BOOK CHAPTER | Detecting Cross-site Scriptings

Cross Site Scripting Detection in Web Applications Using Fuzzy Inference Systems

Bakare K. Ayeni, Junaidu B. Sahalu & Kolawole R. Adeyanju
Ahmadu Bello University
Zaria – Nigeria
E-mail: bakarre@gmail.com
Phone: 2348057494
ORCID: 0000-0002-3556-3976

Abstract

With improvement in computing and technological advancements, web based applications are now ubiquitous on the internet. However, these web applications are becoming prone to vulnerabilities which have led to theft of confidential information, data loss and denial of data access in the course of information transmission. Cross Site Scripting (XSS) is a form of web security attack which involves the injection of malicious codes into web applications from untrusted sources. Interestingly, recent researches on web application security centre on attacks prevention and mechanisms for secure coding, recent methods for those attacks do not only generate high false positives, they have little considerations for the users who oftentimes are the victims of malicious attacks. Motivated by this problem, this chapter describes an "intelligent" tool for detecting Cross Site Scripting flaws in web applications. The chapter describes the method implemented based on fuzzy logic to detect classic XSS weaknesses and provide some results on experimentations. Our detection framework recorded 15% improvement in accuracy and 0.01% reduction in false positive rate which is considerably lower than existing work by Koli et al (2016). Our approach also serves as a decision making tool for the users.

Keywords: Cross Site Scripting, Fuzzy Inference System, Membership Function, Web Application, Vulnerabilities.

Introduction

Over the past decade, the Internet has witnessed tremendous growth in volume, nature and channel of information exchange across several media irrespective of distance or location. In particular, the internet has become the major channel through which global businesses conduct marketing businesses and have become extremely successful over traditional marketing strategies. Typically, web applications allow the capture, processing, storage and transmission of sensitive customer data (such as personal details, credit card numbers and social security information) for immediate and recurrent use.

BOOK Chapter | Web of Deceit - June 2022 - Creative Research Publishers - Open Access — Distributed Free

Citation Bakare K. A., Junaidu B.S. & Kolawole R.A. (2022). Cross Site Scripting Detection in Web Applications Using Fuzzy Inference Systems SMART-IEEE-ACity-ICTU-CRACC-ICTU-Foundations Series Book Chapter on Web of Deceit - African Multistakeholders' Perspective on Online Safety and Associated Correlates Using Multi-Throng Theoretical, Review, Empirical and Design Approaches. Pp 297 -308. www.isteams.net/bookchapter2022. DOI https://doi.org/ 10.22624/AIMS/BK2022-P49

Therefore, web applications have become major targets of hackers who take advantage of web developers' poor coding practices, weaknesses in application code, inappropriate user input authorization or non-adherence to security standards by the software developers. These vulnerabilities could either be on the server side or more dangerously, the client side. The vulnerabilities include SQL injection, cross site request forgery, information leakage, session hijacking, cross site scripting etc. this paper focuses on cross site scripting attack detection. Malicious injection of code within vulnerable web applications to trick users and redirect them to untrusted web sites is called Cross-Site Scripting (XSS).

XSS may occur even when the servers and database engine contain no vulnerability themselves and it is arguably one of the most predominant web application exposures today (see Figure 1). Many researchers have been directed at addressing problems related to XSS vulnerabilities. Most of the approaches focused on preventing XSS attacks in web applications during software security testing (Sharma, 2012), (Sun and He 2012) (Van and Chen, 2012). Few research activities have addressed its detection (Bathia et. al., 2011), (Kanchan and Harmanpreet, 2014) and (Isatou, et. al.,2014).

Several circumvention mechanisms have been implemented, but none of them are complete or accurate enough to guarantee an absolute level of security on web application due to lack of common and complete methodology for evaluation in terms of performance (Nithya et. al., 2015). An emerging soft computing technique applicable to web security is fuzzy logic. Hossein and Hisham, (2016), Georgios and Sokratis, (2009) and Hossein and Hisham (2016) successfully applied fuzzy rule based approach to web vulnerability and intrusion detection with encouraging results. The advantages that fuzzy logic brings to web security are: first, the development of linguistic variables as classifiers to predict whether or not a script is malicious. Second, the provision of classifiers with predictive capability to detect new malicious scripts. Third, by identifying and extracting DOM based features, interpretability is guaranteed which helps with realistic feedback to users. In addition, fuzzy logic is capable of making real-time decisions even with incomplete information (Animesh and Debasish, 2014). Since fuzzy logic systems can manipulate linguistic rules in a natural way, they are particularly suitable in combining the various DOM parameter and rules to provide optimal results.

In this chapter, we propose a Fuzzy-based approach for the detection of DOM based XSS vulnerabilities in web applications. The contributions of this chapter are as follows:

- ✓ Selection and implementation of DOM based features for XSS detection using OWASP web application security guideline.
- ✓ Application of Fuzzy Logic Inference System to web application vulnerability detection.
- ✓ Implementation of user interface for users to have a verdict on their level of exposure to Cross Site Scripting attack while visiting a web site.

The chapter is organized as follows: Section 2 gives literature review, proposed approach and experimental results are discussed in section 3, in section 4, we discussed the proposed implementation and results while Section 5 concludes the chapter.

2. Literature Review

XSS is a type of computer security vulnerability typically found in web applications that enables malicious code to be injected into client-side script of web pages viewed by other users. It is a threat which occurs when a web application gathers malicious data from users. Cross Site Scripting vulnerabilities are security problems that occur in web applications. According to Isatou, et. al., (2014), XSS attacks are of three types namely reflected, stored and Document Object Model (DOM) based.

Stored (Persistent) XSS Attacks save malicious scripts in databases, message forums, and comment fields of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker.

Reflected (Non-Persistent) XSS Attacks are executed by the victim's browser and occur when the victim provides input to the website. Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.

DOM Based Attacks are found on the client side. Attackers are able to collect sensitive information from the user's computer. Therefore, there exist some web applications on the Internet that are vulnerable to DOM Based XSS without showing features of standard XSS (Isatou, et. al., 2014). DOM based XSS attacks accounted for 60% of cross site scripting attacks in the last couple of years. The reason could be justifiably agreed to because attackers now target underlying sensitive documents and information databases in case of sql injection attacks. According to (http://info.cenzic.com, 2013), Cenzic Application Vulnerability Trends Report (2013), XSS represents 25% of the total attack statistics and considered as topmost web application attack.

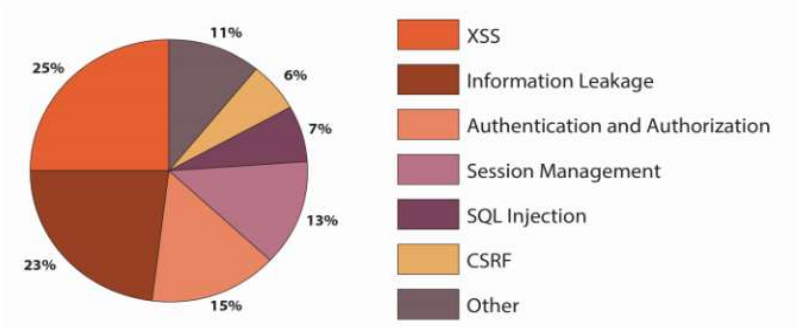

Figure 1: Cenzic Application in Vulnerability Trends Report (2013)
Source: http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vulnerability-Trends-Report-2013.pdf.

2.1 Concept of Fuzzy Logic

Zadeh, (1965) introduced "Fuzzy Sets" and laid out the mathematics of Fuzzy Logic. Fuzzy Logic recognizes not only clear-cut alternatives, but the infinite gradations in between. These numeric values are then used to derive exact solutions to problems. According to Cordon (2001), Fuzzy Inference System (FIS) consists of three components. First, a rule-base which contains a selection of Fuzzy rules. Secondly, a database which defines the membership functions used in the rules and a reasoning mechanism to carry out the Inference procedure on the rules (Mamdani, 1974). Figure 2 gives a standard representation of fuzzy logic procedure.

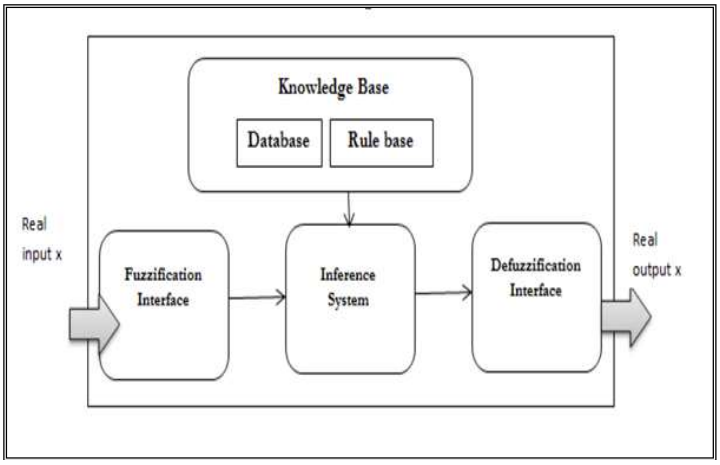

Figure 2: Mamdani Fuzzy Inference System (Cordon, 2001)

Also Alberto et. al., (2015) affirmed that Fuzzy logic is suitable for ambiguous scenarios where there is no certainty about making decisions. Among the several types of Fuzzy Inference Systems, Mamdani Fuzzy Inference System is the most popular, powerful, and widely used methodology in developing Fuzzy models and analyzing data (Animesh and Debasish, 2014).

2.2 Survey On Web Application Vulnerabilities

There are basically three kinds of security vulnerabilities. The categorisation is primarily a function of the possible existence of flaws at each application level. A study by (Vandana et. al., 2014) categorised web application vulnerabilities as follows:

i.) Input validation vulnerability (client side request level).
ii.) Session management vulnerability (session level).
iii.) Application logic vulnerability (whole application).

2.3 Related Works

Current research works try to add intelligence to increase the quality of detection. It can be knowledge on current flaws in browsers/applications (using for example vulnerability bases. It can also be machine learning on the application to find how outputs depend on inputs. A few of these approaches are detailed below.

A novel system for detecting XSS vulnerability was designed based on model Inference and evolutionary fuzzing (Duchene et. al., 2012). The approach simply used a heuristic driven substring matching algorithm to develop a crawler. A solution that uses a Genetic Algorithm-based approach in the detection and removal of XSS in Web applications using three components was designed by Isatou, et. al., (2014). Reviewing a solution proposed by Saleh, et. al., (2015), it was a technique by developing a detection method using Boyer-Moore String Matching Algorithm. The work proposed by Abbass and Nasser, (2016) suggested an algorithm named NUIVT (Novel User Input Validation Test). The algorithm includes three steps to detect the vulnerability but results of each scanning are saved in order to raise the intelligence of the system which leads to repetition of the comparisons, tedious and time consuming.

An SQL and XSS architecture was proposed by Koli et. al., 2016). Major drawback in their research was that if the attack pattern is not stored in their database, then the tool cannot detect the attack successfully. It can be deduced from literature that approaches to detection of XSS using genetic algorithms have not been too effective. The approach using Boyer Moore algorithm was not scalable. Another drawback of existing approaches is considering all types of vulnerabilities equally, and their severity level equally. Therefore, this research intends to add another dimension to detection process with the hope of improving efficiency and reliability.

We are motivated by the application of Fuzzy Logic to detection of web security issues by Mankad, (2014) and phishing website detection by (Alberto et. al., 2015). They applied fuzzy inference system to assess risk due to code injection vulnerabilities based on a set of linguistic terms for vulnerability and severity levels. Some researchers like (Hossein and Hisham, 2016) and (Hossain and Hisham 2014) have used a fuzzy logic based system to assess risk due to different types of code injection vulnerabilities but with some limktations. This chapter is motivated by the observation that XSS vulnerability detection can be modeled in form of Fuzzy Inference System. Also, other approaches in literature do not have the capability to estimate the overall risk due to diverse severity level for a given vulnerability as also corroborated by (Shar and Tan, 2012), and (Mankad, 2014) [21]. Thus, a suitable framework to detect XSS by introducing decision making inference system is hereby introduced.

3. Proposed Approach

XSS is one of the most exploited weaknesses in web application, and one of the most studied. Full protection is hopeless, as any full protection against any programming errors or bugs. Good programming practices, intelligence in libraries and browsers have been developed to protect against XSS (Ali, 2014). There are also a lot of proposed tool to detect XSS risks.In this section, we present our approach for detecting XSS in web applications. We also present the fuzzy inference procedure applied on the detection phase.

3.1. Detecting XSS Attacks

We prepared a background to identify any XSS or redirection vulnerabilities that could be initiated by using a maliciously crafted URL to introduce mischievous data into the DOM of inputted web pages (both static and dynamically generated). If the data (or a manipulated form of it) is passed to one of the following Application Programming Interfaces (APIs), the application may be vulnerable to XSS. We Identify all uses of the APIs which may be used to access DOM data that can be controlled through crafted Uniform Resource Locators(URL). As enlisted by Krishnaveni and Sathiyakumari, (2013), these seven sources through which XSS vulnerabilities could be introduced to web applications are HTTP Referrer Head, Window Location Test, The Document Referrer, The Document Location, Document URL unencoded, Cookies and the Headers . Their view was supported by the OWASP report of 2012, in which the lists of locations where XSS are more prone are summarized.

X1 = document. Location
X2 = document. referrer
X3 =document. location. href
X4 = window. location
X5 = document. cookie
X6 = document. URLUnencoded
X7 = location. Header.

The headers are considered as possible entry points for input-based attacks.

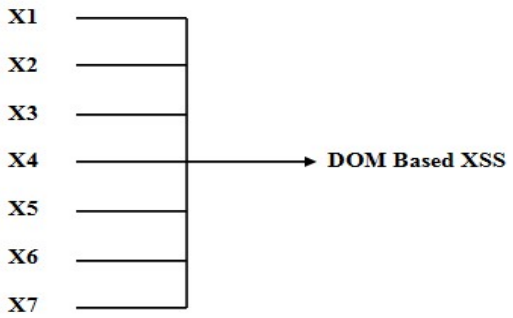

Figure 3: Locations vulnerable to DOM Based XSS Attacks.

Many important categories of vulnerabilities are triggered by unexpected user input and can appear anywhere within the application. Any XSS or redirection vulnerabilities are identified by the injection module and detected where a crafted URL is to introduce malicious data into the DOM of the relevant page. In Algorithm 1, each of the webpages on a URL is inspected for either an unconventional or unsafe use of the APIs in Figure 3. This is achieved by visiting every node in the web page. Each node is subjected to the various DOM tests as outlined in the algorithm. The algorithm returns a vulnerability summary of the content of each node visited. This summary gives an indication of the possibility of XSS in web application. The developed system scans websites recursively, building an internal representation of the site in a tree-like data structure called path state nodes. These path state nodes can be directories, files, or files with POST or GET parameters. This is because in addition to analyzing the page content, the crawling engine does several tests on each potential path, trying to determine if it is a file or a directory.

```
1.  Input web application
2.  Output XSS Vulnerabilities
    // "DOM testing module",
3.  dom DOM = parse ( web application)
4.  node = [ ];
5.  for (var i = 0; i < arguments.length; i++) {
6.          element.push(doc.getElementById(arguments[i]));
7.  return node element;

8.  doc = new HTMLDocument(testDom);
9.  jQuery.setDocument(doc);
    //("Running dom test");
10. var all = jQuery("*"), good = true;
11. for(var i = 0; i < all.length; i++)
12. for (all [i].nodeType) {
13.         run() {
14.                 basic_tests();
15.                 id_test();
16.                 class_tests();
17.                 name_tests();
18.                 window_location_tests();
19.                 header_tests();
20.                 referer_attributes_tests();
21.                 location_header_tests();
22.                 url_encoded_tests();
23.                 document_location_tests();
24.                 pseudo_form_tests();
25.         }
26. return vulns summary();
```

Algorithm 1: Algorithm for Detecting DOM Based XSS

3.2. System Architecture

The input to the detection system shall be obtained by extracting suspected malicious features from web application pages. Then, we develop a script code which will connect to the URL entered and output the attributes associated with the elements which is then forwarded to the Fuzzy Inference System to identify possible vulnerability occurrence. Figure 4 presents the system architecture.

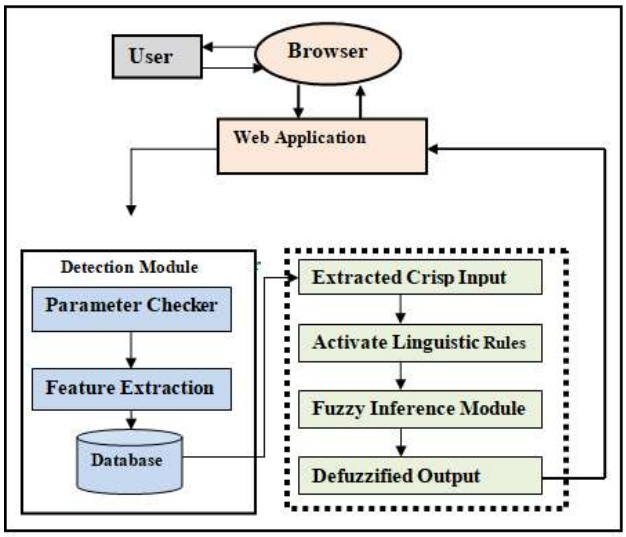

Figure 4: System Architecture

Defining linguistic variables and terms

We consider each of the parameters defined in Figure 3 as crisp input for Fuzzy Inference System. Each of the crisp inputs is mapped to three different linguistic terms (Fuzzy variables): Low, Medium, and High. Table 1 shows the crisp input characteristics (X1-X7) and the corresponding linguistic variables (Low, Medium, High).

Table 1: Crisp input and linguistic variables

Crisp Input	Linguistic Variables
document. Location (X1)	Low, Medium, High
document. Referrer (X2)	
document. location.Href (X3)	
window. Location (X4)	
document. Cookie (X5)	
document. URLUnencoded (X6)	
location. Header (X7)	

Assignment of Membership Functions

We define membership functions for each of the linguistic variables as follows. A membership function converts a crisp input value to another value that ranges between 0 and 1. Fuzzy sets can have a variety of shapes. However, a triangle or a trapezoid shaped membership representation often provides a suitable representation. To define membership function for each of linguistic variable, we apply Triangular Membership Function (TMF) for easy and clear representation. The membership function was obtained by dividing the input space into equal partitions in a triangular format as in Table 2 - three rules each (High, Low, Medium).

Table 2: Linguistic variables and membership function.

Linguistic term	Triangular Fuzzy number
High	(0.00, 0.25, 0.50)
Medium	(0.25, 0.50, 0.75)
Low	(0.50, 0.75, 1.00)

3.3 Design of the Rule Engine

A Fuzzy rule has two parts: the antecedent and the precedent parts. The antecedents are joined together by logical operators. The AND logical operator, being the more popular and frequently used operator, combines the predicate parameters based on the number of rules for design of the fuzzy system and generates the consequence which is used to determine the output. We created twenty-one (21) rules for our Fuzzy Inference using Wang and Mendel's technique. Using this method, AND operator results in maximum of truth (membership) values. It was proposed by Wang and Mendel, (1992) using the Mamdani model.

3.4 Aggregation of rules and defuzzification

The combined result from all crisp inputs are combined and defuzzified to obtain a Fuzzy output value. Defuzzification is the process of converting the degree of membership of output linguistic variables into crisp or fuzzy values. There are various defuzzification strategies. The centre of area method considers the full area under the scaled membership function even if this area extends beyond the range of the output variables. Due to its extended range coverage, we apply the centre of area approach in our defuzzification process. Other possibilities include centre of gravity approach and the maximum or minimum values.

3.5 Programming Implementation

The development was implemented with the Eclipse IDE using the Java Programming Language. JQuery web interaction interface was integrated into the Eclipse IDE as a library for easy usage. Java programming language was used to develop the Fuzzy Inference System and integrated with the Eclipse IDE for optimised performance. The entire Vega application was built in Java with the crawling component written in JavaScript. Vega XSS Detection module was written in JQuery scripting language. The screen shot of scanner progress is shown in Figure 5.

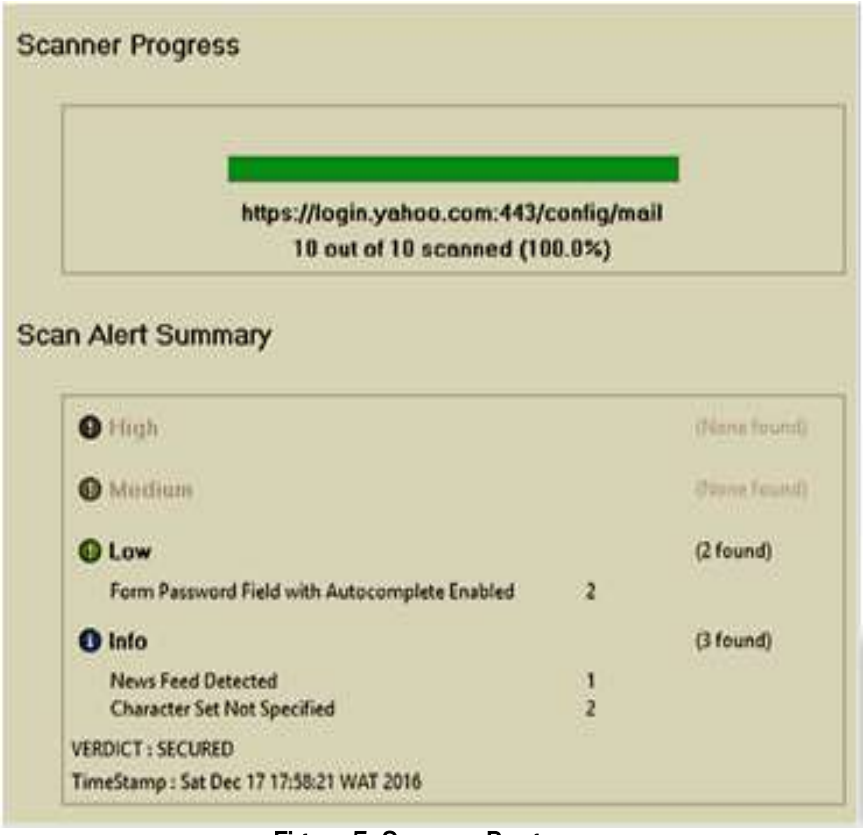

Figure 5: Scanner Progress

3.5. Performance Analysis

In the course of implementation, the following performance metrics were studied:

- Capability to detect vulnerabilities
- Accuracy.
- False positive rate.

Comparison with work by Koli et. al., (2016) using measure (i), (ii) & (iii) based on the number of web pages to be used for evaluation.

A. Accuracy

This is a measure of the degree to which the results of the test scanning conducted on the developed framework conforms to the correct values or the standard data set. Accuracy is calculated using equation 1.

$$A(M) = \frac{TNC + TPC}{(TNC + FPC + FNC + TPC)} \quad (1)$$

Where
TNC= number of true negative cases
FPC= number of false positive cases
FNC= number of false negative cases
TPC= number of true positive cases

B. False Positive Rate:

In performing multiple comparisons, the term false positive ratio is calculated as the ratio between the number of negative events wrongly categorized as positive and the total number of actual negative events. We chose http://yahoomail.com as the test website for this experiment due to its popularity which also makes it a popular target for XSS attacks as well. The false positive rate is given by equation 2.

$FPR = FP/(FP+TN)$ n (2)

Where FP is number of false positives.
TN is number of true negatives.

C. Number of vulnerabilities detected

This is a minimum standard for checking the developed application's capabilities to discover XSS vulnerabilities in applications that are known to be vulnerable.
Table 3 shows the results for the number of vulnerabilities detected.

Table 3: Summary of Performance Metrics

Parameter	NetSparker	Acunetix	WebCruiser	Koli et. al. (2016)	CrawlerXSS
Vulnerability Detection	16	18	15	21	21
Accuracy	55	60	40	80	95
False Positive	3	12	2	1	0.99

The results show that Crawler XSS was able to detect XSS vulnerabilities in all the web sites visited with the results being the same with Koli et. al. (2016). Figure 6 shows the number of sequences observed per web site over a period of 100 visits.

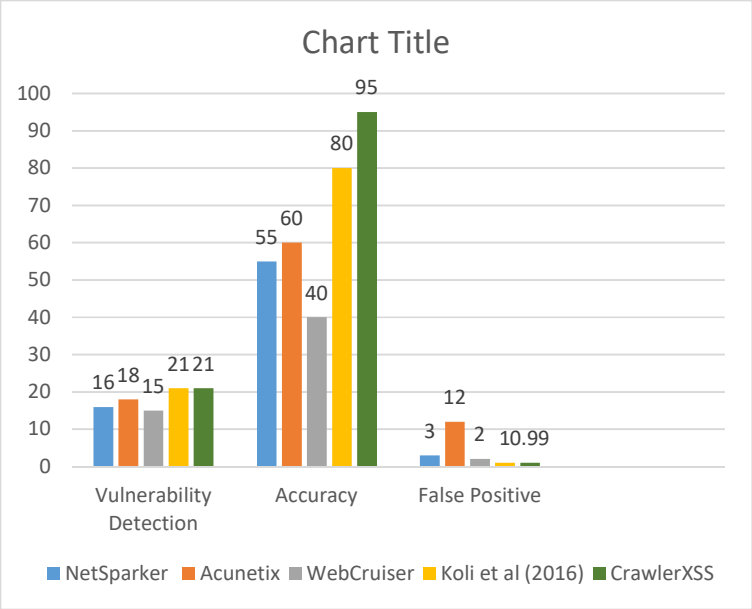

Figure 6: Graphical Summary of Performance Metrics

D. Computational Performance

We measured the performance of the XSS unit testing using experiments performed on desktop machine with Hewlett Packard (HP) laptop with a (Tm) i5-6200U Processor running at 2.40GHz, 8.00GB of RAM and 500GB of hard disk. Our test website was subjected to the number of sequences observed per web site over a period of 100 visits. It takes an average of 240 seconds to evaluate and crawl a complete website. Some websites may contain files with multiple paths which make some websites scan faster than others. We believe the approach described in this paper will scale well for large applications.

4. Discussion Of Results

CrawlerXSS detected XSS vulnerabilities in all the web sites used as data set with 100% capability to detect vulnerabilities. It can be seen that Crawler XSS matched Koli et. al., (2016) in terms of ability to detect vulnerabilities. This implies that CrawlerXSS and Koli et. al., (2016) performed better than other web vulnerabilities scanner. From the results of accuracy test, the implementation of Fuzzy Inference System on the detection of XSS resulted in a noticeable increase in accuracy. The accuracy rate of 95% was higher than 80% accuracy rate obtained by Koli et. al., (2016) and considerably higher than other web vulnerabilities scanners. False positive rate of 0.99% was recorded by CrawlerXSS. This is a marginal reduction in false positive rate compared with Koli et. al., (2016). It is also the least false rate recorded by all the web scanners considered.

5. Conclusion

From the result of all the comparison, it is clear that the CrawlerXSS performed better than other web vulnerability scanners in terms of accuracy and false positive rate. It was also as fully effective as Koli et al (2016) in terms of the number of vulnerabilities detected. The better performance indices could be attributed to the introduction of Fuzzy Inference System.

5.1 Future Work

Future research work on this topic will involve the definition of more DOM based features that could lead to detection of other code and server side injection vulnerabilities like SQL and cross site request forgery attacks. Also, the method could be implemented using other soft computing approaches like genetic algorithm and neural networks.

REFERENCES

1. Hossein S. & Hisham H. (2016). Fuzzy Rule-Based Vulnerability Assessment Framework for Web Applications. International Journal of Secure Software Engineering. 7(2), pp 145 - 160.
2. Isatou H, Abubakr S, Hazura Z & Novia A. (2014). An Approach for Cross Site Scripting Detection and Removal Based on Genetic Algorithms. Ninth International Conference on Software Engineering Advances: France. Pp. 227 – 232
3. NithyaV., Lakshmana P. and Malarvizhi C. (2015). A Survey on Detection and Prevention of Cross-Site Scripting Attack. International Journal of Security and Its Applications. 9(3), pp. 139-152
4. Sharma P., Johari R., and Sarma S. S. (2012). Integrated Approach to Prevent SQL Injection Attack and Reflected Cross Site Scripting Attack, Int. Journal of System Assurance in Engineering. 3(4), pp. 343–351.
5. Sun Y. and He D. (2012). Model Checking for the Defence against Cross-Site Scripting Attacks. In 2012 International Conference on Computer Science and Service System, pp. 2161–2164.
6. Van Gundy M. and Chen H. (2012). Noncespaces: Using randomization to defeat cross-site scripting attacks, International Journal of Computer Security., 31(4), pp. 612–628.
7. Shar L. K. and Tan H. B. K(2012)., Automated removal of cross site scripting vulnerabilities in web applications, Journal of Information Software Technology. 54(5), pp. 467–478.
8. Mamdani E. (1974), Applications of Fuzzy Algorithm for Control of a Simple Dynamic Plant, IEEE, 121 (12), pp.1585- 1588.
9. Trends-Report2013. Applications vulnerabilities report 2013 retrieved 26th March, 2016 http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vuln-
10. Cordon O. (2001). Evolutional Tuning and Learning of Fuzzy Knowledge Base. Advances in Fuzzy Systems – Application and Theory. 19(2): 978 – 981.
11. Bathia P., Beerelli B. R., and Laverdière M. (2011), Assisting Programmers Resolving Vulnerabilities in Java Web Applications in CCIST. Communications in Computer and Information Science, 133(1), pp. 268–279.
12. Kanchan A. and Harmanpreet S.(2014). Anomaly Detection System in SDLC using Data Mining and Fuzzy Logic. International Journal of Scientific & Engineering Research. 5(12), pp
13. Ali M. Alakeel (2014). A New Approach for Assertions Processing during Assertion-Based Software Testing. World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering. 8 (12).
14. Hossain S. and Hisham H. (2014). Risk Assessment of Code Injection Vulnerabilities Using Fuzzy Logic-Based System Security Assessment Conference. Pyeongyang, Korea.
15. Zadeh, A. L.(1965).Fuzzy Sets. University of California: Berkeley
16. Animesh B. and Debasish M.(2014). Genetic Algorithm Based Hybrid Fuzzy System for Assessing Morningness. Advances in Fuzzy Systems. Hindawi Publishing Corporation article ID 732831
17. Duchene F., Groz R. and Rwat S. A.(2012.). Vulerability Detection Using Model Inference Assisted Evolutionary Fuzzing. IEEE Fifth International Conference on Software Testing: Montreal, Canada. Pp. 815-817
18. Mankad A.(2014). Measuring Human Intelligence by applying Soft Computing Techniques genetic Fuzzy approach. International Journal of Emerging Research in Management &Technology. 4(2), p 33 – 40.
19. Alberto P., Sala A., and Olivares M.(2015). Fuzzy Logic Controllers. Methodology, advantages and Drawbacks: http://www.softcomputing.es/
20. Saleh A., Rozalia B., Bujaa B. C., Kamarularifin A., Mohd A., FaradillaA.(2015). A Methodfor Web Application Vulnerabilities Detection by Using Boyer-Moore String Matching Algorithm. Information Systems International Conference. 72 (3) , pp.112 -12
21. Abbass A. N. and Nasser M.(2016). Presentation of a Pattern to Counteract the Attacks of XSS Malware. International Journal of Computer Applications. 143(2), pp. 78 – 88.

22. Koli M., Pooja S., Pranali H.K., and Prathmesh N. G.(2016). SQL Injection and XSS Vulnerabilities Countermeasures in Web Applications. International Journal on Recent and Innovation Trends in Computing and Comm. 4(4), pp 692 – 695.
23. Krishnaveni S. and Sathiyakumari K.(2013). Multiclass Classification of XSS Web Page Attack using Machine Learning Techniques. International Journal of Computer Applications Vol 74.No. 12. Pp 36-40
24. Wang L. X., and Mendel J. M.(1992). Fuzzy rules by learning from examples. IEEE Transactions on Systems, Man and Cybernetics. 22(6). Pp 1414 – 1427
25. Vandana D., Himanshu Y. and Anurag J. (2014). A survey on Web Application Vulnerabilities: International Journal of Computer Applications. 108(1), pp 25 – 31.
26. Georgios P. S. and Sokratis K. K. (2009). Using Fuzzy Inference System to Reduce False Positive in Intrusion Detection. Elsevier Computer & Security Conference. 29(1): 35 – 44.
27. Hossein S. and Hisham H. (2016). Fuzzy Rule-Based Vulnerability Assessment Framework forWeb Applications. International Journal of Secure Software Engineering. 7(2):145 - 160.