Detection and mitigation of Web Services Attacks using Markov Model

Vivek Relan
RELAN1@UMBC.EDU
Bhushan Sonawane
BHUSHAN1@UMBC.EDU
Department of Computer Science and Engineering, University of Maryland, Baltimore County, MD 21250, USA

Keywords: Web service, Web service attacks, and Markov Model.

Abstract

We introduce Markov Model for Web services attacks detection and mitigation. This model is capable of preventing not only injection attacks like SQL injection, Cross-Site Scripting (XSS), buffer overflow but also detect abnormal Web service client behavior. We adopted the well known technique based on Markov model for detecting Web application attacks. Our system contains two Markov models. First model captures Web service input attributes and second model deals with legitimate Web service client behavior. In our experiment, we have generated a test data set by building a shopping cart Web service. We validated the efficacy of proposed method by subjecting test Web service to various attacks.

1. Introduction

Service Oriented Architecture (SOA) [1] is widely used to provide interoperable services. As SOAs are designed to reuse existing services, IT companies are saving a cost. Web services [2] are a way to implement SOAs. In order to build a Web service, a set of standard protocols are used such as XML [5], XML Schema [6], WSDL (Web Services Description Language) [4] and SOAP (Simple Object Access Protocol) [3]. Service provider publishes its application interfaces through WSDL and consumer application consumes it by sending a SOAP request. For example, Web site can present local search capability by using Yahoo search API. This will save cost and do not require reinventing a wheel.

However, Web services threats are increasing with its widespread usage. Most of Web services threats are similar to Web application vulnerabilities like SQL injection [7], buffer overflow [8], session theft etc. In addition, Web services risk is increasing because they are open on Web and communicate with corporate data. Due to increasing Web services security incidents, it is found in survey [9] that most of IT firms have slowed down the adoption of SOA. Therefore, there is a need to detect and prevent Web services attacks.

In order to detect and mitigate Web services threats, various signatures based attack detection schemes [14, 15, and 16] are used. These attack signatures technique compares the existing attack detection string against the input SOAP request. Based on the matching criteria, they filter the malicious request. But, these techniques are not fool proof way to mitigate the attack because new attack vectors are discovered everyday and it is not feasible to update those attack signatures. Moreover, it degrades the system performance.

This paper presents Web services attack detection and mitigation system using Markov model. We have used the similar approach [17] used in past for detecting and mitigating Web application attacks using Markov model. The proposed method comprises two Markov models. The first Markov model is built for every attribute of Web service API. The second Markov model builds legitimate Web service client’s behavior. First set of Markov model is used for detecting Web services injection attacks whereas second set of Markov model will be used to detect malevolent Web services client behavior. We believe that proposed system for Web services attacks detection and mitigation system will strengthen
The organization of this paper is as follows. Section 2 discusses related work for detecting various approaches of Web services attacks. Section 3 introduces notations used in this paper. In section 4, Markov model for Web services attack detection and mitigation is presented. Our system architecture is described in section 5. Section 6 presents experimental results and evaluation of proposed approach. We discussed the limitation of proposed approach in section 7. Finally, section 8 concludes the paper.

2. Related Work

Various signatures based approaches [14, 15, and 16] were presented in the past to detect the attacks in Web application and Web services. However, these methods are not scalable because attack signatures database needs to be updated with discovery of new attack.

In the past, well known Machine Learning technique like learning based anomaly [11, 12, and 13] have been proposed to detect and mitigate injection attacks. These techniques monitor the input request and observe the values of an input attributes. They check whether a given input attribute is valid by comparing it against the legitimate model of an input attribute. Based on the result of model, they detect attacks. However, these techniques are only applicable to injection attacks. They do not capture the sequencing of user behavior which leads to more sophisticated attacks like authentication bypass.

Cheng et al presented user behavior surveillance system [17], a novel Embedded Markov Model to detect various Web application attacks. This model not only considers injection attack but also captures the sequencing of user behavior. Their model can detect unreasonable transition of user behavior and mitigate authentication bypass attack.

In this paper, we propose a similar approach [17] to detect and mitigate Web services attacks using Markov model. Our proposed system will detect both injection attacks like SQL injection, XSS, buffer overflow etc and malevolent behavior of Web service client. To the best of our knowledge, this is the first the paper which discusses the detection and mitigation of Web services attacks using Markov model.

3. Notation

Throughout this paper, we are going to use the following notations. We denote a Web service as WS. C = {C_1, C_2, ..., C_m} is a set of Web service clients. A set of Web service APIs for WS is denoted by API = {API_1, API_2, ..., API_n}. The q^{th} input attribute of API_j is denoted by A_{j,q}. The value of input attribute is considered as string consisting of Unicode characters. Such a string is denoted by a sequence of characters as

S = \{s_i|s_i \in \text{Unicode characters}\}.

The character transition probability of s_i is marked with CT(s_i). The set of all input attributes of WS for API_j is denoted by Q_j = \{A_{j,1}, A_{j,2}, ..., A_{j,k}, 1 <= q <= k\}.

q_{i,j} denotes Web service client C_i sends the SOAP request for API_j to WS. Generally, SOAP request is sent over HTTP via POST method. The attributes required for API are specified in SOAP envelope.

Figure 1. Web service client requests for particular Web service WS

Figure 1 shows the entities in our model along with their notations. It depicts various Web service clients access Web service APIs of WS. We have developed a shopping cart WS. Figure 2 illustrates the example of a Web service API request and corresponding notations mentioned above. Client C_i sends request to API_j which is ‘GetProductsInformationByID’. The attributes of API_j are ‘ID’ and ‘SecurityToken’. Q_j = \{A_{j,1}, A_{j,2}\} = \{ID, SecurityToken\}.

Let’s now define the legitimate transitions of Web service API calls. Legitimate transitions of Web
service API calls define the reasonable flow of Web service APIs for a particular Web service. Here, we used shopping cart Web service for illustration. This Web service provides ten basic operations. To access the product information, Web service client has to authenticate himself by calling 'Authenticate' API. The legitimate transitions of Web service API calls for shopping cart Web service is presented in figure 4.

4. Markov Model for Web service

This section introduces our two sets of Markov model for detecting Web service attacks. In order to detect injection attacks like SQL injection, XSS, buffer overflow, the first set of Markov model is built for each attributes of Web service API. While the second set of Markov model builds legitimate Web service API call transitions. This is useful to detect unreasonable API call transitions from Web service clients. Note, these models are already used in context of detecting and mitigating Web application attacks.

4.1. First Markov Model

The first set of Markov model is built for each attribute of Web service API to detect injection attacks. In order to build a first Markov model, data set is collected for a particular Web service WS. This data set includes all the accessed Web service APIs \(A_{j,q}\) and its corresponding attribute values \(s_i\). After collecting sufficient data, Markov model for each attribute \(A_{j,q}\) by using \(s_i\) and \(CT(s_i)\). Thus, the first set of Markov model shows all the possible transition paths for each input attribute \(A_{j,q}\). If first Markov model cannot find a transition path for a given attribute value, it declares a corresponding attribute value as possible attack string. The probability of \(A_{j,q}\) is product of \(s_i\) and \(CT(s_i)\) as shown in equation 1.

\[
P(A_{j,q}) = P(s_1, s_2, ..., s_n)
\]

\[
P(A_{j,q}) = P(s_1) \times \prod_{2 \leq i \leq k} P(s_i) \times CT(s_i) -- -- (1)
\]

First Markov model for input attribute 'username' of Web service API 'Authenticate' is shown in Figure 4. The Markov model is built from data set gathered for Web service WS. Let \(C_i\) calls Authenticate API with username=admin. Using Markov model for username attribute, we can verify the accuracy of it.

\[
P(\text{admin}) = 1 \times 0.3 \times 1 \times 0.4 \times 1 \times 0.6 \times 1 \times 0.5 = 0.036
\]

As \(P(\text{admin})\) is greater than 0, our model accepts the string. However, when \(C_i\) calls Authenticate API with username=adi<script>, then probability of it is zero.

\[
P(\text{adi} < \text{script}>) = 1 \times 0.3 \times 1 \times 0.4 \times 1.0 \times 0 = 0.0
\]

Thus, our model will declare such input as possible attack vector. Therefore, once the Markov model for an attribute is built, it is easy to predict whether a given attribute value is an attack string or legitimate input string.

4.2. Second Markov Model

The second set of Markov model builds the legitimate transitions of Web service API calls for a particular Web service. This will help in detecting unreasonable transitions of Web service API calls from Web service client. Thus, malevolent Web service client behavior is easily detected by second Markov model. Building first Markov model is preliminary step for building second Markov model. Once, first Markov model is built for each attributes of API, \(Q_j\) for \(API_j\) is calculated as,

\[
Q_j = \prod_{1 \leq q \leq k} P(A_{j,q})^{1/k} -- -- (2)
\]
Legitimate transitions of Web service API calls can be calculated by finding the probability of every API transitions and probability of \( Q_j \) for each \( API_j \). Equation 3 calculates the probability of legitimate transitions of Web service API calls.

\[
P(Q_1, Q_2, Q_n, API_1, API_2, ..., API_n) = P(API_1) \times \prod_{2 < m \leq n} P(API_m | API_{m-1}) \times P(Q_m | API_m)
\] — (3)

Once the second Markov model is built for all the legitimate transitions of Web service API calls, it can predict whether a Web service client behavior is normal or malevolent. For this, we calculate the probability of API transition for a given behavior and compare it with threshold for normal behavior. If the probability is less than threshold, then model will declare it as a malicious behavior.

**5. Architecture**

Overall system is divided into two phases: learning phase and learned phase. In learning phase, system learns Markov model for each input attribute and Web service API transition for Web service (WS). For this, dataset is gathered by capturing normal user’s SOAP requests and extracted information is fed to our model. The first Markov model is built by calculating character transition probability as described in section 4.1. The second Markov model is built for legitimate Web service API transitions as described in section 4.2.

We developed a Web service proxy (WS proxy) which sits between Web service client (WS client) and target Web service (WS). Figure 5 presents the system architecture. In learned phase, WS proxy traps the SOAP request and computes the probability of each parameters of SOAP request. If the estimated probability of input attribute is zero, then WS proxy will drop the request. Otherwise, it will forward the request to WS. In addition, WS proxy remembers the WS API transitions of each user.

If user performs the unreasonable WS API transition, then estimated probability of such sequence is zero. WS proxy detects such malicious sequences and protects WS.

**6. Evaluation**

In order to validate the efficacy of proposed method, we need a data set for a particular Web service. However, there is no standard data set available for it. Therefore, we built a shopping cart Web service as mentioned in figure 4. There are ten Web service APIs and each API has at least two input parameters. In order to gather legitimate product details, we developed a script which downloads 36,000 different product details from Yahoo! product database. Then, we built Markov model for ‘productName’ attribute using this data. Figure 6 shows character transition frequency for each UTF-8 character. It can be found that alpha-numeric characters are frequently occurred in input values. In shopping cart Web service, ‘productName’ parameter is vulnerable to SQL injection. We selectively turned off the WS proxy functionality. When user provides SQL injection string (‘OR 1=1 limit 10; – -’) in ‘productName’ parameter, then he is able to gather product details without having security token.
Detection and mitigation of Web Services Attacks using Markov Model

Figure 6. Character transition frequency for 'productName' attribute

Figure 7 shows above mentioned scenario in absence of our system. In presence of WS proxy, it detects such malicious user’s input by computing the probability of 'productName' attribute using first Markov model. As there is no such character transition from '''' to 'O', its probability is zero. WS proxy drops such request. Figure 8 shows such scenario in presence of our system. Figure 9 shows graph of length of input string versus $-\log(P(string))$. It is found that probability of input attribute decreases with increase in its length. For our dataset, estimated probability of input attribute tends to zero, when its length is more than 50. WS proxy detects such buffer overflow attack in input attribute. Currently, we have not incorporated online detection of unreasonable WS API transition functionality in WS proxy. However, our offline model detects such malicious API transitions. In our shopping cart application, it is necessary that user has to call 'Billing' API before calling 'Shipping' API. When malicious user calls 'Shipping' API without calling 'Billing' API the probability of such API transition sequence is zero. Hence, second Markov model helps in detecting such malicious user behavior. Figure 10 shows the detection of malicious user behavior in our shopping cart application.

7. Limitations

It is found that proposed second Markov model is not sufficient for detecting malicious user behavior. Sup-
pose user can invoke 'GetProductByName', 'GetProductByID', 'GetTopTenProducts' and 'GetProducts' APIs with or without calling 'Authenticate' API first. However rest of the APIs like 'AddItem', 'CheckOut' etc are invoked after invocation of 'Authenticate' API. Following training data dataset captures legitimate WS API transitions. In training data set, there exists transition from 'GetProductByID' to 'AddItem' and 'GetProductByName' to 'AddItem'. Therefore, when malicious user performs one of the following API transitions without calling 'Authenticate' API: our second Markov model is incapable of detecting such malevolent user behavior.

8. Conclusion

The Markov models for Web service attacks detection and mitigation have been designed. Our experimental results show that proposed Markov models are capable of detecting both injection attacks and simple malevolent user behavior.

Acknowledgments

We would like to thank Dr. Tim Oates for his consistent support and useful suggestions throughout the project.

References

[1] Service-oriented architecture (SOA)

[2] W3C notes on Web Services
http://www.w3schools.com/webservices/default.asp


[4] W3C notes on Web Services Description Language (WSDL) http://www.w3.org/TR/wsdl


http://www.w3.org/XML/Schema

[7] SQL injection

[8] Buffer overflow
http://en.wikipedia.org/wiki/Buffer_overflow

[9] Security threats hinder SOA, Case study,


