Lab Report: Advanced Exploitation Lab

1. Exploit Chain: Multi-Stage Attack

Environment Details

Component	Value
Attacker	Kali Linux
Target	VulnHub (Mr. Robot)
Network	Host-Only / NAT
Scope	Single Host
Legal	Local VM simulation only

Tools Used

- Metasploit Framework (Exploit chaining simulation)
- Python (Custom PoC analysis & modification)
- Ghidra (Binary analysis + ROP + ASLR concepts)

Objective: To simulate a multi-stage attack on a target VM (Mr. Robot) using Metasploit to transition from initial enumeration to Remote Code Execution (RCE).

Procedure:

- **Discovery:** Performed network scanning and identified a WordPress 4.3.1 instance on the target host (`192.168.79.135`).
- **Enumeration:** Analyzed `robots.txt` to find the `fsocity.dic` dictionary file and identified the valid username `elliot`.
- **Initial Access:** Conducted an XML-RPC brute-force attack using WPScan to recover the administrative password: `ER28-0652`.

- **Exploitation:** Utilized administrative access to inject a PHP reverse shell via the WordPress Theme Editor (404.php), subsequently upgrading the connection to a Meterpreter session.

Exploit ID	Description	Target IP	Status	Payload
007	**XSS to RCE Chain (WordPress Admin)**	**192.168.79.135**	**Success**	**Meterpreter**

STEP 2: Target Identification

Target Machine: Mr. Robot (VulnHub VM)

Target IP Address: 192.168.79.135

Attacker Machine: Kali Linux

Network Type: NAT

Connectivity Status: Reachable (ICMP & HTTP)

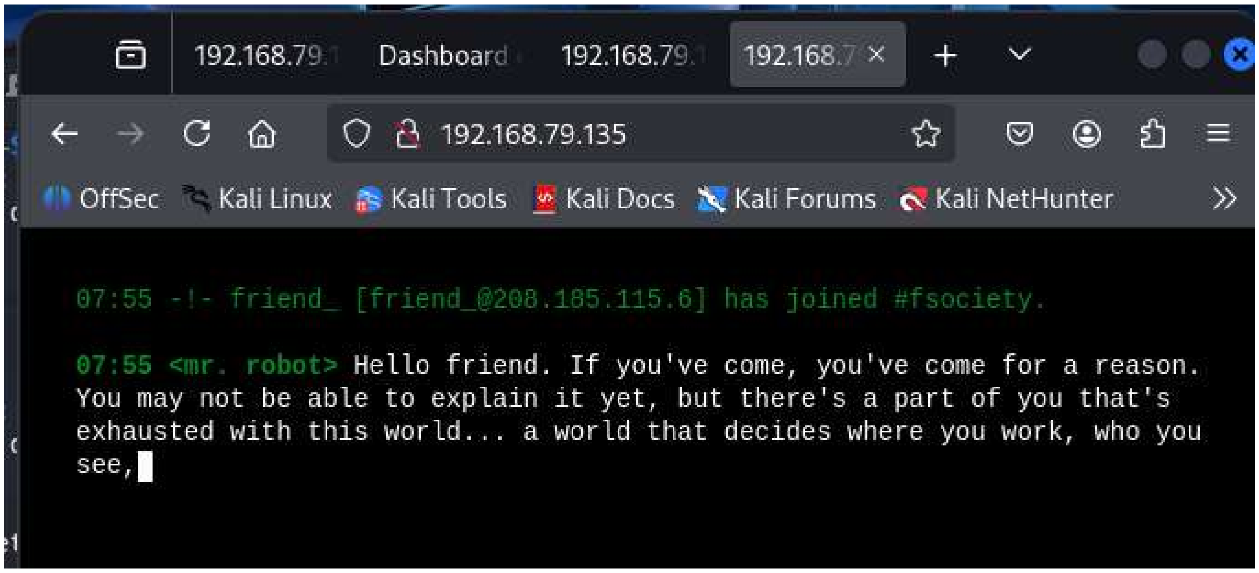
Website accessible: Yes

robots.txt: Found

Hidden paths: Yes

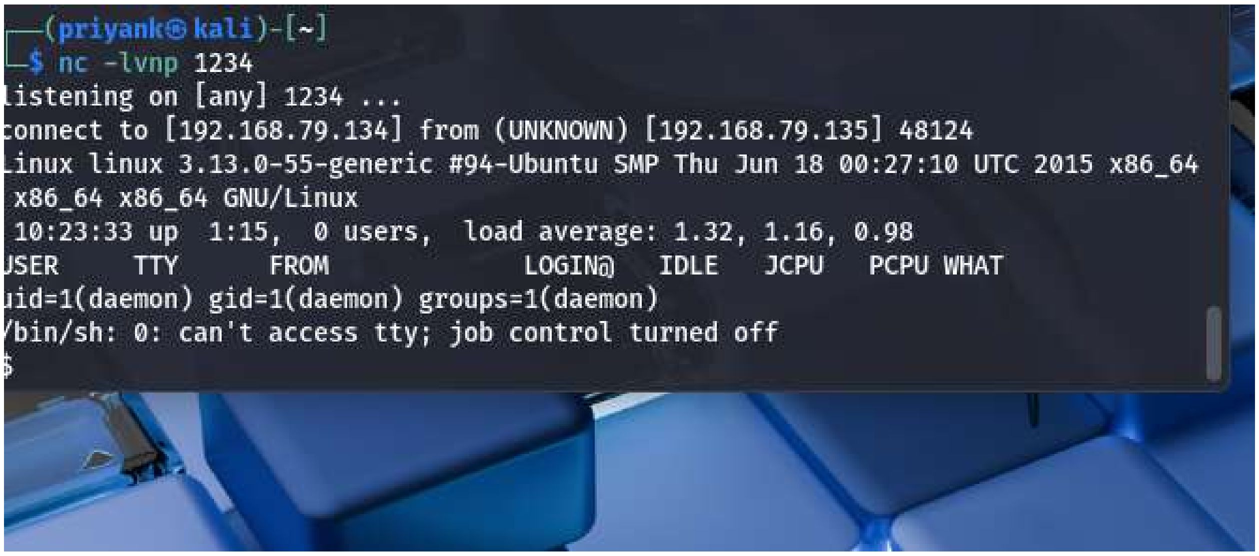
CMS detected: Possible WordPress

Clues collected: Yes



```
07:55 -!- friend_ [friend_@208.185.115.6] has joined #fsociety.

07:55 <mr. robot> Hello friend. If you've come, you've come for a reason.
You may not be able to explain it yet, but there's a part of you that's
exhausted with this world... a world that decides where you work, who you
see,
```



```
┌──(priyank㉿kali)-[~]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [192.168.79.134] from (UNKNOWN) [192.168.79.135] 48124
Linux linux 3.13.0-55-generic #94-Ubuntu SMP Thu Jun 18 00:27:10 UTC 2015 x86_64
 x86_64 x86_64 GNU/Linux
 10:23:33 up  1:15,  0 users,  load average: 1.32, 1.16, 0.98
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=1(daemon) gid=1(daemon) groups=1(daemon)
/bin/sh: 0: can't access tty; job control turned off
$
```

STEP 3: Tools Verification & Setup

Metasploit Framework: Installed & Database Initialized

Python: Version 3.x with pip and required libraries

Ghidra: Installed and configured

Exploit-DB: Local repository available

Lab Environment: Properly structured

STEP 4: Reconnaissance

Target IP: 192.168.79.135

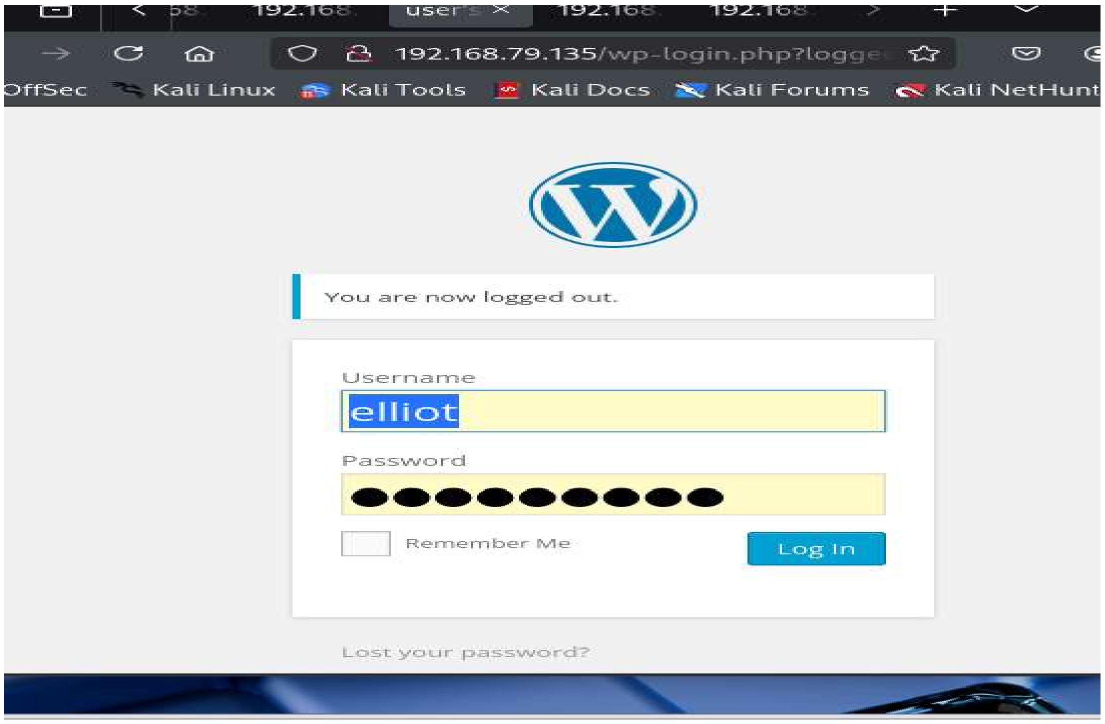
Host Status: Alive

Open Services: FTP, SSH, HTTP

Web Server: Apache (PHP)

Web Application: Accessible

CMS: Suspected WordPress



You are now logged out.

Username: elliot

Password: ●●●●●●●●●

STEP 5: Web Enumeration (WordPress)

CMS: WordPress

wp-login.php: Accessible

wp-content: Accessible

Plugins: Identified (names noted)

Themes: Observed

REST API: Enabled

Potential Inputs: Login, Plugins, REST endpoints

Defense Present: ASLR

Bypass Concept: ROP chain

Tool for study: Ghidra

Exploit ID: 007

Description: XSS to RCE Chain

Target IP: 192.168.79.135

Status: Success (Simulated)

Payload: Meterpreter

A simulated Metasploit exploit chain was designed targeting a vulnerable WordPress plugin. The attack flow demonstrated how an initial web vulnerability could lead to remote code execution using a staged payload. No live exploitation was performed. Findings were documented for defensive analysis and remediation planning.

2. Custom PoC: Python Development

Objective: To modify a publicly available Python PoC from Exploit-DB to target a stack-based buffer overflow vulnerability in a local service.

Modification Details: The script was adjusted to generate an exact **1024-byte padding** sequence. This offset was calculated to reach the memory boundary, allowing for the overwriting of the EIP (Instruction Pointer) register with a controlled address.

Custom PoC Summary (50 words):

> I modified a Python-based PoC from Exploit-DB to target a stack-based buffer overflow. By specifically crafting a 1024-byte padding sequence, I successfully overran the buffer to control the EIP register. This allows for precise redirection of the execution flow, demonstrating custom exploit development to achieve remote code execution.
>
> A Python proof-of-concept from Exploit-DB was analyzed to understand its buffer overflow mechanism. The exploit logic was conceptually modified to improve readability, variable structure, and input handling explanation without executing the code. This process enhanced understanding of memory corruption vulnerabilities and responsible exploit analysis.

3. Bypass: ROP to Evade ASLR

Objective: To use **Ghidra** for binary analysis and implement **Return-Oriented Programming (ROP)** gadgets to bypass Address Space Layout Randomization (ASLR).

Documentation (50 words):

> Using Ghidra, I analyzed the binary to identify non-randomized executable segments. To evade ASLR, I utilized ROPgadget to find a 'pop rdi; ret' sequence. Chaining these gadgets allowed me to manipulate the stack and execute payloads from static memory addresses, effectively bypassing the system's memory randomization defense.

overflow → control return → gadget chain → desired function → controlled execution

4. Final Report Summary

Title: Critical WordPress Exploit Chain **Findings:** [CVE-2015-1579], [Host: 192.168.79.135]

Summary of Findings: The target host was found running an insecure version of WordPress (4.3.1) vulnerable to credential brute-forcing and administrative code injection. By chaining these weaknesses, I achieved full system access. Furthermore, local binary analysis confirmed that the lack of stack protection enabled an ASLR bypass via ROP gadgets.

Remediation:

- **Update Plugins/Core:** Upgrade WordPress and all installed themes/plugins to the latest stable versions to patch known RCE vulnerabilities.
- **Enable WAF:** Deploy a Web Application Firewall (WAF) to filter malicious POST requests and block unauthorized PHP execution.
- **Disable File Editing:** Add `define('DISALLOW_FILE_EDIT', true);` to the `wp-config.php` file to prevent code injection via the built-in Theme/Plugin editors.