Cross-Site Scripting Attack Detection Based on Hidden Markov Model

[1]Yeng-Ting Wu, [1]Shiou-Jing Lin, [2]En-Si Liu, [2]Hsing-Kuo Pao, [2]Ching-Hao Mao, [2,3]Hahn-Ming Lee

[1]Information & Communication Security Lab, Chunghwa Telecom Laboratories

[2]Department of Computer Science & Information Engineering, Taiwan Tech

[3]Institute of Information Science, Academia Sinica

{lyndon,sjlin}@cht.com.tw, {m9615046,pao,d9415004,hmlee}@mail.ntust.edu.tw

***Abstract*—Cross-Site Scripting (XSS) is a well-known type of web vulnerabilities which allows attackers to inject the malicious scripts or codes to compromise the web application services. Based on the characteristics of client side script language, the attackers can launch XSS attack by a single HTTP request to a website easily. Therefore, detection of XSS is a critical issue for all website managers. In this paper, based on a machine learning approach, we propose a new method to detect XSS attacks on a web server. We preprocess an HTTP request to a token sequence, and utilize Hidden Markov Model to determine whether an XSS attack exists in the HTTP request. By filtering HTTP requests on the server side, our approach can label each HTTP request whether it is an XSS attack or not. Moreover, apart from other related methods, ours takes the proximity into consideration. The proposed system performs well with high accuracy rate on a real data set collected from a private telecom company.**

***Index Terms*—Cross site script, hidden Markov model, web security, token sequence.**

1. Introduction

Cross-Site Scripting (XSS) has become one of the most prevalent threats in web security recently. According to the report from Web Application Security Consortium (WASC) [6], the number of vulnerabilities which belong to XSS attack has grown to the largest proportion of all web vulnerabilities. On the other hand, to fit the demands of growing web, disappointedly, HTML has no capability to distinguish trusted code from inline data and the data sanded by users. This characteristic and the widely use of client side technologies that support dynamic web contents give attackers chances to inject malicious scripts or codes to compromise the web application services.

XSS vulnerabilities exist if the web application takes the inputs from trustless users and use them to dynamically generate web pages without sufficiently validating, filtering or encoding the supplied data. Attackers can inject JavaScript or other browser executable content into web pages of legitimate sites with XSS vulnerabilities. When other users visit these legitimate sites and browse the web pages with malicious contents injected by attackers, the malicious contents are executed in users' browsers that cause users to become victims. By XSS exploits, attackers are able to have victims' browsers execute malicious scripts or other browser executable content automatically, which may hijack users' sessions, redirect users to malicious sites, conduct phishing attacks, or even install malicious codes without users' awareness.

There are three main kinds of XSS: Reflected, Stored and DOM injection [11]. The cause of Reflected XSS is that the malicious content is reflected directly back to the browser while attackers inject malicious content to a web page of the vulnerable site. Reflected XSS exploits are delivered to victims via various ways. Most ways for delivering Reflected XSS exploits are to construct a crafted URL which contains the malicious content as parameters, and then send emails to victims with some contents the victims might be interested in and a link to the crafted URL in the emails. When victims click the malicious link, Reflected XSS exploits are executed in victims' browsers. The difference between Reflected and Stored XSS is that Reflected XSS reflects the malicious content directly back to the browser without storing it

while Stored XSS stores the malicious content in a database or other back-end system. In a Stored XSS attack, the attacker injects malicious content to a vulnerable legitimate site first and the malicious content is stored in the vulnerable site. When a user visits the vulnerable site later, the malicious content is retrieved from the database or back-end system, returned, unfiltered, in the HTML page and displayed to the user. Attackers may target sites such as social networking sites, discussion forums, or blogs where a large number of users will see materials provided by other people. DOM-Based exploits are implemented to manipulate the site's JavaScript code and variables instead of the HTML elements.

In this paper, we propose a new method that detects XSS attacks on the web server based on a machine learning approach – Hidden Markov Model (HMM). We preprocess an HTTP request to a token sequence and then use HMM to detect XSS attacks or anomaly behavior in the sequence. Our approach can be implemented in the web server or Web Application Firewall (WAF) for the detection of XSS anomaly attacks.

The novelty of our approach is to treat the detection of XSS attacks as a sequence labeling problem. We present a machine learning method based on probabilistic modeling that considers the correlation of consecutive tokens in the token sequences. By giving labels to the tokens, we can spot the starting and ending points of an attack.

We evaluate this approach against a range of XSS attacks and demonstrate it effectiveness on real world traffic. We gathered access logs of web servers in a private telecom company as the data set. The result demonstrates that our approach achieves high detection rates in our experiments.

This paper is organized as follows: Section 2 contains a review of related work. Section 3 describes the system architecture of our approach. We describe our experiments with more details and results in Section 4 and conclude this work in Section 5.

2. Related Work

Many studies have been proposed for the detection of web attacks. Ingham et al. [1] proposed an approach that utilizes DFA (Deterministic Finite Automata) induction to detect malicious web requests. They tokenize HTTP requests and build a DFA model for the requests; then determine whether an HTTP request is an attack via a similarity measure which is calculated for an HTTP request and the DFA. However, their research does not take the token correlation of sequences into consideration. Wang et al. [7] proposed a content anomaly detector that models a mixture of high-order *n*-grams to detect malicious packet content. However, this approach named Anagram suffers from lower accuracy when the data input is highly dynamic in data type and ordering like web traffic due to the use of hashes to raise its efficiency.

Song et al. [8] used mixtures of Markov chain model to deal with code-injection attacks in web layer. Compared to other related methods, their model performs efficiently with $O(n)$ in its time complexity. For high variability of HTTP requests, this model may get more noise than other models that tokenize the HTTP requests in the preprocessing step. Similarly, by tokenizing HTTP requests, Roberson et al. [9] presented a method that uses a generalization technique to translate abnormal HTTP requests into signatures for abnormal behavior. Collecting features in HTTP requests to form a classification problem is also an interesting approach. Kruegel and Vigna [10] developed an approach that detects web attacks by considering some attributes from the HTTP requests such as length, character distribution, ordering of parameters and so on.

There are other XSS detection mechanisms that are not based on the detection of anomaly behavior. We discuss them as follows. Saxena et al. [12] and Gundy et al. [13] have similar methods to defend XSS attacks by ensuring a fundamental document integrity property. Saxena et al. developed a new approach that combines randomization of web application code and runtime tracking of trustless data both on the server and on the browser to combat XSS attacks. Gundy et al. presented Noncespaces to let a web application randomizes the XML namespace prefixes of tags in each document before delivering it to the client, so that client can distinguish between trusted and trustless

data.

In addition, another two mechanisms aim to prevent XSS attacks from stealing the victim's confidential information. To block URL requests, an approach called Noxes [14] provides a client-side web proxy, which can use manual and automatic rules to detect malicious content. Vogt et al. [15] tracked the flow in the browser in order to prevent malicious content from leaking sensitive information to others. For high dynamic data such as web-layer content, tokenization is an approach to eliminate noises on high dynamic data. Tokenization will be elaborated in next section.

3. The Detection of Cross-Site Scripting Attacks

There are many token-based methods (e.g. DFA) applied in web attack detections. However, most of them do not model token correlation well between a pair of consecutive tokens. In this study, we design a token sequence extractor to transform HTTP requests into token sequences. Furthermore, we intend to model the temporal relations in token sequences to identify XSS attacks by Hidden Markov Model (HMM). Figure 1 shows the system architecture which includes three components, namely: Token Sequence Extractor, HMM-based Token Correlator and XSS Attack Detector. Token Sequence Extractor transforms HTTP requests into token sequences. The HMM-based Token Correlator uses HMM to extract token correlation of neighboring tokens in labeled token sequences. XSS Attack Detector is responsible for determining whether an input token sequence contains any XSS attacks and where the attacks are in.

Meanwhile, the system uses a data source and a HMM Parameter Profile. The data source contains labeled HTTP requests and HMM Parameter Profile supports to build HMM-based Token Correlator model. Labeled HTTP requests are the HTTP requests labeled by domain experts in advance. HMM Parameter Profile contains all parameters of HMM model and the output of the model to determine labels of predicted token sequences via XSS Attack Detector.

3.1 Feature Extraction for Tokenization

In order to model token correlations of token sequences, we utilize Token Sequence Extractor to transform an HTTP request into a token sequence. Ingham et al. [1] tokenize HTTP requests according to HTTP RFC standard [2]. However, it has the difficulty because some web browsers and related programs do not fully follow the RFC standard. Variation which makes anomaly detection system fail to profile attacks under different environments is another problem in tokenizing HTTP requests.

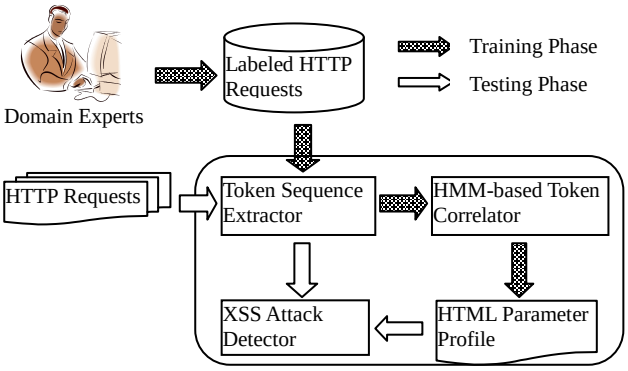


Figure 1: System Architecture.

In order to solve the problems mentioned in previous paragraph, we remove tokens from token sequences to reduce high variability if the tokens give no help to the system. For example, Accept-Language and Accept-Charset contained in HTML header are not important fields in our system to detect XSS attacks, so the tokens will be eliminated in our Token Sequence Extractor.

Table 1: Special Symbols used to identify a token boundary

~	!	@	#	$	%	^
&	*	(	)	_	-	+
=	`	[	]	{	}	\
|	;	:	‘	“	,	<
.	>	/	?			

A token represents a semantic unit and therefore we can discuss the relationship between different units in a token sequence. In the process of transforming HTTP requests to token sequences, we use special symbols to identify token boundaries. For instance, a GET URL string such as “/ics/index.php?id=Bob&pwd=3423” will be separated into fourteen tokens. Note that these

fourteen tokens should also go through the subsequent processes for removing meaningless or non-helpful tokens. Table 1 shows the special delimiter symbols used by Token Sequence Extractor.

To produce token sequences, we propose a transformation algorithm illustrated in Algorithm 1. The algorithm accepts HTTP requests as the input and produces token sequences as the output. For an HTTP request, we split it into an array, and sequentially extract the elements in the transformation process. For each element, we find a token corresponding to it and produce a token sequence by combining those tokens. The set of token types is denoted by *TS*, which includes various token types. For example, we name the token whose original contents are related to script language as "script_word".

Algorithm 1: Generate Token Sequence
Input: An HTTP request R
Output: A token sequence X
1. $Y = (y_1, y_2, \cdots, y_n)$, and n is the length of Y;
2. $TS = (t_1, t_2, \cdots, t_m)$, and m is the length of TS. TS represents the token set.
3. Split R into array Y by special symbols ;
4. P lists special symbols of R orderly, ex. p_1 represents the special symbol located before y_1
5. **for each** $y_i \in Y = (y_1, y_2, \cdots, y_n)$ **do**
6. classify y_i to suitable token $t_j \in TS$;
7. append p_i to X;
8. append t_j to X;
9. **end**
10. remove insignificant tokens and their types from X;

3.2 The Detection of Cross-Site Scripting Attacks

Hidden Markov Model (HMM) includes a finite-state machine that transits its states by probability. In HMM, we do not know the states which are called hidden states, but we can infer state sequence based on the hint from observations. A Markov Chain is useful for estimating the probability for a sequence of events [16]. Since the order of tokens is meaningful in a token sequence, it is suitable for utilizing HMM to correlate the tokens in neighboring locations. Before using HMM, there are three elements that should be specified: 1) the initial probabilities which specify the probabilities of starting each state; 2) the transition probabilities that represents the probabilities from one state to another state; and 3) the observation probabilities, also called emission probabilities, which express the likelihoods of any observation that is generated from any state.

We design two HMM models for HMM-based Token Correlator: one is responsible for profiling normal token sequences, and the other is responsible for profiling XSS token sequences. The model which represents normal behavior is denoted by π_n, and the other model which represents XSS behavior is denoted by π_x. We use θ_n to denote the maximum likelihood value of model π_n, and θ_x to denote the maximum likelihood value of π_x according observed sequences, respectively. Algorithm 2 shows the details of XSS Attack Detector. One input is a token sequence, and the other two inputs are θ_n and θ_x, which are the output of HMM-based Token Correlator. The output of XSS Attack Detector is the labeled token sequence. XSS Attack Detector determines labels of the token sequence such as normal or XSS attack by comparing θ_n and θ_x. Parameter T represents the threshold value, and if θ_n and θ_x are all smaller then T, we label the token sequence as an attack not belonging to XSS attack.

HMM-based Token Correlator uses Expectation-Maximization (EM) algorithm to obtain the appropriate HMM model for normal and abnormal sequences. After feeding training data which is labeled by experts, hopfully we ontain converged HMM parameters such as the initial probabilities, the transition probabilities and the emission probabilities for observations.

4. Experiments

In this section, we demonstrate our experiment result. First, we illustrate the training and test data sets, which include normal sequences and the sequences with XSS attacks. After that, we give details of the computation environment for our experiments. We also explain all the parameters used in

our experiments.

Algorithm 2: XSS Attack Detector
Input: A token sequence, and the other two inputs are θ_n and θ_x which are the output of HMM-based Token Correlator.
Output: Determine what label a token sequence is.
1 *T* represents the threshold
2 **if** $\theta_n > \theta_x$ and $\theta_n >= T$ **then**
3 Label the token sequence as Normal.
4 **else if** $\theta_n <= \theta_x$ and $\theta_x >= T$ **then**
5 Label the token sequence as XSS attack.
6 **else**
7 Label the token sequence as Abnormal

Finally, we analyze the characteristics of our proposed approach. We also show the effectiveness of our method and compare it with other methodologies, such as Logistic Regression and Naïve Bayes.

4.1 Data set and Environment Description

The training and test data sets were obtained from the access logs of web servers in a telecom company called *Chunghwa Telecom Laboratories* [17]. We took the ordinary access logs as normal training and test sets. To generate the access logs of XSS attacks, we used some tools and websites which provide the functionality of producing XSS attack strings. For example, CAL9000 [3] is an OWASP project and provides a collection of web application security testing tools. It gives the flexibility and functionality that we need to generate the XSS attacks. On the other hand, RSnake [4] provides the XSS cheat sheet, which contains lots of XSS attack strings. We wrote some web pages, launched the XSS attacks, and got the access logs of XSS attacks.

Table 2 lists all information about the data sets. The normal access logs were collected from Aug. 20, 2008 to Jul. 20, 2009, and the access logs of XSS attacks were obtained at Aug.28 2009. The total number of entries in normal access logs is 18,469, and training/test set contains 15,000/3469 entries respectively. The total number of entries in XSS access log is 110, and training/test set contains 60/50 entries respectively.

Table 2: Information about the data set

	Normal	XSS attack
Duration	Aug. 20th, 2008 -Jul. 20th, 2009	Aug.28th, 2009
# of access log	18469	110
# of training/test	15000/3469	60/50

4.2 Parameter Setting

We utilized an HMM toolbox [5] based on MATLAB to justify our approach. Table 3 lists all the parameters that we used in our experiments. The training iteration of EM leads the log likelihood of normal and XSS attack models converged to a constant value, as shown in Figure 2. There are 36 kinds of token types in our experiments, so the number of output symbol *O* is 36. We choose Hidden state number *Q* as 7 based on our experience.

4.3 Experiment Results

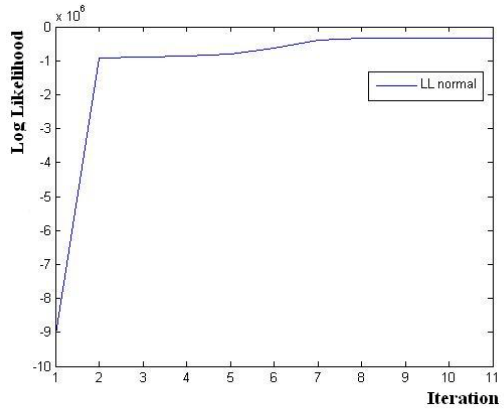
The evaluation results of our proposed method, as well as Logistic Regression, and Naïve Bayes are shown below. Table 4 shows the classification results of test set by three methods. The first row (Normal) and the first column (Normal) of our approach is 3459, which means that there are 3459 data out of normal test set classified as normal behavior. Similarly, the first row (Normal) and the second column (XSS attack) corresponds to 10 and means that 10 entries out of normal test set is classified as XSS attacks. The rests are listed in the same manner.

Figure 3 shows the false positive rate, precision and recall, comparing of three methods. First, the False Positive Rate means the proportion of normal behaviors that are erroneously reported as XSS attacks. Our approach can lead to 0.3% in the false positive rate and the other two methods lead to the false positive rates above 3%.

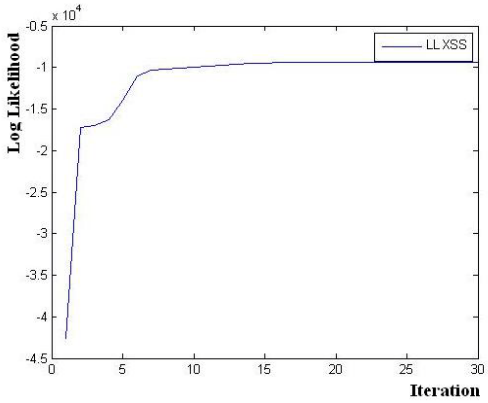
The Precision can be regarded as a measure of exactness or fidelity for the proportion that is classified as XSS attacks. We can see that our method has the highest precision rate in comparison with

two other methods.

The Recall is used to measure the classified XSS attacks the completeness for the whole XSS attacks. Our method has the 100% recall rate, which means that it can identify all the XSS attacks in the test set.



(a)



(b)

Figure 2: The log likelihood of (a) normal model and (b) XSS attack models.

Table 3: All parameters used in our experiments

Parameter (Symbol)	Value
Training iteration of EM (max_iter)	15
Output symbol (O)	36
Hidden state number (Q)	7

Table 4: The confusion matrix of three methods

		Normal	XSS attack
Our Approach	Normal	3459	10
	XSS attack	0	50
Logistic Regression	Normal	3214	255
	XSS attack	12	38
Naïve Bayes	Normal	3354	115
	XSS attack	3	47

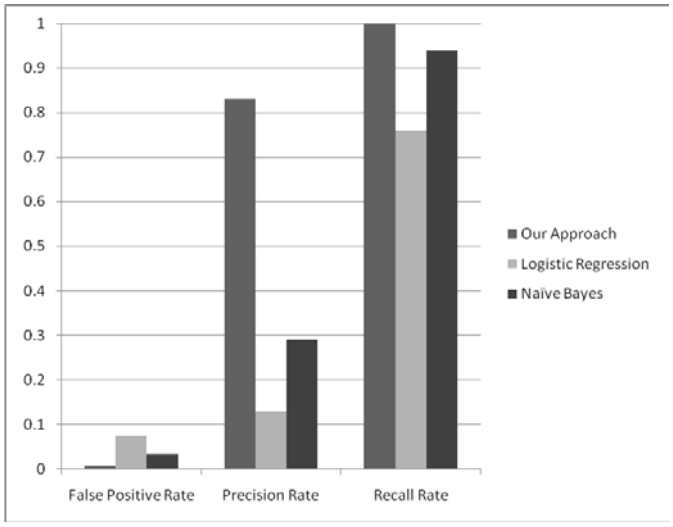


Figure 3: The false positive rate, precision and recall of three methods.

5. Conclusion

In this paper, we proposed a method to detect XSS based on HMM. We transform HTTP requests into token sequences and then apply HMM to correlate tokens in the token sequences. The proposed method is effective in the sense that it successfully identifies XSS web attacks with high accuracy. It is also easy to plug-in the proposed method into web intrusion detection system or WAF for thorough protection on web applications. We have tested our approach on real-world access logs, and the experiment results show that our approach can achieve high accuracy and low false positive rates for various XSS attacks compared to other base line machine learning approaches.

For the future work, we will utilize our approach to detect various kinds of web attacks, like SQL injection, CSRF attack, path traversal, etc.

Moreover, to make our method more robust, we will test our model on large-scale data sets, including not only access logs, but also other sources of IDS or WAF.

Acknowledgements

This work was supported in part by Chunghwa Telecom Laboratories under grants TL-97-7401, also by the National Science Council under grants NSC 97-2221-E-011-105.

REFERENCE

[1] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest, "Learning DFA representations of HTTP for protecting web applications," vol. 51, no. 5. New York, NY, USA: Elsevier North-Holland, Inc., 2007, pp. 1239-1255.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - http/1.1." United States: RFC Editor, 1999.

[3] CAL9000, http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project

[4] RSnake, http://ha.ckers.org/xss.html

[5] HMM Toolbox for matlab, http://people.cs.ubc.ca/~murphyk/Software/HMM/hmm.html

[6] WASC, "Web Application Security Consortium", http://www.webappsec.org/

[7] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *9th Recent Advances in Intrusion Detection (RAID)*, 2006, pp. 226-248

[8] Y. Song, AD. Keromytis and SJ. Stolfo, "Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic," in *the 16th Annual Network and Distributed System (NDSS)*, 2009.

[9] W. Robertson, G. Vigna, C. Kruegel, R.A. Kemmerer, "Using generalization and characterization techniques in the anomaly-based detection of web attacks," in *Proceedings of Network and Distributed System Security Symposium conference*, 2006, Internet Society, 2006

[10] C. Kruegel, G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on computer and communications security*, ACM Press (2003), pp.251-261

[11] OWASP Top 10 2007-Cross Site Scripting, http://www.owasp.org/index.php/Top_10_2007-A1

[12] P. Saxena, D. Song, and Y. Nadji, "Document structure integrity: A robust basis for cross-site scripting defense," in *16th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA, Feb. 2009.

[13] M. V. Gundy and H. Chen, "Noncespaces: using randomization to enforce information flow tracking and thwart crosssite scripting attacks," *16th Annual Network & Distributed System Security Symposium*, 2009.

[14] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. "Noxes: A Client-Side Solution for Mitigating Cross Site Scripting Attacks," In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, Dijon, France, April 2006.

[15] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2007.

[16] D. Jurafsky and J. H. Martin, "An introduction to natural language processing, computational linguistics, and speech recognition," Oct 10, 2006

[17] Chunghwa Telecom Laboratories, http://www.chttl.com.tw/