2014 IEEE International Conference on High Performance Computing and Communications (HPCC), 2014 IEEE 6th International Symposium on Cyberspace Safety and Security (CSS) and 2014 IEEE 11th International Conference on Embedded Software and Systems (ICESS)

Machine Learning based Cross-site Scripting Detection in Online Social Network

Rui Wang, Xiaoqi Jia, Qinlei Li
State Key Laboratory of Information Security,
Institute of Information Engineering,
Chinese Academy of Sciences, China
Email: {wangrui,jiaxiaoqi,liqinlei}@iie.ac.cn

Shengzhi Zhang
Harris Institute for Assured Information,
Florida Institute of Technology,
Melbourne, FL, USA
Email: (zhangs@cs.fit.edu)

***Abstract*—Nowadays online social network (OSN) is one of the most popular internet services in the world. It allows us to communicate with others and share knowledge. However, from the security's point of view, OSN is becoming the favorite target for the attackers, and is under a lot of threats such as cross-site scripting (XSS) attacks.**

In this paper, we present a novel approach using machine learning to do XSS detection in OSN. Firstly, we leverage a new method to capture identified features from webpages and then establish classification models which can be used in XSS detection. Secondly, we propose a novel method to simulate XSS worm spreading and build our webpage database. Finally, we set up experiments to verify the classification models using our test database. Our experiment results demonstrate that our approach is an effective countermeasure to detect the XSS attack.

***Keywords*-Online Social Network, Cross-site Scripting, Machine learning, XSS Detection**

I. Introduction

In recent years, Online social network (OSN) has become one of the most popular internet services in the world . Unlike general networks, OSNs are operating based on users' activity/behavior a lot [1]. Users can sign up or register, establish their own relationship with their friends, publish their profiles, and share any content including links. However, such interaction nature of OSNs leads to more exposure of applications to the clients, especially to some attackers [2].

The cyber attacks caused by web application vulnerabilities have been the major threats to the security of all web services for years [3]. Cross-site scripting (XSS) vulnerability, for instance, is one of the most severe web application vulnerabilities [3], [4]. As a fast growing web application, OSNs have become the main targets of XSS attacks due to their flexibility.

XSS has become a major threat to the security of OSNs [5]. Here we take the security status of OSN sites in China as an example. WooYun [6], which is a vulnerability report platform in China, has set up a database which contains a large number of vulnerbilities. From the data provided by WooYun, we find that a large percent of the vulnerabilities discovered in OSNs are XSS vulnerabilities. Therefore, XSS problem is one of the key research issues in the OSN security.

In this paper, we propose an approach based on machine learning to detect XSS in OSN, and our contributions are as follows:

- We leverage a new method to capture identified features from webpages and then establish classification models which can be used in XSS detection.
- We build two classifiers on the basis of the characteristics of these features, and they are proved to be effective for the proposed features.
- We propose a novel method to simulate XSS worm spreading and build our webpage database.
- Our experiment results demonstrate that our approach is an effective countermeasure to detect the XSS attack.

The rest of the paper is organized as follows. Section II and III describe the features and classifier used in our XSS detecting approach respectively. Section IV shows the implementation of our approach. Section V contains our experiments and evaluation. Section VI describes the related works, and section VII gives the conclusions and the future work.

II. Features Extraction

Considering both the similarities and differences between OSN and general networks, we identify two types of features which need to be extracted from webpages, one for similarities, named *similarity-based features*, and the other for differences, named *difference-based features*.

A. Similarity-based Features

As a web application, online social network has a lot in common with general networks. By taking webpages in OSN's traffic as ordinary pages, we made a static analysis on them. Then, we extracted four groups of features from webpages: keyword features, JavaScript features, HTML tag features, and URL features.

1) Keyword Features: This feature group contains five features: DOM-modified keywords, String-decoding keywords, String-execution keywords, AJAX keywords, and Other keywords.

Keywords are important for detecting XSS malicious JavaScript code. Some researches present that the frequencies of the use of some particular functions between benign and

978-1-4799-6123-8/14 $31.00 © 2014 IEEE
DOI 10.1109/HPCC.2014.137

malicious pages are quite different [7], e.g. the function *eval* is used much more often in malicious pages than it is in benign ones. We take those functions as keywords in our approach.

Likarish et al. [8] propose 50 keywords in their research. In order to reduce the vector dimension and increase the value of items, we combine keywords with similar function into groups. For example, *unescape*, *fromCharCode* and *replace* can be all used for decoding strings, but not likely to appear in the same malicious page. When a webpage has 3 *unescape*, 2 *fromCharCode* and 0 *replace*, we use <5> instead of <3, 2, 0> as the feature vector.

Therefore, we take three steps to improve the efficiency of the keyword features. Firstly, we divide the keywords into certain groups based on their functions. Secondly, we calculate the frequency of each keyword group. Finally, we take these frequencies as keyword features to enhance their efficiency. After the grouping, we obtain fewer keyword features with non-zero values and reduce the feature vector dimension.

2) JavaScript Features: This feature group contains four features: the max length of strings, the number of long string, the max percentage of encoded characters, and the number of concatenation of strings.

XSS JavaScript code is in a different format from the benign one. There have already been many detectors filtering malicious JavaScript. A lot of malicious code programmers use obfuscation techniques such as encoding to bypass these filters. As a result of the obfuscation, the malicious code becomes unreadable and more difficult to be identified. The JavaScript features are highly related to obfuscation techniques. The main characteristics of obfuscated JavaScript code we have found are summarized below: (1) the longer strings. Kim et al. present that when the length of a single string is longer than 200, the webpage is more likely to be malicious [7], (2) more encoded letters, (3) the poor readability of strings. Therefore, we use the features that are extracted based on these characteristics to detect obfuscated malicious code.

3) HTML Tag Features: This group of features aims to check HTML tags and it contains three features: the max length of HTML tags, the number of long HTML tags, and the max number of JavaScript strings in HTML tags.

HTML tags, which are parts of webpages' structures, are extremely vulnerable to malicious code, as they can be inserted and deleted dynamically by JavaScript. Besides, the values of some attributes in HTML tags can be changed or run in browsers automatically. As a result, the malicious code can be injected into the webpages when these attribute values are tampered. For example, the XSS worm "Samy", is hidden in the tag of infected pages [7].

In the analysis, we find that suspicious tags which probably include malicious JavaScript code are large in size. Therefore, we extract the max length of HTML tags, the max number of JavaScript strings, and the number of long HTML tags as the basis HTML tag features.

4) URL Features: This group contains three features: the max length of URLs, the number of long URLs, and the max percentage of encoded characters in URLs.

Capturing malicious URLs is the key point in non-persistent XSS detection. In non-persistent XSS, the malicious URLs will issue a redirect to a compromised web application, and the parameters in them will inject malicious JavaScript code into the requested webpages. Once a user clicks this URL, the attack and propagation can be done simultaneously.

Because of the purpose of malicious URLs, they look different from benign ones, e.g. JavaScript keywords, long URL and poor string readability caused by URL encoding. We can identify the suspicious URLs based on these differences and then detect non-persistent XSS worms in webpages.

B. Difference-based Features

Compared with general networks, online social network has its own features:1) OSN's topology is of high concentration, 2) the average shortest distance between nodes is much lower, and 3) the nodes in OSN's topology trust its neighbors more than the ones in general networks do. Attackters can take advantage of these features to launch XSS attacks. That is, if XSS worms succeed in infecting a single node, they can spread much faster than they do in general networks [9].

We identify the spreading speed of suspicious data in OSN's network traffic as our features. They are the spreading speed of suspicious JavaScript strings, the spreading speed of suspicious HTML tags, and the spreading speed of suspicious URLs.

In our approach, the spreading speed means the number of the users infected by XSS worms in unit time. We take the frequency of the suspicious data in OSN traffic, which is likely to be obfuscated with long string, as the measurement of the spreading speed. We assume that when a XSS worm is alive in OSN, the malicious code related to the worm will appear in network traffic more frequently. Therefore, we calculate the frequencies of suspicious JavaScript strings, HTML tags and URLs appearing in network traffic in the last hour, and take them as the features of the spreading speed.

III. Classification

In this paper, we choose ADTree and AdaBoost as the classifiers.

An alternating decision tree (ADTree) is a machine learning method for classification [10], whose prediction accuracy is higher than other decision trees. It generalizes decision trees and has connections to boosting. ADTree consists of decision nodes, which specify a predicate condition, and prediction nodes, which contain a single leaf. ADTree has fewer strong assumptions, and it even accepts the samples with partial feature values.

AdaBoost, short for Adaptive Boosting, generates a strong classifier using the linear combination of weak classifiers, whose precision rates are greater than 0.5 [11]. The algorithm works by repeatedly running a weak learning algorithm on the training data. In each round of the calculation, it regulates the weight of each sample and each classifier in order to reclassify the false judged samples and to increase the weight of more effective classifiers. Finally, it produces a strong classifier with

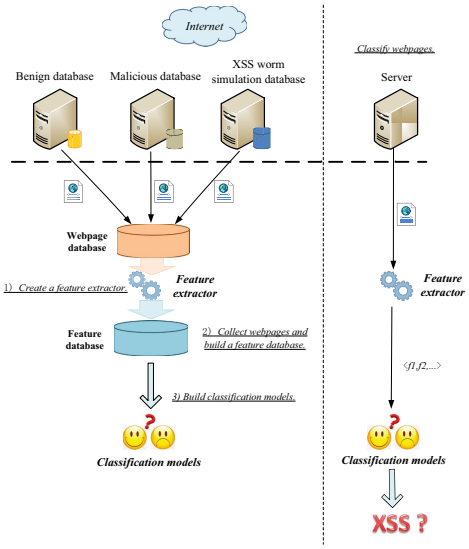

Fig. 1. The architecture of our implementation

higher precision rate than any weak ones. In this work, we choose AdaBoost.M1, a version for binary classification, as the classifier.

IV. Implementation

Fig. 1 shows the detailed process of building the classification models (the left part) and classifying sample webpages (the right part). The process is as follows:

A. Build feature extractor and classifiers

1) **Create a feature extractor.** We create a feature extractor that can capture features automatically. The feature extractor is used to capture features from webpages, which are crucial for the results of classification. The input of the feature extractor is a set of webpages, while the output of it is the corresponding feature vectors that are introduced in Section II.

2) **Collect webpages and build a feature database.** The database is important to the success of a classification method. We collect three types of webpages from Internet: benign webpages, malicious webpages with XSS code, and webpages which can be used to simulate XSS worms in OSN (We will discuss it in Section V. A). Then we build our feature database using the feature extractor which uses the collected webpages as the input. The feature database plays a key role in generating classification models.

3) **Build classification model.** Classification model, which is trained on our collected training data, is a tool to determine whether a webpage is malicious or not based on its features. In this stage, the classification model will be built with feature samples and the chosen machine learning algorithms which are introduced in Section III.

TABLE I
Performance of ADTree and AdaBoost.M1

Classifier	Precision	Recall	F-Measure
ADTree	0.938	0.936	0.936
AdaBoost.M1	0.941	0.939	0.939

B. Classify webpages

With the feature extractor and classifiers, we can detect XSS worms in webpages. When a webpage is to be detected, two steps need to be done: 1) get its features (mentioned in Section II) from the feature extractor, 2) use the extracted features as input to classifiers and obtain the result. The result indicates whether the webpage is infected with XSS worms or not.

V. Experiments and Evaluation

Our data set contains both benign and malicious samples. There are 29,046 benign samples selected from DMOZ database (http://www.dmoz.org), 13,935 malicious ones obtained from XSSed database (http://www.xssed.com). And 8,063 samples, which are from weibo.com, are included in the malicious samples to simulate XSS worm cases in OSN.

Weka [12] is a data mining toolkit which integrates multiple machine learning algorithms, and it can process classification, regression, and clustering, etc. We use Weka as our tool to generate and evaluate the classification models.

In the experiment, we evaluated the classifiers using 10-fold cross validation. In 10-fold cross-validation, the original sample is randomly partitioned into 10 equal size subsamples. One subsample is retained as the validation data for testing the model, and the others are used as training data. The cross-validation process is repeated 10 times, with each of the 10 subsamples used exactly once as the validation data.

For the binary classification, *precision* and *recall* are the values used for evaluations. And *F-Measure* is a harmonic mean of *precision* and *recall*.

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F - Measure = \frac{2TP}{2TP + FP + FN}$$

, TP - the number of positive samples classified as positive, FN - the number of positive samples classified as negative, FP - the number of negative samples classified as positive, TN- the number of negative samples classified as negative.

Before evaluating the classification models, we need to set some parameters for the classifiers. ADTree is used with its initial configuration, while the iteration number of AdaBoost.M1 is set to 50 with the consideration of time cost and accuracy based on our empirical experience.

Table I shows the performance of ADTree and AdaBoost.M1 using both the similarity-based and difference-based features. This experiment is on the datasets containing

TABLE II
EVALUATION OF FEATURES WITH ADTREE

Datasets	Features	TP Rate	FN Rate
Regular network (R data)	Similarity-based	0.902	0.098
	Two groups	0.893	0.107
With simulated OSN (S data)	Similarity-based	0.936	0.064
	Two groups	0.952	0.048

TABLE III
EVALUATION OF FEATURES WITH ADABOOST.M1

Datasets	Features	TP Rate	FN Rate
Regular network (R data)	Similarity-based	0.93	0.07
	Two groups	0.925	0.075
With simulated OSN (S data)	Similarity-based	0.952	0.048
	Two groups	0.958	0.042

simulated instances. We can find that the two classifiers have both achieved good performances.

In order to strengthen the verification of the effectiveness of our approach, we take another experiment. In Table II and Table III, the first two rows show the performance on regular network data and the last two rows show the performance on the datasets with simulated OSN instances. This performance shows that the difference-based features perform effectively on the simulated OSN instances. In general, the results of these experiments verifies that our features are effective in XSS detection in OSN environment.

VI. RELATED WORK

Now researchers are paying much more attention on XSS detection in OSN. Livshits. B et al proposed an automatic detection architecture, Spectator [3], which can detect JavaScript worms by searching for worm-like long propagation chains. Another useful architecture, PathCutter [13], works by blocking two main propagation paths of JavaScript worms. Louw M.T. et al [14] implemented a tool called BLUEPRINT to control the parsing decisions of script to avoid the unreliable parsing behavior in untrusted HTML.

Machine learning have been used in the XSS detection in general networks. Warningbird [15] proposed a machine learning method to do XSS detection. Likarish et al [8] proposed features for classification to detect the obfuscated malicious JavaScript, and they used several classifiers to evaluate these features' effectiveness. A similar approach was proposed by Wang et al [16]. They use 27 features and SVM classifier to detect malicious JavaScript.

Comparing with the above methods, we use machine learning based method to detect XSS in the OSN environment. Moreover, we propose a set of features based on the characteristics of OSN.

VII. CONCLUSION

This paper proposes an novel approach using the machine learning method to detect XSS in OSN. Firstly, we propose a new method to capture identified features related to OSN's XSS worm. Secondly, we build classification models using the ADTree and AdaBoost algorithms. Thirdly, we set up an experiment to verify the effectiveness of our approach. Our experiment results demonstrate that our method is effective and efficient in OSN's XSS detection.

VIII. ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foundation of China (NSFC) under Grant No. 61100228, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06030601 and XDA06010701, and the National High-tech R&D Program of China under Grant No. 2012AA013101.

REFERENCES

[1] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC)*, pages 29–42. ACM, 2007.

[2] Mark EJ Newman and Juyong Park. Why social networks are different from other types of networks. *Physical Review E*, 68(3), 2003.

[3] V Benjamin Livshits and Weidong Cui. Spectator: Detection and containment of javascript worms. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, pages 335–348, 2008.

[4] Angelo Eduardo Nunan, Eduardo Souto, Eulanda M dos Santos, and Eduardo Feitosa. Automatic classification of cross-site scripting in web pages using document-based and url-based features. In *IEEE Symposium on Computers and Communications (ISCC), 2012*, pages 702–707. IEEE, 2012.

[5] Guanhua Yan, Guanling Chen, Stephan Eidenbenz, and Nan Li. Malware propagation in online social networks: nature, dynamics, and defense implications. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, pages 196–206. ACM, 2011.

[6] Wooyun. http://www.wooyun.org/index.php.

[7] Byung-Ik Kim, Chae-Tae Im, and Hyun-Chul Jung. Suspicious malicious web site detection with strength analysis of a javascript obfuscation. *International Journal of Advanced Science & Technology*, 26:19–32, 2011.

[8] Peter Likarish, Eunjin Jung, and Insoon Jo. Obfuscated malicious javascript detection using classification techniques. In *Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 47–54. IEEE, 2009.

[9] Wei Xu, Fangfang Zhang, and Sencun Zhu. Toward worm detection in online social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, pages 11–20. ACM, 2010.

[10] Alternating decision tree, wikipedia. http://en.wikipedia.org/wiki/ADTree.

[11] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[12] Weka, the university of waikato. http://www.cs.waikato.ac.nz/ml/weka/.

[13] Yinzhi Cao, Vinod Yegneswaran, P Possas, and Yan Chen. Pathcutter: Severing the self-propagation path of xss javascript worms in social web networks. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2012.

[14] Mike Ter Louw and VN Venkatakrishnan. Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (Oakland), 2009*, pages 331–346. IEEE, 2009.

[15] Sangho Lee and Jong Kim. Warningbird: Detecting suspicious urls in twitter stream. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2012.

[16] Wei-Hong WANG, Yin-Jun LV, Hui-Bing CHEN, and Zhao-Lin FANG. A static malicious javascript detection using svm. In *Proceedings of the International Conference on Computer Science and Electronics Engineering*, volume 40, pages 21–30, 2013.