PENETRATION TEST

REPORT AND FINDINGS

ON NEW YORK FLANKEES

COMPILED BY:

ALUMASA STEPHEN

Introduction

TryHackMe is an online platform that is dedicated to learning and teaching cybersecurity skills through interactive labs and challenges. The target audience of thm is broad and includes anyone who is curious about hacking and cybersecurity.

Statement of Confidentiality

The contents of this document have been developed by Stephen Alumasa. No portion of this document may be communicated, reproduced,

copied or distributed without prior consent.

The contents of this document do not constitute legal advice. TryHackMe’s offer of services that relate to compliance,

litigation or other legal interests are not intended as legal counsel and should not be taken as such. The assessment

detailed herein is against a fictional company for training and examination purposes.

# SCOPE

This project is aimed at breaching the security of the blog of a mischievous sorcerer{stefan}. The blog is fortified by enigmatic endpoints and guarded by cunning spells of protection. As a rogue adventurer {red teamer} I am aiming to dismantle his digital stronghold.

Now the key skills going to used in this particular project are:

* Network scanning(nmap)
* Web application security
* Vulnerability exploitation
* Cryptography(tokens and cookies)
* Command injection
* Linux command line (gaining shell access)
* Privilege escalation(gaining root access)
* Reconnaissance(information gathering)

## Assessment

My exploration of Stefan's digital stronghold during the New York Flankees challenge revealed a number of weaknesses that, if exploited by a malicious attacker, could have dire consequences for the blog's security.

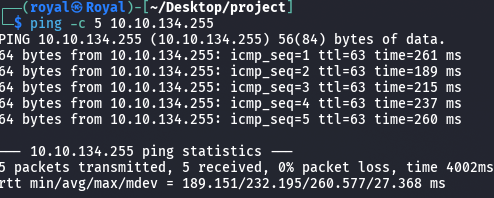
While the challenge description alluded to 'cunning spells of protection’, the reality was a series of exploitable vulnerabilities.

Detailed walkthrough

I performed the following steps to fully compromise Stefan’s blog:

* After navigating through the site as a normal user, i ran gobuster to find possible directories.there are other tools used to bruteforce directories such as dirsearch which is python based, Dirb among others.
* Among the found directories were /debug.html. This directory brought me to the debug page of stefan’s blog.
* I had to do some research on the verbose error. I found out that the error is actually called “padding oracle”. After deeper research i found that to exploit this there were various tools to use such as padre and padbuster which is a perl script for automating padding oracle attacks.
* I went back to the debug page and viewed the source code.something enticing was here. There was an endpoint script and a code like token-ish. The script describes that we should paste the url plus the token at the end. But this just gave a dead end as it shows a ‘custom authentication success’ message. When there is a missing number the message displayed is ‘Decryption error’.
* From the research, we can use padre to decrypt using lowercase hexadecimals.
* I was able to successfully decrypt the username and password for the Admin page.
* After accessing the admin page, there is a command bar. When i entered a command like ‘whoami’ , it returns a response ‘ok’ and nothing more.
* I had a hard time processing this. So then I decided to make a reverse shell. I started a http python server on port 80.
* Using Burp suite , i used curl to fetch the shell file from my IP url and download it to the target in the /tmp directory which is usually executable by all users.
* I opened a listening port on port 8000 so that I could get shell access to the vulnerable blog.
* After gaining shell access , mandatory to stabilize it so that it isn’t lost using the command “python3 -c 'import pty;pty.spawn("/bin/bash")”
* I was able to escalate my privilege and found a suspicious docker file. This file led to final compromise of the system.

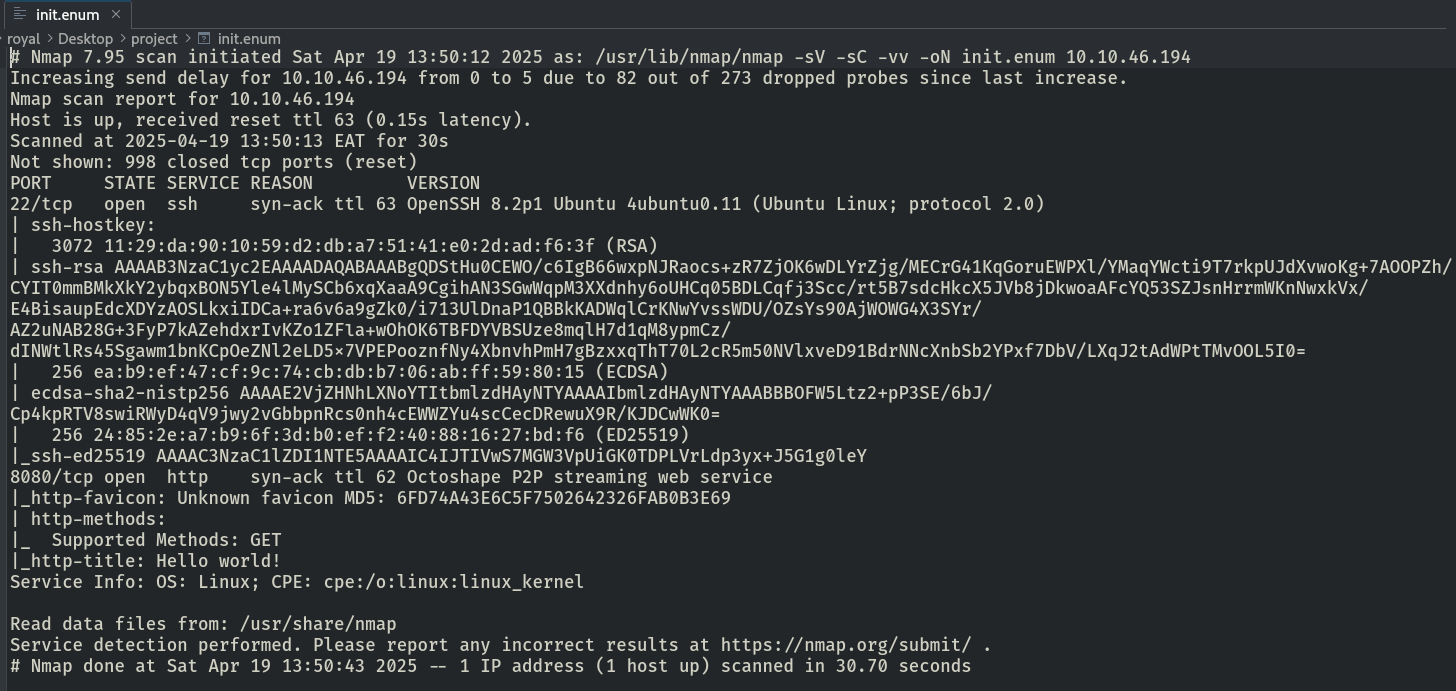
# Detailed steps:

When i connected successfully to the target IP, i ran a ping scan to ensure that the target host is up 

*Figure 1: ensuring host is up*

Next I ran an nmap scan for initial enumeration and directed the output to a file named init.enum.



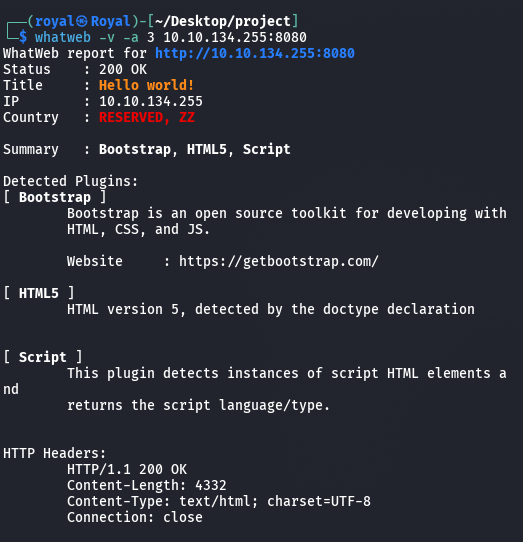


*Figure 3: Initial enumeration*

There are other tools used for enumeration such as metasploit,whatweb,wireshark and Nessus.

* Nessus is a popular vulnerability scanner that helps in finding security flaws, vulnerabilities, and misconfigurations in networks. Its automated scans evaluate systems for any flaws, assisting organizations in effectively prioritizing and addressing risks
* Wireshark is a packet analyzer that captures and analyzes network traffic, allowing for detailed examination of communication patterns and the identification of potential security risks. Mostly used in real time.
* WhatWeb specializes in identifying web technologies and services.

I tried using whatweb and this was my result, still learning how to use it

*figure 4: whatweb enum*

After Network enumeration i found out that the blog ran on a HTTP server on port 8080.

There was also a shell service on port 22.

## PORT 22:[OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)]

* **CVE-2025-26465:** Allows man-in-the-middle attacks when VerifyHostKeyDNS is enabled.
* **CVE-2025-26466:** Enables pre-authentication denial-of-service attacks.
* **Crucially, Ubuntu has released updates (e.g., USN-7270-1) to address these, so updating your Ubuntu 20.04 system is essential.**

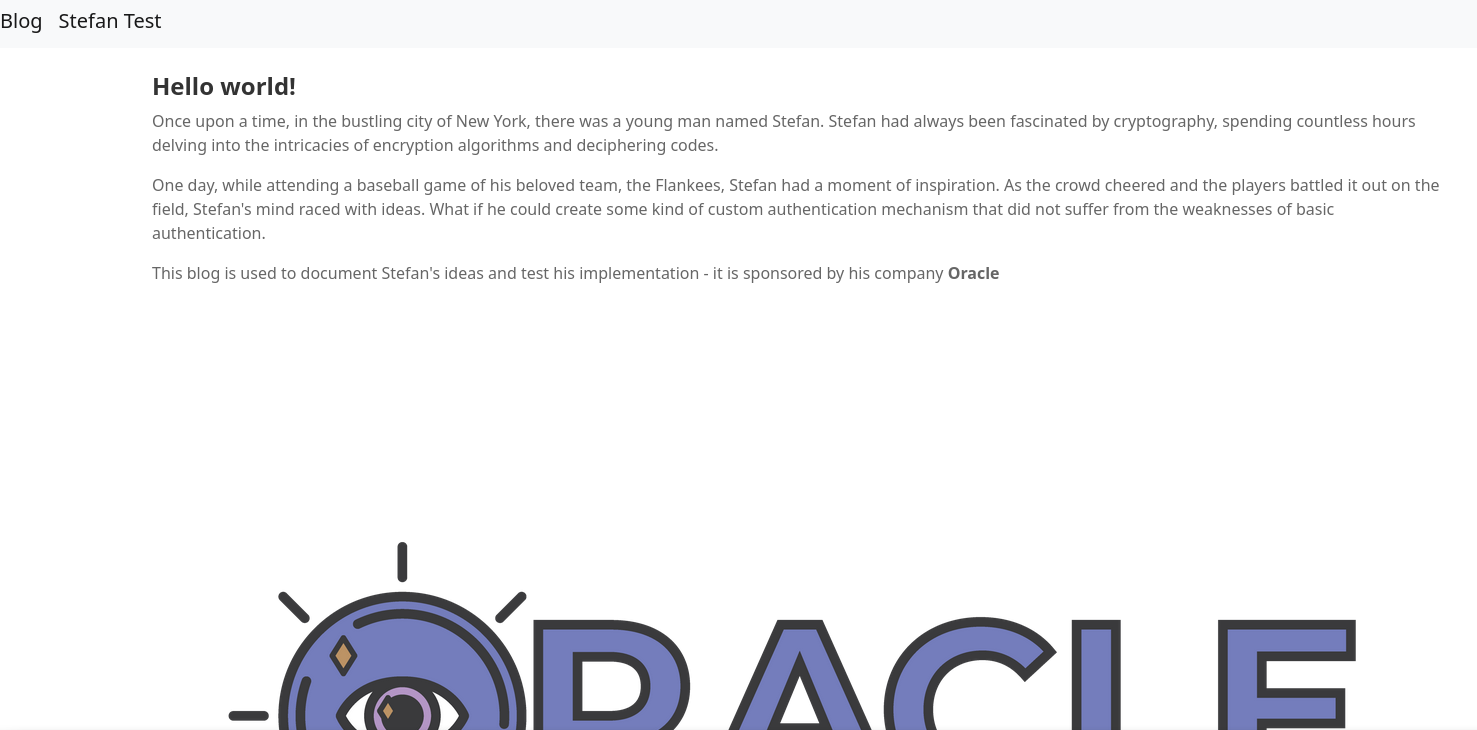
## PORT 8080: (an alternative to port 80)

Octoshape P2P streaming web service

This port normally hosts a HTTP server, hence unsecured.

* **Malware Distribution:** P2P networks, in general, have been known as potential avenues for distributing malware. Malicious actors could potentially inject harmful files into the network, which could then be shared with other users.
* **IP Address Exposure:** In a P2P network, users typically share their IP addresses with other peers. This exposure could potentially be exploited for malicious purposes.
* **Denial of Service (DoS) Attacks:** Malicious peers could potentially flood the network or specific users with excessive requests, leading to a denial of service.
* **Content Pollution:** Attackers might distribute corrupted or fake content disguised as legitimate streams.
* **Privacy Concerns:** Depending on how the P2P communication was implemented, there could be concerns about the privacy of users' viewing habits or shared bandwidth.

I surveyed Stefan’s blog as a normal user:

*figure 5: stefan’s blog*

I was able to bruteforce the directories using gobuster with the command

“gobuster dir -u http://10.10.53.218:8080 -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-small.txt -t 200 -x html,js,txt,php,db 2>/dev/null”

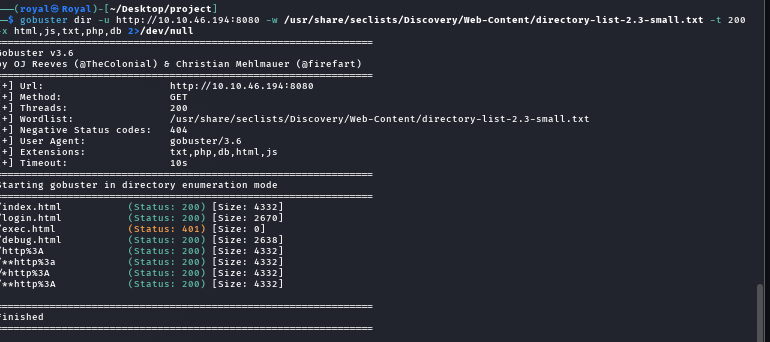
‘-w’ shows the wordlist to be used

‘-t’ specifies the threads to be used. More threads means the search will be faster.

‘-x’ just specifies what to search.

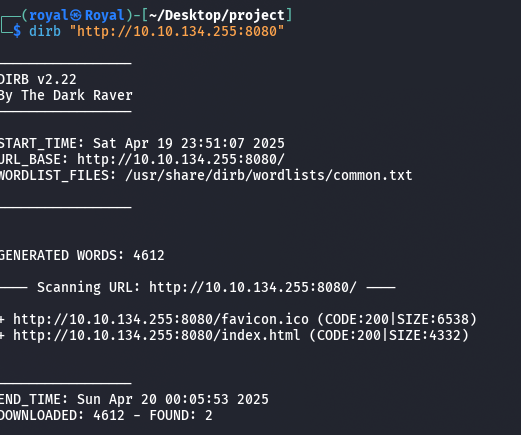
‘2>/dev/null’ tells the command to redirect all errors to /dev/null hence denoising the process.

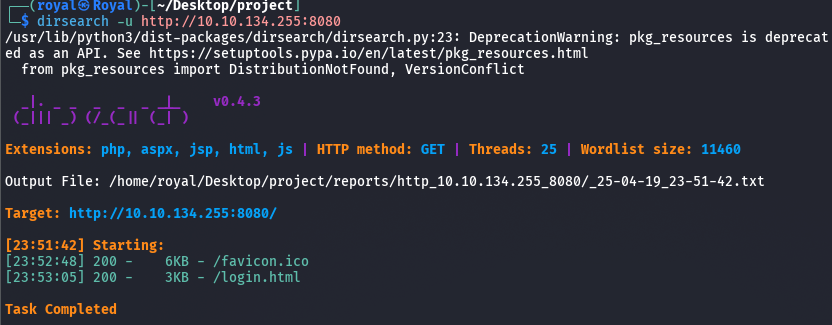
The directories with ‘200’ status code are accessible without necessarily logging in.

*figure 6: directory bruteforce*

There are other tools used to bruteforce directories such as ‘dirb’ and ‘dirsearch’

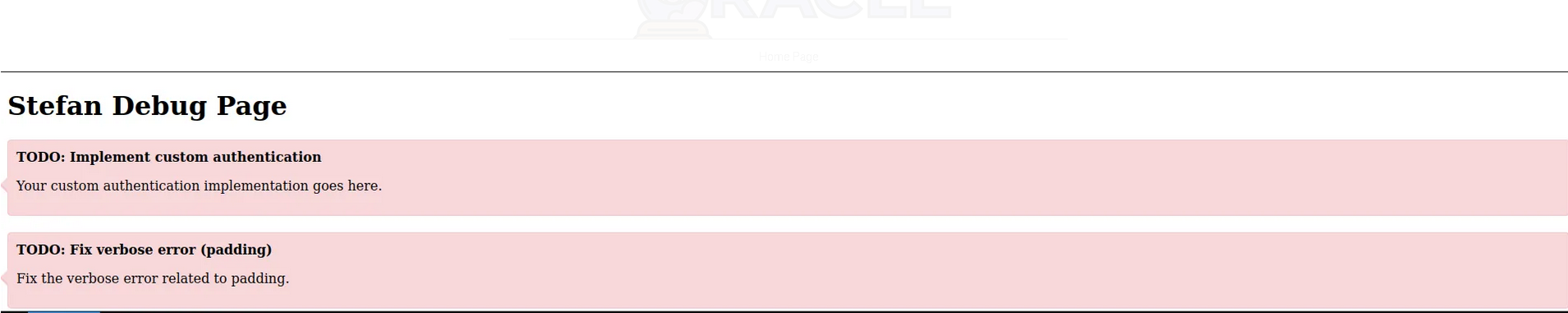
Now these tools are not as thorough as gobuster so i would recommend using gobuster.

*figure7:dirb search*

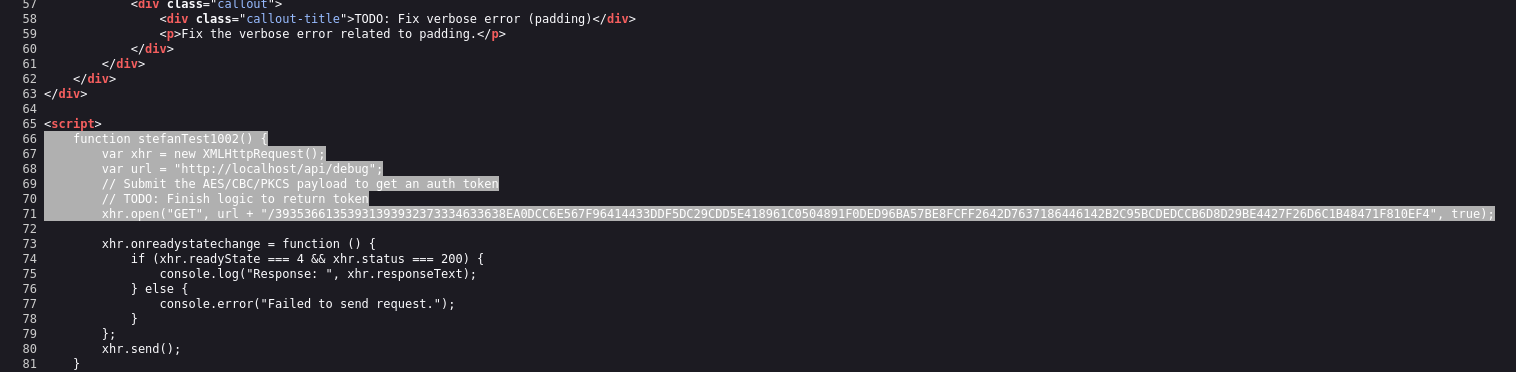
*figure8: dirsearch bruteforce*

These two only find 2 directories unlike gobuster which found 8.

Back on the blog page I navigated to Stefan Test which redirected to the /debug.html directory.



This page seems to have personal notes. After viewing the source code, there is a script that provides us with an endpoint and a token.

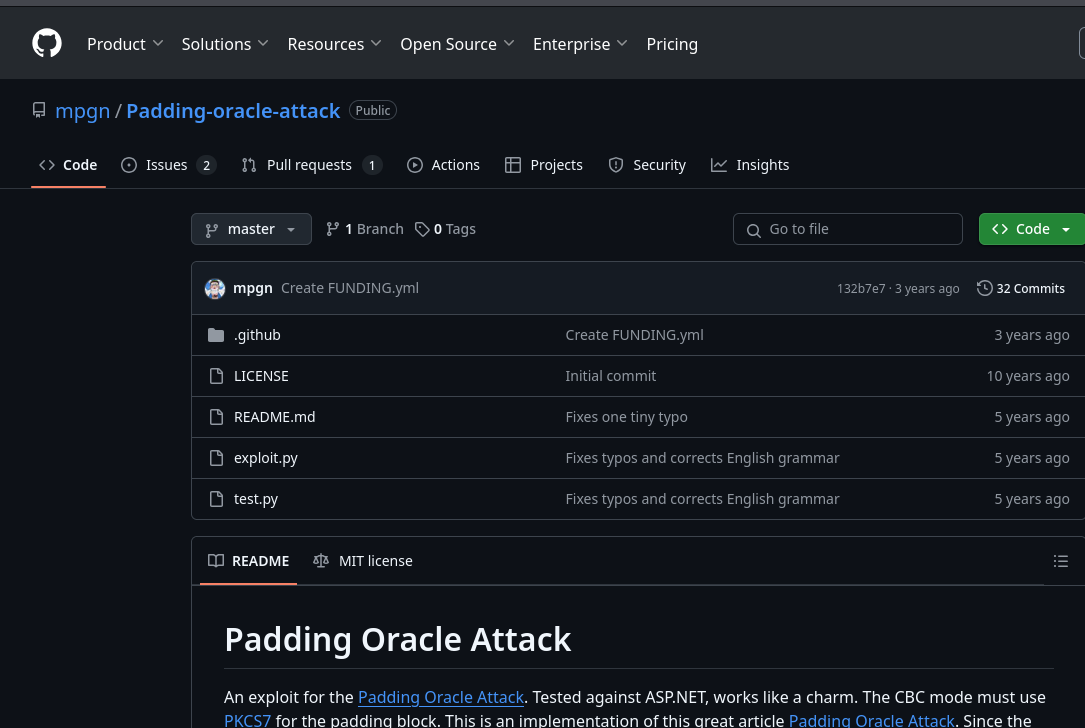
*figure9: /api/debug*

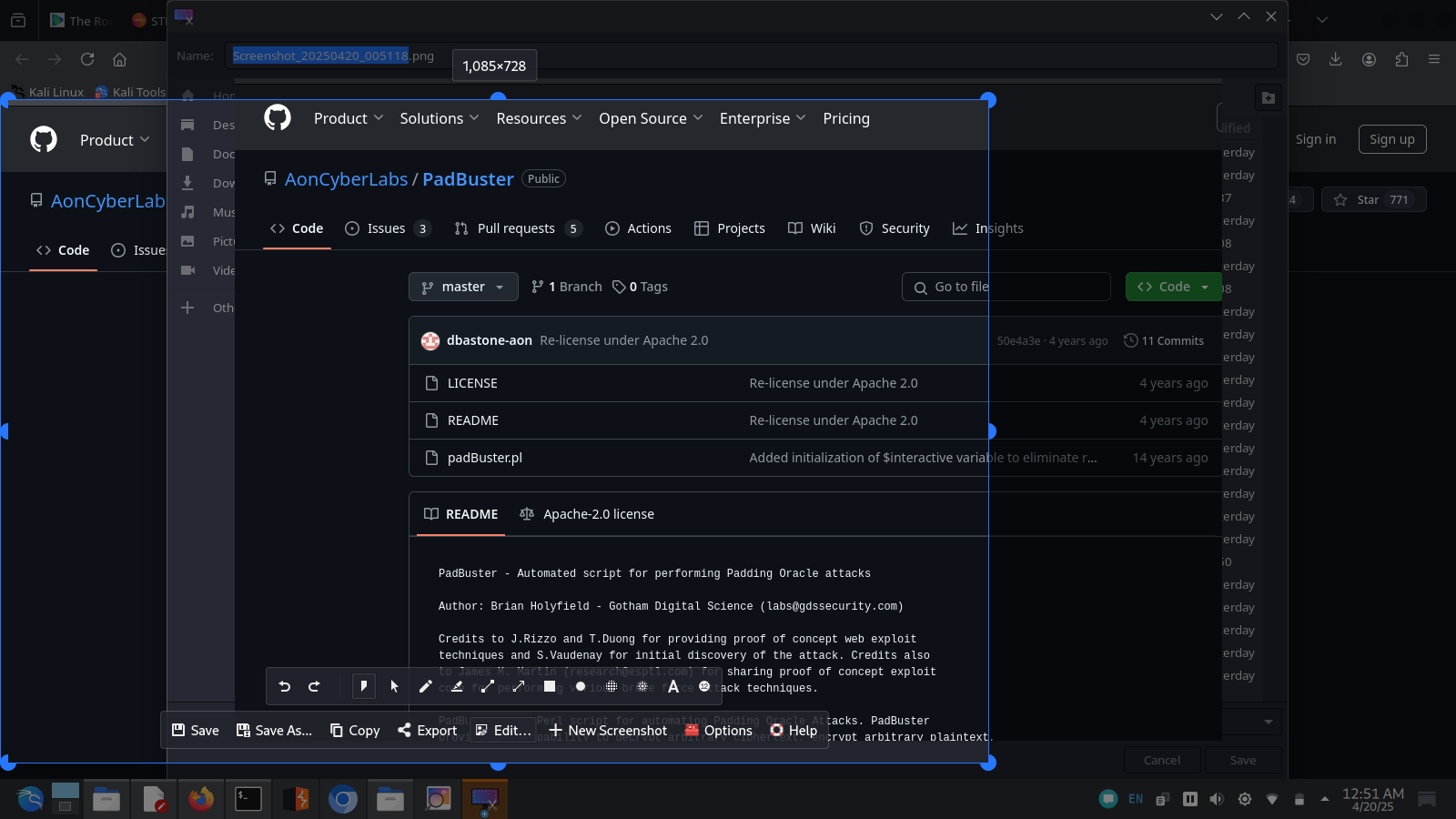
When we provide the endpoint followed by the token on our web browser, it returns a ‘custom authentication success’ message.

When we remove the numbers, it returns a ‘Decryption error’ message.

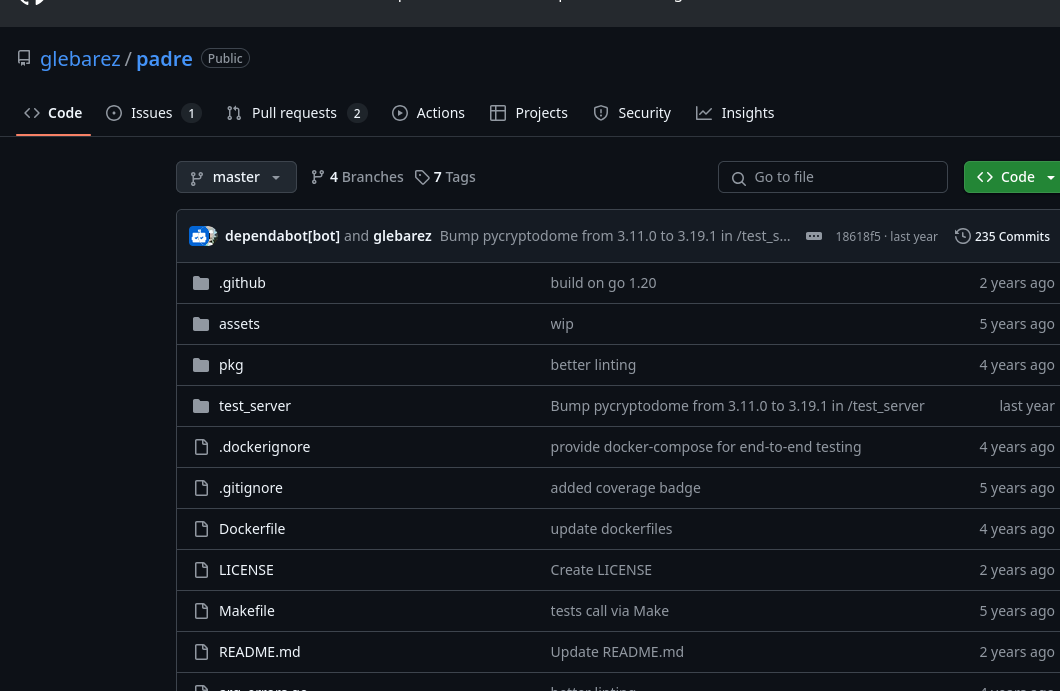
I researched about the verbose error related to padding and found out that it is a vulnerability called oracle padding.

I found several ways of exploiting it but i couldnt make some of them work like padbuster and padding oracle attack.

*figure10: padding oracle*

*figure11: padbuster*

After research i found an easier tool to use called padre on github

*figure 12: Padre decryptor*

Inorder to use it we have to use git clone to get it first.

Now lets run the command:

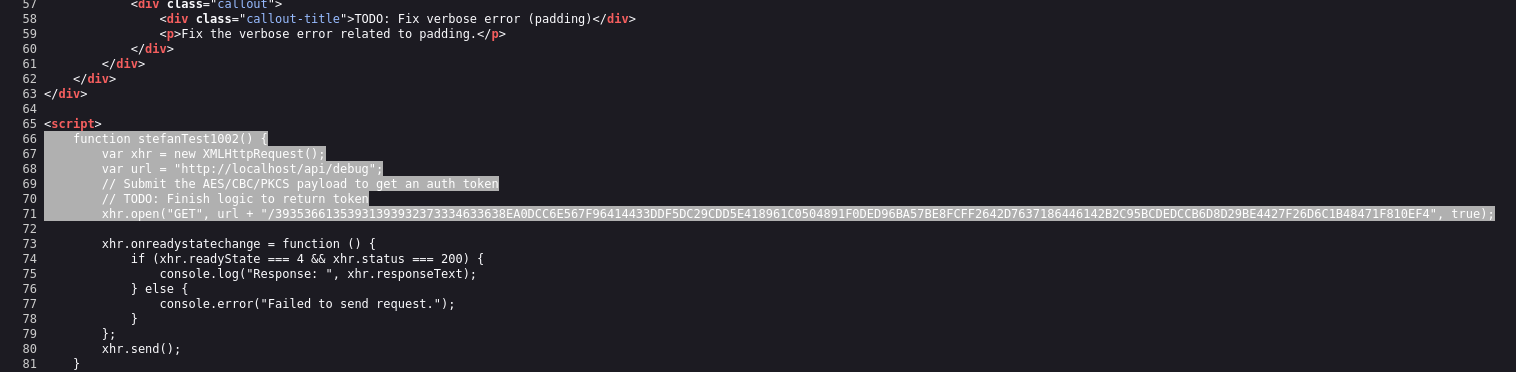
./padre-linux-amd64 -u "http://10.10.103.189:8080/api/debug/$" -e lhex -err "Decryption error" "3935366135393139397BE8FCFF264207637186446142B2C95BCDEDCCB6D8D29BE4427F2606C1848471F810EF4"

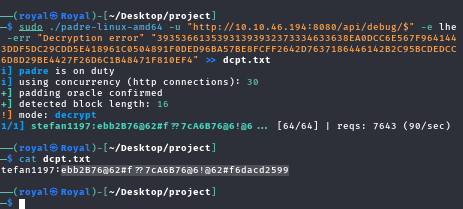
‘-u’ specifies the url

‘-e’ specifies the encoding type, which we set to lowercase hexadecimal.

‘-err’ this specifies the error message to be relayed incase of decryption failure.

The end is the token.

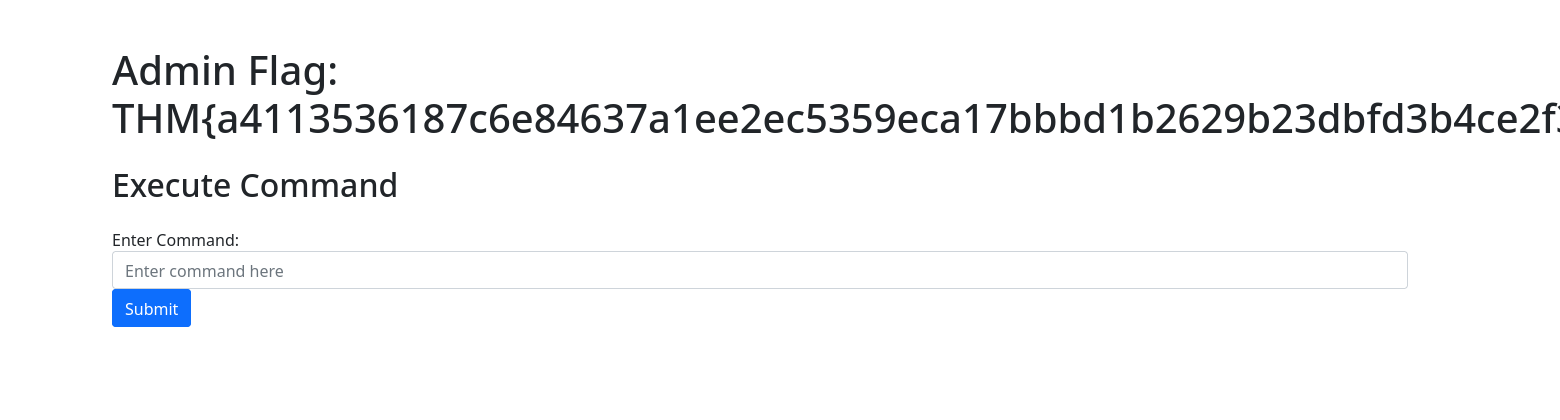


*figure 13: encryption with padre*

The decrypted message is the username and password respectfully.

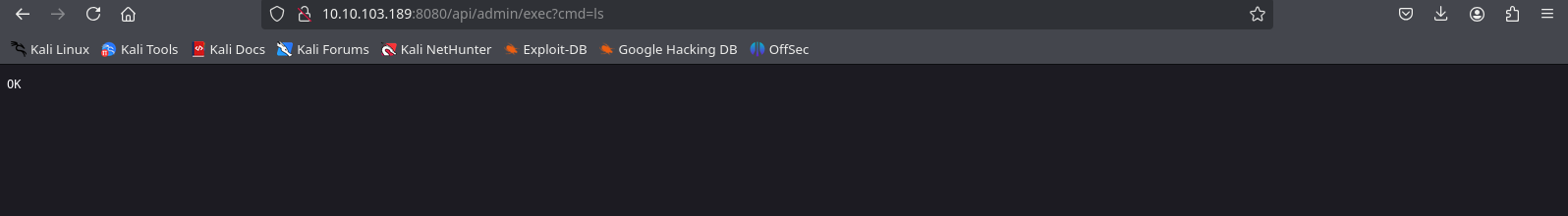
This gives us access to the admin page.(FLAG 1)

When navigating to the admin page, I found a command execution bar.

*figure 14:Debug page*

I opted to use burpsuite to capture the GET requests.

When i ran a command on the command bar, the output message was ‘ok’.

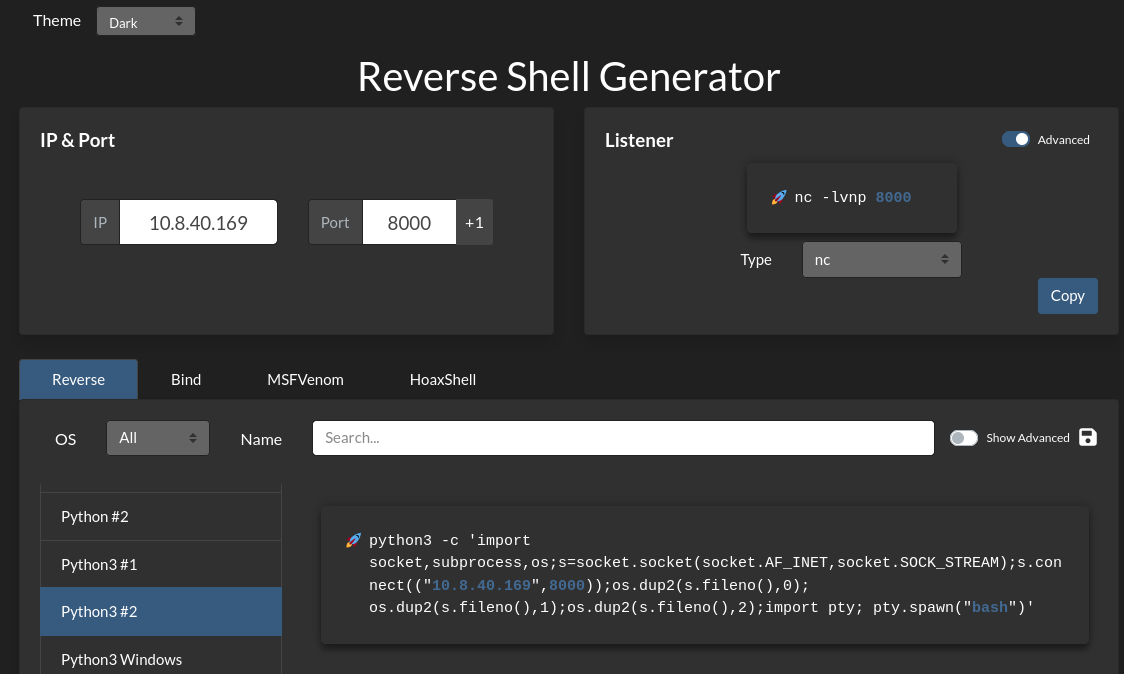


This confirms that the command was executed but we can’t see anything else.

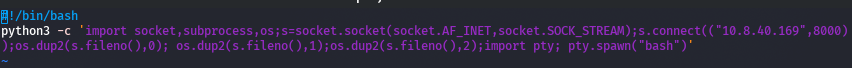
Let's send it to our repeater.

I decided to create a reverse shell to gain shell access to the host and interact with the machine directly.

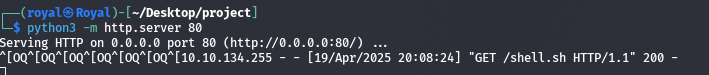
I used an online reverse shell generator to write a python shell



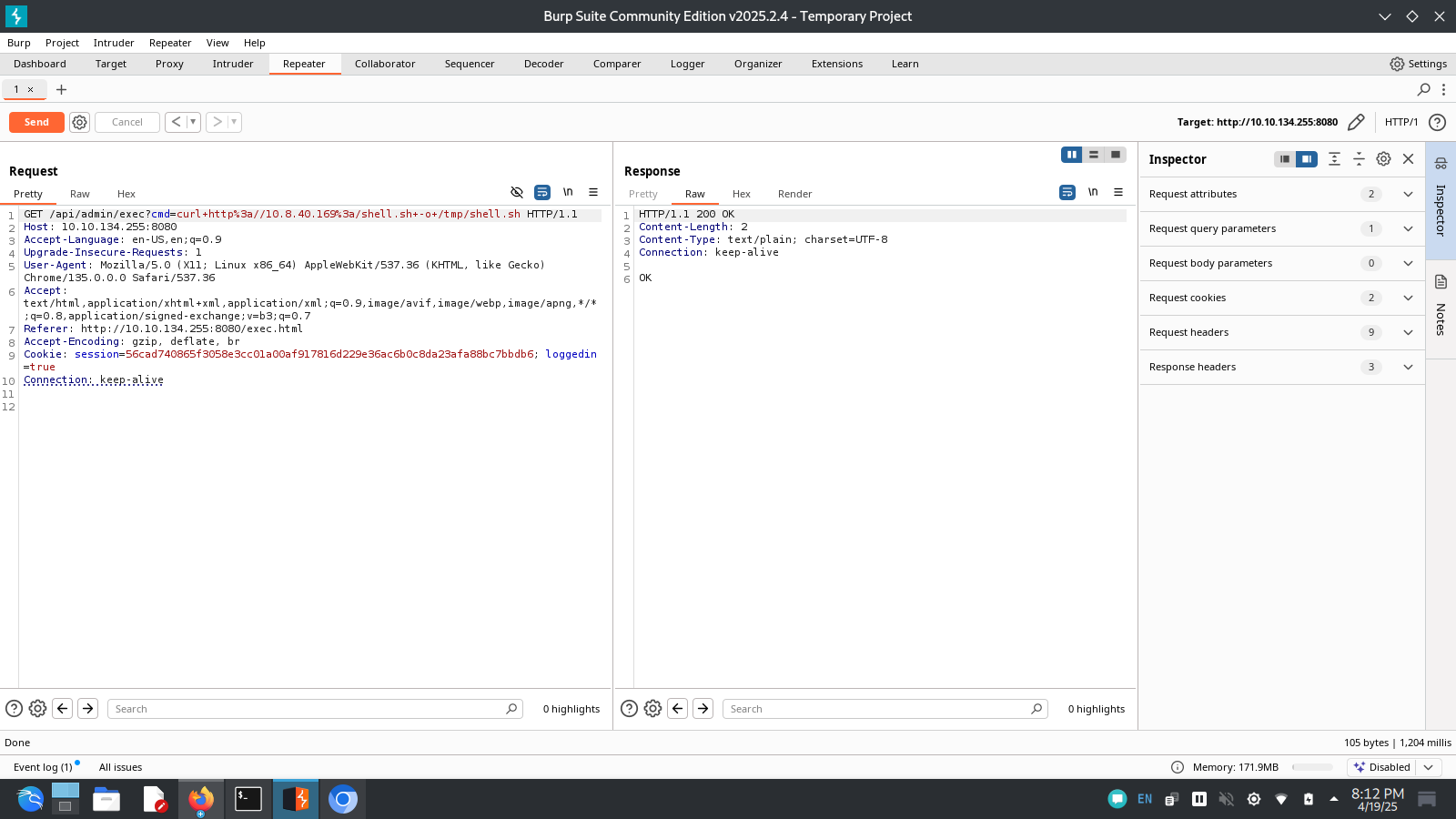
I created a file called ‘shell.sh’ on my computer and copied the python script above.



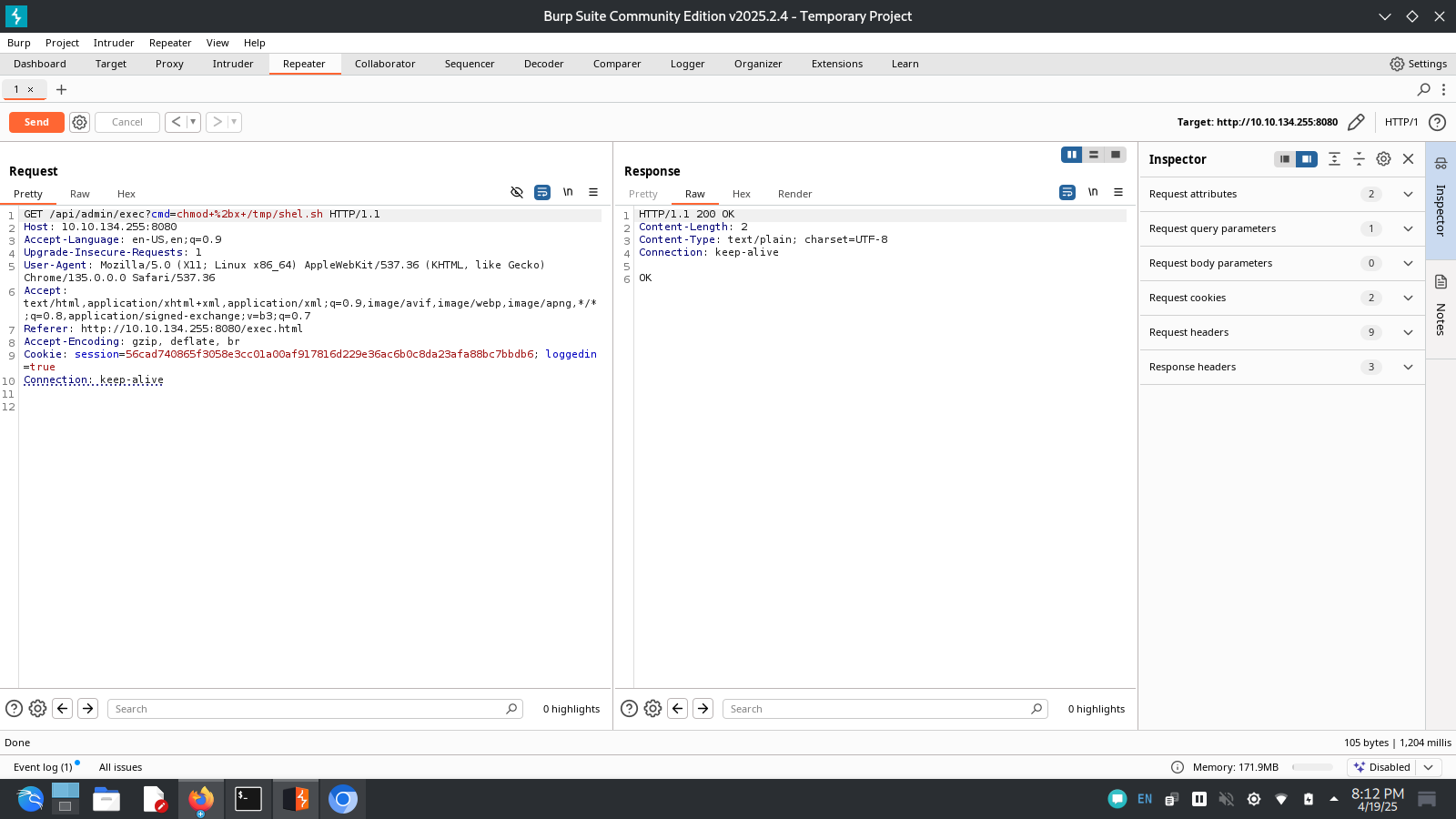
I openned the HTTP python server to download the shell on the target machine .



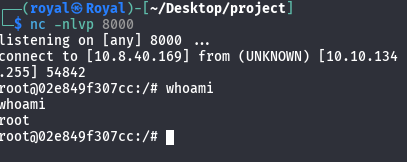
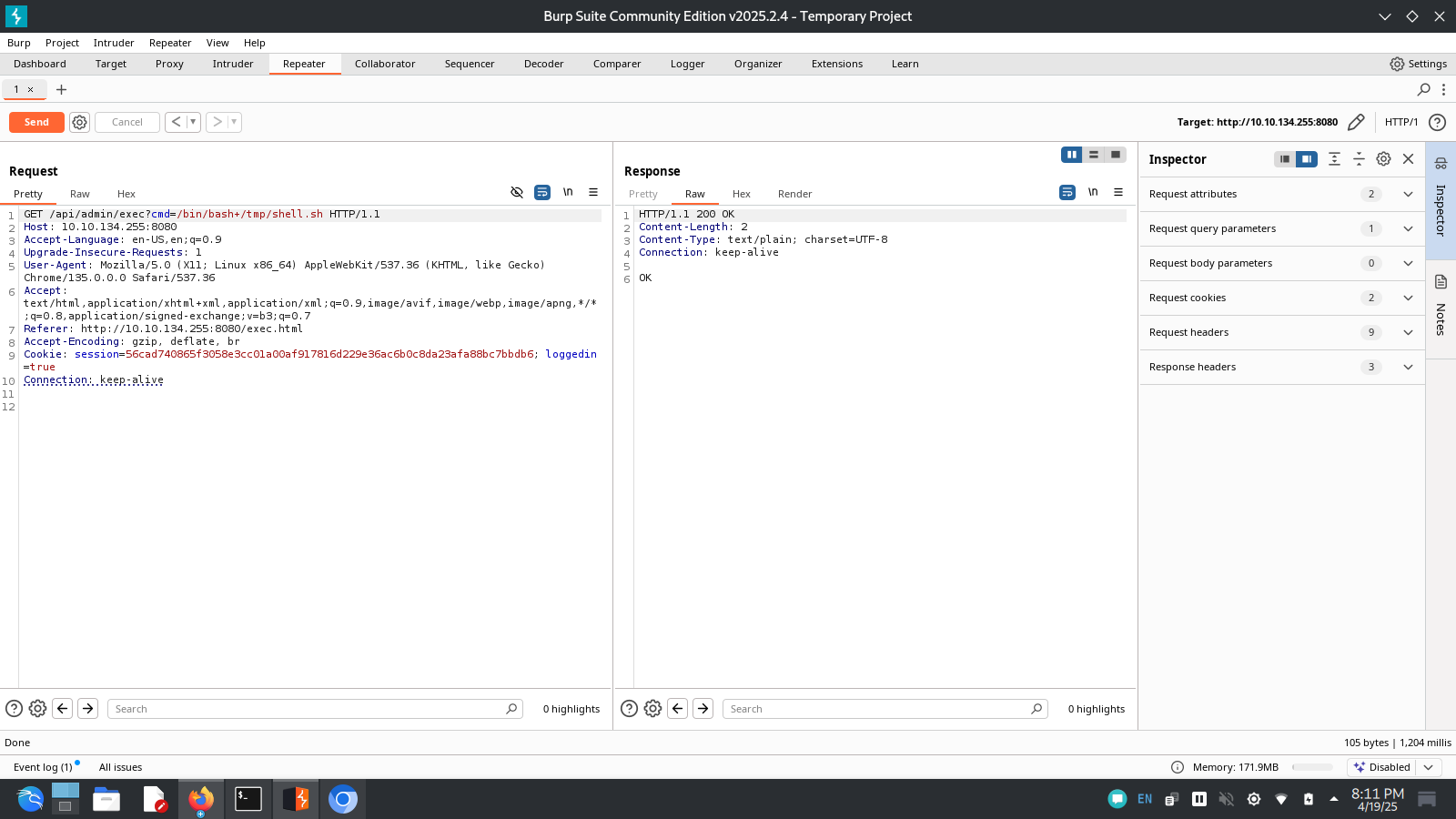
I used the curl command (which is used to transfer date using various protocols) to download the shell into the target computer.



I was able to make the shell executable by using ‘chmod +x’ command:



I used netcat to open a listening port on port 8000 and executed the sh file.



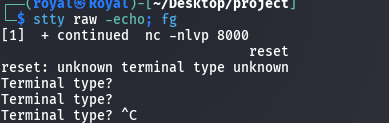
The reverse shell I created gave me root access to the machine.

I backgrounded the process and ran the command ‘stty raw -echo: fg’ on my machine.

‘stty’ is a command used to change and print the terminal line setting.

‘raw’ command sets the terminal mode to raw. In this mode, input is passed directly to the shell program without line processing like buffering.

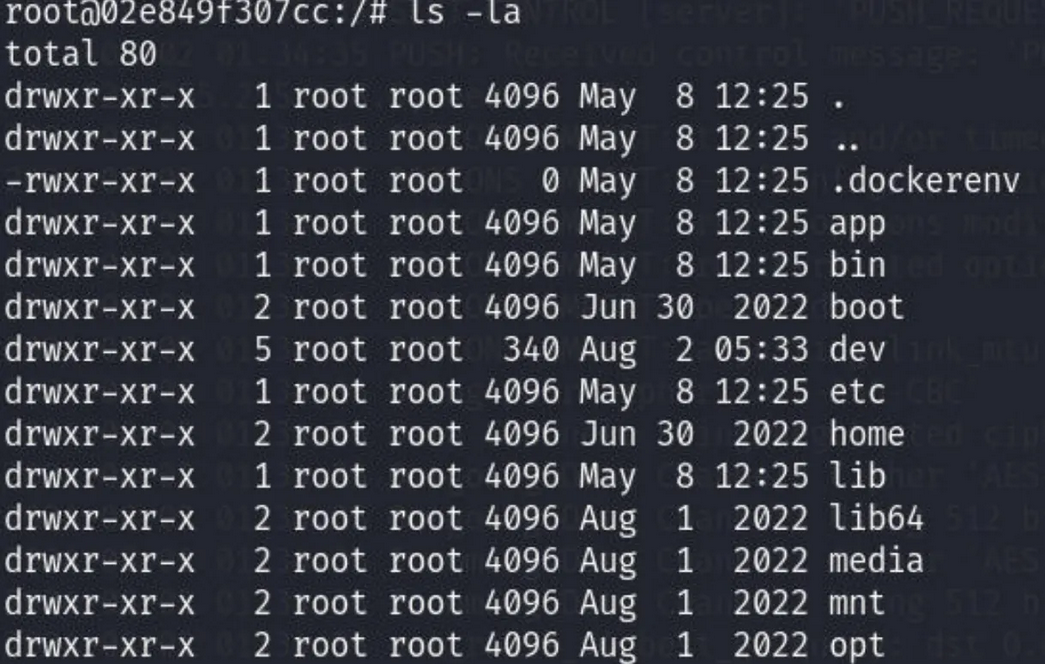
‘-echo’ command disables echoing



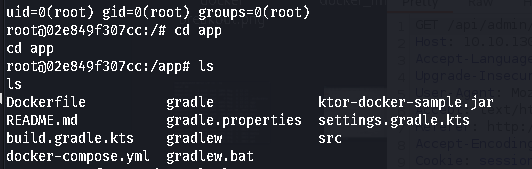
I used the ‘ls -la’ command to list out all files in the system.

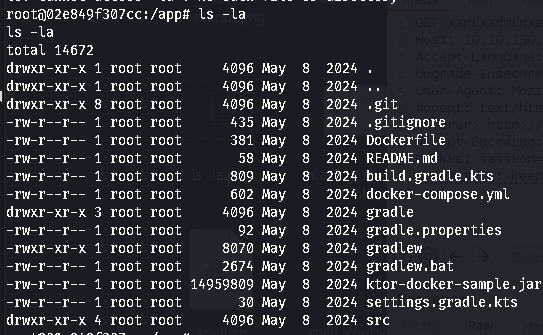
‘l’ makes the system list the files in a long list form

‘a’ makes the system list all files including the hidden ones.

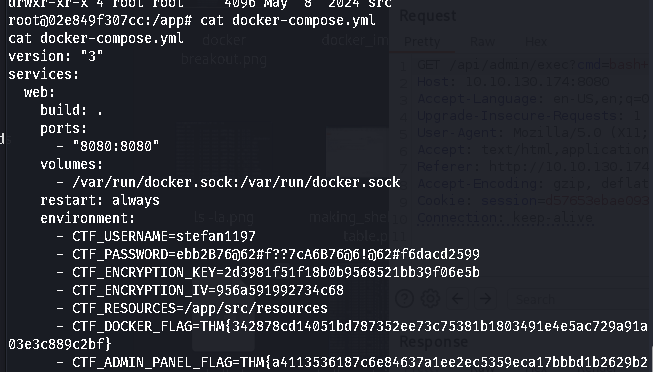


I changed my directory to the “app” directory and surveyed inside it:



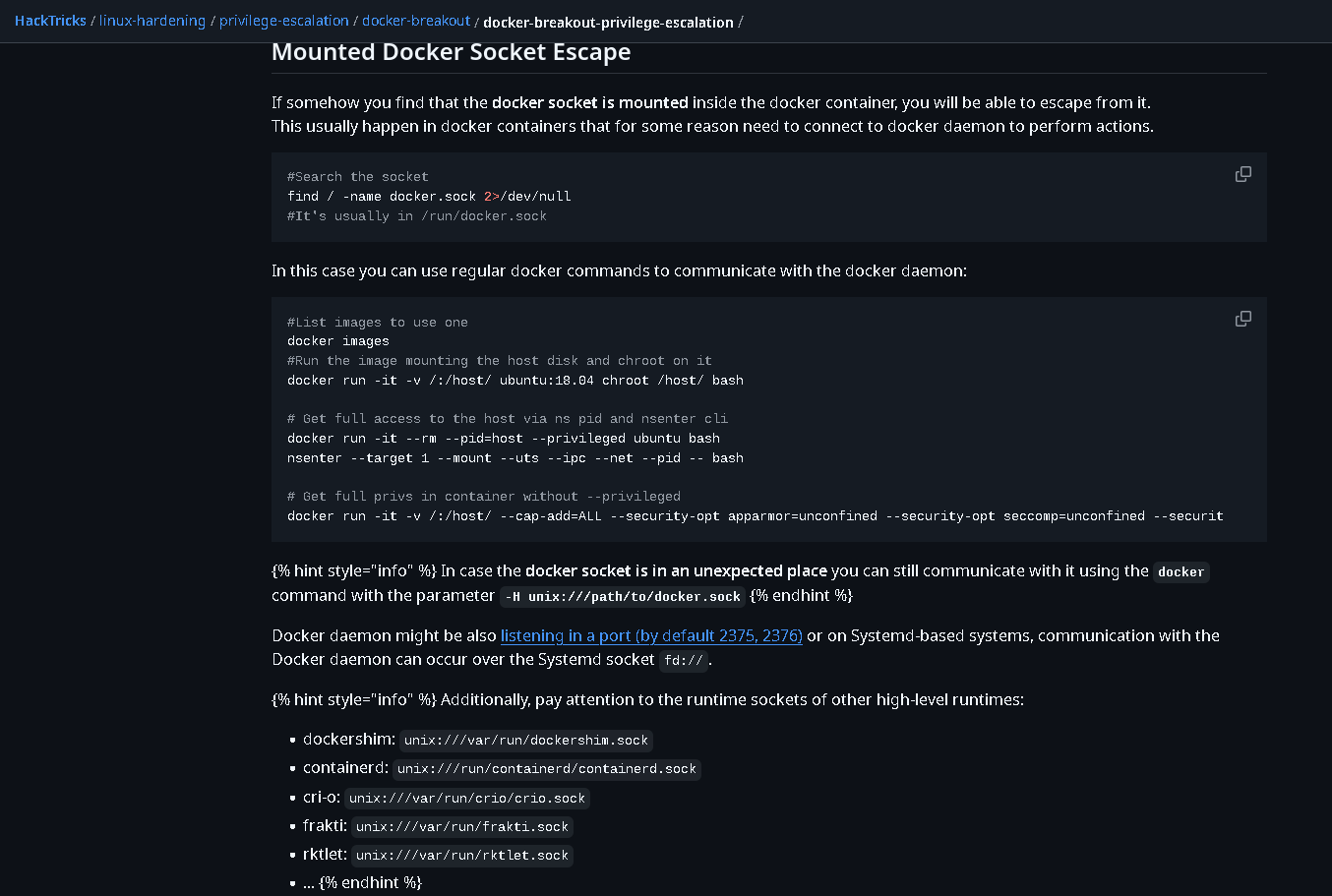


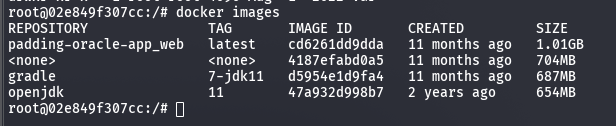
I tried reading each of the files and got the second flag in the “docker-compose.yml” (FLAG 2)



I saw this dockernv file. Its a docker environment. This is a text file that contains a list of environment variables and their values. This file is used to store your Password Pusher environment variables.

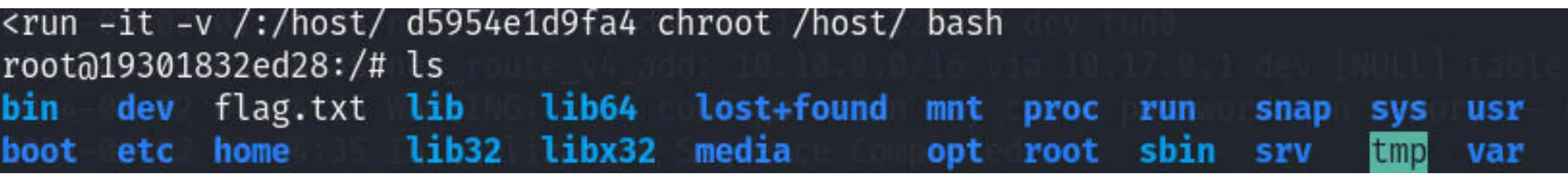
From research, I found that when you run the command ‘docker images’ , it outputs the images and their id’s.





I tried out each image id to find the one i was looking for and with luck i found it. [ID:d5954e1d9fa4]

From the research, in order to run the id we use the command “docker run -it -v /:/host/ <Image\_ID> chroot /host/ bash”



After reading the flag.txt file, i got the final flag leading to full compromise of the system. (FLAG 3)

#### 

#### **Vulnerabilities Assessment**

* Oracle padding attack: a **padding oracle attack** is an attack which uses the [padding](https://en.wikipedia.org/wiki/Padding_(cryptography)) validation of a cryptographic message to decrypt the ciphertext.
* I was able to use padre to decrypt the token provided in the site’s script.
* Command injection: is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application.
* After gaining access to the Admin page, there was a command bar where i was able to download a shell to the target system.
* Docker breakout: the ability of applications or processes running inside a container to access resources outside of the container that are not supposed to be available to them.
* Unrestricted file upload: i was able to download a shell to the target system without restrictions and/or permissions.

REMEDIATION SUMMARY

##### **Mitigations for the found vulnerabilities**

###### Short term mitigations

1. Delay Error Messages : the server can prevent an attacker from quickly identifying successful decryption attempts based on response time and error codes.
2. Consistent Error handling : to ensure all errors stated during decryption are handled uniformly.
3. Avoid/Authenticate CBC
4. Applying principle of least privilege to limit damage that an attacker can do.
5. Using Web Application Firewall to filter out malicious requests from possible attackers.
6. Disabling unnecessary shell functions and conducting regular security audits
7. Escaping shell metacharacters by using inbuilt functions provided by your programming language.
8. Restrict container access and resources. Eg mounting root filesystem as read only.
9. Enhance container networking security by using custom bridge networks for isolation.
10. Focus on monitoring and auditing: consider runtime security monitoring tools.

###### Long term

1. **Eliminating or Minimizing System Calls:** Using language-specific libraries instead of executing external commands.
2. **Using Parameterized Queries:** For database interactions, to prevent SQL injection (a related vulnerability).
3. **Implementing Robust Input Validation and Sanitization:** Employing strict whitelisting and contextual output encoding.
4. **Adopting Secure Development Practices:** Including security training and code reviews with a security focus.
5. **Applying the Principle of Least Privilege (Application Level):** Designing components with minimal necessary permissions.
6. **Utilizing Securely Configured Containers:** To limit the impact of potential vulnerabilities.
7. **Conducting Regular Security Testing:** Including DAST and penetration testing.
8. **Stronger Container Runtimes:** Exploring options like gVisor and Kata Containers.
9. **Robust Isolation:** Utilizing Linux namespaces and cgroups effectively.
10. **Minimal Privileges:** Applying the principle of least privilege on the host and within containers.
11. **Hardened Host OS:** Securing the underlying operating system and Docker engine.
12. **Secure Storage:** Carefully managing and limiting host volume mounts.
13. **Network Security:** Segmenting networks and implementing strict policies.
14. **Secure Image Lifecycle:** Building, scanning, and verifying container images securely.
15. **Runtime Monitoring:** Detecting and responding to suspicious container activity.
16. **Secure Orchestration:** Configuring platforms like Kubernetes with strong security controls.
17. **Robust Input Validation:** Though not a primary defense, it adds a layer of security.
18. **Secure Key Management:** Using strong keys, rotating them regularly, and protecting their storage.
19. **Careful Error Handling:** Returning generic and consistent error messages for decryption failures.
20. **Regular Security Audits and Cryptographic Reviews:** To identify and address potential vulnerabilities.
21. **Using Secure Cryptographic Libraries:** To avoid implementation errors.
22. **Developer Education:** Ensuring the team understands secure cryptographic practices.