REAL TIME XSS DETECTION: A MACHINE LEARNING APPROACH

Implemented in Python 3.6

By,

Mohamed Yilmaz Ibrahim

BE (Comp.Sci) - II Year

Anna University CEG

Under Summer Research Fellowship

Indian Academy of Sciences, Bangalore

mohamedyilmaz98@gmail.com

Under the Guidance of,

Dr B.M. Mehtre

Professor, Centre for Cyber Security

Indian Institute for Development and Research in Banking Technology, Hyderabad

bmmehtre@idrbt.ac.in

June 2017

DECLARATION

I hereby declare that the project entitled "Real Time XSS Detection: Machine Learning Approach" submitted as part of the partial course requirements for the Summer Research Fellowship Programme for the award of the degree of Project Trainee in Cyber Security at the Indian Institute for Development and Research in Banking Technology during May-June (2017) has been carried out by me. I declare that the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles elsewhere.

Further, I declare that I will not share, re-submit or publish the code, idea, framework and/or any publication that may arise out of this work for academic or profit purposes without obtaining the prior written consent of my Guide.

Signature,

Mohamed Yilmaz Ibrahim

Place: IDRBT, Hyderabad
Date:30/06/2017

ABSTRACT

Due to the rapid growth of the internet, websites have become the intruder's main target. An intruder embeds malicious contents in a web page especially JavaScript for the purpose of doing some bad and unwanted-activities such as: credential information and resource theft, luring a user to visit a dangerous website, downloading and installing software to join a botnet or to participate in distributed denial of service, and even damage the visitor system. As the number of web pages increase, the malicious web pages are also increasing and the attacks are becoming more and more sophisticated. Thus, it becomes an imperative to detect such malicious code in real time before any malicious activity is performed.

Three conventional methods have been used for the detection of Malware: Signature-based, Behavioural based and Heuristic based. Though these methods were quite efficient in malware detection, they weren't effective in generating results at quick time which lead to the need for a new malware detection method. Researchers have recently enlisted machine learning in the detection of malware. The advantage of using machine learning is that it can determine whether a code or a file is malicious or not in a very small amount of time without the need of isolating it in a sandbox to perform the analysis. It is also useful for the detection of increasingly polymorphic nature of malware. Machine learning classifiers can then be used in detection of malicious JavaScript code through learning from patterns. In addition to the significant detection rate, my objective is to also find which discriminative features characterize the attack and reduce the false positive rate.

This study proposes an efficient method of detecting previously unknown malicious java scripts using an interceptor at the client side by classifying the key features of the malicious code. The algorithm is based on three features group, the URL lexical, page content and real-time traffic features. A completely automated tool has been developed using Python 3.6 which scans for potential URL lexical, page content and also the real-time traffic features where the XSS payloads can be injected. The interceptor monitors the traffic exchanged between the browser and server and extracts the real-time traffic features needed for XSS detection. Supervised machine learning classifiers such as k-NN, Support Vector Machines, Gaussian Naïve Bayes and Logistic Regression are used on a dataset gathered from [14] for achieving high accuracy in webpage classification through learning from patterns. The experimental results obtained, show that this method can efficiently classify malicious code from benign code with promising results. Thus, the proposed features lead to highly accurate classification of malicious web pages and also the high false positive rate which was produced by the conventional machine learning approaches has been significantly reduced.

Table of Contents

List of Figures

List of Tables

XSS-DETECT

A REAL TIME XSS DETECTION TOOL

Introduction

The internet has become an essential in our daily life; it's the base of banking transactions, shopping, entertainment, resource sharing, news, and social networking. Accessing web applications has become a daily routine for many people. We depend on these applications to accomplish transactions, be it business, personal or otherwise. We interact dynamically with web applications when we access our emails, conduct banking transactions, visit social networking sites, etc. This dynamic nature of the web applications allows users to input information that will determine how a web site responds to the user. In many web sites, these user inputs are not properly validated thus making such a site vulnerable to cross-site scripting (XSS). The attack occurs when a user visits a suspected website, therefore attacker focuses on a web site that has become the centre of attention, and then exploits the vulnerabilities in both client and server-side to launch the attack. Web attack is a challenge; it necessitates a careful understanding of the details and the behavior.

Internet browsers support active client-side content with many techniques, among them JavaScript, which is the most widely used as a primary component of active and dynamic web content, while it provides a great chance for exploits. Further techniques such as PHP language, adobe flash, and visual basic script in common have capability of downloading and executing code from the Internet. Most of the web programming languages do not provide, by default, a safe data transfer to the client. The absence of this procedure can lead to one of the most frequent attacks in web applications, the Cross-Site Scripting (XSS).

Cross-site scripting (XSS) is currently rated as the 3rd most dangerous vulnerability among the Open Web Application Security Project (OWASP) top ten vulnerabilities. As per a report released by Symantec in 2017 [1], the use of JavaScript downloaders in Office files was widespread and accounted for just over 7 million attempted infections in 2016. XSS flaws occur whenever an application includes untrusted data in a new web page without proper input sanitization or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. XSS is a type of injection attack in which malicious scripts are injected around benign code in a legitimate webpage in order to access cookie, session, and other secret information [2]. *Fig 1.0 shows an overview of how attackers inject malicious JavaScript in webpages and steal the session cookie of the victims, which can be used to get their account credentials.* The tool checks for the existence of malicious JavaScript code in the webpage and displays the result whether the page is XSS infected or not. It doesn't provide any solution on how to correct the malicious JavaScript code.

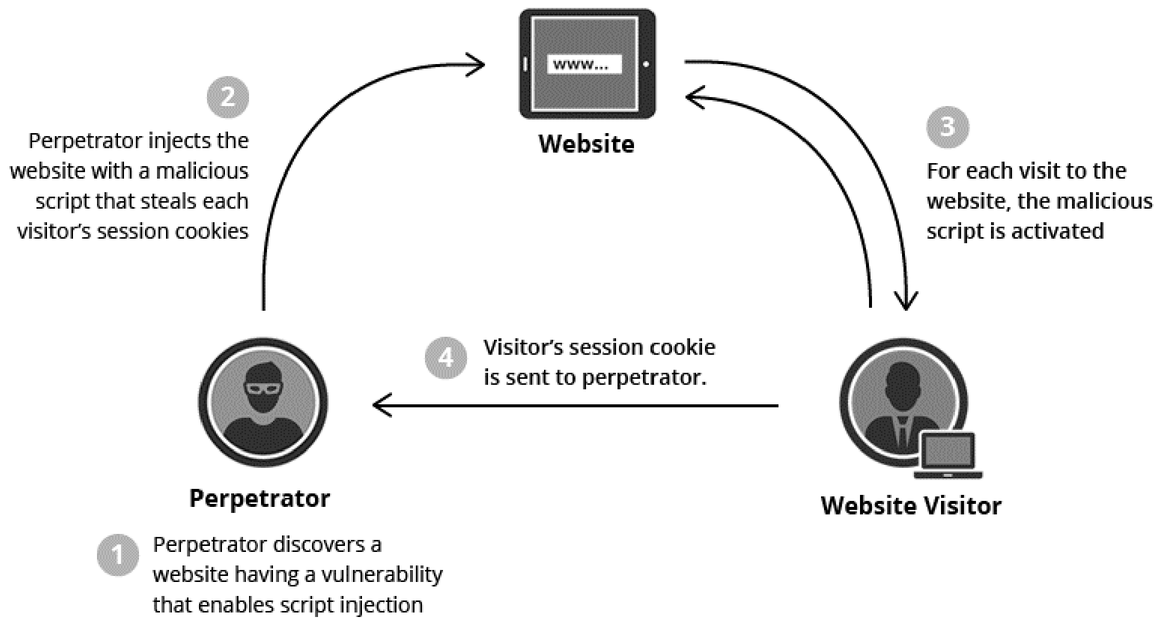

www...
Website
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Fig 1.0: Cross Site Scripting Overview

Understanding Cross-Site Scripting

Grossman [3] defines Cross-Site Scripting (XSS) as an attack vector caused by malicious scripts on the client or server, where data from user input is not properly validated. This allows the theft of confidential information and user sessions, as well as it compromises the client's browser and the running system integrity. The script codes used in XSS are typically developed in JavaScript and embedded in the HTML structure. However, technologies such as Active X, Flash, VBScript or any other technology supported by browsers can also be used as a vector.

But M. Van Gundy and H. Chen [4] have mentioned that the target of XSS attack is a client side whereas SQL injections target server-side [4]. XSS attack is a vulnerability at the application layer of network hierarchy, which occurs by injecting malicious scripts to break the security mechanism. *Fig 2.0 shows how Cross-site Scripting attack is done.* About 70% attacks are reported to occur at application layer. Web browsers are the most susceptible application layer software for attacks. The purpose of the web browser is to get the requested web resource from the server and display it in the browser's windows. The format of the supplied resources is not restricted to HyperTextMarkup Language (HTML) but can also be portable document format (PDF), image, and so on. Attackers run malicious JavaScript in a web browser to target users. Malicious and obfuscated URLs also serve as a carrier for XSS attacks. JavaScript is a scripting language, used in web programming for making web pages more interactive, adds more features, and improves the end user experience. JavaScript also helps in reducing the server-side load and helps in shifting some computation to end user side. The JavaScript is commonly used for client-side scripting to be used in web browsers. When the user requests for a certain web page through a web browser, the server responds to the request and sends back web page which might have JavaScript embedded into it. Then the JavaScript interpreter inside the web browser at the user's side starts executing it directly and automatically unless explicitly disabled by the user. Unfortunately, JavaScript's code provides a strong base for conducting attacks, especially drive-by-download attacks, which unnoticeably attack clients amid the visit of a web page. In contrast to different sorts of network-based attacks, malicious JavaScripts are difficult to detect. JavaScript attacks are performed by looking into the vulnerability and exploiting by using JavaScript obfuscation techniques to evade the detection. Thus, detecting of such malicious JavaScript's attacks at the real time becomes imperative. Malicious and obfuscated URLs are also carriers for XSS attacks.

XSS attacks are of three types namely reflected, stored and DOM-based [5]. Reflected XSS is executed by the victim's browser and occurs when the victim provides input to the web site. Stored XSS attacks store the malicious script in databases, message forums, comments fields, etc. of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker. Both reflected and stored XSS are executed on the server side. On the other hand, DOM-based XSS attacks are executed on the client side. Attackers are able to collect sensitive or important information from the user's computer.

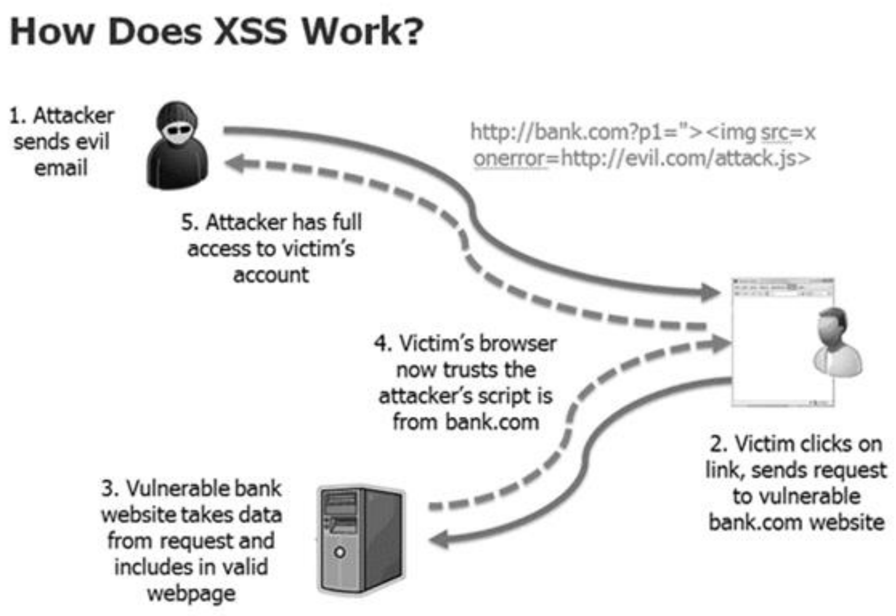

How Does XSS Work?
1. Attacker sends evil email
http://bank.com?p1="><img src=x onerror=http://evil.com/attack.js>
5. Attacker has full access to victim's account
4. Victim's browser now trusts the attacker's script is from bank.com
2. Victim clicks on link, sends request to vulnerable bank.com website
3. Vulnerable bank website takes data from request and includes in valid webpage

Fig 2.0: Working of XSS Attack

Literature Review/Related Work

Existing detection approaches can be classified into three categories: signature-based, behavioural/heuristic-based and machine learning. They are grouped in three categories:

- URL and Domain Analysis
- Page Content Analysis
- Behaviour Analysis

Why Machine Learning Approach?

Security solution providers who play a vital role in providing tools for defending attacks are fundamentally based on two correlative approaches, signature-based and heuristic-based detection approaches. The signature-based approach is based upon the detection of unique string patterns in the binary code. Signature-based attack detection approach fails when a new type of attack occurs. It is easy to fool signature-based solutions by changing the ways in which an attack is made. The more advanced the signature database, the higher the CPU load for the system charged with analysing each signature. Inevitably, this means that beyond the maximum bandwidth packets may be dropped. So, feeds may have to be split and then recombined after analysis, increasing complexity and cost. In addition, it means that the greater the number of signatures searched for, the higher the probability of identifying more false positives. Consequently, this approach is unacceptable in an environment where new attacks are expected to come. Another tradition approach for attack detection is heuristic-based detection. Heuristic-based detection is based upon the set of expert decision rules to detect the attack. The advantage of using this attack approach is that it cannot only detect modified or variant existing malware but can also detect the previous unknown attack. The downside of using this approach is that it takes a long time in performing scanning and analysis, which drastically slows down the security performance. Another problem of the approach is that it introduces many false positives. False positive occurs when a system wrongly identifies code or a file as malicious when actually it is not. Some of the heuristic approaches include file emulation, file analysis, and generic signature detection. [6]

Machine Learning is able to detect previously unknown malware with predictive capabilities and especially useful for the detection of increasingly polymorphic nature of malware. To perform the classification of a benign and a malicious code, machine learning classifiers are used in detection through learning from patterns.

URL and Domain Analysis

Ma et.al [7] introduced malicious web site detection model based on URL lexical and host-based features using online classifiers algorithms, the model was able to achieve 99% accuracy.

Yoshiro Fukushima et.al [8] analyzed characteristics of malicious web sites by their domain information. They studied the IP address, IP address, domain, and registrar with reputations, and then they proposed a blacklisting scheme derived from the combination of IP address and registrars. A hierarchical structure graph is constructed using the extracted information.

Paul Prasse et.al [9] introduced a malware-detection method recently where the detection of malware on client computers is done based on HTTPS traffic analysis. In their setting, malware is detected based on the host IP address, ports, timestamp, and data volume information of TCP/IP packets that are sent and received by all the applications on the client based on a neural networks and sequence classification.

Page Content Analysis

Mitsuaki Akiyama et.al [10] have proposed a model for analysing the eco-system of malicious URL redirection through longitudinal observation from honeypots. They found out that although the main purpose of URL redirection caused by redirect code injections had been drive-by download-based malware infection, click-fraud has become a new purpose in addition to malware infection in recent years. They originally defined that URL redirection additionally includes automatically occurring web access to URLs corresponding to an initial accessed URL and the URL redirection methods such as the tag redirections (iframe, script, meta, etc.) , script redirections (JavaScript location's methods), and HTTP redirections (HTTP-3xx status code).

Rong Wang et.al [11] proposed a new detection method based on hybrid analysis that consists of static and dynamic analyses for detecting malicious web pages. Static analysis utilizes classification algorithms in machine learning to identify certain benign and malicious web pages. As a complement to static analysis, dynamic analysis mainly checks the unknown web pages to determine whether they have malicious shellcodes during their execution. Because of the combination of static and dynamic analyses, the proposed detection method achieves high performance, and it has a light weight and is simple to use.

Behavior Analysis

Xu et.al [12] proposed a model for detecting the infection delivered through vulnerable applications and web browser based on the dependency between a user’s action and system’s event.

Jayasinghe et.al [13] presented a novel approach to detect drive-by downloads in web browser environments using low resource dynamic analysis. By dynamically monitoring the bytecode stream generated by a web browser during rendering, the approach is able to detect previously unseen drive-by download attacks at runtime. The proposed method is effective, space efficient, and performs the analysis with low performance overhead, making the approach amenable to in-browser drive-by download detection on resource constrained devices, such as mobile phones. As a conclusion, the URL and domain analysis method is a very useful idea, but the limitation of this method is that the malicious URL does not mean that the corresponding page holds malicious contents and vice versa. Page content based faces a lot of challenges, such as continuous change of the attack features, Obfuscation and so on.

Junjie Wang, et.al. [14] considered the problem of not only detecting malicious JavaScript, but to also classifying it into a more fine-grained fashion. They considered the following eight classes of malicious JavaScript: attacks targeting browser vulnerabilities, browser hijacking attacks, attacks targeting Adobe Flash, attacks targeting JRE, attacks based on multimedia, attacks targeting Adobe PDF reader, malicious redirecting attacks, and attacks based on Web attack toolkits. The system uses HTML Unit to de-obfuscate the JavaScript as a pre-processing step, and then extracts features. The features are used to detect malware using a learned classifier, and if detected as malicious, it is then classified as one of the defined classes, or presented to the user as a potentially new class.

Al-Taharwa et al. in 2015 [15] proposed an approach for detection of deobfuscation of JavaScripts for detection of malicious JavaScripts.

Yao Wang et.al [16] presented a new deep learning framework for detection of malicious JavaScript code, from which they obtained the highest detection accuracy compared with the control group. The architecture is composed of a sparse random projection, deep learning model, and logistic regression. Stacked denoising auto-encoders have been used to extract high-level features from JavaScript code and logistic regression as a classifier was used to distinguish between malicious and benign JavaScript code.

Gupta et al. [17] proposed a method called XSS-SAFE for XSS attack detection and prevention based on automated feature injection statements and placement of sanitizers in the injected code of JavaScript. The main advantage of this method is that it can detect XSS attacks without any modification to client- and server-side commodities.

Chun et.al [18] introduced a cross-site scripting attacking detection method, which is based on improved algorithm skip list. It can fulfil the rapid detection of cross-site scripting attacks by using some steps, for example, creating a skip list signature, detecting suspicious strings, marking attack vectors and so on.

My work is based on extracting both static and dynamic features based on their quality, which is precisely analyzed to be used during the classification process. These features act as pivotal in initiating a JavaScript attack.

Problem Statement

Web applications have become a primary target for cyber criminals nowadays by injecting malware especially JavaScript to perform malicious activities for impersonation. The attacks on the users by exploiting the vulnerabilities of the browsers have increased at an alarming rate. The existing attack prevention strategies have failed miserably in most of the situations. Moreover, users have also not taken much care of configuring their browsers securely, using available extensions and plug-ins. The scripts are also executed automatically without the user's permission unless he disables it. This proposal puts forward an advanced XSS detection technique by introducing a new scoring system for privilege levels and vulnerability levels of the contents rendered in the browser. The java scripts rendered in the browsers are stored, classified, and analyzed using machine learning algorithms. Machine learning can also be used to predict the browser quirks and generate an attacker pattern. Thus, machine learning attempts to solve a user's problem in a quicker time than the conventional XSS detection techniques.

Objectives

To develop a solution that can effectively defend malicious JavaScript attacks (Cross Site Scripting attacks) using machine learning classifiers and also to find the discriminative features which characterize the attack and thus reduce the false positive rate.

What it contains?

The tool performs static analysis on the webpage source code and checks for any potential vectors where XSS payloads can be injected into the webpage. This was implemented using Selenium web driver module in Python 3.6. It also contains an interceptor which monitors the traffic exchanged between the browser and server and extracts the real-time traffic features needed for XSS detection. Static features such as the URL lexical and page content features and also the real-time features such as number of redirections, script size and the number of instantiated objects have been used.

Basic Framework

Sirageldin et.al [19] proposed a model which composes three components, Feature Extractor, Learning and Model Selector, and the Detector. It starts with the feature extraction process via feature extractor component, and then the extracted features are sent to the detector.

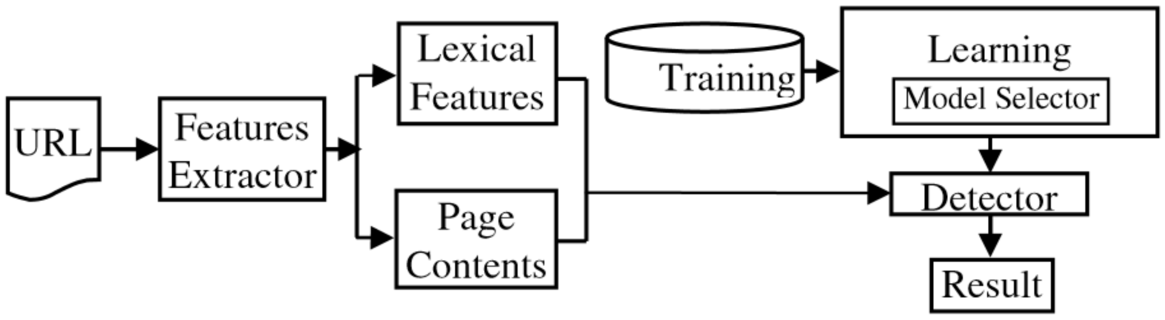


Fig 3.0: Methodology for detecting malicious webpages [19]

Feature Extractor

This is the first component, which is responsible for capturing the web page features. It consists of: BeautifulSoup HTML parser, the Selenium webdriver module and the URL lexical class features using Python. This part is important mainly because, it's essential to collect data set.

Learning Model Selector

It necessitates the data set for learning to build the model, and according to the performance, the learning algorithm with the best result is selected.

Detector

The detector is made using the final classification model generated from the Learning and Model Selector component.

A dataset comprising 609 instances was used initially, out of which 426 instances were used for training data and the rest 182 instances were used as test data. A 13-feature set has been used in this tool for malicious code detection. Three classifiers were used to classify the data and the tool selects the best classifier model based on the accuracy scores to predict the data. The reason for the usage of three classifiers was to maximise the tool performance as each classifier performs differently on different instances. k-NN, SVM and Naïve Bayes algorithms were used and it was estimated that the k-NN algorithm outperforms the other two in terms of performance.

A unique dataset consisting of 420,000 instances has been used specifically for the LR classifier. The accuracy of the LR classifier was found to be 98%. A malicious JavaScript feature dataset from [14] with 70 potential JavaScript features has been used for the other three classifiers (k-NN, SVM and Naive Bayes).

A proxy server is embedded in the tool used to intercept real time traffic and to analyse the traffic results for malicious code detection. A blocking feature has been implemented which blocks the webpage if it is considered malicious. A traffic results button has also been added to the GUI to generate the traffic results captured by the interceptor.

The proxy server creates a HAR JSON blob with the traffic data which is then converted to XML format using the json2xml python module. A web profiler is then used which analyses the xml file and generates the results in a text file.

The traffic_results.txt file which is generated contains a result of the monitored real-time traffic exchanged between the gecko-browser and the server through the interceptor.
The .txt file contains the following details:

- ➔ HTTP Timing details
- ➔ Content size
- ➔ HTTP Requests made
- ➔ Number of redirections
- ➔ File extensions count
- ➔ HTTP status codes

This has been possible by creating a web profiler module to support the proxy server.

Tool Architecture

Keeping in view the need for the real-time detection of malicious JavaScript code, the tool uses a unique approach for the detection of such attacks; the proposed approach consists of an interceptor between a browser and server for detection of malicious JavaScript code. In this approach, all the traffic between server and client is exchanged through the interceptor to check for possible attacks in the source code to be executed by the browser. There are no direct communication channels between browser and server. Browsermob proxy server (A java proxy tool) was used for intercepting and analysing data from the traffic exchanged between the browser and the server.

In this approach, the response from the server is analysed to find the malicious JavaScript code. Features are extracted from the source code of the webpage which is fed into the machine learning classification for decision. If any malicious JavaScript code is found, the page is blocked in the browser.

BrowserMob Proxy [20] is a simple utility coded in Java that makes it easy to capture performance data from browsers, typically written using automation toolkits such as Selenium. BrowserMob Proxy can capture performance data for web apps (via the HAR format) typically from the client side. Once the performance data has been captured, the final step is to perform classification, which will accurately predict to which class a new script belongs, based upon the observations on the training set whose class is already known. Once a malicious script is detected the page will be blocked.

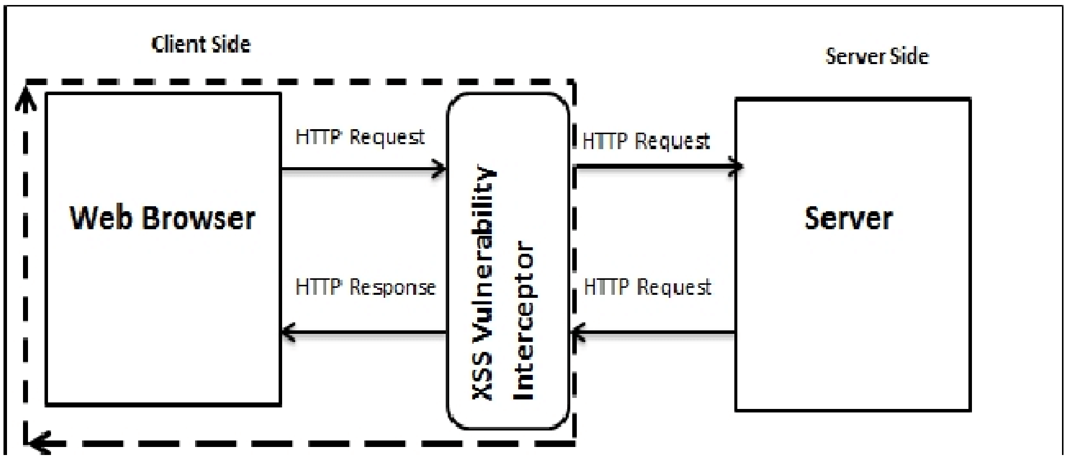


Fig 4.0: The interceptor based approach for detection of malicious JavaScripts [21]

Tool Working

The process of detection of malicious JavaScript code is done in the following steps:

1. The URL is entered in the link text box and the webpage is requested in the headless Firefox Browser(geckodriver).
2. The Browsermob proxy server is started to intercept real time traffic and to extract dynamic features from it.
3. The source code of the webpage is scanned by static analysis in real time for potential features and the values are sent to the classifier.
4. The source code of the webpage needs to be a breakdown by performing lexical analysis to generate a sequence of the tokens to be parsed. Beautiful soup lxml parser has been used for parsing the script tags from the webpage. Regular expressions have been used to match the particular feature against the script contents (JS code).
5. The dynamic features such as the webpage size, number of redirections, http GET and POST requests made along with the status codes are sent by Browsermob proxy to the classifier.
6. The classifier gathers all the input and predicts the class of the entered URL (benign or malicious)
7. If the webpage is found to be malicious, it is blocked before it is interpreted by the browser.
8. Thus accurate classification of webpages as either benign or malicious can be performed by this tool.

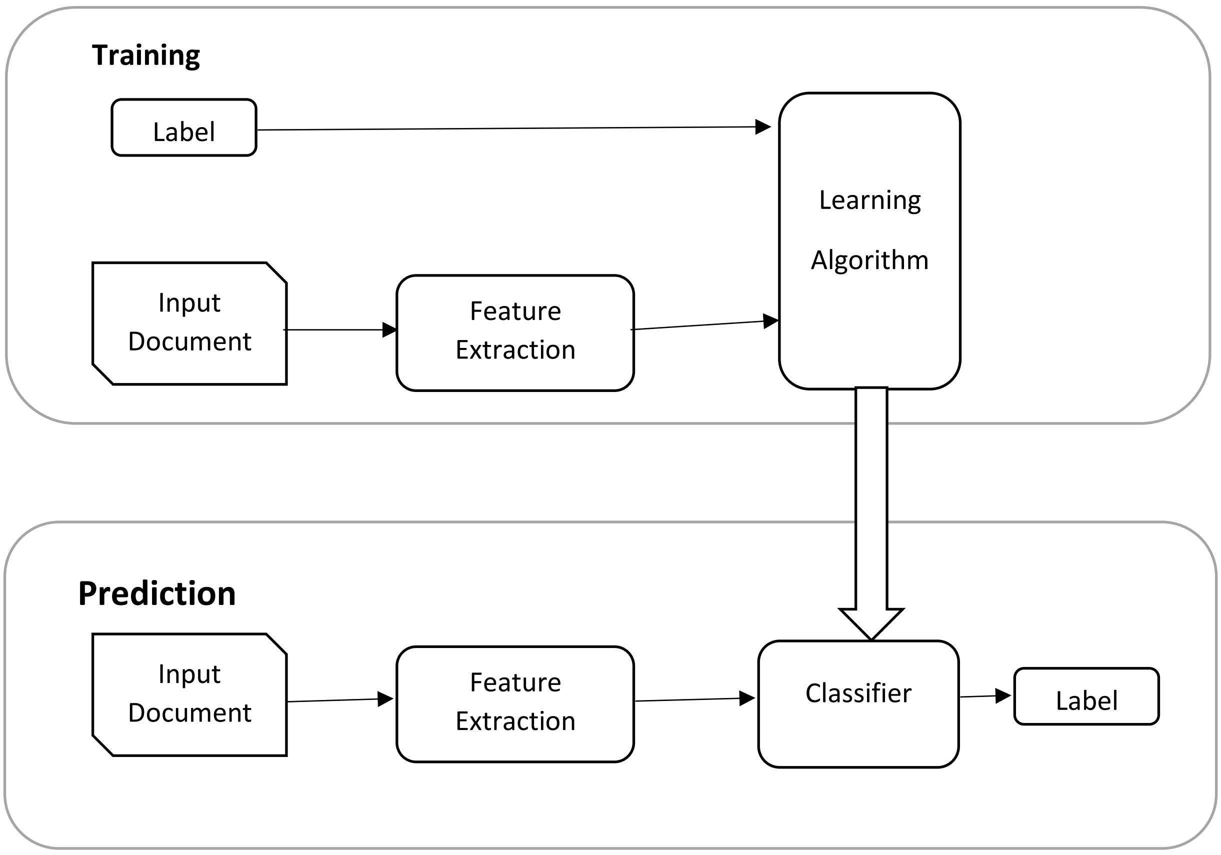


Fig 5.0: Supervised Machine Learning Classification Process

The accuracy of each of the classifiers plays a crucial role to the performance of the tool.

Building the classification model

Dataset

In order to perform classification with high accuracy, the quality of dataset plays a vital role. The dataset is composed of both benign and malicious instances. The collected features are divided into two groups training and test. The total dataset consists of 1919 instances comprising 1510 benign instances and 409 malicious instances obtained from [22].

A separate dataset comprising of 400,000 instances has been used only for the logistic regression classifier in order to maximize its accuracy for result classification.

Features extraction

A number of features can be extracted from JavaScripts that support the degree of maliciousness of a webpage, but not all of them will be helpful in accurate detection of malicious JavaScripts.

A total of 35 features have been used in this tool from a dataset of 70 features. The complete feature-list is shown below:

Complete Feature List:

URL lexical features:	*JavaScript based features:*
1. URL length 2. Number of dots 3. Number of words *Page Content based features:* 4. Link rels 5. Meta 6. Embed 7. Iframe 8. Image 9. Hidden 10. File 11. SetTimeout 12. SetInterval 13. Script 14. Attach 15. Object 16. Autonomous System Number (ASN) 17. PageRank (Host) 18. PageRank (Country)	19. Number of redirections 20. No. of instantiated objects 21. Total bytes allocated 22. Set Attribute 23. Get Attribute 24. Set Request Header 25. Get Response Header 26. Objectcreate 27. Type 28. OpenURL 29. Evaluate 30. Write 31. Import 32. Close 33. Script size 34. Add Event 35. Intent

Table (i)

Classification

This step aims to classify web pages samples as XSS or non XSS. In other words, it is pointed out if a web page is infected with a XSS vector (considered as malicious) using JavaScript code or if it is a benign page.

Four supervised machine learning classifiers were used in this tool for getting the best results:

1. K-Nearest Neighbours Algorithm(k=3)
2. Support Vector Machines
3. Naïve-Bayes Algorithm
4. Logistic Regression

KNN (K- Nearest Neighbor)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K-Nearest Neighbor Algorithm is a simple machine learning algorithm [23] that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbours measured by a distance function. These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing KNN modelling.

Euclidean distance function:

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan distance function:

$$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski distance function:

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Naïve Bayes

Naïve Bayes [24] is a classification technique based on Bayes theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naïve Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).

$$\underset{\text{Posterior Probability}}{P(c\mid x)} = \frac{\overset{\text{Likelihood}}{P(x\mid c)}\,\overset{\text{Class Prior Probability}}{P(c)}}{\underset{\text{Predictor Prior Probability}}{P(x)}}$$

$$P(c\mid X) = P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Look at the above equation.

Here,

- *P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

Support Vector Machines

Support Vector Machines (SVM) were developed by Boser et al. in 1992 [25]. However, the original optimal hyperplane algorithm was introduced by Vapnik and Lerner in 1963 [26]. A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for binary classification of high-dimensional data. The operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. SVM can also be extended to the data that is not linearly separable by applying kernel to map the data into a higher-dimensional space and to separate the data on the mapped dimension.

Logistic Regression

Logistic Regression is part of a larger class of algorithms known as Generalized Linear Model (glm). In 1972, Nelder and Wedderburn [27] proposed this model with an effort to provide a means of using linear regression to the problems which were not directly suited for application of linear regression. In fact, they proposed a class of different models (linear regression, ANOVA, Poisson Regression etc) which included logistic regression as a special case.

The fundamental equation of generalized linear model is:

$$g(E(y)) = \alpha + \beta x1 + \gamma x2$$

Here, g() is the link function, E(y) is the expectation of target variable and α + βx1 + γx2 is the linear predictor (α,β,γ to be predicted). The role of link function is to 'link' the expectation of y to linear predictor.

Based on the results generated, the classifier with the best accuracy score is picked among the four classifiers to generate the result thus accounting to correctness in result prediction.

Performance Evaluation

Performance evaluation acts as a multipurpose tool which is used to measure the actual values within the system against the critical values.

The purpose of evaluation is to study and measure the performance of the classifier in detecting the malicious JavaScript code. In order to obtain the best results, accuracy measurement plays a critical role:

$$\text{Accuracy} = \frac{No.of\ classified\ benign\ scripts}{Total\ No.of\ benign\ samples} \text{ x } 100$$

A false positive scenario occurs when the attack detection approach mistakenly treats a normal code as a malicious code. In a given approach, a false negative occurs when a malicious code is not detected despite its illegal behaviour. Detection rate can be measured by using confusion matrix for the assessment of false positives, and false negatives. False positive and False negative detection rate is calculated by

$$\text{FPR} = \frac{FP}{N} = \frac{FP}{FP+TN}$$

And the false negative rate is calculated by

$$\text{FNR} = \frac{FN}{R} = \frac{FN}{FN+TP}$$

where FPR is false positive rate, FNR is false negative rate, FN is false negative, TN is true negative, and TP is true positive.

True negative shows a number of negative samples correctly identified, false negative implies a number of malicious samples identified as negative, false positive indicates the number of negative samples identified as malicious, and true positive shows a number of malicious samples correctly identified. [28] The performance of the proposed detection will be the rate at which the malicious scripts will be processed. The performance will be calculated by latency time which is taken in presence and absence of an interceptor to display a page and another by calculating the system resource consumption in both scenarios [6].

Experimental Results and Analysis

In the experiment, a dataset comprising 1919 instances with 1510 benign instances and 409 malicious instances was used. The aim of performance evaluation is to study and analyse the performance of classifiers in correctly classifying the instance by using the evaluation metrics such as accuracy, true positive rate and false positive rate. In the tool, the dataset is split in a 4:1 ratio i.e. 80% training data and 20% test data. The train_test_split module was implemented in Python to train, test and split data. Four supervised machine learning classifiers Naïve Bayes, SVM, Logistic Regression and K-Nearest Neighbour algorithms were used.

Experimental-setup:

- Python 3.x (No support for Python 2)
- BeautifulSoup
- Selenium Webdriver module
- Sci-kit learn module with numpy (numpy+mkl) and pandas
- BrowserMob Proxy tool
- BrowserMob Proxy Python Driver
- Runs on Windows 7,8,8.1,10/ Linux
- Matplotlib
- Json2xml

Experimental Procedure: The experiment is carried out in four steps. Firstly, malicious and benign URL lists are prepared. Secondly, features are extracted and grouped into training and test. Thirdly, the model is generated using the training set. And finally, the generated model is tested using the test data set.

Accuracy: The accuracy is defined as the ratio of correctly identified examples over all examples in the test set [19]. The test set is applied to the four classifiers k-NN (K=3), Naïve Bayes (distribution='Gaussian'), SVM (kernel = 'rbf', random_state = None) and Logistic Regression and the performance is shown in the below table.

Features Significance: In this tool, some relevant attack features have been proposed that remain resilient against possible anticipated future attacks. The URL lexical and page content based features rise up the true positive rate where the JavaScript features have a significant effect on the true positive rate. In version 1.0, only the URL lexical and page content features have been used. For real time-data, the tool failed to predict the result correctly due to the presence of lots of false positives leading to lesser accuracy. Thus, in version 3.0, dynamic JavaScript features have been proposed to detect malicious JavaScript pages in real-time. This lead to a very high accuracy of around 98% (Logistic Regression). The experiment shows that the combination of the feature groups has led to a high true positive rate and less false positive rate. The three tables below show the accuracies achieved with different feature-sets used in each version.

Experimental testing was done on a dataset comprising 609 instances (426-benign and 182-malicious) and the following results were obtained:

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Specificity} = \frac{TP}{TP+FP}$$

I) *Accuracies with the existing features*

Classifiers	Recall (%)	Specificity (%)	Accuracy (%)
k-Nearest Neighbor Algorithm	91.3	74.5	88.6
Gaussian Naïve Bayes	90	69	82.5
Support Vector Machine	87	82.3	86.7
Logistic Regression	92.6	84.4	89.3

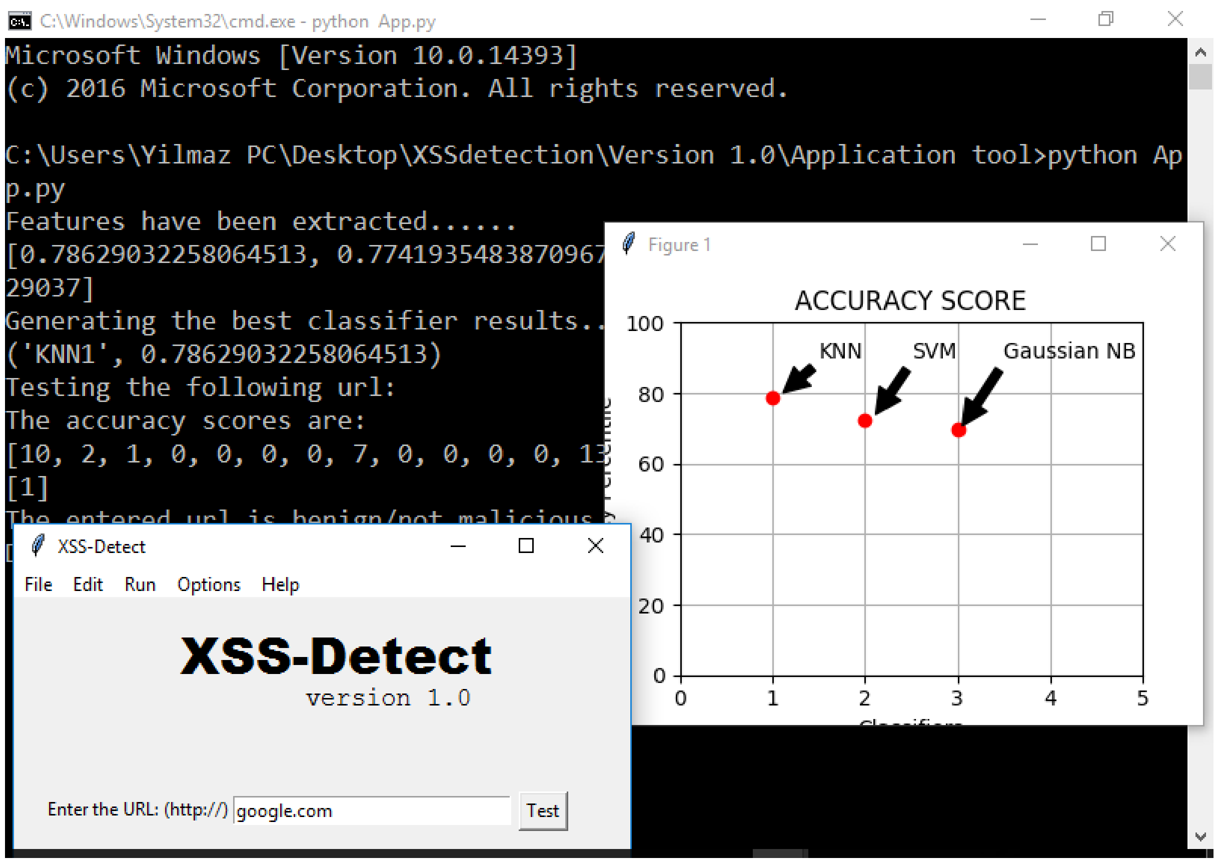
Table (ii): Only URL lexical and page content features have been used here leading to a good accuracy score of around 85% for each of the classifier.

II) *Accuracies with the new features*

Classifiers	Recall (%)	Specificity (%)	Accuracy (%)
k-Nearest Neighbor Algorithm	99.8	80.3	93.75
Gaussian Naïve Bayes	98.8	85.3	94
Support Vector Machine	99.8	89.2	95
Logistic Regression	99.98	87.1	98.4

Table (iii) : After bringing in dynamic JavaScript features, Logistic Regression was found to achieve a high accuracy of 98.3%. SVM, k-NN and Naïve Bayes achieved an average accuracy of around 94%.

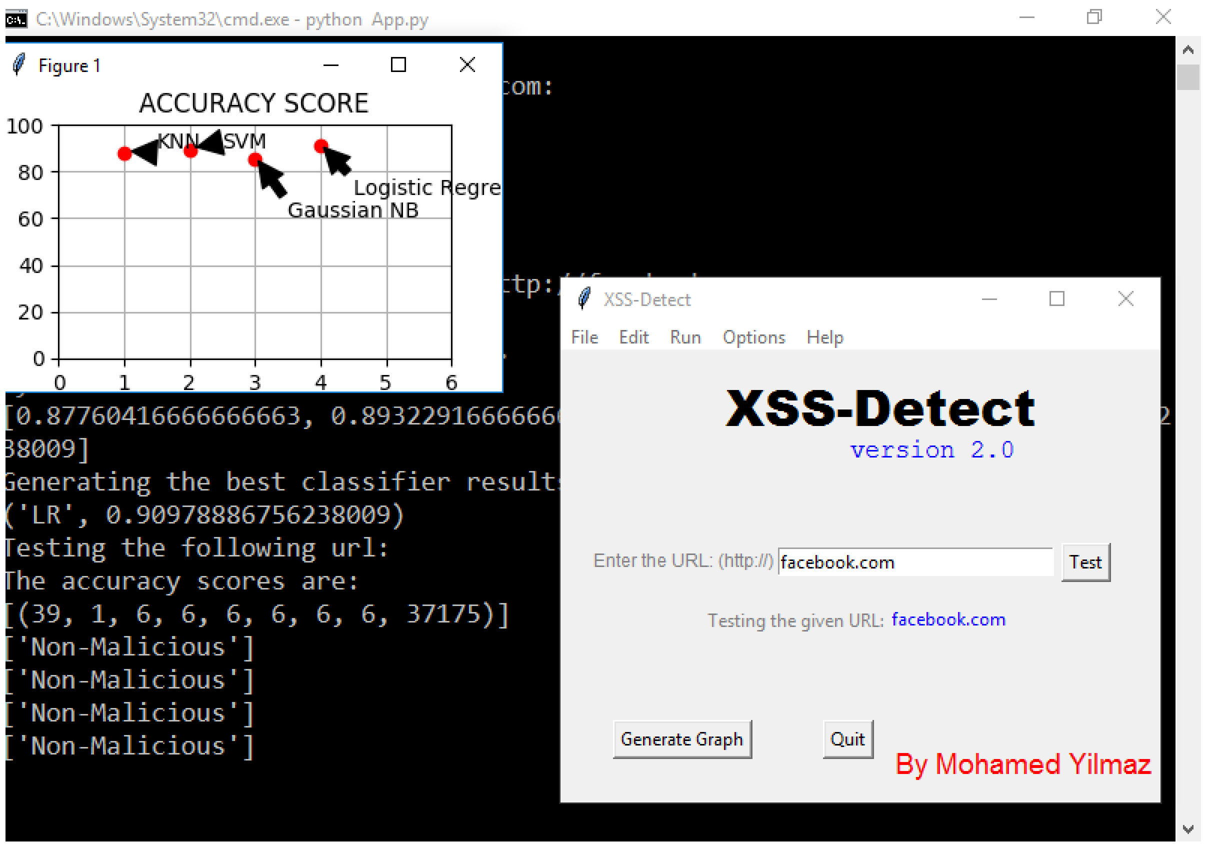
Tool Testing



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Yilmaz PC\Desktop\XSSdetection\Version 1.0\Application tool>python Ap
p.py
Features have been extracted......
[0.78629032258064513, 0.77419354838709677
29037]
Generating the best classifier results..
('KNN1', 0.78629032258064513)
Testing the following url:
The accuracy scores are:
[10, 2, 1, 0, 0, 0, 0, 7, 0, 0, 0, 0, 13
[1]
The entered url is benign/not malicious
```

Fig 6.1: After the tool has performed static analysis on the source code, we can infer that k-NN performs well in this instance with an accuracy of 80.1%.

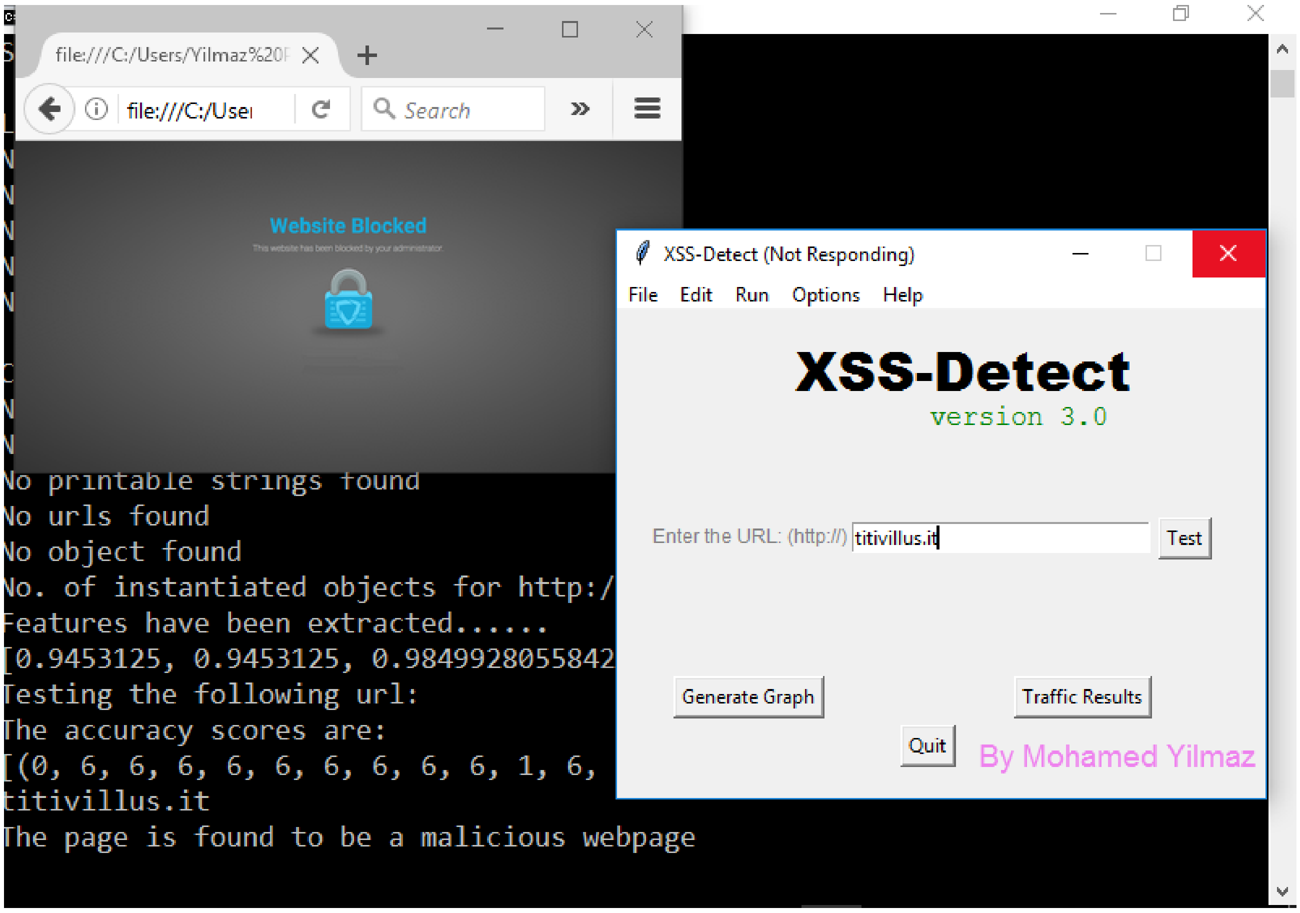


```
com:

tp:

[0.87760416666666663, 0.89322916666666
38009]
Generating the best classifier result
('LR', 0.90978886756238009)
Testing the following url:
The accuracy scores are:
[(39, 1, 6, 6, 6, 6, 6, 6, 37175)]
['Non-Malicious']
['Non-Malicious']
['Non-Malicious']
['Non-Malicious']
```

Fig 6.2: Logistic Regression outperforms the other classifiers with a high accuracy of 89.8%. Gaussian Naïve Bayes performs poorly on this instance having an accuracy of only 36.7%.



```
No printable strings found
No urls found
No object found
No. of instantiated objects for http:/
Features have been extracted......
[0.9453125, 0.9453125, 0.9849928055842
Testing the following url:
The accuracy scores are:
[(0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 1, 6,
titivillus.it
The page is found to be a malicious webpage
```

Fig 6.3: Logistic Regression outperforms the other classifiers here with an accuracy of 98.4%. k-NN, SVM and Gaussian Naïve Bayes also perform very well all having an average accuracy of around 94%. The website is blocked if it considered malicious.

For example, consider the **setAttribute** feature in JS. This can be a potential loophole for attackers to inject malicious code. A malicious JS file can be included in the code in the following way:

setAttribute ('src', 'http://example.com');

Here the attacker can include a malicious URL which would inject malicious JavaScript code in the webpage. The tool checks whether the feature is empty, printable, a URL, an executable path, unprintable or not called in the source code.

All the tests were run on a PC with an Intel Core i5 7300HQ processor and 8GB primary memory.

Conclusions and Future Work:

The idea behind doing this work was to provide a lightweight early assessment model which could quickly determine whether a webpage is infected with a XSS vector or not. This study has proposed an efficient method of detecting previously unknown malicious java scripts using an interceptor at the client side by classifying the key features of the malicious code. A completely automated tool has been developed using Python 3.6 which scans for potential URL lexical, page content and real-time traffic features where the XSS payloads can be injected. Experimental results show that the proposed approach was able to achieve an accuracy of 98.4% with a very low false positive rate (1%), in detection of malicious JavaScript code.

This project presents:

- An analysis of a new set of JavaScript features [14] to classify XSS attack patterns in web pages.
- The implementation of an interceptor based method [6] which employs machine learning techniques to detect XSS attacks in web pages.
- A comparative analysis between Naive Bayes, LR, k-NN and SVM classifiers, in order to show the overall performance of the XSS attacks classification.

Further enhancements can be made on the tool such as the detection mechanism in the proxy can include an ensemble based feature selection approach to distinguish the malicious traffic from the benign traffic. The interceptor can be modified to detect changes in the input fields for possible XSS attacks.

References

1. "Internet Security Threat Report", Symantec, Vol.22, April 2017
2. OWASP Cross-site Scripting Page https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
3. Grossman, J., Hansen R., Petkov, D.P., Rager, A. e Fogie, S. "Cross Site Scripting Attacks: XSS Exploits and Defense". Burlington, MA, EUA, Syngress Publishing Inc. 2007.
4. M. Van Gundy and H. Chen, "Noncespaces: using randomization to enforce information ow tracking and thwart cross-site scripting attacks," in Proceedings of the Network and Distributed System Security Symposium (NDSS '09), pp. 1–18, San Diego, Califo, USA, February 2009.
5. Hydara et al., Current state of research on cross-site scripting (XSS) – A systematic literature review, Inform. Softw. Technol. (2014), http://dx.doi.org/10.1016/j.infsof.2014.07.010
6. Defending Malicious Script Attacks Using Machine Learning Classifiers Nayeem Khan, Johari Abdullah, and Adnan Shahid Khan, Department of Computer Systems & Communication Technologies, Faculty of Computer Science & Information Technology, Universiti Malaysia Sarawak, 94300 Kota Samarahan, Sarawak, Malaysia, February (2017).
7. Ma, J., Saul, L.K., Savage, S., Voelker, G.M.: Learning to detect malicious URLs. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3), 30 (2011)
8. Fukushima, Y., Hori, Y., Sakurai, K.: Proactive Blacklisting for Malicious Web Sites by Reputation Evaluation Based on Domain and IP Address Registration. IEEE (2011)
9. Malware Detection by HTTPS Traffic Analysis Paul Prasse1, Gerrit Gruben1, Jan Kohout2, Lukas Machlika2, Toma´s Pevn ˇ y´ 2, Michal Sofka2,3, and Tobias Scheffer1 (2017)
10. Analysing the ecosystem of malicious URL redirection through longitudinal observation from honeypots, Mitsuaki Akiyama, Takeshi Yagi Takeshi Yada, Tatsuya Mori, Youki Kadobayashi, NTT Secure Platform

Laboratories, Tokyo, Japan, Waseda University, Tokyo, Japan, Nara Institute of Science and Technology, Nara, Japan, January (2017). https://doi.org/10.1016/j.cose.2017.01.003

11. Detection of malicious web pages based on hybrid analysis Rong Wang, Yan Zhu, Jiefan Tan, Binbin Zhou, School of Information Science and Technology, Southwest Jiaotong University, Chengdu, Sichuan, China, May 2017
12. Xu, K., Yao, D., Ma, Q., Crowell, A.: Detecting infection onset with behavior-based policies. IEEE (2011)
13. Jayasinghe, G.K., Culpepper, J.S., Bertok, P.: Efficient and effective real-time prediction of drive-by-download attacks. J. Netw. Comput. Appl. 38, 135-149(2014)
14. Junjie Wang, Yinxing Xue, Yang Liu, and Tian Huat Tan, Jsdc: A hybrid approach for JavaScript malware detection and classification, Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ACM, (2015), pp. 109– 120.
15. I. A. Al-Taharwa, H.-M. Lee, A. B. Jeng, K.-P. Wu, C.-S. Ho, and S.-M. Chen, "JSOD: JavaScript obfuscation detector," Security and Communication Networks, vol. 8, no. 6, pp. 1092–1107, (2015).
16. Yao Wang, Wan-dong Cai, and Peng-cheng Wei, A deep learning approach for detecting malicious JavaScript code, Security and Communication Networks (2016).
17. S. Gupta and B. B. Gupta, "XSS-SAFE: a server-side approach to detect and mitigate cross-site scripting (XSS) attacks in JavaScript code," Arabian Journal for Science and Engineering, vol. 41, no. 3, pp. 897–920, (2016).
18. S. Chun, C. Jing, H. ChangZhen, X. JingFeng, W. Hao, and M. Raphael, "A XSS attack detection method based on skip list," International Journal of Security and Its Applications Vol. 10, No. 5 pp.95-106 (2016)
19. Malicious Webpage Detection: Machine Learning Approach, Abubakr Sirageldin, Baharum B. Baharudin, and Low Tang Jung, Computer & Information Science Department, University Technology Pertonas Bandar Seri Iskandar, 31750 Tronoh, Perak, Malaysia. (2011)
20. Browsermob-Proxy version 2.14, an open source tool. https://bmp.lightbody.net/
21. Towards Vulnerability Prevention Model for Web Browser using Interceptor Approach - Nayeem Khan, Johari Abdullah, and Adnan Shahid Khan, 2015 9th International Conference on IT in Asia, At Kuching, Sarawak, Malaysia, August 2015.
22. leakiEst, School of Computer Science, University of Birmingham, Birmingham, UK.
23. N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," The American Statistician, vol. 46, no. 3, pp. 175–185, (1992).
24. D. Lewis David, "Naive (Bayes) at forty: the independence assumption in information retrieval," in Machine Learning: ECML-98: 10th European Conference on Machine Learning

Chemnitz, Germany, April 21–23, 1998 Proceedings, vol. 1398 of Lecture Notes in Computer Science, pp. 4–15, Springer, Berlin, Germany, (1998).

25. B.E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, pp. 144–152, Pittsburgh, Pa, USA, July 1992.
26. V. Vapnik and A. Lerner, "Pattern recognition using generalized portrait method," Automation and Remote Control, vol. 24, no. 6, pp. 774–780, (1963).
27. Nelder, J.A. and Wedderburn, R.W.M. (1972). Generalised linear models. Journal of the Royal Statistical Society, Series A 135, 370-384.
28. A. E. Nunan, E. Souto, E. M. Dos Santos, and E. Feitosa, "Automatic classification of cross-site scripting in web pages using document-based and URL-based features," in Proceedings of the 17th IEEE Symposium on Computers and Communication (ISCC '12), pp. 000702–000707, Cappadocia, Turkey, July 2012.