WEB APPLICATION VULNERABILITY TESTING SYSTEM USING XSS_SQL_SCANNING ALGORITHM

THINZAR AUNG

M.C.Sc. JUNE 2022

WEB APPLICATION VULNERABILITY TESTING SYSTEM USING XSS_SQL_SCANNING ALGORITHM

BY

Thinzar Aung
B.C.Sc.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science
(M.C.Sc.)

University of Computer Studies, Yangon
JUNE 2022

ACKNOWLEDGEMENTS

First and foremost, I would like to express all my teachers who advised, taught and helped me to complete my thesis successfully during the study at the University of Computer Studies, Yangon.

Secondly, I would like to special thanks to my principal **Dr. Mie Mie Khin**, the Rector of the University of Computer Studies, Yangon who gave me the opportunity to develop this thesis for her general guidance during the period of study.

And I would also like to thanks to **Dr. Mie Mie Thet Thwin**, the former Rector of the University of Computer Studies, Yangon who gave me the opportunity to develop this thesis for her general guidance during the period of study.

I would like to express my respectful gratitude to **Dr. Si Si Mar Win** and **Dr. Tin Zar Thaw**, Deans of the Master 24th batch, University of Computer Studies, Yangon, for her excellent guidance.

I would like to thank and my sincere gratitude to former Course Coordinator, **Dr. Mie Mie Su Thwin** who gave me the opportunity to do this thesis on the topic, “Web Application Vulnerability Testing System Using XSS_SQL_Scanning Algorithm”, and also gave invaluable recommendations regarding to this thesis.

My sincere thanks and respect go to my supervisor, **Dr. Zin Thu Thu Myint**, Associate Professor of Faculty of Information Science, University of Computer Studies, Yangon, for her invaluable advice, giving me detailed guidance, support and comments throughout the thesis process.

In addition, I would like to thank all of my thesis’s board examiners who gave the precious comments and corrections to my work for getting good end result. I would like to thank **Daw Hnin Yee Aung**, Lecturer, Department of English, University of Computer Studies, Yangon, for her valuable supports and editing my thesis from the language point of view.

Last but not at least, I am extremely thankful to my parents and my family for supporting, inspiring and encouragement to me from the childhood to the present time. Finally, I am extremely grateful to my all of teachers, my colleagues and all of my friends for their invaluable and precious help and general guidance.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

------------------------------------	---------------------------------
Date	Thinzar Aung

ABSTRACT

Nowadays, many people use the internet for more than one purpose. Among these purposes, they mostly apply the web application which is one of the internet usage technologies. A web application is composed of a web server and web browser in other terms client-side and server-side. Web applications are typically developed with a limitation of time and usually, application developers make mistakes in the code which can cause application vulnerabilities. If the vulnerability appears, some of the irresponsible people who are attackers will exploit web applications through vulnerability to obtain some privileges in the system. Due to the widespread use of web applications, it is essential to discover vulnerabilities to avoid the exploitation of web applications. Various well-known scanners are available for detecting vulnerabilities. In this thesis, the proposed system can also find out vulnerability as almost as these scanners. The proposed system presented in this thesis can find the two types of vulnerability, Structured Query Language (SQL) injection and Cross-site scripting (XSS) attacks that are a huge risk for victim businesses and they mostly occur in the web application. Besides, the proposed system applies the Naïve pattern matching algorithm even though other several methods completed in the string searching process, because they are still having complexities in constructing the preprocessing phase. Moreover, the response message returned by the proposed system is too short enough to match by this pattern matching algorithm approach. Finally, the proposed system is being used by the well-known scanner and is evaluated how accurate the results based on having false negative and false positive rate.

Keywords: XSS_SQL_Scanning Algorithm, Naïve Pattern Matching Algorithm, Cross-site Scripting (XSS), Structured Query Language (SQL) Injection, Web application

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

		Page
SQL	Structured Query Language	iii
XSS	Cross-site Scripting Language	xi
CIA	Confidentiality, Integrity, and Availability	5
HTML	Hyper Text Markup Language	6
CSS	Cascading Style Sheets	6
CSRF	Cross-Site Request Forgery	7
URL	Uniform Resource Locator	8
DOS	Denial of Service	10
HTTP	Hypertext Transfer Protocol	11
OWASP	Open Web Application Security Project	16
GUI	Graphical User Interface	17

CHAPTER 1
INTRODUCTION

As a trend of technology, many people are mostly using web applications for developing their businesses, education, communications to access anywhere, anytime, and thus it saves time and effort. This is why web applications become an important role in lives. In the act of using web applications, people give personal information to the organization, and then store sensitive information on them. On the other hand, some attackers who are unethical and selfish exploit the web application to gain unauthorized access and do other things such as identity theft, privacy violation, and other cyber-attacks. These illegal points allow the attackers to make whatever they want through the weaknesses of the web application.

Vulnerability is the weak point of the web application caused by unawareness of the developers who cannot be handled validation the user inputs, appropriate validation methods, and so on. Because of those facts, detection of vulnerability is needed more. There are so many different kinds of vulnerabilities but, it is indicated to OWASP in 2019 that SQL injection and XSS attacks reach number one and seven vulnerabilities found in web application [15].SQL injection is a type of database-control attack where the attackers inject SQL command into the database and then manipulate the websites in order to manage the database and then they try to store, modify, retrieve, and steal user's data whereas, XSS attack is the opposite of SQL injection that uses JavaScript code to obtain the information and its target is to control the client-side and also called as a client-side attack.

At present, people use various types of vulnerability detection tools for web application security that are either free or paid versions, in addition, some of them face a high rate of a false negative and false positive. A false positive means when detecting results show that the web application has vulnerability, however such vulnerability actually does not exist whereas a false negative means in contrast to a false positive where the outcome results show no vulnerability but in reality, it exists.

Thus, the XSS_SQL_Scanning algorithm is proposed to assist in vulnerability detection with a low rate of false negative and positive. Moreover, a new idea in terms of applying the proposed system about using the Naïve pattern matching algorithm because it supports this proposed algorithm to be the simpler, no preprocessed state and reliable in searching state. Normally, other pattern matching algorithms are good at finding patterns comparing with predefined ones though they have a preprocessing phase therefore they need more storage space and time. In summary, this proposed method protects the web application from the strike of attacks by discovering the vulnerability. The following section discusses some related works of this thesis.

1.1 Related Work

The system in this thesis intends to support the admins who require to obtain secure web applications without vulnerabilities and to prevent any attacks from the attacker. They usually use secure coding guidelines and also detect vulnerabilities. In this section, we discuss the previous studies of detecting web application vulnerabilities concerning with SQL injection and XSS attacks as well as one of the web application vulnerability scanners.

The first study is a SQL injection scanner using the Boyer-Moore string matching algorithm is implemented by Teh Faradilla Abdul Rahman et al [12]. Their experiment was the technique of using pattern matching that could detect vulnerability with high accuracy and also avoid exploiting the vulnerability. But despite that, they required extra steps and storage in case of searching the pattern string due to the use of the Boyer-Moore algorithm.

In the other study, Priti Singh et al. implemented a detection method for SQL injection and Cross-site scripting attacks [7]. The approach method was good for detecting SQL injection vulnerability but had some weaknesses in detecting XSS attacks because of using only one payload. When detecting a web application with a payload, if this payload was restricted or encoded, it could not try with another payload; in this case it could not find any vulnerability although it should. This system will be perfect if many payloads were used for detection.

Simon Wahstrom admitted a paper about the evaluation of string searching algorithms [10]. In this paper, he described the weaknesses and strengths of the most common algorithms. And he also presented a naïve algorithm that was probably used when we expected to find a relatively short pattern.

1.2 Objectives of the Thesis

The main objectives of the thesis are as follows:

- to discover the possibility of web application vulnerabilities.
- to assist web developers in detecting their web applications that have weaknesses or not.
- to provide the basic information on how to detect SQL injection and XSS attacks for some people who begin to study the web application security.
- to support the security of the web application.

1.3 Organization of the Thesis

This thesis consists of five chapters.

Chapter 1 is the introductory section where the introduction to web application vulnerability. And the related works, the objectives, and the organization of the thesis are presented.

Chapter 2 describes the background theory related to this thesis such as security vulnerabilities, web application vulnerabilities, and pattern matching algorithm that are described in details.

Chapter 3 presents the design of the proposed system describing system flow, the detail explanation with algorithms and the evaluation of the output resulting from the post detection.

Chapter 4 describes the implementation of the proposed system in detail and the experimental result.

Finally, Chapter 5 concludes this thesis that includes the benefits, limitation and further extension of the proposed system.

CHAPTER 2
BACKGROUND THEORY

The related background theory with the research work is presented in this chapter. In the first section, the security vulnerability and the types of security vulnerability are described. The next section describes about the web application, web application vulnerability, types and detection of web application vulnerability. In the last section, pattern matching algorithm and the most common pattern matching algorithm are presented in details.

2.1 Security Vulnerability

The security vulnerability is the attackers can get an unauthorized access to the system and compromises the confidentiality, integrity, and availability of the system. These are the fundamental requirements for every software. It reduces the guarantee of the system's information**.**

- **Confidentiality:** It means to protect data or information from unauthorized access and misuse by setting rules.
- **Integrity:** It means that the data or information is ensured reliable, correct, and authentic.
- **Availability:** Availability is ensuring that only authorized people can reliably access the information.

The computer system has numerous types of vulnerabilities. Some of the common vulnerabilities to the damaging system are:

- **Application Vulnerabilities:** This vulnerability can occur from coding errors or security flaws.
- **Network Vulnerabilities:** The unsecure network hardware or software such as firewalls and wireless access points that can cause a risk in the network.
- **Operating System Vulnerabilities:** If an operating system such as the windows operating system or Linux operating system has program errors, this operating system will cause vulnerability. The attackers can

hide or install backdoor programs within the operating system that can exploit to gain unauthorized access or to make damage.

- **Software Vulnerabilities:** A software vulnerability is a flaw in software that can enable attackers to take advantage of them. The web application is also a type of software that uses the internet for browsing.

2.2 Security Vulnerability Testing

Security testing is a kind of software testing that uncovers the flaw of the software system. It helps to identify the vulnerabilities and threats in the system. The vulnerability and threat can cause damage and disruption to the organization’s system [13]. For example, if the student management system has insufficient safety, the attackers can update the exam information. It also detects all of the possible risks and controls the CIA [16]. So, security testing can prevent malicious attacks from unauthorized persons. It assists the solution of program coding to the application developers.

2.2.1 Types of Security Testing

Security testing plays an essential role in producing successful and quality software. By security testing, this application is more trustworthy and safer than other no detected software. It detects the threats and quantifies all potential vulnerabilities in the system. It identifies security risks in the system and solves the problems via coding errors. Security testing methodology has seven kinds of tests. They are described as follows:

- **Vulnerability Scanning:** This can be executed on the automated vulnerability scanning software that has the acknowledgment of vulnerability. It scans a system to figure out any potential system flaws.
- **Penetration Testing:** This testing simulates an attack from a malicious hacker. This includes analyzing the system to predict and detect potential vulnerabilities.
- **Risk Assessment:** The goal of the risk assessment process is to reduce the risk and control the threat. After analyzing the risks, the precautions are based on the risk. These risks are classified as three levels according to their severity. They are low, medium, and high levels.

- **Security Scanning:** This type of scanning involves manual and automatic scans in the application and networks. It includes identifying the weakness and risks and provides solutions to reduce these flaws.
- **Ethical hacking:** Ethical hacking is hacking of an organization's software programs. The purpose is to penetrate the application with authorization for searching security vulnerabilities. The benefit of this hacking is to protect the system from malicious hacker attacks.
- **Security Auditing:** Security auditing can also call security reviewing. It is a method of auditing any suspicious features by reviewing each line of code. In this, the security audit does the internal analysis of the application and manages the system for possible security flaws.
- **Posture Assessment:** Posture Assessment is the integration of security scanning, risk assessment, and ethical hacking. This assessment determines the entire security position and gives the security situation of the organization.

In this thesis, application vulnerabilities in web application detects with the security testing methods of vulnerability scanning.

2.3 Web Application

A web application is defined as a program of the computer that composes a web browser (client-side) and web server (server-side) to perform tasks over a network. The users can use web applications through many kinds of the web browsers such as Google Chrome, Microsoft Edge, Mozilla Firefox, Internet Explorer, Safari, and Opera. The client-side programming language typically utilizes JavaScript, HTML, and CSS which help build a front-end application. This allows to present information to users and interact with the web server. Python, Ruby, PHP, ASP, and Java are commonly used in the server-side programming language.

The web server needs to operate a web application and handle the retrieval and storage of information. When a client sends HTTP requests through a web browser interface, HTTP is generally used to send GET or POST method to the web server. The web server executes the request and sends the requested response to the web browser and the browser accepts the response and then displays the requested web page and content. Web applications are commonly used in shopping cart, e-commerce shops, information sharing, content management systems, and webmail.

2.4 Web Application Vulnerability

Web application vulnerability occurs when the web-based application involves a weakness or system flaw in this application. The website visitors submit and retrieve data to and from a database over the internet with the web application by using the web browser. Most importantly, many of these databases consist of valuable information such as sensitive customer data, financial details, and personal details, which are profitable targets for an attacker. The attackers have numerous methods to steal these important data if they have serious weaknesses or vulnerabilities. As the application developer does not sanitize the input form, misconfigures the web servers, and has a design flaw in the web application, so the use of the web application will lead to vulnerabilities. This vulnerability can harm related users to use the web application. The attackers take advantages of these vulnerabilities to harm people. For example, if the attackers gain privileges on a website, they may be able to upload sensitive files and also control that website and then get public and direct access to the databases.

There are many types of vulnerabilities in web applications [14]. The most common types of vulnerabilities are SQL injection, XSS, Broken Authentication and Session Management, Security Misconfiguration, Insecure Direct Object References, CSRF, Insecure Cryptographic Storage, Failure to restrict URL Access, Insufficient Transport Layer Protection and Unvalidated Redirects and Forwards.

- **SQL injection:** SQL injection is directed access to the database and the root system. It occurs when pre-compared SQL command with the user input or query is sent into the database by the attacker. Many web servers stored most of the critical data such as personal credentials (username and password) and other personal information. The attacker controls and manages data in these databases if the injection is successful to the vulnerable websites.
- **XSS:** Unlike SQL injection, an attacker visits a vulnerable website to target the user's browsers. One of the similarities to SQL injection attacks is that XSS uses malicious code to inject websites. This code will be injected via the input field and automatically run in the injected

vulnerable web page. The credentials data can be hijacked when the website visitors view the infected web page without these users realizing it.

- **Broken Authentication and Session Management:** When the users visit the website, the username and password are invalidated as a session cookie and session ID that are created by the website. This cookie and session ID should be done or should be invalided to logout or browser closed otherwise the sensitive data will remain in the system that is exposed to an attacker. When the attacker uses the same computer, the credential data is compromised. Checking the website should include proper configuration, the strength of the authentication process, and session management.
- **Security Misconfiguration:** Security misconfiguration can be defined as the misconfiguration of the application server, database server, and web server. The attacker deploys the developer is not keeping the latest software that will be occurred the vulnerability. For example, if the database version is not up to date, sensitive information appears in the error message. They can enumerate the information of the application server, database, and platform.
- **Insecure Direct Object References:** Insecure direct object occurs the developers use reference objects in URL. The attacker can change the value in reference objects and can view other information and then can do the directory traversal attack. For example, the URL of https://www.example.com/example.php?id=1 is not secure in id parameter. The attacker can change the id value from 1 to 2 and then can view the information of the value as 2. In addition, the attacker can gain the sensitive file like https://www.example.com/example.php?../../../etc/passwd.
- **CSRF:** Cross-Site Request Forgery is a malicious HTTP request that comes from the attacker to the victim’s browser. This request persuades as a request arrives from a trusted site. If the user clicks the malicious request who logged into the original website, then sensitive data will be

stolen and will be performed unauthenticated behavior on the user's behalf.

- **Insecure Cryptographic Storage:** Insecure cryptographic storage is focused on application databases that are not stored securely. The developer uses an encryption algorithm inappropriately in the database. This often occurs when the database is compromised by the attacker with SQL injection. The attacker retrieves data in the database such as personal information. If this credentials data is not stored properly by using hashing or encryption algorithms, they will be easily controlled and accessed by the attackers.
- **Failure to restrict URL Access:** Web applications are not restricting and protecting privileged URL access called failure to restrict URL access. So, the attacker can use the brute force technique to access the URL of an unprotected page. And the attacker can also guess this unprotectable URL for direct access to the source file without using the web application. As a result, attackers can access and view sensitive pages, confidential information, and privileged page.
- **Insufficient Transport Layer Protection:** This vulnerability occurs deals with data exchange over the network. The user sent data to the server without passing strong algorithms, valid or unexpired certificates or SSL can allow data compromising. And the attacker can sniff network traffic that may cause a Man-in-the-Middle attack.
- **Unvalidated Redirects and Forwards:** Web developers frequently develop web applications by using redirect and forward methods for the intended purpose. But sometimes, that occurs with unvalidated redirecting and forwarding in those web applications. They can harm the reputation of websites because they redirect the users to malware sites or phishing and forward unauthorized parts of these sites.

Among these common vulnerabilities, the most common vulnerabilities that occur in the web application are SQL and XSS. In this thesis, the most exploited web application is emphasized the vulnerabilities of SQL injection and XSS.

2.4.1 SQL Injection Attack

This attack is called a server-side code injection attack based on SQL that exploits the system through the weakness of web application with the predefined SQL command and its query is inserted into the URL or input fields of the web application. The web application sends this query to the database and executes this query and then sends data back to the web application. In this way, the attackers take the privileges of the database through SQL injection [11].

The attackers can take sensitive data such as username and password from the database, modify this sensitive information, lack database control, and cause a DOS attack. It mostly occurs when the developers properly invalidated the user input as a part of the SQL query. SQL injection can be executed in several methods to take many advantages from the organization's network. The three major types of SQL injection are as follows.

- In-band SQL injection
- Inferential SQL injection
- Out-of-band SQL injection

2.4.1.1 In-band SQL Injection

In-band SQL injection is one of the most common injections of the web applications. It occurs when the attacks launch in the same communication channel and then collect results in this channel. In-band SQL injection is the easiest method to exploit among injections. There are two types of In-band SQL injection.

- Union-Based SQL Injection
- Error-Based SQL Injection

2.4.1.1.1 Union-Based SQL Injection

In this case, when a web application's response contains the error message with SQL query, the attacker uses the **UNION** SQL operator for exploiting the database to obtain the data of tables. This is called a Union-based attack. By using the **UNION** keyword, the attacker joins the original query and another additional SQL query. For Example:

SELECT column1, column2 **FROM** table1 **UNION SELECT** column3, colunm4 **FROM** table2

When this **UNION** query executes, the query will return the result of the original query and another result of the concatenating query.

2.4.1.1.2 Error-Based SQL Injection

Error-based SQL injection is a kind of in-band SQL injection technique when an attacker inserts the malicious query into the database and gains error messages which contain the information of the database. It was shown that the “supplied argument is not a valid MySQL result resource” as an example of an error message. So, the attacker knows the type of database and the error of syntax and then they triggered the web application by an SQL injection attack.

2.4.1.2 Inferential SQL Injection

In inferential SQL injection, any HTTP response like database error or the result of SQL query does not return from the database. Unlike UNION attacks, they are not relied on the application’s responses due to the injected query. In this situation, there are two ways for testing the application with SQL injection attack.

- Content-Based Blind SQL injection
- Time-Based Blind SQL Injection

2.4.1.2.1 Content-Based Blind SQL Injection

The Content-based blind SQL injection is also called Boolean-based blind SQL injection. In this injection, the attacker sends a couple of queries to the databases which make the application return the result based on whether TRUE or FALSE result. If the condition of the injected AND 1=1-- query is true, the response of the application will normally display. However, the other injected AND 1=2-- the condition is false, and the web application does not return any results. Depending on these two results, the attacker is able to exploit this application through content-based blind SQL injection.

2.4.1.2.2 Time-Based Blind SQL Injection

When the injected SQL query is executed, the application’s response is not any different from the original page. The conditional errors of the previous method do not work in this state. So, one of the next exploitable techniques in blind SQL injection is Time Based on blind SQL injection. This type of injection relies on the time delay of the response time that occurs the attacker sends the SQL query to the database. Successful SQL injection indicates how long the database waits for a given time before responding.

For instance,

; WAITFOR DELAY '00:00:10'- -

This example value will cause the query to wait 10 seconds before displaying the HTTP response. Using this technique, the attacker can retrieve unauthorized data.

2.4.1.3 Out-of-band SQL Injection

Out-of-band SQL injection is the least common injection, and that occurs when an attacker uses the different channel to either launch an attack or gather information. This type of injection relies on the features of the database server which uses in web applications.

2.4.1.4 Prevention of SQL Injection

Ensuring developers develop their web applications with SQL injection mitigation strategies. The common ways to reduce SQL vulnerability are input validation and sanitization, escaping, parameterization, and stored procedure.

2.4.1.4.1 Input Validation and Sanitization

Input validation is the most commonly used to prevent SQL injection attacks. This process needs to determine the required SQL query whether the user input is validated or not. It filters the unvalidated SQL statements and creates whitelisting to validate the user-controlled data.

The other mitigating component against SQL injection attack is data sanitization. The proper function and regular expression are used to configure the input from users that ensure any damaging characters such as double quote and single quote is not passed to the database.

2.4.1.4.2 Escaping

Escaping is a function that escapes special characters from the user input through the URL or POST data. This escaped data is passed through the database that doesn't confuse the DBMS's character functions. For example, JSON's special characters are quotation mark ("), reverse solidus (\), solidus (/), backspace, form feed, new line, carriage return, and horizontal tab. These characters are escaped through the STRING_ESCAPE () function. The escape function returns the string with the encoded sequence.

2.4.1.4.3 Parameterization

The developers usually use parameterized queries for distinguishing the user input and code in the database. This is also known as variable binding. Defining parameterized queries in all database queries are executed on the providing parameters. For example, the statement likes that

'**SELECT** * **FROM** UserTable **WHERE** User=? **AND** Pass=?'

parameters.add ("User", username)

parameters.add ("Pass", password)

where username and password are related to the entered username and password.

2.4.1.4.4 Stored Procedure

The prepared group of SQL statements are called stored procedure, which are stored in the database management system, so it can be called from the website and used repeatedly. The procedure performs the input value of the parameters and returns the output values of multiple parameters. So, this stored procedure can call to execute when the parameter values are passed. The example of the stored procedure syntax and the executable syntax of this stored procedure is shown in the following. The sample stored procedure is with one parameter.

- **CREATE PROCEDURE** studentName @name nvarchar(40)
  (*procedure name with one pareameter*)
  AS
  SELECT * **FROM** Student **WHERE** Name = @name
  (*SQL statement*)
  GO;
- **EXEC** studentName @name = ‘Mg Hla’; (procedure call)

2.4.1.4.5 Active Updating Software

SQL injection regularly exploits outdated web applications. That is why the application developers no longer support the application and ignore updated software. Outdated software becomes system failure which can cause data loss, leaking of sensitive information, poor software performance, bug disruption, and incompatibility. Prevention is the best way before happening the system failure. Therefore, the developers need to update all of the software components such as database server, plug-ins, libraries, and frameworks.

2.4.2 Cross-site Scripting (XSS) Attack

XSS is different from SQL injection because it is a client-side code injection attack rather than server-side and is generally written with JavaScript code and sometimes with HTML code for injection. This attack occurs when the attacker inserts the malicious code into the vulnerable website via the input field or the URL, the victim visits that website, and then the malicious script automatically executes within the web application and infects the victim's web browser [8]. For example, the attacker embeds malicious JavaScript in the image post.

Cross-site scripting is mostly used to modify the content of the website, steal user's cookies or session information, redirect the website that is created by the attacker, and carry out the abnormal behavior in the vulnerable site. This attack can happen if the web application does not validate the user input requesting from the untrusted source and does not sanitize the script in the HTTP response. There are three different types of XSS attacks.

- Reflected XSS
- Stored XSS
- DOM-based XSS

2.4.2.1 Reflected XSS

In reflected XSS, the attacker generally uses phishing techniques to perform this attack. This attack is different from stored XSS because it can only act on the client-side and, not the server. The attackers constructed the script code as a link in the web application from this web application vulnerability. If the user accesses this link, then the malicious script will execute in the user's browser and the browser transmits the victim's information to the attacker.

The reflected XSS mainly occurs in the web application rather than stored XSS even if the reflected XSS is less vulnerable attack. It only needs the user's action to click the malicious link. Furthermore, the attacker gains the user's private information by creating a fake login page.

2.4.2.2 Stored XSS

Stored XSS is also known as persistent XSS. This attack happens when the attacker stores the injected code into the persistent storage known as a database. In this attack, the attacker sends the malicious script code along with the comment form of the blog page and then saved it into the database. After that, the attacker waits for the user to visit this page. This attack occurs when the user logged into this page, the script code automatically executes and then the attacker collects the user credential such as users' cookie.

This is the most harmful attack because any user navigates the injected web page at any time and then the effect of the injected code takes place as a part of the web page. The attacker uses this script code for stealing the user cookie. For example,

<script>
window.location=http://www.example.com?id=+**document.cook**;
</script>

2.4.2.3 DOM-based XSS

DOM-based XSS means Document Object Model-based Cross-site Scripting and is also called type 0 XSS. Unlike stored XSS and reflected XSS, DOM-based XSS is possible for the attackers to write malicious code in the document object model of the HTML page and that occurred in the HTTP response page.

DOM-based XSS is mostly manipulated from the vulnerable page of the user's browser address bar wherein the malicious payload is executed as a modification of the document object model. This vulnerability can cause when the programmers are not sanitization in dynamic code execution.

2.5 Detection of Web Application Vulnerability

The developers require to protect their web applications from attackers. According to OWASP, the most effective method of detecting vulnerability in the web application is reviewing code manually. This technique does not apply automation tools and detects the software to discover any unusual behavior in the software. This manual testing takes a lot of time and requires specialized skills.

On the other hand, security experts develop automated approaches for detecting vulnerabilities. The automated testing performs automation testing using the prepared scripts and automated tools. These tools quickly and repeatedly run these scripts for testing. This testing is faster than manual testing and does not require human intervention.

The numerous types of web application scanners provide the application developer to test their web application vulnerabilities automated. These web application scanners mostly detect the application by inserting the malicious input through the crawling page of these applications and evaluating these responses.

The most popular web application security scanners are Netsparker, Acunetix, Sucuri Sitecheck, W3af, Rapid7 InsightAppSec, Qualsys SSL Server Test, Mozilla Obervatory, Burp Suite, HCL App Scan, Qualsys Web Application Scanner, and Tenable. They are open sources and commercial scanners. Here are some of the most popular security scanning tools that can be used to discover vulnerabilities in web applications.

2.5.1 Netsparker

Netsparker is a web application security scanner that can accurately scan many types of vulnerabilities. It accurately detects vulnerabilities because it uses proof-based scanning technology. It helps in exploiting vulnerabilities that inject automatically, quickly, and safely.

2.5.2 Acunetix

Acunetix is an effective web application security scanner that features advanced macro recording to scan complicated web applications and make certain to report a detected vulnerability. This function saves time on having false positives[2].

2.5.3 Burp Suite

Burp Suite is a fully automated web application security scanner that allows scheduling and prioritizing scans. It means that Burp Suite schedules scanning time and prioritizes threat levels when detecting vulnerabilities. It is correct and quick because it integrates CI/CD tracking systems. CI means continuous integration and CD refers to continuous delivery.

2.5.4 W3af

W3af stands for web application attack and audit framework. It uses the black box testing technique to detect the web application. It can identify over 200 vulnerabilities of the web application. It has both GUI and a command-line interface. So, the users can be easily used the interface.

2.5.5 Sucuri Sitecheck

Sucuri Sitecheck is an open source web application scanner. This scanner takes only two steps for scanning. The user pastes the website URL to the text box and goes to the “Scan Website” button. This scanner will discover vulnerabilities and out-of-date plug-ins.

Apart from these scanners, Acunetix performs efficient scanning in a lot of vulnerabilities including SQLi and XSS. It is an effective web security scanner that can accurately detect over 7000 vulnerabilities in many websites. It is also an automated tool for testing web application security. It can detect vulnerabilities more than other web security scanners with a fewer false positive. Because it uses AcuSensor technology. This technology supports more information concerning the vulnerabilities that allow to find the vulnerability faster and more accurately.

In this thesis, the proposed XSS_SQL_Scanning algorithm uses the methodology of web application scanner and is built for detecting XSS and SQL injection vulnerabilities. It uses the Acunetix scanner to evaluate the accuracy of the system. In addition, it applies pattern matching algorithm for searching malicious strings.

2.6 Pattern Matching Algorithm

The main task of pattern matching in computer science is to seek the specific sequences in the raw data files such as notepad, word file, web browser, and database. The pattern matching algorithm includes the finding string (also called the pattern) and a given text string. Pattern matching algorithms apply to multiple solutions to real-world problems. They are most useful in plagiarism detection, natural language and image processing, bioinformatics processing, intrusion detection, digital forensics, web application vulnerability detection, and other domains.

There are many kinds of pattern matching algorithms, the most common are as follows:

- Rabin Karp Pattern Matching Algorithm
- Boyer-Moore String Matching Algorithm
- Knuth-Mooris-Pratt Algorithm
- Naïve Pattern Matching Algorithm

All of these algorithms have their own advantages and disadvantages.

2.6.1 Rabin Karp Pattern Matching Algorithm

Rabin Karp algorithm is also a pattern matching algorithm that searches patterns in the text using the hash function. It calculates the hash value in the pre-processed state and compares the hash value rather than the character of the string. And then the current position of the hash value is used to compute the next position of the hash value. There is no need to travel each character of the string like a Naïve pattern searching algorithm.

The average complexity of the Rabin Karp algorithm is O(m+n) and the worst-case complexity is O(mn) in the spurious hits. Although the hash value of the pattern matches the hash value of the text, if this text is not existing then it is called a spurious hit. This algorithm is quite effective because it uses hashing function. But it requires extra space and is practically slow. It was widely used in multiple patterns matching such as plagiarism checking. The flow diagram of the Rabin Karp algorithm is shown in Figure 2.1.

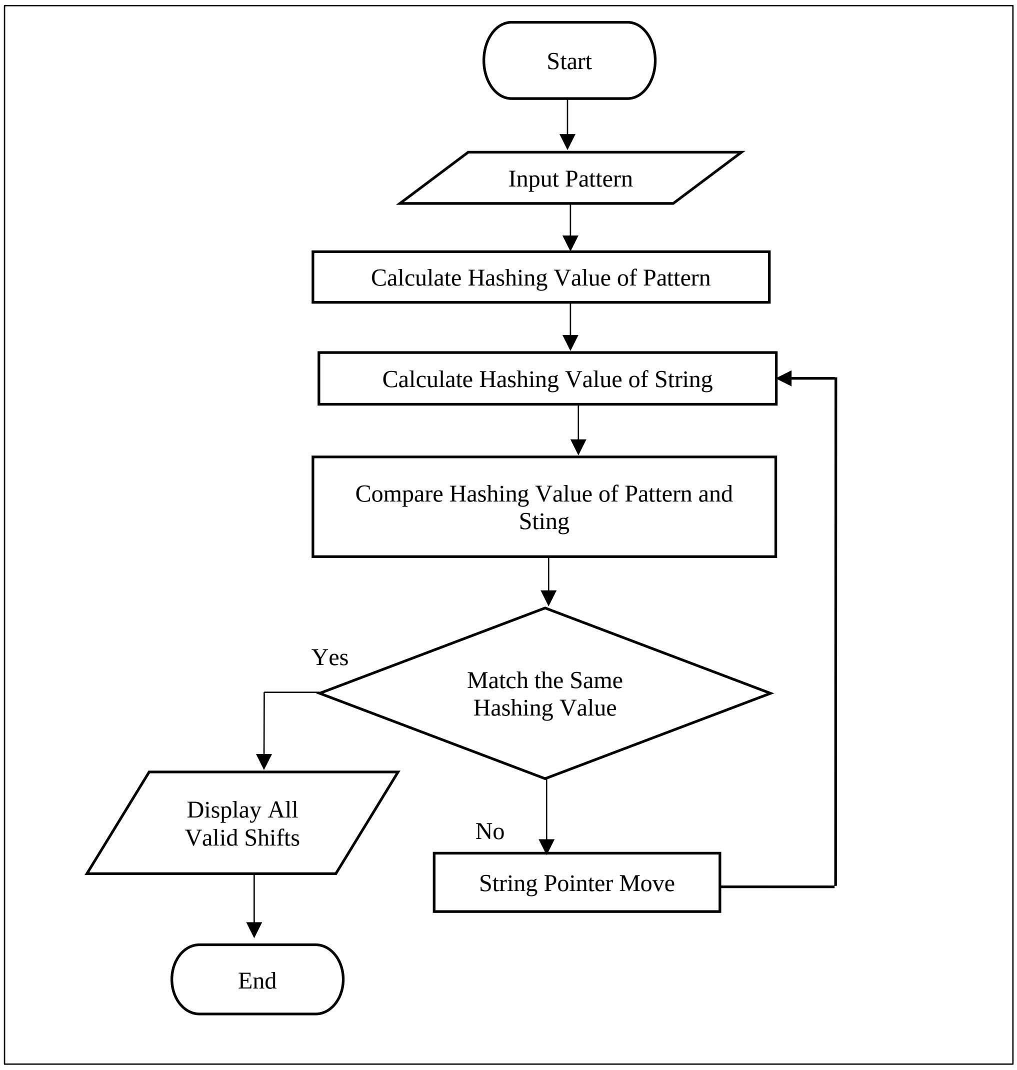

Start
Input Pattern
Calculate Hashing Value of Pattern
Calculate Hashing Value of String
Compare Hashing Value of Pattern and Sting
Match the Same Hashing Value
Yes
No
Display All Valid Shifts
String Pointer Move
End

Figure 2.1 Flow Chart of Rabin Karp Pattern Matching Algorithm

2.6.2 Boyer-Moore String Matching Algorithm

One of the most efficient string searching algorithms is the Boyer-Moore algorithm. Boyer-Moore algorithm preprocesses the pattern. This algorithm gathers information during the preprocess state to skip the length of shifts. It works by jumping characters via the text instead of searching for every character in the string. The main characteristic of this algorithm is to match the end of the pattern rather than the first character of the pattern. Pattern is split into two halves, namely P1 and P2. Boyer-Moore algorithm is also called the index search algorithm just like the Knuth-Morris-Pratt algorithm. This algorithm is shown in the following Figure 2.2.

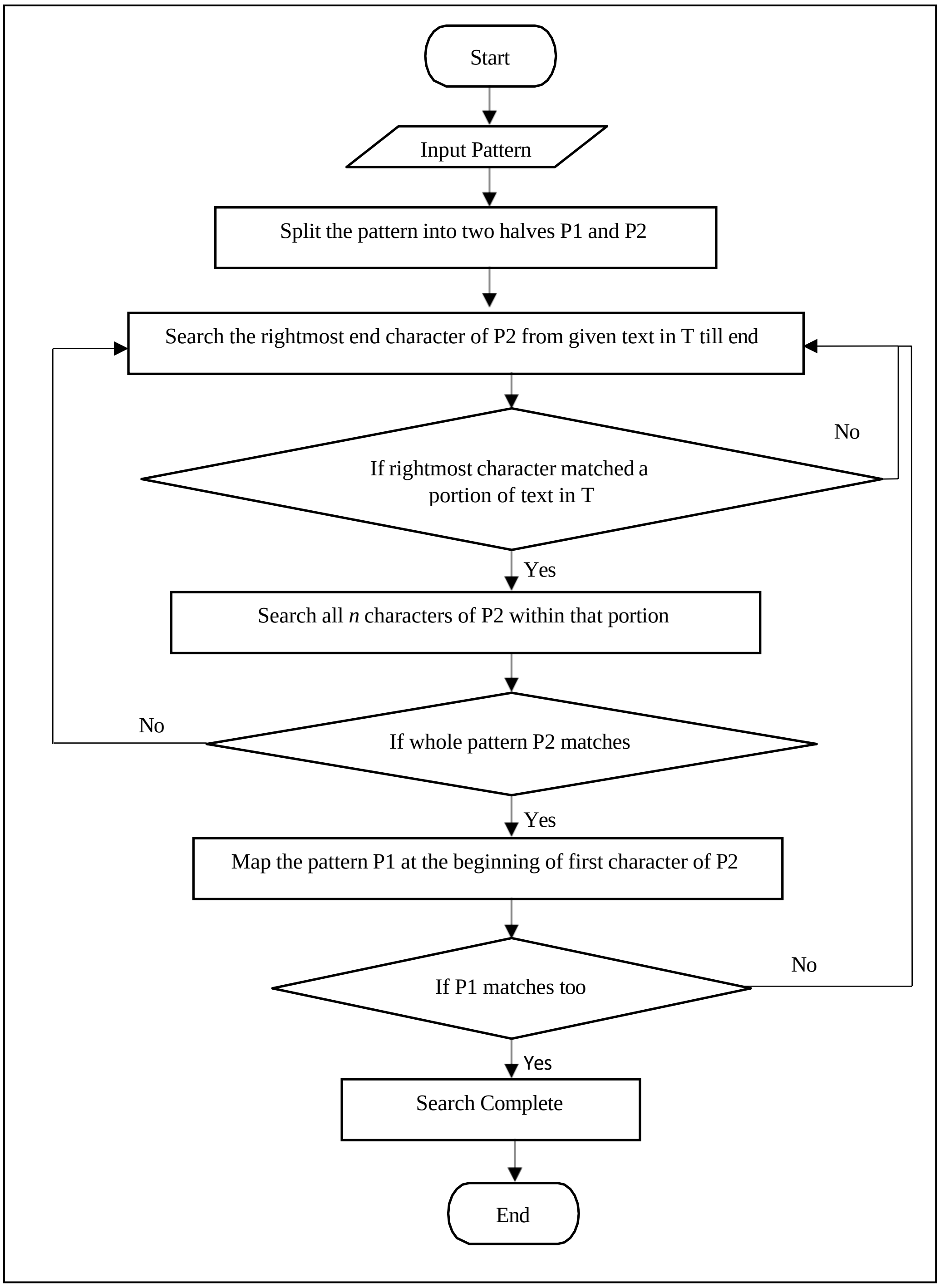

Start
Input Pattern
Split the pattern into two halves P1 and P2
Search the rightmost end character of P2 from given text in T till end
If rightmost character matched a portion of text in T
No
Yes
Search all *n* characters of P2 within that portion
If whole pattern P2 matches
No
Yes
Map the pattern P1 at the beginning of first character of P2
If P1 matches too
No
Yes
Search Complete
End

Figure 2.2 Flow Chart of Boyer-Moore Pattern Matching Algorithm

Unlike other pattern matching algorithms, the Boyer-Moore algorithm scans the character of the text from left to right and the shifting process is decided based on two rules. They are bad character rule and good suffix rule.

The time complexity of the Boyer-Moore-Algorithm is O(n/m) and the worst case is O(mn) where m is the length of the pattern and n is the length of the string. If the character does not match, the pattern pointer shifts with the calculated preprocessed state. The good result of complexity is due to the preprocess state of the pattern. Both two combination rules provide the best shift value, though the preprocessing of these rules are complicated than another pattern matching algorithm.

2.6.3 Knuth-Mooris-Pratt Pattern Matching Algorithm

The Knuth-Mooris-Pratt algorithm is similar to the Naïve algorithm, except that the KMP algorithm needs preprocessing phase. This algorithm requires two functions to search. These functions are prefix function and string-matching function. This preprocess phase computes how many characters are to be skipped. It improves the length of shifts and indexes parts of the text that match the pattern.

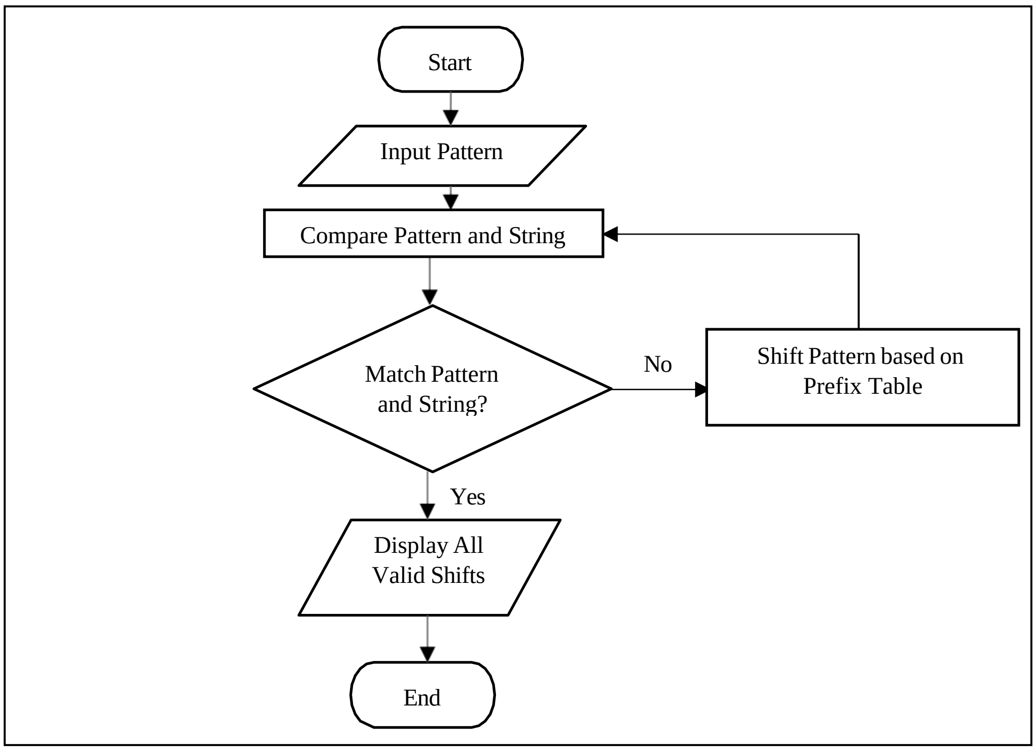

Start
Input Pattern
Compare Pattern and String
Match Pattern and String?
No
Shift Pattern based on Prefix Table
Yes
Display All Valid Shifts
End

Figure 2.3 Flow Chart of Knuth-Morris-Pratt Pattern Matching Algorithm

The time complexity of the first method takes O(m) where m is the length of the pattern. Next, it compares the same way as the Naïve pattern matching algorithm. If the pattern does not match the current index of the string, then it does the preprocessing phase and shifts the index of the text, and then matches again between the characters of the text and the character of the pattern. This pattern and string compare through the length of string ‘n’. So, the time complexity of the string-matching method is O(n). Thus, the time complexity of the Knuth-Morris-Pratt Algorithm is

O(m+n) which is pretty fast. The flow diagram of the Knuth-Morris-Pratt pattern matching algorithm is given in Figure 2.3.

On the other hand, it does not work when the size of the alphabet is increased. It may happen in mismatching cases. It optimizes the Naïve pattern matching algorithm by reducing the times of comparison.

2.6.4 Naïve Pattern Matching Algorithm

The Naïve algorithm is the simplest algorithm and easier to implement and understand among other pattern matching algorithms. It is widely used in pattern searching, matrix multiplication and string matching. For finding patterns, it compares pattern and text from left to right and character by character until a match is found.

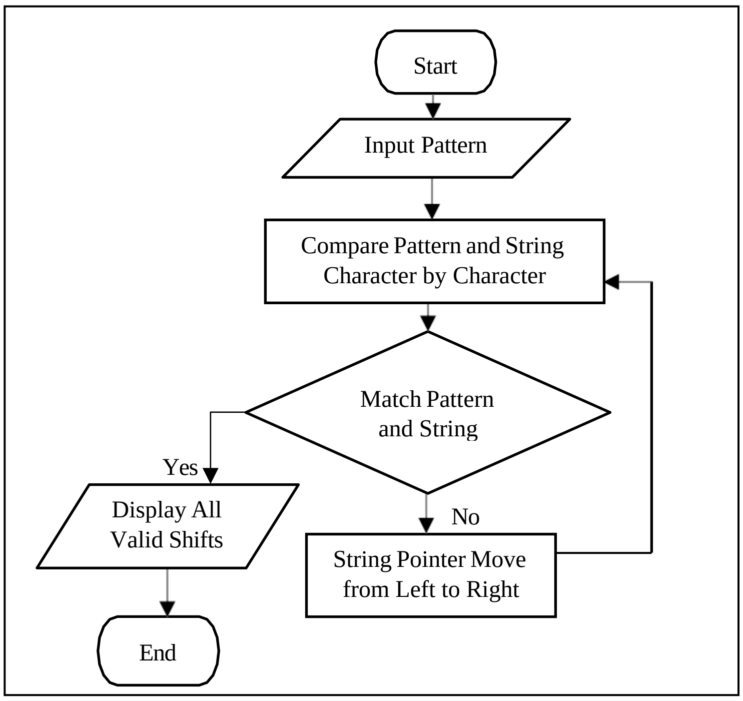

Start
Input Pattern
Compare Pattern and String Character by Character
Match Pattern and String
Yes
No
Display All Valid Shifts
String Pointer Move from Left to Right
End

Figure 2.4 Flow Chart of Naive Pattern Matching Algorithm

The Naïve algorithm finds all valid shifts using a loop that checks the condition P [1...m] = T [s+1 ...s+m] for each of the n – m + 1 possible values of s. T refers to text and the symbol of P is pattern, m is the length of pattern, and n is the length of text. The testing of the loop determines whether the current shift of pattern upon the text is valid or not. This loop involves an implicit loop for checking the corresponding character positions until all positions match successfully or mismatch is found. When a match is found then it returns the starting index of the found pattern and then it slides once again to check the subsequent matches [5]. The flow chart of the Naïve Pattern Matching

Algorithm is shown in Figure 2.4. The searching phase of the comparison of the character can be ended in any order because the Naïve algorithm does not need preprocessing state.

It takes a running time was O(mn) in the worst case, but searching the ordinary text takes O(m+n) where m is the length of the pattern and n means the text length that is pretty quick. In this thesis, the Naïve pattern matching algorithm is chosen to search for the specious features in web applications.

CHAPTER 3
DESIGN OF THE PROPOSED SYSTEM

The main goal of this thesis is to detect the web application vulnerability with the vulnerability scanning method. The XSS_SQL_Scanning algorithm is applied for scanning vulnerabilities. Firstly, the overview of the proposed system of system architecture is described. And each of the algorithm that takes part in the main program is described in a detailed explanation. This chapter mainly focuses on the design of the system.

3.1 Overview of the Proposed System

The proposed algorithm, XSS_SQL_Scanning algorithm is deal with vulnerabilities of SQL injection and Cross-site Scripting. The expected architecture of the system is shown in Figure 3.1.

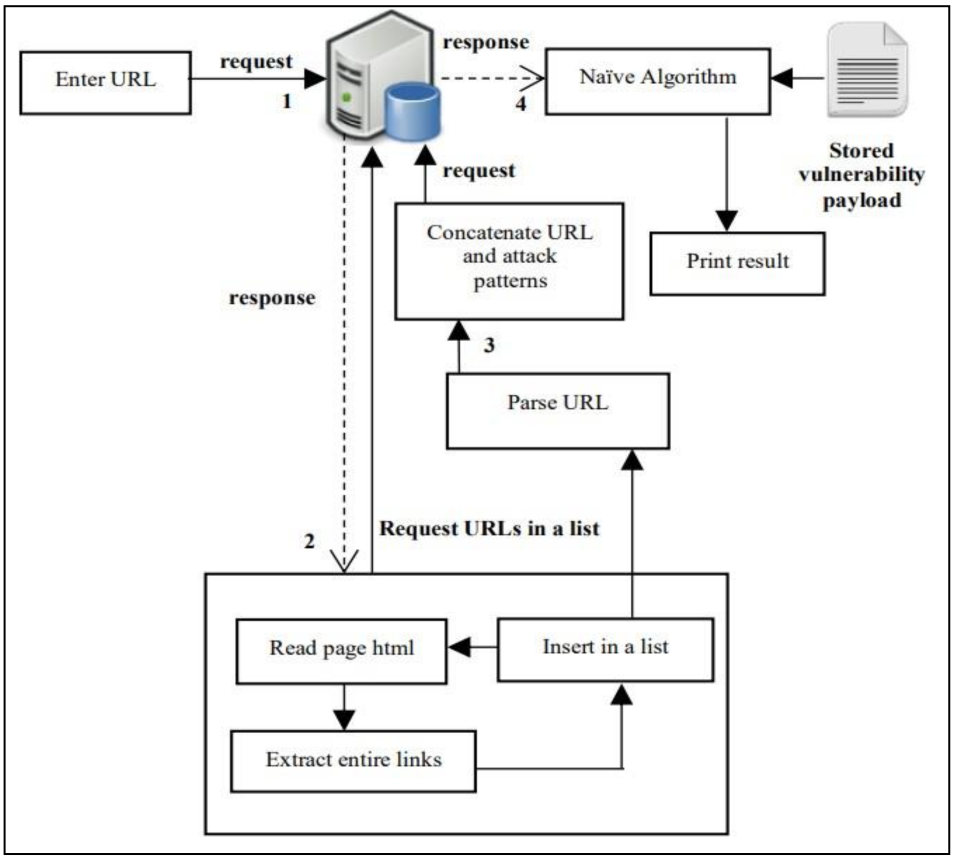


Figure 3.1 Architecture of Web Application Vulnerability Testing System Using Proposed XSS_SQL_Scanning Algorithm

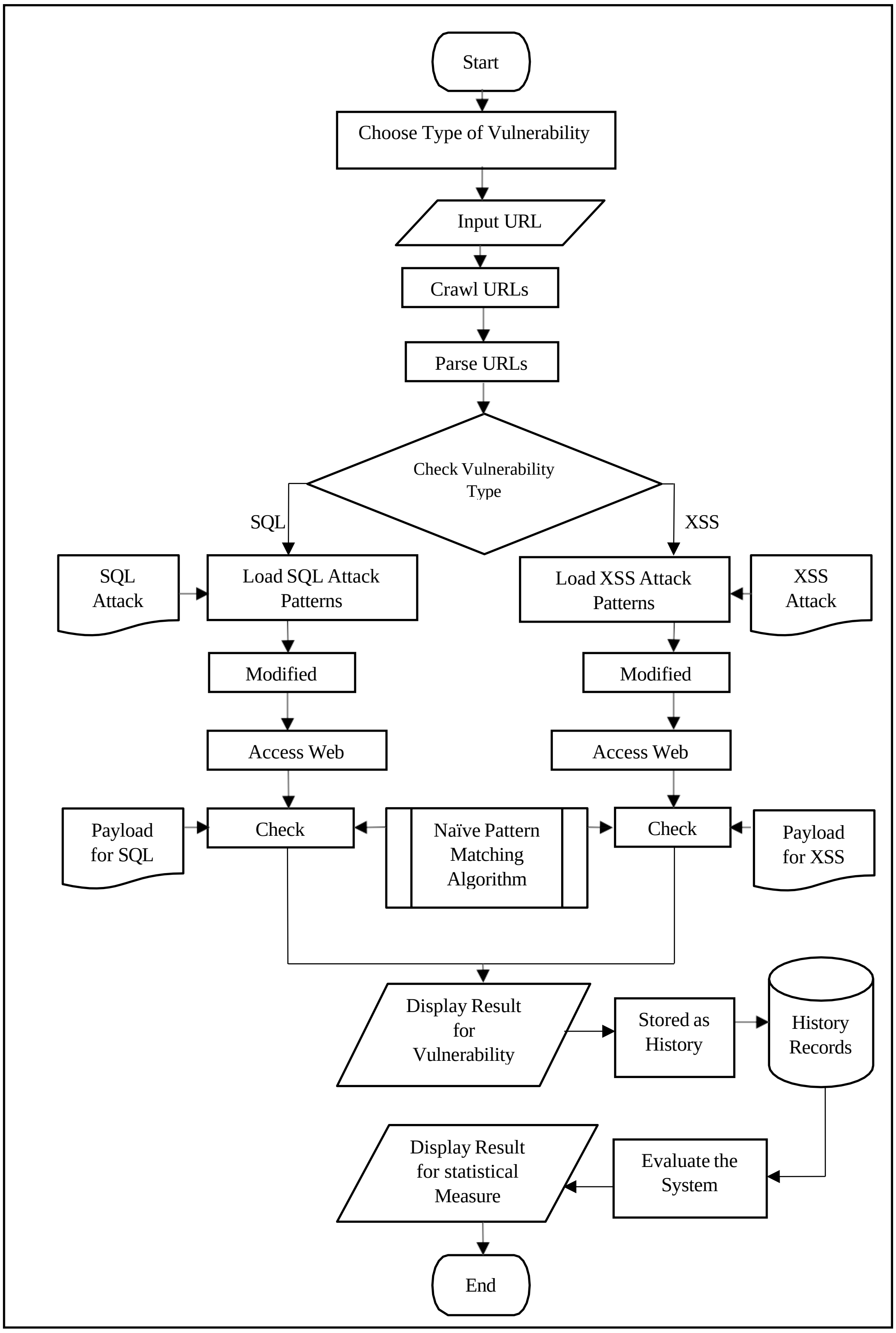


Figure 3.2 Overview of the Proposed System

Figure 3.1 and Figure 3.2 show the system architecture and an overview of the proposed system. Both of these two vulnerabilities detect to look for attack signatures that are caused by the system sending HTTP requests to the web server with attack patterns. The proposed system can detect the types of SQL injection such as error based, blind based, time based, and XSS vulnerability. Each attack pattern is collected from the vulnerable websites. For detecting vulnerability, the proposed system uses a Naïve pattern matching algorithm and analyzes the behavior of the web application response and delay time. Naïve pattern matching algorithm helps to search attack signatures in the response by using stored vulnerability payloads. The proposed XSS_SQL_Scanning Algorithm, which discovers Cross-Site Scripting and SQL injection vulnerabilities that are presented in Figure 3.3 and then the detailed explanation is described in this subsection.

```
Algorithm: XSS_SQL_Scanning Algorithm
Input Data:
url                    =    URL of the detected website
payload                =    the previous existed attack patterns associated with
                            SQL injection and XSS attack
type_of_vulnerability  =    the types of vulnerability such as SQL or XSS
Output:
status whether vulnerability found or not
Begin
   if type_of_vulnerability == SQL
           urls    ← Call Crawling(url)
           result  ← Forwarding_Payload_and_Analyzing_Response (payload, urls)
           Check SQL Blind-based
           Check SQL Time-based
   else if type_of_vulnerability == XSS
           urls    ← Call Crawling(url)
           result  ← Forwarding_Payload_and_Analyzing_Response (payload, urls)
   end
End
```

Figure 3.3 XSS_SQL_Scanning Algorithm

In order to complete this proposed algorithm, there are four phases included.

- **Phase 1:** Scanning every possible page on web application
- **Phase 2:** Looking for HTTP request methods and parsing the URL
- **Phase 3:** Forwarding payloads to detect the web application
- **Phase 4:** Analyzing the response to find the vulnerabilities

3.1.1 Scanning Every Possible Page on Web Application

Scanning or crawling the entire web application is necessary to achieve all possible links. This process is based on the tree structure in order to scan this web application. The following proposed algorithm, name as crawling performs the process described in these steps.

```
Algorithm: Crawling Algorithm
Input Data:
url     =     URL of the detected website
Output:
all possible urls in the website
Begin
    function Crawling (url)
        Request url
        while each new urls is not already in the list do
            response(rep) ← Get the response of url
            page_html     ← Read page html in rep
            links         ← Extract entire links in page_html
            list          ← Insert links to a list
            urls          ← Request URLs in the list
        end while
        return urls
    end function
End
```

Figure 3.4 Crawling Algorithm

In this algorithm, the user enters the target URL of the web application when starting this system. Firstly, it calls the target URL and then requests this URL. The crawling process starts from the response of the requested URL. And then it examines page HTML (*page_html*) in the target URL's response for extracting the respective links (*links*). Each link is requested and appended to a list (*list*). This procedure executes recursive crawling until no more links are visited in a list. The links are automatically visited by simulating the user's clicking on URLs (*urls*). All of the possible links in the target web application are collected in this scanning phase.

After the website links are collected in this phase, the next phase is searching method for sending data to the web application before detecting the web application vulnerabilities.

3.1.2 Looking for HTTP Request Method and Parsing the URLs

The browser sends HTTP requests to the server. In general, there are two methods as the GET method and the POST method. GET method requests through the URL with the parameters as in this example is http://website.com/example.php?id=1 in which the position of id is a parameter and then this parameter's value is 1 whereas POST method requests through input elements such as search boxes, login boxes, and input box, etc.

The purpose of searching for these methods is to send data such as the URL with the attack patterns to the web application. Each type of request URL determines the input points why the GET method sends the information by appending the URL with a query string while the POST method requests the data enclosed in the HTTP message body. These input points allow the attackers to inject the database by sending malicious SQL commands or scripting language (XSS). And then, the system parses the URL to know the position of the input points. Parsing the URL means that the URL string splits into its components in which the attributes in the URL's components such as scheme, netloc, path, params, query, and fragment.

The word scheme means the name of the protocol and is usually expressed as http or https. The word netloc is the combination of two words. They are networks and locations which contain the domain, subdomain, and port number. The path contains the predetermined resource for accessing. The params is an element that combines fine-tuning with the path. The query is a query component of the URL. The fragment is a fragment identifier that concatenates with the path or query. The SQL injection code and script code are treated through the user’s input variables such as the input points. This input points position is most of the params and query from the URL parse. This is for example of URL parsing.

- https://www.example.com/product.php?cat_id=5
    - scheme (protocol) = ‘http’
    - netloc (hostname) = ‘www.example.com’
    - path (pathname) = ‘product.php’
    - params = ‘’
    - query (search) = ‘cat_id = 5’
    - fragment = ‘’

After the website links are collected in this phase, the next phase is the searching method for sending data to the web application before detecting the web application vulnerabilities.

3.1.3 Forwarding Payloads to Detect the Web Application

After identifying the input points in the previous step, the next step is to generate attacks. For making the attack, the algorithm needs to establish the attack payload. This payload is based on the previous existed attack patterns associated with SQL injection and XSS attacks. They are used by attackers in generating SQL injection and XSS attacks. The system can detect the web application by concatenating the payload to each crawled URL.

For example, the URL of http://sis.xyz.edu.mm/MajorList.jsp?major=1 transforms “http://sis.xyz.edu.mm/MajorList.jsp?major=1 AND 1=2 -- - “for SQL injection and “http://sis.xyz.edu.mm/MajorList.jsp?major=1<svg/onload=alert(1)>” for XSS detecting. This modified URL requests to the web server and then the server generates the response. If the detected web application has the vulnerability, the possible vulnerable signature is attached with the response. The sample payloads of SQL injection and XSS attack are shown in Table 3.1. These payloads are used to verify the web applications which ae vulnerable to SQL and XSS [2].

Table 3.1 Sample payloads to detect web application vulnerability for SQL and XSS attacks

No.	XSS	SQLi
1.	<>'	SELECT SLEEP(2)
2.	<>"	sqlite3_sleep(2000)
3.	" '	WAITFOR DELAY '00:00:01'
4.	<svg "ons>	\
5.	onfocus="alert(1);	'AND 1=1--', ' AND 1=2--’
6.	<svg/onload=alert(1)>	' AND 1=1#', ' AND 1=2#’
:	:	:

In the next phase, the system finds out the possible vulnerable signatures or patterns in this response.

3.1.4 Analyzing the Response

In this phase, the system analyzes the modified url’s (*modified_url*) response from the web server. For normal SQL injection and XSS vulnerability detection, the system uses the Naïve Pattern matching algorithm that uses this modified HTTP response and stored vulnerability payloads (*stored_vulnerability_payload*) such as database errors and attack patterns that are already described in the previous phase. The vulnerability payloads for database errors are collected from each type of database error in different kinds of databases.

The system analyzes the response of the modified URL with the corresponding payloads. For detecting blind-based SQL injection, the server does not return any queries or errors. So, the system asks the database true or false questions and examines the content of the page based on the response to these questions. In time-based SQL injection, the system employs SQL queries with time delay and conditional error statements. Then the URL concatenates this query and requests to the database. The modified URL forces the database for waiting the delay time before responding. If the database accepts this payload, the returned database response will be delayed for a specified amount of time. If not, the HTTP response will be returned immediately or any data from the database. And afterwards, the system analyzes the response and discovers any suspicious features in this response.

When any vulnerability is found in the response page, it will show the corresponding URL and the vulnerable input points as an example of URL like "http://sis.xyz.edu.mm/MajorList.jsp?major=1 AND 1=2 -- - possible blind SQL". In addition, it also provides a report as a text file that consists of all vulnerable URLs, the processing time, and the list of prevention factors associated with SQL and XSS. Table 3.2 shows sample examples of SQL database errors [3].

Table 3.2 Database errors of SQL

No.	SQL database error
1.	"You have an error in your SQL syntax"
2.	"supplied argument is not a valid MySQL result resource"
3.	"check the manual that corresponds to your MySQL"
4.	"mysql_fetch_array():supplied argument is not a valid MySQL"
5.	"function fetch_row()"
6.	"Microsoft OLE DB Provider for ODBC Drivers error"
7.	"mysql_num_rows()"
8.	"mysql_free_result()"
9.	"Error Occurred While Processing Request"
10.	"Invalid Querystring"
11.	"sybase_"
12.	"Input string was not in a correct format"
13.	"Warning: require"
14.	"Warning: main"
:	:

The following algorithm, name as Forwarding_payload_and_Analyzing_response can make the process of sending payload and analyzing the response. This algorithm is also a combination of three phases which are phases two, three and four.

```
Algorithm: Forwarding_payload_and_Analyzing_response Algorithm
Input Data:
 payload  =      the previous existed attack patterns associated with SQL injection
                 and XSS attack
 stored_vulnerability_payload       =       database errors and attack patterns
 url      =      the crawled URLs
Output:
 status whether vulnerability exist or not
Begin
   function Forwarding_payload_and_Analyzing_response (payload, url)
            Search request method whether GET or POST
            input_point ← Parse url
            Load attack patterns associated with SQL or XSS
            modified_url ← Concatenate attack patterns with possible URL's
                                   input_point
             Make request for the modified_url
            response(rep) ← Get the response of the modified_url
            result(res) = NAÏVE_ALGORITHM (rep, stored_vulnerability_payoad)
            if (res = = TRUE)
                     Display vulnerability is Found
            else
                     Display vulnerability is not Found
            endif
   end function
 End
```

Figure 3.5 Forwarding_payload_and_Analyzing_response Algorithm

3.2 Performance Evaluation

The proposed system was evaluated in terms of true positive and negative, false positive, and negative rates. It was based on the correctness of vulnerability detection. Or in other words, accuracy indicates that the detected results are nearly similar to the accepted or correct value and it is also based on the condition of false positive and negative rates.

For the evaluation in this system, the commercial tool, Acunetix was compared to the result of the proposed system. The contingency table used to evaluate the result of the proposed system is shown in Table 3.3.

Table 3.3 Specification of true positive, false positive, true negative and false negative rates

	Has Vulnerability	**Does not have vulnerability**
Positive	**True Positive (TP)** **TP rate =** $\frac{TP}{FN+TP}$	**False Positive (FP)** **FP rate =** $\frac{FP}{TN+FP}$
Negative	**False Negative (FN)** **FN rate =** $\frac{FN}{FN+TP}$	**True Negative (TN)** **TN rate =** $\frac{TN}{TN+FP}$

The description of each variables are as follows.

- **True Positive (TP):** The proposed system can accurately detect the vulnerability when the website is actually vulnerable.
- **True Negative (TN):** The proposed system cannot actually detect the vulnerability when the website is not vulnerable.
- **False Positive (FP):** The proposed system can detect vulnerability while the website has no vulnerability.
- **False Negative (FN):** The proposed system cannot detect vulnerability but there is actually the vulnerability in the website.

For example, the Acunetix result showed that www.example.com had SQL injection vulnerability and the proposed system also showed that this website had SQL injection vulnerability. They showed the same result. Thus, it could be said True Positive (TP). True Negative (TN), False Positive (FP) and False Negative (FN) were calculated in this way.

The accuracy was calculated by the following equation.

$$\text{Accuracy} = ((\text{TPR}+\text{TNR})/(\text{TPR}+\text{FPR}+\text{TNR}+\text{FNR})) * 100 \quad (3.1)$$

- TPR (True Positive Rate) = TP/(TP+FN)
- FPR (False Positive Rate) = FP/(FP+TP)
- TNR (True Negative Rate) = TN/(TN+FP)
- FNR (False Negative Rate) = FN/(FN+TP)

3.3 URLs Used in the Experiment

To evaluate the performances of the system, the 201 different URLs are used as case study. For stored URLs, the 201 websites' URLs of web applications were firstly collected and saved them into the database. All of these web applications are gathered from both manual and exploit databases. Some of them are shown in Table 3.4.

Table 3.4 Sample URLs Collection of Different Web Applications

No.	URL	Vulnerable
1.	"http://www.anfield.com.hk/whampoa/aboutus.php?id=14"	XSS
2.	"https://www.vuototecnica.co.uk/news.php?id=105"	XSS
3.	"http://gwg-gabrovo.com/products.php?id=970"	XSS
4.	"https://www.vuototecnica.co.uk/news.php?id=105"	XSS
5.	"http://www.agilebull.com.cn/aboutus.php?id=6"	XSS
6.	"http://aceronline.net/product.php?cid=24"	SQLi
7.	"https://www.maritimewelding.com/products.php?id=2"	SQLi
8.	"https://www.ninenik.com/content.php?arti_id=107"	SQLi
9.	"https://elevonwatches.com/product.php?id=5768"	SQLi
10.	"http://miurashoren.com/shop_search.php?id=172"	SQLi
11.	"https://saberes.senado.leg.br/course/index.php?categoryid=134"	NOVul
12.	"https://bim.easyaccessmaterials.com/index.php?location_user=cchs"	NOVul
13.	"http://www.mfsociety.org/page.php?pageID=185"	NOVul
14.	"https://www.nutrivene.com/product-category/minerals/"	NOVul
15.	"http://www.honeycombine.com/product.php?id=1"	NOVul

CHAPTER 4

IMPLEMENTATION OF THE PROPOSED SYSTEM

4.1 Experimental Setup

The purpose of this chapter is to present the implementation, design, and performance evaluation of the proposed system. The web application testing system can use to detect the vulnerability of the web application. This system is implemented by using Python programming language and MySQL server is used to store the previously detected result for performance evaluation.

4.2 Implementation of the System

When the system starts, the user can see the main form of the system as shown in Figure 4.1. The main form consists of the entry box for entering the URL, the two main options namely SQL and XSS, and the three panels namely, Crawl Page, Links with Parameters and Testing, and test accuracy button respectively.

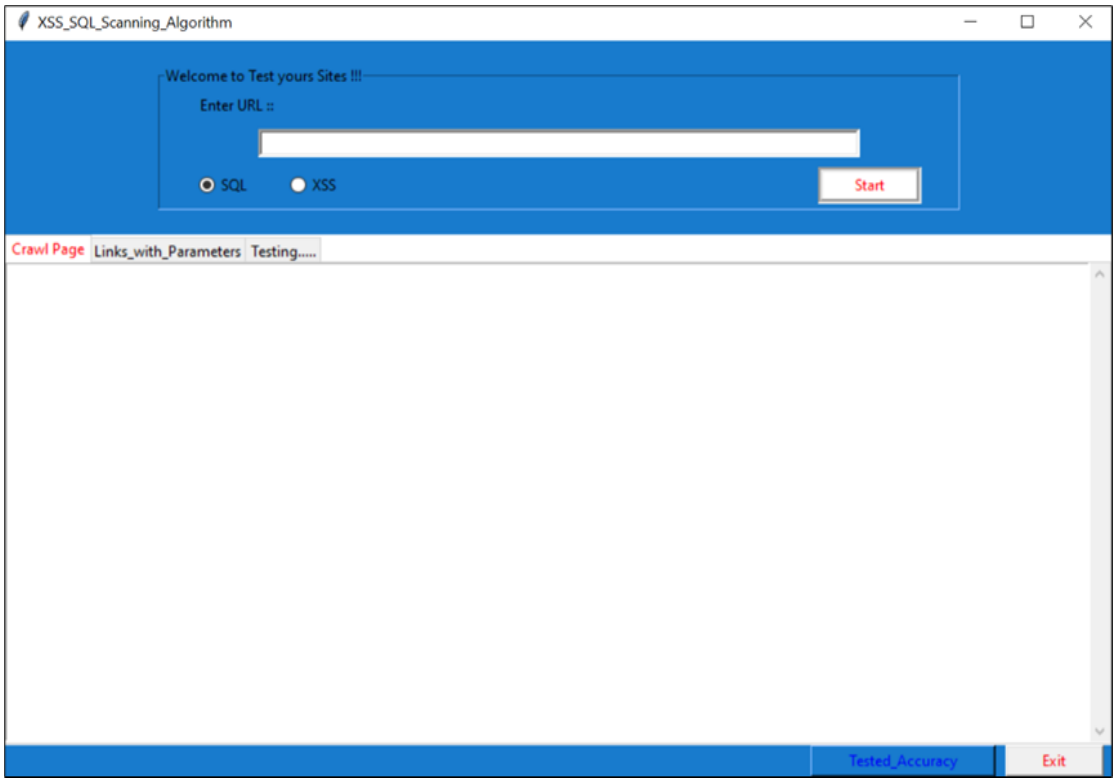
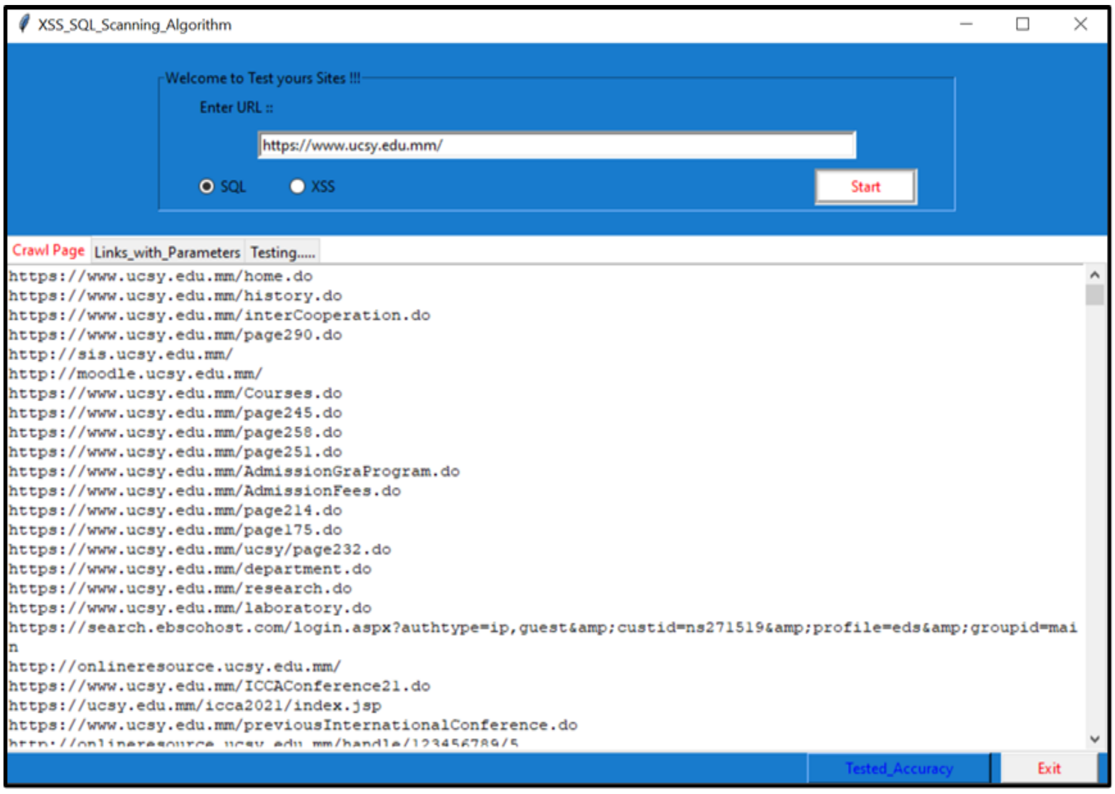


Figure 4.1 Main form of the proposed system

In the main form, the entry box is used for the required URL information tested in the system. The user fills the URL in this entry box. The two options are used to detect the XSS vulnerability and SQL injection vulnerability. The user needs to choose one of them to detect the desire vulnerability. The three panels are used to show the detail description and analysis results of the proposed the system. They are appeared page by page when testing the website. The first panel of Crawl Page is shown in Figure 4.2. And, the last button of the system, Tested Accuracy button, allows to view the result comparison and the accuracy calculation of the proposed system.



Enter URL :: https://www.ucsy.edu.mm/

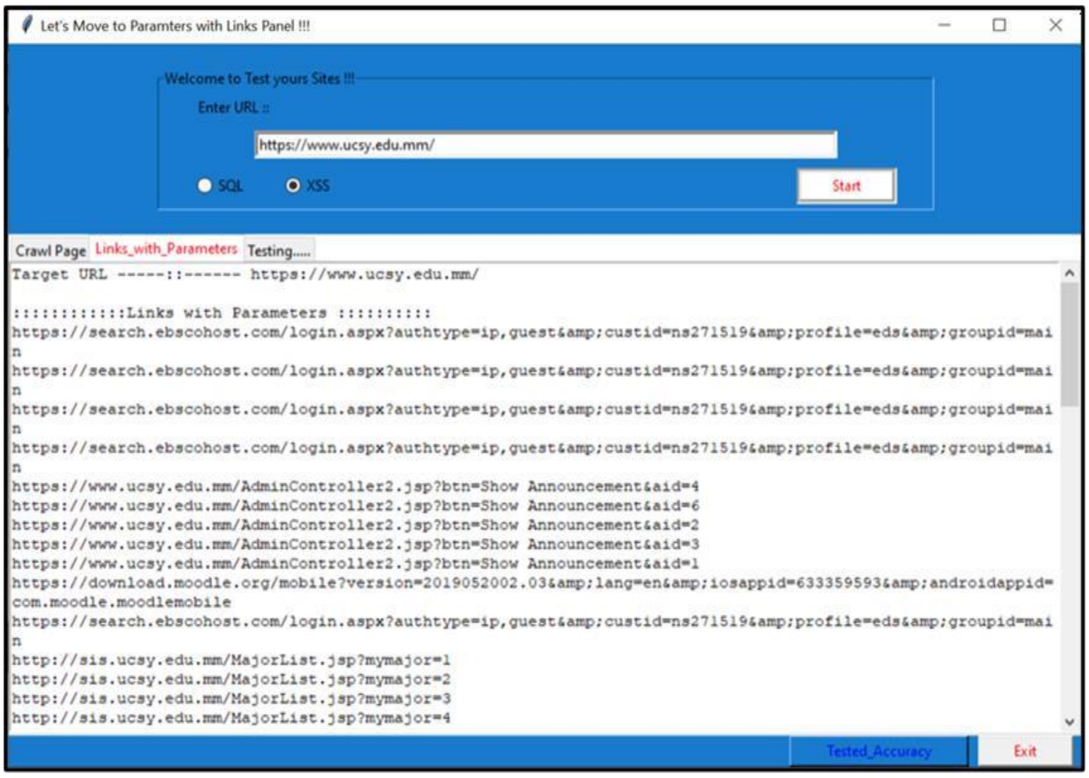
[✓ SQL] XSS

```
https://www.ucsy.edu.mm/home.do
https://www.ucsy.edu.mm/history.do
https://www.ucsy.edu.mm/interCooperation.do
https://www.ucsy.edu.mm/page290.do
http://sis.ucsy.edu.mm/
http://moodle.ucsy.edu.mm/
https://www.ucsy.edu.mm/Courses.do
https://www.ucsy.edu.mm/page245.do
https://www.ucsy.edu.mm/page258.do
https://www.ucsy.edu.mm/page251.do
https://www.ucsy.edu.mm/AdmissionGraProgram.do
https://www.ucsy.edu.mm/AdmissionFees.do
https://www.ucsy.edu.mm/page214.do
https://www.ucsy.edu.mm/page175.do
https://www.ucsy.edu.mm/ucsy/page232.do
https://www.ucsy.edu.mm/department.do
https://www.ucsy.edu.mm/research.do
https://www.ucsy.edu.mm/laboratory.do
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
http://onlineresource.ucsy.edu.mm/
https://www.ucsy.edu.mm/ICCAConference21.do
https://ucsy.edu.mm/icca2021/index.jsp
https://www.ucsy.edu.mm/previousInternationalConference.do
http://onlineresource.ucsy.edu.mm/handle/123456789/5
```

Figure 4.2 Scanning possible URLs of Web Application

Firstly, the user who wants to detect the vulnerability, the user needs to enter the website URL into the entry box. After filling the input URL, the user requires to choose the desire options. By choosing the desire option from the radio button and clicking start button, all of the possible URLs of the detected web application can be seen in Figure 4.2. Start button is used to operate this process.

And then the scanning process moves on the next panel. This panel allows the user to view the query of the URLs that filter the query of the URL's parse. This query comes from the result of crawled URL. The result of filtering is shown in Figure 4.3.

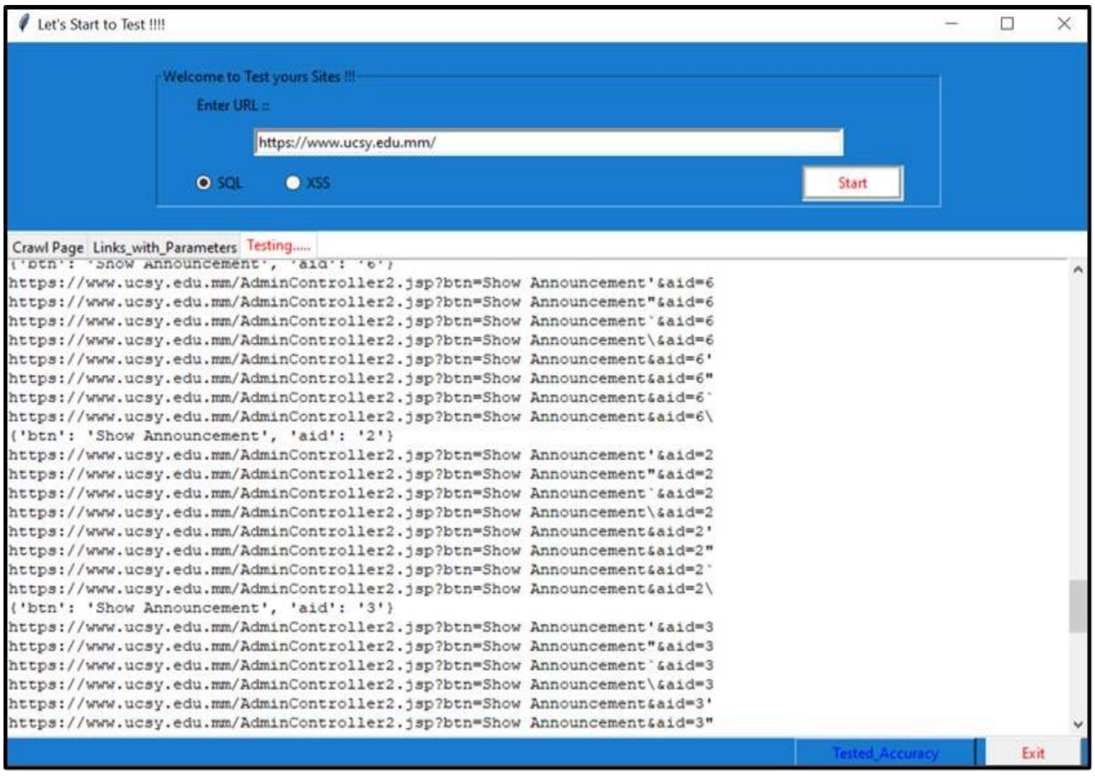


```
Target URL -----::------ https://www.ucsy.edu.mm/

::::::::::::Links with Parameters ::::::::::
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=4
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=6
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=2
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=1
https://download.moodle.org/mobile?version=2019052002.03&amp;lang=en&amp;iosappid=633359593&amp;androidappid=
com.moodle.moodlemobile
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=1
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=2
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=3
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=4
```

Figure 4.3 Filtering query of URL's parse from the result of crawled URL

After filtering panel has finished, testing process is appeared in the testing panel. Figure 4.4 shows concatenating SQL injection quotes with URLs for testing vulnerability.

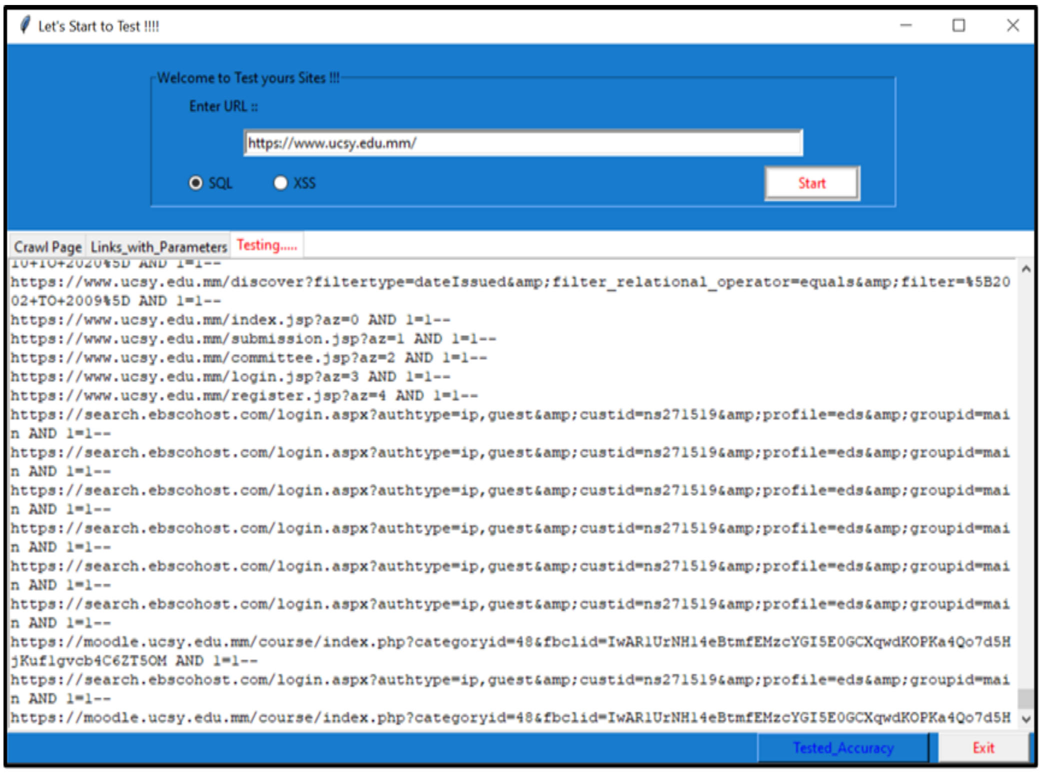


```
{'btn': 'Show Announcement', 'aid': '6'}
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement'&aid=6
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement"&aid=6
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement`&aid=6
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement\&aid=6
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=6'
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=6"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=6`
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=6\
{'btn': 'Show Announcement', 'aid': '2'}
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement'&aid=2
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement"&aid=2
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement`&aid=2
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement\&aid=2
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=2'
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=2"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=2`
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=2\
{'btn': 'Show Announcement', 'aid': '3'}
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement'&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement"&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement`&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement\&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3'
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3"
```

Figure 4.4 Testing URLs with SQL quotes for SQL injection

If the user detects the website with SQL injection vulnerability, the system is tested using SQL payloads. These payloads include SQL quotes, SQL time-based payload and SQL blind based payload. Testing process of SQL injection vulnerability is seen in Figure 4.4, Figure 4.5 and Figure 4.6 respectively.

Figure 4.5 shows the process of testing URLs with blind-based SQL injection payloads.

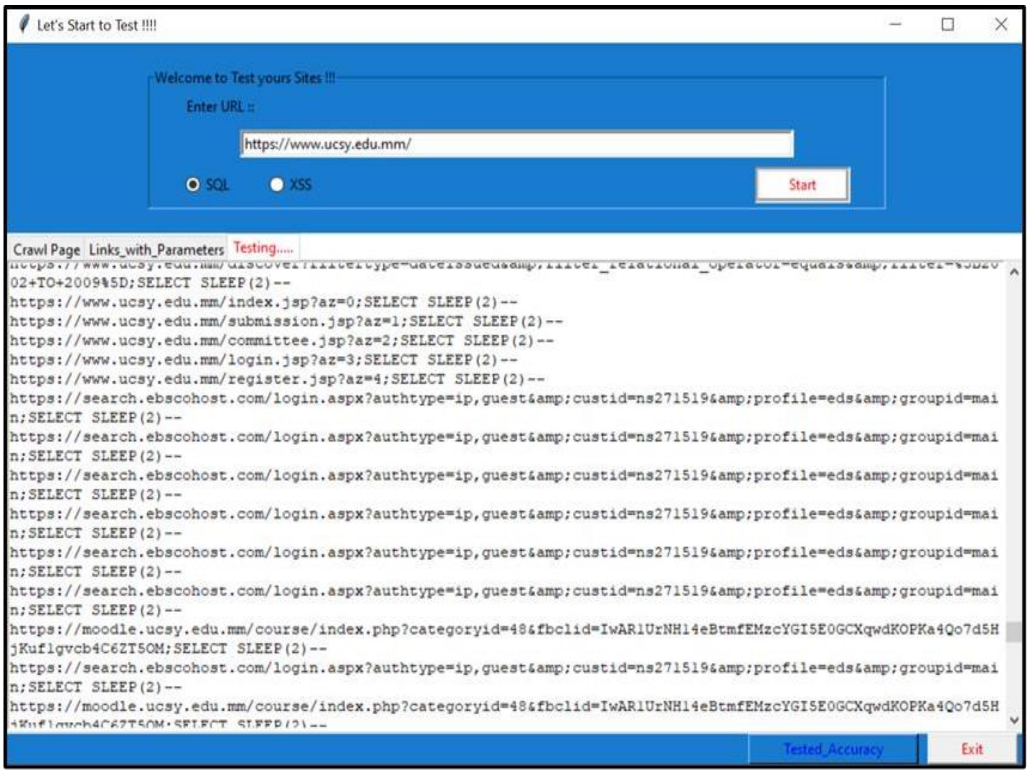


```
10+10+2020%5D AND 1=1--
https://www.ucsy.edu.mm/discover?filtertype=dateIssued&amp;filter_relational_operator=equals&amp;filter=%5B20
02+TO+2009%5D AND 1=1--
https://www.ucsy.edu.mm/index.jsp?az=0 AND 1=1--
https://www.ucsy.edu.mm/submission.jsp?az=1 AND 1=1--
https://www.ucsy.edu.mm/committee.jsp?az=2 AND 1=1--
https://www.ucsy.edu.mm/login.jsp?az=3 AND 1=1--
https://www.ucsy.edu.mm/register.jsp?az=4 AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://moodle.ucsy.edu.mm/course/index.php?categoryid=48&fbclid=IwAR1UrNH14eBtmfEMzcYGI5E0GCXqwdKOPKa4Qo7d5H
jKuf1gvcb4C6ZT5OM AND 1=1--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n AND 1=1--
https://moodle.ucsy.edu.mm/course/index.php?categoryid=48&fbclid=IwAR1UrNH14eBtmfEMzcYGI5E0GCXqwdKOPKa4Qo7d5H
```

Figure 4.5 Testing URLs with SQL blind-based payloads for SQL injection

Figure 4.6 shows the process of testing URLs with time-based SQL injection payloads.

If the user wants to detect the website to know whether it has XSS vulnerability or not, the user chooses the option of XSS. And then the system will be detected with XSS payloads. Detecting XSS vulnerability is processed by using XSS vulnerability payloads shown in Figure 4.7.


Let's Start to Test !!!!

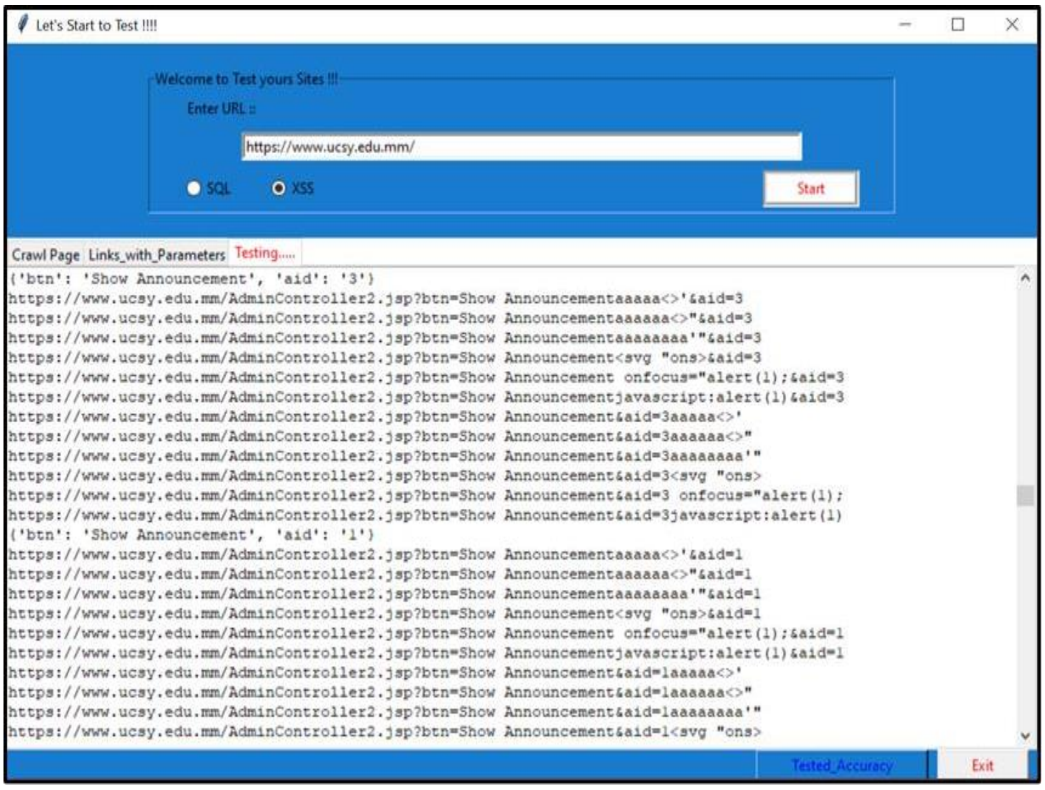
Welcome to Test yours Sites !!!

Enter URL :: https://www.ucsy.edu.mm/

[✓ SQL] XSS

```
https://www.ucsy.edu.mm/discover?filtertype=dateissued&amp;filter_relational_operator=equals&amp;filter=%5B20
02+TO+2009%5D;SELECT SLEEP(2)--
https://www.ucsy.edu.mm/index.jsp?az=0;SELECT SLEEP(2)--
https://www.ucsy.edu.mm/submission.jsp?az=1;SELECT SLEEP(2)--
https://www.ucsy.edu.mm/committee.jsp?az=2;SELECT SLEEP(2)--
https://www.ucsy.edu.mm/login.jsp?az=3;SELECT SLEEP(2)--
https://www.ucsy.edu.mm/register.jsp?az=4;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://moodle.ucsy.edu.mm/course/index.php?categoryid=48&fbclid=IwAR1UrNH14eBtmfEMzcYGI5E0GCXqwdKOPKa4Qo7d5H
jKuflgvcb4C6ZT5OM;SELECT SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT SLEEP(2)--
https://moodle.ucsy.edu.mm/course/index.php?categoryid=48&fbclid=IwAR1UrNH14eBtmfEMzcYGI5E0GCXqwdKOPKa4Qo7d5H
jKuflgvcb4C6ZT5OM;SELECT SLEEP(2)--
```

Figure 4.6 Testing URLs with SQL time-based payloads for SQL injection


Let's Start to Test !!!!

Welcome to Test yours Sites !!!

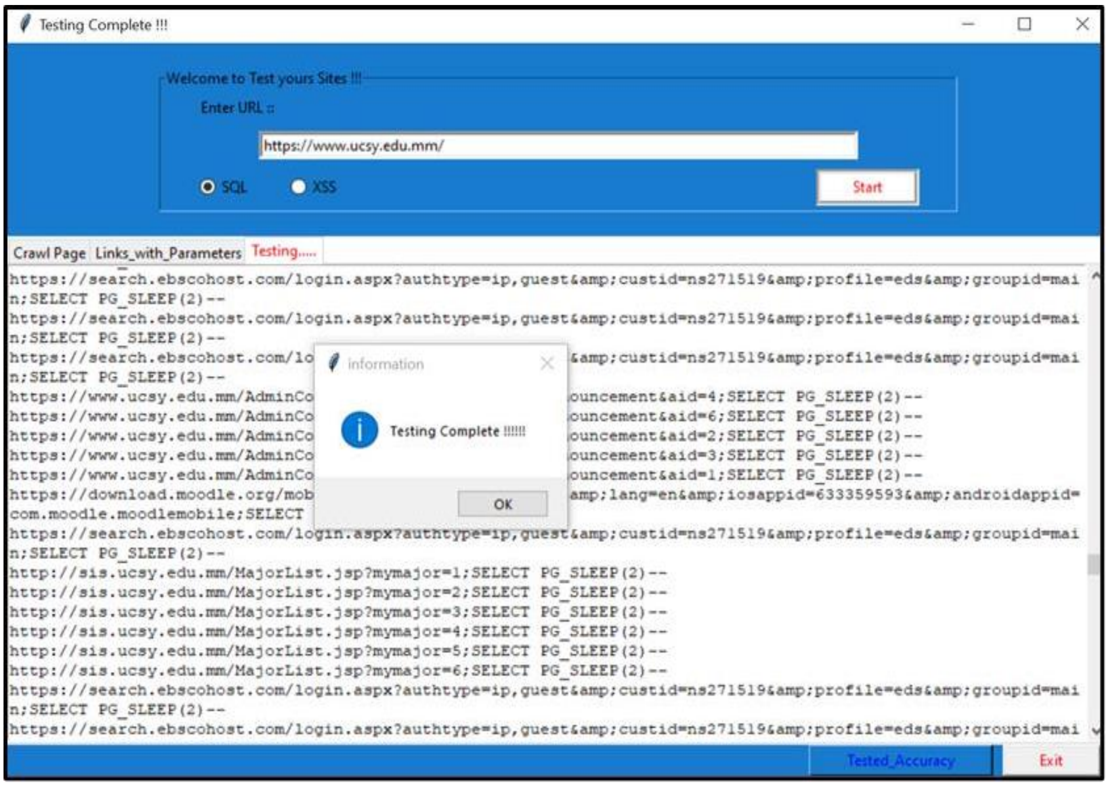
Enter URL :: https://www.ucsy.edu.mm/

SQL [✓ XSS]

```
('btn': 'Show Announcement', 'aid': '3')
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaa<>'&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaaa<>"&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaaaaa'"&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement<svg "ons>&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement onfocus="alert(1);&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementjavascript:alert(1)&aid=3
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3aaaaa<>'
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3aaaaaa<>"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3aaaaaaaa'"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3<svg "ons>
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3 onfocus="alert(1);
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=3javascript:alert(1)
('btn': 'Show Announcement', 'aid': '1')
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaa<>'&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaaa<>"&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementaaaaaaaa'"&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement<svg "ons>&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement onfocus="alert(1);&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcementjavascript:alert(1)&aid=1
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=1aaaaa<>'
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=1aaaaaa<>"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=1aaaaaaaa'"
https://www.ucsy.edu.mm/AdminController2.jsp?btn=Show Announcement&aid=1<svg "ons>
```

Figure 4.7 Testing URLs with XSS payloads for Cross-site Scripting

After detecting process of SQL injection or XSS vulnerability is finished, the tested URLs are shown in the testing panel. If the tested URL is vulnerable, the emphasize message can be seen with the vulnerable URL. And then the information type of message box is appeared when the testing process is finished as shown in Figure 4.8.

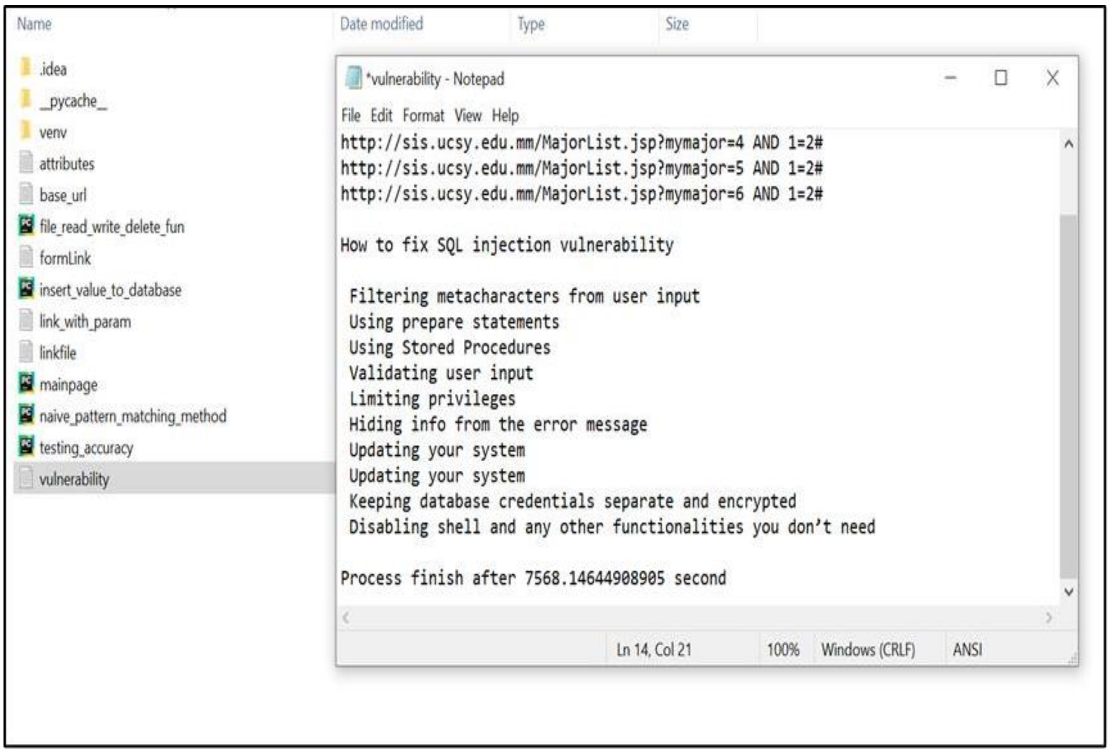


```
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT PG_SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT PG_SLEEP(2)--
https://search.ebscohost.com/lo                                &amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT PG_SLEEP(2)--
https://www.ucsy.edu.mm/AdminCo                                ouncement&aid=4;SELECT PG_SLEEP(2)--
https://www.ucsy.edu.mm/AdminCo                                ouncement&aid=6;SELECT PG_SLEEP(2)--
https://www.ucsy.edu.mm/AdminCo      Testing Complete !!!!!!   ouncement&aid=2;SELECT PG_SLEEP(2)--
https://www.ucsy.edu.mm/AdminCo                                ouncement&aid=3;SELECT PG_SLEEP(2)--
https://www.ucsy.edu.mm/AdminCo                                ouncement&aid=1;SELECT PG_SLEEP(2)--
https://download.moodle.org/mob                                amp;lang=en&amp;iosappid=633359593&amp;androidappid=
com.moodle.moodlemobile;SELECT
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=1;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=2;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=3;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=4;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=5;SELECT PG_SLEEP(2)--
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=6;SELECT PG_SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
n;SELECT PG_SLEEP(2)--
https://search.ebscohost.com/login.aspx?authtype=ip,guest&amp;custid=ns271519&amp;profile=eds&amp;groupid=mai
```

Figure 4.8 Showing information type of message box for completion

When the web application has the vulnerable URL, this URL is then saved into a text file. This file contains vulnerable URL, fixing methods for vulnerability and taking time to finish this process. It was a text file that the user can be seen vulnerable URLs without searching in the testing panel. The example text file is shown in Figure 4.9.

The comparison results of the proposed system and the commercial tools on tested websites can be seen by clicking the button of Tested Accuracy. Calculating the accuracy based on the tested result of the proposed system. Figure 4.10 presents a screenshot showing the stored comparison result and accuracy calculation result.



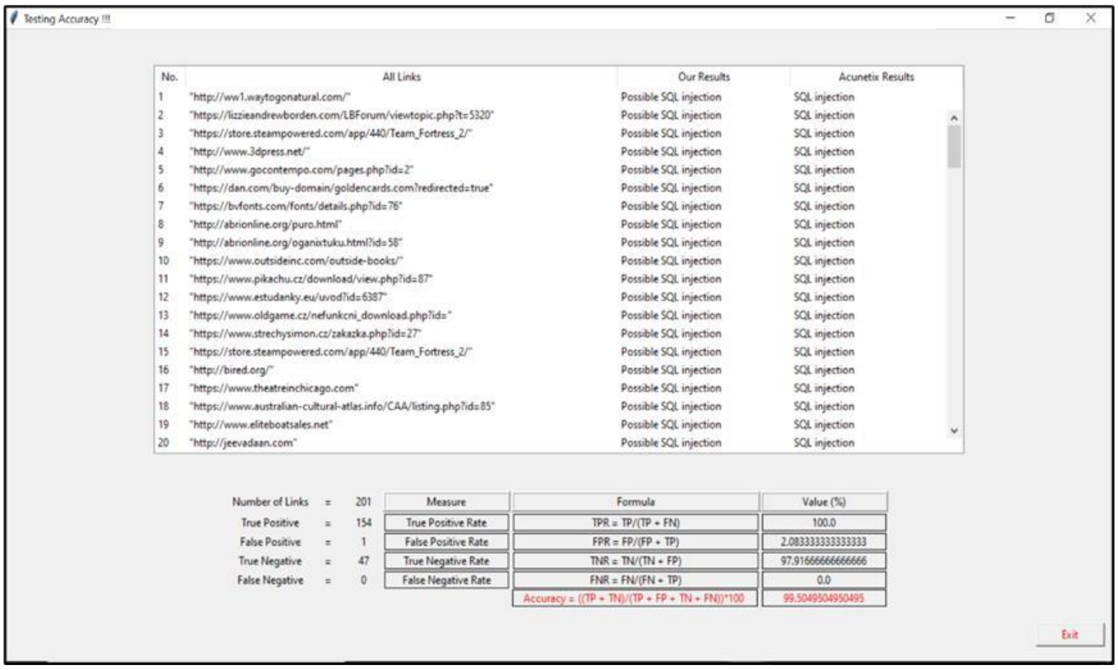
```
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=4 AND 1=2#
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=5 AND 1=2#
http://sis.ucsy.edu.mm/MajorList.jsp?mymajor=6 AND 1=2#

How to fix SQL injection vulnerability

 Filtering metacharacters from user input
 Using prepare statements
 Using Stored Procedures
 Validating user input
 Limiting privileges
 Hiding info from the error message
 Updating your system
 Updating your system
 Keeping database credentials separate and encrypted
 Disabling shell and any other functionalities you don’t need

Process finish after 7568.14644908905 second
```

Figure 4.9 Saving as a text file for vulnerable URLs in case of blind based SQL injection



No.	All Links	Our Results	Acunetix Results
1	"http://ww1.waytogonatural.com/"	Possible SQL injection	SQL injection
2	"https://lizzieandrewborden.com/LBForum/viewtopic.php?t=5320"	Possible SQL injection	SQL injection
3	"https://store.steampowered.com/app/440/Team_Fortress_2/"	Possible SQL injection	SQL injection
4	"http://www.3dpress.net/"	Possible SQL injection	SQL injection
5	"http://www.gocontempo.com/pages.php?id=2"	Possible SQL injection	SQL injection
6	"https://dan.com/buy-domain/goldencards.com?redirected=true"	Possible SQL injection	SQL injection
7	"https://bvfonts.com/fonts/details.php?id=76"	Possible SQL injection	SQL injection
8	"http://abrionline.org/puro.html"	Possible SQL injection	SQL injection
9	"http://abrionline.org/oganixtuku.html?id=58"	Possible SQL injection	SQL injection
10	"https://www.outsideinc.com/outside-books/"	Possible SQL injection	SQL injection
11	"https://www.pikachu.cz/download/view.php?id=87"	Possible SQL injection	SQL injection
12	"https://www.estudanky.eu/uvod?id=6387"	Possible SQL injection	SQL injection
13	"https://www.oldgame.cz/nefunkcni_download.php?id="	Possible SQL injection	SQL injection
14	"https://www.strechysimon.cz/zakazka.php?id=27"	Possible SQL injection	SQL injection
15	"https://store.steampowered.com/app/440/Team_Fortress_2/"	Possible SQL injection	SQL injection
16	"http://bired.org/"	Possible SQL injection	SQL injection
17	"https://www.theatreinchicago.com"	Possible SQL injection	SQL injection
18	"https://www.australian-cultural-atlas.info/CAA/listing.php?id=85"	Possible SQL injection	SQL injection
19	"http://www.eliteboatsales.net"	Possible SQL injection	SQL injection
20	"http://jeevadaan.com"	Possible SQL injection	SQL injection

			Measure	Formula	Value (%)
Number of Links	=	201			
True Positive	=	154	True Positive Rate	TPR = TP/(TP + FN)	100.0
False Positive	=	1	False Positive Rate	FPR = FP/(FP + TP)	2.083333333333333
True Negative	=	47	True Negative Rate	TNR = TN/(TN + FP)	97.91666666666666
False Negative	=	0	False Negative Rate	FNR = FN/(FN + TP)	0.0
				Accuracy = ((TP + TN)/(TP + FP + TN + FN))*100	99.5049504950495

Figure 4.10 Calculating the accuracy of the proposed system

4.3 Experiment Results

The analysis includes stored URLs, the well-known tool and the proposed system that was examined with the corresponding type of vulnerabilities. Through examining, the well-known tool, Acunetix was chosen to quantify the accuracy of the proposed system. Acunetix tool and the proposed system evaluated the vulnerabilities by using these stored URLs.

After the evaluation process is finished, both of these two methods had 154 URLs in true positive, 47 URLs in true negative and 0 URL false negative. In false positive, the Acunetix has 0 URL and 1 URL in the proposed system. The proposed system and Acunetix were nearly equal to the comparing results of true positive, false positive, true negative and false negative rates. Table 4.1 shows the percentage of the accuracy of the proposed system.

Table 4.1 Accuracy results for Proposed System

Measure	Value (%)
True Positive Rate	100%
False Positive Rate	2.0833%
True Negative Rate	97.9167%
False Negative Rate	0%
Accuracy	99.5050%

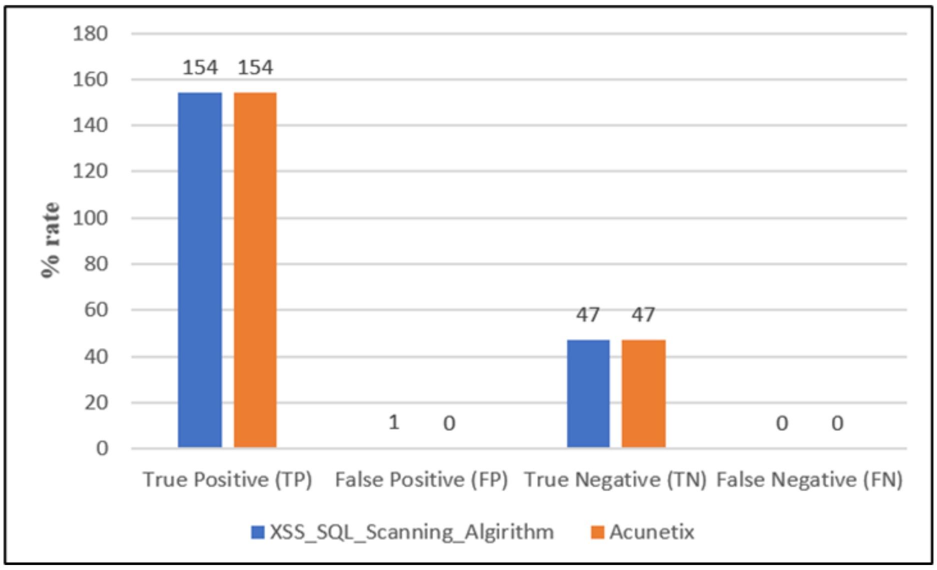


Figure 4.11 Comparison results of proposed system and Acunetix tool

Figure 4.11 shows the accuracy comparison of Acunetix and the proposed system. This study was based on 201 URLs for both methods. Out of 201 URLs, the XSS and SQL vulnerabilities have 85 URLs in commercial domains (.com), 19 URLs in non-profit top-level domain (.org), 7 URLs in Japanese domain, and 45 URLs in other domains. The proposed system shows that most of the commercial domains are normally affected to vulnerabilities.

The memory usage of the proposed system and Acunetix is described in the following table.

Table 4.2 Memory usage of proposed system and Acunetix

	Proposed System	**Acunetix**
Memory (RAM)	23.3 MB	89.9 MB

Table 4.2 displays that the proposed system took 23.3 MB in memory usage and Acunetix used 89.9 MB. Acunetix needed minimum of 2 GB system memory (RAM). The memory consumption of proposed system was less than Acunetix. The proposed system saved 66.6 MB and helped to decrease the memory usage.

The processing time was caused by the crawled URL result and different input points of injection in each crawled web page. Furthermore, after all tests completed, the result showed that this proposed system did not take too much time on crawling web pages and scanning vulnerabilities for both SQL injection and Cross-site scripting. For example, the system generally took 15 seconds to crawl on URL that includes 335 linked URL and the time for detection process was less than 200 seconds.

CHAPTER 5
CONCLUSION

In this thesis, the detection of web applications vulnerability is focused on particularly in two main types of attack, namely SQL injection and Cross-site Scripting (XSS). This thesis presents the proposed algorithm, XSS_SQL_Scanning Algorithm to detect all pages on the websites that have possible vulnerabilities for SQL and XSS attacks. The experimental results are compared with the proposed system and commercial tool. The system is implemented using python programming language on the window platform. In this chapter, the summary of the main conclusion and advantages, limitations, and further extensions are suggested.

The proposed system uses URLs as input for detecting web application. It has the option to either detect SQL or XSS vulnerability. It supports the detailed report, which includes the vulnerable URLs, the list of fixing vulnerability and the processing time of the detection website. It also applies the Naïve algorithm to search payloads in response, which makes to be more lightweight the proposed system.

The experimental results of the proposed algorithm produce comparable result with the commercial tool, Acunetix. Additionally, the memory usage of the proposed algorithm is lower than Acunetix. The users can customize the type of vulnerability, which is either SQL or XSS. The proposed algorithm uplifts the accuracy by reducing the rate of false positive and false negative when testing the web application vulnerability which makes it effective and reliable.

In conclusion, the proposed XSS_SQL_Scanning Algorithm is customizable, applicable, reliable and lightweight. And it helps web developers to secure their web applications from being attacked and also though who start to learn security by showing step by step of detection stages.

5.1 Limitation and Further Extension

In this study, XSS_SQL_Scanning algorithm does not consider other web application vulnerabilities such as buffer overflow, cross-site request forgery, command injection, and so on. The future work will be dedicated to testing with all vulnerabilities in websites and retaining the low false positive and false negative rate.

AUTHOR’S PUBLICATION

[1] Thinzar Aung, Zin Thu Thu Myint, “*Effective Web Application Vulnerability Testing System Using Proposed XSS_SQL_Scanning_Algorithm*”, to be published in the Proceedings of the 19th International Conference on Computer Application (ICCA 2021), Yangon, Myanmar, 2021.

REFERENCES

[1] Ain Zubaidah Mohd Saleha, a*, Nur Amizah Rozalia, b*, Alya Geogiana Bujaa, b, c*, Kamarularifin Abd. Jalila, Fakariah Hani Mohd Alia, Teh Faradilla Abdul Rahmana, c, A method for web application vulnerabilities detection by using Boyer-Moore string matching algorithm, pp 112-121, Information Systems International Conference (ISICO2015), Universiti Teknologi MARA, Shah Alam 40000, Selangor, 2015.

[2] Invicti, Acunetix Web Security Blog, Date of access: April, 2022. https://www.acunetix.com/blog/

[3] Justin Clarke, SQL injection and database errors, Date of access: January, 2021. https://www.sqlinjection.net/errors/

[4] José Fonseca, Marco Vieira, Henrique Madeira, Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks, CISUC - Polithecnic Institute of Guarda and DEI/CISUC - University of Coimbra, Portugal, 2009.

[5] Kamran Mahmoudi, Pattern mattching algorithms, pp 1-56, Imam Khomeini International University, April 2017.

[6] M.S. Jasmine, Kirthiga Devi, Geogen George, Detecting XSS based web application vulnerabilities, International Journal of Computer Technology & Applications, Vol8(2),291-297, pp 291-297 Mälardalen University, M. Tech (ISCF). Student, Department of Information Technology SRM University, TamilNadu, India, ISSN:2229-6093, March 2017.

[7] Priti Singh, Kirthika Thevar, Pooja Shetty, Bushra Shaikh, "Detection of SQL Injection and XSS Vulnerability in Web Application", International Journal of Engineering and Applied Sciences (IJEAS) ISSN: 2394-3661, Volume-2, Issue-3, pp 16-21 March 2015.

[8] Prakhar Tripathi and Rahul Thingla, Cross site scripting (XSS) and SQL-injection attack detection in web application, pp 1037-1041, International Conference on Sustainable Computing in Science, Technology & Management (SUSCOM-2019), RajAvinash Kumar Singh and Sangita Roy, asthan, Jaipur, India, February 26 - 28, 2019.

[9] Ricardo Kano Aguilar, Vulnerable sites, Date of access: January 2021. https://www.scribd.com/doc/153033099/7000-vulnerable-sites-1-2-txt

[10] Simon Wahstrom, Evaluation of string searching algorithms, Mälardalen University, Västerås, Sweden, 2012.

[11] Samira Mehrnoosh, Behrooz Shahi Sheykhahmadloo, Abdolkhalegh khandouzi ghenare, "SQL injection and vulnerability detection", (IJCSIS)

International Journal of Computer Science and Information Security, Vol. 11, No. 5, pp 55-58, May 2013.

[12] Teh Faradilla Abdul Rahman*, Alya Geogiana Buja, Kamarularifin Abd. Jalil, Fakariah Mohd Ali, SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm, pp 183 -189, Journal of Computers, Department of Computer, Technology and Network, Universiti Teknologi MARA, Malaysia, December 26, 2015.

[13] Thomas Hamilton, Security Testing, Date of access: February 2019. https://www.guru99.com/what-is-security-testing.html

[14] Teamques10, Different types of vulnerability, Date of access: March 2019. https://www.ques10.com/p/3550/what-are-the-different-types-of-vulnerability-thre/?

[15] Top Ten Vulnerabilities in OWASP, Date of access: January 2018. https://owasp.org/www-project-top-ten/

[16] Wesley Chai, Confidentiality, Integrity and Availability, Date of access: June 2021. https://www.techtarget.com/whatis/definition/Confidentiality-integrity-and-availability-CIA#