

SQL Injection Prevention System using a efficient Machine Learning Approach

Mrs.M.Saratha¹, Mr.T.Thangarasan²

¹Assistant Professor, Department of MCA, M.Kumarasamy College of Engineering (Autonomous)

²Assistant Professor, Department of CSE, K.S.R College of Engineering (Autonomous)

Abstract - Web application firewalls are an important safeguard for any online software system. SQL Injection attacks are the most serious security issue for insecure online applications in the Internet age. With the growing threat of SQL Injections, Web Application Firewalls (WAF) must be updated and tested on a regular basis to keep attackers at bay. As technology advances, the number of attackers seeking to attack applications expands, resulting in a plethora of new ways for them to gain access to the system. As a result, existing systems are struggling to keep up with new hackers and new technologies in order to entirely rescue the system. The white box testing and static analysis approach in the existing WAF requires access to source code. Model-based testing necessitates a larger number of rules. For detecting SQL injection attacks, black box testing is ineffective. Machine learning is an artificial intelligence application that allows computers to learn and improve on their own without having to be explicitly designed. Collaboration between machine learning and web application firewalls improves the efficiency of the current system. Unsupervised Learning Technique is the method employed in this paper. The k-means approach, which is commonly used for clustering issues, is employed for unsupervised learning. The system's flow can be described as follows. When a Web application's end user makes a request, the request's values are retrieved and transmitted to the SQL injection detector, which provides two layers of protection. For low-level attacks, patterns are generated utilising CFGs in the first layer of security. Unsupervised Learning Algorithm is used to train the second layer of protection for high-level assaults.

Key Words: *SQL Injections, SQL Injection Detector, Two layer Security, Unsupervised Learning Technique*

1.INTRODUCTION

A WAF or web application firewall helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as cross-

site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, among others. A WAF is a protocol layer 7 defense (in the OSI model), and is not designed to defend against all types of attacks. This method of attack mitigation is usually part of a suite of tools which together create a holistic defense against a range of attack vectors.

By deploying a WAF in front of a web application, a shield is placed between the web application and the Internet. While a proxy server protects a client machine's identity by using an intermediary, a WAF is a type of reverse-proxy, protecting the server from exposure by having clients pass through the WAF before reaching the server.

A WAF operates through a set of rules often called policies. These policies aim to protect against vulnerabilities in the application by filtering out malicious traffic. The value of a WAF comes in part from the speed and ease with which policy modification can be implemented, allowing for faster response to varying attack vectors; during a DDoS attack, rate limiting can be quickly implemented by modifying WAF policies.

A WAF can be implemented one of three different ways, each with its own benefits and shortcomings:

- A network-based WAF is generally hardware-based. Since they are installed locally they minimize latency, but network-based WAFs are the most expensive option and also require the storage and maintenance of physical equipment.
- A host-based WAF may be fully integrated into an application's software. This solution is less expensive than a network-based WAF and offers more customizability. The downside of a host-based WAF is the consumption of local server resources, implementation complexity, and maintenance costs. These components typically require engineering time, and may be costly.
- Cloud-based WAFs offer an affordable option that is very easy to implement; they usually offer a turnkey installation that is as simple as a change in DNS to redirect traffic. Cloud-based WAFs also have a minimal upfront cost, as users pay monthly or annually for security as a service. Cloud-based WAFs can also offer a solution that is consistently updated to protect against the newest threats without any additional work or cost on the user's end. The drawback of a cloud-based WAF is that users hand over the responsibility to a third party, therefore some features of the WAF may be a black box to them.

2. PROPOSED SYSTEM

There is no automatic detection system for identifying and preventing SQLi attacks in the current

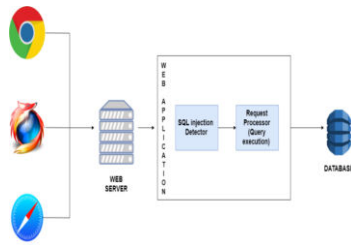
system. It does the detection using a set of rules. The system examines the query against each and every rule before detecting an attack. If a certain sort of rule isn't kept in the set and an attack is launched, the system will allow the query to be run because that rule isn't kept in the set. It's also tough to keep track of and test the regulations. In addition, for large applications, we will need to develop more complicated rules. The machine learning-based method for blocking SQL injections aids in the server-side exploitation of any database. A component for the server side of this system is being developed. The system will check before performing the client's request.

This system has two levels of security: the first is the patterns generated by context free grammar rules, and the second is the comparison of values with the patterns generated by the given rules for sql attacks. The machine learning algorithm (k-means) in the second level of security groups the pattern based on the given data set, categorises fresh data into one of those clusters, and detects the injection. Figure 1 depicts the System Architecture, which explains how the system is put together. The client uses the web browser to submit the appropriate requests to the web server.

The web browser sends the client's input to the web server, which has a security component called the SQL Injection Detector. It looks for SQLi Attacks in the values given by the user. If the values are correct, the request is passed to the Request Processor, who evaluates it and runs the query. The database is then accessed, and the client's request is processed and executed.

3. SYSTEM ARCHITECTURE

This system consists of three modules, namely, URL Intercept Engine, Context-Free Grammar for SQLi Attacks and Classify pattern through Machine Learning.



A. URL Intercept Engine

The most vulnerable aspect of a web application is the text fields, which are where the majority of sql injection attacks occur. End users or hackers that want to mess with a company's database type in some harmful sql queries in the text field, which can be appended to the sql query previously defined on the server side, causing the database to be affected. As a result, the first module pulls data from text fields and sends them to a pattern checking algorithm.

B. Context Free Grammar for SQLi Attacks:

The initial degree of protection against sql injection attacks is provided by the second module. This module builds the sql injection attack pattern using context-free grammar rules for sql injection attacks. The retrieved value from the first module is compared to the rule that creates the various assault patterns. If the extracted value matches the attack pattern generated by the Context free grammar rules, the value is transferred to the second level security, which determines whether or not the value entered is harmful. The value entered is sent to the second level security even if it does not meet the pattern generated by the Context free grammar rules.

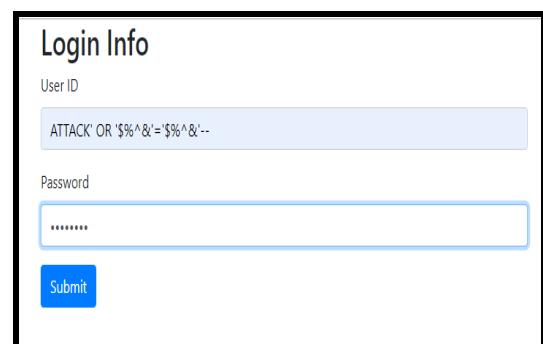
C. Classification of pattern through Machine Learning

This module provides the system with the second level of security. This module contains an unsupervised

machine learning algorithm that groups different sorts of attacks into separate clusters and tests the value provided to it by determining whether the value is fit for the cluster and whether it contains harmful queries. If not, the request is allowed to run the query and access the database by the system. If it matches the cluster associated with any attack pattern, the system refuses to execute the request and keeps the system in the same state.

4. EXPERIMENTAL RESULTS

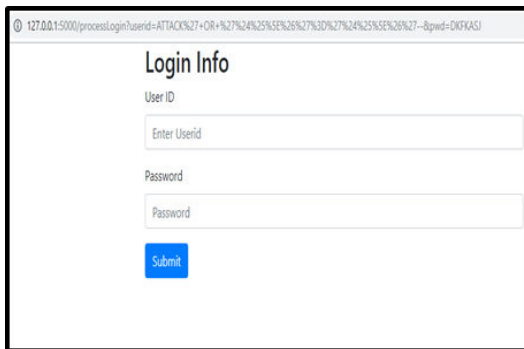
The forms submitted by the end user, as well as any injected values into the url by end users or hackers, provide input to our model. Finally, all of the values submitted in the form or injected in the url are inserted into a query, which is then run to contact the database and retrieve or enter new data. The training datasets for our Machine Learning Algorithm are a set of SQL statements used to categorise the pattern employed in piggyback and union attacks, as well as some of the attack patterns used to classify boolean attacks.



```

Your Password: DSFS
["ATTACK'or(1=1)--", "ATTACK'unionselect", "ATTACK'se
ATTACK'OR'%^&'='%^&'--
not found in cfg
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\Rishi\AppData\Roaming\nltk_d
[nltk_data] Package stopwords is already up-to-date
Prediction
OR '%^&'='%^&'--
ATTACK' OR '%^&'='%^&'-- : boolean attack [1]
found attack pattern

```



5. CONCLUSION

The suggested solution employs a machine learning approach to detect and prevent SQL injection attacks in web applications via user requests. URL Intercept Engine, Context-Free Grammar for SQLi Attacks, and Classify Pattern using Machine Learning are the three modules of the system.

The URL Intercept Engine extracts the values submitted by the user on the client side, which are subsequently transmitted to the first level of protection. The Context-Free Grammar rules are used to detect the pattern in the first level of security. The patterns are fed into the Machine Learning Algorithm, which uses the pattern to classify the value into clusters. The client's request will be revoked if the value is judged to be harmful by the implemented security. If the request isn't malicious, it will be processed regularly.

REFERENCES

1. A. Doupe, M. Cova, and G. Vigna, "Why johnny can't a^A~ Zt pentest: An analysis of black-box web vulnerability scanners," in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, 2010, pp. 111–131.
2. A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in Proc. 31st Int. Conf. Softw. Eng., 2009, pp. 199–209.
3. A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou, "Sqlprob: A proxy-based architecture towards preventing sql injection attacks," in Proc. 2009 ACM Symp. Appl. Comput., 2009, pp. 2054–2061.
4. Dennis Appelt, Cu D. Nguyen, Annibale Panichella and Lionel C. Briand, "A Machine-Learning -Driven Evolutionary Approach for Testing Web Application Firewalls", IEEE Transactions on Reliability, vol. 67, pp. 733 - 757, 2018.
5. D. Appelt, N. Alshahwan, and L. Briand, "Assessing the impact of fire-walls and database proxies on SQL injection testing," in Proc. 1st Int. Workshop Future Internet Testing, 2013, pp. 32–47.
6. D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for sql injection vulnerabilities: An input mutation approach," in Proc. 2014 Int. Symp. Softw. Testing Anal., 2014, pp. 259–269.
7. E. Al-Shaer, A. El-Atawy, and T. Samak, "Automated pseudo-live testing of firewall configuration enforcement," IEEE J. Sel. Areas Commun., vol. 27, no. 3, pp. 302–314, Apr. 2009.
8. Inyong Lee, Soonk Jeong, Sangsoo Yeo and Jongsub Moon, 'A novel method for SQL injection attack detection based on removing SQL query attribute values', Elsevier, vol. 55, pp. 58 - 68, 2012.
9. J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box web application vulnerability testing," in Proc. IEEE Symp. Security Privacy, 2010, pp. 332–345.
10. J. Hwang, T. Xie, F. Chen, and A. X. Liu, "Systematic structural testing of firewall policies," in Proc. IEEE Symp. Rel. Distrib. Syst., 2008, pp. 105–114.