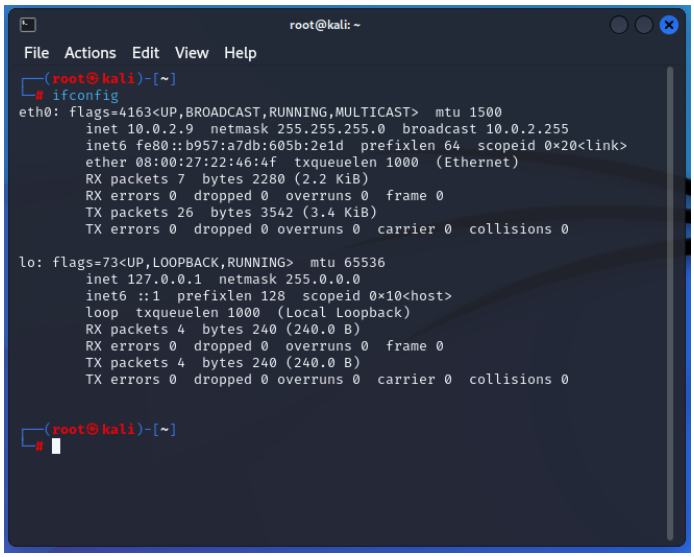
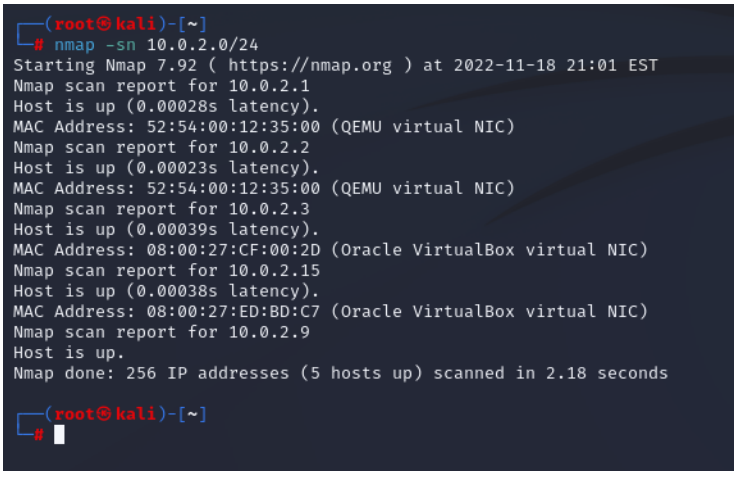
Description of techniques and exploits:

After setting up the environment, we needed the target machine's (web machine n7) IP address to get more information. So to figure it out, we first checked our machine's IP address using the 'ifconfig' command.



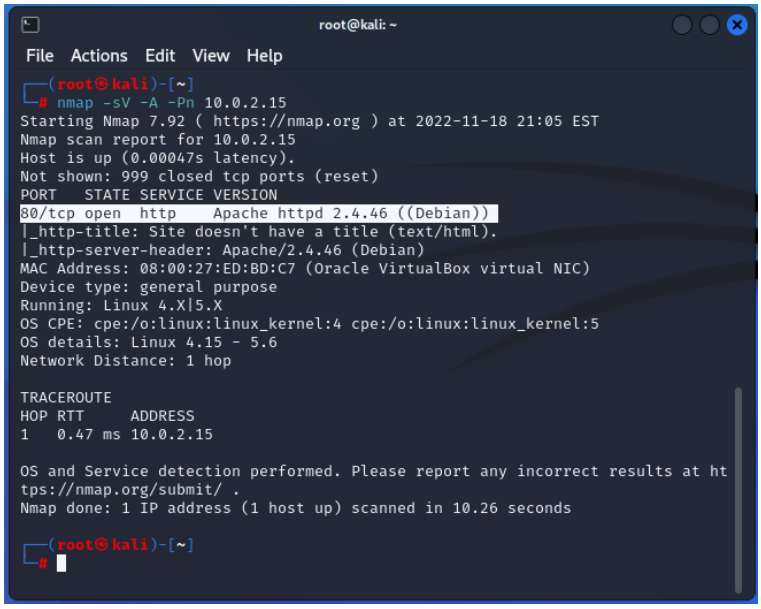
So, we identified '10.0.2.9' as our machine's (kali Linux) IP address. As we saw before, both machines are in the same NAT network. So, to check the target machine's IP address, we used nmap as it is a straightforward and flexible utility tool for exploring a network. It helps to determine which hosts are active, their ports and protocols, and TCP/UDP IP fingerprinting. To identify the target machine's IP address, we used 'nmap –sn 10.0.2.0/24'; this command will scan all the active connections and execute the ping test within the range from '10.0.2.0 to 10.0.2.24' and display them.



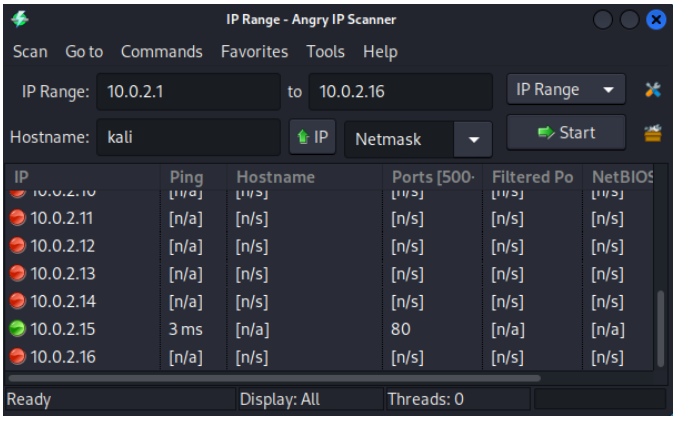
We saw that our machine's IP address (10.0.2.9) shows as 'Oracle VirtualBox virtual NIC'; there was another IP address with a similar label, so we pinpointed that the target machine (web machine n7) IP address was '10.0.2.15'.

After that, we used three tools to get all the information about the target machine: OS system and software, security standards, ports, and services like DNS servers and web services. First, we used nmap;

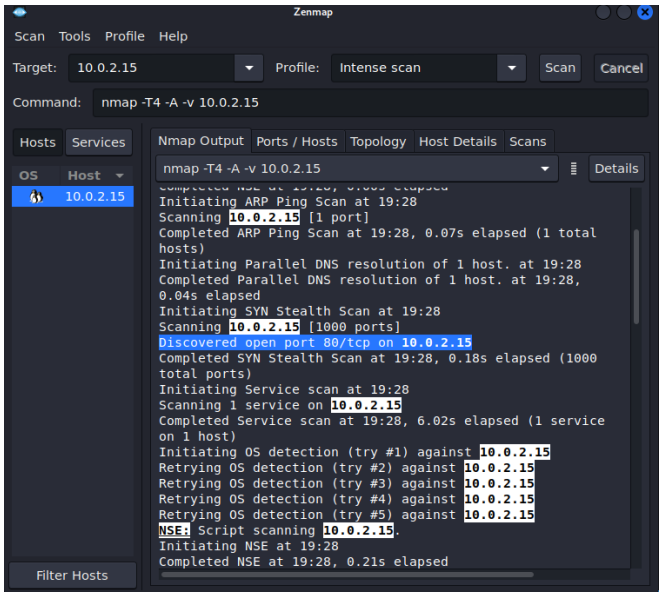
here “-sV” parameter detects all the open ports and shows all the services running. “-A” parameter detects OS and its services, and the “-Pn” performs a ping test to check the IP address given is active or passive.



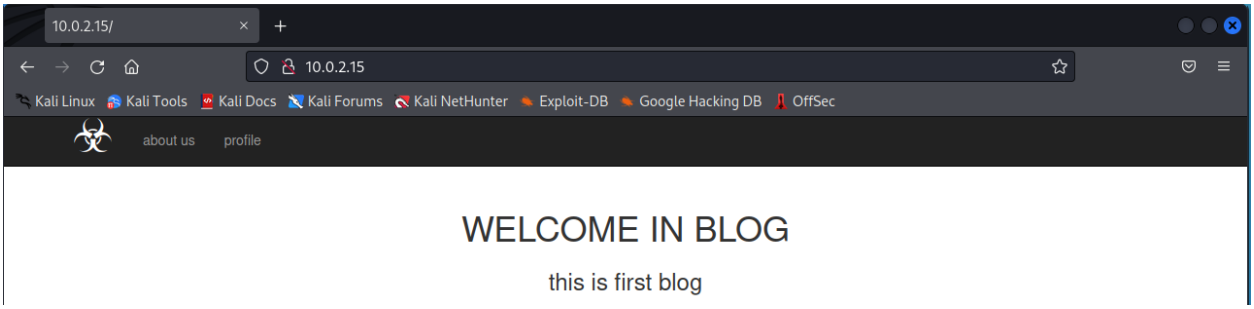
Nmap revealed a lot of information regarding our target machine by which we can connect it; our main focus was on the open port, which displayed as ‘80’, but we were still hesitant with this information because many times nmap shows only 1 port even though multiple ports like ’21, 23’ would be open, so to cross-verify we used another tool named as ‘Angry IP Scanner.



Angry IP Scanner is an extensible and easy-to-use network scanner tool that helps quickly scan all the available networks. So, to scan and verify the IP addresses and open ports, we kept the range ’10.0.2.1 to 10.0.2.16’ and confirmed that the target machine had an ‘80’ port open after scanning. But there might be some chances that other ports might be blocked on this tool, so we used one more tool named ‘Zenmap’.



Zenmap is also a network scanner cross-platform tool. To gather and verify information regarding our target machine, we used parameters like “-T4” for fast execution combined with the “-A” parameter to detect OS and its services, and the “-v” parameter to check the versions. And when it also gave similar results, just like other tools, we confirmed our target machine’s IP address and port, opened the Mozilla web browser from our machine, and entered the target machine’s IP address ’10.0.2.15’.



By loading up the website, we saw only this page which provided nearly zero information on interacting with it. We looked everywhere by inspecting the source page and the profile and about us page, but it was pointless. So, to gather the directories of the HTTP server, we made a directory traversal attack by which we obtained files and directories stored in the root folder using tools.

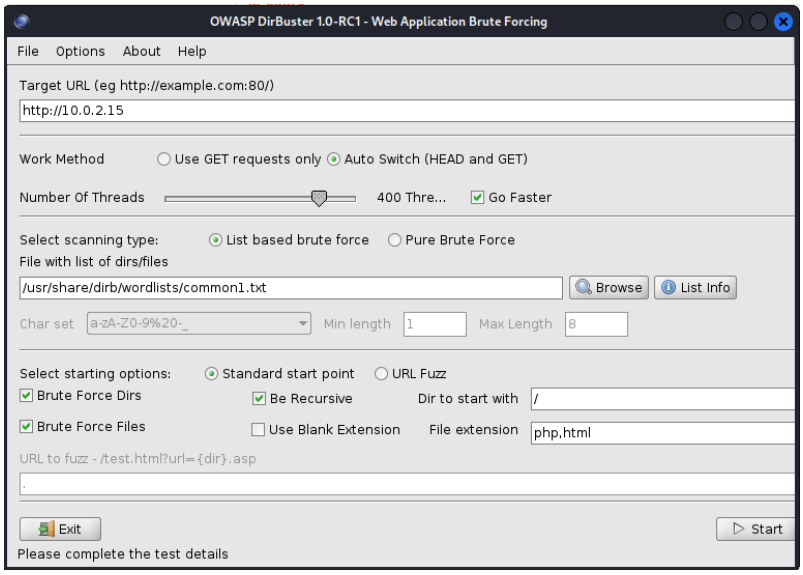
First, we used the Dirb tool, a web analysis tool that iteratively scans web directories and looks for hidden web objects to analyze the responses further. It comes with preconfigured wordlists which we use to obtain hidden web directories and files. So, we executed dirb on the target’s IP address with ‘common1.txt’ preconfigured wordlists combining it with the ‘-w’ parameter, which scanned intensely by not avoiding any warning. We did not add any extensions as a parameter because dirb, by default, checks for ‘html, php, txt, asp’ extensions.

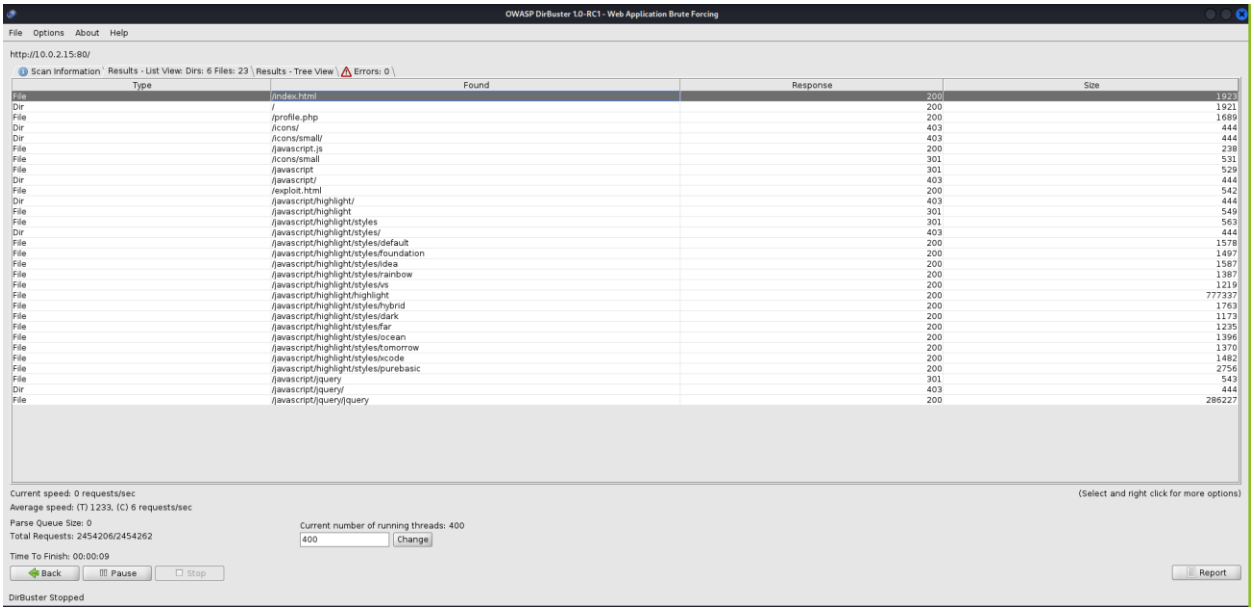
```
┌──(root㉿kali)-[/usr/share/dirb/wordlists]
└─# dirb http://10.0.2.15 -w /usr/share/dirb/wordlists/common1.txt

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sun Nov 27 16:47:01 2022
URL_BASE: http://10.0.2.15/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Not Stopping on warning messages
-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.0.2.15/ ----
+ http://10.0.2.15/index.html (CODE:200|SIZE:1620)
==> DIRECTORY: http://10.0.2.15/javascript/
+ http://10.0.2.15/server-status (CODE:403|SIZE:274)

---- Entering directory: http://10.0.2.15/javascript/ ----
==> DIRECTORY: http://10.0.2.15/javascript/jquery/

---- Entering directory: http://10.0.2.15/javascript/jquery/ ----
+ http://10.0.2.15/javascript/jquery/jquery (CODE:200|SIZE:275451)

-----------------
END_TIME: Sun Nov 27 16:47:27 2022
DOWNLOADED: 13836 - FOUND: 3

┌──(root㉿kali)-[/usr/share/dirb/wordlists]
└─#
```

After an in-depth scan by the dirb tool, we found only 3 files, and those were not useful. So we used the second tool named Gobuster, a tool written in the Go language that brute forces URLs/domains and reveals files and directories of the webserver. And also, it is faster compared to the previous tool. It also uses preconfigured wordlists which we use to obtain hidden web directories and files. So, we executed gobuster on the target's IP address by using the '-u' parameter along with '.txt' preconfigured wordlists combining it with the '-w' parameter, which scanned intensely by not avoiding any warning and also with 'dir' mode which enables brute-forcing on the targeted url. Lastly, the '-x' parameter extracts all directory paths corresponding to 'html, php' extensions.

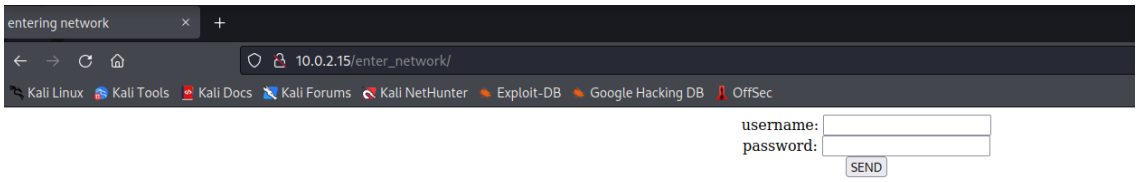
```
┌──(root㉿kali)-[~]
└─# gobuster dir -u http://10.0.2.15/enter_network -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x php,html
===============================================================
Gobuster v3.3
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.0.2.15/enter_network
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.3
[+] Extensions:              html,php
[+] Timeout:                 10s
===============================================================
2022/11/29 23:09:24 Starting gobuster in directory enumeration mode
===============================================================
/.php                 (Status: 403) [Size: 274]
/index.php            (Status: 200) [Size: 324]
/.html                (Status: 403) [Size: 274]
/admin.php            (Status: 200) [Size: 126]
```

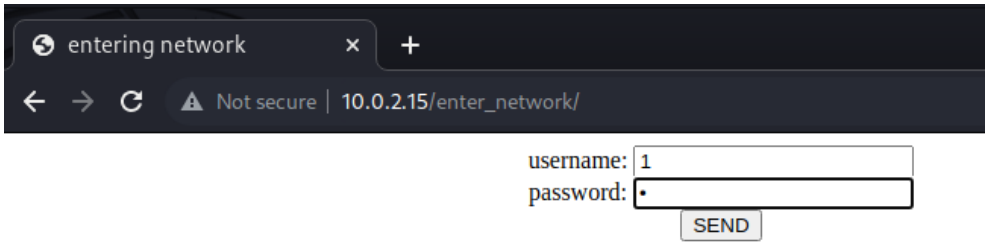
After an in-depth scan by the gobuster tool, we found 28 files and directories, but we used one more tool for double-checking. Our third tool was Dirbuster, an OWASP content discovery application that uses brute force on web servers by sending GET requests and analyzing the responses, revealing hidden files and directories. We placed the target's url in the url section and set the number of threads to 400 for faster execution. We set the same as the dirb tool for the wordlists part and kept the 'php, html' extensions.



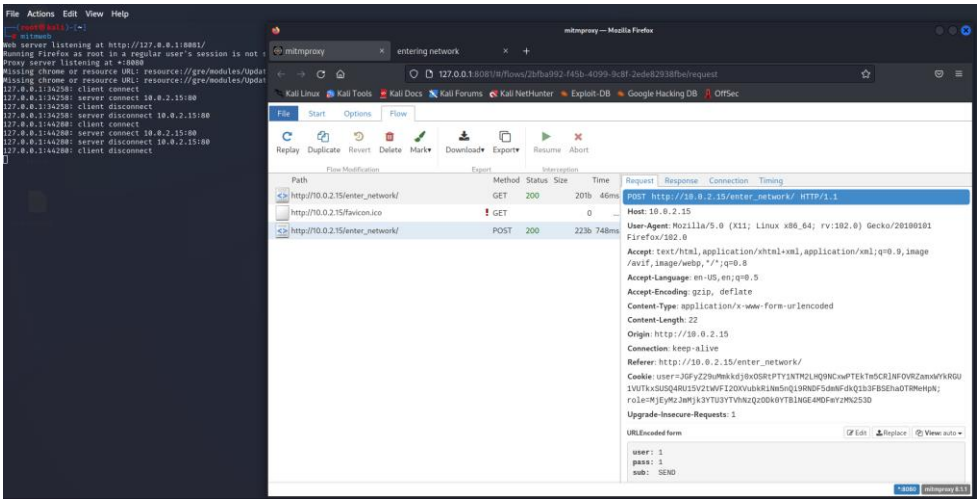


After an in-depth scan by the dirbuster tool, we found 53 files and directories. We reviewed each file and directory, looked inside their contents, researched a lot, and discovered a file named ‘enter_network’ in the target machine’s web server.

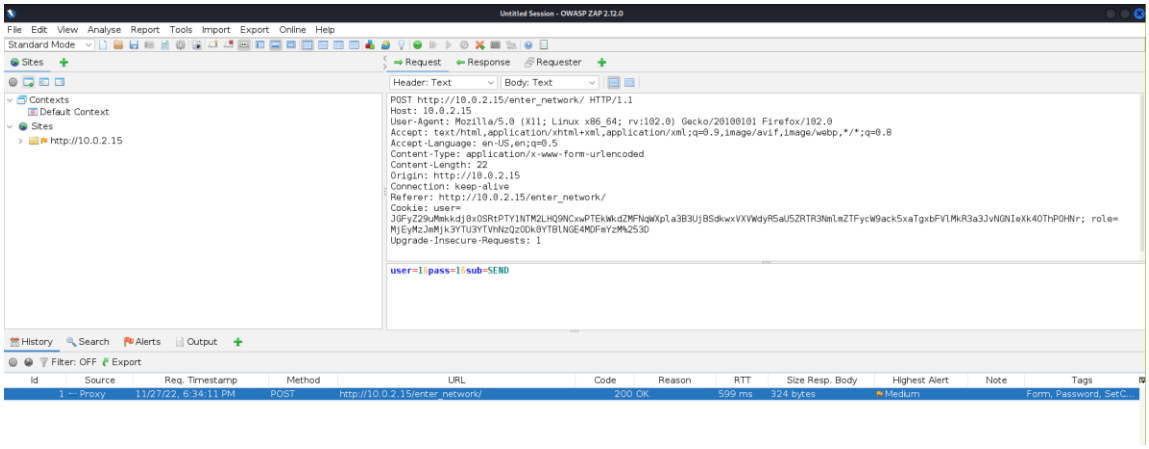




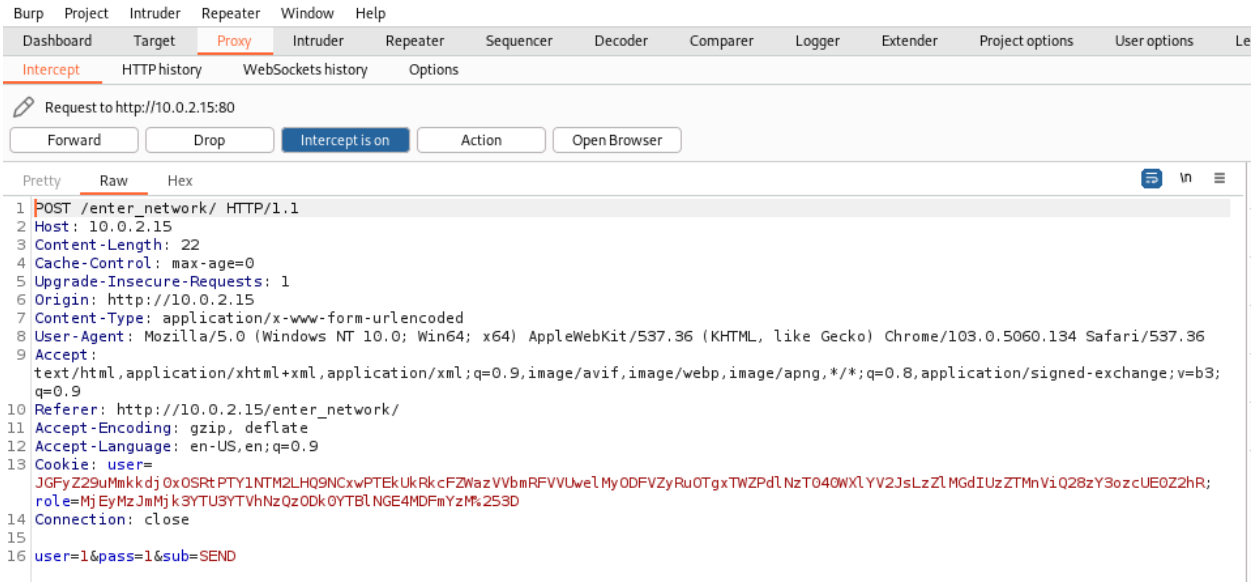
By examining the webpage, we observed that HTTP GET and POST requests were happening by entering any credentials, but we could not learn anything from the Mozilla Firefox inspector tool. We used tools that would intercept proxy on the website to gather all the information between HTTP requests. First, we used MITM proxy, a man-in-the-middle proxy tool that intercepts HTTP requests and can modify them. So after setting up the tool, when we pressed send button on the website mitmproxy tool, we identified different parameters of GET and POST requests.



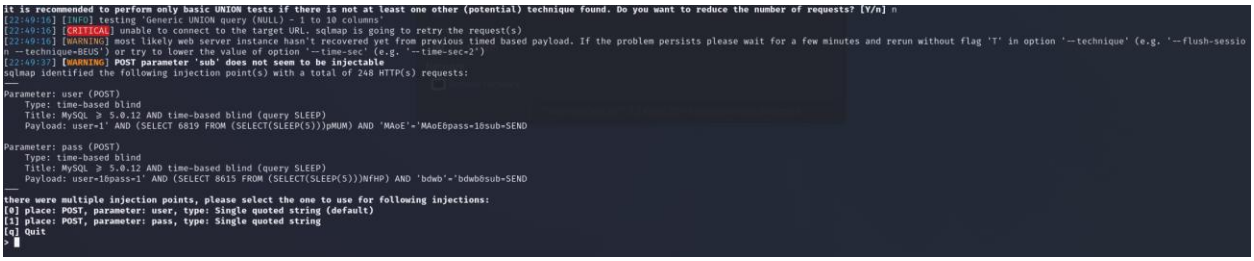
The tool showed us different parameters, but our main focus was on the cookie of the POST request as we wanted to save the POST request file and use it in the sqlmap tool, so to cross-verify, we used another tool named ZAP. It is a scanning tool with the same functionalities as the previous tool. So after setting up the tool, when we pressed send button on the website zap tool, we identified different parameters of GET and POST requests.

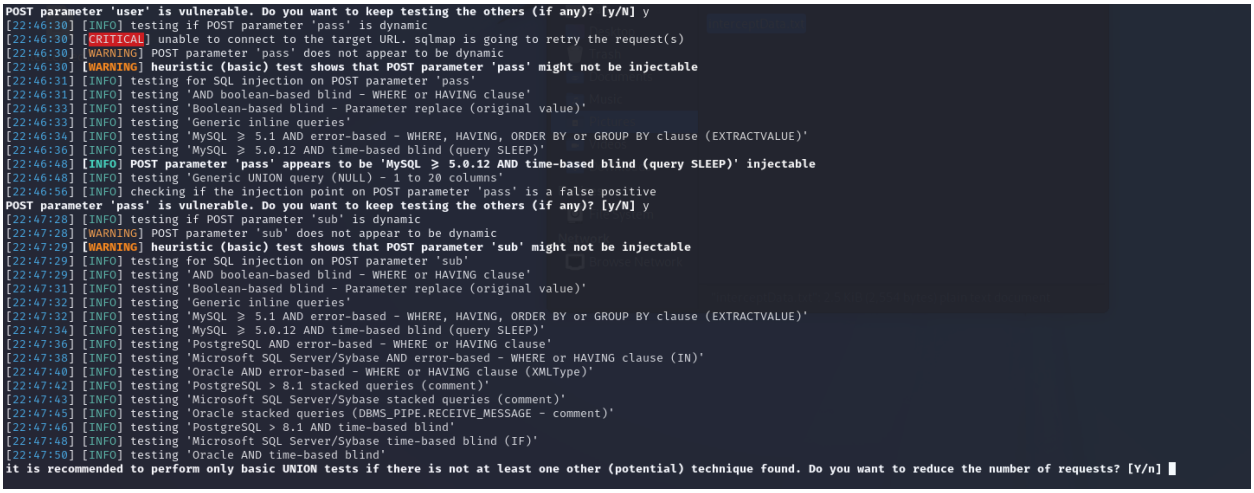


The tool showed us different parameters, but our main focus was on the cookie of the POST request as we wanted to save the POST request file and use it in the sqlmap tool, so to cross-verify, we used another tool named Burpsuite. It is a scanning tool with the same functionalities as previous tools, like intercepting proxy and modifying it. So after turning the intercept on when we pressed send button on the website burpsuite tool, we identified different parameters of GET and POST requests. We saved the POST request file as 'wm7.txt' and used it in the sqlmap tool to check vulnerabilities.

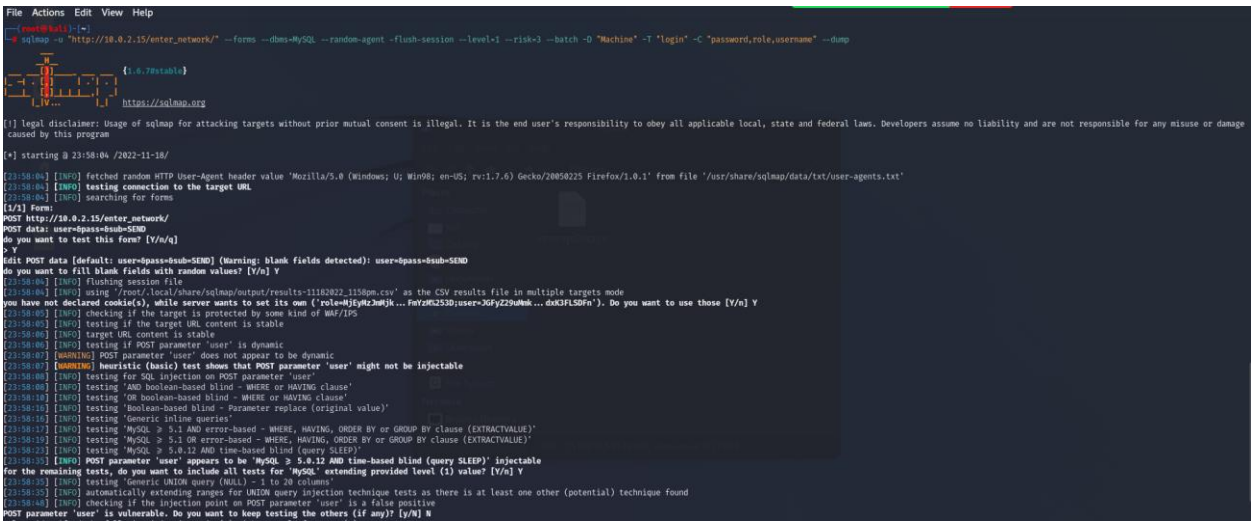


Sqlmap is a tool that detects and exploits SQL injection. It also supports almost all database management systems. Once our vulnerability gets detected, it will show us a few options to exploit. So, we used the 'sqlmap -flush-session -r /home/kali/pictures/wm7.txt -- dbs' command; the 'flush-session' parameter is used to remove any previous sessions which would hinder the process along with the '-r' command to run the text file and also '-- dbs' parameter to extract database names.





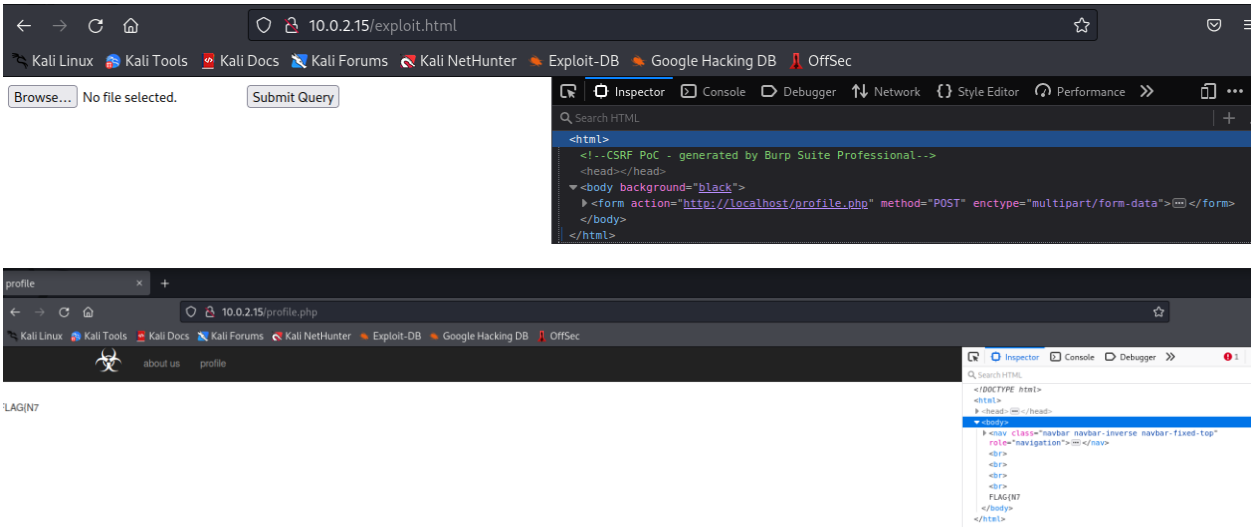
As sqlmap ran, it asked for many options to process the file, like entering random values to check and entering tests for an in-depth scan with more levels. So after a while, it found vulnerabilities in username, password, and databases mysql and machine. But we could not find the result as it stopped because of some miss entry in the file, so we verified with other tools, but we could not find the mismatch happening, due to which we could not see the result. So we did another method where instead of using a file, we directly used url combing with multiple parameters to skip options while processing which would increase execution time.



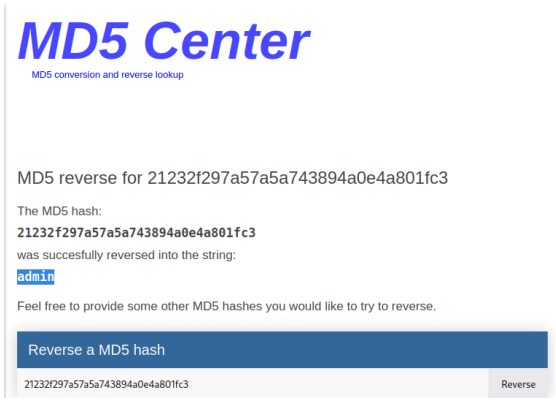
Let’s disassemble the parameters used in the second method for skipping options that are asked while doing the previous method, so the ‘-u’ parameter is used for the url, and ‘--forms’ to skip the option where it asks to test from user and pass, as we have seen previously mysql database was found so to use

this directly we used '—dbms-MySQL' parameter, and this saves a lot of execution time. 'batch' parameter enters random entries on both username and password field and '-flush-session' parameter is used to remove any previous sessions which would hinder the process. After that, '—random-agent' will use a random user agent, which has been asked in one of the options, '-D' parameter is used so that it directly uses the machine database, and '--level1 --risk3 ' parameter performs level 1 and risk 3 tests which are also one of the options. After that '-T' parameter is to use tables with the "login" type directly, and the '-C' parameter to directly use columns with the "password, role, username" type. And lastly, we dump everything in the machine database using the '—dump' parameter. And finally, we get the result shown in the report's results section.

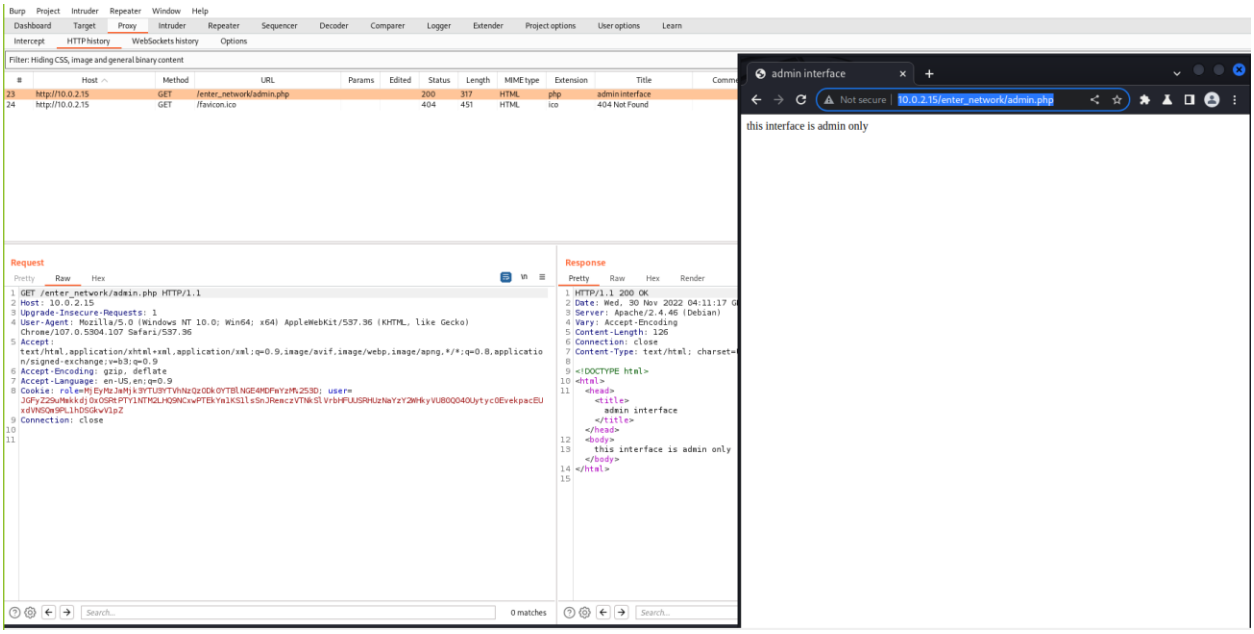
After this, we tried a few other approaches, and amongst them, we found the vulnerability and acquired the same result by doing one another method. So, when we were looking for different files we obtained from dirbuster, there was a file named exploit.html that caught sight. So, we accessed this webpage, and when we inspected the source page, we figured out that 'http://localhost/profile.php' is linked with submit query button. So, instead of using localhost, we modified it with the target machine's IP address, ' 10.0.2.15'.



We found only the first part of the flag and thought the second would be on another file just like this one, and we would get the second part. So again, we tried to figure out which file would have a second part. It took us a lot of time, and then suddenly, we realized for the first and second methods, we used the 'enter_network' file, and the cookie part was crucial in both approaches, so we again used burpsuite to track HTTP GET and POST request, and we examined that role has been encoded in the form of the MD5 hash. So we went on an online MD5 converter, converted the encoded text, and found out it is named 'admin'.



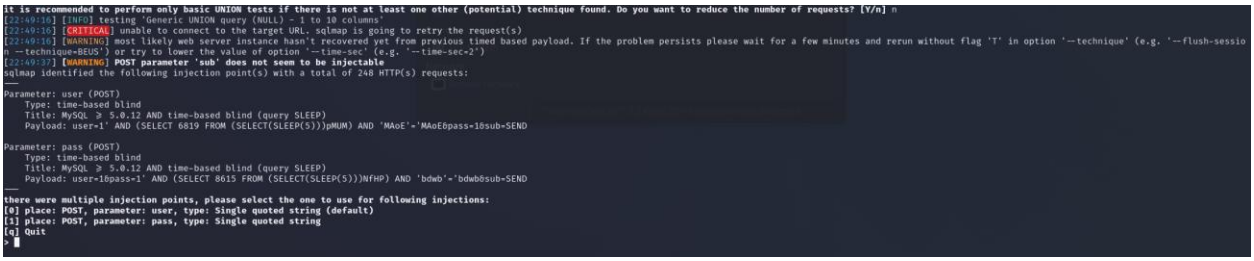
So, to edit the part, we transferred the details to the repeater module and changed the encoded version to 'admin,' but we were again stuck. So, again we looked at the files revealed by the dirbuster tool and found a file named 'enter_network/admin.php'.



When we accessed the webpage, it was written as 'this interface is admin only'. So we changed the encoded version to 'admin' and got the second part shown in the report's results section.

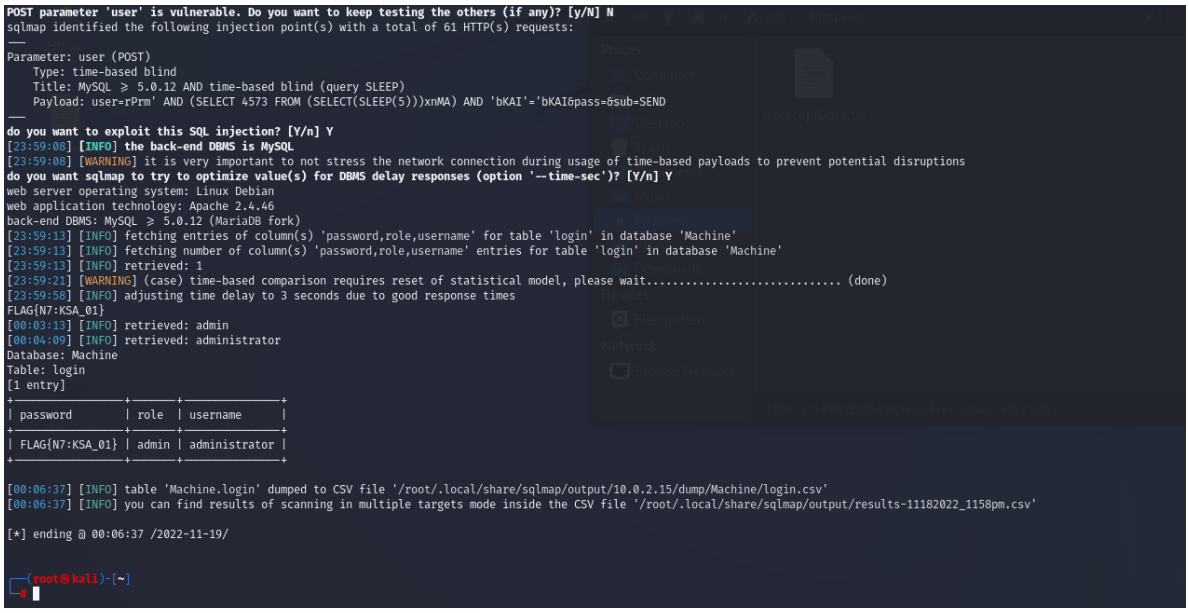
Exploits:

In the beginning, we had significantly less knowledge of the web server. Still, by thoroughly analyzing this vulnerable machine's components, we found that it had SQL Injection Vulnerability. So, by using sqlmap, we discovered that the vulnerability was at username and password, which had time-based sql injection vulnerability; that is, when sql query gets submitted, there was a database pause/delay for getting the result. Hence, sqlmap used payload for both vulnerabilities where it triggered 5ms time delays when sql query gets executed and looked for the difference to respond.

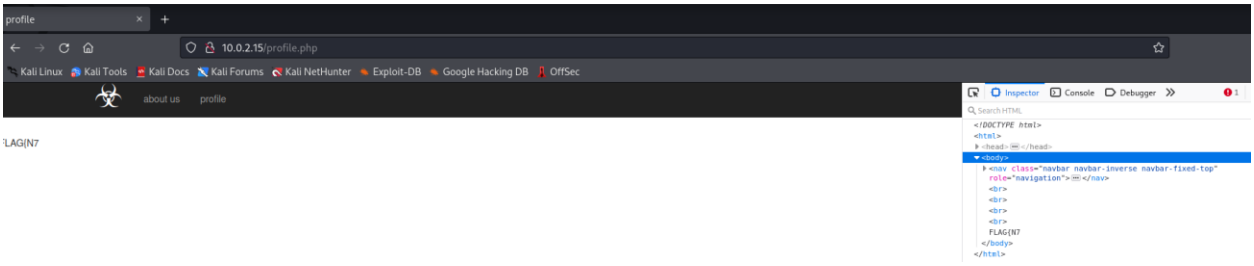


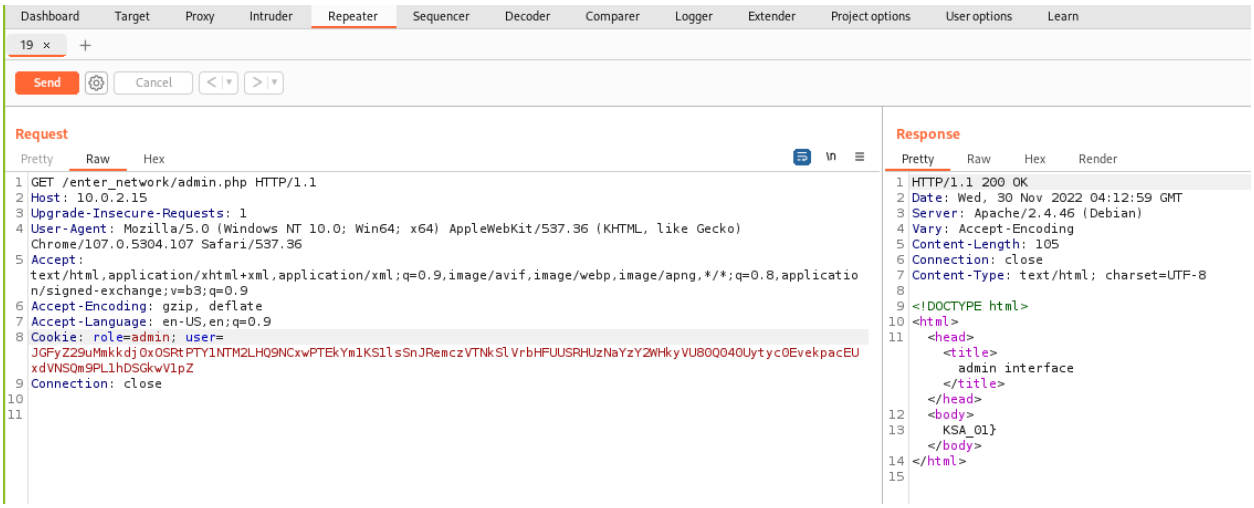
Results:

From the first approach, when we used the '.txt' file in sqlmap, we got many options for proceeding with the process and found vulnerabilities. Still, we couldn't get the result as it stopped because of some miss entry in the file, which we verified with other tools, but we could not find the mismatch happening, due to which we could not see the result. So we did another method where instead of using the file, we directly used url combing with multiple parameters to skip options while processing, which would increase execution time, and by this approach, we found the result flag, which is shown below.



After we got the flag from the second approach, we tried a few other techniques, and amongst them, another method gave us a similar resulting flag, but it was split into two parts. The first part was captured by modifying the 'exploit.html' file with the target machine's IP address, and the second part was captured by renaming the role section of the 'enter_network/admin.php' file to 'admin'.





Conclusions and Alternative approach to experiment in the future:

The program's tiniest vulnerability can cause problems for a person/organization. As we saw from this project, at the start, we had less knowledge of the targeted machine, ' web machine n7'. But then, by using network scanning tools, we found its IP address, and by using a directory traversal attack, we found hidden files/directories. And after a thorough analysis of this vulnerable machine's components, we narrow it down to sql-injection vulnerability (time-based sqli), and by exploiting the webpage, we found the flag. We tried 11 approaches, and 2 gave us the result.

We were planning to use the Metasploit framework approach, but we had less time, so in the future, we will try to exploit the 'Apache httpd 2.4.46' server, which is used by the target machine (web machine n7) by using "CVE-2022-31813" vulnerability with proper payload where the server couldn't use headers back on client side connection due to which IP based authentication gets bypassed or with other exploits, and after that once we get root access we could find the flag located in one of the files.