Hindawi
Journal of Computer Networks and Communications
Volume 2018, Article ID 8159548, 10 pages
https://doi.org/10.1155/2018/8159548

Research Article

Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System

Bakare K. Ayeni, Junaidu B. Sahalu, and Kolawole R. Adeyanju

Department of Computer Science, Faculty of Sciences, Ahmadu Bello University, Zaria, Nigeria

Correspondence should be addressed to Bakare K. Ayeni; bakarre@gmail.com

Received 18 December 2017; Revised 28 May 2018; Accepted 14 June 2018; Published 1 August 2018

Academic Editor: Youyun Xu

Copyright © 2018 Bakare K. Ayeni et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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 research studies on the web application security centre focus on attack prevention and mechanisms for secure coding; recent methods for those attacks do not only generate high false positives but also have little considerations for the users who oftentimes are the victims of malicious attacks. Motivated by this problem, this paper describes an "intelligent" tool for detecting cross-site scripting flaws in web applications. This paper describes the method implemented based on fuzzy logic to detect classic XSS weaknesses and to provide some results on experimentations. Our detection framework recorded 15% improvement in accuracy and 0.01% reduction in the false-positive rate which is considerably lower than that found in the existing work by Koli et al. Our approach also serves as a decision-making tool for the users.

1. Introduction

Over the past decade, the Internet has witnessed tremendous growth in the 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.

Almost every business today is tending towards growth beyond borders; hence, the worldwide web plays a critical role in almost all human endeavours and overall development. One of the best ways to have this critical online presence is through web applications. Web applications are computer programs that utilise web technology to perform tasks on the Internet. It is therefore not surprising that the advent of web applications and other smart devices like smartphones, tablets, and other mobile phones has changed the medium of communication and information exchange across platforms. With the substantial proliferation and ubiquitous nature of these applications on the Internet, application developers are forced to rethink their development strategies and model their security concerns in a way not to fall prey of target to hackers and web attackers who are daily on the Internet seeking for improper coding practices they can capitalise on to steal sensitive information and perpetrate their evil plots. Also, as the number of web applications grows, so also do vulnerabilities and have become a major talking point in various web applications development and security fora.

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 [1]. Therefore, web applications have become major targets of hackers who take advantage of web developers' poor coding practices, weaknesses in the application code, inappropriate user input authorization, or nonadherence to security standards by the software developers. These vulnerabilities could be either on the

server side or more dangerously on the client side. The vulnerabilities include SQL injection, cross-site request forgery, information leakage, session hijacking, and cross-site scripting. This paper focuses on cross-site scripting attack detection.

Malicious injection of the code within vulnerable web applications to trick users and redirect them to untrusted websites 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 (Figure 1).

A web application could be exposed to XSS vulnerability if input data are not properly sanitized. This could occur with the use of special characters to cause web browser interpreters to switch from data context to code context. It is exhibited through flaws in the application code, inappropriate user input authorization, or nonadherence to security standards by software developers.

The input sources manipulated by attackers include HTML forms, cookies, hidden fields, and get and post parameters. The usual process involves three parties—the attacker, a client, and the website. XSS breaches could lead to fraud and identity theft, regulatory fines, loss of goodwill, litigations, and loss of customers.

According to the Open Web Application Security Project (OWASP) 2015 Report, the two most common web application vulnerabilities threatening the privacy and security of clients and web applications nowadays are Structured Query Language (SQL) injection and cross-site scripting (XSS).

Many research studies 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 [2–4]. Few research activities have addressed their detection [5–7].

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 [8]. Fonseca et al. [9] proposed a method to evaluate and benchmark automatic web vulnerability scanners; using software fault injection techniques, they found out the scanners' coverage is low and the percentage of false positives is very high. Without a clear idea of the coverage and false positive rate of web scanning tools, it is difficult to judge the relevance of the results the scanners provide and comparing them may be difficult [10].

Not only is inadequate attention given to detection frameworks, recent research efforts come with high false rates, and these existing approaches do not have consideration for measuring the level of threats being exposed to and the degree of severity of XSS present in web applications.

Methods of preventing and detecting vulnerabilities in web applications can be broadly divided into two: (i) static analysis which is a review of the source code prior to program execution. This approach guards against the occurrence of certain types of vulnerabilities and does not give room for unspecified vulnerability at the time of coding. (ii) Dynamic analysis which is the specification of what the program does during execution mostly through the interface with the interpreter by analyzing the syntactic structure.

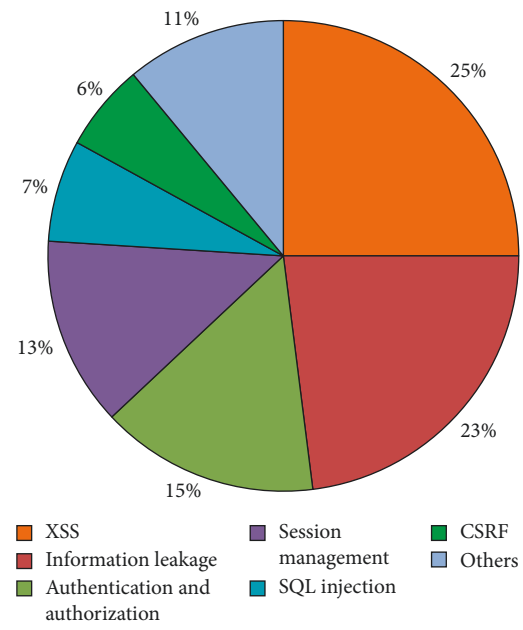


Figure 1: Cenzic Application Vulnerability Trends Report (2013). Source: http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vulnerability-Trends-Report-2013.pdf.

An emerging soft computing technique applicable to web security is fuzzy logic. The studies [11–14] successfully applied the fuzzy rule-based approach to web vulnerability and intrusion detection with encouraging results. The advantages that fuzzy logic brings to web security are (i) the development of linguistic variables as classifiers to predict whether or not a script is malicious; (ii) the provision of classifiers with predictive capability to detect new malicious scripts; and (iii) 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 [15]. Since fuzzy logic systems can manipulate linguistic rules in a natural way, they are particularly suitable in combining the various DOM parameters and rules to provide optimal results.

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

(i) Selection and implementation of DOM-based features for XSS detection using the OWASP web application security guideline

(ii) Application of the fuzzy logic inference system to web application vulnerability detection

(iii) Implementation of the user interface for users to have a verdict on their level of exposure to cross-site scripting attack while visiting a website.

This paper is organized as follows: Section 2 gives a background of XSS and fuzzy logic. Section 3 reviews related research conducted on detection of XSS vulnerabilities. In Section 4, we describe our proposed approach and experimental results. In Section 5, we discussed the proposed implementation and results, while Section 6 concludes the paper.

2. Background

XSS is a type of computer security vulnerability typically found in web applications which enables malicious code to be injected into the client-side script of webpages 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. [5], XSS attacks are of three types, namely, reflected, stored, and document object model-(DOM-) based.

2.1. Stored (Persistent) XSS Attacks. Stored 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. Stored attacks are those where the injected code is permanently stored on the target servers. The victim then retrieves the malicious script from the server when it requests the stored information.

The persistent or stored XSS attack occurs when the malicious code submitted by the attacker is saved by the server in the database and then permanently runs in the normal page. Many websites host a support forum where registered users can make contributions by posting messages, which are stored in the database. An attacker could post a message containing the malicious JavaScript code instead. If the server fails to sanitize the input provided, it results in execution of the injected script. The code will be executed whenever a user tries to read the post.

2.2. Reflected (Nonpersistent) XSS Attacks. Reflected 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 responses that include some or all of the inputs sent to the server as part of the request. When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server.

2.3. DOM-Based Attacks. DOM-based XSS attacks are found on the client side. Attackers are able to collect sensitive information from the user's computer. For instance, an attacker might place a malicious flash file on a site that clients visit. When the client's browser downloads the video, the file triggers a script in the browser and the attacker can take control of elements of the pages inside the client's browser. DOM-based XSS represents a new threat dimension to web applications, with different prerequisites as compared to the standard XSS. Therefore, there exist some web applications on the Internet that are vulnerable to DOM-based XSS without showing features of the standard XSS [5].

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 the underlying sensitive documents and information databases in case of SQL injection attacks.

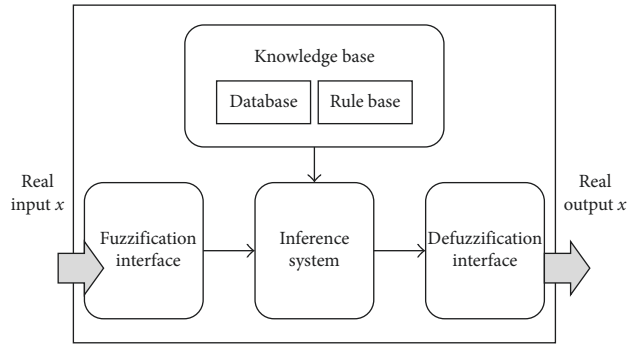


Figure 2: Mamdani fuzzy inference system [18].

According to the Cenzic Application Vulnerability Trends Report (2013) [16], XSS represents 25% of the total attack statistics and is considered as the topmost web application attack.

2.4. Concept of Fuzzy Logic. Zadeh [17] introduced "fuzzy sets" and laid out the mathematics of fuzzy logic. Fuzzy logic recognizes not only clear-cut alternatives but also the infinite gradations in between. These numeric values are then used to derive exact solutions to problems.

According to Cordon [18], the fuzzy inference system (FIS) consists of three components: (i) a rule base which contains a selection of fuzzy rules similar to collection or rules in an expert system [19], (ii) a database which defines the membership functions used in the rules, and (iii) a reasoning mechanism to carry out the inference procedure on the rules [20].

Figure 2 gives a standard representation of the fuzzy logic procedure.

Fuzzy logic is concerned with imprecision and approximate reasoning. According to Mankad [21], fuzzy logic may be viewed as an extension to the classical logic system which provides the conceptual framework for handling problems of knowledge representation in an uncertain and imprecise environment. Also, Alberto et al. [22] 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, the Mamdani fuzzy inference system is the most popular, powerful, and widely used methodology in developing fuzzy models and analyzing data [15].

3. 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. [23] 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).

3.1. Input Validation Vulnerabilities. An application is exposed to input validation vulnerabilities if an attacker finds that the application makes untested assumptions about the type, duration, format, or scope of input data. When inputs are not properly sanitised, attackers are ready to introduce maliciously crafted inputs, which might alter program performances or allow unauthorized access to resources. Improper input validation may invite a range of attacks, like buffer overflow attacks, SQL injection attacks, cross-site scripting, and other code injection attacks [23].

SQL injection attack is the insertion of the SQL query through the input data from a client to the application. A successful SQL injection exploit can read and modify sensitive data from the database, implement administrative privileges on the database, and in some cases issue commands to the operating system [24]. SQL injection attacks are a type of injection attack, in which SQL commands are injected into the data-plane input in order to affect the execution of predefined SQL commands [11].

Cross-site scripting (XSS) vulnerabilities occur when data get into web application through an untrusted source, mainly a web request, and the data are included in dynamic content that is sent to a web user as the HTTP response without being validated for the malicious script. Cross-site scripting attacks and their various forms have been extensively discussed in Section 2.

3.2. Session Management Vulnerabilities. Session management enables a web application to keep track of user inputs and maintain application states. In web application development, session management is accomplished through the cooperation between the client and the server. Since session ID is the only proof of the client's identity, its confidentiality, integrity, and authenticity need to be ensured to avoid session hijacking [15]. Vulnerabilities that are specific to session management are great threats to any web application and are also among the most challenging ones to find and fix. Sessions are targets for attackers because they can be used to gain access to a system without having to authenticate.

Broken authentication, session management, cross-site request forgery (CSRF), and insufficient transport layer protection are some of the session management vulnerabilities within the OWASP top ten security risks.

3.3. Application Logic Vulnerabilities. Application logic attack aims to bypass or evade the expected order with which application features are set up. Generally, such attacks are aimed at a website, but they can also be targeted at a site's visitors and their private customer data.

Unlike common application attacks, such as SQL injection, each application logic attack is usually unique, since it has to exploit a function or a feature that is specific to the application. This makes it more difficult for automated vulnerability testing tools to detect such attacks because they are caused by flaws in the logic. When application logic attacks are successful, it is often because developers do not build sufficient process validation and control into the application. This lack of flow control allows attackers to perform certain steps incorrectly or out of order.

The decentralized structure of web applications poses significant challenges to the implementation of business logic. First, since web application modules can be accessed directly through their URLs, the interface hiding mechanism has been commonly used as a measure for access control in web applications. However, this mechanism alone, which follows the principle of "security by obscurity," is not sufficient to enforce the control flow of a web application. Application logic vulnerabilities are highly dependent on the intended functionality of a web application.

4. 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, e.g., 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 [25]. This approach simply used a heuristic-driven substring-matching algorithm to develop a crawler. They proposed an approach for inferring models of web applications to form an attack grammar. The attack grammar used produces slices which narrow the search space. Genetic algorithms are then used to schedule the malicious inputs which are sent to the application. Lofty as the idea was, it assumed the ability to reset the application to its initial node, which might not always be practical. Also, the framework hypothesized that an XSS is the result of only one fuzzed value.

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. [5]. The first component involves converting the source codes of the application to control flow graphs (CFGs). The second component focuses on detecting the XSS. The third component concentrates on its removal. The approach failed to detect XSS whose paths cannot be identified in the OWASP Enterprise Security Application Programming Interfaces (ESAPI) standards. Therefore, XSS vulnerabilities that are not defined in the ESAPI are completely missed. The work proposed by Huang et al. [26] used a number of software testing techniques such as black-box testing, fault injection, and behavior monitoring of web application in order to deduce the presence of vulnerabilities. This approach combines user experience modeling as black-box testing and user behavior simulation. The approach was unable to provide instant web application protection, and they cannot guarantee the detection of all flaws as well.

In a solution proposed by Saleh et al.[27], Boyer–Moore string matching algorithm was used for the technique developed for detection method. It compares the characters of the inputted pattern with the characters of the webpage from the right to the left by using two heuristics called the bad-character shift and the good-suffix shift. The core idea of the

module is to fulfill the desired criteria which are able to scan from the right to the left and scan character by character for the inputted pattern. However, when the length of URL is long, the scanner takes a long time to complete its scanning.

The work proposed by Abbass and Nasser [28] suggested an algorithm named "NUIVT" (novel user input validation test). The algorithm includes three steps: the first step analyzes the user input fields and detection in the input forms. The second step converts input types to regular phrases. The third step tests for invalid inputs in order 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 and is tedious and time-consuming.

An SQL and XSS architecture was proposed by Koli et al. [29]. They developed an SQL injection and XSS detection method that looks for attack signatures by using filters for the HTTP requests sent by users. A detection component is used for determining whether the script tag is present or not. The result is stored in a database as a response to users. They carried out a comparison of their work with well-known vulnerability scanners to determine its efficiency. Major drawback in their research was that if the attack pattern is not stored in its database, then the tool cannot detect the attack successfully.

It can be deduced from the literature that approaches to detect XSS using genetic algorithms have not been too effective. The approach using the 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 the detection process with the hope of improving efficiency and reliability.

We are motivated by the application of fuzzy logic for detection of web security issues by Mankad [17] and phishing website detection by Alberto et al. [22]. They applied the fuzzy inference system to assess risks due to code injection vulnerabilities based on a set of linguistic terms for vulnerability and severity levels.

The work of Mankad [21] develops a rule-based security assurance system. It relies on extracting the exploitation paths of an application, and then, it represents the path as a finite-state automata (FSA) that can be used as rule-based signatures to detect exploitations. The work of Abbass and Nasser [28] proposes a fuzzy logic-based approach for detecting buffer overflow vulnerability in C programs.

The work done by Hossain and Hisham [11] was a fuzzy logic-based system to assess risks due to different types of code injection vulnerabilities. They proposed code-level metrics that were used to establish the linguistic terms to relate the subjective magnitude and the corresponding impact due to the actual exploitation of the vulnerability. The fuzzy system developed obtained information about webpages using web services. The information obtained is used to determine the fuzzy input, for which they attached a rating which has a value between one and three.

Also, to assess vulnerability risks in web applications, Hossain and Hisham [30] developed a fuzzy logic-based system (FLS) framework to assess code injection vulnerabilities present in an application. They also developed a set of rule bases to assess the risk level. The FLS could be a useful tool to aid application developers and industry practitioners to assess the risk and plan ahead by employing necessary mitigation approaches. The authors evaluated their proposed approach using three real-world web applications implemented in PHP. The initial results indicate that the proposed FLS approach can effectively discover vulnerabilities in high-risk applications.

In a paper presented by Alakeel [31], a novel software testing metric technique for assertion-based software testing based on fuzzy logic technology was discussed. The main goal of the proposed approach was to enhance the performance of assertion-based software testing in the presence of a large number of assertions. The results of this experiment are very encouraging, where applying the proposed approach has enhanced the performance of assertion-based testing as shown by the increase in the number of assertions violated in the programs considered in the experiment.

Kanchan and Harmanpreet [7] proposed a learning algorithm that can select a set of attributes from a given data set based on weight by the SVM technique and then classify into fuzzy rules based on the processing of the Apriori algorithm and application of the fuzzy inference engine to detect the anomalies in the software development process. It selects the relevant attributes by outlier analysis and computes the association rules based on the Apriori algorithm. Thereafter, it generates the fuzzy association rules based on min-max derivation. The inputs are analysed with the Mamdani fuzzy inference system.

This paper is motivated by the observation that XSS vulnerability detection can be modeled in the form of the fuzzy inference system. Also, other approaches in the literature do not have the capability to estimate the overall risk due to diverse severity levels for a given vulnerability as also corroborated by Mankad et al. [21, 32]. Thus, a suitable framework to detect XSS by introducing the decision-making inference system is hereby introduced.

5. Proposed Approach

XSS is one of the most exploited weaknesses in web application and one of the most studied ones. Full protection is not possible, as any full protection against any programming errors or bugs might be difficult to achieve. Good programming practices, intelligence in libraries, and browsers have been developed to protect against XSS. There are also a lot of proposed tools to detect XSS risks.

Many approaches have been proposed as detailed in the previous two sections (Introduction and Related Works). In this section, we present our approach for detecting XSS in web applications. We also present the fuzzy inference procedure applied in the detection phase.

5.1. Detecting XSS Attacks. We prepare 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 webpages (both statically and dynamically generated). If the data (or

a manipulated form of them) are 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 (URLs). As enlisted by Krishnaveni and Sathiyakumari [33], the seven sources through which XSS vulnerabilities could be introduced to web applications are the following: $X1$ = document.location, $X2$ = document.referrer, $X3$ = document.location.href, $X4$ = window.location, $X5$ = document.cookie, $X6$ = document.URLUnencoded, and $X7$ = location.header. Their view was supported by the OWASP 2012 Report, in which the lists of locations where XSS are more prone are summarized.

We briefly discuss the attributes and the usage of these DOM-based data and elements that make them a subtle target of XSS attacks based on *Web Applications Hacker's Handbook* [34] and the tests conducted on them.

5.1.1. HTTP Referrer Head. The HTTP referrer is a header field that identifies the address of the webpage that is linked to the resource being requested such that the new webpage can see where the request originated. More generally, a referrer is the URL of a previous item which led to the present request. As a result of the sensitive information the referrer header carries, it can be easily used to violate privacy and introduce vulnerabilities. We created module checks for referrer header injection vulnerabilities by creating tags for all referrer headers to check whether there are altered requests. In this alteration, the module checks if the referrer is subject to XSS payload injection.

5.1.2. Window Location Test. This property returns a location object with information about the current location of the document. Although window location is a read-only object, it can be assigned a DOM string. This means that you can work with location as if it were a string in most cases. This introduces some elements of possible manipulation which need to be checked as it can be an entry point for possible malicious code injection. We trace the relevant data through the code to identify what actions are performed with it. If the data (or a manipulated form of them) are passed to one of the window location APIs, the application may be vulnerable to XSS.

5.1.3. The Document Referrer. This is pointed to the page which is linked to the current page inside the Iframe. If the content contains links, which allow users to navigate through a few pages, then only the first page loaded inside the Iframe will have a parent frame URI as document.referrer. However, many developers do not pay adequate attention to this restriction. Each page loaded by clicking a link inside the Iframe will have the URI of the page containing the link in the document referrer. The fact that the user controls every aspect of every request, including the HTTP headers, means this control can be easily circumvented by using an intercepting proxy to change the value of the document referrer to the value that the application requires. A part of the DOM testing module is to check the URL of the webpage being visited to return a value of true or false if there is a disabled document referrer object.

5.1.4. The Document Location. This read-only property returns a location object which contains information about the URL of the document and provides methods for changing that URL and loading another URL. Releasing information about sensitive documents that may be contained on a webpage is a good spot for attackers to manipulate unguarded web applications.

5.1.5. Document URL Unencoded. This returns the URL of the current document, but with any URL-encoded characters returned to their plain-language version (e.g., %20 is converted to a space character). Applications frequently transmit data via the client using preset URL parameters. When URL-containing parameters are displayed in the browser's location bar, any parameters can be modified easily by any user without the use of tools.

5.1.6. Cookies. Cookies are often used in web applications to identify users and their authenticated session. Stealing a cookie from a web application will lead to hijacking the authenticated user's session. The cookie value string ensures that the strings do not contain any commas, semicolons, or whitespace (which are disallowed in cookie values). Some user agent implementations support cookie prefix signals to the browser, and cookie request should be transmitted over a secure channel. Cookies must be restricted and traced to a secure origin. This prevents the cookie from being sent to other domains.

5.1.7. Headers. These are used to provide information about the HTML document in a Meta tag or to gather information about another document. They can be used to describe the size of the data, to check another document that the server should return (i.e., instead of returning a virtual document created by the script itself), and to record HTTP status codes. Headers could easily be redirected, and information they contain could be easily made available to attackers. The headers are considered as possible entry points for input-based attacks.

Many important categories of vulnerabilities are triggered by unexpected user inputs 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 webpage. 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

```
(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 HTML Document (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.

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 whether it is a file or a directory.

5.2. 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 are then forwarded to the fuzzy inference system to identify possible vulnerability occurrence. Figure 4 presents the system architecture.

5.3. The Fuzzy Logic Component. This component describes the design strategy for generation of the fuzzy inference procedure. The fuzzy inference system employs the Fuzzy IF-THEN rules which can model the qualitative aspect of human knowledge without employing precise qualitative analysis [17]. Due to their concise form, Fuzzy IF-THEN rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision.

Here, each input fuzzy set defined in the fuzzy system includes three membership functions and an output fuzzy set which also contains three membership functions. Each membership function used triangular function for the fuzzification strategy.

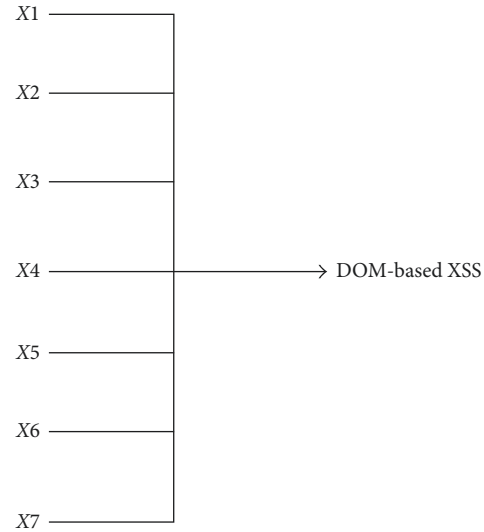


Figure 3: Locations vulnerable to DOM-based XSS attacks.

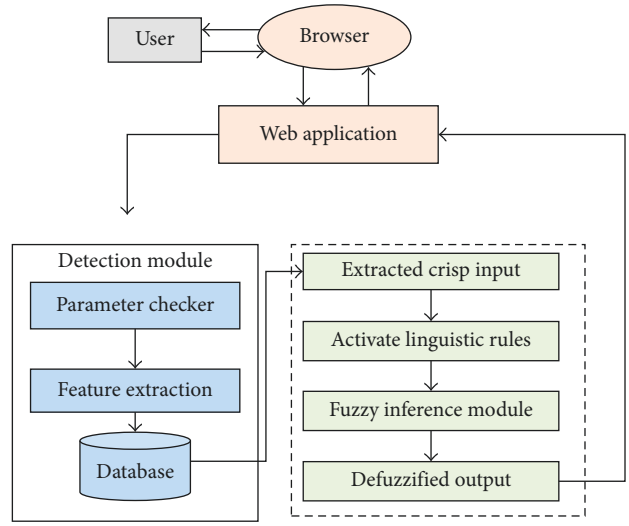


Figure 4: System architecture.

5.4. Defining Linguistic Variables and Terms. We consider each of the parameters defined in Figure 3 as crisp inputs for the 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, and High).

5.5. 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-shaped or a trapezoid-shaped membership representation often provides a suitable representation. To define membership function for each of the linguistic variables, we apply triangular membership function

Table 1: Crisp input and linguistic variables.

Crisp input	Linguistic variables
document.location ($X1$)	
document.referrer ($X2$)	
document.location.href ($X3$)	
window.location ($X4$)	Low, Medium, and High
document.cookie ($X5$)	
document.URLUnencoded ($X6$)	
location.header ($X7$)	

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)

(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, with three rules each (High, Low, and Medium).

5.6. 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 rules for our fuzzy inference using Wang and Mendel's technique. Using this method, the AND operator results in the maximum of truth (membership) values. It was proposed by Wang and Mendel [35] using the Mamdani model.

5.7. Aggregation of Rules and Defuzzification. The results 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 the centre of gravity approach and the maximum or minimum values.

5.8. 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 screenshot of scanner progress is shown in Figure 5.

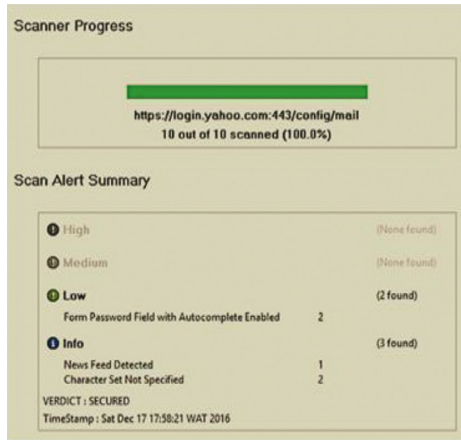


Figure 5: Scanner progress.

6. Performance Analysis

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

(i) Capability to detect vulnerabilities

(ii) Accuracy

(iii) False-positive rate.

Comparison with the work by Koli et al. [29] was done using the measures (i), (ii), and (iii) based on the number of webpages to be used for evaluation.

6.1. Accuracy. This is a measure of the degree to which the results of the test scanning conducted on the developed framework conform to the correct values or the standard data set. Accuracy is calculated using the following formula:

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

where TNC = the number of true-negative cases, FPC = the number of false-positive cases, FNC = the number of false-negative cases, and TPC = the number of true-positive cases.

6.2. False-Positive Rate. In performing multiple comparisons, the 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

$$\text{FPR} = \frac{\text{FP}}{(\text{FP} + \text{TN})}\, n, \quad (2)$$

where FP is the number of false positives and TN is the number of true negatives.

6.3. Number of Vulnerabilities Detected. This is a minimum standard for checking the developed application's capabilities

Table 3: Summary of performance metrics.

Parameter	Netsparker	Acunetix	WebCruiser	Koli et al. [29]	CrawlerXSS
Vulnerability detection	16	18	15	21	21
Accuracy	55	60	40	80	95
False positive	3	12	2	1	0.99

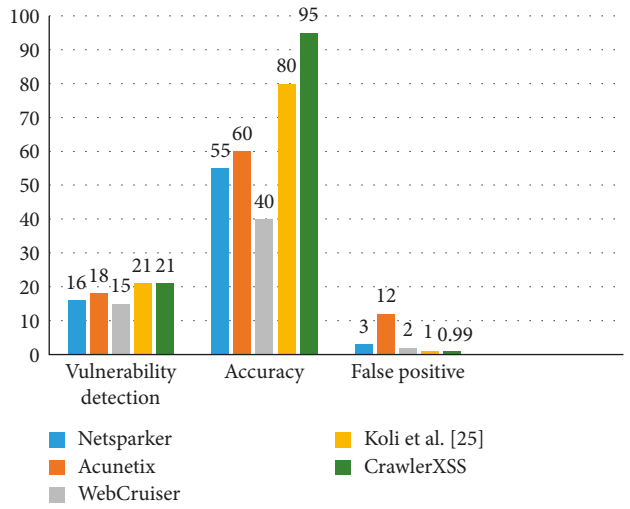


Figure 6: Graphical summary of performance metrics.

to discover XSS vulnerabilities in applications that are known to be vulnerable.

Table 3 shows the results for the number of vulnerabilities detected.

The results show that CrawlerXSS was able to detect XSS vulnerabilities in all the websites visited with the results being the same with Koli et al. [29].

Figure 6 shows the number of sequences observed per website over a period of 100 visits.

6.4. Computational Performance. We measured the performance of the XSS unit testing using experiments performed on a desktop machine with a Hewlett–Packard (HP) laptop with the (TM) i5-6200U Processor running at 2.40 GHz, 8.00 GB of RAM, and 500 GB of hard disk. Our test website was subjected to the number of sequences observed per website 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.

7. Discussion

CrawlerXSS detected XSS vulnerabilities with 100% capability in all the websites used as data sets. It can be seen that CrawlerXSS matched the architecture proposed by Koli et al. [29] in terms of ability to detect vulnerabilities. This implies that CrawlerXSS and the architecture proposed by Koli et al. [29] performed better than other web vulnerability scanners.

From the results of the accuracy test, the implementation of the fuzzy inference system for the detection of XSS resulted in a noticeable increase in accuracy. The accuracy rate of 95% was higher than the 80% accuracy rate obtained by Koli et al. [29] and considerably higher than other web vulnerability scanners.

The false-positive rate of 0.99% was recorded by CrawlerXSS. This is a marginal reduction in the false-positive rate compared with that in the study [29]. It is also the least false rate recorded by all the web scanners considered.

8. Conclusion

From the result of all the comparisons, it is clear that CrawlerXSS performed better than other web vulnerability scanners in terms of accuracy and false-positive rate. It was also as fully effective as the architecture proposed by Koli et al. [29] in terms of the number of vulnerabilities detected. The better performance indices could be attributed to the introduction of the fuzzy inference system.

9. 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.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] H. Hibshi, *Composite Security Requirements in the Presence of Uncertainty. Societal Computing Institute for Software Research*, Carnegie Mellon University, Pittsburgh, PA, USA, 2016.

[2] P. Sharma, R. Johari, and S. S. Sarma, "Integrated approach to prevent SQL injection attack and reflected cross site scripting attack," *International Journal of System Assurance Engineering and Management*, vol. 3, no. 4, pp. 343–351, 2012.

[3] Y. Sun and D. He, "Model checking for the defence against cross-site scripting attacks," in *Proceedings of the 2012 International Conference on Computer Science and Service System*, pp. 2161–2164, Maui, Hawaii, January 2012.

[4] M. Van Gundy and H. Chen, "Noncespaces: using randomization to defeat cross-site scripting attacks," *International Journal of Computer Security*, vol. 31, no. 4, pp. 612–628, 2012.

[5] 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.

[6] P. Bathia, B. R. Beerelli, and M. Laverdière, "Assisting programmers resolving vulnerabilities in Java web applications in

CCIST," *Communications in Computer and Information Science*, vol. 133, no. 1, pp. 268–279, 2011.

[7] A. Kanchan and S. Harmanpreet, "Anomaly detection system in SDLC using data mining and fuzzy logic," *International Journal of Scientific and Engineering Research*, vol. 5, no. 12, 2014.

[8] V. Nithya, P. Lakshmana, and C. Malarvizhi, "A survey on detection and prevention of cross-site scripting attack," *International Journal of Security and its Applications*, vol. 9, no. 3, pp. 139–152, 2015.

[9] J. Fonseca, M. Vieira, and M. S. Madeira, "Vulnerability attack and injection for web applications," in *Proceedings of the IEEE/IFP International Conference on Dependable Systems and Networks*, pp. 93–102, Lisbon, Portugal, June 2009.

[10] V. Antunes and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009*, Lisbon, Portugal, June 2009.

[11] S. Hossain and H. Hisham, "Fuzzy rule-based vulnerability assessment framework for web applications," *International Journal of Secure Software Engineering*, vol. 7, no. 2, pp. 145–160, 2016.

[12] P. S. Georgios and K. K. Sokratis, "Using fuzzy inference system to reduce false positive in intrusion detection," *Elsevier Computer and Security Conference*, vol. 29, no. 1, pp. 35–44, 2009.

[13] H. Shahriar and H. Haddad, "Fuzzy rule-based vulnerability assessment framework for web applications," *International Journal of Secure Software Engineering*, vol. 7, no. 2, pp. 145–160, 2016.

[14] B. Prithvi and V. N. Venkatakrishna, "Precise dynamic prevention of cross site scripting attacks," in *Proceedings of the Fifth Detection of Intrusion and Malware Vulnerabilities Assessment (DIMVA) Conference*, pp. 23–43, Paris, France, 2008.

[15] B. Animesh and M. Debasish, "Genetic algorithm based hybrid fuzzy system for assessing morningness," *Advances in Fuzzy Systems*, vol. 2014, Article ID 732831, 9 pages, 2014.

[16] Infosecurity Europe, *Applications Vulnerabilities Report*, 2013, http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vuln-Trends-Report2013.

[17] A. Zadeh, *Fuzzy Sets*, University of California, Berkeley, CA, USA, 1965.

[18] O. Cordon, "Evolutional tuning and learning of fuzzy knowledge base," *Advances in Fuzzy Systems–Application and Theory*, vol. 19, no. 2, pp. 978–981, 2001.

[19] J. Durkin, *Expert System Design and Development*, Prentice-Hall, Upper Saddle River, NJ, USA, 1994.

[20] E. Mamdani, "Applications of fuzzy algorithm for control of a simple dynamic plant," *Proceedings of the Institution of Electrical Engineers*, vol. 121, no. 12, pp. 1585–1588, 1974.

[21] A. Mankad, "Measuring human intelligence by applying soft computing techniques genetic fuzzy approach," *International Journal of Emerging Research in Management and Technology*, vol. 4, no. 2, pp. 33–40, 2014.

[22] P. Alberto, A. Sala, and M. Olivares, *Fuzzy Logic Controllers. Methodology, Advantages and Drawbacks*, 2015, http://www.softcomputing.es/.

[23] D. Vandana, Y. Himanshu, and J. Anurag, "A survey on web application vulnerabilities," *International Journal of Computer Applications*, vol. 108, no. 1, pp. 25–31, 2014.

[24] S. Priti, T. Kirthika, S. Pooja, and S. Bushra, "Detection of SQL injection and XSS vulnerability in web application," *International Journal of Engineering and Applied Sciences (IJEAS)*, vol. 2, no. 3, 2015.

[25] F. Duchene, R. Groz, and S. A. Rwat, "Vulnerability detection using model inference assisted evolutionary fuzzing," in *Proceedings of the IEEE Fifth International Conference on Software Testing*, pp. 815–817, Montreal, QC, Canada, 2012.

[26] Y. W. Huang, C. H. Tsai, and D. T. Lee, "Non detrimental web application security scanning," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'04)*, pp. 219–230, Beijing, China, September 2008.

[27] A. Saleh, B. Rozalia, B. C. 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.

[28] A. N. Abbass and M. Nasser, "Presentation of a pattern to counteract the attacks of XSS Malware," *International Journal of Computer Applications*, vol. 143, no. 2, pp. 78–88, 2016.

[29] M. Koli, S. Pooja, H. K. Pranali, and N. G. Prathmesh, "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.

[30] S. Hossain and H. Hisham, "Risk assessment of code injection vulnerabilities using fuzzy logic-based system," in *Proceedings of the Security Assessment Conference*, Pyeongyang, Democratic People's Republic of Korea, 2014.

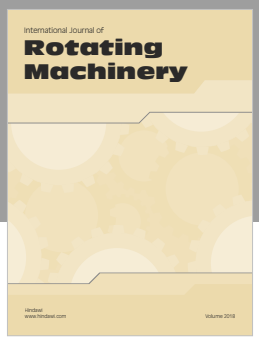
[31] A. M. Alakeel, "A new approach for assertions processing during assertion-based software testing," *World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering*, vol. 8, no. 12, 2014.

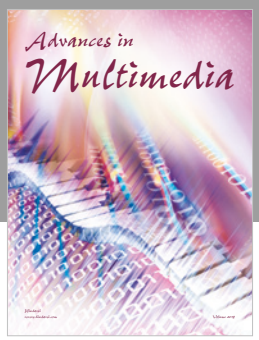
[32] L. K. Shar and H. B. Tan, "Automated removal of cross site scripting vulnerabilities in web applications," *Journal of Information Software Technology*, vol. 54, no. 5, pp. 467–478, 2012.

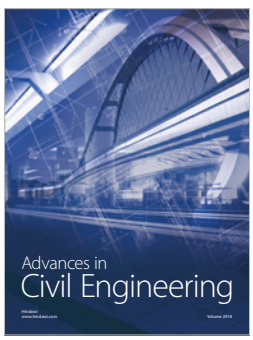
[33] S. Krishnaveni and K. Sathiyakumari, "Multiclass classification of XSS web page attack using machine learning techniques," *International Journal of Computer Applications*, vol. 74, no. 12, pp. 36–40, 2013.

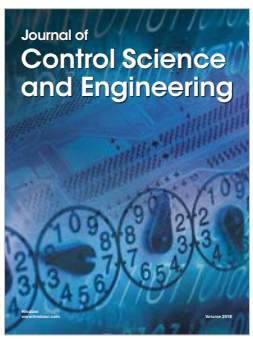
[34] D. Stuttard and M. Pinto, *Web Applications Hacker's Handbook: Finding and Exploiting Security Flaws*, Wiley Publishing Co., Indianapolis, IN, USA, 2nd edition, 2011.

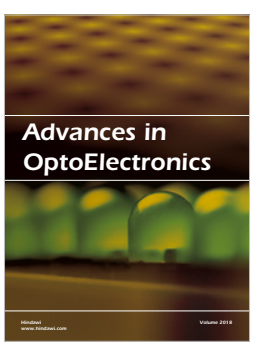
[35] L. X. Wang and J. M. Mendel, "Fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 6, pp. 1414–1427, 1992.

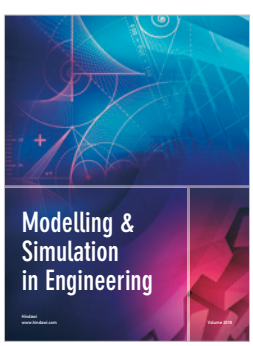


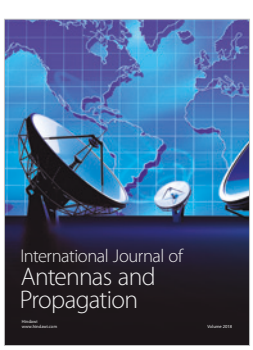


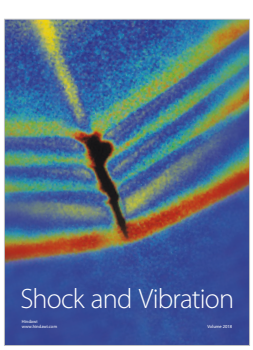


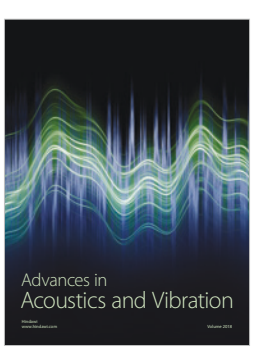












Hindawi

Submit your manuscripts at
www.hindawi.com