DETECT SQLI ATTACKS IN WEB APPS USING NVS

J.Abinaya M.E.,

Department of Computer Science and Engineering, NPR college of Engineering and Technology (Chennai Anna university)

Abstract:

Now-a-days the world of information era, we can get information just our single click by using Web application. Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. It are playing a major role in this, every organization are mapping their business from a room to the world with the help of these Web Application. It consist of a three tier structural design where database is in the third pole, which is the most valuable assets in any organization, as the adaptation of web applications are increases day by day, various attacks are possible increasing day by day. An attack which is directly compromises the database that is most threatening attack is called SQL injection. There are various Vulnerability scanners has been proposed to deal with this attack, but none of them are able to detect SQLI completely. In my tools have the accuracy ratio very less as well as they produce a high rate of false positive, apart from that all these tools take much time to scan. To avoid these problem and detect SQL completely we are presenting a NVS that is Network Based Vulnerability Scanner approach this provides a better coverage and with no false positive with a short span of time.

Keywords:

SQL injection attacks, Vulnerability Scanner, Web application

I.INTRODUCTION

Web applications have introduced a new way of business, where we have the facility to book our railway as well as flight ticket, we can buy any product as well as we can submit our phone bills to, just on a single click without visiting to the concerned office. It saves lots of our time and effort to. A web application is consist of three-tier architecture, where at first tier client submit their request, on second tier application server performs the logic according to the request, and at the third tier database works, which is used for storing the client credentials as well as other information such as companies records. So database is the most valuable assets in any web application, as every organization is moving towards web application and they are accessed without any restriction, various attacks are possible against this SQL injection is the most dangerous attack against any web application because in SQLI an attacker can executes their SQL queries within the server database, so they can easily retrieves the confidential records as well as append and modify [6].

Various techniques are available to deal with the SQLI, like black box testing where various Vulnerability scanners are used to detect the SQLI inside the web application [2, 5], but none of these existing vulnerability scanners provide full coverage and also not provide a good result without false positive, so we are presenting a network based vulnerability scanner which will work more faster than their previous approaches and will also provide a good coverage ratio without false positive.

If we scan the web application before being deployed to public use by using our vulnerability scanner then we can actually find the vulnerabilities inside it and we can fix it, for that we crawl the whole web application and for each page we generate the attack payload, perform the simulation attack and analyze the response, and on behalf of that response we make our report. To show the effectiveness of our tool we have created 5 different web applications in localhost and 4 working web application, results shows that our tool provide better results than previous ones.

This paper is organized as follows. Section II defines background and related work. Section III tells about our proposed model NVS. Section IV shows the implementation details. Section V contains the result and analyses. termination and expectations work has been discussed in section VII.

II.BACKGROUND AND ASSOSIATED JOB

Structure Query Language injection refers to a class of code-injection attacks in which information provided by the user is included in an SQL query in such a way that, that is treated as SQL code. The main form of SQL injection that consists of direct addition of code into user-input variables that are occurred one after another with SQL commands and execute . For example:

We have a url like www.site.com/product.php? product_id=10

If product_id parameter is taking part for forming the SQL queries inside the web application without being sanitized then an attacker can concatenated their special crafted queries with this. Like “www.site.com/product.php?product_id=10; drop table user –“

The following figure shows the execution of SQL commands inside the web application.

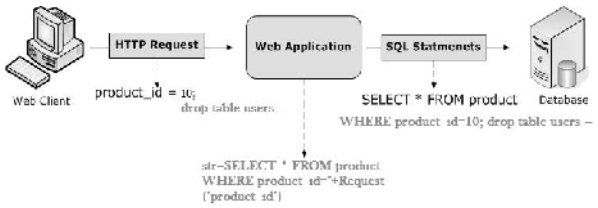

Fig. A. Work Flow of SQLI

Web Application Vulnerability Scanners are designed to penetrate the web applications against the security issues. They are the automated tools designed in such a way that they will perform the same attack as we do manually, the advantage of using Scanners is that they generate the

automated report which shows what are the input points which are vulnerable.
In our literature survey we have gone through various commercial vulnerability scanners, which are being used widely nowadays, for checking the effectiveness of these tools we have created 5 web applications in localhost and check with these tools. These tools provide large amount of time according to the size of the web application.

The tools we surveyed are as follows:

Acunetix: Acunetix Web Vulnerability Scanner is a proprietary web application vulnerability scanning program. Acunetix checks for various vulnerabilities in the application and generates report according to the findings. It also comes with few semi automated and manual tools for testing the web applications further. Acunetix WVS mechanically checks web apps for vulnerabilities such as SQL Injections, cross site scripting, and also file create/delete and weak password strength on authentication pages.

Netsparker: Netsparker Community Edition helps not only Penetration testers but also web developers who can scan their web application instantly and find the vulnerabilities for free. But this edition scans only for XSS, SQL injection, Boolean SQL injection, backup files and static tests. So you can mainly consider this as an XSS and SQLi scanner. The user interface will definitely impress you. Netsparker scans all kinds of web applications without limiting itself to any platforms or technologies. It also find and report security issues such as SQL Injection and Cross-site Scripting in all web apps regardless of the platform and the technology they are built on.

WebCruiser: this is an effective and powerful web penetration testing tool [9]; it has a Vulnerability Scanner and a series of security tools. It can support scanning website as well as POC (Proof of concept) for web vulnerabilities: SQL Injection, Cross Site Scripting, XPath Injection etc. So, WebCruiser is also an automatic SQL injection tool, an XPath injection tool, and a Cross Site Scripting tool!

In order to test the performance of these scanners in a realistic environment we studied web applications available on the Internet and tried to re-create these applications
locally. These sanners are ran in some popular web apps in online.

- By analyzing SQL queries and discover vulnerable spots. These tools need to evaluate the SQL queries for all possible HTTP request hence the overheads are very high.

- This tood try to discover SQL injection vulnerabilities by applying malicious code into the spot. For example, Acunetix took 4 hours (approx) to scan a web application that had 100 pages

- Thus we conclude an attack as successful if the application returns a response different from the previously known SQL Injection has actually failed.And hence the false positive rate of the scanners is very high.

The clarification from our study motivated us to design a light-weight, fast scanner with low false positive rate. The method of our proposed scanner is given in the following section.

III.NVS

In this section we are presenting a Network based Vulnerability Scanner (NVS) model to deal with SQLI. The following figure shows the architecture of our vulnerability scanner.

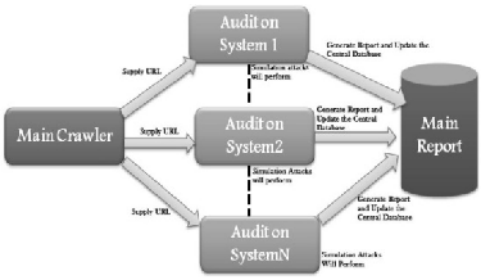

Fig.B. Network Based Vulnerability Scanner

All the vulnerability scanners as we surveyed are based on the standalone system, so they took very high time as well as these also affect the accuracy, because their attack rule library is not very much efficient due to this get false positive the network approach, its efficiency will depend on how many system we are using and also how many connections MySQL supports, because for storing our data we are using MySQL in the backend In our approach we moved one step forward. We are checking what are the SQL commands actually an attacker can run in the background and as well as the privileges of different commands against different tables and users, so at the same time we are providing a report which is effective or and also we discover problems which comes in coding and as well as database privileges problem.

This model consists of three main parts: Crawler (Scanning), Attack Simulation and the Network Setup.

A. Scanning the entire web Apps

Scan the entire web apps the fundamental structure will be in the form of a tree. Like as below figure: In the figure 3, a.php is represented as the home page of the web application and all the child nodes b.php, c.php, etc are the respective pages of the web application. The scanning will perform in the following way.

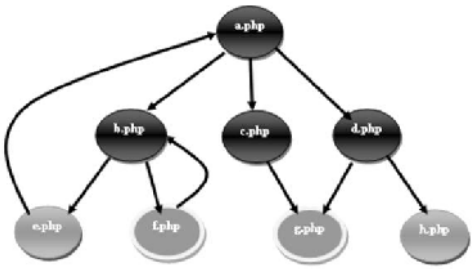

Fig.C. Tree Structure of a Web application

Step1: FIFO queue is created with two fields URL that is primary key, STATUS.
Step2: Include the target URL and set STATUS=0.
Step3: Update STATUS=1

Send Request for the URL

Analyze the response, Extract its entire links. Insert these links in FIFO Queue and set STATUS=0.

Step4: While STATUS=0 ELSE Go To Step5.
Step5: Finish.

B. Attack Simulation

Attack Simulation consist of three parts: (1) Payload Setup (2) Generating Attack and Response Analysis and (3) Report Generation.

1) Payload Setup: In this phase the attack payload will be created based on the previous existed SQLI attacks, for generating the payload we have created a list in which we have grouped all the common SQLI which an attacker used to reveal the database, for each type of database we have specially modified the attack.

2) *Generating Attack and Response Analysis:* In this phase we generate the attack by concatenating the attack payload with the original query URL of the web application, and make request of this specially crafted attack URL. Afterward analyze the response and find out the patterns of SQLI inside the responses.

3) Report Generation: On behalf of the response, if any SQLI pattern found inside this, then the corresponding URL will be added in the report. At the completion of the attack simulation and scanning phase we have automated report, which consist of all the pages list which are vulnerable.

C. Network Setup

For connecting different systems within a network we can create an ad-hoc network and connect each system with this. On each system we have created a RMI server which will sense the request coming from the crawler system. By using the RMI (Remote Method Invocation) we will send different urls to the different attack system, on behalf of these urls each system will put attack and check for the vulnerability, if any vulnerability is found then it will update its report.

IV.IMPLEMENATION INFORMATION

In this phase we have shown the practical approach of our implementation, for implementing to this approach we have used JAVA as a programming language and MYSQL as database for storing the data. This consists of three things, Scanner Implementation, Attack Implementation and Network Implementation.

A. Scanner Implementation

For scanning the whole web application we have created two functions. First one is SeedUrl() function which provides the URLs to the second one i.e. Crawling() function which have the STATUS= 0, Crawling() function takes the value from SeedUrl() and send request for it, after that it writes the response in file and extract all its links and insert it into the database and set STATUS=0. The overall process will repeat till any URL has STATUS=0. The JAVA code has been given below.

```
//SeedUrl() Starts here

Void SeedUrl()

    {

    Boolea ag=true;

    Void SeedUrl()

    {

    Boolea flag=true;

ResultSet rs=stmt.executeQuery("select distinct status,url from `"+TableName+"`where
status=0");

While(rs.next)

{

String str=rs.getString("url");

//get the url from the database
Crawling(str);

//send str to the main crawler
Flag=false;g

If (ag==false)
SeedUrl() Else

System.out.print("Crawling Complete");

}

//SeedUrl() ends here //Crawling() Starts here
```

```
Crawling(String str)

{
Update status=1 where url=str; URL u = new URL(str);

URLConnection uc = u.openConnection();

//Request send to the str to the apps server
FileWriter fw=new FileWriter("URL.txt",false);

//Create a file

while ((ct = r.read( )) != -1) fw.write((char) ct);

//Response is write into file
String regex="href";

Pattern pattern = Pattern.compile(regex,Pattern.CASE
INSENSITIVE|Pattern.MULTILINE);

//All the href attributes finding and extraction qry="insert into url values(0,"+"'"+href
url+"'"+")"; stmt.executeUpdate(qry);

}
```

Attack Implementation

In this phase we have surveyed different SQLI and perform the simulation attack on web application. For that we have divided this into different units.

1) Structure of Attack Database: In this phase we have defined various types of SQLI attacks [3] , and on behalf of that we have set up our attack library, i.e., for each type of database (which is in the back end of the web application) we have defined and grouped all types of possible SQLI against it and put it into a column and named it like MYSQL, similarly grouped different types of SQLI against oracle, and etc. For testing the blind SQLI we also group all type of SQLI in a column named blind_sqli. So the structure of the attack database will be like following:

MYSQL	ORACLE	SQL SERVER
SELECT @@version	SELECT version FROM v$instance;	SELECT @@version
SELECT user();	SELECT user FROM dual	SELECT user_name();
SELECT host, user, password FROM mysql.user; --	SELECT name, password, astatus FROM sys.user$ --	SELECT name, password FROM master..syslogins --

Fig D. Structure of Attack Database

2) Banner grabbing: For grabbing the banner of the back end, means which database is running, we concatenate on the bad character with the original url and make a modified attack url, showing in the figure 5. Then make request for this attack url and analyze the response. If we will not get any database error inside the response we will remove all the different parameters inside the url, then place bad character and make request. The steps are as follows:

Step1: define bad characters like{‘,/,/*,’ or /* or ’), ’ or /* or ')} etc
Step2: String attck_url=url+badcharcater; **Step3:** make request for attack_url;
Step4: analyze the response

if any database error found in the response. Find the banner

Then

setup the attack library according banner go to step 6:

Step4: remove the parameters of the original url
Step5: go to step 2;
Step6: exit;

Fig.E . Banner Grabbing

When we execute this code, suppose we get the url like-http://www.inside-india.com/level1.php?cate_id=1 and our modified url will be like http://www.inside-india.com/level1.php?cate_id=1’ So if we place a request for this we will get an error like- “You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "1'" at line 1”.

3) Setting up attack payload: In this phase we draw different attack from the concerned attack

database column and concatenate with the original url and then we send request
Step1: lib=banner name//define attack library
Step2: ResultSet rs=stmt.executeQuery("SELECT * FROM attack");
Step3: while(rs.next)

{

String attack_url=url+rs.getString(lib);

URL u = new URL(attack_url); //sending request URLConnection uc =
u.openConnection( ); //open conncetion

FileWriter fw=new FileWriter("Attack.txt",false);//define attack file

while ((ct = r.read( )) != -1)
fw.write((char) ct);//write response in file

String[] regex={"Microsoft SQL Server ","5.0.27-community-nt"}; //define pattern for
each attack Pattern pattern = Pattern.compile(regex,Pattern.CASE_INSENSITIVE|
Pattern.MULTILINE);

Matcher m = pattern.matcher(str);//finding pattern in response

while (m.find())

{

flag=true;

error=regex;

}

If(flag==true)

Update the central database with these values(url,error,vulnerability)

} //while ends

Step4: finish

C. Network Setup

In a network, Server accepts tasks from clients, runs the tasks, and returns any results. The server code consists of an interface and a class. The interface defines the methods that can be invoked from the client. Essentially, the interface defines the client's view of the remote object. The class provides the implementation. Client needs to call the Server, but it also has to define the task to

be performed by the server.

//Interface Setup

//Setting up the interface between client and server public interface Calculate extends Remote

{

public String GetUrl(String url,String table) throws RemoteException;

public String CreateTable(String tab) throws RemoteException;

}

//Server Setup
//Listen the request

CalculateEngine ce = new CalculateEngine( ); Naming.rebind("Calc", ce); System.out.println("Attack Engine ready.");

//Client Bindup

//Sending request from client

Object o = Naming.lookup("rmi://10.0.33.35/Calc"); Calculate c = (Calculate) o; c.GetUrl(str,TableName);

V.RESULTS AND COMPARISON INFORMATION

Here we are designed five different types of web appls in the local host for effectiveness of penetration test.The existing tool was tested online in real web application.Names of the web apps given below.

Tested for vulnerability and the SQLI vulnerabilities present in each of them is shown in table I. The sum total of vulnerabilities present in all the web applications is twenty one

TABLE I: VULNERABILITY DETAILS OF DIFFERENT WEB APPS

Web Apps	Field	Susceptible
Travel	LocalHost	No
On line Real State	LocalHost	SQLI (1)
ICC World Cup11	LocalHost	SQLI (1)
On line Tutorial	LocalHost	SQLI (1)
Graphics	LocalHost	SQLI (1)
Travel	Public	SQLI(1)
Job Site	Public	SQLI(4)
Education	Public	SQLI(12)
Total susceptibility		21

In order to quantify the performance of our tool three other well known SQLI scanners were also tested on these web applications. Out of the total 21 the number of vulnerabilities each tool could detect, the average time taken and number of false positives is shown in table II.

TABLE II: COMPARATIVE STUDY OF SQLI SCANNERS

Parameter	Acunetix	Netsparker	WebCruiser	NVS
Vulnerability Detected	18/21	16/21	15/21	21/21
Average Time(hr)	2.24	1.2	0.15	0.01
False Positive	12	3	1	0

The result shows that none of the tools provides full coverage ratio and they also take much time to generate the report. Figure 6 shows the results for the execution of penetration testing tools and for the NVS tool. As we can see, different tools have reported different numbers of vulnerabilities. An important observation is that the NVS is able to identify a much higher number of vulnerabilities than the remaining tools. In fact, all the penetration-testing tools detected less than 85% of the vulnerabilities, while our tool detected all vulnerabilities. Considering only the penetration testing tools, Acunetix identified the highest number of vulnerabilities (85% of the total vulnerabilities). However, it was also the scanner with the highest number of false positives (it detected 12 vulnerabilities that, in fact, do not exist). The lowest number of vulnerabilities was detected by WebCruiser with false positive is only 1. As different tools detect different sets of vulnerabilities an interesting analysis is how these sets intersect each other.

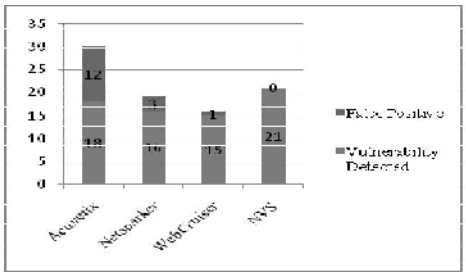

Fig. F. Various scanner graph

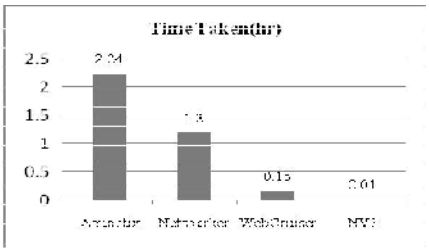

Fig. H. Time Taken by each Scanner

In Figure G, each and every circle represent the vulnerabilities detected by a tool and each intersection area represents vulnerabilities found by more than one tool. The circle area is roughly proportional to the represented number, but the same does not happen with the intersection areas, as it would be impossible to represent it graphically. As we can see, there are 21 vulnerabilities that are detected only by NVS. We observe that Acunetix misses only 3. The Netsparker had a lower reporting than Acunetux it could detect 1 of the 3 vulnerabilities that was miss by the latter. WebCruiser could detect 15 vulnerabilities and all of these were detected by the other testing tools.

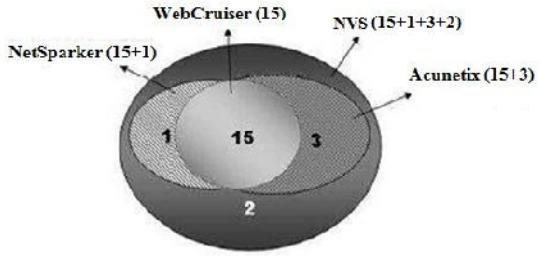

Fig. G. Vulnerability Covered by each tool

Time taken by each tool is also very high for scanning a large web application. Acunetix average time of making a report (overall scanning and auditing) is almost 2.30 hours, means we have to run the system and wait for 2.30 hours, other scanners also take much time as well, the time graph of each vulnerability scanner has shown below.

VI. Advantage of using NVS

There are many recent tools for network scanner like Acunetix WVS, Vega,NTO Spider,App scan,Net sparker and etc., these tools are having advantages and also disadvantages. Network Based Vulerability Scanner provide full coverage and also provide a good result without false positive. And also check automatically and work more faster then previous it Only take few seconds to scan. The NVS provide a good coverage ratio without false positive.

VII.CONCLUSIONS

This paper proposes a Network Based Vulnerability scanner (NVS), which is able to detect all the pages in a web application which are vulnerable to SQLI, on behalf of the simulation attack, this tool makes a report which helps programmers to work and fix only the vulnerable pages, so this approach helps programmer to focus only the bad pages rather than the whole web application, at the same time NVS provides no false positive, provides up to maximal of coverage, and also the completeness. The greatest advantage of NVS is it generates the report within the average time .01 hour. Its efficiency is basically dependent upon the number of system connected within the network.

REFERENCES

[1] Justin Clarke "SQL Injection Attacks and Defense", ISB 9781597494243, published may 2009, Syngress Publishing, Inc.Elsevier, Inc.

[2] Herbert Schildt," Java 2 The Complete Reference fifth edition", published by McGraw-HIll Companies, Inc., ISBN: 0-07-213084-9

[3] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures", proceeding of 2006 14th ACM SIGSOFT international symposium on Foundations of software engineering

[4] Netsparker of Mavituna Security Ltd., http://www.mavitunasecurity.com/netsparker/ , visit on January 2011.

Author

I like to share a little moment in my life. I am Abinaya. I Was born in Madurai ,Tamil Nadu. My father was an Business Men. I am a simple Middle class and an independent girl. I like to stand in my own legs .About my Education I did my schooling in Holy family Hr sec school. When I was in 10th std I passed type writing in both Higher and lower grade. And I did my Bachelor degree (B.E.,) in Bharath Niketan Engineering College (Anna University Chennai) 2009-2013. During my academic period I attended many National and International level conferences and presented papers in the area of Cloud Computing. My area of interest is cloud computing .During my final year I did my project named "A Network Based Vulnerability Scanner for Detecting SQLI Attacks in Web Application". I like teaching so I decided to do M.E., degree, for that I passed My Entrance Exam And now I pursuing M.E., (Computer Science And Engineering) in NPR College of Engineering and Technology (Anna University Chennai). After completion of my Master degree I like to start my profession as an Assistant professor in only one of the top Engineering College.