Applied Data Science and Analysis
Vol.2024, **pp**. 39–51
DOI: https://doi.org/10.58496/ADSA/2024/005; ISSN: 3005-317X
https://mesopotamian.press/journals/index.php/ADSA

Research Article

An Innovative Method of Malicious Code Injection Attacks on Websites

Hussein Alnabulsi [1, *], Rafiqul Islam [2], Izzat Alsmadi [3], Savitri Bevinakoppa [4]

[1] *Victorian Institute of Technology, Australia*.

[2] *School of Computing and Mathematics, Charles Sturt University, Albury, 2640, Australia.*

[3] *School of Computing, Texas A&M University-San Antonio, Texas, 4385, USA.*

[4] *School of Information Technology and Engineering, Melbourne Institute of Technology, Melbourne, 3001, Australia.*

ARTICLEINFO

Article History

Received 15 Feb 2024
Revised 23 Mar 2024
Accepted 28 Apr 2024
Published 20 May 2024

Keywords

Code Injection Attack (CIA)

SQL Injection Attack

Cross-Site Script (XSS) Attack

Web Vulnerabilities

ABSTRACT

This paper provides a model to identify website vulnerability to Code Injection Attacks (CIAs). The proposed model identifies website vulnerabilities to Code Injection Attacks (CIAs). The lack of existing models in providing checking against code injection has motivated this paper to present a new and enhanced model against web code injection attacks that uses SQL injections and Cross-Site Script (XSS) injections. This paper previews a self-assessment model which enables web administrators to know whether their current protection program is adequate, or whether a website needs stronger protection against CIAs. The Automated Injection Model assesses website vulnerability to code injection. The checking methodology consists of many intrusion methods that the attacker may use to launch code injection attacks. Methodology can give a high precision of CIA vulnerability checking for a website compared with other approaches (the minimum accuracy different between proposed approach and other approaches is 3.15%). CIAs can be a serious problem for vulnerable websites including stealing, deleting, or altering important data. Extensive experiments are conducted and compared with existing research [e.g. 1, 5, and 9] to study the effectiveness of the proposed model that can check whether a website is vulnerable to CIAs. The performance of the suggested approach has been tested on SQL injections and XSS injections. The studies showed that the detection rate of our model is 95.27%, and the false positive rate is 5.55%.

1. INTRODUCTION

Many organizations have security issues in securing their web applications against code injection attacks. SQL injection and Cross-Site Scripting (XSS) attacks allow hackers to exploit web application vulnerabilities, leading to data modification, business disruption, and data loss. In this paper we present a model to help organizations protect themselves against these types of attacks. The model enables organizations to check if their web applications are vulnerable to injection attacks. Then allows organizations to secure their web applications [28-32].The model works by attempting to inject SQL and XSS meta-codes to the target website to check for vulnerability to Code Injection Attacks (CIAs). Firstly, we specify the meta-codes for SQL and XSS injections, as shown in Table 1.

TABLE I. SQL INJECTION ATTACK AND XSS ATTACK SYNTAX. [27]

SQL injection code	XSS injection code
'="OR'	<Script>Alert('Xss Attack')</script>
or 0=0–	<IMG SRC="javascript:alert(88,83,83);">
or '2' = '2'	<script>alert(document.cookie)</script>
') or ('1' = '1'	%22sCrIpt%2Balert(%27XSS%27)%2B/sCrIpt%22
Union	<img Src=JavaScript:Alert(String.fromCharCode('XSS'))>
'	<img Src=# onMouseOver="Alert('XSS Attack')">
Shutdown	<scRipT>alERt(string.Fromcharcode('XSS'))</scRipT>

*Corresponding author. Email: hussein.a@vit.edu.au

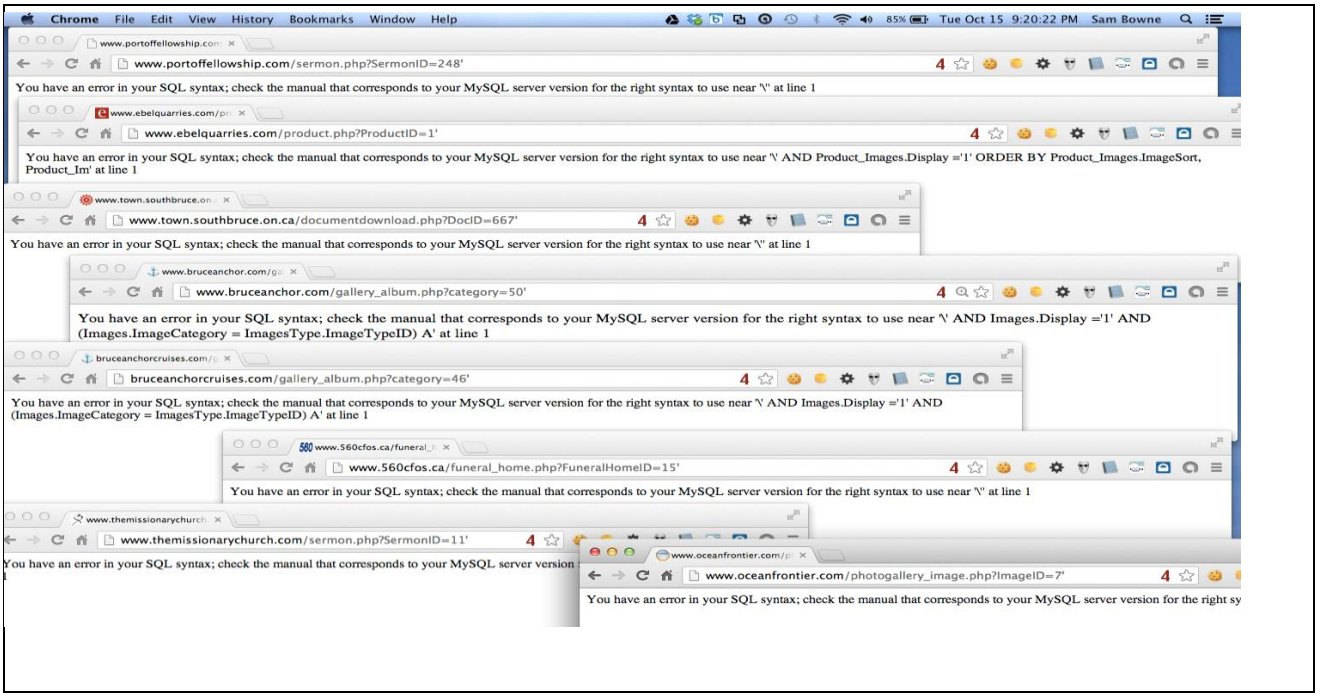

Fig. 1. Example of SQL Injection Attack Errors Displayed on Websites [11]

The model injects SQL and XSS meta-codes into a website to test its vulnerability to CIAs. If the injection operation succeeds, the Error details are displayed on the webpage as shown in Figure 1.

Figure 1 shows eight vulnerable websites. The websites were injected with the meta-code ('). The injection results show where there are Errors in the MySQL server or SQL syntax and leaving the websites vulnerable to injection attacks.

Website vulnerability can be checked on web pages connected to a database of a website. If a webpage is connected to a database, the URL ends with a sequence of a question mark (?), database identifier, equal sign (=) and a number, such as ?databaseid=x; for example: image.php?imageid=7, sermon.php?sermonid=11.

Manual checks for XSS vulnerability that are performed by inserting XSS scripts, such as <script>alert (string.Fromcharcode('xss'))</script> into form fields, XSS scripts sent by using HTTP requests, then pop-up responses occur.

There are three sorts of XSS injection attacks: 1. stored cross site scripting; 2. reflected cross site scripting; and 3. Dom-based cross site scripting.

1. Stored Cross Site Scripting: Stored XSS vulnerability are tested by a user entering an XSS script to a form in a website. A message is displayed to a second user on the website as shown in Figures 2 and 3 [22, 23].

Fig. 2. An Example of injecting a Stored cross site scripting attack.

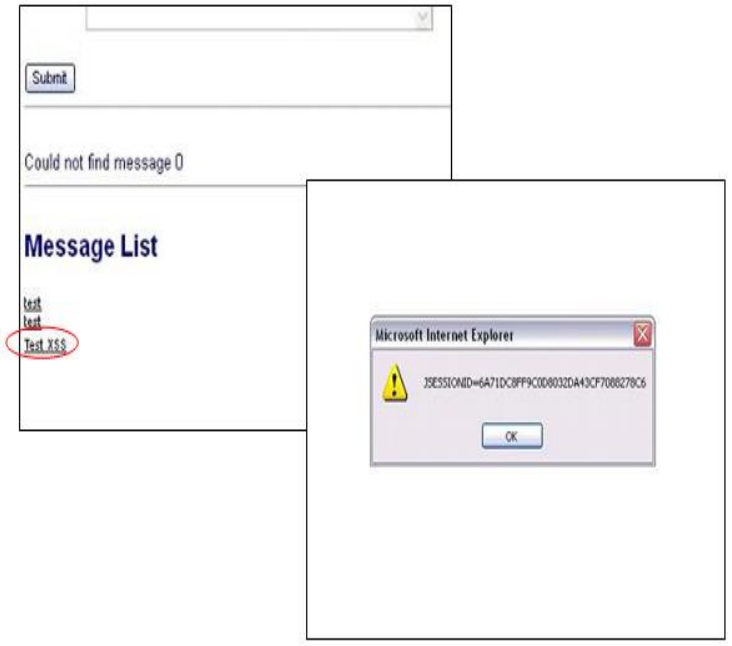

Fig. 3. Result of stored cross site scripting attack

2. The Reflected Cross-Site Scripting: Reflected XSS is the most common sort of XSS attack. The attack is created by sending many malicious URLs with the URL parameters to other users. The malicious URLs are sent by instant messages, e-mail, forums, or blogs.

   Even though some users might not click on links that look like malicious links, the reflected XSS can be executed using JavaScript.

   An example of reflected XSS using denial of service attack could be performed on the server by executing the script: (Article.PHP ? Title = <Meta%20Http-Equiv = "Refresh"%20Content="0;">). This script is a refresh request that happens every 0.3 second to particular webpage. It leads to an infinite loop of refresh requests, so it might bring down the database server or the website by flooding them with refresh requests [22].

3. The DOM-Based cross site scripting: The HTML documents have an associated Document Object Model (DOM), consisting of objects representing the document properties of the browser. The DOM-Based XSS attack payload is executed as a result of modifying the DOM in the target user's browser, so when a script is executed in the client side, the browser obtains the code with the DOM of the webpage. A hacker then can access various properties of the page. There are many objects that the attacker may change for generating an XSS condition. The common objects are:

   Document.location object, Document.referrer object, and Document.url object.

   An example, http://www.website1.com.au /web1.html?Default=<SCRIPT>Alert(Document.Cookie)</SCRIPT>. If the hacker administrates a hacking website that contains a URL of a vulnerable website on a client's local system, a XSS script could be executed with privileges of the user on the vulnerable website [24].

The paper is structured as follows: Section two presents the Problem Statement; Section three is the Literature Review; Section four is on Proposed Model; Section five is the Methodology; Section six is the Experimental Setup; Section seven focuses on the comparison with other approaches; Finally, Section eight is the Conclusion of the main themes covered here.

2. PROBLEM STATEMENT

A significant security challenge in the web applications is unaware of CIA's. Many website administrators do not know if their websites are vulnerable to CIA. When an organization creates its website, the admin of the website may not be aware of which anti-malware software to utilize for protection against malware. The organization chooses a protection system depending on many factors including how much they can afford, and how strong the anti-malware program is. In some cases,

the web administrator does not know if the anti-malware program is beneficial to protect against CIA's and can give full protection to a website against CIA's [12]. This uncertainty justifies further research. Although there is a great deal of research about detection of and protection from hacking attacks, and also many systems can provide some kind of protection against hacking attacks. There is a lack of research in area of checking the website vulnerability against CIA's. The review of current literature reveals the detection and protection against CIA, but not on checking of vulnerability before any attack occurs. This issue is important because it can provide a web administrator an alert for any susceptible of vulnerable websites against CIA.

3. LITERATURE REVIEW

The literature review discusses investigations of CIA. Some studies [4, 7] provide a model to detect different types of CIA and malware such as SQL injection, XSS attack, Missing Function Call fault (MFC), and fault injection attack. This differs from our work, in that we are checking on code injection vulnerabilities in web pages rather than detecting malware. In other research [3], the authors present AJECT (Attack inJECtion Tool) to locate many sorts of code injection vulnerabilities such as buffer over-flow, format string, and information disclosure bug. In contrast, our work focuses on finding SQL injection and XSS vulnerabilities. Other studies [6, 1, and 8] evaluates web pages' vulnerabilities by using commercial web scanners (black-box), and compares the efficiency of different scanner tools in the detection of web page vulnerabilities. This differs from our work in a web checking model whereas the previous research [1,6,8] used commercial web scanners. One study [8] compared their open source scanner with commercial web vulnerability scanners, by creating two web servers for the comparison process and using different web applications to build the web servers such as: Mutillidae, WebGoat, Swapp, and DVWA (Damn Vulnerable Web Application). The significance of our work provides a model for self-checking for global web pages on the Internet. This is more significant than the study by Bhojak et al. in [8]. They used just two custom-built web servers to compare the efficiency of their open source scanner with commercial web scanners, whereas our model can check vulnerabilities of any web page on the Internet.

With the focus on detection of CIA, Stott et al. [7] proposed a framework for assessing dependability in distributed systems with lightweight fault injectors where the authors used Networked Fault Injection and Performance Evaluator (NFTAPE) to detect automated fault injection attacks. Authors used NFTAPE to help in solving many issues, such as monitors, fault injector, target specific, and performance fault. The authors used fault injections to increase the error rate in the system. By presenting the weakness and evaluating the coverage of fault injection technique, the authors can analyse the dependability of the system. Different tools of fault injection techniques proposed by the authors, these injectors consist of simulated fault injectors, physical fault injectors, and Software Implemented Fault Injectors (SWIFI). SWIFI is not expensive, easy tool to develop, and runs in the framework. The architecture components help to create easier in many operating systems such as windows, and Linux by using NFTAPE. Authors used NFTAPE for the experiments on fault injection technique. They proposed two experiment examples of fault injection technique. The first experiment example was by using a hardware fault injector. The authors injected bit Errors into the physical layer of a LAN link. The second experiment example was a fault injector of Debugger-Based by using a framework of the real space imaging. NFTAPE is flexible with an ability to execute Simulated-Based Fault Injector, SWIFI, or Hardware-Based Fault Injector.

In evaluation of commercial scanners, Fonseca et al. [6] evaluated web pages' vulnerabilities by using commercial scanners, and injection software faults into websites. The authors compare the efficiency of many different commercial scanners in the detection of the vulnerabilities. By analyzing the False Positive (FP) and the True Positive (TP), the authors could evaluate three different types of the commercial vulnerability scanners to decide which scanner has better ability to detect code injection vulnerabilities in the web applications. The result shows that the scanners have different results but the three of them obtained high False Negative (FN) and False Positive (FP) percentage values (FP is in between 20% to 77%). In future, the authors want to evaluate the same method to different web applications that have not been subjected to this method, and then compare many other approaches to get precise experimental results. They intend studying on type of programming code in the web application leads to code injection vulnerabilities in order to further support the prevention of code injection vulnerabilities.

4. PROPOSED MODEL

In this section, we present our model for checking the website vulnerabilities to CIAs. Figure 4 illustrates five phases; phase 1: URL Collection, phase 2: Proposed Method, Phase 3: Dataset Testing, phase 4: Proposed Classifier, phase 5: Classifier Result. These phases are explained in separate subsections as follows:

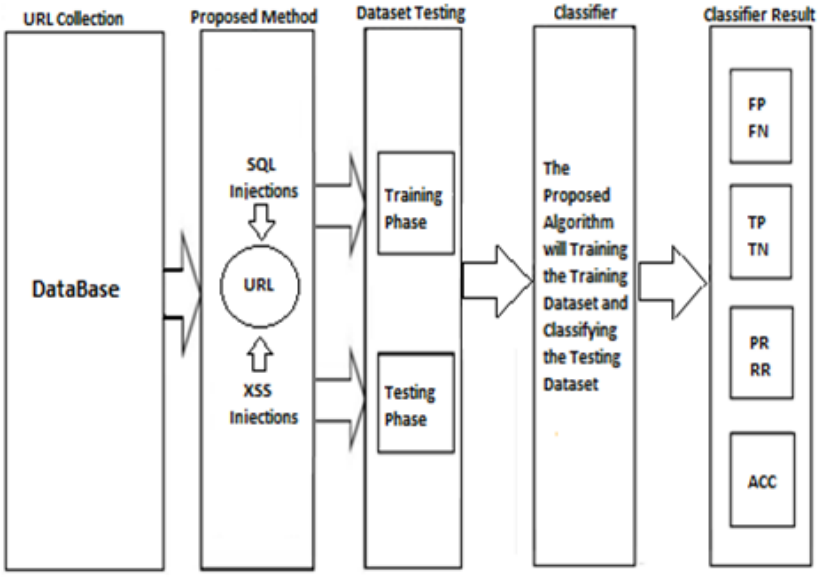

Fig. 4. The Components of Checking the Website Vulnerabilities of our Model

4.1 URL Collection

Initially, the dataset was collected from many different sources. After collecting the datasets, we sort them in the MySQL program as vulnerable and not vulnerable websites, in order to utilize them as training and testing datasets.

URLs from eleven different open sources are collected [13, 14, 15, 16, 17, 18, 19, 20, 21 and 25]. In addition, we collected the benign dataset from our previous research [10]. URL of the page is entered from the dataset to check if it is vulnerable to SQL and XSS injection attacks. The proposed model provides the URL links related with the URL link that has been entered.

4.2 Proposed Method

This phase mixes the vulnerable and not vulnerable websites together, and prepare them to create three training datasets. Proposed model is trained with the training datasets to check whether it works correctly or not, and verifies with each SQL and XSS code that is inject into every URL link in the training dataset.

The C# platform and code for checking the vulnerable website is as shows in Figures 5 and 6.

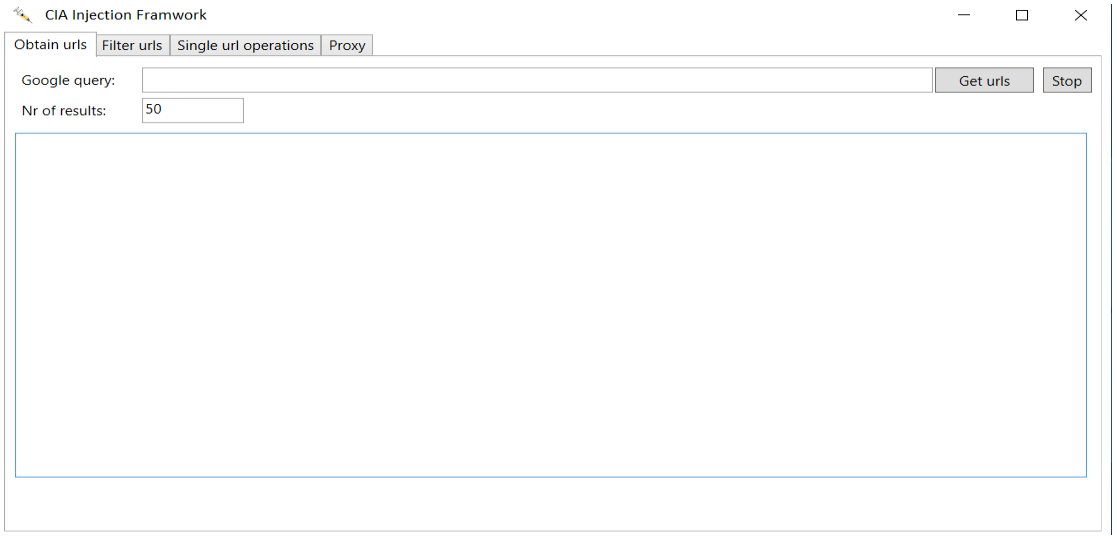

Fig. 4. The C# Platform of checking the website vulnerabilities

```
    int k = 0;
    for (k = 0; k < parts.Length; k++)
    {  if (parts[k].Contains("www"))
       {  fileName = parts[k].Remove(0, 4); }   }
       if (fileName.Length < 2)
    { fileName = parts[2];   }
        return fileName;       }
System.IO.StreamWriter sw1;
private void btnCheckUrls_Click(object sender, RoutedEventArgs e)
{   string name = GetFileName(txtSearchEngineUrl.Text);
    name = name + "_injections"+ ".csv"; sw1 = new System.IO.StreamWriter(name);  string[] injects = new string[10];
  Inject[0] = "'='or'"; Inject[1] = "OR 1=1-"; Inject[2] = "'OR"; Inject[3] = "%22sCrIpt%2Balert(%27XSS%27)%2B/sCrIpt%22"; Inject[4] = "OR '0'= '0'"; Inject[5] = "') OR ('0' = '0'"; Inject[6] = "'UNION'
  ;alert(%26quot;XSS%26quot;)>"; Inject[12]= "%22%20Style %3D%22BackGround:url (JavaScript: Alert (%27xss %27))%22%200S%22"; Inject[13] = "%22%2BAlert(%27XSS%27)%2B%22"; Inject[14] =" <table background="
    IList<string> vulnerableResults = new List<string>();
    IList<string> urlsToCheck = new List<string>();
    string[] separators = new string[] { Environment.NewLine };
    IList<PatternDetails> patterns = new List<PatternDetails>();
    string urlBatch = txtUrls.Text;  btnCheckUrls.IsEnabled = false; txtProbablyVulnerableUrls.Clear();
    bool possiblyVulnerable = false;
    System.IO.StreamWriter tempSW;
      var th = new Thread(() =>
    { var queryRunner = new SimpleQueryRunner();
       if (!string.IsNullOrEmpty(urlBatch))
          urlsToCheck = urlBatch.Split(separators, StringSplitOptions.RemoveEmptyEntries).ToList();
         foreach (var url in urlsToCheck)
       {   if (_stopCurActionFilterUrlsTab == true)  break;
          possiblyVulnerable = false;
          IList<string> possiblyVulnerableUrls = Seringa.Engine.Utils.UrlHelpers.GeneratePossibleVulnerableUrls(url);//TODO:multiple possible vulnerable urls
          foreach(var possiblyVulnerableUrl in possiblyVulnerableUrls)
          { string pageHtml = string.Empty;
                    try { pageHtml = queryRunner.GetPageHtml(possiblyVulnerableUrl, null); }
             catch (Exception ex)        {        }
             patterns = XmlHelpers.GetObjectsFromXml<PatternDetails>(FileHelpers.GetCurrentDirectory() + "\\xml\\patterns.xml", "pattern", null);
             string pats = "";
             foreach (var pattern in patterns)
             {    if(pattern != null && !string.IsNullOrEmpty(pattern.Value))
                if(pageHtml.IndexOf(pattern.Value) > -1)
                {   possiblyVulnerable = true;
                   pats+=pattern.Dbms+"," + pattern.Value+",";   break;      }          }
             if (possiblyVulnerable)
             {   string name10 = "";
                for (int k=0; k< injects.Length; k++)
                {   name10 = possiblyVulnerableUrl + injects[k];
                    try {   pageHtml = queryRunner.GetPageHtml(name10, null);
                if(pageHtml.Length> 0)       {
                          System.Console.WriteLine("injecttion succeeded....." + name10);  }
```

Fig. 5. The C# code of checking the website vulnerabilities

If the code injection succeeded, proposed model would not obtain an Error in related to the code injection method. If an Error is notified, the code injection method did not succeed, so the website is not vulnerable to CIA.

4.3 Dataset Testing

In the Dataset Testing phase, various datasets are created for both training and testing phases to utilize them in the Proposed Classifier. The sizes of the Training Dataset samples are 297, 311, and 530. The sizes of the Testing Dataset samples created by using MySQL program are 847, 1084, and 1212.

4.4 Classifier

Proposed Classifier receives the URL link from the dataset, opens the webpage of the URL and checks whether it is vulnerable to CIA t by injecting both XSS and SQL syntaxes automatically into the website. Datasets can be checked and classified with the proposed model. XSS and SQL syntaxes are collected from different resources. These syntaxes are the most common injections that have been used by hackers to attack web applications. Next subsection explains the process in detail.

The existing classifications in the literature are similar to the proposed classification, except existing classifications did not build a model to inject XSS and SQL syntaxes on the website, worked on commercial scanners and did XSS and SQL code injections manually.

4.5 Classifier Result

The last phase is the Classifier Result. In this phase, the proposed model informs whether websites in the dataset are vulnerable to CIAs. If the intrusion of the URL link succeeds, then the model notifies that the website is vulnerable to CIAs. Otherwise, if an Error is notified, it means the intrusion did not succeed and the webpage is not vulnerable to CIAs.

Results of the Proposed Classifier are collected and determine the output result which consists of the precision rate (PR), the recall rate (RR), the false positive (FP), the false negative (FN), the true positive (TP), the true negative (TN), and the accuracy (ACC).

5. METHODOLOGY

Proposed model differs from previous research using existing tools, injects code automatically to check whether the web application has a vulnerability to CIAs.

The structure of the proposed model is as follows: The first step is to enter the URL of the page to check vulnerability to SQL injection and XSS attacks. Model gives the URL links related with the URL link that has been entered. The basic idea is to find the URL links that contain the databases of the main web page entered for vulnerability checking in the model.

In the second step, vulnerabilities of URL checks links by injecting SQL and XSS code in every URL. In this step, the SQL and XSS scripts are as follows:

Inject[0] = "'="or'".
Inject[1] = "OR 1=1–".
Inject[2] = "'OR".
Inject[3] = "%22sCrIpt%2Balert(%27XSS%27)%2B /sCrIpt%22".
Inject[4] = "OR '0'= '0'".
Inject[5] = "') OR ('0' = '0'".
Inject[6] = "'UNION'".
Inject[7] = "'WHERE'".
Inject[8] = " ‘ ".
Inject[9] = “""<script>alert(‘XSS')</script>”.
Inject[10] = “%22%27><img%20src%3d%22javascript:alert(%27XSS%27) %22>”.
Inject[11]= “ '" < img%20src%3D%26%23x6a;%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23x69;%26%23x70;%26%23x74;%26%23x3a;alert(%26quot;XSS%26quot;)>”.
Inject[12]= “%22%20Style%3D%22BackGround:url (JavaScript: Alert (%27xss %27))%22%20OS%22”.
Inject[13] = “%22%2BAlert(%27XSS%27)%2B%22”.
Inject[14] =” <table background="javascript:alert(([code])"></table>”.

Inject[15] = “<object type=text/html data="javascript: alert(([code]);"></object>”.
Inject[16] = “<body onload="javascript:alert(([code])"> </body>”.
Inject[17] = “waitfor delay '0:0:5'”.
Inject[18] = “SHUTDOWN;” [26].

In the final step, if the code injections succeed, model will not obtain an Error about the code injection method, thus indicating vulnerability. If an Error is notified, the code injection method did not succeed, so the website is not vulnerable to CIA. Testing dataset for checking website vulnerabilities are taken from eleven different resources and from real websites.

The methodology of the research is different than other research methodologies have been previewed in the literature review. Proposed methodology depends on checking vulnerabilities of the web applications by injecting XSS and SQL code automatically to a website, whereas others depend on commercial vulnerability scanners, or building a protection model for the web application against CIAs.

6. EXPERIMENTAL SETUP

Vulnerable datasets are collected for experiment from ten different open sources [13, 14, 15, 16, 17, 18, 19, 20, 21 and 25]. In addition, the benign datasets are from our previous research [10].

Proposed model is built using C# programming language and MySQL server to build dataset. Every data line of dataset code is checked automatically.

6.1 Proposed Algorithm

In this subsection, the algorithm for web page vulnerability checking is presented. The algorithm is as shown below.

Algorithm 1. Web Page Vulnerability Checking Algorithm

Step one:
Enter the potentially vulnerable URL link in the text box of the search button of our program.

Step two:
Obtain the URL of web pages in related to the URL link that has been entered in the Text-Box of the search button.

Step three:
From step two, our model obtains URLs of pages that may contain the database of the website’s link that we have previously entered for vulnerability checking.

Step four:
Press on the start checking button to check the URL’s links, and start injecting SQL and XSS code automatically.

Step five:
The model injects XSS and SQL scripts as follows:
Inject[0] = "'="or"'.
Inject[1] = "OR 1=1–"…etc.

Step six:
If the code injection attack succeeds, our model will not give any Error about the code injections, indicating vulnerability. If an Error is notified, the code injection did not succeed, so the website is not vulnerable.

6.2 Algorithm Evaluation

Proposed algorithm has been evaluated with three different datasets. Three datasets from eleven different data sources are used. The sizes of the datasets 1, 2, and 3 are:

78, 75, and 106 respectively. Table 2 represents three different datasets of dataset A; the highest value of the accuracy result is 0.9622 in dataset A, 3.

TABLE II. THE RESULT TABLE OF DIFFERENT DATASETS THAT CLASSIFIED BY THE MODE

Data set A	**TP**	**TN**	**FP**	**FN**	**TNR**	**PR**	**RR**	**Accuracy**
Data set 1	36	37	3	2	0.9250	0.9230	0.9473	0.9358
Data set 2	31	41	2	1	0.9534	0.9393	0.9687	0.9600
Data set 3	42	60	3	1	0.9523	0.9333	0.9767	0.9622
Average	**36.333**	**46.00**	**2.66**	**1.33**	**0.9435**	**0.9318**	**0.9642**	**0.9527**

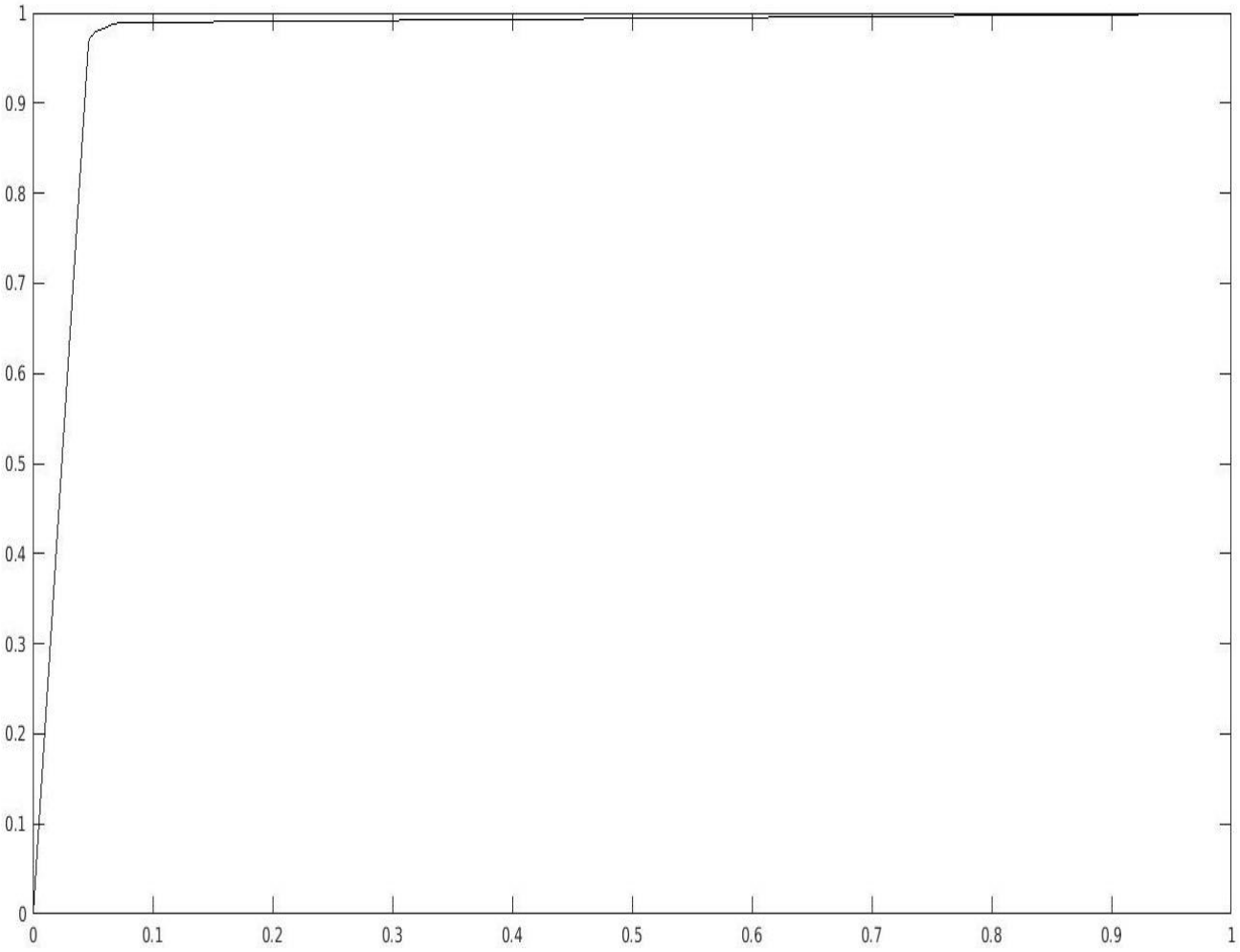

Fig. 6. The ROC Diagram of the Result of the Dataset A

Figure 6 shows the Receiver Operating Characteristic (ROC) curve for the performance of the proposed results. The ROC curve shows the relationship between two characteristics: The True Positive Rate (TPR) and False Positive Rate (FPR) of the dataset A. According to Figure 5, the high accuracy values interpreted by the curve represent the relationship between the recall rate (RR), and the false positive rate (FPR).

7. COMPARISON WITH OTHER APPROACHES

This section explains comparison of the results with those of previous approaches using different methods

7.1 Training Security team

Fonseca et al. (2008) detected web vulnerabilities against CIAs by code injection. The changes on the code of the webpage do not affect the running of the webpage, and it runs without any Errors in the execution [5]. Table 3, below, illustrates how one of these changes is executed, depending on which code is surrounding the function of the code.

TABLE III. CHANGES IN THE CODE AND THE FUNCTION OF THE CODE

	Functions of the code	The code	Example
1	If the function is used in an assignment as the only line of code and the variable is not inside a $_Get, $Http_Get_Vars, $_Post, $Http_Post_Vars PHP Variable Array	We omit the whole line of the code.	omit the line “$vuln_var = interval($vuln_var);”;
2	If the function is used in an assignment as the only line of code and the variable is inside a $_Get, $Http_Get_Vars, $_Post, $Http_Post_Vars PHP Variable Array	We just omit the function from the code, Leave the argument without changing.	replace “$vuln_var = interval($_GET[‘vuln_var’]);” With “$vuln_var = $_GET[‘vuln_var’];”;
3	If the variables is inside, the $_Get, $Http_Get_Vars, $_Post, $Http_Post_Vars PHP Variable Array	We Leave the variables in the code, only the function is omitted.	replace “…“’str1’.intval($vuln_var). ’str2’”;” With …“’str1’.$vuln_var.’str2’”;”.

The results show that the integer variables ("int" as in C language) are the main code injection vulnerable target. Those variables have been collected from subtype A and are responsible for 45% of all the vulnerabilities found. Code checks “int" variables’ values for non integer values such as “Display” and “Assign”. The variables that contain integer values can be a target for the penetration of code injection attacks; and a penetration attack can succeed by adding syntax to them. For example:

1. Penetration of SQL injection attacks, for example: “or 3=3”, “or ’D’=’D’”.
2. B- Penetration of XSS injection attacks, for example: “<scrIpT>Alert(’alert xss attack 1’)</scrIpT>”.
3. C- Adding at the begin or at the end of the variables, for example: [”], [( ] ,[ )] , [>], [<] ,[’].

The existing codes used commercial vulnerability scanners for web pages to check if the commercial scanners can detect the vulnerabilities and assist the security penetration teams to interpret the results. HP-WEBINSPECT, IBM WATCHFIRE-APPScan scanners are also used and named them as Q1 and Q2, in order to make them anonymous because the commercial license does not allow the publication of evaluation of the commercial scanner.

7.1.1 The Test Results:

The precision rate of Commercial Vulnerability Scanners is twenty percent. The precision rate of the security teams after the basic training period is 30%, while the precision rate of the security teams after the specific training period is 70% [5].

7.1.2 Code Inspection Results:

The precision rate of the security teams after the basic training period is around 56%, whereas the precision rate of our model is 93.36%, and the recall rate of our model is 98.26%.

The limitation is that the authors did not build a model to check the vulnerability tested CIAs. They tested commercial vulnerability scanners, and trained Security Teams to detect the vulnerability against code injection attacks in the web pages.

7.2 Web Application Protection (WAP) implementation

Medeiros et al. (2014) presented a methodology to protect web pages. Their methodology depends on analysing the source code of the web pages to detect the code injection vulnerabilities and modify the source code to hackers from successfully penetrating the webpage [9]. Web sites are the biggest source of problems from a security point of view with reports indicating an increase of web site attacks in 2012 of around 33%.

The main reason for an increase in web page attacks is that many users do not have enough knowledge on secure web pages. The contribution of this research for web application security is that programmers can learn from their mistakes by practicing finding the code injection vulnerabilities and fixing them.

The authors use a static analysis mechanism with a hybrid method to detect code injection vulnerability, which gives an effective way to detect the code injection vulnerabilities in the source code, but it may report many false positives.

Design and implementation of Web Application Protection (WAP) includes analysing the PHP programs. An attacker enters the web page by using entry points, such as $ GET, then penetrates the vulnerability using a database query language such as MySQL. Many attackers use benign inputs with meta-characters or meta-data such as ', OR. WAP can protect the websites by adding sanitization functions between the entry points ($ GET) and the database query language (MySQL).

Table 4 shows the sanitization functions that have been used to remove the SQL injection vulnerabilities, by using the PHP as a programming language.

TABLE IV. SANITIZATION FUNCTIONS USED TO FIX PHP CODE FOR SQLI VULNERABILITIES

Vulnerability	**Entry Points**	**Sensitive Sinks**	**Sanitization Functions**
SQL injection vulnerability	1. $Get 2. $Post 3. $Cookie 4. SQLI-$-REQUEST 5. HTTP-Get-VARS 6. HTTP-Post-VARS 7. HTTP-Cookie-VARS 8. HTTP-Request-VARS	mysql-Query mysql-unbuffered-Query mysql-db-Query	mysql-real-escape-String
		mysqli-Query mysqli-real-Query mysqli-master-Query mysqli-multi-Query	mysql-real-escape-String
		mysqli-stmt-execute mysqli-execute	mysqli-stmt-bind-Param

CIA on the web page was launched, and checked whether the result was false positive or true positive. Source code of the web page was viewed to correct and validate it, the malicious attack of some types of meta-characters can be stopped.

Existing research consists of two model tools, nine open-source frameworks, and the PHP code from National Institute of Standards and Technology (NIST). The SAMATE dataset is from [http://samate.nist.gov/SRD/].

The WAP detector detected 68 vulnerabilities (22 to SQL injection attacks and 46 to XSS attacks), and 21 false positives. Pixy detected seventy 3 web page vulnerabilities to CIA (20 to SQL injection attacks and 53 to XSS attacks), 41 false positives and 5 false negatives. WAP-TA detected 5 web page vulnerabilities to CIA, while the other detectors could not detect them.

The Pixy's accuracy is around 44%, while WAP-TA's accuracy is around 69%, and the WAP's accuracy is around 92%. The limitation of these models is the high false positive rate which is 20%. In contrast, the accuracy result of our model is 95.97%, the false positive rate is 5.55%, and the true positive is 98.11%.

8. CONCLUSION

This paper has proposed a model to check vulnerable websites to CIAs. The proposed approach provides significant performance in terms of precision compared with similar existing work. The proposed algorithm detects various types of CIAs such as SQL injection attacks, and cross site scripting attacks (XSS). The dataset in this research is built for executing, testing and measuring the performance of the model. The accuracy result of our model is 95.27%, the precision rate is 93.18%, and the recall rate is 96.42%. Existing research have been adopted only for checking the vulnerability of web pages by using commercial applications downloaded from the Internet, or by checking and encoding some locations in the source code of the web page to sanitize the web page from code injection vulnerabilities. In contrast, the proposed methodology is injecting SQL injection meta-codes and XSS meta-codes automatically using proposed model to validate whether the web page is vulnerable to CIA. It achieves greater accuracy compared to another research.

Conflicts of Interest

The author's disclosure statement confirms the absence of any conflicts of interest.

Funding

The author's paper clearly indicates that the research was conducted without any funding from external sources.

Acknowledgment

The author extends appreciation to the institution for their unwavering support and encouragement during the course of this research.

References

[1] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in Proc. IEEE Symp. Security and Privacy , May 2010, pp. 332-345.
[2] J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & attack injection for web applications," in Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. , June 2009, pp. 93-102.
[3] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using attack injection to discover new vulnerabilities," in Proc. Int. Conf. Dependable Syst. Netw. (DSN'06) , June 2006, pp. 457-466.
[4] J. Fonseca and M. Vieira, "Mapping software faults with web security vulnerabilities," in Proc. IEEE Int. Conf. Dependable Syst. Netw. (DSN) , June 2008, pp. 257-266.
[5] J. Fonseca, M. Vieira, H. Madeira, and M. Henrique, "Training security assurance teams using vulnerability injection," in Proc. 14th IEEE Pacific Rim Int. Symp. Dependable Comput. , Dec. 2008, pp. 297-304.
[6] J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks," in Proc. 13th Pacific Rim Int. Symp. Dependable Comput. (PRDC 2007) , Dec. 2007, pp. 365-372.
[7] D. T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. K. Iyer, "NFTAPE: A framework for assessing dependability in distributed systems with lightweight fault injectors," in Proc. IEEE Int. Comput. Perform. Dependability Symp. (IPDS 2000) , Mar. 2000, pp. 91-100.
[8] P. Bhojak, V. Shah, K. Patel, and D. Gol, "Automated Web Application Vulnerability Detection With Penetration Testing," in ICRISET2017 , vol. 2, 2017.
[9] I. Medeiros, N. F. Neves, and M. Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives," in Proc. 23rd Int. Conf. World Wide Web , Apr. 2014, pp. 63-74.
[10] H. Alnabulsi, M. R. Islam, and Q. Mamun, "Detecting SQL injection attacks using SNORT IDS," in Proc. Asia-Pacific World Congr. Comput. Sci. Eng. , Nov. 2014, pp. 1-7.
[11] Websmart Inc., "100000 Vulnerable Websites," Sep. 2017. [Online]. Available: https://samsclass.info/125/ethics/smart-websites.htm
[12] H. Alnabulsi and M. R. Islam, "Identification of susceptible websites from code injection attack," in Proc. 1st Int. Conf. Mach. Learn. Data Eng. (iCMLDE 2017) , 2017, pp. 1-9.
[13] E. Bertino, L. D. Martino, F. Paci, and A. C. Squicciarini, "Web services threats, vulnerabilities, and countermeasures," in Security for Web Services and Service-Oriented Architectures, Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, pp. 25–44, 2009
[14] A. A. Sarhan, S. A. Farhan, and F. M. Al-Harby, "Understanding and discovering SQL injection vulnerabilities," in Proc. Int. Conf. Applied Human Factors and Ergonomics, Cham, Switzerland: Springer International Publishing, pp. 45–51, 2017
[15] R. A. Grimes, Hacking the Hacker: Learn from the Experts Who Take Down Hackers. Hoboken, NJ, USA: John Wiley & Sons, 2017
[16] M. S. Aliero, I. Ghani, K. N. Qureshi, and M. F. A. Rohani, "An algorithm for detecting SQL injection vulnerability using black-box testing," J. Ambient Intell. Humaniz. Comput., vol. 11, no. 1, pp. 249–266, 2020
[17] "150 SQL Vulnerable Websites 2017 List," Aug. 2017. [Online]. Available: https://www.scribd.com/document/325850327/150-SQL-Vulnerable-Websites-2017-List
[18] "List Of web sites Vulnerable For SQL Injection," Aug. 2017. [Online]. Available: http://listsql.blogspot.com.au/
[19] "20 famous websites vulnerable to cross-site scripting," Aug. 2017. [Online]. Available: http://thehackernews.com/2011/09/20-famous-websites-vulnerable-to-cross.html
[20] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," in Proc. 18th Int. Conf. Advanced Information Networking and Applications (AINA 2004), vol. 1, pp. 145–151, 2004.
[21] A. K. Kassem, Intelligent System Using Machine Learning Techniques for Security Assessment and Cyber Intrusion Detection, Ph.D. dissertation, Université d'Angers, Angers, France, 2021.
[22] A. Avancini and M. Ceccato, "Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities," in Proc. IEEE 11th Int. Working Conf. Source Code Analysis and Manipulation (SCAM), pp. 85–94, 2011.
[23] M. Mohammadi, B. Chu, and H. R. Lipford, "Detecting cross-site scripting vulnerabilities through automated unit testing," in Proc. IEEE Int. Conf. Software Quality, Reliability and Security (QRS), pp. 364–373, 2017.
[24] C. S. Cheah and V. Selvarajah, "A review of common web application breaching techniques (SQLi, XSS, CSRF)," in Proc. 3rd Int. Conf. Integr. Intell. Comput. Commun. Security (ICIIC 2021) , Sep. 2021, pp. 540-547.
[25] A. Javed, "Revisiting XSS Sanitization," in Proc. ISACA Ireland Conf. 2014 , 2014.
[26] H. Alnabulsi, R. Islam, and M. Talukder, "GMSA: Gathering Multiple Signatures Approach to Defend against Code Injection Attacks," IEEE Access, pp. 1-12, Nov. 2018.
[27] H. Alnabulsi, R. Islam, and Q. Mamun, "A novel algorithm to protect code injection attacks," in Proc. Springer Int. Conf. Appl. Techn. Cyber Security Intell. (ATCSI) , vol. 580, Springer China, pp. 281-292, 2018.
[28] S. K. Iqbal et al., "Real-time-based E-health systems: Design and implementation of a lightweight key management protocol for securing sensitive information of patients," Health Technol. , vol. 9, pp. 93-111, 2019.
[29] A. H. Mohsin et al., "Based medical systems for patient's authentication: Towards a new verification secure framework using CIA standard," J. Med. Syst. , vol. 43, no. 7, art. 192, 2019.

[30] M. L. Shuwandy et al., "mHealth authentication approach based 3D touchscreen and microphone sensors for real-time remote healthcare monitoring system: comprehensive review open issues and methodological aspects," Comput. Sci. Rev., vol. 38, art. 100300, 2020.
[31] A. Alamleh et al. , "Federated learning for IoMT applications: A standardization and benchmarking framework of intrusion detection systems," IEEE J. Biomed. Health Inform. , vol. 27, no. 2, pp. 878-887, 2022.
[32] M. Abuhamad, A. Abusnaina, D. Nyang, and D. Mohaisen, "Sensor-based continuous authentication of smartphones' users using behavioral biometrics: A contemporary survey," IEEE Internet Things J., vol. 8, no. 1, pp. 65–84, 2020.