Cisco Final Capstone Project- Ethical Hacker
Objectives
For this Final Capstone Activity, you will conduct a complete penetration test starting with reconnaissance and then launching exploits against vulnerabilities that you have discovered. Finally, you will propose remediation for the exploits.

This assessment is in the form of a cybersecurity **capture the flag** exercise. You will use your ethical hacking skills to locate files that contain flag values. You will then report the flag values that you found as part of the assessment.

In this simulation of an ethical hacking engagement, you will use tools to exploit vulnerabilities that you discover in order to reach a goal. This can entail a trial-and-error approach that requires persistence and may include a degree of struggle. For your own skill development, working through this struggle can be productive. If you are completely stuck, ask your instructor for assistance.

NOTE: We will be using Kali VM customized for the Ethical Hacker course

- **Challenge 1** – Use SQL injection to find a flag file.
- **Challenge 2** – Use web server vulnerabilities to investigate directories and find a flag file.
- **Challenge 3** – Exploit open Samba shares to access a flag file.
- **Challenge 4** – Analyze a Wireshark capture file to find the location of a file containing flag information.

Background / Scenario

You have been hired to conduct a penetration test for a customer. At the conclusion of the test, the customer has requested a complete report that includes any vulnerabilities discovered, successful exploits, and remediation steps to protect vulnerable systems. You have access to hosts on the 10.5.5.0/24 and 192.168.0.0/24 networks.

BrainStorm
Challenge 1: SQL Injection
In this part, you must discover user account information on a server and crack the password of **Bob Smith's** account. You will then locate the file that contains the Challenge 1 code and use **Bob Smith's** account credentials to open the file at 192.168.0.10 to view its contents.
Goal
Discover user account info on the server: reconnaissance and findings to get injection points
Injection points can either be in:
- login forms
- search bars
- URL parameters(e.g. ?id=1, ?user=bob)
- HTTP headers(user agent, referer, cookie)

Possible tools:
- browser- manually test fields with ‘ or “ to trigger sql errors
- burpsuite: intercept and modify HTTP requests
- sqlmap: automated SQL injection discovery and exploitation

Crack Bob’s password
- have the password hash (likely MD5, SHA1, bcrypt):

Possible tools:

- hashcat- GPU- accelerated cracking
- John the Ripper:

Access the file at 192.168.0.10 using Bob's credentials

Key Tips

- **Always note error messages:** they reveal DB type (MySQL, MSSQL, SQLite)
- **Try both GET and POST** injection points
- **Check for second-order injection** (stored, not just reflected)
- If the password is plaintext in the DB... lucky you!
- The file might be a .txt, .pdf, or protected archive: check what protocol is being used to serve it

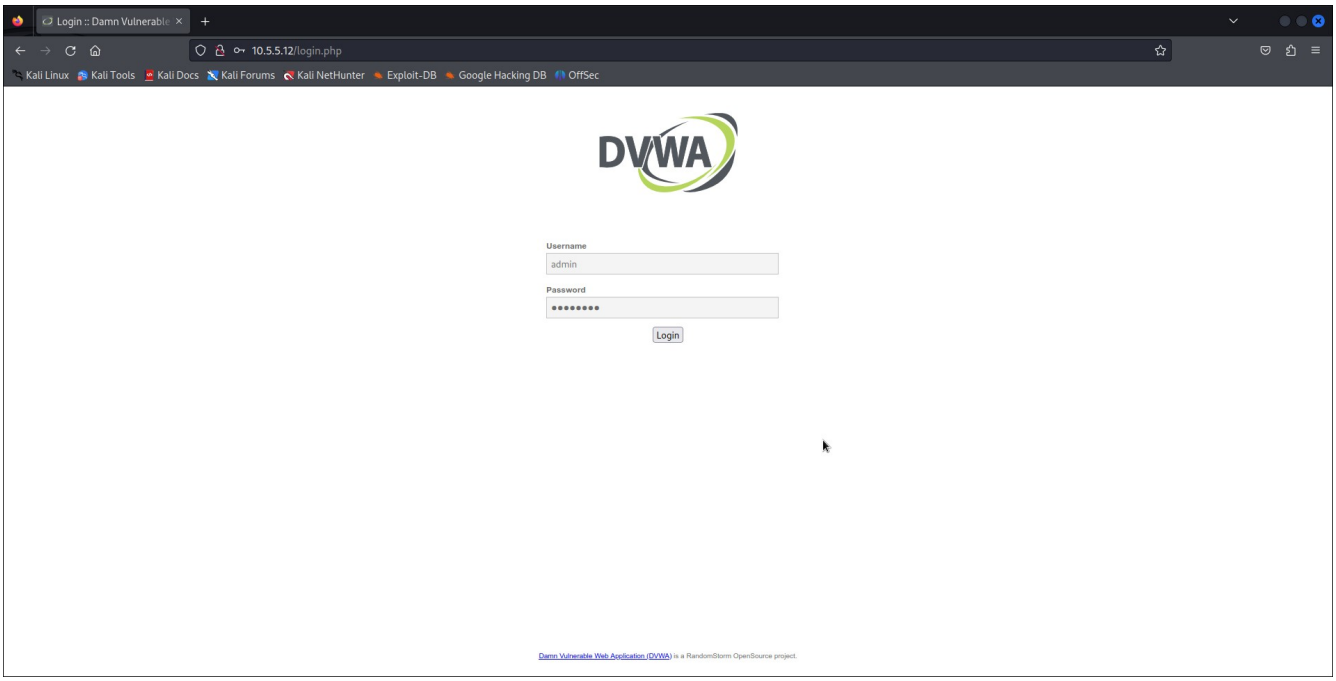
SOLUTION&Questions

Step 1: Preliminary setup

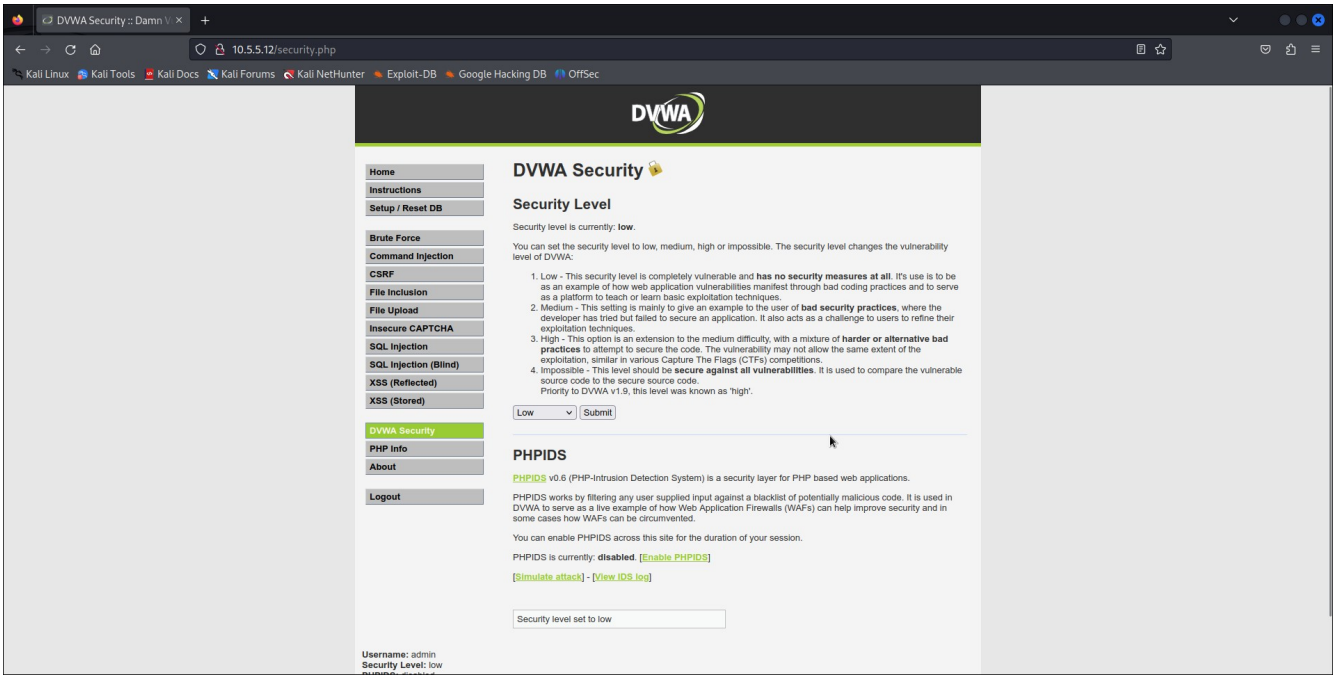
a. Open a browser and go to the website at 10.5.5.12(**DVWA website**).

Note: If you have problems reaching the website, remove the https:// prefix from the IP address in the browser address field.

b. Login with the credentials **admin / password**.



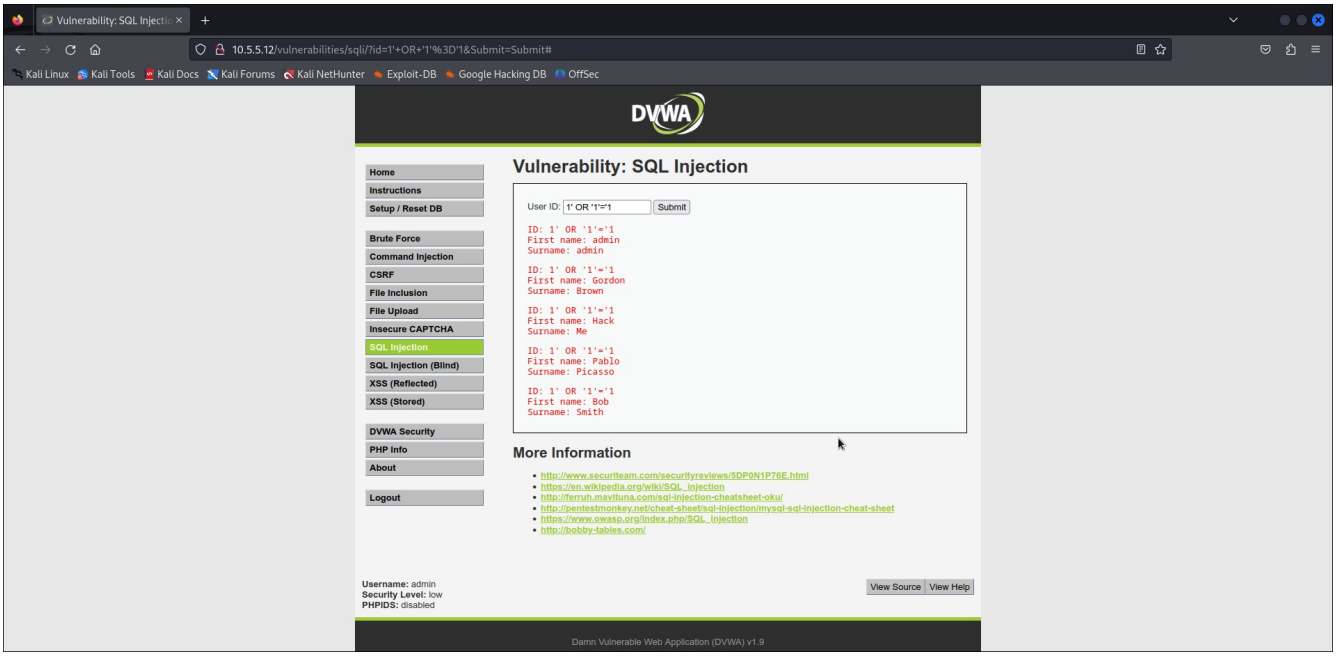
c. Set the DVWA security level to **low** and click **Submit**.



Step 2: Retrieve the user credentials for the Bob Smith's account.

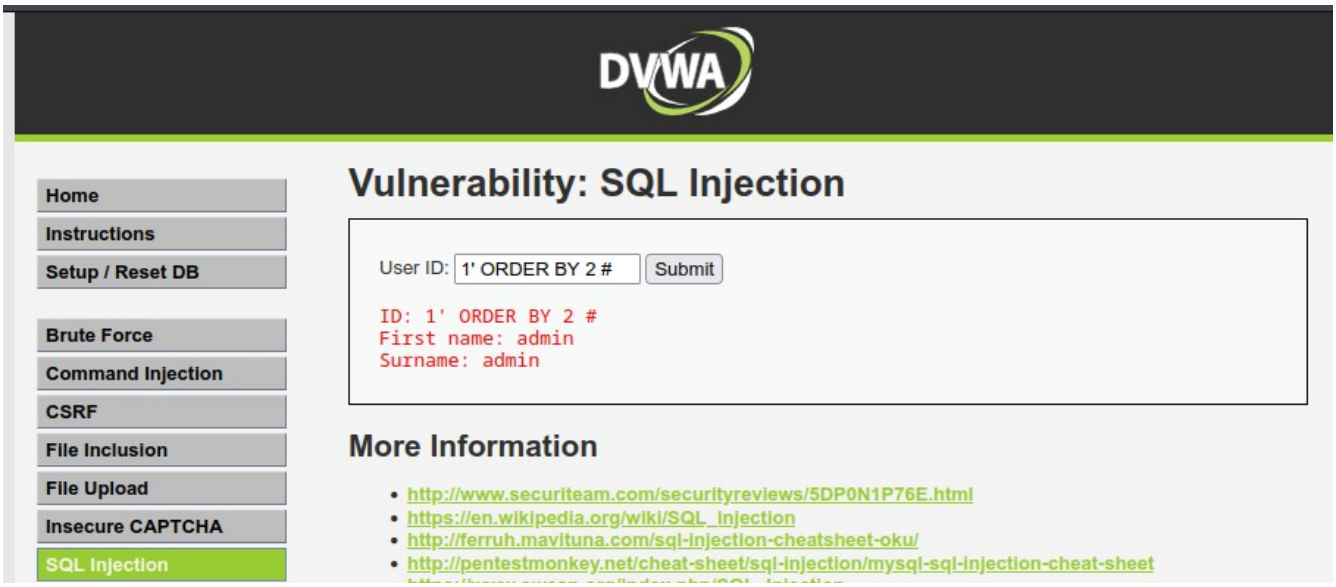
a. Identify the table that contains usernames and passwords.
b. Locate a vulnerable input form that will allow you to inject SQL commands.
c. Retrieve the username and the password hash for **Bob Smith's** account.

Initial testing used the payload **1' OR '1'='1** to confirm the site executes injected SQL. The payload terminates the surrounding string, adds an always-true condition, and comments out the remainder of the query.



The response confirms **SQL injection** is possible: the application processed an always-true expression and returned database results.

2. Use ORDER BY probes to detect the number of returned columns. For example: (1' ORDER BY 1 #) and 1' ORDER BY 3 #. The error Unknown column '3' in 'order clause' shows the query uses two columns, which informs subsequent UNION payloads.

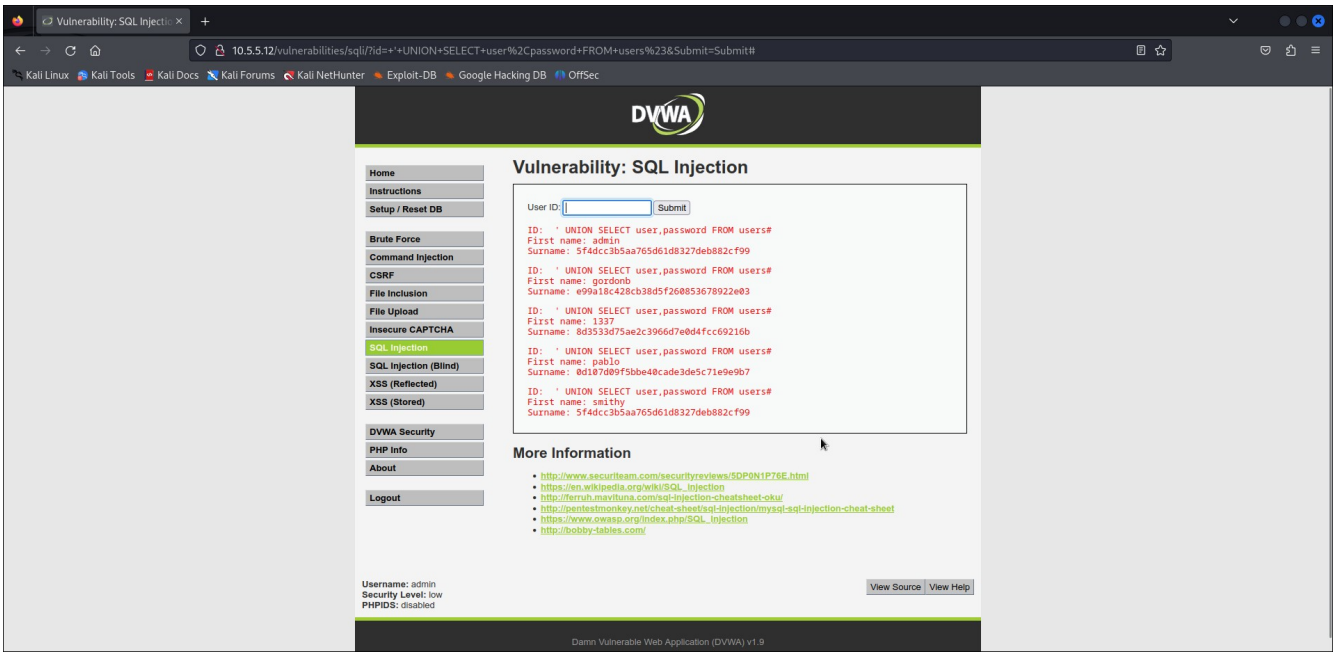



```
Unknown column '3' in 'order clause'
```

Use UNION-based payloads to enumerate database properties. Example:

Command	What it does	Output
1' OR '1'='1	confirm the site executes injected SQL.	application processed an always-true expression and returned database results.
1' ORDER BY 2 #	probes to detect the number of returned columns	Return the DB results
' UNION SELECT database(),NULL#	Get current database name	Returns the DB in our case: dvwa
' UNION SELECT table_name,NULL FROM information_schema.tables WHERE table_schema='dvwa'#	List all tables in dvwa	Returns users, guestbook
' UNION SELECT user,password FROM users#	Dump all usernames and password hashes	Returns usernames and hashed passwords

Command: ' UNION SELECT user,password FROM users# → Dump all usernames & password including Bob's username and hashed password

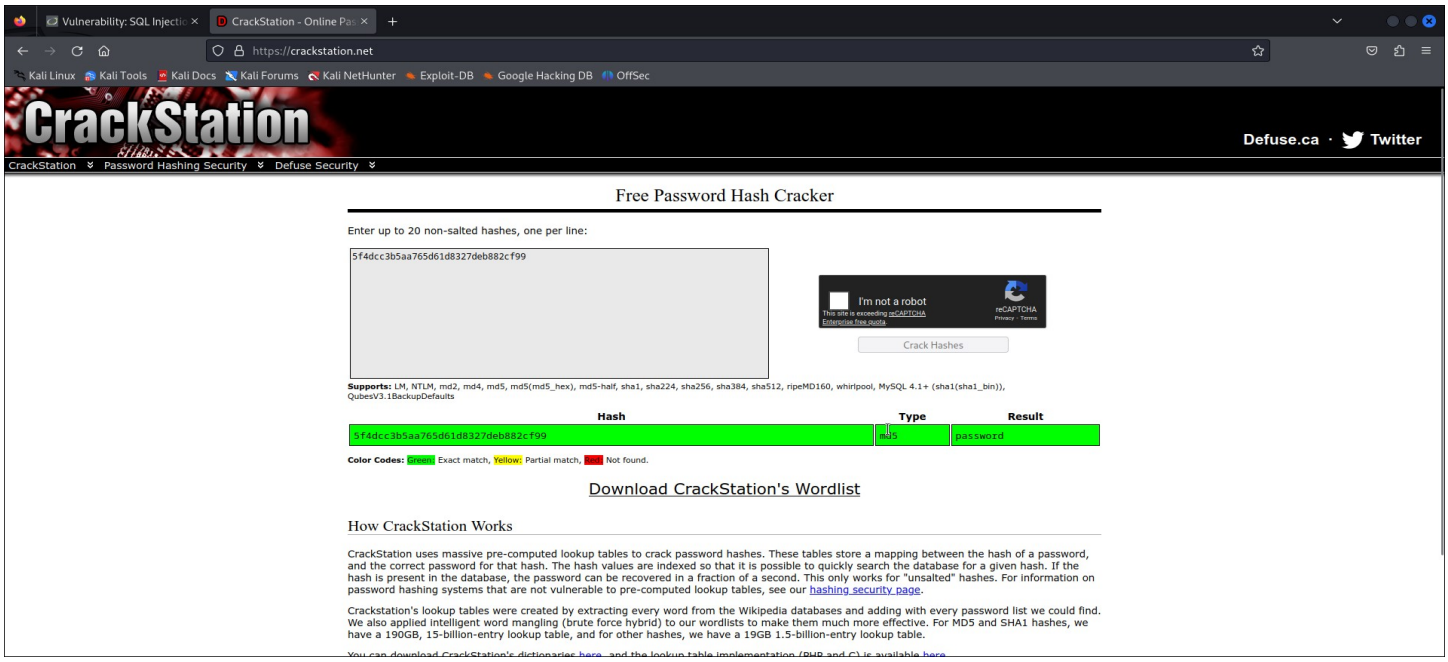


Step 3: Crack Bob Smith's account password.
Use any password hash cracking tool desired to crack **Bob Smith**'s password

What is the password of Bob Smith's account? **Password**

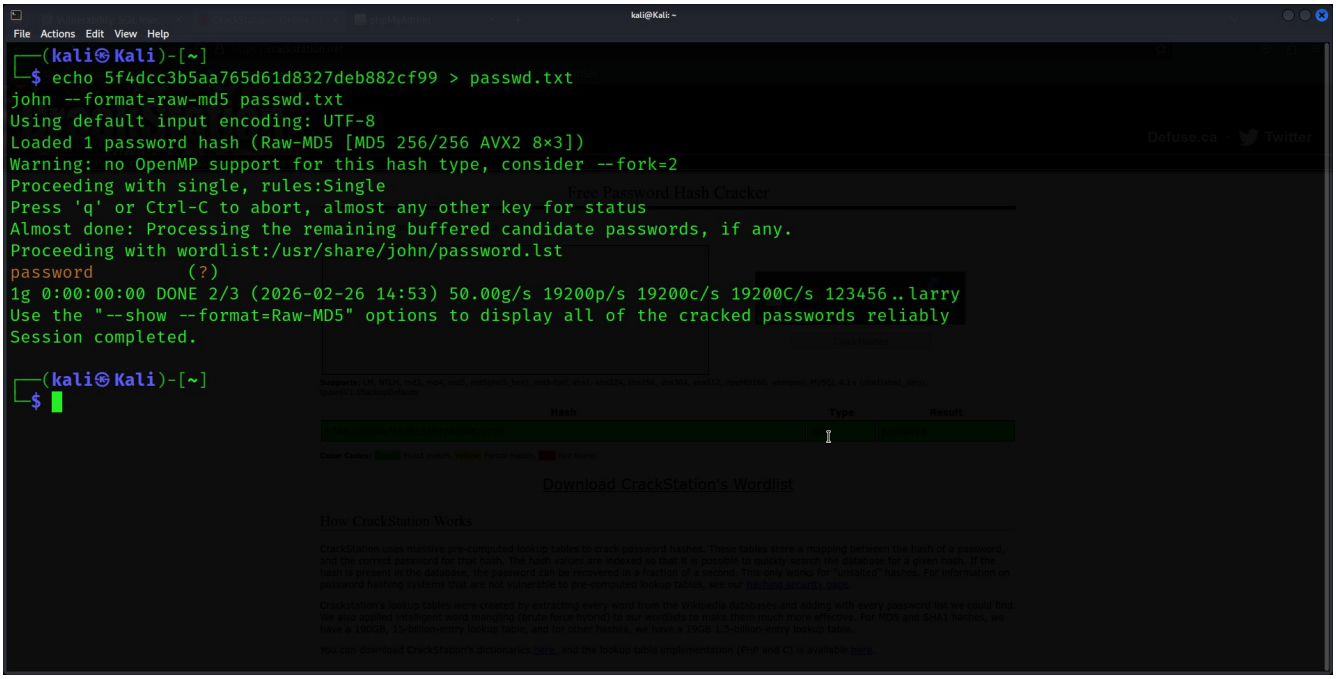
I recovered the plaintext password from the MD5 hash using two approaches: an **online hash lookup** and an **offline cracking tool.**

Part 1: Crackstation(online hash lookup)



Part 2 — John the Ripper (offline cracking tool)

Paste the hash password in a .txt file and use john to crack it as shown below.



Step 4: Locate and open the file with Challenge 1 code.

a. Log into **192.168.0.10** as **Bob Smith**.
b. Locate and open the flag file in the user's home directory.

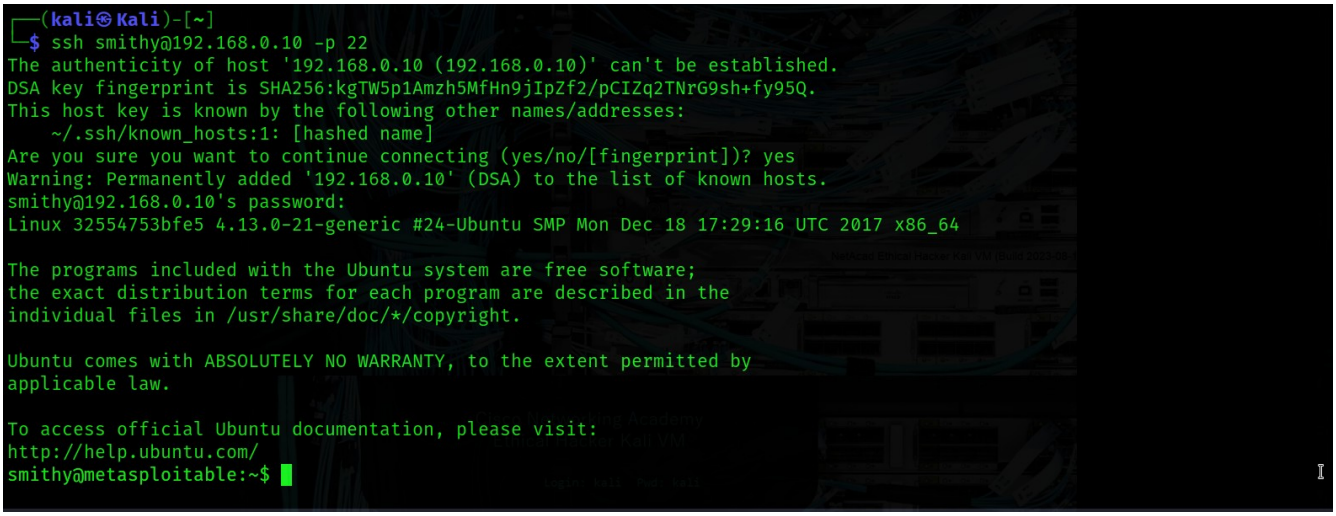
Ping to confirm communication:

```
┌──(kali㉿Kali)-[~]
└─$ ping -c 4 192.168.0.10
PING 192.168.0.10 (192.168.0.10) 56(84) bytes of data.
64 bytes from 192.168.0.10: icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 192.168.0.10: icmp_seq=4 ttl=64 time=0.032 ms

── 192.168.0.10 ping statistics ──
4 packets transmitted, 4 received, 0% packet loss, time 3083ms
rtt min/avg/max/mdev = 0.032/0.033/0.035/0.001 ms

┌──(kali㉿Kali)-[~]
└─$
```

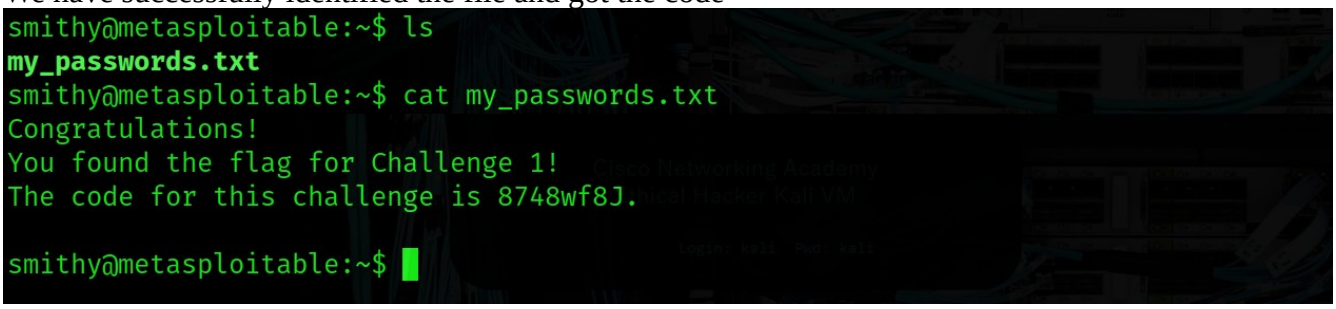
Scan using **nmap** to confirm open ports and their services

```
File  Actions  Edit  View  Help

┌──(kali㉿Kali)-[~]
└─$ nmap -p- -sV 192.168.0.10
Starting Nmap 7.94 ( https://nmap.org ) at 2026-02-26 15:03 UTC
Nmap scan report for metasploitable.pc (192.168.0.10)
Host is up (0.00017s latency).
Not shown: 65514 closed tcp ports (conn-refused)
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         vsftpd 2.3.4
22/tcp    open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet      Linux telnetd
25/tcp    open  smtp        Postfix smtpd
80/tcp    open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind     2 (RPC #100000)
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp   open  exec        netkit-rsh rexecd
513/tcp   open  login?
514/tcp   open  shell       Netkit rshd
1099/tcp  open  java-rmi    GNU Classpath grmiregistry
1524/tcp  open  bindshell   Metasploitable root shell
2121/tcp  open  ftp         ProFTPD 1.3.1
3306/tcp  open  mysql       MySQL 5.0.51a-3ubuntu5
3632/tcp  open  distccd     distccd v1 ((GNU) 4.2.4 (Ubuntu 4.2.4-1ubuntu4))
5432/tcp  open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
6667/tcp  open  irc         UnrealIRCd (Admin email admin@Metasploitable.LAN)
6697/tcp  open  irc         UnrealIRCd
8787/tcp  open  drb         Ruby DRb RMI (Ruby 1.8; path /usr/lib/ruby/1.8/drb)
35961/tcp open  java-rmi    GNU Classpath grmiregistry
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; 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 129.12 seconds

┌──(kali㉿Kali)-[~]
└─$
```

*We can use **ssh** to login*
We retrieved our username and password early
username: **smithy**
password: **password**
ssh smithy@192.168.0.10 -p 22



We have successfully logged in and we can confirm it is a metasploitable
We have successfully identified the file and got the code



What is the name of the file with the code? **my_passwords.txt**
What is the message contained in the file? Enter the code that you find in the file: **8748wf8J.**

Step 5: Research and propose SQL attack remediation.

What are five remediation methods for preventing SQL injection exploits?

1. **Parameterized Queries (Prepared Statements):** The most effective defense. Instead of concatenating user input into SQL strings, use placeholders that the database treats strictly as data, never as executable code.
2. **Stored Procedures:** Encapsulate SQL logic in pre-compiled database procedures. When implemented correctly (without dynamic SQL inside them), they separate code from data and prevent injection by design.
3. **Input Validation & Allowlisting:** Validate all user-supplied input against strict rules — expected data type, length, format, and character set. Prefer **allowlisting** (only permit known-good values) over **denylisting** (blocking known-bad characters), since attackers continuously find new bypass techniques.
4. **Least Privilege (Database Account Hardening):** Ensure the database account used by your application has only the minimum permissions necessary (e.g., **SELECT** only if writes aren't needed; no access to system tables or admin functions). This limits the blast radius if an injection does occur an attacker can't drop tables or read sensitive system data.
5. **Web Application Firewall (WAF):** Deploy a WAF to detect and block common SQL injection patterns in HTTP traffic before they reach your application. While not a substitute for secure coding practices, a WAF provides an important additional layer, especially for legacy systems that are harder to recode. Rules can be tuned to your specific application profile.

Challenge 2: Web Server Vulnerabilities

In this part, you must find vulnerabilities on an HTTP server. Misconfiguration of a web server can allow for the listing of files contained in directories on the server. You can use any of the tools you learned in earlier labs to perform reconnaissance to find the vulnerable directories.

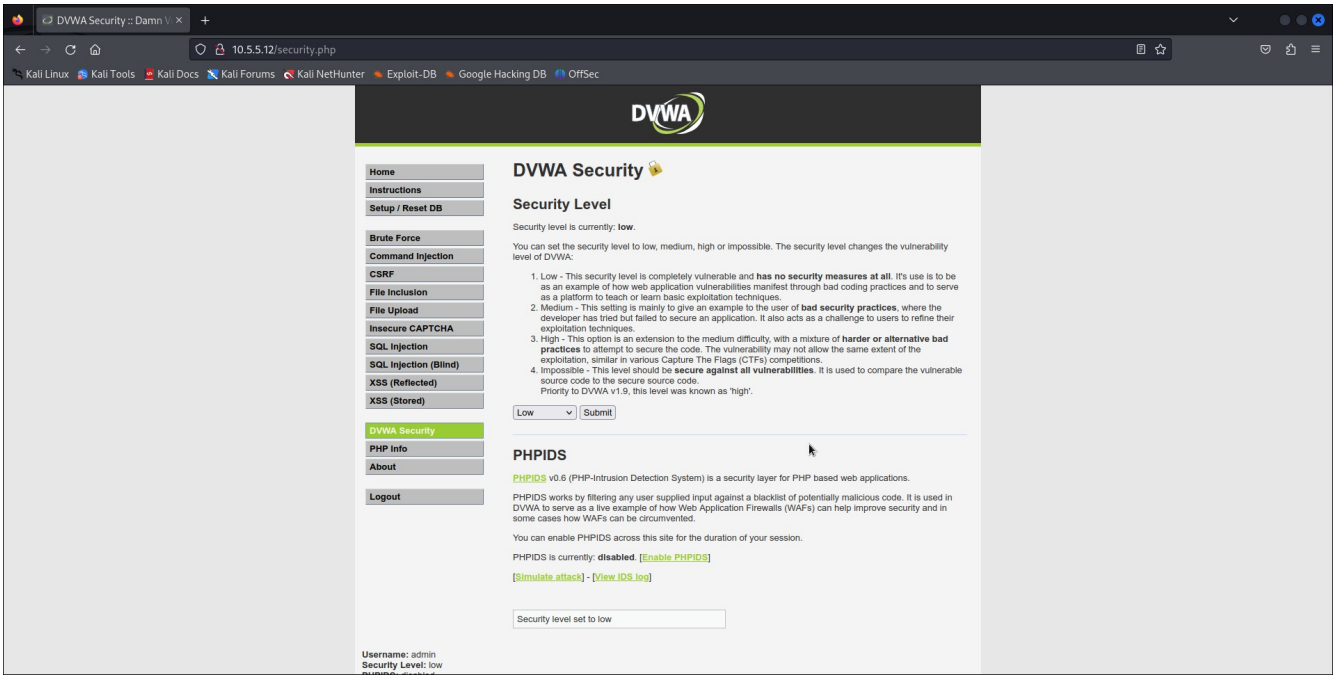
In this challenge, you will locate the flag file in a vulnerable directory on a web server.

Goal

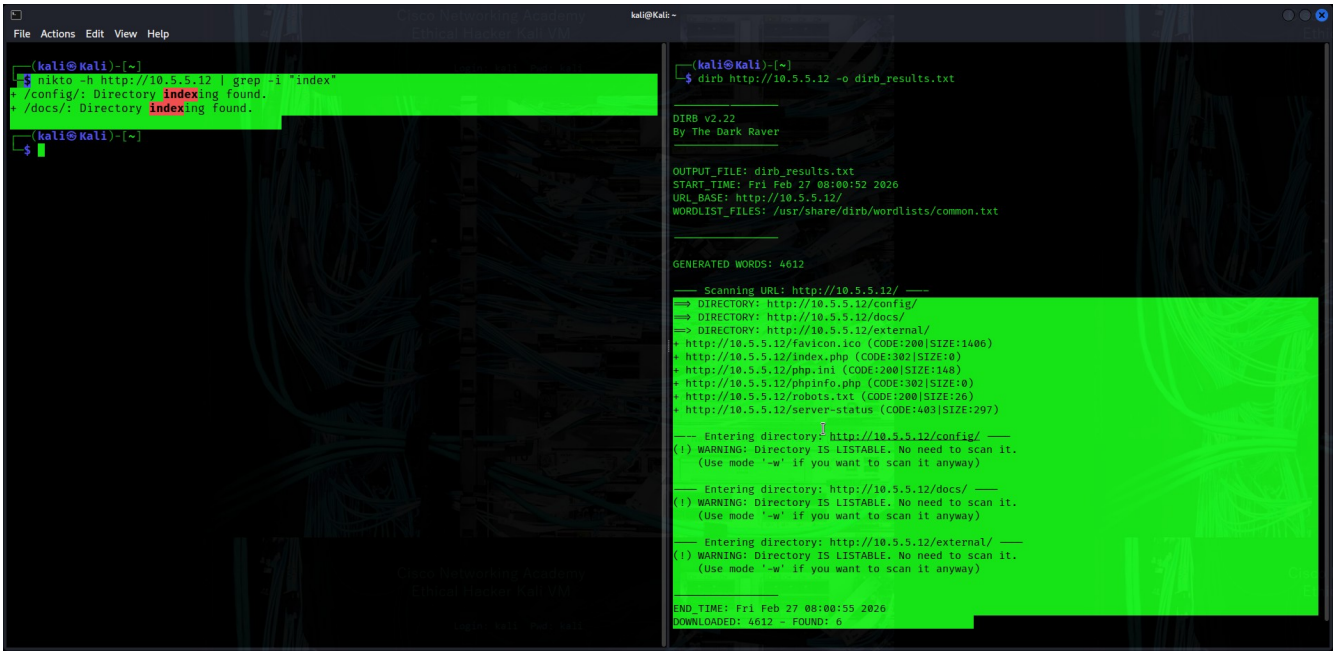
Find directory listing vulnerabilities on HTTP server and locate the flag file in a vulnerable directory on a web server.

Step 1: Preliminary setup

If not already, log into the server at 10.5.5.12 with the **admin / password** credentials.
Set the application security level to low.



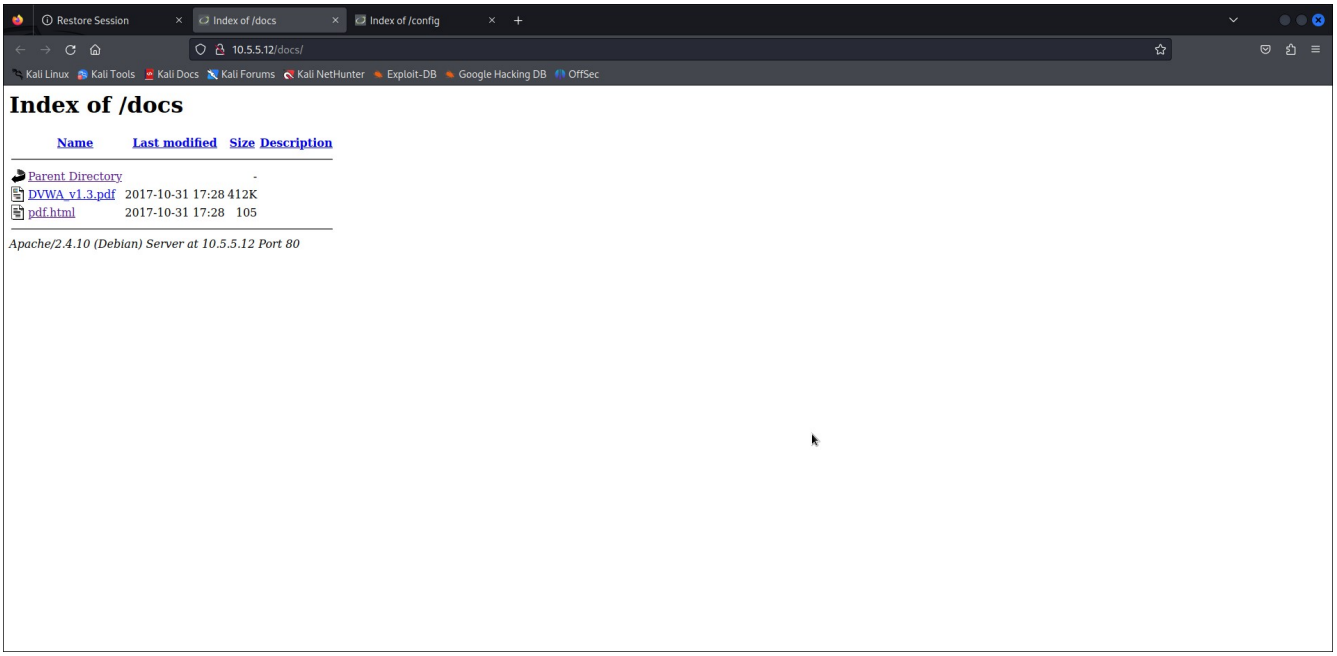
Step 2: determine which directories are viewable using a web browser and URL manipulation.
Perform reconnaissance on the server to find directories where indexing was found.
Run Nikto: Nikto will specifically report **"Directory indexing found"** for vulnerable directories
Run Dirb simultaneously: It will discover **hidden or unlinked files** and directories on a web server



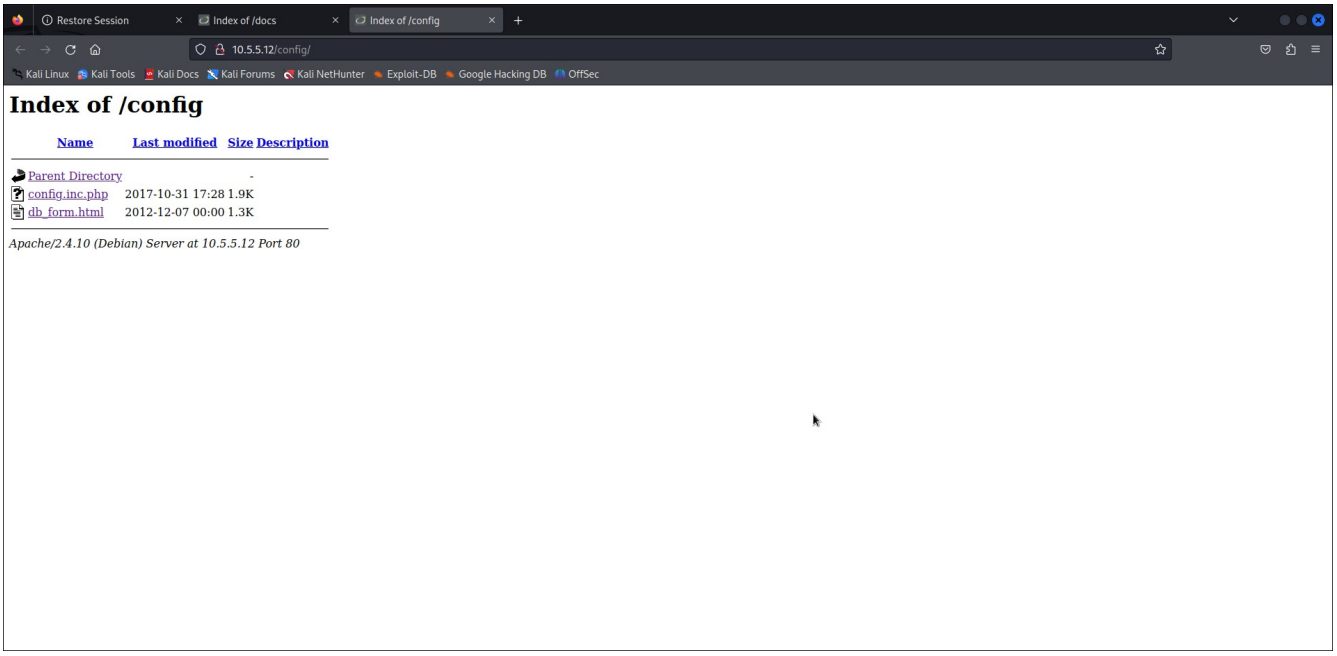
Common discoveries included /docs and /config directories often containing documentation or configuration artifacts.

Open the directories in the web browser to see the existing files

/docs contains:

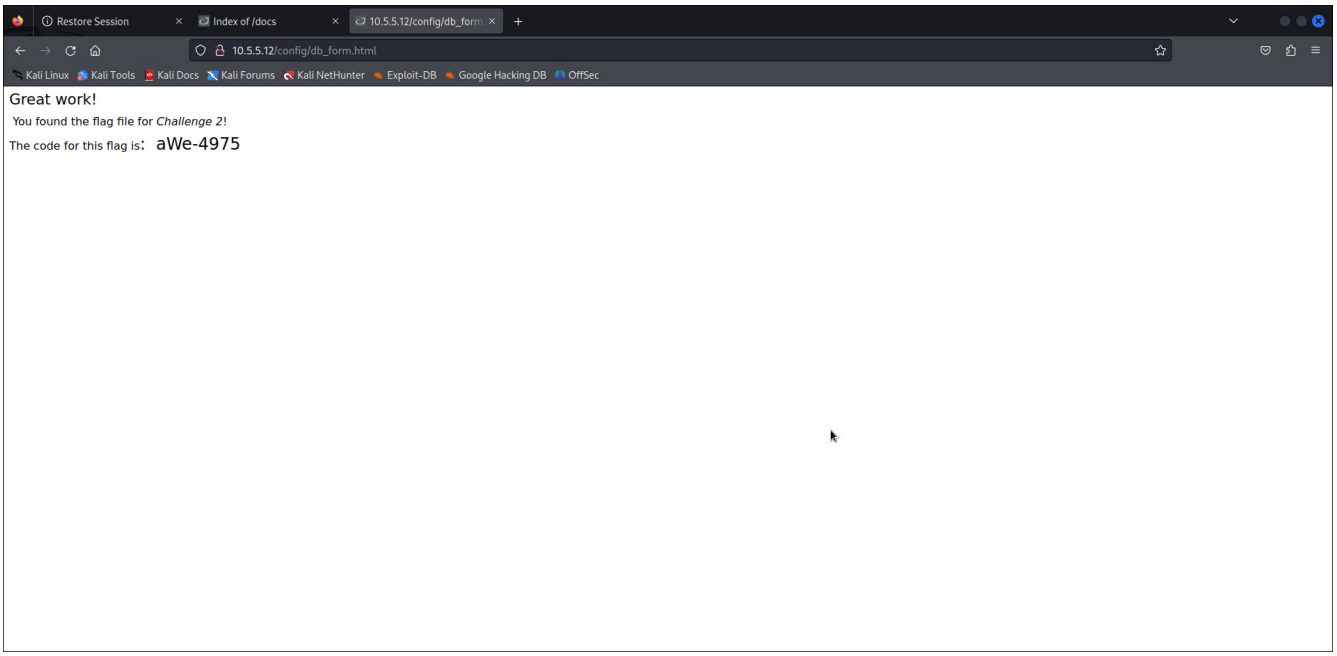


The /config contains



Navigate through all the files in the directories to find the flag location:

The flag was contained in the: http://10.5.5.12/config/db_form.html



In which two subdirectories can you look for the file? **/docs and /config**

What is the filename with the Challenge 2 code? **db_form.html**
Which subdirectory held the file? **Config**
What is the message contained in the flag file? Enter the code that you find in the file. **aWe-4975**

Step 4: Research and propose directory listing exploit remediation.

What are two remediation methods for preventing directory listing exploits?

- **Disable Directory Listing/Browsing:** Modify the web server configuration file (e.g., .htaccess for Apache or web.config for IIS) to turn off the indexing feature, which prevents the server from generating a listing of files.
- **Add Default Index Files:** Place a blank index.html, index.php, or similar file into all directories within the web root. If a user requests a directory, the server will serve this file instead of listing its contents.
- **Move Sensitive Data Outside Web Root:** Ensure sensitive, non-public files are stored outside the public web root directory, making them unreachable by browsing.
- **Configure Access Controls:** Use Access Control Lists (ACLs) to restrict access to specific directories and files.

Challenge 3: Exploit open SMB Server Shares

Total points: 25

In this part, you want to discover if there are any unsecured shared directories located on an SMB server in the 10.5.5.0/24 network.

Target Network: 10.5.5.0/24

Protocol: SMB (Server Message Block)
Goal: Find unsecured/open shared directories and locate the flag file

Key Questions to Think About:

Which hosts are existing?

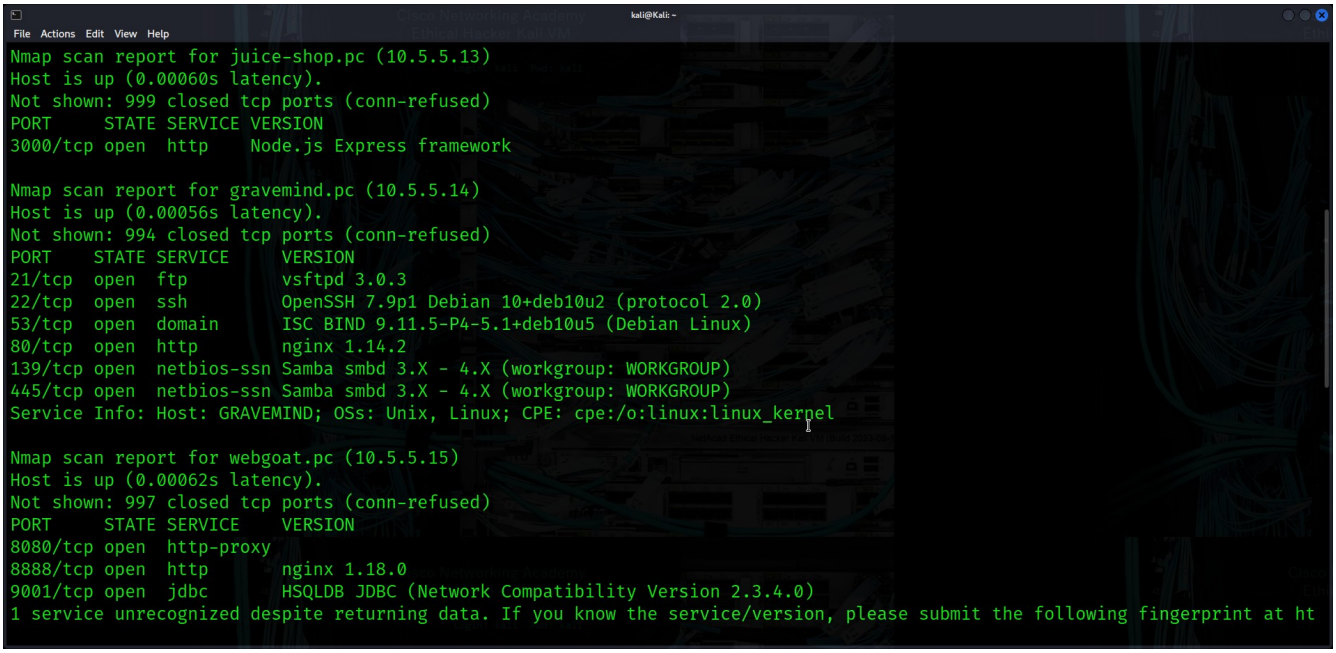
Which shares are open anonymously?

Which shares require credentials?

Where is the flag likely hidden?

Step 1: Scan for potential targets running SMB.

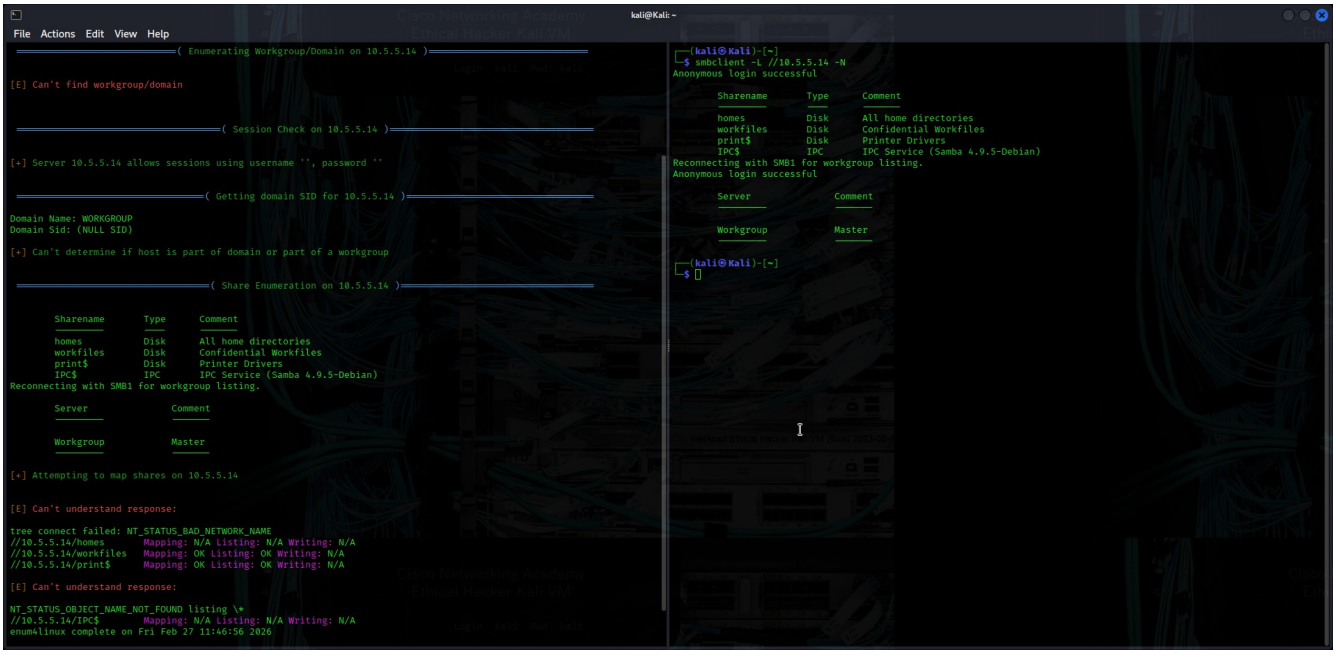
Run a service discovery scan with Nmap: **nmap -sV 10.5.5.0/24**



The scan reveals open ports including ports 139 and 445 indicating SMB file-sharing services.

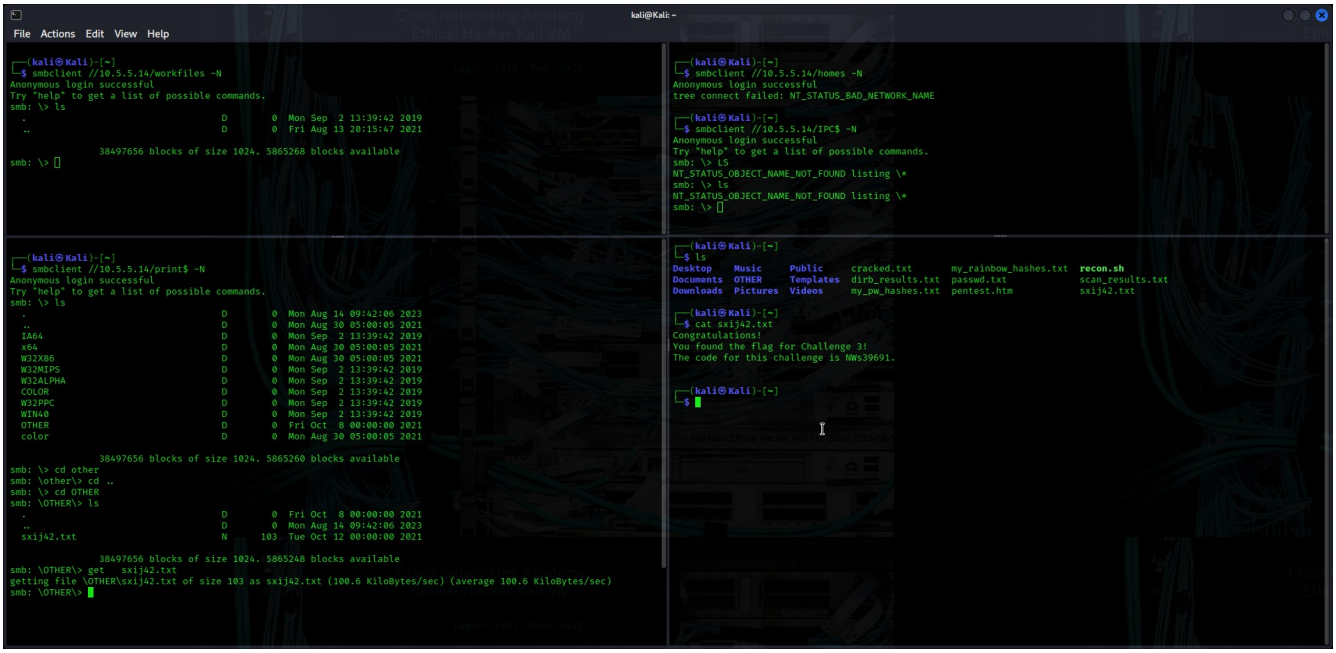
Step 2: Determine which SMB directories are shared and can be accessed by anonymous users.

Use **enum4linux** and **smbclient** to list shares and test anonymous access. For target: **10.5.5.14** that we found with Open SMB ports.



Step 3: Investigate each shared directory to find the file.

Connect to the share with **smbclient //<target IP>/<share> -N** and navigate using **ls**, **cd**, and **get**.



We were able to locate the flag in the print$ shared folder and extracted the .txt file to read it's contents as shown above.

In which share is the file found? **print$**
What is the name of the file with Challenge 3 code? **sxij42.txt**

Enter the code for Challenge 3 below: **NWs39691**

Step 4: Research and propose SMB attack remediation.

- **Disable Anonymous/Guest Access and Implement Proper Authentication**: Restrict SMB shares from allowing null sessions or anonymous guest access.
- **Firewall — Block SMB Ports** — Block ports 139 and 445 using UFW or iptables to prevent unauthorized external SMB connections

Challenge 4: Analyze a PCAP File to Find Information.

Total Points: **25**

As part of your reconnaissance effort, your team captured traffic using Wireshark. The capture file, **SA.pcap**, is located in the **Downloads** subdirectory within the **kali** user home directory.

Information

File: SA.pcap

Location: /home/kali/Downloads/SA.pcap

Tools: wireshark

Goal: analyze captured traffic to solve the challenge

Key Questions to Think About:

What protocols are in the capture?

Are there any credentials in plaintext?

Are there any suspicious packets?

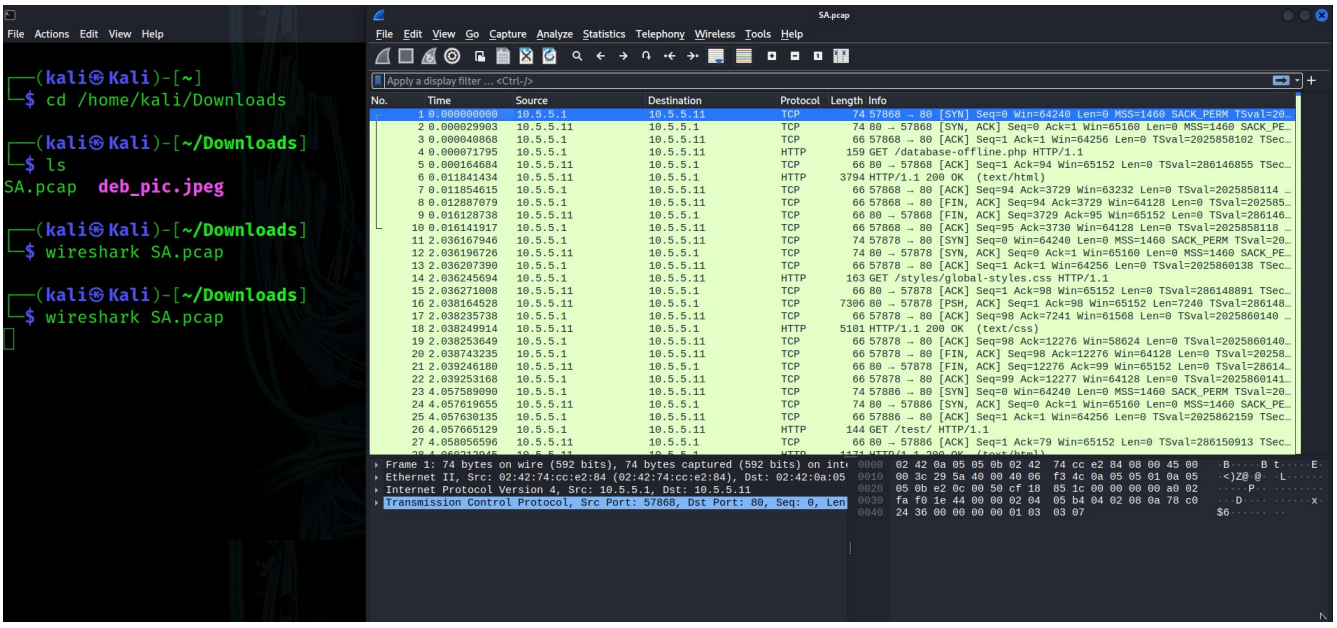
What conversations took place?

Step 1: Find and analyze the SA.pcap file.

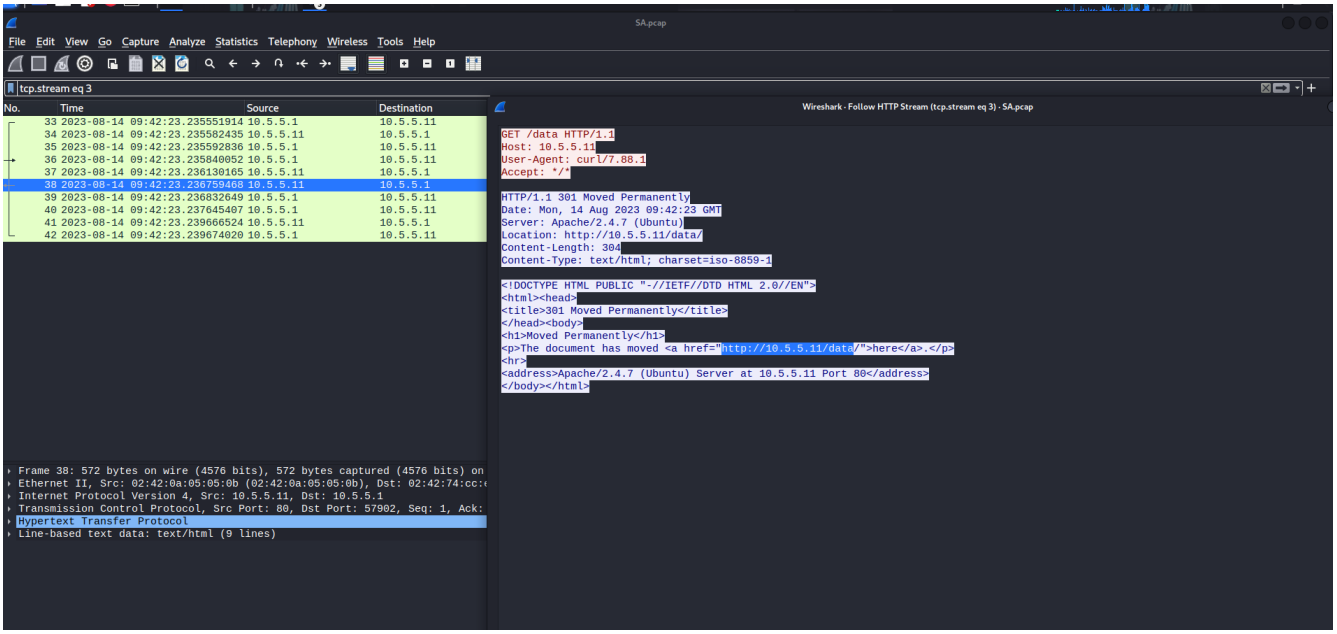
Analyze the content of the **PCAP** file to determine the IP address of the target computer and the URL location of the file with the Challenge 4 code.

Confirm the file exists

Open in wireshark: **wireshark SA.pcap**

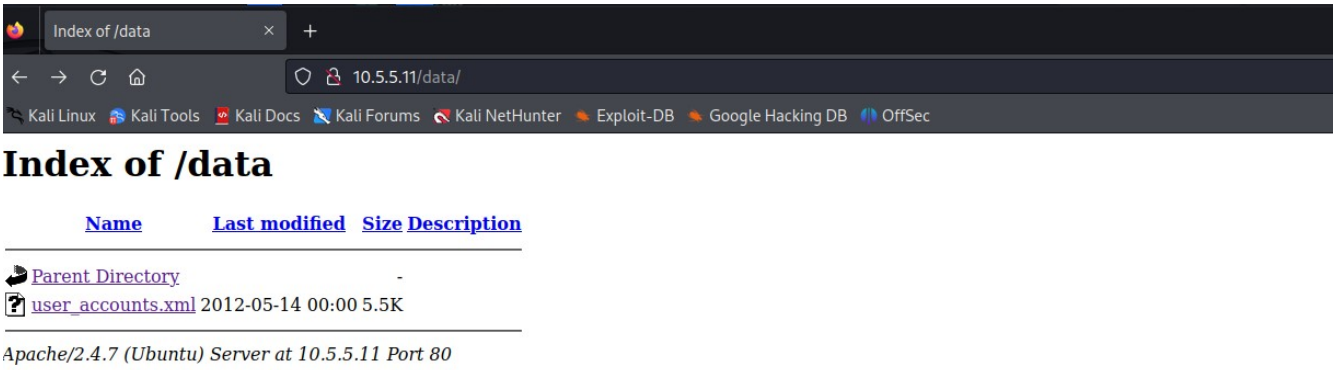


Inspect HTTP requests or follow TCP streams to identify URLs and file paths transmitted in plain text.

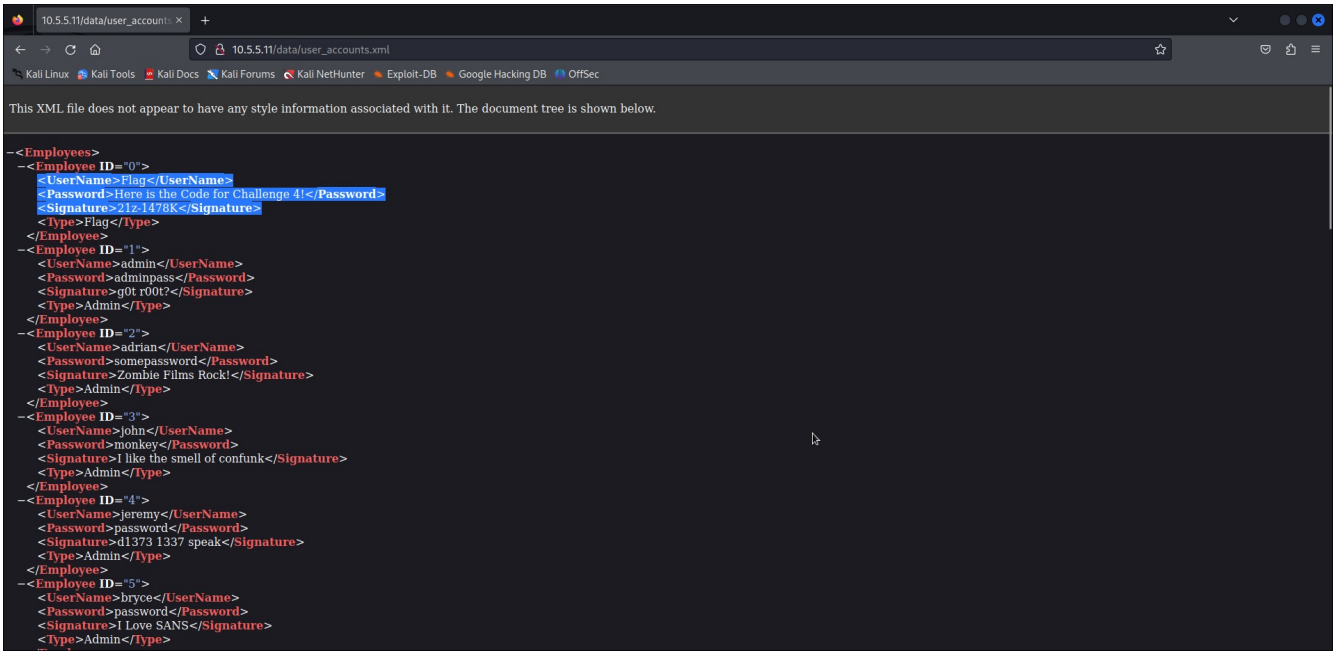


Step 3: Use a browser to access the recovered path.

Path: http://10.5.5.11/data



The directory listing contained **user_accounts.xml**, which held the required information.



What is the IP address of the target computer? **10.5.5.11**

What directories on the target are revealed in the PCAP? **Data**

What is the URL of the file? **http://10.5.5.11/data/user_accounts.xml**

What is the content of the file? **Employees info: ID, usernames, passwords signatures and types**
What is the code for Challenge 4? **21z-1478K**

Step 3: Research and propose remediation that would prevent file content from being transmitted in clear text.

- **Implement Transport Layer Security (TLS):** TLS uses a combination of asymmetric and symmetric cryptography to create a secure "tunnel." Before any data is sent, the client and server perform a "handshake" to verify identities and exchange session keys.
- **End-to-End File-Level Encryption:** The sender encrypts the file locally using a strong algorithm before it is ever uploaded or transmitted. The recipient must possess the corresponding decryption key to view the content.