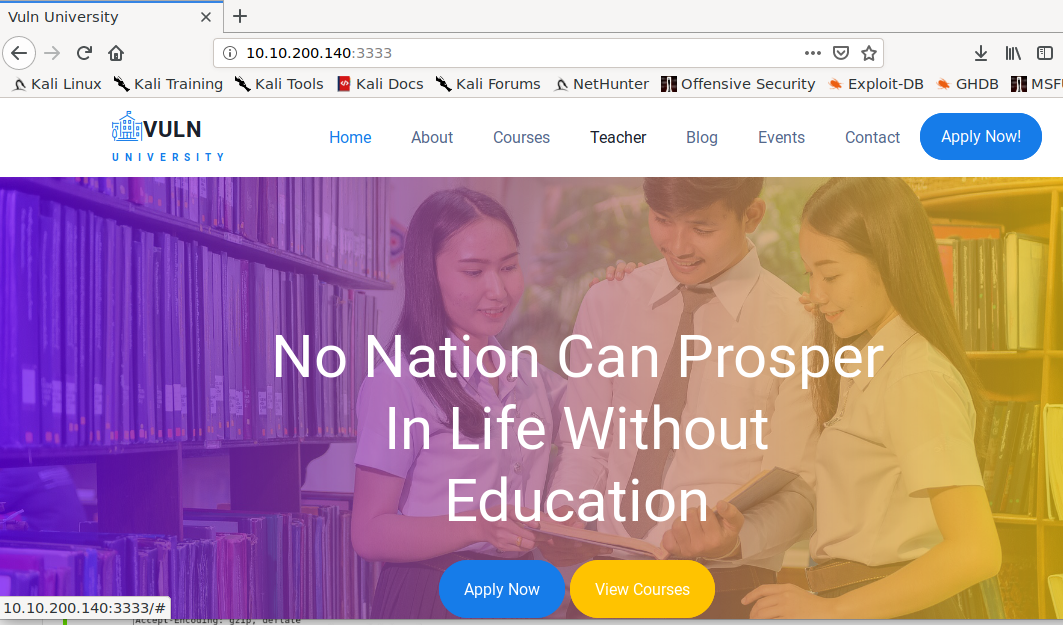




Output shows:

* six ports are open.
* http-proxy Squid http proxy 3.5.12
* Web Server running on port 3333
* OS is Linux, Ubuntu flavor

Checking out the web port on port 3333



You might want to scan all ports.

Nmap -sC -sV -oA Vulnersity\_fullscan -p- 10.10.200.140

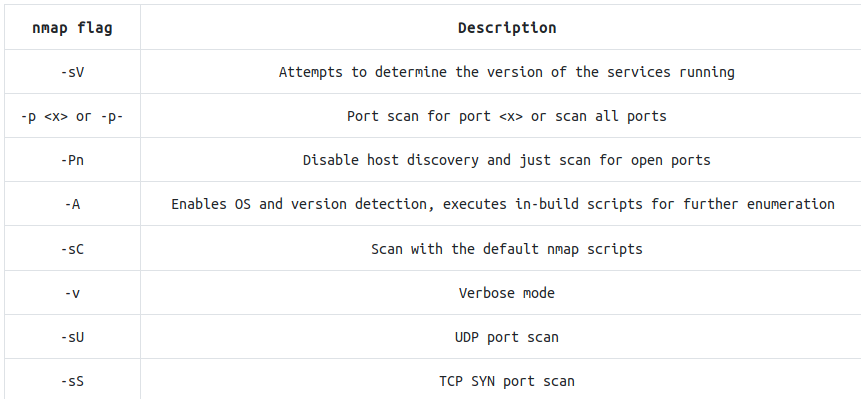
“-sC” full script

“-p-” ports 1-65535

# Enumerating Web Directory

## Locating Directory using GoBuster

On Kali **sudo apt install gobuster**



## 

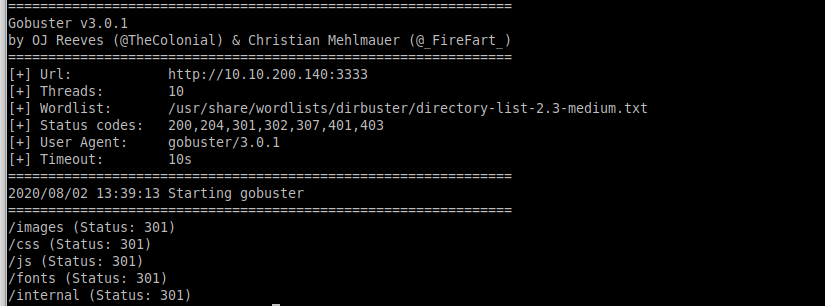
gobuster dir -u http://10.10.200.140:3333 -w \ /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

Note: mileage vary depending on wordlist

Wordlist:

/usr/share/wordlists/dirb/common.txt

/usr/share/wordlists/dirbuster/directory-list-2.3.medium.txt



HTTP Status[[1]](#footnote-0) Code: 301 permanent redirect.

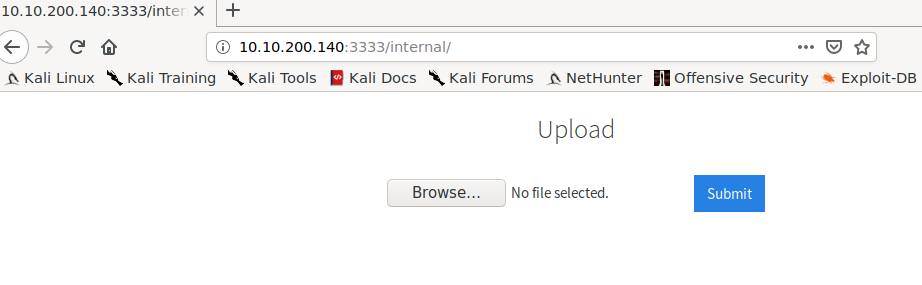
1. Informational responses (100–199),
2. Successful responses (200–299),
3. Redirects (300–399),
4. Client errors (400–499),
5. and Server errors (500–599).

# Compromise the WebServer

I used Burp Community Edition version 1.7.x as the later version seems to have taken the spider option away. I use FoxProxy for my proxying traffic to Burp.

From the previous gobuster scan, we note the “/internal” and web port: 3333.

Checking out: <http://10.10.200.140:3333/internal> - it seems to be an upload page.



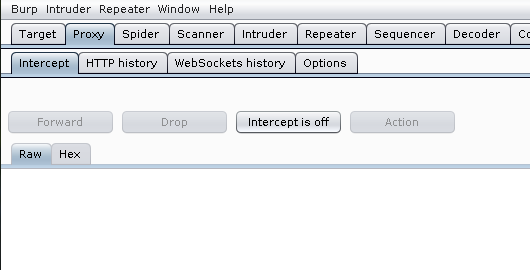
Let’s use BurpSuite Intruder to enumerate file extension

First I need to generate a payload response for uploading file action so I can send it to Intruder.

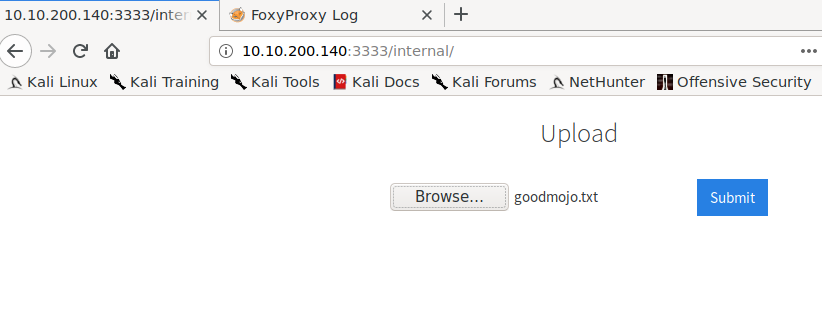
Making sure my FoxyProxy is proxying only the target web server traffic to BurpSuite.

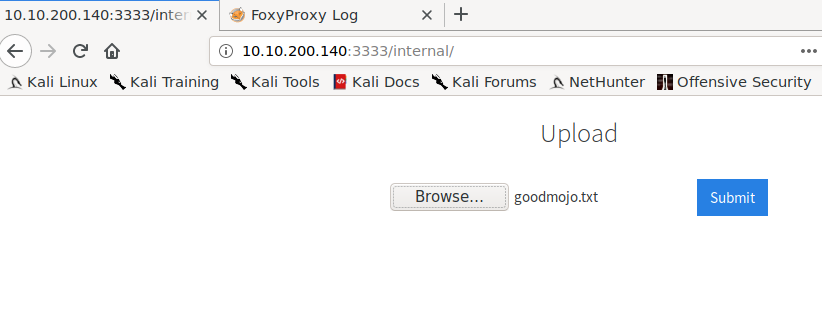


Start BurpSuite, make sure the interceptor is off and the Burp proxy is on localhost port 8080.

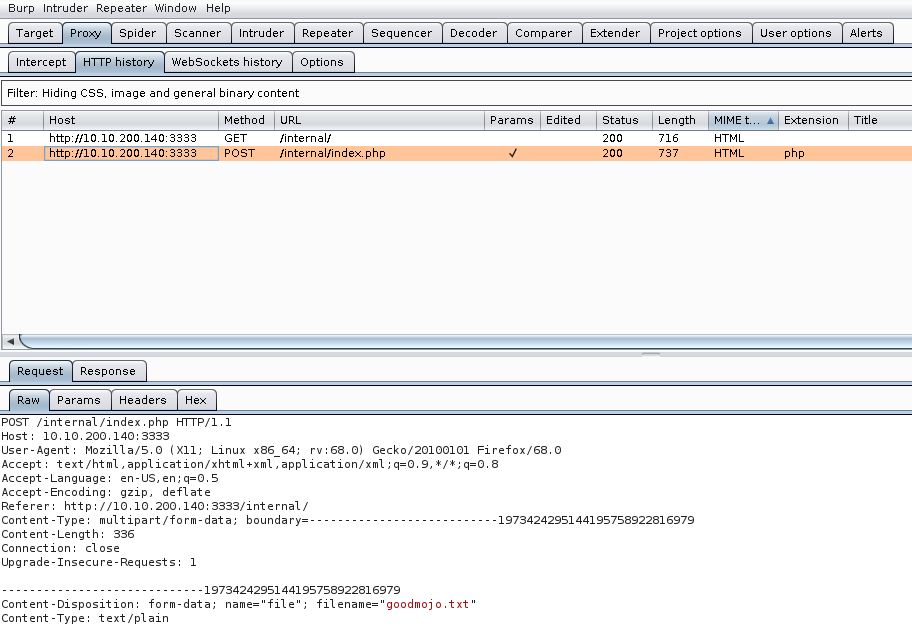


Back to the upload page. Upload a file and track the upload in BurpSuite.

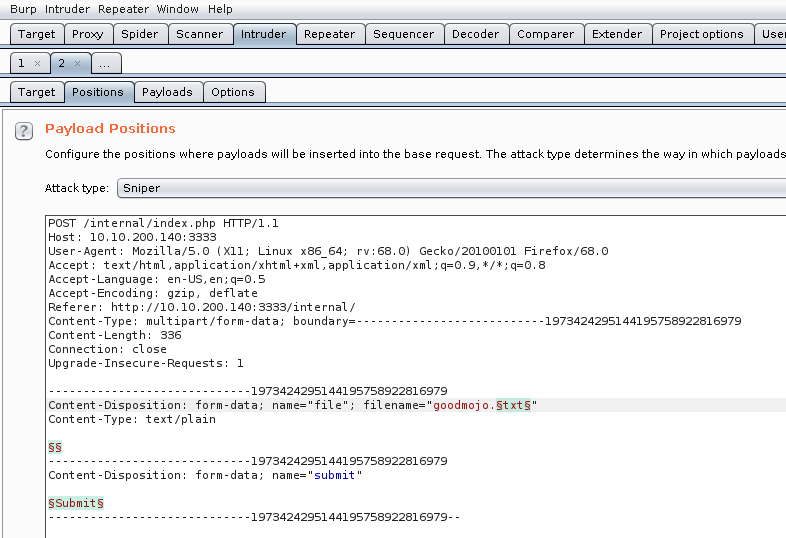




Back to BurpSuite Track the upload. Send this to Intruder, shortcut key to send to intruder, “ctr+I”



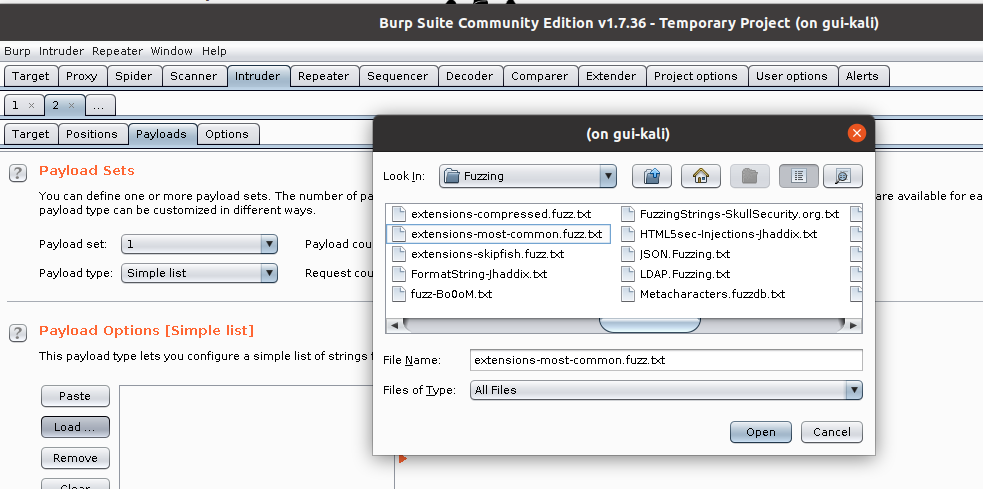
Fuzz extension. Go to Intruder, position tab.

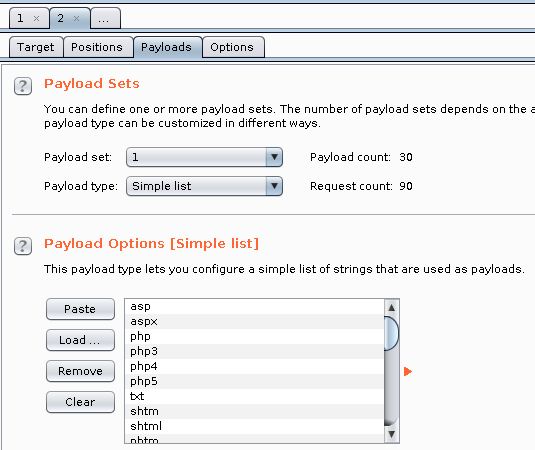


Go to the Payload tab to select and load payload (file extension to Fuzz).

On Kali - sudo apt install seclists

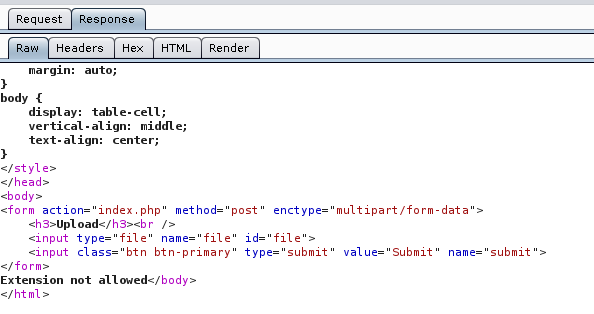
Load wordlist: **/usr/share/seclists/Fuzzing/extensions-most-common.fuzz.txt**



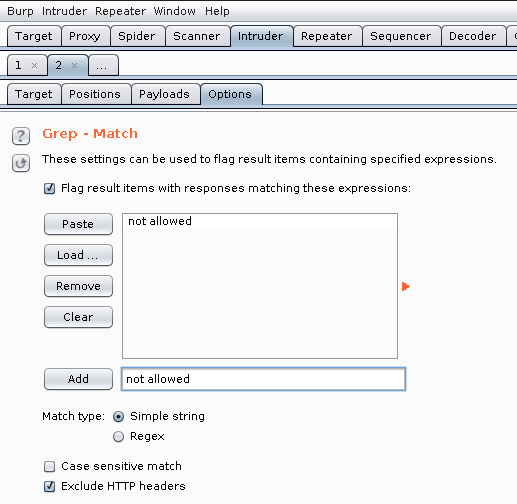


It is ready to run, but I want to tweak and flag the output as failure.

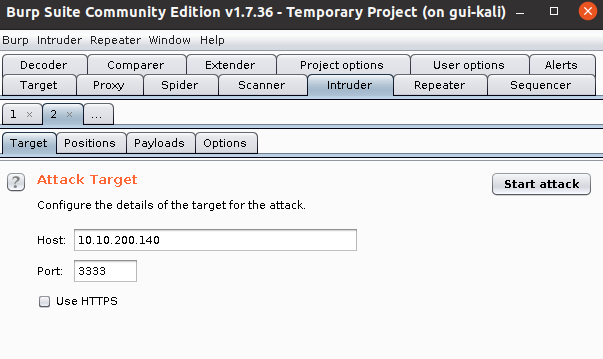
You can also eyeball the bytesize, and observer patterns of bytes and note if it fails or passes.

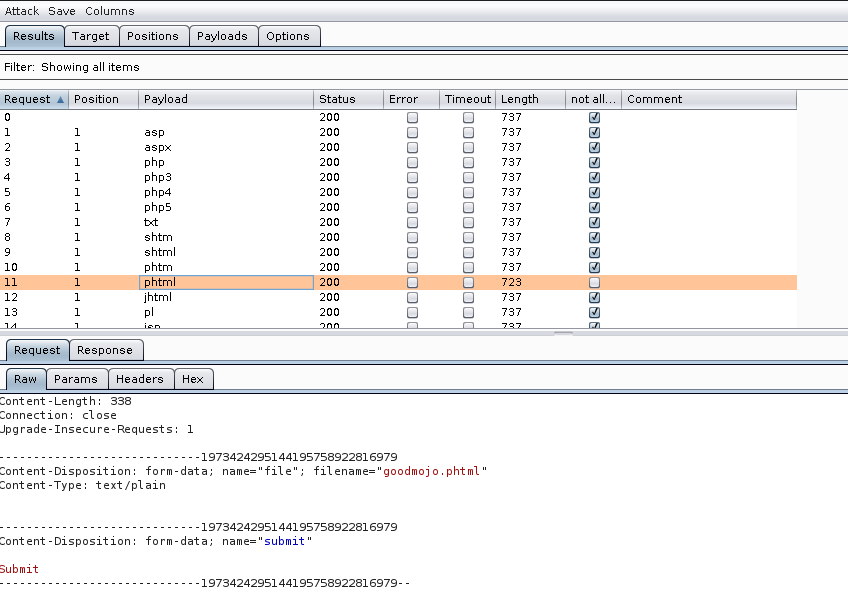


Set Grep Match to look for failure, “not allowed”



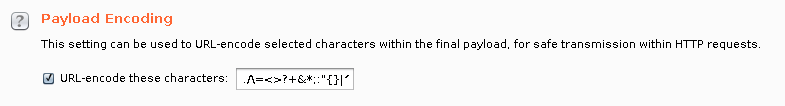
Kick off the Fuzz, click on “start attack” located on the upper right corner.



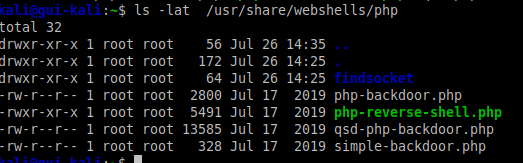


Note: the Length on request #11 differ and it did not match the failure pattern we set up earlier. The web server is accepting the .phtml extension as noted in result, the not allow column is un-check (not flag as fail).

## Something you should note

If you have this option turned on in Burp under the payload tab and your payload file contains a prefix with “.” then extension format like “.php”. When this is sent to the web server, it doesn’t appear to parse/decode properly as I am getting all failure even with the working file extension “.phtml”. I am not sure if this has something to do with how the web server is set up. To get this to work, just uncheck it.

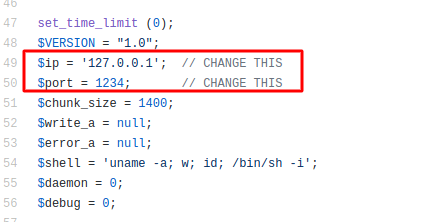
Now that we know the type of file the web server is accepting for upload. Let’s look for a reverse shell. On Kali, /usr/share/webshells.



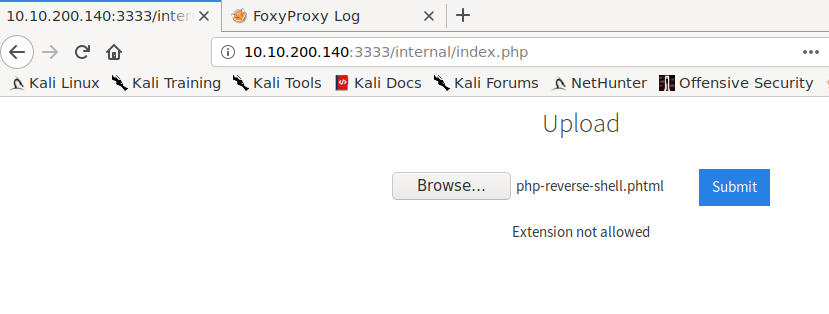
Or you can use the instruction on the Vulnersity room:

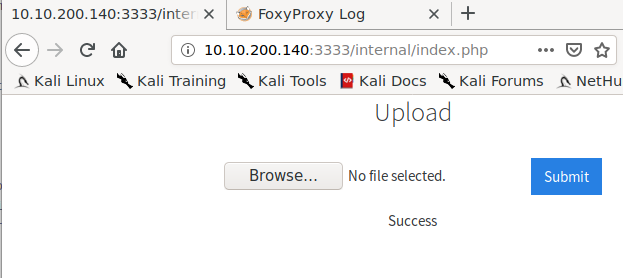
Download the following reverse PHP shell [here](https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php).

Change the script setting to point back to the host you’re expecting the revershell to call back to. Optionally, you can keep the ports or change.



After tweaking the reverse shell, Upload the reverse shell.

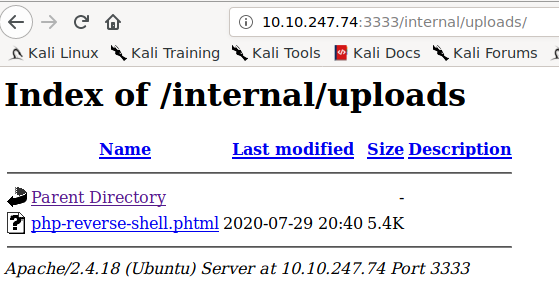




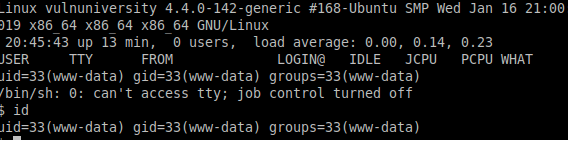
Note: My target IP is different from above because my target VM has expired and I have started up a new target VM for this room.

Set up netcat listener: nc -lvnp 1234

Per instruction from Vulnersity room, Upload your shell and navigate to http://<ip>:3333/internal/uploads/php-reverse-shell.phtml - This will execute your payload.

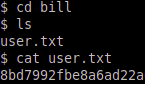


Payload is executed when the uploaded reverse shell is executed.





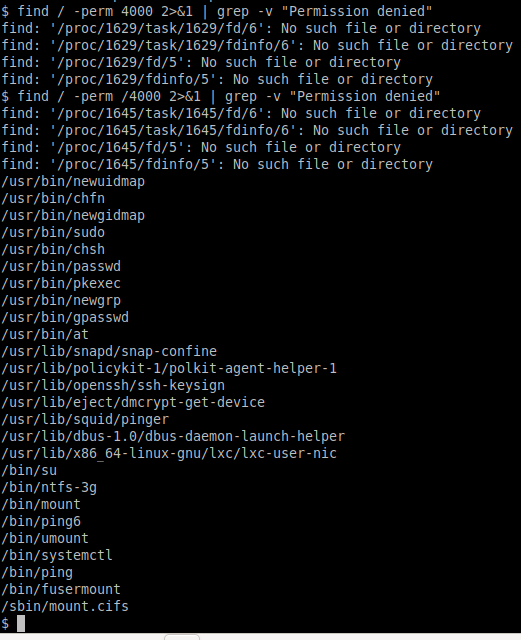
Partial flag shown:



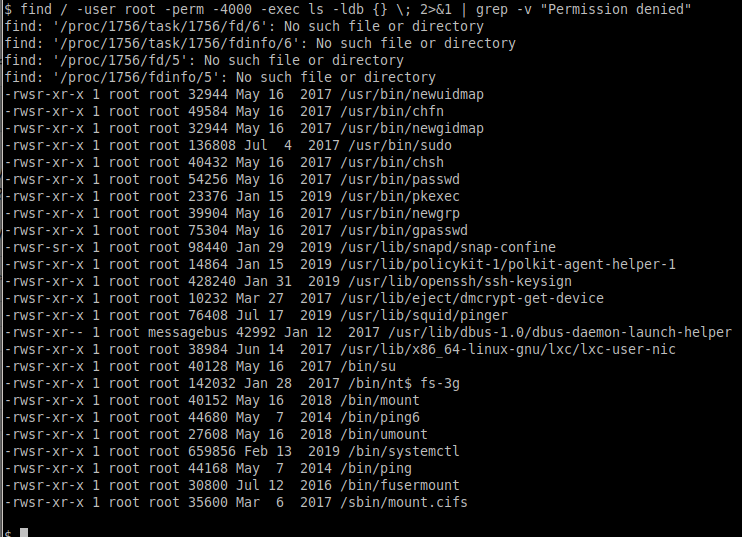
# Privilege Escalation

SETUID allows executable to be executed with the privilege of the file’s owner. In this case if the file is owned by root and SETUID bit is set, then when the executable is run, it is run as root. Search on the web for SETUID bit, 4000 is set SETUID. Great reference to [SETUID](https://www.linuxnix.com/suid-set-suid-linuxunix/).

Reference to [SGID](https://www.linuxnix.com/sgid-set-sgid-linuxunix/).



Just another way to find SETUID which listed out the file permission



I got this reference from reading the writeup from others.

Use [PEASS](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite) to perform privilege escalation enumeration. A good source of reference and tool if you need to get up to speed with privilege escalation on Linux, Windows, and MAC.

I have used systemctl to start and stop service. I have not used to roll my own service.

I have checked the permission on my current system and systemctl does not appear to have the setuid bit set.

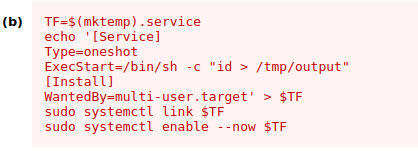
Systemctl is how you control [systemd](https://www.linode.com/docs/quick-answers/linux-essentials/what-is-systemd/) init server, stop and start your Linux service.

**Our target system allows for any logged user to create a system service and run it as root. The SETUID bit set and the systemctl executable is owned by root allows any user to do this.**

Create the Linux service script.

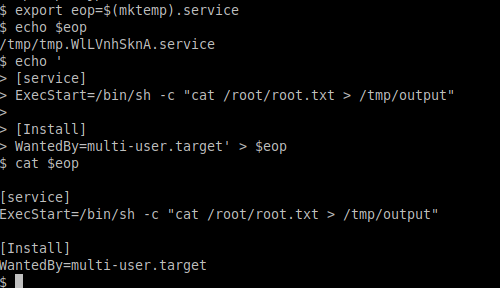
Yet another reference by fellow WriteUP. Another tool to add to my tool belt.

[GTFOBins](https://gtfobins.github.io/#+suid) a cheat sheet for binary hacks. From the list, find [systemctl](https://gtfobins.github.io/gtfobins/systemctl/). The option I use to model the service script is b.



I want to confirm what mktemp output is, so echo $eop appears to some random file name was generated. Normally the output of the configuration goes to /etc/systemd/system, but the current user we login as does not have permission to write to the “/etc/systemd/system” dir. We get around this by using the echo command.

When the service starts it calls the ExecStart, which cat the file root.txt owns by root to /tmp/output. The “[Install]” section sets the state/runlevel the service will run.



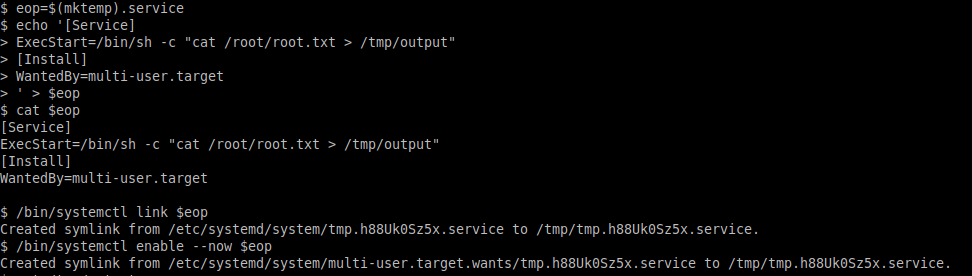
I wonder if rolling the configuration in a temp using an editor works as well?

Something to try when I have time.

/bin/systemctl link $eop - makes the config file available for systemctl command even though it is outside of the standard search path.

/bin/systemctl enable --now $eop, this run and reload the daemon.

Output is sent to “/tmp/output”



Flat that was in “/root/root.txt” which we did not have access to, which is now in “/tmp/output” and is readable by non-root users.

xxxxxxx

1. HTTP Status Code reference: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status> [↑](#footnote-ref-0)