Network Scanning and Sniffing

Nmap Full Scan on IP Address 8.8.8.8

Objective: To perform a full scan on the IP address 8.8.8.8 using Nmap to identify open ports, services running on those ports, their versions, and the operating system of the host.

STEPS:

STEP 1: Install Nmap

STEP 2: To perform the scan, open a terminal and run the following Nmap command nmap -p- -sV -O 8.8.8.8 where:

- -p-: This option scans all 65535 ports.
- -sV: This option attempts to detect the service versions running on open ports.
- -O: This option detects the operating system of the target machine.

STEP 3: Analyze Nmap Output

Example:

List of Open Ports:

- Port 53 (TCP) - DNS
- Port 80 (TCP) - HTTP
- Port 443 (TCP) - HTTPS

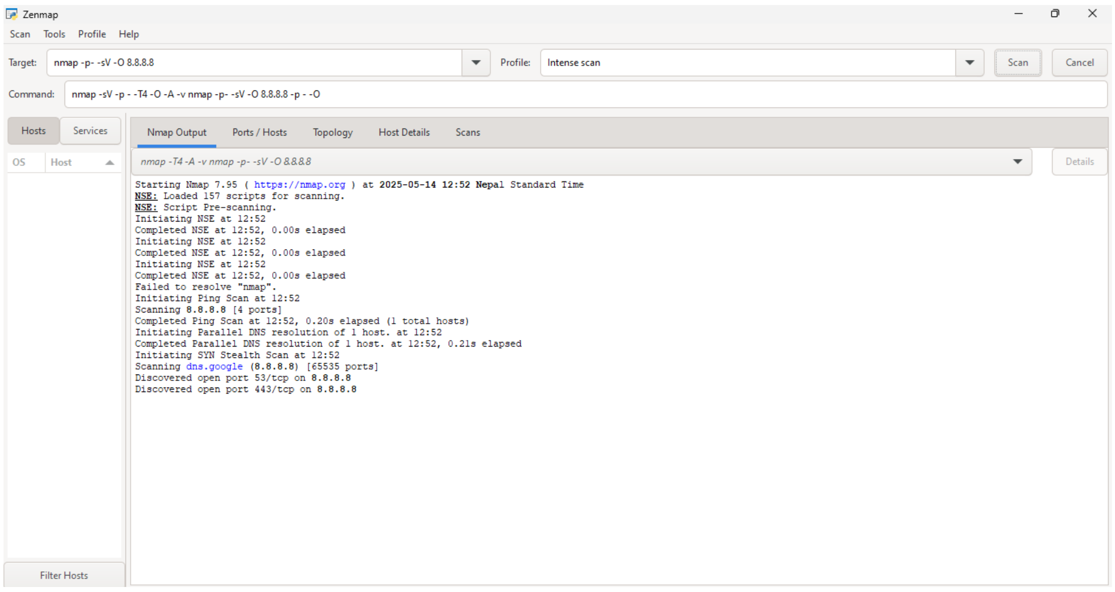
Services and Versions:

- Port 53 (TCP): dnsmasq version 2.80
- Port 80 (TCP): Apache httpd version 2.4.46
- Port 443 (TCP): Apache httpd version 2.4.46

Operating System (if detected):

- Microsoft Windows 7 or 8

Screenshot of Output:



Target: nmap -p- -sV -O 8.8.8.8

Profile: Intense scan

Command: nmap -sV -p - -T4 -O -A -v nmap -p- -sV -O 8.8.8.8 -p - -O

nmap -T4 -A -v nmap -p- -sV -O 8.8.8.8

```
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-14 12:52 Nepal Standard Time
NSE: Loaded 157 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 12:52
Completed NSE at 12:52, 0.00s elapsed
Initiating NSE at 12:52
Completed NSE at 12:52, 0.00s elapsed
Initiating NSE at 12:52
Completed NSE at 12:52, 0.00s elapsed
Failed to resolve "nmap".
Initiating Ping Scan at 12:52
Scanning 8.8.8.8 [4 ports]
Completed Ping Scan at 12:52, 0.20s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 12:52
Completed Parallel DNS resolution of 1 host. at 12:52, 0.21s elapsed
Initiating SYN Stealth Scan at 12:52
Scanning dns.google (8.8.8.8) [65535 ports]
Discovered open port 53/tcp on 8.8.8.8
Discovered open port 443/tcp on 8.8.8.8
```

Conclusion:

The Nmap scan successfully identified open ports, services with their respective versions, and the operating system details of the target IP 8.8.8.8. This information can be used for further network analysis or security auditing.

Capturing HTTP POST Traffic Using Wireshark

Objective: To capture network traffic for 5 minutes while browsing a login page using Wireshark, apply filters to extract HTTP POST requests, and identify any credentials or sensitive data transmitted in plaintext.

STEPS:

STEP 1: Install Wireshark

STEP 2: Start Wireshark and Select Interface

- Launch Wireshark.
- Select the appropriate network interface to start capturing.
- Click **Start Capturing Packets** (the blue shark fin icon).

STEP 3: Capture Traffic While Browsing a Login Page

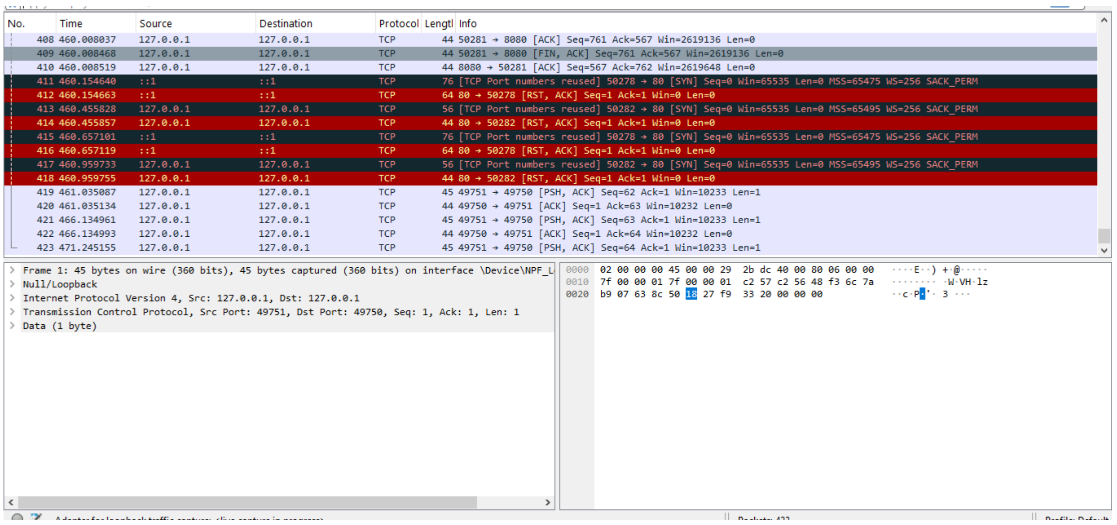
- Open your web browser
- Navigate to a website that uses HTTP and contains a login form
- Example: A test site like http://testphp.vulnweb.com/login.php

STEP 4: Apply Filters to View HTTP POST Requests

STEP 5: Inspect Packets for Sensitive Data

- Click on any POST request.
- Expand the Hypertext Transfer Protocol section.
- Check for **username**, **password**, or other form data in plaintext.

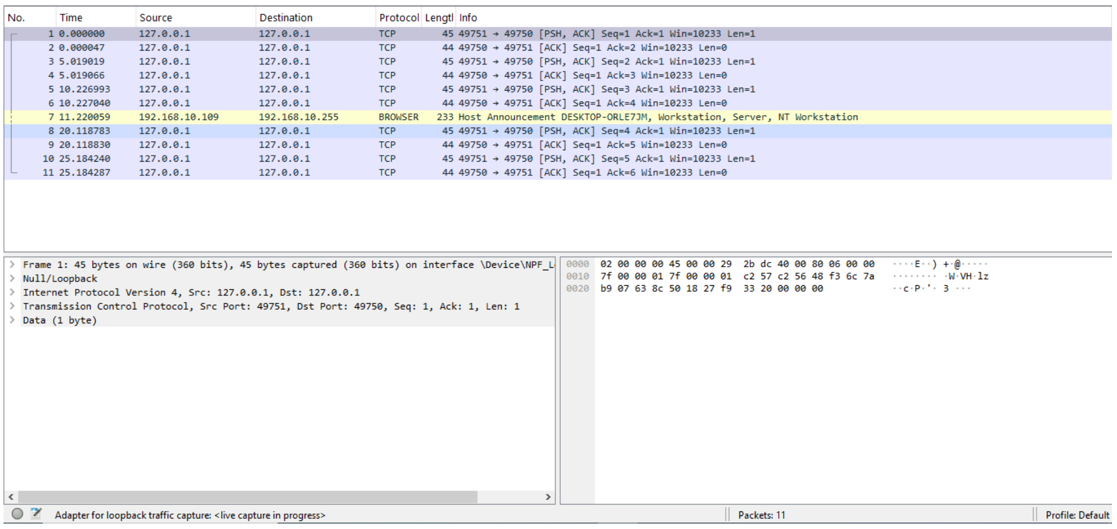
Screenshot of Output:



```
No.  Time        Source      Destination  Protocol Lengtl Info
 408 460.008037  127.0.0.1   127.0.0.1    TCP       44 50281 → 8080 [ACK] Seq=761 Ack=567 Win=2619136 Len=0
 409 460.008468  127.0.0.1   127.0.0.1    TCP       44 50281 → 8080 [FIN, ACK] Seq=761 Ack=567 Win=2619136 Len=0
 410 460.008519  127.0.0.1   127.0.0.1    TCP       44 8080 → 50281 [ACK] Seq=567 Ack=762 Win=2619648 Len=0
 411 460.154640  ::1         ::1          TCP       76 [TCP Port numbers reused] 50278 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 412 460.154663  ::1         ::1          TCP       64 80 → 50278 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 413 460.455828  127.0.0.1   127.0.0.1    TCP       56 [TCP Port numbers reused] 50282 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 414 460.455857  127.0.0.1   127.0.0.1    TCP       44 80 → 50282 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 415 460.657101  ::1         ::1          TCP       76 [TCP Port numbers reused] 50278 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 416 460.657119  ::1         ::1          TCP       64 80 → 50278 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 417 460.959733  127.0.0.1   127.0.0.1    TCP       56 [TCP Port numbers reused] 50282 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 418 460.959755  127.0.0.1   127.0.0.1    TCP       44 80 → 50282 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 419 461.035087  127.0.0.1   127.0.0.1    TCP       45 49751 → 49750 [PSH, ACK] Seq=62 Ack=1 Win=10233 Len=1
 420 461.035134  127.0.0.1   127.0.0.1    TCP       44 49750 → 49751 [ACK] Seq=1 Ack=63 Win=10232 Len=0
 421 466.134961  127.0.0.1   127.0.0.1    TCP       45 49751 → 49750 [PSH, ACK] Seq=63 Ack=1 Win=10233 Len=1
 422 466.134993  127.0.0.1   127.0.0.1    TCP       44 49750 → 49751 [ACK] Seq=1 Ack=64 Win=10232 Len=0
 423 471.245155  127.0.0.1   127.0.0.1    TCP       45 49751 → 49750 [PSH, ACK] Seq=64 Ack=1 Win=10233 Len=1
```

```
> Frame 1: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface \Device\NPF_L
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 49751, Dst Port: 49750, Seq: 1, Ack: 1, Len: 1
> Data (1 byte)
```

```
0000  02 00 00 00 45 00 00 29  2b dc 40 00 80 06 00 00   ····E··) +·@·····
0010  7f 00 00 01 7f 00 00 01  c2 57 c2 56 48 f3 6c 7a   ········ ·W·VH·lz
0020  b9 07 63 8c 50 18 27 f9  33 20 00 00 00           ··c·P·'· 3 ···
```



```
No.  Time        Source          Destination     Protocol Lengtl Info
   1 0.000000    127.0.0.1       127.0.0.1       TCP       45 49751 → 49750 [PSH, ACK] Seq=1 Ack=1 Win=10233 Len=1
   2 0.000047    127.0.0.1       127.0.0.1       TCP       44 49750 → 49751 [ACK] Seq=1 Ack=2 Win=10233 Len=0
   3 5.019019    127.0.0.1       127.0.0.1       TCP       45 49751 → 49750 [PSH, ACK] Seq=2 Ack=1 Win=10233 Len=1
   4 5.019066    127.0.0.1       127.0.0.1       TCP       44 49750 → 49751 [ACK] Seq=1 Ack=3 Win=10233 Len=0
   5 10.226993   127.0.0.1       127.0.0.1       TCP       45 49751 → 49750 [PSH, ACK] Seq=3 Ack=1 Win=10233 Len=1
   6 10.227040   127.0.0.1       127.0.0.1       TCP       44 49750 → 49751 [ACK] Seq=1 Ack=4 Win=10233 Len=0
   7 11.220059   192.168.10.109  192.168.10.255  BROWSER  233 Host Announcement DESKTOP-ORLE7JM, Workstation, Server, NT Workstation
   8 20.118783   127.0.0.1       127.0.0.1       TCP       45 49751 → 49750 [PSH, ACK] Seq=4 Ack=1 Win=10233 Len=1
   9 20.118830   127.0.0.1       127.0.0.1       TCP       44 49750 → 49751 [ACK] Seq=1 Ack=5 Win=10233 Len=0
  10 25.184240   127.0.0.1       127.0.0.1       TCP       45 49751 → 49750 [PSH, ACK] Seq=5 Ack=1 Win=10233 Len=1
  11 25.184287   127.0.0.1       127.0.0.1       TCP       44 49750 → 49751 [ACK] Seq=1 Ack=6 Win=10233 Len=0
```

```
> Frame 1: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface \Device\NPF_L
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 49751, Dst Port: 49750, Seq: 1, Ack: 1, Len: 1
> Data (1 byte)
```

```
0000  02 00 00 00 45 00 00 29  2b dc 40 00 80 06 00 00   ····E··) +·@·····
0010  7f 00 00 01 7f 00 00 01  c2 57 c2 56 48 f3 6c 7a   ········ ·W·VH·lz
0020  b9 07 63 8c 50 18 27 f9  33 20 00 00 00           ··c·P·'· 3 ···
```

Adapter for loopback traffic capture: <live capture in progress>   Packets: 11   Profile: Default

Conclusion:

Using Wireshark, we successfully captured HTTP POST requests during login. Sensitive credentials (username and password) were visible in plaintext, confirming the importance of using HTTPS for secure data transmission.

Web Application Vulnerabilities

SQL Injection on a Vulnerable Web Application

Objective: To test a vulnerable web application running at localhost:8080 for SQL Injection by attempting to bypass the login form using malicious SQL input.

STEPS:

STEP 1: Set Up the Vulnerable Web Application

STEP 2: Open the Login Page

- Navigate to: http://localhost:8080/login

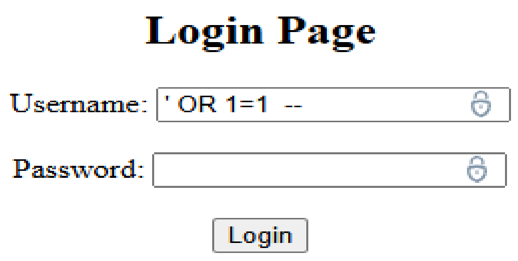
STEP 3: Use a Common SQL Injection Payload

- In the **Username** or **Email** field, enter one of the following SQL payloads:
  - ' OR '1'='1
  - admin' --

STEP 4: Submit the Form

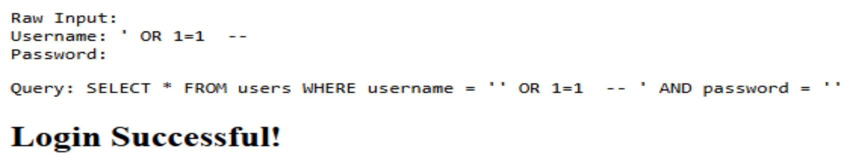
- Click the **Login** button.
- Observe the response
- If the SQL injection is successful, you will be **logged in** without valid credentials.

Screenshot of Output:


Login Page

Username: ' OR 1=1 --

Password:



```
Raw Input:
Username: ' OR 1=1  --
Password:

Query: SELECT * FROM users WHERE username = '' OR 1=1  -- ' AND password = ''
```

Login Successful!

Conclusion:

A successful SQL injection was performed on the login form of the web application running on localhost:8080. The vulnerability demonstrates the importance of using prepared statements and input sanitization in web applications.

Stored XSS Attack on a Web Application

Objective: To perform a **stored cross-site scripting (XSS)** attack on a vulnerable comment form of a test web application and demonstrate JavaScript execution when the comment is viewed.

STEPS:

STEP 1: Access the Vulnerable Web App

STEP 2: Craft a Malicious XSS Payload

- In the **comment** field, enter the following payload:
    - <script>alert('XSS Attack');</script>

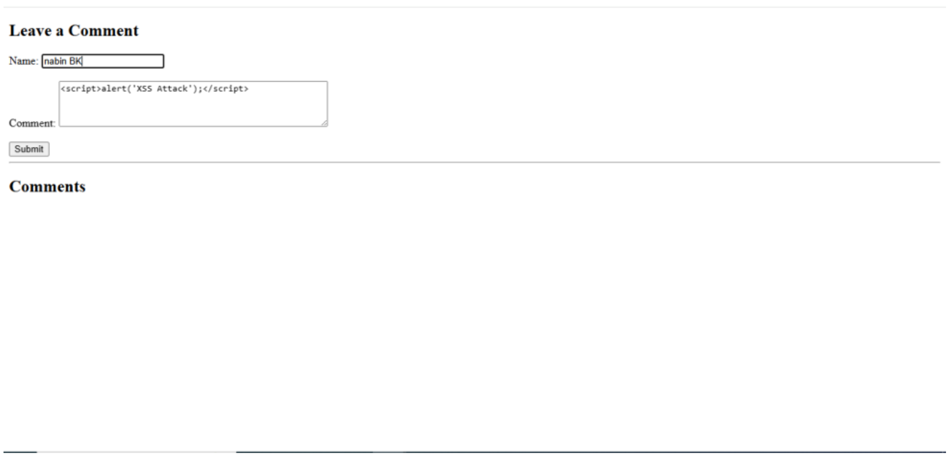
STEP 3: Submit the Comment

- Fill in the comment field with the payload above.
- Submit the form.

STEP 4: View the Stored Comment

- Navigate to the page where submitted comments are displayed.
- If the application is vulnerable to stored XSS:
    - A **JavaScript alert box** saying "XSS Attack" will appear when the page loads.

Screenshot of Output:

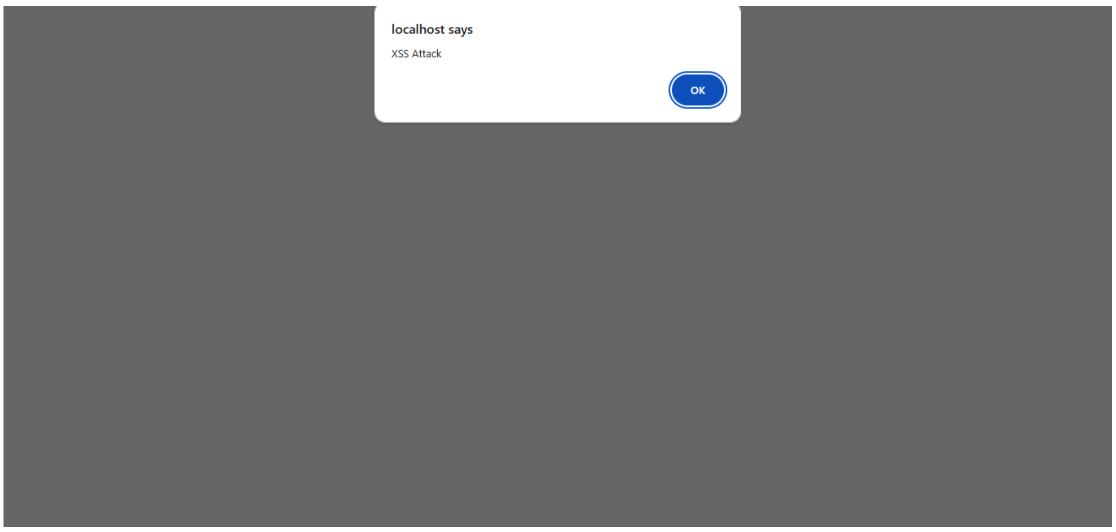


Leave a Comment

Name: nabin BK

Comment: <script>alert('XSS Attack');</script>

Comments


localhost says

XSS Attack

Conclusion:

The test web application was found to be vulnerable to **stored XSS**. A malicious JavaScript payload was stored in the database and executed when the comment was viewed, proving a real risk to user safety. This emphasizes the need for **input sanitization and output encoding** in user-submitted content.

Social Engineering Toolkit (SET) and Phishing

Create a Phishing Email with a Cloned Facebook Login Page

Objective:

To simulate a phishing attack using SET by cloning the Facebook login page, capturing dummy credentials, and understanding the risks associated with social engineering attacks.

STEPS:

STEP 1: Created a folder named facebook_clone.

STEP 2: Inside that folder, created a file index.html that looked like a Facebook login page with fields for email and password.

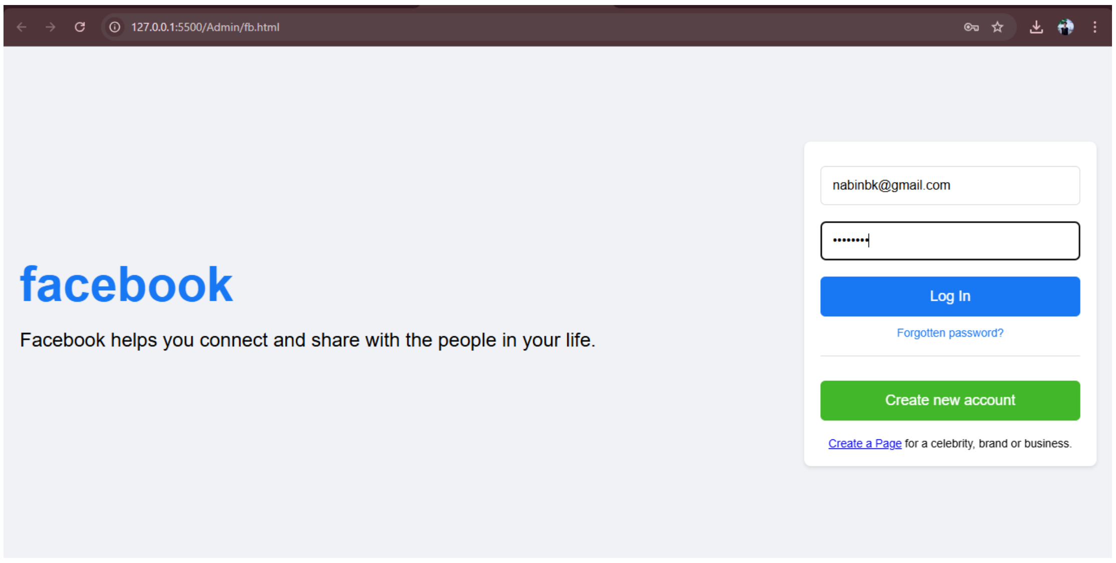
STEP 3: Created a file login.php to collect form data and save it to credentials.txt.

STEP 4: Linked the form in index.html to submit to login.php using the POST method.

STEP 5: Entered dummy credentials (e.g., testuser@example.com and dummy123) into the login form.

STEP 6: After submission, the data was saved in credentials.txt in the same folder.

Screenshot of Output:


127.0.0.1:5500/Admin/fb.html

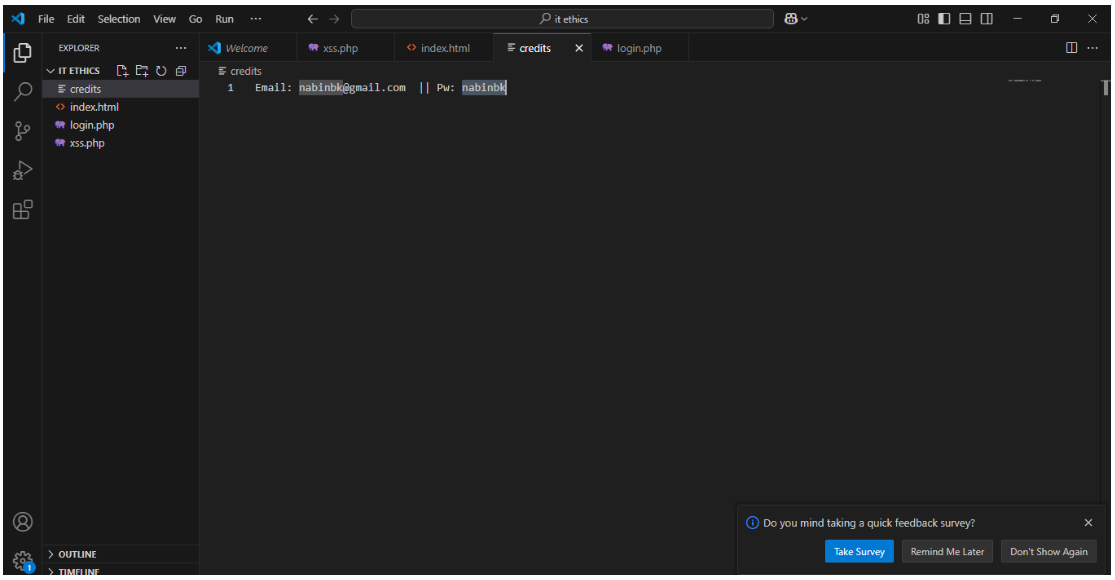
facebook

Facebook helps you connect and share with the people in your life.

nabinbk@gmail.com

••••••••

Create a Page for a celebrity, brand or business.



```
Email: nabinbk@gmail.com  || Pw: nabinbk
```

Conclusion:

In this lab, a simulated phishing page resembling Facebook was successfully created using basic HTML and PHP on a Windows system. The experiment demonstrated how user credentials entered into a fake login form can be captured and stored in a local text file. This highlights the simplicity and effectiveness of phishing attacks in deceiving users, reinforcing the importance of user awareness, strong security practices, and education to prevent credential theft.

Password Cracking

Password Cracking Lab Using John the Ripper

Objective:

To demonstrate how password hashes can be cracked using dictionary-based attacks with John the Ripper on a Windows system, using a provided list of hashed passwords and a common password wordlist.

STEPS:

STEP 1: Downloaded and extracted John the Ripper for Windows from the official OpenWall website.

STEP 2: Placed the `hashes.txt` file (containing hashed passwords) inside the `run` directory of John.

STEP 3: Copied the password wordlist (e.g., rockyou.txt) into the same directory.

STEP 4: Opened Command Prompt and navigated to the John directory.

STEP 5: Executed John with the wordlist to begin cracking:

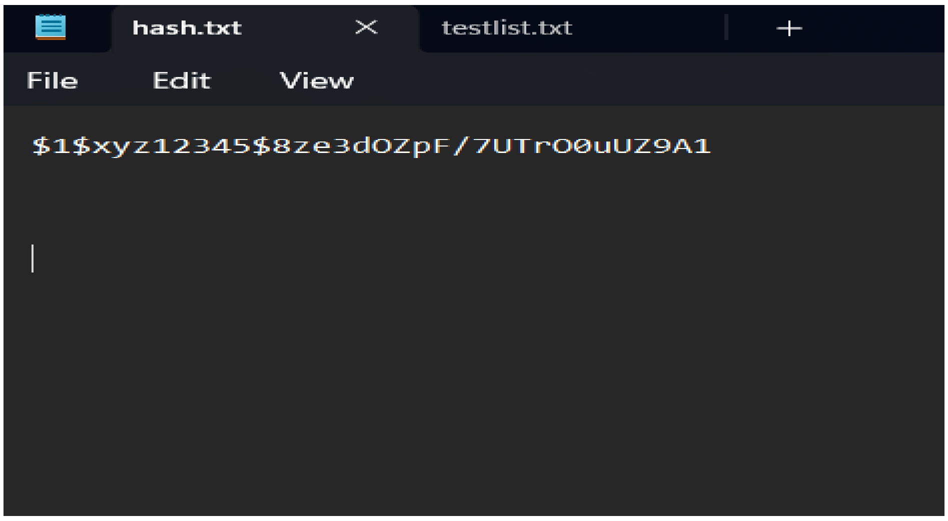
- john --wordlist=rockyou.txt hash.txt

STEP 6: Waited until John attempted all dictionary words against the hashes.

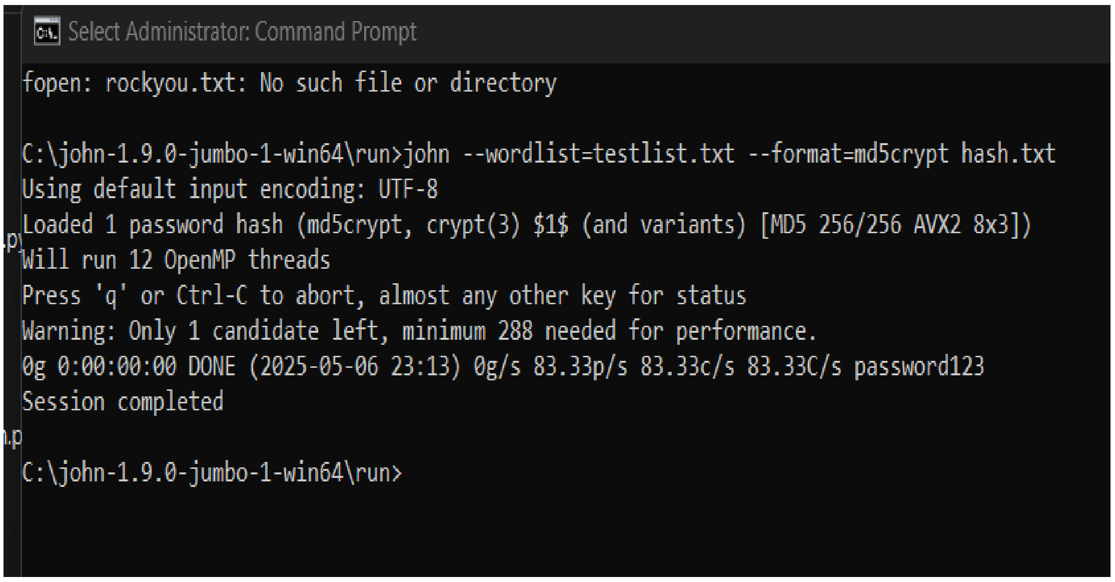
STEP 7: Displayed the cracked passwords using:

- john --show hash.txt

Screenshot of Output:



```
$1$xyz12345$8ze3dOZpF/7UTrO0uUZ9A1
```



```
fopen: rockyou.txt: No such file or directory

C:\john-1.9.0-jumbo-1-win64\run>john --wordlist=testlist.txt --format=md5crypt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8x3])
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 1 candidate left, minimum 288 needed for performance.
0g 0:00:00:00 DONE (2025-05-06 23:13) 0g/s 83.33p/s 83.33c/s 83.33C/s password123
Session completed

C:\john-1.9.0-jumbo-1-win64\run>
```

Conclusion:

The password hashes provided in the file were successfully cracked using **John the Ripper** on a Windows system with a wordlist-based attack. This lab highlights the vulnerability of using weak or common passwords, which can be easily exposed using publicly available tools. It reinforces the importance of selecting strong, complex passwords that resist dictionary-based and brute-force attacks.

Firewall Configuration and Analysis

Objective:

To configure a firewall using ufw (Uncomplicated Firewall) with the following security rules:

- Allow web traffic on HTTP (port 80) and HTTPS (port 443)
- Allow SSH (port 22) **only** from the local subnet
- Block all other incoming traffic

STEPS:

STEP 1: Opened "Windows Defender Firewall with Advanced Security" from the Start menu.

STEP 2: Created a new inbound rule to **allow HTTP (port 80)**:

- Rule Type: Port
- Protocol: TCP
- Specific Port: 80
- Action: Allow
- Profile: All
- Name: Allow HTTP

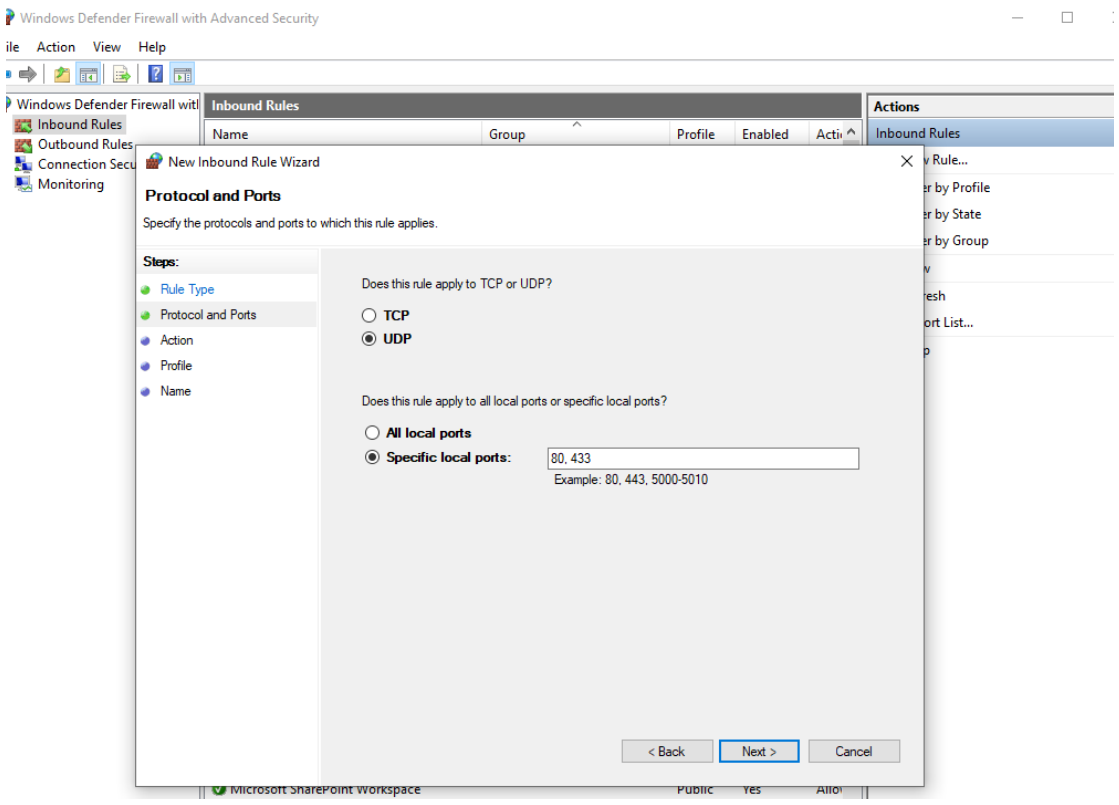
STEP 3: Created a new inbound rule to allow HTTPS (port 443)

STEP 4: Created a new inbound rule to allow SSH (port 22) only from local subnet

STEP 5: Modified firewall default behavior to block all other inbound connections

STEP 6: Left outbound connections allowed (default Windows setting)

Screenshot of Output:


New Inbound Rule Wizard

Protocol and Ports

Specify the protocols and ports to which this rule applies.

Does this rule apply to TCP or UDP?

TCP [✓ UDP]

Does this rule apply to all local ports or specific local ports?

All local ports [✓ Specific local ports:] 80, 433

Example: 80, 443, 5000-5010

Conclusion:

The firewall was successfully configured using the built-in Windows Defender Firewall with Advanced Security. HTTP and HTTPS ports were allowed for web access, while SSH access was restricted to the local subnet, and all other incoming traffic was blocked. This demonstrates a practical method of securing a Windows environment by explicitly controlling network traffic.

Firewall Log File Analysis (`/var/log/ufw.log`)

Objectives:

1. To analyze a firewall log file (`/var/log/ufw.log`) for suspicious network activity.
2. To identify potential threats, such as brute-force SSH attacks or insecure service scans (e.g., Telnet).
3. To recommend appropriate actions based on the analysis of the log data

STEPS:

STEP 1: Accessed the ufw.log File

STEP 2: Searched for Blocked Traffic

- The search term **UFW BLOCK** was used with Ctrl + F in VS Code to find all blocked connection attempts.
- This revealed blocked incoming and outgoing traffic, which can indicate unauthorized access attempts or malicious activity.

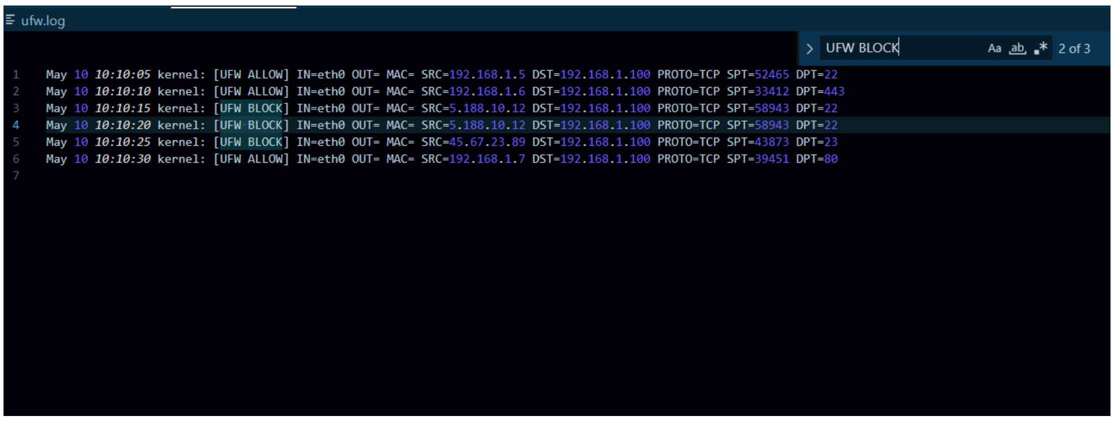
STEP 3: Analyzed Target Ports

- Additional searches for destination ports were conducted:
  - **DPT=22** → Identified SSH (port 22), commonly targeted for brute-force attacks.
  - **DPT=23** → Identified Telnet (port 23), which is outdated and insecure.

STEP 4: **Identified Suspicious Patterns**

- Suspicious Activity 1: Multiple blocked connection attempts from the IP 5.188.10.12 targeting port 22 (SSH), indicating a brute-force SSH attack.
- Suspicious Activity 2: A connection attempt from 45.67.23.89 targeting port 23 (Telnet), suggesting a Telnet service scan or attempt to exploit an outdated service.

Screenshot of Output:



ufw.log

UFW BLOCK 2 of 3

```
May 10 10:10:05 kernel: [UFW ALLOW] IN=eth0 OUT= MAC= SRC=192.168.1.5 DST=192.168.1.100 PROTO=TCP SPT=52465 DPT=22
May 10 10:10:10 kernel: [UFW ALLOW] IN=eth0 OUT= MAC= SRC=192.168.1.6 DST=192.168.1.100 PROTO=TCP SPT=33412 DPT=443
May 10 10:10:15 kernel: [UFW BLOCK] IN=eth0 OUT= MAC= SRC=5.188.10.12 DST=192.168.1.100 PROTO=TCP SPT=58943 DPT=22
May 10 10:10:20 kernel: [UFW BLOCK] IN=eth0 OUT= MAC= SRC=5.188.10.12 DST=192.168.1.100 PROTO=TCP SPT=58943 DPT=22
May 10 10:10:25 kernel: [UFW BLOCK] IN=eth0 OUT= MAC= SRC=45.67.23.89 DST=192.168.1.100 PROTO=TCP SPT=43873 DPT=23
May 10 10:10:30 kernel: [UFW ALLOW] IN=eth0 OUT= MAC= SRC=192.168.1.7 DST=192.168.1.100 PROTO=TCP SPT=39451 DPT=80
```

Conclusion:

In conclusion, the analysis of the ufw.log file successfully identified potential security risks in the form of a brute-force SSH attack and an attempt to access the insecure Telnet service. Based on the suspicious activities, appropriate actions such as blocking malicious IPs, disabling unnecessary services, and implementing tools like **fail2ban** were recommended to strengthen system security.

This process emphasized the importance of regularly monitoring firewall logs to detect and mitigate potential threats, ensuring a more secure and stable network environment.