Automatic Classification of Cross-Site Scripting in Web Pages Using Document-based and URL-based Features

Angelo Eduardo Nunan, Eduardo Souto, Eulanda M. dos Santos, Eduardo Feitosa
Institute of Computing (ICOMP)
Federal University of Amazonas
Av. Gal. Rodrigo Octávio Jordão Ramos, 3000
CEP 69.077-000 – Manaus-Brazil
{nunan, esouto, emsantos, efeitosa}@icomp.ufam.edu.br

***Abstract*— The structure of dynamic websites comprised of a set of objects such as HTML tags, script functions, hyperlinks and advanced features in browsers lead to numerous resources and interactiveness in services currently provided on the Internet. However, these features have also increased security risks and attacks since they allow malicious codes injection or XSS (Cross-Site Scripting). XSS remains at the top of the lists of the greatest threats to web applications in recent years. This paper presents the experimental results obtained on XSS automatic classification in web pages using Machine Learning techniques. We focus on features extracted from web document content and URL. Our results demonstrate that the proposed features lead to highly accurate classification of malicious page.**

Keywords- cross-site scripting; scripting languages security; web application security; machine learning.

I. INTRODUCTION

Dynamic web applications play an important role in providing resources manipulation and interaction between clients and servers. The features currently supported by browsers as HTML tags, scripts, hyperlinks, and advanced functions have increased business opportunities, providing greater interactivity in Web-based services, such as e-commerce, Internet banking, social networking, blogs, forums, among others.

On the other hand, such features also increased vulnerabilities and threats which allow malicious exploiting. According to Wasserman and Su [1], 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). According to Uto and Melo [2], XSS is an attack that exploits a vulnerable web application, used to carry malicious code usually written in JavaScript, to other user browser. The focus of this vulnerability is the lack of user input data validation [2, 3].

Researches show that XSS remains at the top of the lists of the greatest vulnerabilities in web applications in recent years [4]. In order to deal with the large volume and range of XSS attacks, different approaches and techniques have been proposed in the literature, among which we highlight the use of formal languages and automata [1], primitive markup language elements [5, 6], blacklists, whitelists [7], combinations of techniques [8], etc. However, machine learning techniques have been successfully used to detect web-based anomaly [9, 10, 11].

In this paper, we identify a set of features that allows the accurate automatic classification of XSS in web pages based on supervised machine learning techniques. We apply two machine learning methods, namely Naive Bayes and Support Vector Machines, to classify web pages using a dataset composed of 216.054 websites, where 15.366 of these samples correspond to the attacks occurred from June, 23, 2008 to August, 02, 2011, obtained from XSSed dataset (http://www.xssed.com). We evaluate both classifiers in terms of three criteria, i.e., detection, accuracy and false alarm rates.

The remainder of this paper is organized as follows: Section II introduces the concepts related to XSS and discusses some related works. Section III describes the proposed features for automatic classification of XSS attacks in Web pages. Section IV presents the experimental results and their analysis. Finally, Section V presents conclusions and future work.

II. UNDERSTANDING CROSS-SITE SCRIPTING (XSS)

Grossman [12] 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 [2, 12]. However, technologies such as Active X, Flash or any other technology supported by browsers can also be used as a vector [12].

The XSS attacks can be categorized as Persistent, Reflective and DOM-based [3]. In the first case, the malicious code is permanently stored on server resources. Persistent is the most dangerous type of XSS [3]. In the second case, the code runs in the client browser without being stored on the server. This attack is usually made possible through links to malicious code injection. According to the OWASP (Open Web Application Security Project) [3], this is the most frequent type of XSS attack. Finally, instead of using malicious code embedded into the page that is returned to the client browser, the DOM-based XSS enables dynamic scripts on components of the document, modifying the DOM environment (Document

978-1-4673-2713-8/12/$31.00 ©2012 IEEE

Object Model) [3]. According to Klein [13], the identification of such an attack requires execution or completion simulation of DOM objects. However, many potentially dangerous schemes [14] that enable this type of attack can be detected prior to its execution.

To better illustrate, Fig. 1 presents an overview of XSS attacks. In one of the actions, an attacker can insert a malicious script in resources of a web's server (persistent XSS). Then, it will be permanently displayed on "legitimate" pages returned to other users during regular browsing. In another attack, a hacker can send an e-mail to the target user, which have a link to the malicious script code (reflective XSS) to an URL of a legitimate site. However, this site is hosted on a vulnerable web server. In both attacks, the user receives the requested web page while the browser interprets and executes the page contents and the malicious script code that was injected.

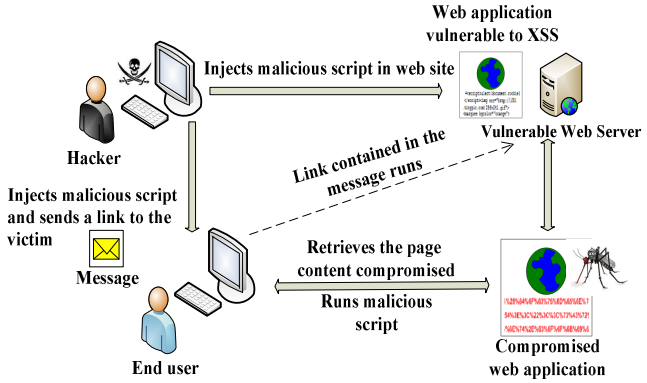

Figure 1. Overview of XSS attacks .

A. *Related Work*

According to Ernst [15] and Vogt et al. [8], recent approaches proposed for identification and classification of XSS attacks are divided into static, dynamic or the combination of both. This categorization takes into account the paradigm used to analyze the code. The static analysis examines the program code and the reasons of all possibilities of behavior that may arise at runtime. The dynamic analysis operates by observing the behavior of the program code execution. This work uses static analysis due to its advantage related to lower execution time when compared to dynamic analysis [15].

1) Static approaches: Nadji et al. [5] proposed a solution based on the same origin policy, which applies the minimum serialization technique though the use of suffixes and prefixes (markers) belonging to a CFG (Context Free Grammar). The CFG is used to delimit the regions of the web page that contain unstrusted data. A similar approach was proposed by Gundy and Chen [6]. In this case, the content of the website is divided into nodes, represented by classes of confidence marked with a XML namespace prefix, randomly generated. Wasserman and Su [1] presented an approach that uses formal language techniques supported by CFG and a FST model (Finite State Transducers) to perform the syntactic structure analysis.

2) Dynamic approaches: In [16], the authors present a policy-based approach, which employs a function that parses the input script, the DOM's node in which the script is inserted and the event that invokes the JavaScripts interpreter. Kirda et al. [7] proposed a proxy-based solution, called Noxes, which acts as a personal firewall for web applications. This method analyzes HTTP requests, allowing connections based on defined security policy through rules, blacklist, and the user's action.

There are works that use a combination of static and dynamic analysis. For instance, Vogt et al. [8] apply a mechanism to identify sensitive data on the client-end in order to ensure that sensitive information is sent by the JavaScript code only to the processed site.

B. *Discussion*

The main difference between this work and the methods previously mentioned is the following. There is no need to adhere to the same-origin policies, which are established by mechanisms that allow scripts to have access only to pages of the same domain, or to change the source code of the current browsers. In addition, we use machine learning techniques that can be applied as a complementary solution to other techniques currently used. For instance, a classifier which can detect XSS attacks successfully can then simplify the development of policies, filters or detection systems. In addition, few work has been dedicated to the use of machine learning in the specific context of XSS attacks.

Recent proposals successfully apply machine learning techniques on web applications in the context of JavaScript. Likarish et al. [9] proposed models for malicious code classification (malware) in JavaScript code present in web pages, based on obfuscation features. Rieck et al. [10] described a web-proxy-based solution called CUJO (*Classification of Unknown JavaScript Code*), that is able to extract static and dynamic features from malicious patterns to detect drive-by-download - a technique used for downloading malicious code (malware) on target machine. It is noteworthy that these studies have presented features focused on drive-by exploits detection.

III. AUTOMATIC CLASSIFICATION OF XSS ATTACKS ON WEB PAGES

As previously mentioned, we use information extracted from web document and URL in order to obtain relevant features of the most frequent XSS attacks. This section presents our selected features and all steps involved in automatic classification of XSS attacks on Web pages.

A. *Features*

Here we describe the features proposed to provide relevant information to accurately separate samples infected with XSS code from the non-infected ones. These features are extracted from Web document and URLs to compose a vector, which is submitted to the classifier. The features are categorized into three groups: (1) obfuscation-based, (2) suspicious patterns and (3) HTML/JavaScript schemes.

1) Obfuscation-based Features: This group is composed by two features and a set of values, described

below, that correspond to alternative encodings, commonly used to obfuscate malicious code. As as consequence, it can bypass data input validation filters and access restrictions, leading to denial of service of an application. In addition, it allows access to the list of directories, spreading malwares or facilitating the exploitation of vulnerabilities and local resources [9, 17]. Likarish et al. [9] sucessfuly used obfuscation-based features in their work.

a) Obfuscated code (Encoded): this feature corresponds to obfuscated strings of characters overshadowed by alternative encodings: Hexadecimal, Decimal, Octal, Unicode, Base64, HTML reference characters. For example, a character string that presents a sequence of more than 40 characters in the range [a-zA-Z0-9], including characters "+, /" and a possible suffix "=" or "==" at the end of the text is assumed to be encoded in base64. Although also found in non malicious JavaScript code, malicious code obfuscation using alternative encodings is widely used in XSS attacks [9,17].

Fig. 2 illustrates an attack via an URL that displays an alert popup with the current cookie for the accessed site by using an alternative encoding (obfuscation) in hexadecimal format [17]. The malicious code inserted after **type** = (lines 1 and 4) is converted by the attacker to hexadecimal format.

b) URL Lenght: is the number of characters of an URL. This feature has been used with good results to classify non-malicious and malicious URLs on phishing detection [18]. Fig. 2 shows an example of XSS attack where the URL presents obfuscated malicious code, which have a standard size, commonly larger than URLs without encoding.

```
1  http://www.trustedsite.com/search.html?type=<''<<sCrIpT>alert
2  (document.cookie)</sCrIpT><''<<sCrIpT>alert(document.cookie)
3  </sCrIpT>
4  http://www.trustedsite.com/search.html?type=%3C%22%3C%3C
5  %73%43%72%49%70%54%3E%61%6C%65%72%74%28%64
6  %6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%
7  65%29%3C%2F%73%43%72%49%70%54%3E%3C%22%3C
8  %3C%73%43%72%49%70%54%3E%61%6C%65%72%74%28
9  %64%6F %63%75%6D%65%6E%74%2E%63%6F%6F%6B%
10 69%65%29%3C%2F%73%43%72 %49%70%54%3E
```

Figure 2. Example of attack using obfuscated malicious code.

2) Suspicios patterns-based feature: This group is composed of three features and a set of values that make up suspicious patterns.

a) Number of Domains (URL_Chain): this feature corresponds to the number of domains found in the URL. Attacks such as Redirect present URLs in chain [12], which are inserted to redirect the victim to pages stored on servers controlled by attackers. The goal is to execute external resources with malicious content as files ".js". Such files may contain malicious JavaScript code which are able to transfer cookies or critical information to the potentially dangerous schemes handling. For example, the URL (www.benignsite.com/redir.php?url=http://www.maliciousite.com./) has two domains ".com" indicating the presence of another URL. In this case, "redir.php?url=" redirects the user to the attacker's site.

b) Duplicated Special Characters (Doubled_Char): it corresponds to the identification of a ill-formed special string of characters, inserted into the opening and closing of tags [19]. These characters are often found in attacks that aim to circumvent anti-XSS filters since many browsers ignore the extra characters and automatically correct the code, allowing its execution. The presence of such feature indicates the existence of potentially malicious XSS. (e.g., <"<<sCrIpt>alert(String.fromCharCode(88,,83,83)</sCrIpt> <"<< alert(String.fromCharCode(88,,83,83)</sCrIpt>).

c) Keywords (Keywords): corresponds to the keywords commonly found on page redirects related to the spread of malware and phishing attacks associated with XSS attacks [12]. (e.g., XSS, banking, redirect, root, password, crypt, shell, spray, evil, etc.).

3) HTML/JavaScript schemes-based feature: This group is composed of a feature and a set of values that are often used in conjuction with potentially dangerous schemes [14].

a) HTML/JavaScript Schemes (Tags_Scheme): It identifies the presence of potentially vulnerable elements to execution of malicious code such as: *i)* HTML tags:<*script*>, <*iframe*>, <*meta*>, and <*div*>; *ii)* HTML properties: href, http-equiv and lowsrc; *iii)* EventHandlers: onclick, onmouseover, and onload; *iv)* DOM objects: Windows, Location, and Document; *v)* Properties such as Cookie, Referrer, and InnerHtml and; JavaScript methods such as *write*(), *getElementsByTagName*(), *alert*(), *eval*() and *fromCharCode*().

Elements which execute dynamic content are important features since they may provide functionalities and greater user interaction. However, many of these schemes are often explored in XSS attacks. In Yue and Wang work [14], the authors present a study of unsafe practices related to the use of JavaScript language in webpages. Jim et al. [16] also employ these schemes as input arguments to a function that analyzes the scripts execution. The W3C Consortium lists on its website (http://www.w3c.org) a series of schemes that invoke the JavaScript interpreter, including *EventHandlers* specified by the combination of HTML elements and attributes that can run scripts without using tag <*script*> <*/script*>. For example, the HTML *img* element can run a script to be specified with the attribute *onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onclick or ondblclick.*

B. Method

To perform the automatic classification of XSS attacks on Web pages, we follow as series of steps illustrated in Fig. 3: detection and extraction of obfuscated feature by alternative encoding, decoding of the webpage, decoded feature extraction and classification of webpages.

1) Detection of Obfuscated Code Step: This step detects the presence of obfuscation on the web page, in encodings of hexadecimal, decimal, octal, Unicode, Base64, HTML

reference characters. Considering the detection of obfuscated characters strings, features based on obfuscation are extracted. In this paper, we consider only the alternative encodings of characters equivalent to the range covered by the ASCII and extended ASCII encoding.

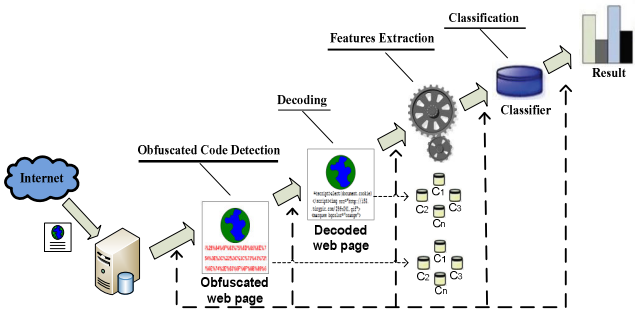

Figure 3. Method used in the automatic classification of XSS.

2) Web Page Decoding Step: After execution of the previous step, the web page is then decoded by routines that perform alternative encodings conversion to ASCII format in order to make the text intelligible and enable the analysis and extraction of other features. Many methods consider only the static code analysis originally found in its form, whether it is obfuscated (encrypted) or not. Thus, the method also aims to allow the extraction of features that would hardly be detected in its obfuscated state.

3) Decoded Characteristics Extraction Step: After the webpage decoding, the extraction of decoded features is performed. At this stage, features are extracted based on HTML/ JavaScript schemes and on suspicious patterns. Features related to code obfuscation can not determine alone whehter or not the code is malicious because the alternative encodings are also used in non malicious code. Thus, this step extracts other features that enhance machine learning and improve accuracy in the classification process.

4) Classification Step: 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 XSS vector using JavaScript code or if it is a benign page. In order to accomplish this objetive, a set of training samples is provided to the classifier which designs a predictive model.

IV. Experiments and Results Evaluation

This section describes the database, classifiers and other parameters used to conduct the experiments, as well as the results obtained. It is important to mention that data mining tool Weka (http://www.cs.waikato.ac.nz/ml/weka) was used.

A. Machine Learning Methods

The experiments were carried out with two machine learning methods: Naive Bayes and Support Vector Machines (SVM).

1) Naive Bayes: It is a statistical method based on Bayes rule [20]. This method makes classification decisions by calculating the probabilities and costs related to each decision. The Bayesian classifiers adopt the concept of feature conditional independence by assuming that a feature value on a given class is independent of the values of other features. In order to classify a sample "x", the Bayes classifier calculates the a *posteriori probability* of each class given "x" [20, 21], and assigns to "x" the class with higthest a *posteriori probability*. This classification method usually achieves high recogniton rate keeping a low computational cost. Moreover, it is widely used in web applications and in anomaly detection [27].

2) SVM: It is a method based on the statistical learning theory and mathematical optimization. This technique works by selecting an optimal hyperplane, that maximizes a class separating margin, in order to divides sampels into different classes. The basic idea of SVM is to nonlinearly map the original feature vector into a space of higher dimension, in which data can be linearly classified. SVM employs a kernel function, which allows calculating the hyperplane without performing the mapping step. This classifier is usually sucessfully applied in clasification problems [20, 21].

B. Database

Naive Bayes and SVM deal with a set of labeled sample data, which include positive and negative samples whether pages are infected with XSS code or not. For the positive class, 15.366 websites were used. These samples were obtained from *XSSed* database (http://www.xssed.com) regarding attacks occurred from June, 23, 2008 to August, 02, 2011. We used two sets of negative samples: (1) 57.207 web pages collected from *Dmoz* database (http://www.dmoz.org); and (2) 158.847 web pages selected from *ClueWeb09* database (http://www.lemur project.org). The negative samples were randomly selected and are pages with its content in English.

C. Performance Measures

The so-called 10-fold cross-validation [20] was used to evaluate the results. This technique aims to predict and to estimate how correct a model will be executed in practice. First, the original whole datasets are divided into 10 folds. Each time, one of the folds is used as test set, and the other 9 folds are put together to form the training set. This process is repeated 10 times. Thus, the performance rates are obtained as the mean across all 10 trials.

"Confusion Matrix" is used as a metric for performance analysis, which enables the result assessment of false positives and false negatives, as shown in Table I. Based on confusion matrix, we calculate the following measures:

- Detection rate = TP/(TP + FN).
- Accuracy rate = (TP + TN)/(TP + TN + FP + FN).
- False alarm rate = FP/(FP + TN).

where: TN (True Negative) indicates the amount of negative samples correctly classified; FN (False Negative) indicates the amount of malicious samples classified as negative, FP (False Positive) indicates the amount of negative samples classified as malicious, and TP (True Positive) indicates the amount of malicious samples correctly classified.

TABLE I. CONFUSION MATRIX

Real Class	Classification	
	Attack (XSS)	*Normal (Non XSS)*
Attack (XSS)	TP	FN
Normal (Non XSS)	FP	TN

D. Performance Analysis in XSS Automatic Classification

The classification methods were evaluated with different configuration values and hyper-parameters. Naïve Bayes was used with its original configuration. SVM achieved the best results with *polynomial kernel,* degree 1.0 (one), and regularization parameter "C" as 1.0 (one).

In order to evaluate the efficiency of the proposed features regarding recognition rates, all of the features were put together to compose the feature vector used to train the classifiers. In this series of experiments, both negative datasets (non-XSS) were used to compose two different databases, even though the same positive dataset (malicious) was used in both databases. The experiment with *ClueWeb09* dataset aimed at validating the effectiveness of the proposed features on a dataset different from *Dmoz*. This choice is due to the fact that there are no XSS public databases focusing on using machine learning techniques.

Table II reports the results attained by SVM and Naïve Bayes. First, it is interesting to note that both classifiers achieved high performances in terms of detection, accuracy and false alarm rates. These results show that the proposed features are discriminant enough to be successful used in automatic XSS classification of web pages. In addition, better performances were obtained on *ClueWeb09* database when compared to the results achieved on *Dmoz* database. This means that our features are not problem-dependent, since *ClueWeb09* has a larger number samples and is totally different from *Dmoz* database. Hence, these features may be widely reused on automatic XSS classification.

TABLE II. COMPARISON AMONG THE RESULTS ACHIEVED BY NAIVE BAYES AND SVM CLASSIFIERS

Classifier	**Naïve Bayes**		**SVM**	
Database	***XSSed***		***XSSed***	
	Dmoz	***ClueWeb09***	***Dmoz***	***ClueWeb09***
Detection rate	95,02%	99,00%	94,07%	98,86%
Accuracy rate	98,54%	99,70%	98,58%	99,89%
False alarm	0,51%	0,22%	0,20%	0,02%
Total of instances	72.573	174.213	72.377	174.213

Fig. 4 shows the performance analysis of classifiers on *Dmoz and ClueWeb* databases. Despite the best overall performance has been achieved by SVM, it can be observed that Naive Bayes attained performance close to SVM. This result indicates that the proposed features may accomplish the conditional independence required by Naive Bayes, i.e., conditioned to a class, the distributions of the features are independent. Moreover, Naive Bayes presents lower computational cost when compared to SVM, which is particularly important in online application.

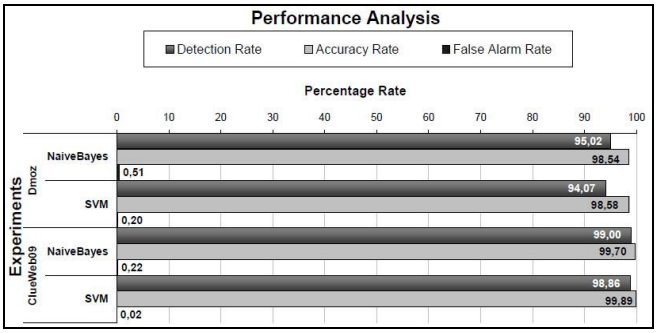

Figure 4. Performance Analysis of Naive Bayes and SVM classifiers.

E. Results Comparison

In order to validate our features, the obtained results were compared to the work of Likarish et al. [9], which applied features based on obfuscation, such as accounting of characters for *Unicode* and Hexadecimal, amount of characters in the script, among others. Another series of experiments was carried out by employing SVM with Likarish et al.'s features over samples contained only on *Dmoz* and *XSSed* datasets. Then, we compare the obtained results with the results achieved with our features. Table III shows these comparison results. The metrics adopted by the authors were: PPP (Positive Predictive Power), equivalent to TP/(TP + FN) and NPP (Negative Predictive Power), corresponding to the equation TN/(TN + FP), which were kept for the purpose of comparison.

TABLE III. COMPARISON OF PERFORMANCES OBTAINED USING SVM

Experiments		
Used features	**Likarish [9]**	**Proposed Features**
Database	Dmoz and XSSed	Dmoz and XSSed
PPP	91,3%	94,0%
NPP	99,1%	99,4%

This experiment shows that our features increased the features used by Likarish et al. [9] in terms of PPP. This gain may be attributed to features based on HTML/JavaScript schemes and suspicious patterns proposed in this paper. However, it is noteworthy that the impossibility of access to training database used by the authors in [9] does not allow us to calculate more detailed measurements on the efficiency of used methods. Considering features based on obfuscation, we evaluated its impact on the overall result in the XSS classification obtained in our experiments using a wrapper-based strategy, as it can be seen in Table IV. When analyzing Table IV, we observe that features based on obfuscation are the most relevant features. However features based on HTML/JavaScript schemes and suspicious patterns added a 1.49% gain of accuracy on XSS classification results in our

experiment using *Dmoz* dataset and a gain of 2.53% when using *Clueweb* dataset. Hence, these results highlight the contribution of these features.

TABLE IV. IMPACT ANALYSIS OF FEATURES BASED ON OBFUSCATION USING THE SVM CLASSIFIER

Analysis of features			
Database		XSSed	
		Dmoz	ClueWeb09
Performance metric		Accuracy Rate	Accuracy Rate
Used features	***Features based on obfuscation***	97,09%	97.36%
	All of them	98, 58%	99,89%
Total of instances		72.573	174.213

F. Conclusions

This paper focus on automatic classification of XSS attacks on Web pages by extracting and analyzing predictive features of the web document content and URL, using Naive Bayes and SVM classifiers.

To this end, experiments were performed using *Dmoz* and *ClueWeb09* datasets as non-XSS web pages, while XSSed web pages were used as malicious samples. Moreover, a brief comparison with the features described by Likarish et al. [9], in terms of performance, is also presented. The classification rates achieved by both classifiers using the proposed features presented stable values and helped the whole set of features to outperform the baseline features.

As contributions, this paper presented:

- The definition and analysis of features to classify patterns of XSS attacks in web pages.
- The presentation of a method which employ machine learning techniques to detect XSS attacks in web pages.
- A comparative analysis between Naive Bayes and SVM classifiers, in order to show the overall performance of the XSS attacks classification.

With regard to future work and considering the magnitude of XSS attacks, it is clear that the search for features that represent new attacks is a good opportunity for new experiments seeking for extending knowledge and conclusions. Moreover, the experiments conducted were limited to two classifiers and manipulation of its parameters. Thus, there are other classifiers that can fit the problem and obtain good results. Finally, even though we tried to deal with large databases, this work does not reach all attacks possibilities.

REFERENCES

[1] Wasserman, G. e Su, Z. “Static Detection of Cross-Site Scripting Vulnerabilities”. In: 30th Internation Conference on Software Engineering, 2008.

[2] Uto, N., Melo, S.P. “Vulnerabilidades em Aplicações Web e Mecanismos de Proteção”. Minicursos SBSeg 2009. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, Campinas, São Paulo, Brazil, 2009.

[3] OWASP, Foundation. “OWASP Testing Guide”, 2008. V3.0. http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf, October, 2011.

[4] OWASP, Foundation. “OWASP Top 10 – 2010. The Ten Most Critical Web Application Security Risks”, release 2010. http://owasptop10.googlecode.com/files/ OWASP%20Top%2010%20-%202010.pdf, Outubro, 2010.

[5] Nadji, Y., Saxena, P., Song, D. “Document Structure Integrity: A Robust Basis for Cross-Site Scripting Defense”. In: 16th Annual Network & Distributed System Security Symposium, NDSS Symposium, 2009.

[6] Gundy, M. e Chen, H. “Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-site Scripting Attacks”. In: 16th Annual Network & Distributed System Security Symposium. NDSS Symposium, 2009.

[7] Kirda, E., Kruegel, C., Vigna, G. e Jovanovic, N. “Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks”. In: 21th ACM Symposium on Applied Computing, ACM, 2006.

[8] Vogt, P., Nentwich, Nenad, J., Kirda, E., Kruegel, C., Vigna, G. “Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis”. In: In Network and Distributed System Security Symposium, NDSS, 2007.

[9] Likarish, P., Jung, E. e Jo, I. “Obfuscated Malicious Javascript Detection using Classification Techniques”. In: 4th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2009.

[10] Rieck, K., Krueger, T. e Dewald, A. “Cujo: Effcient Detection and Prevention of Drive-by-Download Attacks.” In: 26th Annual Computer Security Applications Conference 2010, ACSAC, 2010, ACM.

[11] Chandola, V., Banerjee, A. e Kumar, V. “Anomaly Detection: Survey. A Modified Version of this Technical Report Will Appear”. In: ACM Computing Survey, ACM, September 2009.

[12] 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.

[13] Klein, A. “DOM Based Cross Site Scripting or XSS of the Third Kind: A look at On Overlooked Flavor of XSS”. http://www.webappsec.org/projects/articles/071 105.html, November, 2010.

[14] Yue, C. e Wang, H. “Charatering Insecure JavaScript Practice on the Web”. 18th International Conference on the World Wide Web, Madri. Spain, 2005.

[15] Ernst, M. “Static and dynamic analysis: synergy and duality”. In Proceedings of WODA’2003 (ICSE Work-shop on Dynamic Analysis), Portland, pp. 25–28, May 2003.

[16] Jim, T., Swamy, N. e Hicks, M. “Defeating Script Injection Attacks With Browser-Enforced Embedded Policies (BEEP)”. In:16th International World Wide Web Conference, ACM, 2007.

[17] Common Attack Pattern Enumeration and Classification. “CAPEC-72: URL Encoding”. http://capec.mitre.org/data/definitions/72.html, January, 2011.

[18] Ma, Justin., Saul, L., Savage, S. e Voelker, G. “Identifying Suspicious URLs: An Application of Large-Scale Online Learning“. In: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canadá, 2009.

[19] Common Attack Pattern Enumeration and Classification. “CAPEC-245: Cross-Site Scripting Using Doubled Characters.” http://capec.mitre.org/data/definitions/245.html, October, 2010.

[20] Alpaydin, E.. “Introduction to Machine Learning”. Cambridge, Massachusetts, London, England, 2004. The MIT Press.

[21] Witten, I. e Frank, E. “Data Mining: Practical Machine Learning Tools and Techniques, 2ed. Elsevier.