2020 15th International Conference on Computer Engineering and Systems (ICCES) | 978-0-7381-0559-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICCES51560.2020.9334626

Evaluating the Modsecurity Web Application Firewall Against SQL Injection Attacks

Basem Ibrahim Mukhtar
Nile University, Cairo, Egypt
b.mukhtar@nu.edu.eg

Marianne A. Azer
National Telecommunication Institute
Nile University, Cairo, Egypt
mazer@nu.edu.eg

***Abstract*—SQL injection attacks target databases of web servers. The ability to modify, update, retrieve and delete database contents imposes a high risk on any website in different sectors. In this paper, we investigate the efforts done in the literature to detect and prevent the SQL injection attacks. We also assess the efficiency of the Modsecurity web application firewall in preventing SQL injection attacks.**

Keywords*— *Firewall, Modsecurity, SQL injection, WAF, Web attacks

1. INTRODUCTION

Web applications have the advantage of being publicly accessible from everywhere around the world, and their services are available to almost anyone. A lot of information is distributed daily through web applications, it can include private and confidential data which are stored in the database of the webserver along with other data specific to the website itself.

When malicious users want to get confidential information or bring damage for a web application, they can provide unintended input to the application instead of a normal user input [1]. This unintended input which forms SQL statements is known as the SQL Injection (SQLI) attack [1]. According to OWASP top 10 [2], SQLI is one of the most harmful and common attacks which target the database of the webservers. SQLI involves injecting SQL commands in the input data field to interfere with the commands sent to the database to execute attacker's commands instead of the developer intended commands [2]. Figure 1[3] illustrates the SQLI attack methodology. Statistics show that digital security-attacks cost the economy nearly $50 billion yearly on average, and SQL injection attacks contribute by more than 20% of this number [4]. Combating web application attacks has been a major concern for many years. There are two ways typically used to defend a web application [5]. The first is the sanitization of inputs from users. It involves the usage of sanitization methods or functions specific to each language such as php, python or others to process inputs from the users before querying the database [5]. The second is to employ a protection component such as Web Application Firewall (WAF) or application delivery controller which acts as a reverse proxy intercepting each request from the user and using filters to remove malicious payloads [5]. In this paper, we test and evaluate the employment of Modsecurity WAF to protect a vulnerable web application against SQL injection. The remainder of this paper is organized as follows. Section 2 classifies and surveys the work done in the literature for detecting and preventing SQL injection attacks. In Section 3, our contribution in evaluating Modsecurity web application firewall against SQL injection attacks through experimental work is presented. Finally, in section 4 conclusions and future work are discussed.

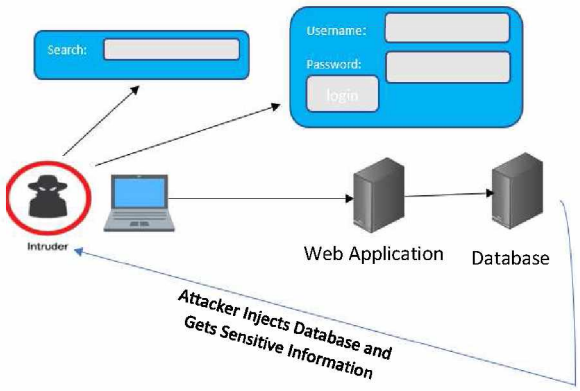

Figure 1: How SQLI Works [3]

2. RELATED WORK

In this section, we classify and survey the research efforts done in the literature in the area of SQL Injection (SQLI) detection and prevention.

978-0-7381-0559-8/20/$31.00 ©2020 IEEE

i. SQLI *Prevention Using Regular Expressions and Escape Strings*

The authors in [6], proved that using the combination of regular expressions, prepare statement, and MYSQL escape string techniques is efficient and secure in protecting against SQLI injection. The regular expression is used to convert the URL to the proper format [6]. The authors in [6] introduced a technique called "prepare statements" to avoid SQLI as the program repeatedly executes the same SQL query with efficiency. The escape string was proposed to filter the forbidden characters in the text box where the users insert their input [6].

ii. *SQLI Prevention Using Hashing, Syntax Awareness and Signature Matching*

An approach to prevent SQLI in the embedded SQL queries was proposed in [7]. It relies on hashing the original and generated queries and comparing them to detect any malicious queries. In addition, syntax awareness is used to prevent SQLI stored procedures by using signatures and regular expressions. The same hashing technique proposed above was used in [8] to block malicious SQL queries. However, it only addressed the authentication queries, not all SQL queries. Signatures, fingerprinting and pattern matching techniques were used in [9] to differentiate malicious queries from the genuine ones. The disadvantage of this technique is being unable to detect zero-day attacks or unknown malicious patterns.

iii. *SQLI Detection and Prevention using Machine Learning Techniques*

The authors in [10] showed that, using machine learning techniques, can increase the accuracy and efficiency of Modsecurity Web Application Firewall (WAF) to detect and prevent SQL injection attacks. The usage of supervised Machine Learning (ML) on a generated data set containing attack patterns of SQLI to predict and detect malicious SQL queries was explored in [11]. An automated approach using ML techniques to fix WAF after successful exploitation to prevent any future attacks was proposed in [12]. The authors in [13] used supervised ML methods to detect anomalous behavior in SQL queries and proved that an intrusion detection system using the above technique can increase the chances to detect SQLI vulnerabilities.

iv. SQLI Prevention Using Database Embedded Tools

A tool implemented in the database to detect malicious queries was introduced in [5]. This approach is different than the common tools which are usually at the application level not the database level. The authors showed that their tool had a low performance impact and high efficiency compared to other protection mechanisms.

v. SQLI Prevention Using Client-Side Controls

The authors in [14], emphasized that usually the defenses are server side deployed. They proposed a client-side protection mechanism to filter SQLI queries before reaching the server.

vi. SQLI Detection Using Source Code Review, Static and Dynamic Analysis

The authors in [15] proved that using text mining approaches to detect SQLI vulnerabilities in java code is not applicable, and other models should be tested. A hybrid approach to detect SQLI using static and dynamic analysis was used in [16]. The downside of this approach is that it is only theoretical, and its performance needs to be tested. The same approach was used in [17] using combined static and dynamic analysis and proved its efficiency in terms of computing time, response time and detection rate.

vii. SQLI Detection Using Blackbox Testing

The authors in [18] focused on improving the efficiency and accuracy of vulnerability scanners to detect SQLI vulnerabilities using object-oriented approach.

Table1. SQLI Detection and Prevention Approaches

Approach	Description	Advantages	Disadvantages
SQLI Prevention using Regular Expressions and Escape Strings [6]	Using regular expression, prepare statement, and MySQL escape string techniques to filter and prevent SQLI attacks.	The technique proved its efficiency in blocking SQL injection.	The performance overhead in implementing the prevention methods has not been tested yet.
SQLI Prevention Using Hashing, Syntax Awareness and Signature Matching [7][8][9]	The use of hashing to make a digital signature for the genuine query and then comparing each query against it. The signature matching is like filtering the query for known attack signatures.	The advantage of hashing is being able to detect any zero-day attack. While the advantages of signature matching maybe its low performance and easy implementation.	The disadvantage of Hashing in [7] and [8] is the usage of SHA1 algorithm which is proved to be vulnerable to a dangerous attack. The drawback of signature is inability to detect zero-day attacks.

SQLI Detection and Prevention using Machine Learning Techniques [10][11][12][13]	Using Supervised and unsupervised Machine Learning (ML) techniques to enhance the detection of SQLI injection and the also the repairing of WAF after successful attacks.	ML Techniques can increase the efficiency in detecting and preventing SQLI attacks.	The main disadvantage of ML implementations is the lack of datasets to train the algorithm.
SQLI Prevention Using Database Embedded Tools [5]	Implementing a tool inside the database to detect SQLI attacks before execution stage.	The main advantage for this tool is solving the semantic mismatch problem i.e. the gap between the perception of the developers of how SQL queries are executed and the way they are execute.	The tool was tested inside MYSQL database only. It needs to be tested on different types of database to prove its efficiency.
SQLI Prevention Using Client-Side Controls [14]	An approach aiming to prevent SQLI attacks at the client-side not server-side.	Preventing the malicious SQL queries before reaching the server.	The need to embed this tool inside every browser, which requires a lot of modification to all client-side browsers.
SQLI Detection Using Source Code Review, Static and Dynamic Analysis [15][16][17]	Using code review, static and dynamic analysis for SQL queries to detect SQLI attacks.	Theoretically, using these approaches SQLI attacks can be stopped.	The proposed methods need further testing for efficiency and performance.
SQLI Detection Using Blackbox Testing [18]	An automated vulnerability scanner to detect SQLI vulnerabilities.	Compared to known vulnerability scanners, it is the only scanner to detect tautology SQLI vulnerability.	The study does not test the solution in terms of performance and stability.

3. EXPERIMENTAL WORK

In this section, we provide our evaluation of Modsecurity WAF against SQL injection attacks. The lab set up, methodology and results are presented in sections 3.1, 3.2, respectively.

3.1 Lab Setup and Methodology

A WAF is a hardware appliance or software residing in front of the web server to intercept and inspect the traffic arriving the server to look for signatures of attacks in packets [10]. The captured attack packets can be logged, filtered or processed according to certain rules [10]. Modsecurity is an open source WAF aimed to protect web application from attacks to provide logging, monitoring and access control [19]. Modsecurity takes its actions according to a set of rules by means of regular expressions [19]. The OWASP Modsecurity core rule set is a set of easily pluggable set of common attack rules that are updated periodically to protect from the common attacks, its latest version is version 3.

We performed our tests using the software and components in Table 1.

Table 2. Software Tools of the Experimental Work

Tools	Description
Ubunto	OS version 18.04
Apache2	web server
DVWA	An intentionally vulnerable web application used for testing and training
Modsecurity	WAF version 2.9.2
CRS	Modsecurity Core Rule Set version 3
SQLMAP	A tool used for detecting and exploiting SQL injection vulnerabilities

Our methodology for testing is shown in Figure 2. It involves the following phases.

Phase 1: Testing DVWA using SQLMAP before installing the Modsecurity WAF and assuring the existence of vulnerabilities.

Phase 2: Installing Modsecurity in embedded mode, which means that the WAF will be a module inside apache, and not a separate appliance. The reason for choosing this type of deployment is its suitability for testing and to avoid consuming a lot of resources. In production environments, a separate appliance would be a more robust solution. In this phase, we start by running SQLMAP without any attributes against the target URL and examine the results.

Phase 3: We follow on by adding some SQLMAP options, which are specific modules aimed to evade WAF protection or filtering, and we proceed by recording the results.

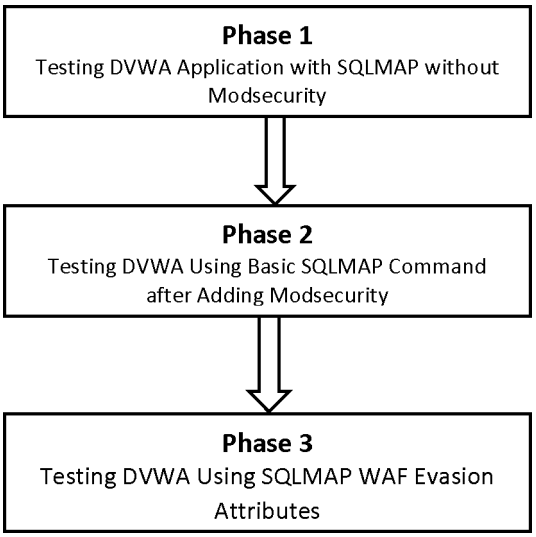

Figure 2: SQL Injection Testing Methodology

3.2 Results

In this section, we present the results obtained in the different phases.

i. Phase 1 Results

The initial testing for SQLI in DVWA website shown in Figure 3 shows the existence of SQLI vulnerability. We executed the basic SQLMAP command without adding any additional parameters. The command consists of 3 parts, the first is the URL of the DVWA website, the second is the cookie parameter we extracted after initially logging in to the DVWA website. The third part is the DBs parameter used to enumerate and get information about the DBMS databases.

The results shown in Figure 4 indicate that the parameter ID is injectable to three types of SQLI attacks, Boolean based blind, error based, and time based. The output shows the payload which can be used in the three above techniques.

ii. Phase 2 Results

Phase 2 involves deploying the Modsecurity WAF in front of the DVWA website to carry on the tests again. The output shown in Figure 5 reveals that all the tested parameters do not appear to be injectable, which means that the SQLI vulnerabilities have been remediated and that SQLMAP has failed to exploit the DVWA website.

iii. Phase 3 Results

We modified SQLMAP command to add 3 more parameters shown in Table 3. We added the level parameter with value "5" which means testing all the input parameters and http headers for SQLI. We also added the risk parameter with value "3" to add heavy query time-based SQLI and the OR-based SQLI tests. Finally, we included the tamper parameter which allows including advanced scripts used to evade web application firewall existence. The complete command is shown in Figure 6. Using the above parameters, we ensured that all the advanced SQLI methods were tested. The results seen in Figure 7 indicate that all tested parameters are not injectable and again SQLMAP has failed to attack the web application.

Table 3. SQLMAP Additional Parameters

Parameter	Description
Level	Parameter ranging from 1 to 5 with increasing the value means testing more input parameters and http headers
Risk	Parameter used to test more advanced SQLI techniques.
Tamper	Parameter used to evade WAF existence using advanced scripts

4. CONCULSIONS AND FUTURE WORK

SQLI attacks have caused a devastating impact on SQL databases over the past years. Their ability to update, delete, and give shell access to an attacker makes their effect more dangerous. In this paper, we presented a classified survey of the SQL injection detection and prevention methods. We tested the Modsecurity firewall against SQL injection and it has proved to be efficient to block our attacks based on SQLMAP iterations. It is worth saying that SQLI techniques evolve continuously. To achieve the required protection, it is necessary to regularly update the rules of Modsecurity to be able to block the new techniques of SQL injection attacks that daily evolve. For future work, we plan to test the Modsecurity against the other common web attacks especially the OWASP top 10.

```
root@kali:~# sqlmap -u http://10.0.2.5/dvwa/vulnerabilities/sqli/?id=ss&Submit=Submit -–cookie='security=low; PHPSESSID=13f7160329237b2fd6043790ec9528c3' --dbs
```

Figure 3: Phase One Initial SQLI Testing

```
Parameter: id (GET)
    Type: Boolean-based blind
    Title: OR Boolean-based blind – WHERE or HAVING clause (MYSQL comment) (NOT)
    Payload:  id=ss' OR NOT 1835=1835#&Submit=Submit

    Type: error-based
    Title: MYSQL >= 4.1 OR error-based – WHERE or HAVING clause (FLOOR)
    Payload:  id=ss' OR ROW(2275,9875)>(SELECT COUNT(*),CONCAT(0x71716a7071),(SELECT
(ELT(2275=2275,1))),0x7176706271,FLOOR(RAND(0)*2))x FROM SELECT 6775 UNION SELECT 6350 UNION
SELECT 2900 UNION SELECT 7322)a GROUP BY x)– gSnm@Submit=Submit

   Type: AND/OR time-based blind
   Title: MYSQL >=5.0.12 OR time-based blind
   Payload : id=ss' OR SLEEP(5)– imEr@Submit=Submit
```

Figure 4: Phase One DVWA SQLI Vulnerabilities

```
[02:04:44]  [INFO] testing 'Generic UNION query (NULL) – 1  to  10 columns'

[02:04:44]  [WARNING] GET parameter 'Submit' does not seem to be injectable

[02:04:44]  [CRITICAL]  all tested parameters do not appear to be injectable. Try to increase values for '- -
level'/'- -risk' options if you wish to perform more tests. If you suspect that there is protection mechanism
involved (e.g. WAF) maybe you could try to use option '—tamper' (e.g.  '—tamper=space2comment')

[02:04:44]  [WARNING] HTTP error codes detected during run:

403 (Forbidden)   -   265 times
```

Figure 5: Phase Two Results

```
root@kali:~# sqlmap -u "http://10.0.2.5/dvwa/vulnerabilities/sqli/?id=ss$Submit=Submit –
cookie='PHPSESSID=968dfpjc2pch4tu089vefflrav; security=low' --tamper=
between,bluecoat,charencode,charunicodeencode,concat2concatws,equaltolike,greatest,halfversionedmorekey
words,ifnull2ifisnull,modsecurityversioned,modsecurityzeroversioned,multiplespaces,nonrecursivereplacement,
percentage,randomcase,securesphere,space2comment,space2hash,space2morehash,space2mysqldash,space2pl
us,space2randomblank,unionalltounion,unmagicquotes,versionedkeywords,versionedmorekeywords,xforwaded
for  --level=5 --risk=3 --dbms "MYSQL"
```

Figure 6: Phase Three SQLI Aggressive Testing

```
[03:20:06]  [WARNING] cookie parameter 'security' does not seem to be injectable
[03:20:06]  [CRITICAL] all tested parameters do not appear to be injectable
[03:20:06]  [WARNING] HTTP error codes detected during run:
400 (Bad Request)  - 33215 times, 403 (Forbidden)  - 14237 times, 414 (Request-URI Too Long) – 872 tunes
```

Figure 7: Phase Three Results

REFERENCES

[1] Vemulakonda, Rajesh, and Ketha Venkatesh. "SQLIADP: A Novel Framework to Detect and Prevent SQL Injection Attacks." Smart Intelligent Computing and Applications. Springer, Singapore, 2020. pp. 41-50.

[2] OWASP TOP 10, https://owasp.org/www-project-top-ten/

[3] Z. C. S. S. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks," *2020 IEEE Conference on Computer Applications (ICCA)*, Yangon, Myanmar, 2020, pp. 1-6.

[4] P. Tang, W. Qiu, Z. Huang et al., Detection of SQL injection based on artificial neural network, Knowledge-Based Systems (2020).

[5] I. Medeiros, M. Beatriz, N. Neves and M. Correia, "Demonstrating a Tool for Injection Attack Prevention in MySQL," 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, 2017, pp. 551-558.

[6] Benfano Soewito, Fergyanto E. Gunawan, Hirzi, Frumentius,Prevention Structured Query Language Injection Using Regular Expression and Escape String, Procedia Computer Science,Volume 135,2018,Pages 678-687,ISSN 1877-0509.

[7] Q. Temeiza, M. Temeiza and J. Itmazi, "A novel method for preventing SQL injection using SHA-1 algorithm and syntax awareness," 2017 Joint International Conference on Information and Communication Technologies for Education and Training and International Conference on Computing in Arabic (ICCA-TICET), Khartoum, 2017, pp. 1-4.

[8] K. D'silva, J. Vanajakshi, K. N. Manjunath and S.Prabhu, "An effective method for preventing SQL injection attack and session hijacking," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, 2017,pp. 697-701.

[9] B. Appiah, E. Opoku-Mensah and Z. Qin, "SQL injection attack detection using fingerprints and pattern matching technique," 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2017, pp. 583-587.

[10] Martínez, Rodrigo & Betarte, Gustavo & Pardo, Alvaro. (2018). Web Application Attacks Detection Using Machine Learning Techniques. 10.1109/ICMLA.2018.00174.

[11] S. O. Uwagbole, W. J. Buchanan and L. Fan, "Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention," 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, 2017, pp. 1087- 1090.

[12] D. Appelt, A. Panichella and L. Briand, "Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks,"2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, 2017, pp. 339-350.

[13] S. Jayaprakash and K. Kandasamy, "Database Intrusion Detection System Using Octraplet and Machine Learning," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2018, pp. 1413-1416.

[14] N. Gunaseeli and D. J. Mala, "Client Side Countermeasures for the Prevention of SQLIA in Web Applications," 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Coimbatore, 2017, pp. 1-3.

[15] K. A. Jackson and B. T. Bennett, "Locating SQL Injection Vulnerabilities in Java Byte Code Using Natural Language Techniques," SoutheastCon 2018, St. Petersburg, FL, 2018, pp. 1-5.

[16] A. Ghafarian, "A hybrid method for detection and prevention of SQL injection attacks," 2017 Computing Conference, London, 2017, pp. 833-838.

[17] R. A. Katole, S. S. Sherekar and V. M. Thakare, "Detection of SQL injection attacks by removing the parameter values of SQL query," 2018 2nd International Conference on Inventive Systems and
Control (ICISC), Coimbatore, 2018, pp. 736-741.

[18] Aliero, M.S., Ghani, I., Qureshi, K.N. et al. An algorithm for detecting SQL injection vulnerability using black-box testing. J Ambient Intell Human Comput 11, 249–266 (2020).

[19] Modsecurity, https://www.modsecurity.org/.