Cross-Site Scripting (XSS) Detection Integrating Evidences in Multiple Stages

Jingchi Zhang
Information Security Institute
Johns Hopkins University
jzhan161@jhu.edu

Yu-Tsern Jou
Information Security Institute
Johns Hopkins University
yjou2@jhu.edu

Xiangyang Li
Johns Hopkins University
Information Security Institute
xyli@jhu.edu

Abstract

As *Cross-Site Scripting (XSS) remains one of the top web security risks, people keep exploring ways to detect such attacks efficiently. So far, existing solutions only focus on the payload in a web request or a response, a single stage of a web transaction. This work proposes a new approach that integrates evidences from both a web request and its response in order to better characterize XSS attacks and separate them from normal web transactions. We first collect complete payloads of XSS and normal web transactions from two databases and extract features from them using the Word2vec technique. Next, we train two Gaussian mixture models (GMM) with these features, one for XSS transaction and one for normal web transactions. These two models can generate two probability scores for a new web transaction, which indicate how similar this web transaction is to XSS and normal traffics respectively. Finally, we put together these two GMM models in classification by combining these two probabilities to further improve detection accuracy.*

1. Introduction

This research aims at a new approach of detecting Cross-Site Scripting (XSS) web attacks. Using information from multiple stages instead of one, the evolution of XSS can be better captured. Moreover, a dual model is employed that integrates an XSS model and a normal traffic model, presenting two opposite perspectives of one web transaction. In practice, this approach can be supported by a honeypot for evidence generation and collection.

1.1. Cross Site Scripting (XSS) attack

Cross Site Scripting (XSS) attackers take advantages of improper input validation to inject malicious code into web pages. XSS attacks can lead to defacing web pages and leaking sensitive data [1]. For its seriousness and prevalence, OWASP has listed the XSS vulnerability in its Top 10 Most Critical Web Application Security Risks from 2013 to 2017 [2].

There are three types of XSS attacks, i.e., stored, reflected, and DOM-based. In the stored XSS attack, it embeds a piece of malicious code into a vulnerable web page, which is stored on the web server for later use. As a result, the attack will be executed when a victim visits the vulnerable web page. In this case an attacker does not need to craft special URLs since the malicious payload is already on the web server. In a reflected XSS attack, the attacker tricks the victim into clicking on an ill-formed URL, which sends malicious code to a vulnerable web application on a server. If not properly handled, the response from the server is then directed back to the victim's browser that executes the XSS payload to enable the attacker to access the victim's data. In the DOM-based XSS, the attacker also tricks the victim into clicking on a maliciously crafted URL. But the malicious code will not be sent to the vulnerable application. Instead, it will be executed at some point when a web page is loaded onto the victim's machine. Our study focuses on reflective and stored XSS attacks.

1.2. Word2vec technique

Traditionally, a malicious script is inserted within a Hypertext Markup Language (HTML) code piece, which can be analyzed through natural language processing (NLP). It includes tag labels like <script>, <img> or <body> that can be considered as subjects, then follows with verb or noun, something like '*alert()*' or '*onload=*' in HTML. This mapping can be applied to all the elements in the HTML code.

There may be hidden relationships between tags or elements. Treating words in a piece of text as discrete atomic symbols in most of NLP systems seems not appropriate. More precisely, these words are arbitrarily encoded as integer values using a hash table. However, such encodings may fail to truly represent the words, as some important information, i.e., the linkage between

URI: https://hdl.handle.net/10125/60153
ISBN: 978-0-9981331-2-6

tag and elements, is missing in coding. In contrast, representing such rich, high-dimensional relations as vectors, called word embeddings, can overcome the above problem. There is one of the most popular techniques used in NLP systems, called Word2vec. It is a predictive model used for learning the word embeddings from a text corpus, which is a list of words or tokens [3]. It comes with two model structures, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. In our work, we utilize the CBOW model based on the following observations.

1) CBOW model focuses on the existence of a word in a window of surrounding words, but the Skip-Gram also values the order of words. However, CBOW model is good enough in our problem since switching two words likely does not make a big impact.

2) Compared to Skip-Gram model, CBOW model is more time-efficient.

After grouping similar words together in the vector space and assigning each word its corresponding vector value, Word2vec can "understand" the meaning of a piece of text, e.g., a web request or response, by establishing the association of any word pair. And the relationship of each word pair is measured by its cosine similarity in vector space.

1.3. Gaussian mixture model (GMM)

For statistical learning, [15] suggests that using GMM to fit the dataset and identify any outliers may be an alternative option if the Gaussian distribution is not applicable for the data. And as [16] states that GMM can well define all the possible data points by assigning the probability rather than a cluster in the k-means algorithm. Therefore, GMM can also be viewed as an extension to the k-means method, which is found to be the most popular statistical learning algorithm that finds probabilistic cluster assignments.

Last but not least, a mixture model is a probabilistic model that is used to solve the subpopulation assignment. For example, given the height data of a group of people (population) that includes an unknown number of female or male in the group (subpopulation), the mixture model assumes that such data act as the sum of two shifted and scaled normal distributions. It can learn the subpopulations and predict which subpopulation an unknown identity, i.e., an observation, belongs to using a probability score. If the prediction is achieved without labeling the datasets, similar to clustering, a mixture model is refereed as unsupervised learning model. One such a model is a Gaussian mixture model (GMM), which can have more than two components or subpopulations.

As illustrated in [4], XSS attacks, as one population, are multimodal with more than one variant of attack payloads in terms of words being using in them. In one subpopulation, the word distribution of attacks may follow a normal distribution. If using only one distribution in modeling, the overall population may be poorly represented. It is important to model a multimodal distribution using a GMM model of multiple components.

In this work, we develop an approach that uses GMMs to analyze XSS and normal web transactions by examining their payload as evidences. This study aims to improve detection performance by two enhancements: (1) correlating evidences from multiple request and response stages of an attack instance, common in any client/server transactions, and (2) integrating two GMMs modeling XSS and normal respectively.

2. Related work

Cross-Site Scripting (XSS) web attacks have been studied from different perspectives, including browser filter [11], intrusion detection system [12], [13] or firewall. Generally, they can be detected through signature-based, anomaly detection, or a combination of these two [14, 16]. Anomaly detection can handle zero-day attacks while signature-based methods may not; however, anomaly detection may perform poorly with a non-comprehensive normal profile.

2.1. Multi-stage attack detection

Several studies have employed machine learning algorithms to detect computer attacks while considering different attack stages. Katipally et al [5] analyzed attacker's behavior by utilizing a hidden Markov model (HMM). Their analysis considered a continuous sequence of different activities as one attack. In particular, they conceptualized five stages in such an attack model, where each stage represents a different operation. These stages include scanning through network mapping, enumeration through DNS requests, exploitation by access attempt through buffer overflow and SQL injection, exploitation by denial of service through flooding the system, and exploitation by malware through shell code. They generated input to train the HMM model based on previously learned alerts and intrusions that is effective in predicting the attacker's behavior. However, a typical problem associated with most machine learning methods is a tendency of generating false alarms.

Lee et al [6] also used HMM in intrusion detection using audit data. Their definition of a multi-stage attack consists of multiple attack activities, where each stage represents one type of attack. Installing an IDS agent based on Snort, they collected intrusion traces from each attack stage and fed the information to the HMM model. Their system showed efficiency in detection, but with a relatively high false positive rate.

Sampath et al [7] took one step further in using the HMM and bag of words model by including the context in analysis. The contextual information they included varies from the source and destination IP addresses to the alert type and category. The intrusion alerts were first categorized according to K-means clustering. Then they fed collected sequences of alerts labeled with the cluster information to an HMM model that can predict the next probable alerts. The prediction can provide information about future attack strategies. Similarly, they considered one complete sequence of a multi-stage intrusion, which also consists of multiple attack activities. Their proposed system was sensitive to the number of clusters chosen in K-means clustering. It can produce an accuracy of 88% using 5 clusters while the accuracy drops to 31% using 50 clusters.

Almutairi et al [8] proposed a method to detect multi-stage attacks based on examining the reputation of network IP addresses using fuzzy logic. They captured network traffic in three multi-stage attack scenarios. They evaluated the reputation of the IP address using fuzzy rules. The four fuzzy rules were formulated based on blocked IPs, anonymous proxy IPs, malicious geographical IPs, and low rated IPs. Their approach was efficient with a zero false positive rate for IP addresses classified as malicious. However, relying on whitelisting, it failed to detect new malicious IPs.

2.2. XSS attack detection

XSS attack detection based on machine learning has been well studied by researchers in these years. Fang et al presented a novel signature-based detection system using Word2vec and Long Short-Term Memory (LSTM) models [17]. After extracting the features by Word2vec, a LSTM model can be trained to segregate the XSS from normal data. And the precision rate is 99.5% in real dataset.

Liu addressed web attack detection and employed several machine learning algorithms [9]. The author utilized Gaussian HMM and a lexical segmentation technique based on the bag of words technique. Both training and testing the HMM model used payloads in web requests, i.e., a URL link, to classify the traffic as normal or XSS. It claimed to achieve an accuracy of over 90% in detection.

Wang et al [10] utilized an HMM model combined with the Bayes theorem. The authors stated that this method can learn the structure of attack vectors and minimize flaws of traditional sanitization procedures. The authors tried to evaluate the performance of such learning model using mutated XSS attacks in the XSSed database (http://xssed.com). Such XSS attacks may not include realistic attack variants. Moreover, design of this HMM model as well as how data collection was done lacked specific details.

In contrast to many server-side XSS detection systems, Pelizzi et al [11] presented a new client-side XSS filter called XSSFilt. It was claimed to outperform NoScript, a Firefox plug-in, XSS Auditor on Google Chrome, and other server-side solutions in detecting reflected XSS attacks. The key functions of this filter include use of approximation rather than exact string match, utilization of their own improved *syntactic confinement policy* that are not SQL specific, and analysis on web responses. However, its approximate substring matching can lead to overfitting that increases the false positive rate. It may fail to identify unseen and varied XSS instances.

In summary, most of studies that employ a machine learning approach in XSS detection, being either signature-based or anomaly-based, by use the evidence in only either the web request or the response. Consequently, these detection systems may fail to identify malicious script if such evidences alone are not discriminating enough. We propose an innovative approach utilizing dual GMM models to better characterize the difference between XSS and normal web transactions to fill this gap.

3. Our approach

We try to correlate evidences in both request and response payloads of an XSS attack. With richer information, it is believed that number of false positives and negatives can be potentially reduced. Practically, we utilize separate GMM models to characterize XSS and normal web transactions respectively and integrate predictions from these models in order to provide better classification of a new web transaction.

Table 1. Comparison of our approach to others

		Evidence		
		Single Stage	**Multiple Stages**	
Detection Model	**XSS**	Others	*Our approach*	*Dual model*
	Normal	*Our approach*	*Our approach*	

Table 1 shows how our work (in shade) is compared to other existing efforts of XSS modeling and detection. The payloads of a complete request/response chain are assembled together to represent an entire web transaction. After collecting such information, an XSS model and a normal model are trained respectively. Before a web transaction is fed to the GMM model, payloads are divided into words and are vectorized by the Word2vec technique aforementioned. An GMM model is trained using either XSS or normal payloads to characterize distributions of these two types of transactions. The XSS model is trained by an XSS dataset and the normal model by a normal dataset.

There are two models in detection, one is trained by XSS transactions and another by normal transactions. The former acts as a *misuse* detection system and the later acts as an *anomaly* detection system. The trained models calculate the probability scores of an unknown web transaction indicating how likely the instance is to XSS and normal subpopulations respectively. The output score from each model can be used to classify an unknown web transaction with a control threshold.

4. Design and implementation

The general framework and procedure of data collection and processing are shown in Figure 1.

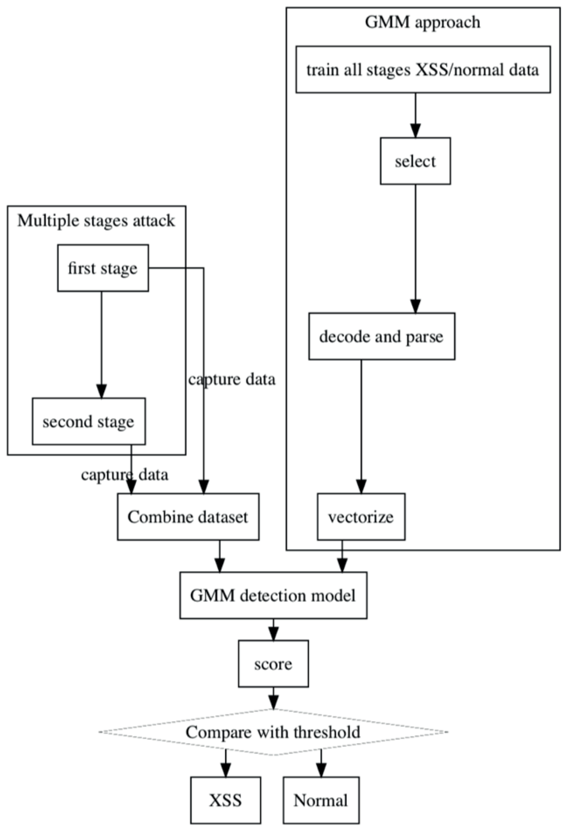


Figure 1. System design

4.1. General design

A reflective XSS attack is an application layer web attack that exploits vulnerabilities of a web server by specially formed web request scripts. A user is also known as the victim; an attacker uses the victim to exploit a web server vulnerability; and a web server has vulnerabilities that an XSS attack can exploit. An attacker is a computer node that generates a URL that contains a malicious payload and sends it to the victim to inject it into the web server. The victim is the client browser or user interacting with the web server, thereby, the one loading and executing a webpage with the malicious script injected.

The process described above may vary based on the type of XSS attack. This research does offline analysis of the web traffic captured on the client browser side. Data collection collects web traffic packets at every stage of a web transaction. We use a web archive called XSSed (http://xssed.com) to generate and collect XSS transaction instances and an open source project called Web09 (http://boston.lti.cs.cmu.edu/Data/web08-bst/planning.html) for normal transaction instances.

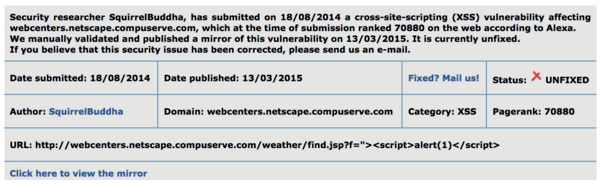
The offline analysis uses XSS GMM and normal GMM models trained on the collected XSS and normal instances. We use a python package (http://scikit-learn.org/stable/modules/mixture.html) to build GMM models and choose different numbers of components in investigation.

4.2. Data collection

Training and testing need labeled XSS and normal data. For the XSS dataset, 45,884 web transactions are collected. These samples are obtained from the *XSSed* database containing attacks occurring from February 7, 2003 to March 15, 2013. The normal samples have 34,561 web transactions from the *Web09* database, which is a collection of web pages crawled in January and February 2009. These samples are randomly selected to be used for training and testing.

4.2.1. Normal traffic collection. The original dataset is a 25-terabyte dataset of about 1 billion crawled webpages, which follows the order of the OPIC metric. The complete dataset is not available, so our research focuses on the 34,561 web transactions included in a sample dataset provided by the database. This dataset is distributed in the Web Archive (WARC) format, which is used to store "web crawls" as sequences of contents. More precisely, it contains warcinfo, response, resource, request, metadata, revisit, conversion, and continuation information. Parsing this sample dataset generates 34,561 web transactions for the normal data.

4.2.2. XSS traffic collection. The XSSed project has one of the largest online archives of vulnerable websites. As shown in Figure 2, XSSed lists every XSS vulnerability sample's patching status, domain name, web application attack category, attack payload, and a link to the mirror destination of the website in question,


Security researcher SquirrelBuddha, has submitted on 18/08/2014 a cross-site-scripting (XSS) vulnerability affecting webcenters.netscape.compuserve.com, which at the time of submission ranked 70880 on the web according to Alexa.
We manually validated and published a mirror of this vulnerability on 13/03/2015. It is currently unfixed.
If you believe that this security issue has been corrected, please send us an e-mail.

Date submitted: 18/08/2014	Date published: 13/03/2015	Fixed? Mail us!	Status: ✗ UNFIXED
Author: SquirrelBuddha	Domain: webcenters.netscape.compuserve.com	Category: XSS	Pagerank: 70880

URL: http://webcenters.netscape.compuserve.com/weather/find.jsp?f="><script>alert(1)</script>

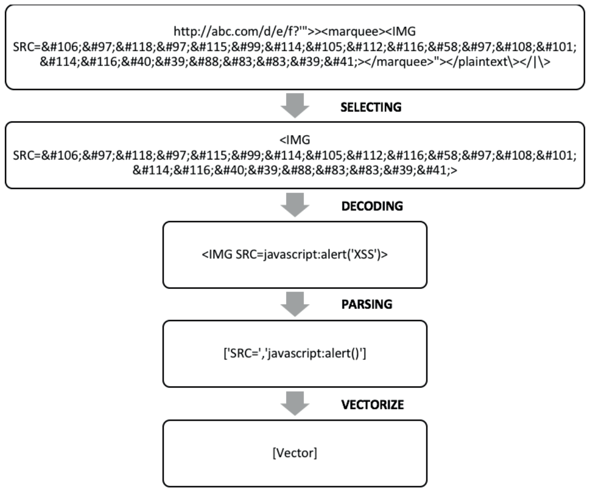
Click here to view the mirror

Figure 2. Information for one vulnerable website in the XSSed archive

Using Wireshark, a network sniffer, we capture both request and response after executing the mirror link and saved them to a PCAP file.

4.3. Data processing

As in Figure 3, each web transaction instance is processed into tokens then transformed into a corresponding vector through the Word2vec technique. It is obvious that raw requests and responses contain plenty of non-relevant information, including randomly generated digits and value, HTML tag name, and other special characters. Putting all such information into a GMM model may dilute the training result. Thus, we first filter out these contents in three steps: Selection, Decoding and Parsing.


```
http://abc.com/d/e/f?'">><marquee><IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;
&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;></marquee>"></plaintext\></|\>
```

```
<IMG
SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;
&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>
```

```
<IMG SRC=javascript:alert('XSS')>
```

```
['SRC=','javascript:alert()']
```

Figure 3. Processing one instance into an input vector for the GMM model

4.3.1. Selection. The features are extracted from all the listed strategies and scenarios in the XSS Filter Evasion Cheat Sheet by OWASP [4]. Incorporating the RFC 2616 Hypertext Transfer Protocol - HTTP/1.1, we summarize comprehensive HTML tag and attribute pairs that are frequently used in XSS attack payloads. In summary, those attack signatures come in three flavors, embedded as an external file, hidden in "harmless" parameter values, and hidden in data fields. Table 2 shows some XSS payload examples.

As we can see, some attribute values are as unexpected as defined by the web transaction protocol. For example, when the browser is requesting a link: "*<LINK REL="stylesheet" HREF="javascript:alert('XSS');">*". Instead hidden JavaScript codes may be triggered from their places. Hiding malicious operations in this way can confuse traditional XSS filters. Thus, we select these potential malicious parameter values and pass them to the decoder for further analysis.

Table 2. Examples of malicious XSS payload

Attribute	Value	Value Expected	Description
HREF	javascript:alert()	URL	Specifies the address of the external file to embed
STYLE	background-image: url(javascript:alert('XSS')	CSS style	Specifies an inline CSS style for an element
SRC	onmouseover=" alert('xxs')"	URL	Specifies the address of the external file to embed

4.3.2. Decoding. Encoding is a commonly used mechanism to obfuscate malicious code. As browsers automatically decode the HTML traffic and execute them, simple text filters may fail to recognize XSS payloads in a different encoding scheme. More precisely, the obfuscated strings can be overshadowed by multiple ways of encodings: Hexadecimal, Decimal, Octal, Unicode, Base64, and HTML reference characters. For example, generating a sequence of the special character "*%*" followed by two hexadecimal digits is used to represent the alphanumerical characters in the range of [a-z, A-Z, 0-9] and some special characters, which is called URL reference encoding. As in the second example in Table 3, '*%77*' represent ASCII Value '*w*' followed by '*%2E*' as '.' Although such encoding is sometime found in non-malicious JavaScript code, obfuscation using alternative encodings is widely used in XSS attacks.

Table 3. Character references

Encoding Type	HTML-Encoded	<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>
	Original	<IMG SRC=javascript:alert('XSS')>
	URL-Encoded	<A HREF="http://%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D">XSS</A>
	Original	<A HREF="http://www.google.com">XSS</A>
	ASCII-Encoded	<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
	Original	<IMG SRC=javascript:alert(String.fromCharCode(XSS))>

In addition to using different encoding schemes as in Table 3, obfuscation can also be done by inserting redundant characters to opening and closing of tags. Since the browser ignores the extra characters and automatically corrects the code, the presence of these characters may indicate the potential existence of XSS.

4.3.3. Parsing. A parser tokenizes a string based on the following rules:

1) Replace digits with blank space;
2) Replace any value that only contains characters in ranges of [a-z] or [A-Z] with blank space;
3) Replace special characters other than those in {'.', '_', '-', ':', '@', '/', '='} with blank space;
4) Move the content in the parentheses as in {(), [], <>} to the back of them;
5) Remove blank spaces and empty tokens;
6) Substitute network location by 'domain' and directory by 'path' for the normal GMM model;
7) Remove network locations for the XSS GMM model.

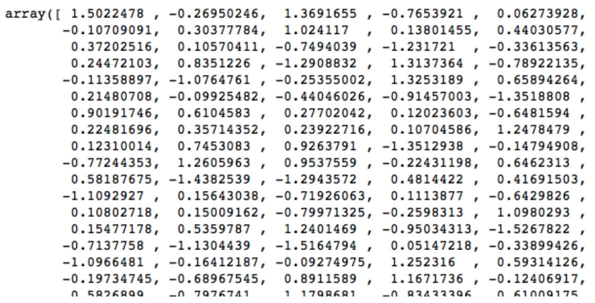
As shown in Table 4, these rules remove non-critical value, such as in '*Page=2*', '*alert('XSS')*' and '*String.fromCharCode('XSS')*' and keep special characters used in critical JavaScript methods or interpreter, such as 'q*uery=*', '*alert()*', '*document.cookie*', and '*onclick()*'. Then they move out the parameter from the JavaScript function for further analysis as it may also contain special structure, e.g., '*javascript:alert(String.fromCharCode())*' is changed to '*javascript:alert()*' and '*String.fromCharCode()*'. The resulting tokens are used to represent an instance of requests and responses for further processing.

Table 4. Example of tokenizing a web request

Request	http://www.m86security.com/support/searh/SiteSearch.asp?query=Search...&Product="'--></style></script><script>alert('XSS')</script>&Page=2?<IMG SRC=javascript:alert(String.fromCharCode('XSS'))>
Token	['domain', 'path', 'query=', 'Search...', 'Product=', 'alert()', 'Page=', 'javascript:alert()','String.fromCharCode()']

Finally, we substitute the network location by '*domain*' and '*path*' only for normal GMM model. The XSS GMM model focuses on tokens unique to XSS attacks. The network location tokens of '*domain*' and '*path*' are common to all URL payloads, which are not useful to the XSS model. However, the normal GMM model establishes the normal profile. Thus, we only use '*domain*' and '*path*' for the normal model.

4.3.4. Vectorization. We use the Word2vec technique to convert each word to a corresponding vector value. The Word2vec model is trained on complete instances of requests and responses, containing all possible tokens. The input is formatted as a list of token sequences after the pre-processing steps. The number of dimensions is set to 200, a common choice for this parameter. For each token, Word2vec will output its corresponding unique vector shown in Figure 4.



```
array([ 1.5022478 , -0.26950246,  1.3691655 , -0.7653921 ,  0.06273928,
       -0.10709091,  0.30377784,  1.024117  ,  0.13801455,  0.44030577,
        0.37202516,  0.10570411, -0.7494039 , -1.231721  , -0.33613563,
        0.24472103,  0.8351226 , -1.2908832 ,  1.3137364 , -0.78922135,
       -0.11358897, -1.0764761 , -0.25355002,  1.3253189 ,  0.65894264,
        0.21480708, -0.09925482, -0.44046026, -0.91457003, -1.3518808 ,
        0.90191746,  0.6104583 ,  0.27702042,  0.12023603, -0.6481594 ,
        0.22481696,  0.35714352,  0.23922716,  0.10704586,  1.2478479 ,
        0.12310014,  0.7453083 ,  0.9263791 , -1.3512938 , -0.14794908,
       -0.77244353,  1.2605963 ,  0.9537559 , -0.22431198,  0.6462313 ,
        0.58187675, -1.4382539 , -1.2943572 ,  0.4814422 ,  0.41691503,
       -1.1092927 ,  0.15643038, -0.71926063,  0.1113877 , -0.6429826 ,
        0.10802718,  0.15009162, -0.79971325, -0.2598313 ,  1.0980293 ,
        0.15477178,  0.5359787 ,  1.2401469 , -0.95034313, -1.5267822 ,
       -0.7137758 , -1.1304439 , -1.5164794 ,  0.05147218, -0.33899426,
       -1.0966481 , -0.16412187, -0.09274975,  1.252316  ,  0.59314126,
       -0.19734745, -0.68967545,  0.8911589 ,  1.1671736 , -0.12406917,
        0.5826899   -0.7976741    1.1798681   -0.83433396   0.61009175
```

Figure 4. Partial vector for the token '*alert()*'

A vector represents the position of a token in the vector space based on its relationship with other tokens. In other words, two closely related tokens should possess neighboring positions for their closeness. Then, for each web transaction that contains multiple tokens, we calculate the average of these vectors to represent this instance, as the input to the GMM model.

4.4. Training and testing

In training, payloads of XSS or normal transactions are processed into vectors as input to a GMM model. The number of components varies from 1 to 10.

In testing, the GMM model generates a probability score that represents the likelihood of a testing instance belonging to any subpopulation of the training dataset. This output score is in the form of log probability. Then we can set up a threshold for classification. For example, if the model is trained on XSS data then the higher this score is, the more likely the tested observation is XSS. The classification rule is that if the score is greater than a set threshold, this instance is classified as an XSS attack; otherwise it is classified as a normal instance.

4.4.1. Two detection models. Our approach uses two GMM detection models. These two models include the one trained with only XSS dataset and the other trained with only normal dataset. Another difference of these two detection models is that the network location tokens of '*domain*' and '*path*' are only used for the normal GMM model.

4.4.2. A dual model to integrate the two detection models. The two GMMs are tested on a same testing dataset that contains both XSS and normal instances. Ideally, these two models should give the same result, i.e., one is true and the other is false, since the ground truth of this instance is either normal or malicious. However, there may be instances that possess similarity to both normal and XSS instances in the training datasets. Intuitively, a consensus decision made after reconciling these two scores may improve the detection performance, lowering the false positive rate and the false negative rate at the same time.

In a dual model, we combine these two scores for a testing instance from the two GMM models by calculating their difference $C = log\ P_1 - log\ P_0$, where P_1 represents the probability score from the XSS model, and P_0 represents the probability score from the normal model. We then vary a control threshold on this difference in order to classify the instance into either a normal or XSS class.

5. Performance evaluation

5.1. Experiment setting

The dataset includes all available web transaction instances in the XSSed and Web09 databases. The sizes of the training and testing datasets used for XSS and normal GMM models are summarized in Table 5. These training and testing instances are randomly selected.

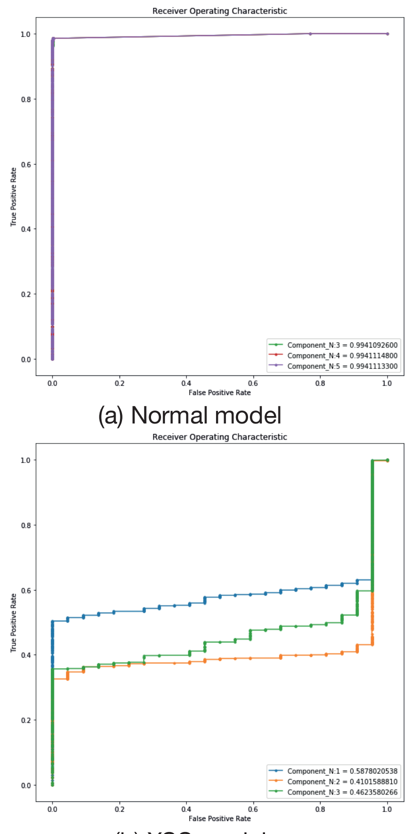
We use Receiver Operating Characteristic (ROC) curve plots to evaluate detection models of different settings. A ROC curve is obtained by displaying pairs of True Positive Rate and False Positive Rate. The True Positive Rate is the ratio of the number of true positives to the total number of XSS instances; the False Positive Rate is the ratio of the number of false positives to the total number of normal instances. A common measure of the goodness of a ROC curve is the Area under the Curve (AUC), which is also used in our analysis.

Table 5. Training and testing datasets

Dataset	XSS	Normal	Total
Training	35884	24561	60445
Testing	10000	10000	20000

5.2. Single-stage models using web request

For models using web request information only shown in Figure 5, the normal model seems to have perfect performance with regard to AUC value. On the other hand, the performance of XSS model is not satisfying. This can be explained by a significant difference between methods of XSS request processing.



(a) Normal model

(b) XSS model

Figure 5. Single-stage detection models using only web requests

From Table 6, we notice that a normal request is very "simple" compared to an XSS request. Instead of having a query in form of “*srch="><script> alert(document.cookie)</script>&x=0&y=0*” in an XSS request, the normal request only has a network location. Such contrast can be found extensively.

Table 6. Normal and XSS request examples

	Normal Request	XSS Request
Original	http://1.assets.lingr.com/room/l1ty5ap5ItK/related	http://www.iiar-anticancer.org/search.php?srch="><script>alert(document.cookie)</script>&x=0&y=0
Input to Normal Model	[‘domain’,’path’]	[‘domain’,’path’,’srch=’,’alert()’,’document.cookie’,’x=’,’y=’]
Input to XSS Model	[]	[’srch=’,’alert()’,’document.cookie’,’x=’,’y=’]

Most of the normal request vectors consist of only ‘*domain*’ and ‘*path*’ tokens. Dominating the testing input, the normal detection model can easily recognize XSS requests. However, this is not the case for XSS instances. Such normal requests do not capture useful tokens like ‘*alert()*’ or ‘*document.cookie*’ in more complex requests, which results in an empty list when it is fed to the XSS model. In fact, the testing dataset only has 22 non-empty input when feeding XSS model, which is shown in Table 7. Hence, testing results of single-stage detection models using web requests only are not meaningful in general.

Table 7. Web requests for testing

	Valid XSS Instance/Total	Valid Normal Instance/Total
Normal Model	10000/10000	10000/10000
XSS Model	9853/10000	22/10000

5.2. Single-stage models using web response

In contrast, a web response has much richer information than a web request as shown in Table 8. Such a response may contain multiple tokens that are useful for both normal and XSS models.

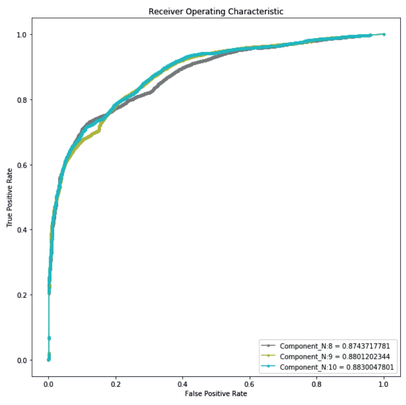
Table 8. Normal and XSS response examples

	Normal Response	XSS Response
Original	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd"> <HTML><HEAD>…	<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd'><html xmlns='http://www.w3.org/1999/xhtml' >…
Input to Normal Model	['javascript:copyit()', 'document.getElementById()', '.value']	['homePage= ', 'alert()', 'document.cookie']
Input to XSS Model	['javascript:copyit()', 'document.getElementById()', '.value']	[’srch=’,’alert()’,’ document.cookie’, ’x=’,’y=’]

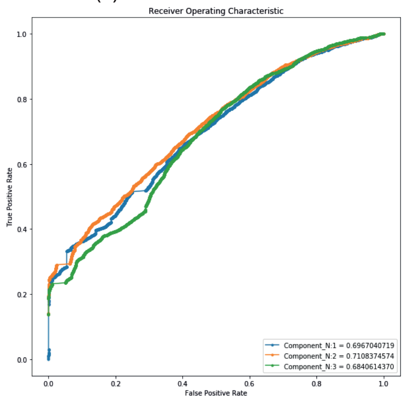
Table 9. Web responses for testing

	Valid XSS Instance/Total	Valid Normal Instance/Total
Normal Model	9398/10000	7544/10000
XSS Model	9398/10000	7544/10000

As in Table 9, web responses generate sufficient testing instances for both the normal and XSS models.



(a) Normal model



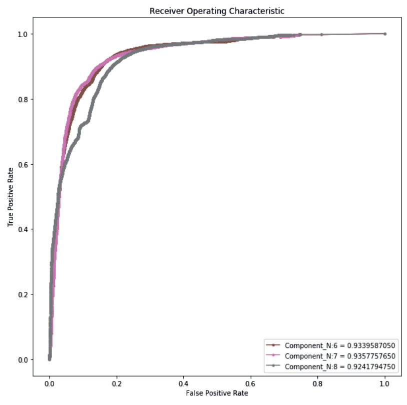
(b) XSS model

Figure 6. Single-stage detection models using only web response

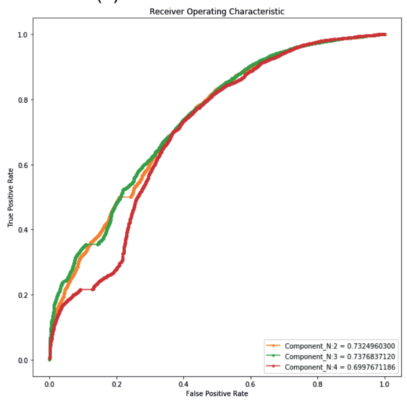
The ROC curves and AUC scores for these models in Figure 6. Overall, it seems that normal detection models are better than XSS models in detection. And the number of components in a GMM model has an impact on the detection performance. However, increasing the components does not always improve the performance. As shown in Figure 6, the AUC score for 3 components is lower than that for 2 components.

5.3. Multi-stage and dual detection models

Multi-stage detection models correlate web response and response tokens as in their input. Figure 7 shows the performance of detection models using both web request and response with different numbers of components. In comparison to the single-stage models in Figure 6, there is an improvement in their performance with respect to the AUC scores.



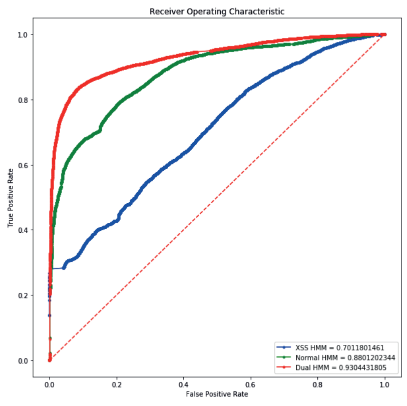
(a) Normal model



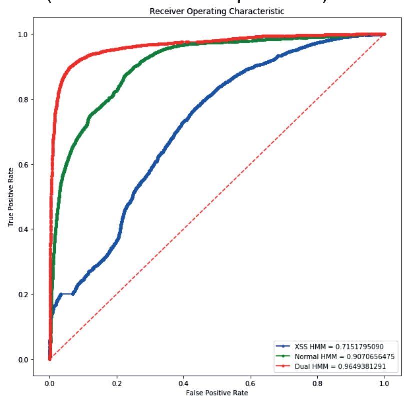
(b) XSS model

Figure 7. Multi-stage detection models using both requests and responses

We test dual models by integrating the two scores from the XSS and normal GMM models. We can do this for single-stage models and multi-stage models respectively. We select the number of components having the best performance in this process. Figure 8(a) shows that the single-stage dual GMM model outperforms the comparable individual single-stage models in Figure 6. Figure 8(b) shows that the multi-stage dual model is better than those individual multi-stage models in Figure 7.



(a) Single-stage dual model using only web responses (GMM with 9 components)



(b) Multi-stage dual model using both requests and responses (GMM with 10 components)

Figure 8. Dual detection models integrating normal and XSS models

There are enough XSS and normal instances in these testing results. We are comfortable to draw the conclusion that multi-stage and dual detection models can improve the accuracy of XSS attack detection.

5.4. Limitation of the results

However, as also showed in several other studies [10], [11], [17] and [18] that use malicious scripts from the XSSed project and very different benign scripts from Dmoz or ClueWeb09, some of the results are too good to be convincing. Therefore, testing out approach on more realistic traffics may yield more useful findings. A couple projects collected large datasets that can be very useful on this front as in [19] and [20]. We can continue to investigate this approach by studying better data collection and feature extraction techniques.

6. Conclusion

Most of the existing solutions of detecting XSS attacks examine the evidence in only one stage. We have studied an approach that looks at the information in both request and response stages and employs a dual model of combining anomaly detection with misuse detection. This approach utilizes the Word2vec technique and Gaussian mixture models. Evaluation using real data coming from two databases of XSS and normal web transactions has shown its effectiveness.

7. Acknowledgement

This work has received partial support from NSF through Award No. 1525485. We like to thank Qiqing Huang, Bayan Al Muhander and Likitha Satish from Johns Hopkins University Information Security Institute for their effort in a previous relevant project.

8. References

[1] K. Spett, "Cross-site scripting", SPI Labs, 2005, pp. 1-20.

[2] "Top10-2017 Top 10", 2017. [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10. [Accessed: 30- Dec- 2017]

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", NIPS, Lake Tahoe, NV, 2013, pp. 3111-3119.

[4] "XSS Filter Evasion Cheat Sheet". [Online]. Available: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet. [Accessed: 30- Mar- 2018]

[5] R. Katipally, L. Yang, and A. Liu, "Attacker behavior analysis in multi-stage attack detection system", Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW'11), New York, NY, USA, 2011, No. 63.

[6] D.H. Lee, D.Y. Kim, and J.I. Jung, "Multi-Stage Intrusion Detection System Using Hidden Markov Model Algorithm", ICISS, Seoul, 2008, pp. 72-77.

[7] U.S.K.P.M. Thanthrige, J. Samarabandu, and X. Wang. "Intrusion Alert Prediction Using a Hidden Markov Model", arXiv:1610.07276, 2016.

[8] A. Almutairi, D. Parish and J. Flint, "Predicting multi-stage attacks based on IP information", ICITST, London, 2015, pp. 384-390.

[9] L. Yan, "Machine learning for the web security", Mechanical Industry Press, 2017.

[10] Y.H. Wang, C.H. Mao, and H.M. Lee, "Structural Learning of Attack Vectors for Generating Mutated XSS Attacks", TAV-WEB, 2010.

[11] R. Pelizzi, and R. Sekar, "Protection, Usability and Improvements in Reflected XSS Filters", ASIACCS 2012, New York, NY, 2012.

[12] T. Pietraszek, C.V. Berghe, "Defending against injection attacks through context-sensitive string evaluation", Recent Advances in Intrusion Detection, Berlin, Heidelberg, 2005, pp. 124–145.

[13] S. Goswami, N. Hoque, D.K. Bhattacharyya, and J.K. Kalita, "An Unsupervised Method for Detection of XSS Attack", I. J. Network Security, 2017, pp. 761-775.

[14] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection", IEEE Communications Surveys & Tutorials, 2016, pp. 1153–1176.

[15] N. Moustafa, G. Creech, and J. Slay, "Big data analytics for intrusion detection system: Statistical decision-making using finite dirichlet mixture models", Data Analytics and Decision Support for Cybersecurity, 2017, pp. 127–156.

[16] N. Moustafa, G. Misra and J. Slay, "Generalized Outlier Gaussian Mixture technique based on Automated Association Features for Simulating and Detecting Web Application Attacks", IEEE Transactions on Sustainable Computing, 2018.

[17] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross Site Scripting Detection Based on Deep Learning", Proceedings of the 2018 International Conference on Computing and Artificial Intelligence (ICCAI 2018), New York, NY, 2018, pp. 47-51.

[18] F.A. Mereani, and J. M. Howe, "Detecting Cross-Site Scripting Attacks Using Machine Learning", Advances in Intelligent Systems and Computing, Cham, 2018, pp. 200-210.

[19] N. Moustafa, and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)", Military Communications and Information Systems Conference (MilCIS), 2015.

[20] N. Moustafa, and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set", Information Security Journal: A Global Perspective, 2016, pp. 1-14.