Detection of Cross-Site Scripting Attacks using Dynamic Analysis and Fuzzy Inference System

Conference Paper · March 2020

DOI: 10.1109/ICMCECS47690.2020.240871

5 authors, including:

Olorunjube Falana
Federal University of Agriculture, Abeokuta
8 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Ife Olalekan Ebo
Mountain Top University Ogun State Nigeria
7 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Oreoluwa Carolyn Oloruntoba-Tinubu
University of Agriculture, Abeokuta
3 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Adejimi Alaba
University of Agriculture, Abeokuta
2 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Detection of Cross-Site Scripting Attacks using Dynamic Analysis and Fuzzy Inference System

Olorunjube James Falana
Department of Computer Science
Federal University of Agriculture
Abeokuta, Nigeria
falanaoj@funaab.edu.ng

Ife Olalekan Ebo
Department of Computer Science and Mathematics
Mountain Top University
Ibafo, Nigeria
ioebo@mtu.edu.ng

Carolyn Oreoluwa Tinubu
Department of Computer Science
Federal University of Agriculture
Abeokuta, Nigeria
tinubuco@funaab.edu.ng

Olusesi Alaba Adejimi
Department of Computer Science
Federal University of Agriculture
Abeokuta, Nigeria
adejimioa@funaab.edu.ng

Andeson Ntuk
Department of Computer Science and Mathematics
Mountain Top University
Ibafo, Nigeria
andexdguy08@gmail.com

2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS) 978-1-7281-3126-9/20/$31.00 ©2020 IEEE 10.1109/ICMCECS47690.2020.240871

***Abstract*- Many prevalent problems of web applications are induced by injected codes, which pose great security threats. Vulnerabilities found in web applications are commonly typically exploited to perpetrate attacks. With cross-site scripting (XSS), attackers can infuse malevolent contents into website pages, in this way gaining access-privileges to sensitive page content of the user such as, session cookies, user's data or credentials and several other information often kept up by the browser on behalf of the users. This paper presents a hybrid mechanism for detecting XSS attacks using Dynamic Analysis and Fuzzy Inference. The approach scans the website for possible points of injection before generating an attack vector launched via an HTTP request to a web application. The analysis of the HTTP response predicts the presence of an attack vector. The detection capability of the system is evaluated using some active world web applications and the results show a high rate of detection**.

Keywords: Cross-Site Scripting (XSS), internet, vulnerability, web application, code injection

I. INTRODUCTION

Recently, the use of the Internet has rapidly increased through the use of sophisticated computers and portable devices coupled with interactive web pages. Records show that in June 2019, a total number of 4.5 billion individuals accessed the internet [2]. This usage will continue to increase as web applications, which are client-server software programs that run on web browser continue to drive various sectors of the economy such as e-commercial, health, banking, academic, entertainment etc.

Web applications are becoming one of the standard platforms for representing data and services released over the World Wide Web. Several vulnerabilities have been found in modern web applications. A high fraction of Internet-based web applications is increasingly becoming vulnerable. Consequently, major security concerns have been raised on web applications and Internet-based services.

Cross-site scripting (XSS) attack is one of the topmost attacks plaguing web applications. With the existence of cross-site scripting, every user of the web is a potential victim to an attack that could lead to various kind of cyber-thefts. Symantec reportedly blocked more than 3.7 million form-jacking attempts in 2018 [24]. With XSS attack, an attacker can execute some malicious scripts on a victim's web browser resulting in consequences such as data compromise, theft of cookies, passwords, credit card numbers etc.

XSS is an application layer attack that injects malicious code into trusted content of vulnerable web applications. The user executes the web application and is served the malicious content, which disguises as part of legitimate code of the web application on the victim's browser. The browser runs the embedded malicious script because of its inability to differentiate between malicious and genuine content [3]. One of the major vulnerabilities in a web application is the lack of validation of input data [4], which permits input data sent back as output without validating or scrutinizing, paving way for injected malicious code

The major cause of XSS attack is the inability of the vulnerable web application to validate and sanitize user inputs before generating output to the victim that requested the page [4]. The vulnerability depends on the failure of the application to check up on its input. XSS attack takes advantage of exact guidelines to gain access to system resources just like a genuine web application with access privilege [5]. The client's browser at that point succumbs to the malicious aims of the attacker, as it cannot separate between the authentic and malicious content conveyed by a similar site [3]. Once the user runs the web application, the affected application serves the malicious code as part of the page and is then executed in the context of the trusted and legitimate web application. At the end of successful execution, the victim is hence open to any type of attack dependent on the attacker.

XSS attacks occur almost daily. Websites such as Twitter, Facebook and Google have already become victims for XSS attacks. Effects of XSS attacks include session

hijacking, account hijacking, Distributed Denial of Services (DDoS) attacks, evasion by worms, disclosure of sensitive information, loss of confidentiality, etc.

In this paper, a framework to process web activities and capture fuzzy boundaries between web activities is proposed for the detection of cross-site scripting attack.

The rest of the paper is organized as follows. Section II describes a cross-scripting attack in details, as well as existing defense mechanisms. Section III presents the architecture for the proposed system. In Section IV, the implementation and results are discussed. The paper concludes in Section V.

II. LITERATURE REVIEW

Recently, websites have become more user-friendly, interactive and dynamic, as these sites no longer make use of static web pages. This enhances more activities to be carried out on web applications leading to injectable flaws that are prone to manipulations. Cross-site scripting (XSS) is one of the injection-based attacks and one of the most dangerous web-application based attacks that arose from the adaptation of dynamic web pages in web browsers.

There are three types of XSS attacks namely Reflected XSS, Stored XSS and Document Object Model–based (DOM) XSS. A Reflected XSS attack is a non-persistent or type I attack where an attacker tricks the victim to click or access a link that contains the malicious code, after which the malicious code is sent back to the user from the trusted context of the vulnerable web application [5]. The vulnerability of the web application of not encoding or sanitizing the input causes the malicious code within the HyperText Mark-up Language (HTML) code to be executed within the trust context of the trusted site. This causes the cookies of the trusted site to be sent to the repository of the hacker's site. These actions make the attacker have access to sensitive information of victims, which can be used to carry out account hijacking [6]. The server, however, does not store the malicious script but bounces the original input from the server to the user, which cannot be traced since the victim deliberately initiated the execution of malicious code [7].

In the case of stored XSS attack or persistent attack, the targeted server stores the input in form of a message to either a database or visited logs and this data becomes part of the server and is not reflected [7, 8]. This attack is difficult to spot, as it does not require any form of social engineering. For instance, in a blog, that accepts comment via a text box and stores the message in the database. If an attacker injects a malicious code like tracking session ID cookie and if the server fails to validate the input, the code is stored on the server and executed, stealing the cookie.

Unlike the other, two types of XSS attacks that exploit the vulnerabilities on the server-side, DOM-based XSS exploits the vulnerabilities at the client's side [3]. The attack executes when JavaScript in the page gets a uniform resource locator (URL) parameter and utilizes this data to compose HTML to the page [9]. The attacker controls the items in the DOM and improperly handles the properties of the page; such attacks are hard to distinguish, as they are not included in the response but part of the DOM of the HTML page [10].

Detection Methods

There are majorly three detection techniques for identifying cross-site scripting attacks. They are static analysis, dynamic analysis and hybrid analysis.

1. Static analysis: focuses on the application's source code where it reviews the source code aimed at finding security flaws. This method does not involve the execution of the web application. An application can be reviewed either manually by inspection or automatically with the use of automated analysis tools [3, 7, 11, 12]. The approach has the benefits of detecting potential vulnerabilities that can prove too expensive, time-consuming and prone to a human error leading to lack of accuracy.

2. Dynamic analysis: implements on the runtime behavior of an application in contrast to static analysis. Hallaraker and Vigna [14] proposed an auditing mechanism to detect malicious JavaScript code by monitoring and logging the JavaScript code execution within the Mozilla Web. These techniques which do not go through source codes are relatively precise in distinguishing vulnerabilities resulting in lower false-positive rates.

3. Hybrid Analysis: combines the mechanism of both static and dynamic analysis to thwart XSS attack. Patil and Patil [15] proposed a client-side automated sanitizer for detecting cross-site scripting attacks based on the hybrid analysis. In [17], a dynamic analysis was also utilized for the identification and evacuation of XSS in web applications.

In [18], a few software-testing techniques such as fault injection, black box testing and web application monitoring were used to prove the existence of vulnerabilities. However, it was unable to provide instant web application protections, and could not detect flaws. In [19], the authors introduced an algorithm called the Boyer-Moore string match algorithm to detect XSS vulnerabilities. It works by looking at the characters of the inputted design with the characters of the page from ideal to left utilizing the two heuristics called bad character shift and good-suffix shift. The main goal of this module was to scan from the right to left, scanning character by character for the inputted pattern. However, took a long time to scan when the length of the URL is long. XSS architecture proposed by [20] to search for assault marks by utilizing channels for the HyperText Transfer Protocol (HTTP) use identification segment to decide if a content tag is available or not. If the content tag is not in the database, the tag is automatically removed. The approach will fail to identify an attack if there are multiple instances in the database.

Krugel and Vigna [21] proposed an anomaly detection of web-based attacks using log file with an HTTP request. Log files were used to learn the behavior of a web page for anomaly detection. JavaScript monitoring, data tainting, code rewriting [22, 23, 24] and intrusion detection have been proposed mitigating XSS attacks. Code-rewriting technique uses applications like BrowserShield and

CoreScript as well as other tools for rewriting codes and executing them according to a security policy alongside monitoring their runtime behavior of JavaScript

.

III. METHODOLOGY

The general architecture of our proposed approach is shown in 1. The proposed system is an intelligent system capable of detecting XSS attack using a hybrid methodology consisting of dynamic analysis and fuzzy logic to detect vulnerabilities in web applications. The system carries out a series of dynamic security analysis check on the web application using attack vectors that are previously recognized by Application Entry Points (AEP). AEP comprises of fields that require filling by the user (i.e. GET and POST parameters, forms with their elements as well as anchor or links with parameters) which are required for generation of the HTTP requests sent to the web application in a testing phase.

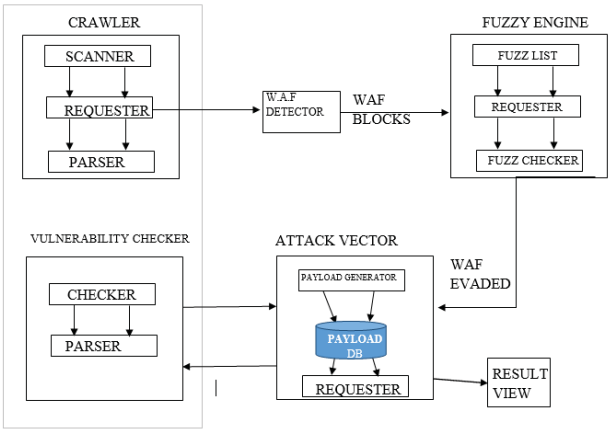

Figure 1: The Proposed Detection System

The detection system is divided into four modules namely: Web Crawler, Vulnerability checker, Attack vector and Fuzzy engine

A. *Web Crawler*

A web crawler also known as crawler or spider is a bot that scans the World Wide Web (WWW) for indexing. The Crawler module scans the web application and collects all the information belonging to the web application. Crawling process starts with the URL and proceeds to the web link tree to collect all the web pages, and this is done by interacting with the web application for gathering AEPs and the Web pages, which are further sent to the parser function. The crawler employs a queue scheduling system to access all input URLs and terminates when the queue is empty and all accessible web pages have been identified and parsed. The crawler-parser function scans through the gathered information and sorts the web pages to extract the AEPs that are further sent to the "Vulnerability Checker" module.

The crawler in the system has been configured to avoid links that will terminate the current session and scan. The crawler carries out three functions:

i. Scanning: The module gathers all the necessary parameters from the URL of the target website. These parameters are used to scan for DOM vulnerability, the result of the scan is transferred to Requesting Module in an encoded form.

ii. Requester: Receives the parameters given by the Scanning Module and replaces the input data with XSS_test data (a non-malicious script to test for vulnerability and receive a response). The receive response is stored in an encoded format which is thereafter converted to a text file and passed to the vulnerability checker.

iii. HTML Parser: The result of Requester is checked against xss_test script by attributes (position, context, and value) through series of searching for script in HTML context, attribute context, and comment and displays the position. Algorithm 1 shows the steps for performing HTML parser.

Algorithm 1: HTML Parser

```
Input: response, xss_test
        If encoding specified
Replace encoded xss_test with original xss_test)
Reflections= response.count(xss_test)
Search (responses)
For each occurrence of xss_test in Reflection
        Collect position;
        Context=" script"
Var position_and_context = []
If (len of position_and_context) < reflection {

        Search for xss_test in attribute context, html context, comment context
If found {
        Add position, context and details to Database[]
Return Database
```

B. *Vulnerability Checker*

A vulnerability checker is a program that scans the website for security issues. Based on the information returned by parsing function in the filter checker and the Inject checker module, the attack vector generator module analyse this information to determine the payload scheme that perfectly fits the attack properly. It scans each occurrence of a reflected string and uses the context information to constructs the malicious scripts to be injected by the Inject function. It also assigns a value of confidence to every allocated set of attack code generated by the Attack vector for each AEP and passes the payload to the checker to determine the payload success. The number of confidence is from range (0-10), the higher the number, the more effective it is. The efficiency value is derived from comparing the injected string and the reflected string in the response and the list are ranked according to efficiency value where greater efficiency is injected first.

C. *Fuzzy Inference Engine*

The Fuzzy Inference is designed to bypass the Web Application Firewall (WAF). The fuzzy module is called when the request is blocked due to the script being recognized by the signatures of the Web Application firewall

1. *The Web Application Firewall Detector*

The Web Application detector sends a noisy malicious string in the data to be requested by the web application to

check if the web applications security would block and deny response, if the string is flagged and blocked, the information is sent to the Fuzzy engine.

2. *The Fuzzy Engine*

The Fuzzy Engine extracts a string from a list of fuzz strings and replaces the string with another to be tested again by the Web Application Firewall Detector. If the string is blocked, the string is returned to the fuzzy engine to be replaced with a less "noisy" string; this module randomly generates a delay before sending a new request with the newly fuzzy generated string until the Firewall is evaded as shown in Algorithm 3. The Fuzzy Engine applies a formula known as the Levenshtein distance to compare and switch strings in the system.

The Levenshtein Distance is a string metric for estimating the distinction between two successions. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions, or substitutions) required to change one word into the other. Mathematically, the Levenshtein distance between two strings, a and b (of length |a| and |b| respectively), is given by lev a, b (|a|,|b|) where:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_{i\neq b_j}} \end{cases} \end{cases}$$

$$if\ \min(i,j)=0, otherwise \qquad (1)$$

Where $1_{(ai\neq bj)}$ is the indicator function equal to 0 when $a_i=b_i$ and equal to 1 otherwise, and $lev_{a,b}(i, j)$ is the distance between the first i characters of a and the first j characters of b. i and j are 1-based indices.

Algorithm 2: Fuzzy Engine

```
Input Fuzzes.
While (Fuzzes is not empty)
        Extract fuzz from Fuzzes
        If encoding {
                Fuzz=encode(fuzz)
                    }
        If delay ==0;
                Delay=0
                Time = delay + Random time
        Replace test string data with fuzz
        Add fuzz to request parameters
        Return response from request
        If encoding {
                Fuzz=encode(fuzz)
                }
        If (fuzz in lower case  in response in lower case){
        Result = "passed"}
        Else{
        Result="blocked"}
        Return result
```

IV. IMPLEMENTATION

The hardware and software requirements of the system are as follows:

Operating System: Kali Linux
RAM: 4GB or greater
Processor Speed: 1.8GHz or greater
Processor: Dual Core or greater
Python version: 3.4

After consideration of various blueprints for the proposed XSS defense strategy, we translated the design and system specifications into implementable programming codes using python programming language and fuzzywuzzy package [1] to run on the terminal. The proposed system was tested live on the internet by scanning a few targeted websites to gather vulnerability information to carry out the scan.

Table 1 shows different sample data used to test our simulated model.

TABLE 1: SAMPLE OF TEST DATA USED

S/N	Web Site
1	www.dramaonline.pk
2	www.mtu.edu.ng
3	www.sherylblas.com
4	www.tabletworld.com
5	www.nichegardens.com

Figure 2 shows the result of scanning and executing an injected payload on www.dramaonline.pk (a movie retails site). The injected payload revealed that parameter 'q' was found and prioritized (sent as request), reflection was found, proving vulnerability in the website. WAF status is offline because there is no firewall protecting the website. The result of the scan proved that the web application was vulnerable to XSS attack.

Figure 3 shows the result of scanning www.mtu.edu.ng an academic website. The result of this scan revealed a potential vulnerability for DOM-based attack due to the presence of an object function found in the web-tree of the website.

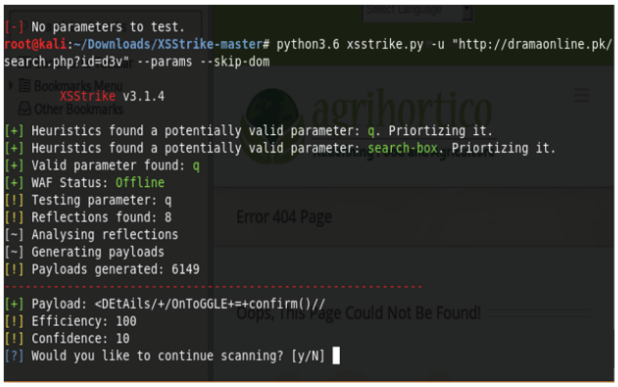

Figure 3: scanning URL on the terminal

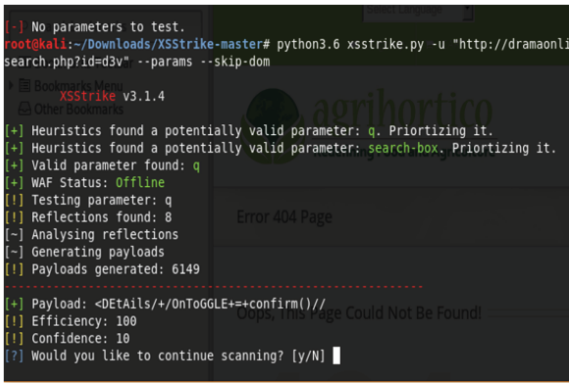

Figure 3: scanning URL on the terminal

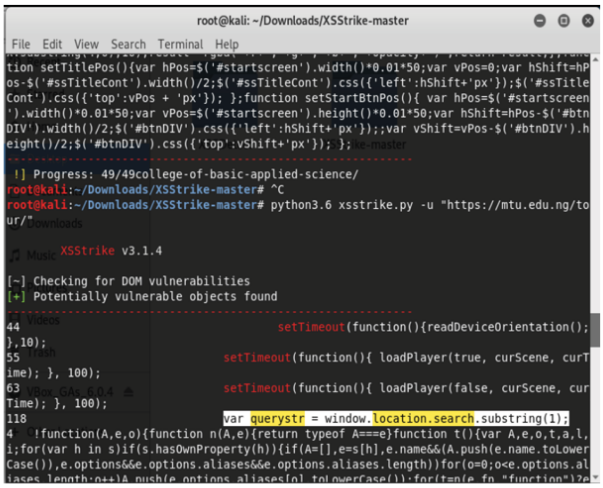

Figure 4: URL Scan for DOM Vulnerability

Figure 3: URL Scan for DOM Vulnerability

Scanning www.sherylblas.com URL for hidden parameters in the website revealed some hidden parameters even though it was not vulnerable to attack as shown in Figure 4. WAF status is offline because there is no firewall protecting the website

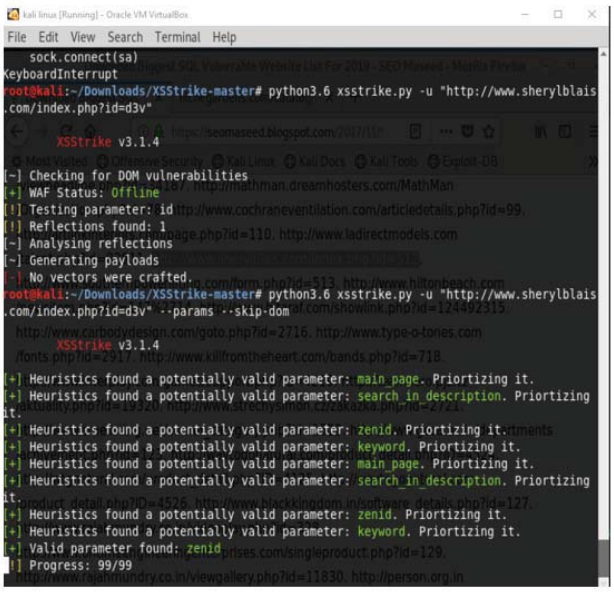

Figure 5: Hidden Parameters check

Figure 4: Hidden Parameters Check

Using fuzz scan to analyze www.tabletworld.com, it showed that the fuzz string could not bypass the website as firewall filtered had blocked all fuzz string requested as shown in Figure 5.

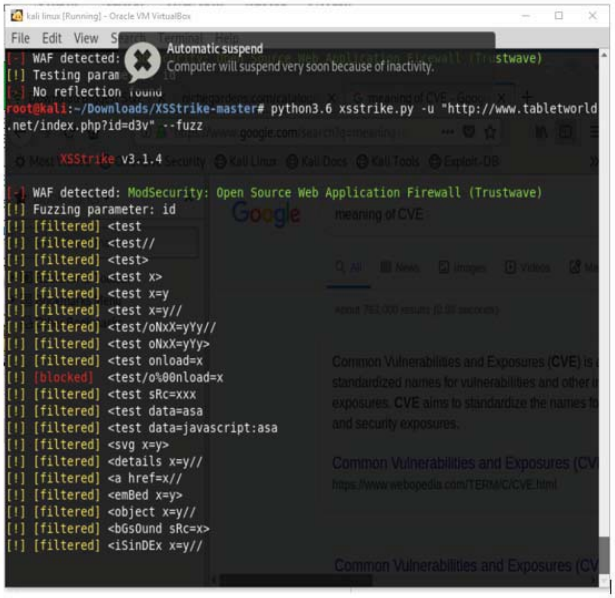

Figure 6: Fuzz check of a WAF protected site

Figure 5: Fast check of a WAF protected site

In a similar experiment in Figure 6, the result showed that fuzz scan on www.nichegardens.com was able to reveal all the strings requested.

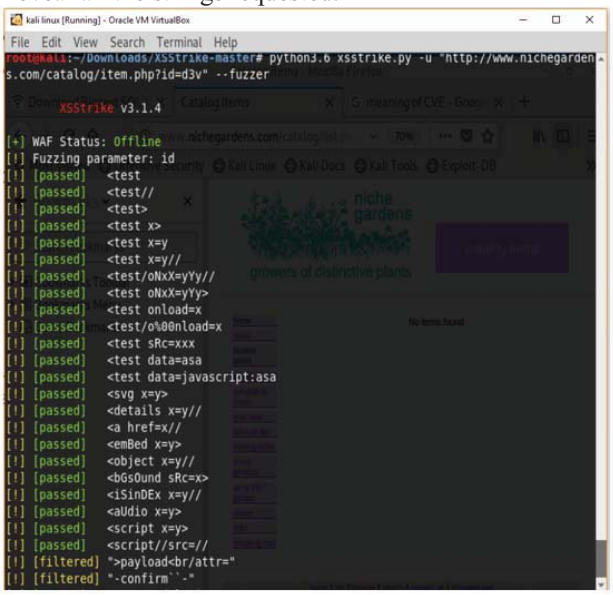

Figure 7: Fuzz check of a WAF unprotected site

Figure 6: Result of Fuzz Scan

Fuzz strings were able to bypass all fuzz strings requested and filtered two strings. This result shows that the website is not protected hence vulnerable to attack.

V. CONCLUSION

Cross-site scripting (XSS) attack has been a major threat in web applications. This attack leverages on vulnerabilities in web applications. Effects of XSS attacks include session hijacking, account hijacking, DDoS attacks, evasion by worms, disclosure of sensitive information, loss of confidentiality, etc. Therefore, in this paper, we presented

a framework for detecting XSS attack. A hybrid system consisting of dynamic analysis and fuzzy techniques was employed. Fuzzy model incorporates Levenshtein distance to compare and switch string in the system The performance of the detection system shows the high accuracy of detecting vulnerabilities in web applications, thus, providing users with a reliable and effective way of mitigating XSS attacks.

REFERENCES

[1] C. Adam. "FuzzyWuzzy" https://pypi.org/project/fuzzywuzzy/ [Accessed: 19-Dec. 2019]

[2] Internet World Stats, "Internet Usage Statistics, the Internet Big Picture, World Internet Users and 2019 Population Stats". [Online]. Available: https://www.internetworldstats.com/stats.htm. [Accessed: 13- Dec-2019]

[3] U. Sarmah, , D. Bhattacharyya, K, J. Kalita, K. "A survey of detection methods for XSS attacks", Journal of Network and Computer Applications, 2018, doi: 10.1016/j.jnca.2018.06.004

[4] B. K. Ayeni, B, J. B Sahalu, and K. R. Adeyanju, "Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System", Journal of Computer Networks and Communications 2018, Vol. 2018, pp. 10.

[5] J. Ruderman, "The same origin policy", 2001.[Online]. Available: http://www.mozilla.org/projects/sec urity/components/same-origin.html [Accessed: 25-Nov.-2019]

[6] J. Garcia-alfaro, G. Navarro-arribas, "Prevention of cross-site scripting attacks on current web applications", OTM, Lecture Notes Comput. Sci., vol. 4804, 2007.

[7] G. R. K Rao, R. S. Prasad, and M. Ramesh, "Neutralizing Cross-Site Scripting Attacks using Open Source Technologies", Proceedings of the Second International Conference on ICT for Competitive Strategies No 24 2016 doi: https://doi.org/10.1145/2905055.2905230.

[8] A. Kieżun, J. G Philip, J. Karthick, and D. E Michael, "Automatic creation of SQL injection and cross-site scripting attacks", In ICSE Proceedings of the 31st International Conference on Software Engineering, (Vancouver, BC, Canada), May 2009, pp. 199-209.

[9] E. Kirda, N. Jovanovic, C. Kruegel and G. Vigna, "Client-side cross-site scripting protection", Computer and Society 28(7), pp. 592-604, 2009, doi: 10.1016/j.cose.2009.04.008

[10] Wang et al C.-H.Wang, Y.-S. Zhou, A New Cross-Site Scripting Detection Mechanism Integrated with HTML5 and CORS Properties by Using Browser Extensions, in: 2016 International Computer Symposium (ICS), IEEE, 2016, pp. 264–269.

[11] P. Bhojak, K. Patel, V. Agrawal and V. Shah, "SQL Injection and XSS Vulnerability Detection in web Application", Int.Journal of Advanced Research in Computer Science and Software Engineering 5 (12) Dec. 2015, pp. 110-115.

[12] G. Lucca , A. Fasolino, M. Mastroianni., and P. Tramontana, "Identifying Cross Site Scripting Vulnerabilities in Web Applications" In Sixth IEEE International Workshop on Web Site Evolution (WSE), pages 71 – 80, 2004.

[13] G. Wassermann, Z. Su, Static Detection of Cross-site Scripting Vulnerabilities, in: Proceedings of the 30th International Conference on Software Engineering, ACM, 2008, pp. 171–180.

[14] O. Hallaraker and G. Vigna, "Detecting malicious JavaScript code in Mozilla ", 10th Proceedings of IEEE International Conference of Eng. of Complex Computer Systems 2005, doi: 10.1109/iceccs.2005.35

[15] D. K. Patil, and K. Patil, "Client-side Automated Sanitizer for Cross-Site Scripting Vulnerabilities", International Journal of Computer Applications 121, 2015

[16] Curtsinger C., Livshits B., ZornB. G., Seifert C, (2011). ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection, in: USENIX Security Symposium, 2011, pp. 33–48.

[17] H. Isatou, S. Abubakr, Z. Hazura, and A. Novia, "An approach for cross site scripting detection and removal based on genetic algorithms," in Proceedings of the Ninth International Conference on Software Engineering Advances: France, pp. 227–232, Nice, France, October 2014.

[18] Huang Y.-W., Yu F.,Hang C., Tsai C.-H., Lee D.-T., Kuo S.-Y., (2004). Securing Web Application Code by Static Analysis and Runtime Protection, in: Proceedings of the 13th International Conference on World Wide Web, ACM, 2004, pp. 40–52.

[19] A. Saleh, B Rozalia, B. Bujaa, A. Kamarularifin, A. Mohd, and A. Faradilla, "A method for web application vulnerabilities detection by using Boyer-Moore string matching algorithm," Information Systems International Conference, vol. 72, no. 3, p. 112, 2015.

[20] M. Koli, S. Pooja, H. K. Pranali, and N. G. PraSthmesh, "SQL injection and XSS vulnerabilities countermeasures in web applications," International Journal on Recent and Innovation Trends in Computing and Communication, vol. 4, no. 4, pp. 692–695, 2016

[21] C. Krugel and G. Vigna, "Anomaly detection of web-based attacks" Proceedings of 10th ACM Conference onComputer and Communications Security, CSC 2003, Washington, DC, USA, October.

[22] H. Oystein and V. Giovanni, "Detecting malicious javascript code in mozilla", in ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, pages 85–94, Washington, DC, USA, IEEE Computer Society, 2005.

[23] C. Reis, J. Dunaga, H. Wang, O. Dubrovsky, and S. Esmeir, "Browsershield: Vulnerability-driven filtering of dynamic html", ACM Transactions on the Web, 1(3), 2007.

[23] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis", 14th Annual Network and Distributed System Security Symposium, NDSS, Vol. 2007, p. 12, 2007.

[24] Symantec. (2019). Internet Security Threat Report. 23.