Effective Web Application Vulnerability Testing System Using Proposed XSS_SQL_Scanning_Algorithm

Thinzar Aung
University of Computer Studies, Yangon
Yangon, Myanmar
thinzaraung@ucsy.edu.mm

Zin Thu Thu Myint
Faculty of Information Science
University of Computer Studies, Yangon
Yangon, Myanmar
zinthuthumyint@ucsy.edu.mm

***Abstract*—Nowadays, many people use the internet for more than one purposes. 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 a 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 paper, our proposed algorithm can also find out vulnerability as these scanners. The proposed algorithm presented in this paper 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 are mostly occur in the web application. Besides, our proposed algorithm 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 algorithm is too short enough to match by this pattern matching algorithm approach. The proposed system does not take too much system memory so that it saves the memory consumption. Finally, we test our proposed system using the well-known scanner and evaluate how accurate the results based on having false negative and false positive rate.**

Keywords—XSS_SQL_Scanning_Algorithm, Naïve String Search Algorithm, Cross-site Scripting (XSS), Structured Query Language (SQL) Injection, web application

I. INTRODUCTION

As a trend of technology, many people are mostly using web applications for developing their businesses, education, communications to accessing anywhere, anytime, and thus it saves time and effort. This is why web applications become an important role in our life. 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 [11].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 [12].

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 a 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 [10].

Thus, we proposed the XSS_SQL_Scanning_Algorithm that assists in vulnerability detection with a low rate of false-negative and false positive. Moreover, a new idea in terms of developing the proposed algorithm about using the Naïve pattern matching algorithm because it supports that our 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 system protects the web application from the strike of attacks by discovering the vulnerability.

The rest of this paper is arranged as follows. Section 2 presents reviews of the related work. Section 3 explains the background of the system such as web application vulnerability including SQL vulnerability and XSS attack and Naïve pattern matching algorithm. The proposed system is demonstrated in section 4 and section 5 discuss the evaluation of the output resulting from the post detection. Section 6 concludes the paper and what the work would be done in the future.

II. RELATED WORK

First, the purpose of administrators aims to 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 attack as well as one of the web application vulnerability scanners.

The first study is that a SQL injection scanner using the Boyer-Moore string matching algorithm is implemented by Teh Faradilla Abdul Rahman et al. [1] Their experiment was the technique of using pattern matching that could detect vulnerability with a high accuracy and also avoided 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.

The other study in which Priti Singh et al. [2] implemented a detection method for SQL injection and Cross-site Scripting attack. The approach method was good for detecting SQL injection vulnerability but some weaknesses in detecting XSS attack 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 found any vulnerability although it should. This system will be perfect if many payloads were used for detection.

Simon Wahstrom [4] admitted paper about evaluation of string searching algorithms. In this paper, he described the weaknesses and strength of the most common algorithms. And he also presented naïve algorithm that was probably used when we expected to find a pattern was relatively short.

We continue to study one of the well-known scanners called Acunetix, a web application security scanner, that can detect over 1200 vulnerabilities on many platforms. The users of Acunetix are from small businesses up to political websites, large enterprises, developers, and more, in addition this scanner is known for its very low false positive rate [3].

III. Background

A. Web Application Vulnerability

A web application vulnerability is a weakness in the application, which can harm related users to use the web application. Many types of vulnerabilities in applications are SQL injection, XSS, LDAP, Xpath, or NoSQL queries, OS commands, XML parsers, and so on. The attackers take advantage 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. In this paper, we emphasize SQL injection and XSS attacks [8].

B. SQL injection attack

This attack is called a server-side code injection attack based on Structured Query Language that exploits the system through the weakness of web application with the predefined SQL command and its query is inserted into URL or input fields of the web browser and then generate this query to take the privileges of the database [9]. It mostly occurs when the developers properly invalidated the user input as a part of the SQL query.

C. Cross-site Scripting (XSS) attack

XSS is different from SQL injection because it is a client-side code injection attack and 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, the victim visits that website and then the malicious script executes within the web application and infects the victim's web browser [5].

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.

D. Naïve 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. There are many kinds of pattern matching algorithms, the most common are Naïve, Boyer-Moore, Knuth-Mooris-Pratt, and Rabin Karp pattern matching algorithms. All of these algorithms have their own advantages and disadvantages [4].

In this paper, we choose the Naïve pattern matching algorithm to search for the specious features in web application. Naïve algorithm is the simplest algorithm and easier to implement and understand among other algorithms. For finding patterns, it compares pattern and text from left to right until a match 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 [6]. It takes running time was O(mn) in the worst case, but searching of ordinary text takes O(m+n) where m is the length of the pattern and n means the text length that is pretty quick.

IV. Overview of the Proposed System

In this section, we present about the proposed XSS_SQL_Scanning_Algorithm to deal with XSS and SQL injection vulnerabilities. The following figure shows an overview of the proposed system.

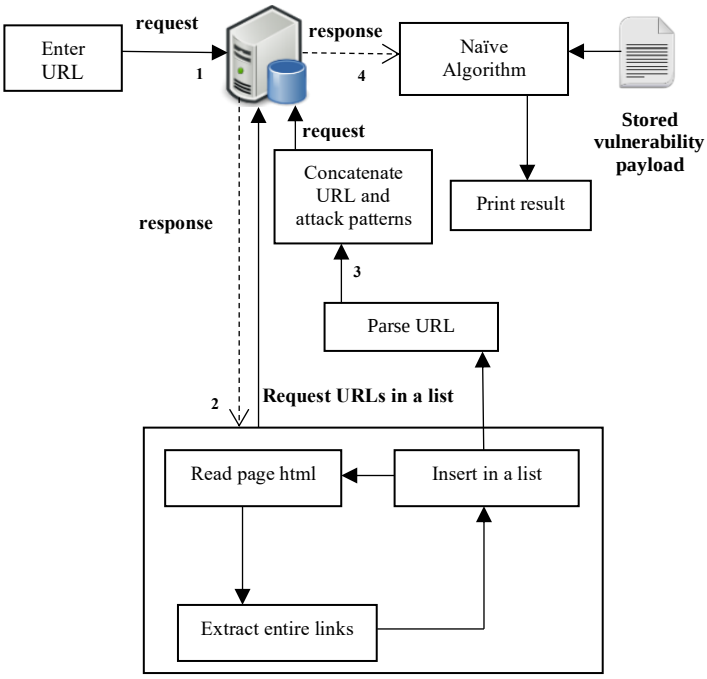

Fig. 1. Architecture of Web Application Vulnerability Testing System Using Proposed XSS_SQL_Scanning_Algorithm

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. Also, the proposed system applies the Naïve pattern matching algorithm in order to discover attack signatures in HTTP response, hence the systems will be simple, quick and accurate for detecting process. Finally finishing detection, the proposed algorithm can be compared with a well-known scanning tool as well as finds the vulnerabilities on different sites to be more accurate.

A. Proposed Algorithm

The proposed XSS_SQL_Scanning_Algorithm, which discovers Cross-site Scripting and SQL injection vulnerabilities that present in Algorithm 1 and then the detailed explanation is described in this subsection.

Algorithm 1 XSS_SQL_Scanning_Algorithm

```
1: Input    : url of the required website, payload bases on
              the previous existed attack patterns associated
              with SQL injection and XSS attack,
              type_of_vulnerability to choose either SQL
              or XSS vulnerability
2: Output   : status whether vulnerability found or not
3: if (type_of_vulnerability = = SQL)
4:      urls ← Call Crawling(url)
5:      result ←
            Forwarding_payload_and_Analyzing_response
            (payload,urls)
6: else if (type_of_vulnerability = = XSS)
7:      urls ← Call Crawling(url)
8:      result ←
            Forwarding_payload_and_Analyzing_response
            (payload,urls)
9: end if
```

1) Scanning to every possible page on web application: Scanning or crawling the entire web application is necessary to achieve all possible links. Firstly, the user inserts the target URL and then requests this URL. The crawling process starts from the response of the requested URL and extracts the respective links by analyzing page html in this response and then these links are appended in a list where a list of links to act recursive crawling until all links are visited. The following proposed algorithm, name as crawling performs the process described in this steps.

Algorithm 2 Crawling Algorithm

```
1: Input    : url of the required website
2: Output   : all possible urls in the website
3: function Crawling (url)
4:      Request with url
5:      while each new urls is not already in the list do
6:          response(rep) ← Get the response of url
7:          page_html ← Read page html in rep
8:          links ← Extract entire links in page_html
9:          list ← Insert links to a list
10:         urls ← Request URLs in the list
11:     end while
12:     return urls
13: end function
```

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.

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 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 id is parameter and value is 1 whereas POST method requests through input elements as search boxes, login boxes, etc. For 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 GET method sends the information by appending the URL with query string while POST method requests the data enclosed in the HTTP message body. And then, we parse the URL to know the position of the input points. Parsing the URL means that URL string splits into its components in which the attributes in URL's components such as scheme, netloc, path, params, query and fragment.

3) Forwarding payloads to detect the web application: After identifying the input points at the previous step, the attack payload is established for generating attack. This payload is based on the previous existed attack patterns associated with SQL injection and XSS attack that are used by attackers in generating SQL injection and XSS attack. The sample payloads of SQL injection are {',",`,\, [AND 1=1-- -, AND 1=2-- -]}etc. and XSS attack's sample payloads are {<>', <>", <svg/onload=alert(1)>, onfocus="alert(1);} etc. Next, we detect the web application by concatenating the payload to each crawled URL and then this modified URL makes request to the web server then the server generates the response. In the next phase, we find out the possible vulnerable patterns inside the response.[7].

4) Analyzing the response: In this phase, we analyze the response by applying the Naïve pattern matching algorithm that uses HTTP response and stored vulnerability payload such as database errors and attack patterns that are already described at the previous phase. The vulnerability payload for database errors are collected each type of database errors in different kind of databases. For instance, "You have an error in your SQL" (MySQL), "ERROR: syntax error" (PostgreSQL)", etc. When any vulnerability is found in the response page, it will show the corresponding URL and the vulnerable input points and also providing a report as a text file that consists of all vulnerable URLs. The following proposed algorithm, name as Forwarding_payload_and_Analyzing_response can take the process of sending payload and analyzing the response. This algorithm is also a combination of phase two, three and four.

Algorithm 3 Forwarding_payload_and_Analyzing_response Algorithm

```
1: Input    : payload bases on the previous existed attack
              patterns associated with SQL injection and
              XSS attack, stored_vulnerability_payload
              such as database errors and attack patterns, url
```

2: **Output** : *status* whether vulnerability exist or not
3: **function** Forwarding_payload_and_Analyzing_response (*payload,url*)
4 Search request method whether GET or POST
5: *input_point* ← Parse *url*
6: *modified_url* ← Concatenate attack patterns with possible URL's *input_point*
7: Make request for the *modified_url*
8: *response(rep)* ← Get the response of the *modified_url*
9: *result(res)* = NAÏVE_ALGORITHM (*rep, stored_vulnerability_payoad*)
10: **if** (*res* = = TRUE)
11: **Display** vulnerability is Found
12: **else**
13: **Display** vulnerability is not Found
14: **end if**
15: **end function**

V. Experimental Result

The proposed algorithm was evaluated in terms of true positive, true negative, false positive and false negative rate. It was based on the correctness of vulnerability detection. Or in other words accuracy indicates that the detected results are nearly similar to the accept or correct value and it also bases on the condition of false positive and false negative. For the evaluation in this system, we used the commercial tool, Acunetix that was compared to the result of proposed method. The contingency table used to evaluate the result of the proposed system is shown in TABLE I.

TABLE I. Specification of True Positive, False Positive, True Negative and False Negative Rate

	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 = $\frac{TN}{TN+FP}$

TP means that the proposed system can manually detect the vulnerability when the website is actually vulnerable, while TN means that the proposed system cannot actually detect the vulnerability when the website is not vulnerable. FP means that the proposed system can detect vulnerability while the website has no vulnerability, and FN means that the proposed system cannot detect vulnerability but the website is actually vulnerable. Figure 2 illustrates the calculated rate results between proposed system and Acunetix based on 121 URLs.

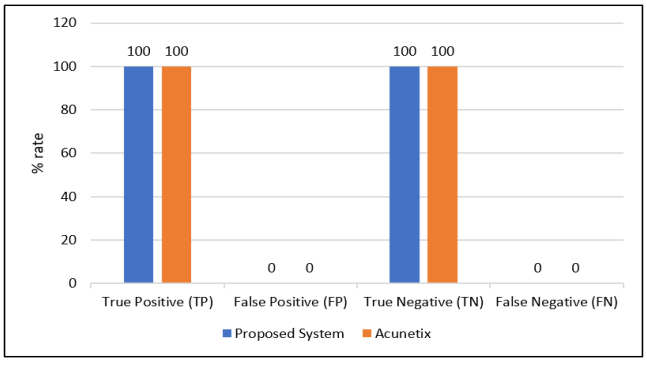

Fig. 2. Comparison result of proposed system and Acunetix

For evaluating these result, we firstly collected 121 URLs of the web applications and examined these URLs with the corresponding type of vulnerabilities and then kept the results in the database. Through examining, the well-known tool, Acunetix was chosen to quantify the accuracy of the proposed system. Both of these two methods had the number of 115 URLs in true positive, 6 URLs in true negative and zero URL in both false positive and false negative. The proposed system and Acunetix were equal to the comparing results of true positive, false positive, true negative and false negative rate. The accuracy of proposed system was calculated by the following equation and the result is shown in TABLE II.

*Accuracy = ((TPR+TNR)/(TPR+FPR+TNR+FNR)) *100*

TABLE II. Accuracy for Proposed System

Measure	**Value (%)**
True Positive Rate	100
False Positive Rate	0
True Negative Rate	100
False Negative Rate	0
Accuracy	100

TABLE II shows that the percentage of the accuracy of the proposed system was achieving 100% accuracy. This study was based on the 121 URLs for both methods. The processing time for the system was measured in seconds. 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-stie scripting. The system generally took 15 seconds to crawl one URL that includes 335 linked URL and the time for detection process was less than 200 seconds. The memory usage of the proposed system and Acunetix is described in TABLE III.

TABLE III. Memory Usage of Proposed Algorithm and Acunetix

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

TABLE III displays that the proposed algorithm 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.

VI. Conclusion and Future Work

This paper focuses on the detection of web applications vulnerability, particularly in two main types of attack, namely SQL injection and Cross-site Scripting (XSS). This paper 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 proposed system uses URLs (Uniform Resource Locator) as input for detecting web application. It has the option to either detect SQL or XSS vulnerability. It supports the detail report, which includes the vulnerable URLs, the list of fixing vulnerability and the processing time of detection website. It applies the Naïve algorithm to search payloads in response, which makes to be more lightweight the proposed algorithm. The experimental result of the proposed algorithm produces 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 make it effective and reliable. In conclusion, the proposed XSS_SQL_Scanning_Algorithm is customizable, applicable, reliable and lightweight. And it helps the web developers to be secure their web application from being attacked and also though who start to learn the security by showing step by step of detection stages.

In this study, we do 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 test with all vulnerabilities in websites and retain the low false positive and false negative rate.

REFERENCES

[1] Teh Faradilla Abdul Rahman*, Alya Geogiana Buja, Kamarularifin Abd. Jalil, Fakariah Mohd Ali, "SQL injection attack scanner using Boyer-Moore string matching algorithm", Journal of Computers, Department of Computer, Technology and Network, Universiti Teknologi MARA, Malaysia, December 26, 2015.

[2] 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, March 2015.

[3] Acunetix Web Security Blog, [online] available: https://www.acunetix.com/blog/, visit on January 2019.

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

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

[6] Kamran Mahmoudi, "Pattern mattching algorithms", Imam Khomeini International University, April 2017.

[7] SQL injection and database errors, [online] available: https://www.sqlinjection.net/errors/, visit on January 2019.

[8] Prakhar Tripathi and Rahul Thingla, "Cross site scripting (XSS) and SQL-injection attack detection in web application", 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] 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, May 2013.

[10] 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", Information Systems International Conference (ISICO2015), Universiti Teknologi MARA, Shah Alam 40000, Selangor, 2015.

[11] OWASP Top Ten, [online] available: https://owasp.org/www-project-top-ten/, visit on February 2019.

[12] 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.