B.M.S COLLEGE OF ENGINEERING, BENGALURU
Autonomous Institute, Affiliated to VTU

A Technical Seminar Report
on

An in-depth practical examination of SQL Injection attacks and Prevention techniques

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:
Kizhakel Sharat Prasad
1BM19CS074

Work carried out at

Internal Guide

Dr. Rajeshwari B S
Assistant Professor

Department of Computer Science and Engineering
B.M.S College of Engineering
Bangalore-560019

Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2020-2021

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Kizhakel Sharat Prasad (1BM19CS074) student of 4th Semester, B.E, Department of Computer Science and Engineering, B.M.S College of Engineering, Bangalore, hereby declare that, the technical seminar entitled "An in-depth practical examination of SQL Injection attacks and Prevention techniques" has been carried out under the guidance of Dr. Rajeshwari B S, Assistant Professor, Department of CSE, B.M.S College of Engineering, Bangalore during the academic semester April - July 2021. I also declare that to the best of our knowledge and belief, the technical seminar report is not from part of any other report by any other students.

Signature of the Candidate

Kizhakel Sharat Prasad(1BM19CS074)

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Technical Seminar titled "An in-depth practical examination of SQL Injection attacks and Prevention techniques" has been carried out by Kizhakel Sharat Prasad(1BM19CS074) during the academic year 2020-2021.

Signature of the guide
Dr. Rajeshwari B S
Assistant Professor
Department of Computer Science and Engineering
B.M.S College of Engineering, Bangalore

Abstract

Preserving potentially sensitive data from any kind of attacks is a necessity in any web applications. SQL injections exploit poorly designed web applications by ’injecting’ a malicious query via the client to the application when there is non-validated input. This can result in unauthorized access to sensitive data. Ever since its inception two decades before, OWASP still lists injections in the top 10 application vulnerabilities list as of 2021. SQL injections are still an active threat particularly for PHP applications. In this paper I have provided an examination of the underlying principle of an SQL Injection attack, their various types and have concluded with a note of several exisiting defense techniques that can be used to prevent the same. The last section has taken into account some novel approaches for SQLIA defense, from exisiting papers on the topic.

Chapter 1: Introduction

1.1 Overview

There are many types of SQL injection, however majority involve an attacker inserting arbitrary SQL into a web application database query. The simplest form of SQL injection is through user input. Web applications typically accept user input through a form, and the front end passes the user input to the back-end database for processing. If the web application fails to sanitize user input, then the attacker can inject SQL query of their choice into the back-end database and cause loss in confidentiality,integrity and authority [1] of the database. Another method of SQLIA is modifying cookies to poison a web application's database query. Cookies store client data locally, and web applications commonly load cookies and process that information. A malicious user can be used to modify cookies to inject SQL into the database.Server variables such as HTTP headers are also used as a SQL injection attack vector. The cause of SQL attack is mostly: no proper validation for user input. To find a solution for this problem, various coding guidelines have been proposed such as validation and encoding of the user input. These techniques are implemented by humans, so it is also prone to error [2]. The basic nature of an SQLIA has been shown in the code snippet below [3]:

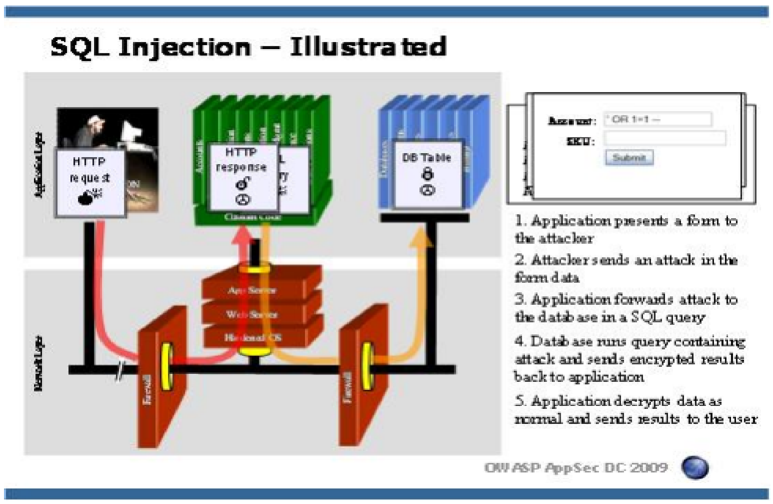
Listing 1: SQLIA classic bypass

```
//Database connection
con = mysql connect(  localhost  ",  uname  ",  pass  ");
//Dynamically generating SQL query with user input Q=
     SELECT   * FROM item WHERE cost <   $
GET[  val  "]     "."ORDER BY itemDescription";
//Executing the query against the database
ResultSet rs = con.executeQuery($Q);
The URL   http  ://www.altoromutual.com/item.php?val=1000

In this case the data is in the body of the user request
    i.e \$dollar
When user provides malicious input 1000    OR     1   =
     1
and the corresponding URL http://www.altoromutual.com/
   item
.php?val=    1000    OR     1   =   1   , the
   dynamically generated
```

```
SQL query is actually  SELECT  * FROM item WHERE price
   <    1000
OR     1   =   1    ORDER BY itemDescription // this
   query will return all details of the given database
   since 1=1 is a universal truth irrespective of the
   other condition. This is called tautology-based SQLIA
   .Depending on
the different intention, user could provide even more
   malicious input like:
   ; DROP TABLE item -- and the dynamically
   constructedstructed SQL query  SELECT  * FROM item
   WHERE
price <         ; DROP TABLE item -- ORDER BY
   itemDescription //deletes the   item   table
   from the database since the input string has been
   closed at the beginning itself and the text after it
   is ignored by '--'.
```

However, the main cause of this attack is the effect of direct involvement of code into parameters which are concatenated with SQL statements and will execute. Therefore,an effective and efficient knowledge and techniques to detect and prevent SQLIA are needed [figure 1].



1. Application presents a form to the attacker
2. Attacker sends an attack in the form data
3. Application forwards attack to the database in a SQL query
4. Database runs query containing attack and sends encrypted results back to application
5. Application decrypts data as normal and sends results to the user

Figure 1: SQL Injection Attack

1.2 **Motivation**

SQL injection is a subset of an even larger exploit known as an injection, which also includes application code, web components, networking hardware, and the other various components that make up the framework of an application. This threat is the most frequent and consistently rated top security exploit in the history of database software. Despite years of research, identification, and attention, SQL injection persists and continues to plague organizations via unprotected endpoints. With advancement in technology, modern society has accomplished many unthinkable goals. However, as technology develops, so does the risk involved in using it. Same is the case with web applications. Today's applications are plagued with vulnerabilities. Since 2003, SQL Injection has remained in the OWASP Top ten list of application security risks that companies are struggling to deal with. My main aim by conducting research on this topic is because it is a very much real cybersecurity issue with not many preventive mechanisms and I could directly do practical testing on the vulnerabilities as it is related to my Database Management Systems course for 4th sem.

1.3 **Objective**

The main objective of this seminar is to bring to the readers attention the basic mechanism of the SQL Injection Attack, its different types which have been explained along with their corresponding implementations. The second part deals with a discussion on semi-automated SQL Injection tools and most importantly, preventive techniques against SQL Injection. The preventive techniques have been taken from recent papers on the topic. The security issues exposed by the websites are increasing, and the situation is not looking optimistic. Today, a large number of Web systems use a database to store various data of a website, which may be the user's personal information, or may be a company's confidential information. If that information is leaked, it is a huge loss and risk to the individual or the company. SQL injection attacks can achieve the purpose of obtaining illegal data, so it is conceivable that the harm of SQL injection is huge. From the point of view of SQL injection, SQL injection attacks are still one of the most common and most dangerous attacks. This paper introduces the concept and mechanism of SQL injection attack, introduces the type of SQL injection, analyzes the basic implementation process of SQL injection attack, and finally gives the defense mechanism of preventing SQL injection attack.

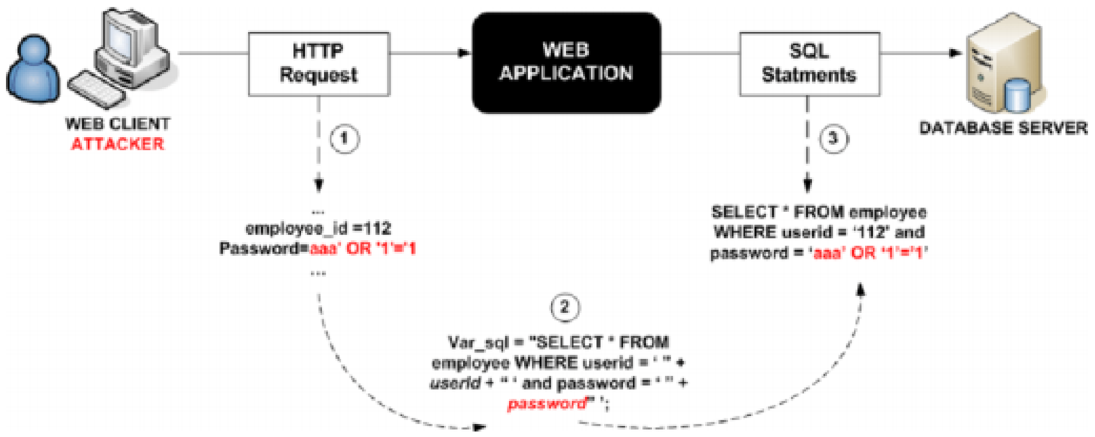

WEB CLIENT
ATTACKER
HTTP Request
WEB APPLICATION
SQL Statments
DATABASE SERVER
1
...
employee_id =112
Password=aaa' OR '1'='1
...
3
SELECT * FROM employee WHERE userid = '112' and password = 'aaa' OR '1'='1'
2
Var_sql = "SELECT * FROM employee WHERE userid = ' " + userid + " ' and password = ' " + password" ';

Figure 2: SQL Injection Attack mechanism

Chapter 2: LITERATURE SURVEY

The use of technology has become a new standard for us especially when it comes to safety and security. The database serves as the data connection of other components which contains various data. This information were processed to create valuable knowledge, sent and stored in using the Internet is vulnerable. These inherent vulnerabilities can be used by the attackers to take data from their target. Implementation of the appropriate security measures on all the available assets should be taken into consideration. Thus, to ensure that stored, shared or transferred information are secured against possible attacks. Security assurance in any communication lines is a necessity which has been an important part in the life of its user. Payment, mobile banking, important reminders such as stocks and news alerts, traffic updates, weather information and business related information are being catered through online systems. Widespread adoption of the internet has resulted in rapid advancement in information technologies. The internet is used by almost anyone for the information, in a way that allows the information owners quick access while blocking break-in attempts from unauthorized users. Over last several years, SQLI has been the most prevalent form of attack on web databases and this attack used to extract the content of databases without given permission. SQLI is a type of utilization caused by processing invalid user inputs. The concept of injection attacks is to inject (or insert) malicious code into a program so as to change structure of SQL query. Such an attack may be performed by adding strings of malicious characters into data values. Injection attacks generally take advantages of improper validation over input/output data. SQLI Attack or SQLIA is a type of code injection attacks which consist of injection of malicious SQL commands by means of input data from the client sent as an HTTP request to the application that are later passed to the instance of the database for execution and aim to affect the execution of predefined SQL commands. They mainly occur when malicious input is passed into a database without being properly validated or encoded. Tautology and Comment-line attack are the most common SQLIA in this type of attack the client is attacking the web server database. The input that the attacker passes into the interpreter is crafted to be eligibly the attacker. These attacks can expose all sensitive user and business related data, but it could even go as far as executing operating system command or giving an attacker complete control of a web application. When tautologies don't work blind SQLI are performed in order to see the difference between response times. They work on getting "inference" by testing based on true or false statements or by putting alternate sleep times. Similarly coming to prevention techniques right now the ones that are known to be effective are parameterized queries and stored procedures. Parametric queries are currently the most effective defense against SQL injection attacks. The main idea of a parameterized query statement is basically: First send the default SQL statement of the website to the database server for pre-compilation which will generate a template, and then send the template back to the web server. Stored Procedures require the developer to just build SQL statements with parameters which are automatically parameterized unless the developer does something largely out of the norm.

Chapter 3: METHODOLOGY/TECHNIQUES USED

The technique that has been used in performing SQL Injection Attacks practically using the burp suite tool are as follows:

- Tautologies:SQL injection queries are injected into one or more conditional statements so that they always evaluate to true.
- Illegal/Logically Incorrect Queries:Using error messages rejected by the database to find useful data facilitating injection of the backend database. It comes under in-band SQLi.
- Union Query:Injected query is joined with a safe query using the keyword UNION in order to get information related to other tables from the application. It comes under in-band SQLi.
- Piggy-Backed Queries:Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes.
- Stored Procedures:Additional malicious queries are inserted into an original injected query.
- Inferential:In inferential SQLi attacker derives logical conclusions from the answer to a true/false question concerning the database.

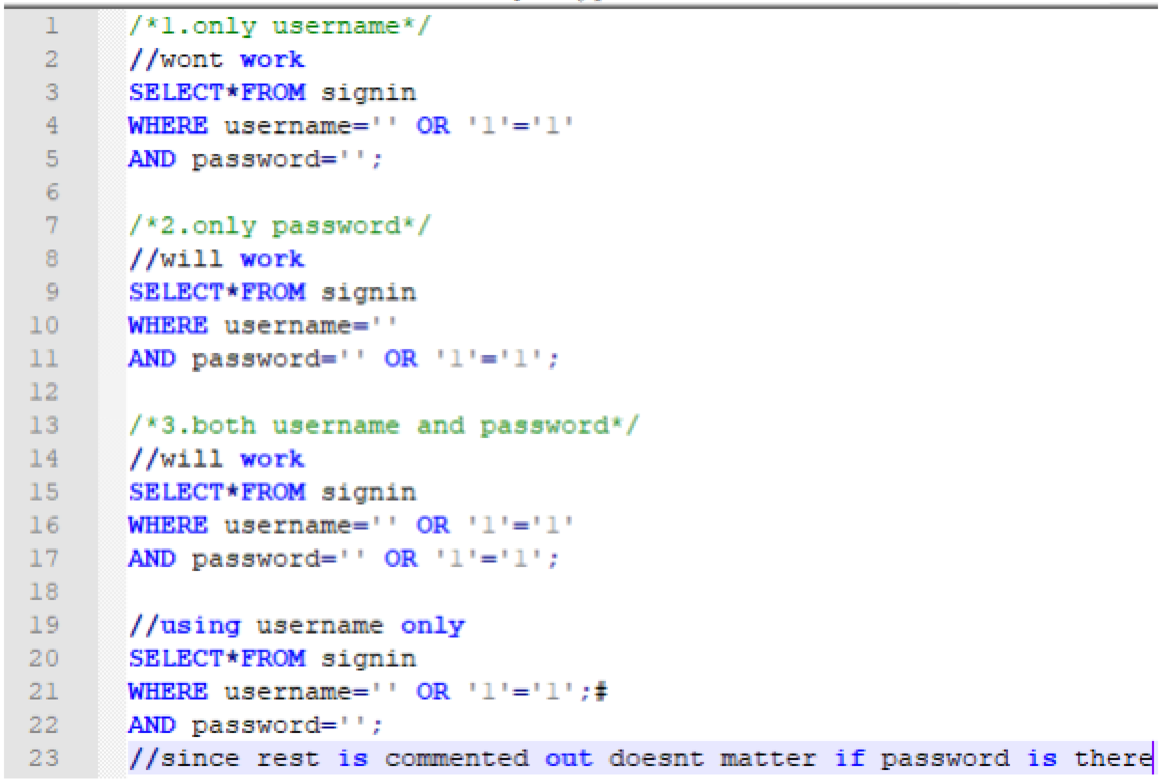
  i.Blind Injection: Information is collected by inferring from the replies of the page after questioning the server true/false questions.

  ii.Timing Attacks: An attacker collects information by observing the response time (behavior) of the database which is set differently based on true or false.

- Alternate Encodings:It aims to avoid being identified by secure defensive coding and automated prevention mechanisms. Hence, it helps the attackers to evade detection. It is usually combined with other attack techniques.

Depending on whether the attacker and response are coming on the same channel the appropriate method needs to be used.

classic SQLi bypass

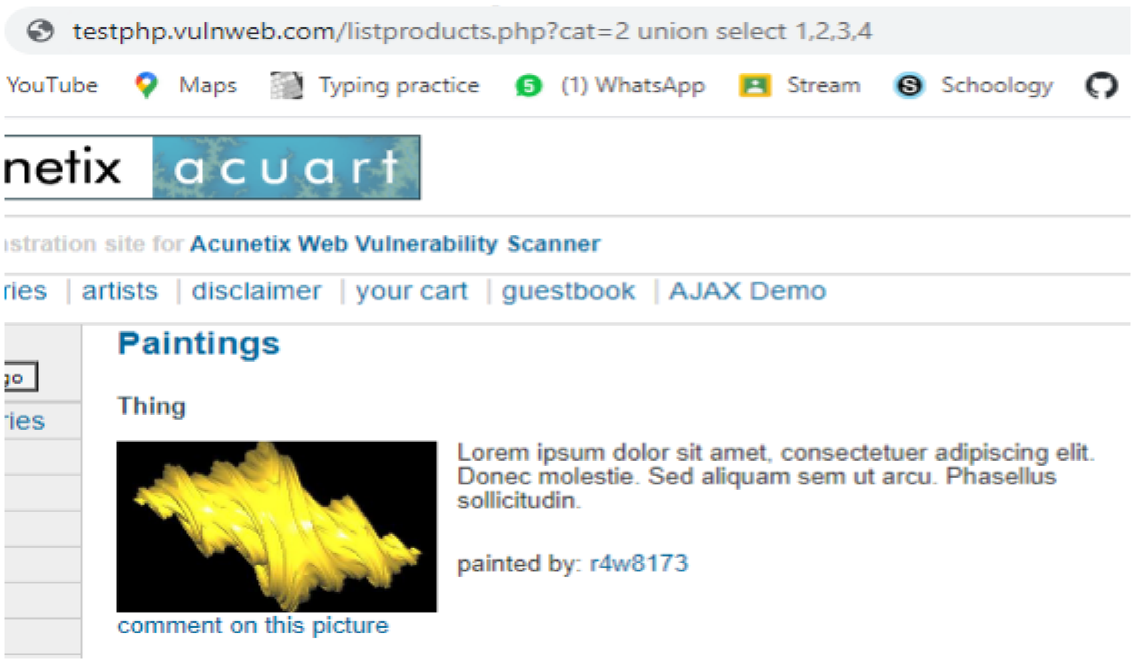


```
/*1.only username*/
//wont work
SELECT*FROM signin
WHERE username='' OR '1'='1'
AND password='';

/*2.only password*/
//will work
SELECT*FROM signin
WHERE username=''
AND password='' OR '1'='1';

/*3.both username and password*/
//will work
SELECT*FROM signin
WHERE username='' OR '1'='1'
AND password='' OR '1'='1';

//using username only
SELECT*FROM signin
WHERE username='' OR '1'='1';#
AND password='';
//since rest is commented out doesnt matter if password is there
```



```
testphp.vulnweb.com/listproducts.php?cat=2 union select 1,2,3,4
```

YouTube
Maps
Typing practice
(1) WhatsApp
Stream
Schoology
netix
acuart
istration site for Acunetix Web Vulnerability Scanner
ries | artists | disclaimer | your cart | guestbook | AJAX Demo
go
ies

Paintings

Thing

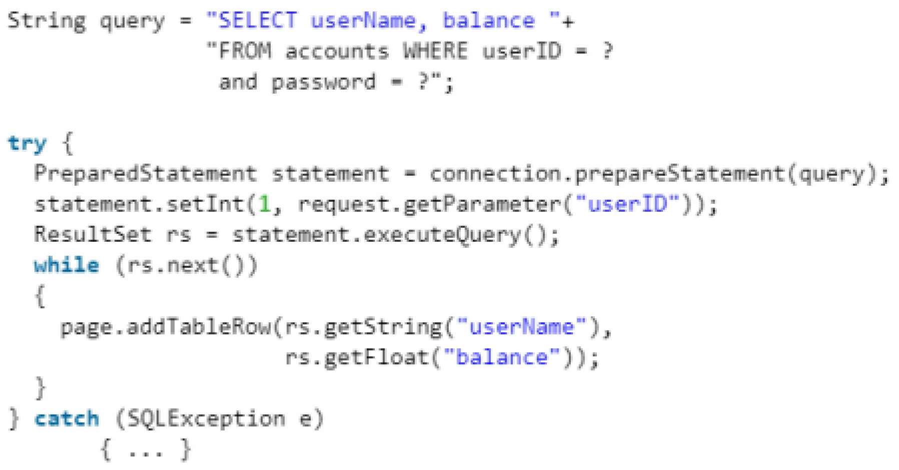
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

painted by: r4w8173

comment on this picture

Union based SQLi

While taking into consideration the defense techniques the parameterized queries approach was use to prevent SQLIA. The attack occurs because the characters entered by the user are being executed as SQL instructions but to prevent them from being executed as SQL instructions, we can use parameterized query statements. Parametric queries are currently the most effective defense against SQL injection attacks. The main idea of a parameterized query statement is basically: First send the default SQL statement of the website to the database server for pre-compilation which will generate a template, and then send the template back to the web server. After that, the user's input can only represent one parameter string, and can't be constructed as a new SQL statement. This eliminates the user-entry characters being executed as SQL statements, thus directly eliminating SQL injection.



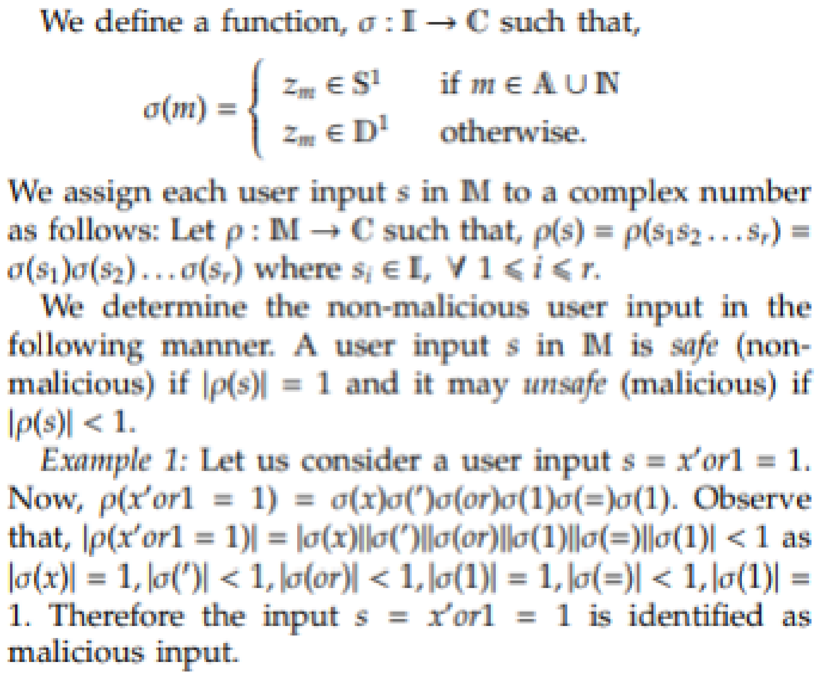
```
String query = "SELECT userName, balance "+
               "FROM accounts WHERE userID = ?
                and password = ?";

try {
  PreparedStatement statement = connection.prepareStatement(query);
  statement.setInt(1, request.getParameter("userID"));
  ResultSet rs = statement.executeQuery();
  while (rs.next())
  {
    page.addTableRow(rs.getString("userName"),
                     rs.getFloat("balance"));
  }
} catch (SQLException e)
     { ... }
```

creating a parameterized query object

The next approach was a novel approach based on complex numbers. It is as follows:

Let K, A, S and N denote set of keywords, set of alphabets in the upper case and lower case, set of special characters and set of numbers form 0 to 9 respectively. Let M be the set of strings in the domain $I = K \cup A \cup S \cup N$. In particular, M is the set of all possible user inputs for the web applications on I. Let S
1 = $\{z \in C \mid |z| = 1\}$
and D1 = $\{z \in C \mid |z| < 1\}$.



We define a function, $\sigma : \mathbb{I} \rightarrow \mathbb{C}$ such that,

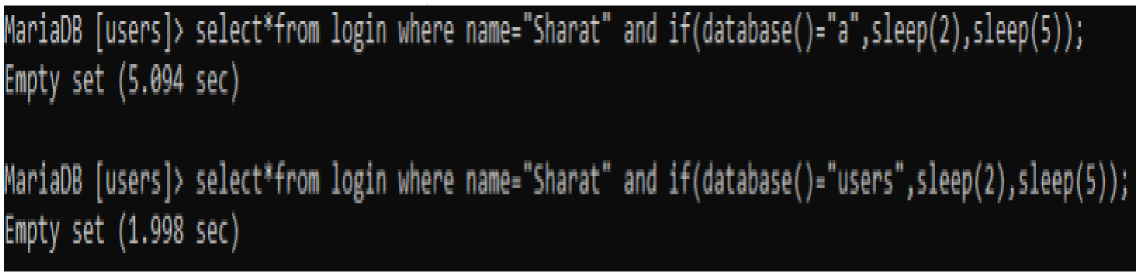
$$\sigma(m) = \begin{cases} z_m \in \mathbb{S}^1 & \text{if } m \in \mathbb{A} \cup \mathbb{N} \\ z_m \in \mathbb{D}^1 & \text{otherwise.} \end{cases}$$

We assign each user input s in $\mathbb{M}$ to a complex number as follows: Let $\rho : \mathbb{M} \rightarrow \mathbb{C}$ such that, $\rho(s) = \rho(s_1 s_2 \ldots s_r) = \sigma(s_1)\sigma(s_2)\ldots\sigma(s_r)$ where $s_i \in \mathbb{I}$, $\forall\ 1 \leqslant i \leqslant r$.

We determine the non-malicious user input in the following manner. A user input s in $\mathbb{M}$ is *safe* (non-malicious) if $|\rho(s)| = 1$ and it may *unsafe* (malicious) if $|\rho(s)| < 1$.

Example 1: Let us consider a user input $s = x'or1 = 1$. Now, $\rho(x'or1 = 1) = \sigma(x)\sigma(')\sigma(or)\sigma(1)\sigma(=)\sigma(1)$. Observe that, $|\rho(x'or1 = 1)| = |\sigma(x)||\sigma(')||\sigma(or)||\sigma(1)||\sigma(=)||\sigma(1)| < 1$ as $|\sigma(x)| = 1, |\sigma(')| < 1, |\sigma(or)| < 1, |\sigma(1)| = 1, |\sigma(=)| < 1, |\sigma(1)| = 1$. Therefore the input $s = x'or1 = 1$ is identified as malicious input.

As seen in the picture above the sigma function decides if a given input is safe or not by breaking it down character by character and then applying the code-based analysis algorithm to classify it as malicious if the function output is<1 and the function output is =1 then its classified as safe. In the above example since there are two unsafe characters the malicious input query is classified as unsafe before hand itself. For executing SQLIA Burp suite tool was used which has an intercept tab that can intercept an incoming or outgoing HTTP request and forwards the wrong/correct request to the repeater then the custom payload is provided and the attack is started afresh. Sometimes the union based approach works well when we get the data back on the same channel. But sometimes when there is no way to use union based approach to get data from the database a different approach called inferential approach is used which is based on true or false logic and information is extracted from database by observing different response from the web page based on some condition.



```
MariaDB [users]> select*from login where name="Sharat" and if(database()="a",sleep(2),sleep(5));
Empty set (5.094 sec)

MariaDB [users]> select*from login where name="Sharat" and if(database()="users",sleep(2),sleep(5));
Empty set (1.998 sec)
```

Chapter 4: DESCRIPTION OF TOOL SELECTED

Languages used:SQL,PHP

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDBMS) like MySQL, MS Access, Oracle, Sybase, Postgres and SQL Server use SQL as their standard database language.

PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side. PHP is well suited for web development. PHP stands for Hypertext Preprocessor. It is an interpreted language, i.e., there is no need for compilation.

Semi automated tool: Burp Suite

Burp Suite is a Java based Web Penetration Testing framework. It has become an industry standard suite of tools used by information security professionals. Burp Suite helps you identify vulnerabilities and verify attack vectors that are affecting web applications.

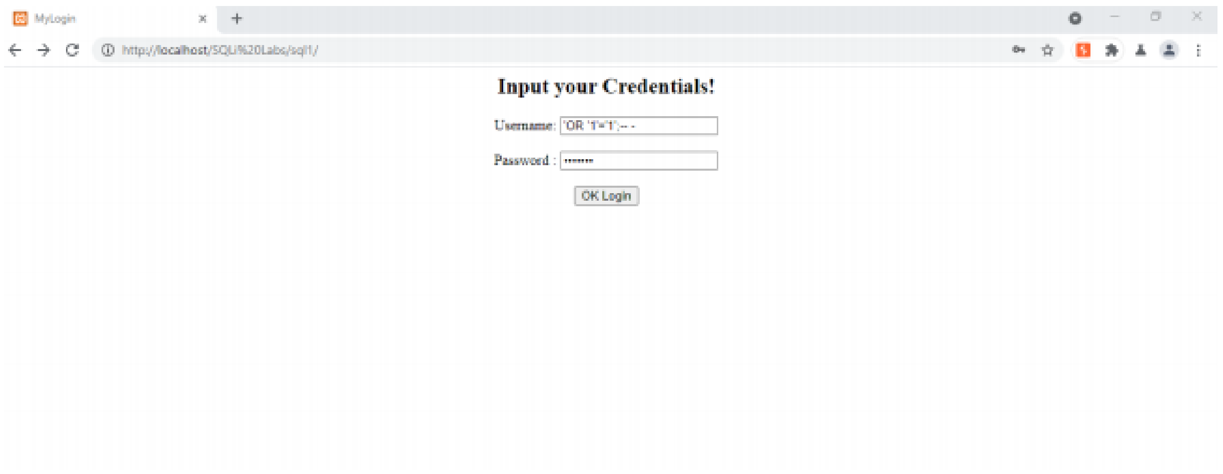
Other tools for testing on local webserver: XAMPP

XAMPP is an abbreviation where X stands for Cross-Platform, A stands for Apache, M stands for MYSQL, and the Ps stand for PHP and Perl, respectively. It is an open-source package of web solutions that includes Apache distribution for many servers and command-line executables along with modules such as Apache server, MariaDB, PHP, and Perl. XAMPP helps a local host or server to test its website and clients via computers and laptops before releasing it to the main server. It is a platform that furnishes a suitable environment to test and verify the working of projects based on Apache, Perl, MySQL database, and PHP through the system of the host itself.

Chapter 5: DETAILED DESCRIPTION OF MODULES IMPLEMENTED

Classic injection Bypass(no security controls) :

In this type of attack its very basic payload we use a tautology and try to detect the character for which error is generated. Based on that we brute force different payloads till we get the right one. Line comments are used to ignore rest of the query so that syntax doesn't become an issue.



Input your Credentials!

Username: 'OR '1'='1';-- -

Password : ••••••••

OK Login

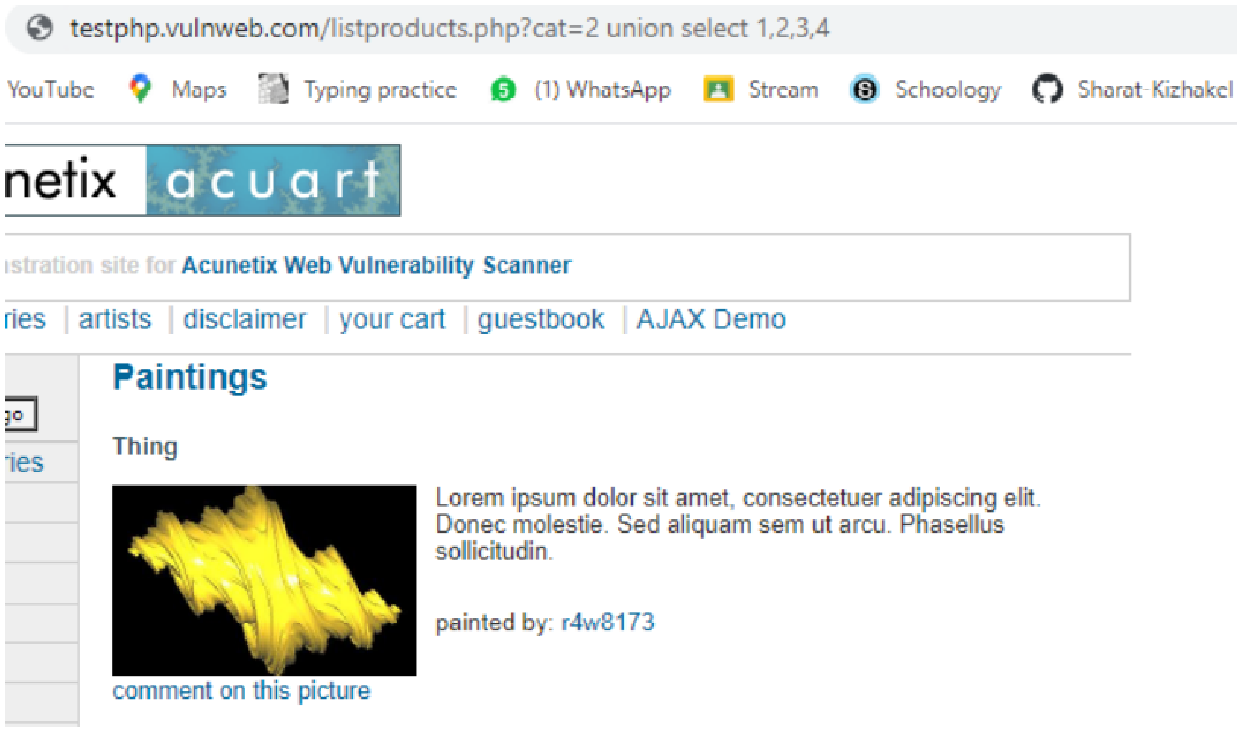
Union-based SQLi :

When an application is vulnerable to SQL injection and the results of the query are returned within the application's responses, then one possibility is that the UNION keyword can be used to retrieve data from other tables within the database. This results in an SQL injection UNION attack. For a UNION query to be used, the two key requirements that are needed are:

1.The individual queries must return the same number of columns.

2.The data types in each column must be compatible with the individual queries.

Basically Union-based SQLi is an in-band SQL injection technique that leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.



testphp.vulnweb.com/listproducts.php?cat=2 union select 1,2,3,4

YouTube
Maps
Typing practice
(1) WhatsApp
Stream
Schoology
Sharat-Kizhakel
netix acuart
istration site for Acunetix Web Vulnerability Scanner
ries | artists | disclaimer | your cart | guestbook | AJAX Demo
go
ies

Paintings

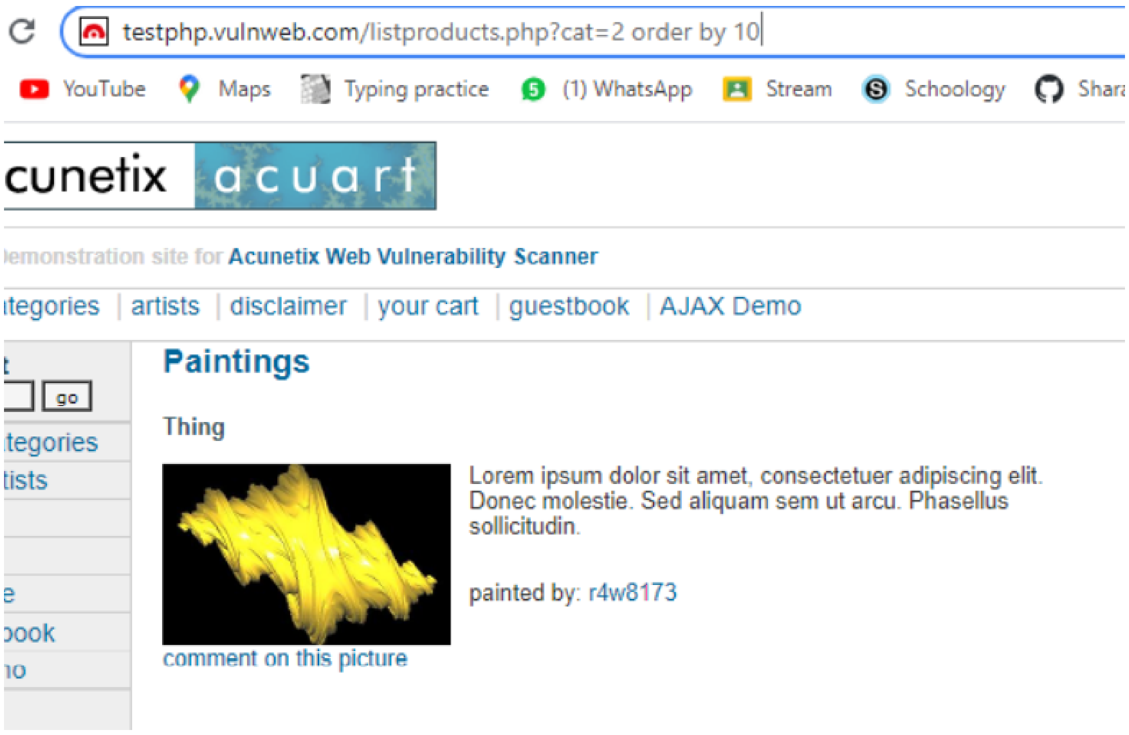
Thing

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

painted by: r4w8173

comment on this picture

Union based SQLi



testphp.vulnweb.com/listproducts.php?cat=2 order by 10

YouTube
Maps
Typing practice
(1) WhatsApp
Stream
Schoology
Shar
cunetix
acuart
emonstration site for Acunetix Web Vulnerability Scanner
tegories | artists | disclaimer | your cart | guestbook | AJAX Demo
go
tegories
tists
e
ook
o

Paintings

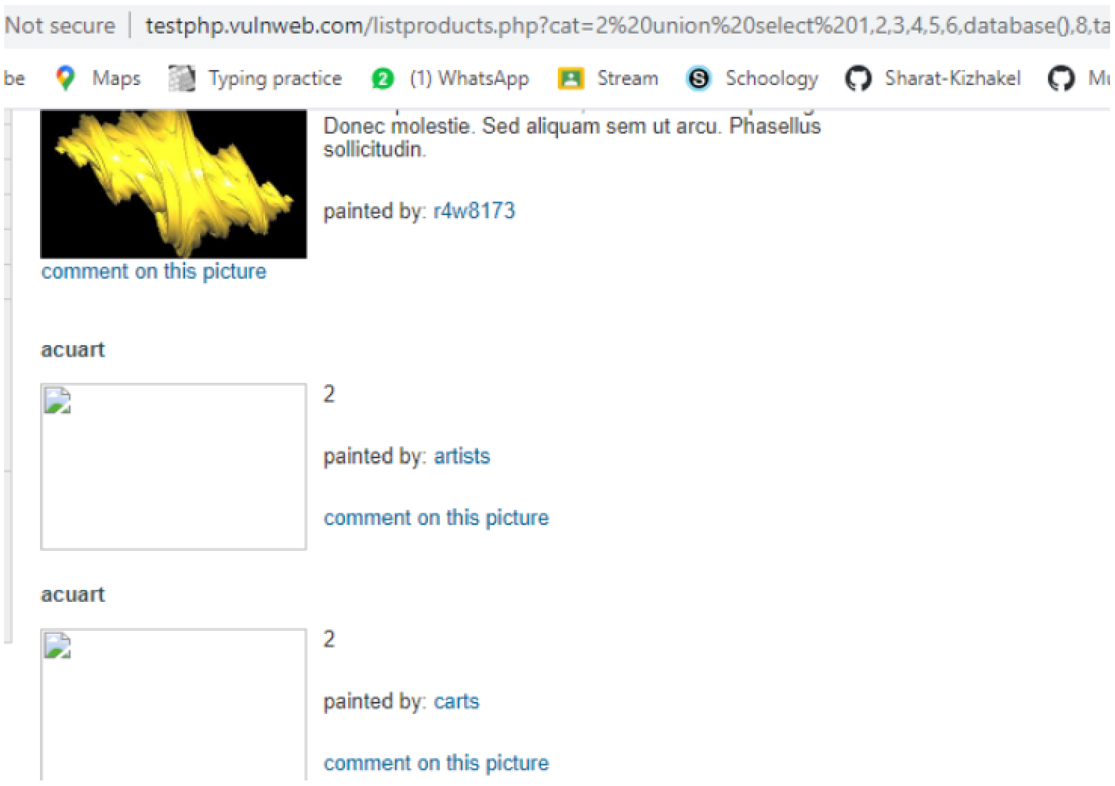
Thing

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

painted by: r4w8173

comment on this picture

Union based SQLi using order by



Not secure | testphp.vulnweb.com/listproducts.php?cat=2%20union%20select%201,2,3,4,5,6,database(),8,ta

Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

painted by: r4w8173

comment on this picture

acuart

2

painted by: artists

comment on this picture

acuart

2

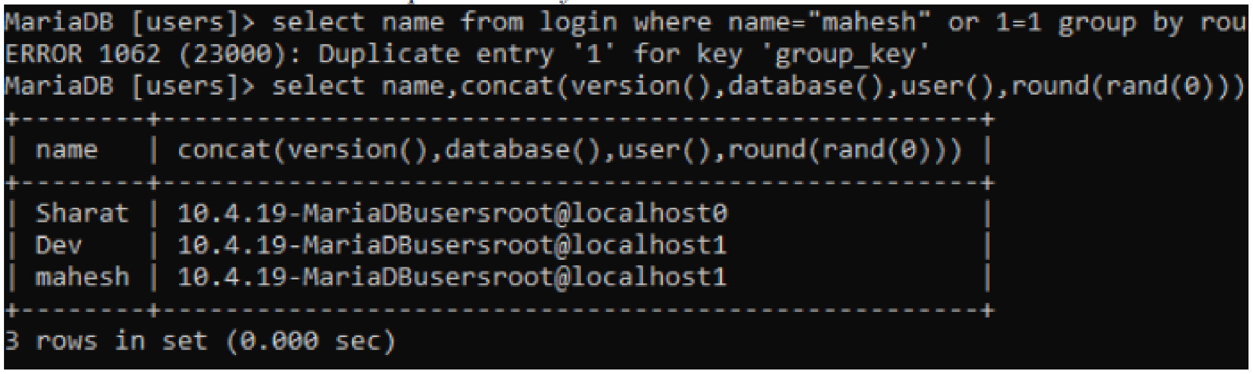
painted by: carts

comment on this picture

Error-Based SQli:

Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site or logged to a file with restricted access instead. The error messages can be used to return the full query results, or gain information on how to restructure the query for further exploitation.

Duplicate entry error



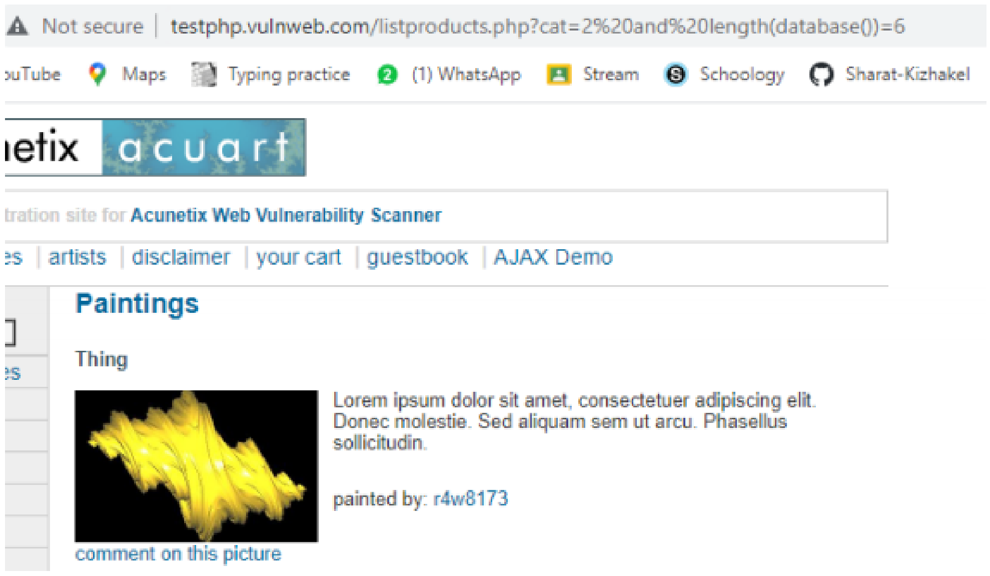
```
MariaDB [users]> select name from login where name="mahesh" or 1=1 group by rou
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
MariaDB [users]> select name,concat(version(),database(),user(),round(rand(0)))
+--------+-----------------------------------------------+
| name   | concat(version(),database(),user(),round(rand(0))) |
+--------+-----------------------------------------------+
| Sharat | 10.4.19-MariaDBusersroot@localhost0           |
| Dev    | 10.4.19-MariaDBusersroot@localhost1           |
| mahesh | 10.4.19-MariaDBusersroot@localhost1           |
+--------+-----------------------------------------------+
3 rows in set (0.000 sec)
```

Inferential SQLi(Blind SQLi):

Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as dangerous as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the web application and the attacker would not be able to see the result of an attack in-band. This is the reason they are called blind SQLi attacks. Instead, an attacker is able to reconstruct the database structure by sending payloads, observing the web application's response and the resulting behavior of the database server.With blind SQL injection vulnerabilities, many techniques such as UNION attacks are not effective, because they rely on being able to see the results of the injected query within the application's responses. It is still possible to exploit blind SQL injection to access unauthorized data, but different techniques must be used.

boolean-based blind SQLi:

Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result. Depending on the result, the content within the HTTP response will change or remain the same.



Not secure | testphp.vulnweb.com/listproducts.php?cat=2%20and%20length(database())=6

Paintings

Thing

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

painted by: r4w8173

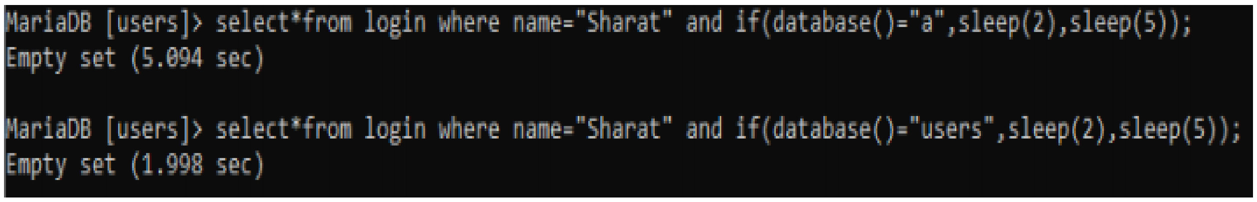
comment on this picture

uTube
Maps
Typing practice
(1) WhatsApp
Stream
Schoology
Sharat-Kizhakel
ıetix
acuart
tration site for Acunetix Web Vulnerability Scanner
s | artists | disclaimer | your cart | guestbook | AJAX Demo

page loading on true condition

Time-based blind SQLi:

Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned.

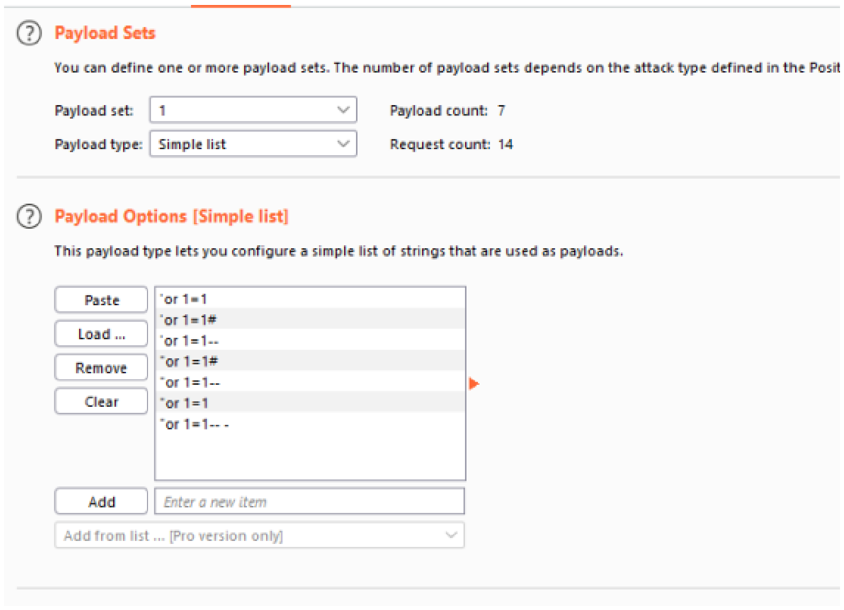


```
MariaDB [users]> select*from login where name="Sharat" and if(database()="a",sleep(2),sleep(5));
Empty set (5.094 sec)

MariaDB [users]> select*from login where name="Sharat" and if(database()="users",sleep(2),sleep(5));
Empty set (1.998 sec)
```

Semi-automated tool for SQL Injection :

It can be a time consuming process to manually check various payloads in the attack query. Different permutations of the same will be possible depending on ’ or ” or different comment injections like ’–’,’’,’– -’ and even ’/*’. Therefore instead of manually checking all the combinations, we can save the different possible payloads in a separate file which can then be uploaded on the intruder tab of Burp suite, an integrated platform for security testing. With the help of Burp suite we can directly get to know which payloads are successfull and which are not.



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Posit

Payload set: 1 Payload count: 7

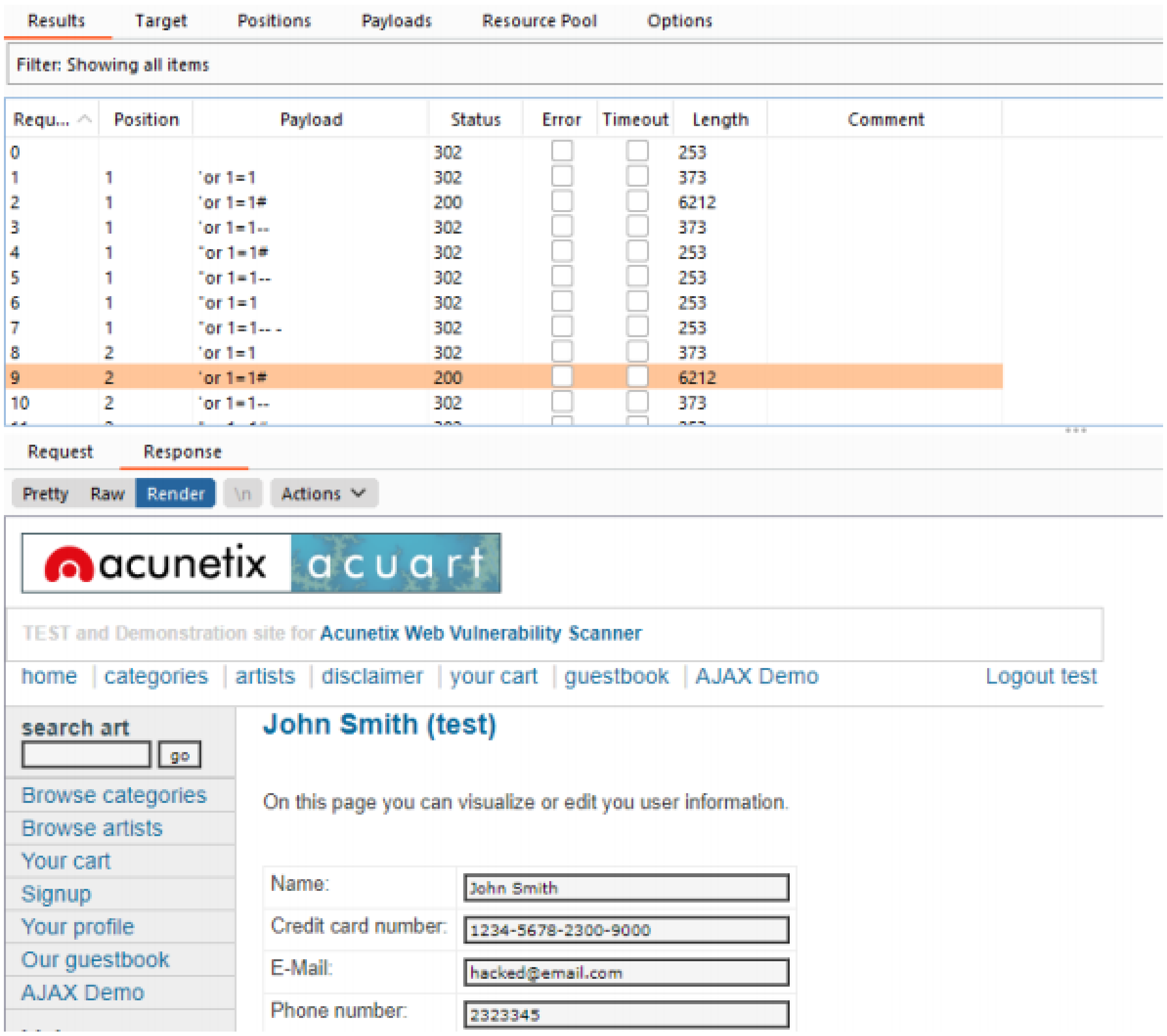
Payload type: Simple list Request count: 14

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
'or 1=1
'or 1=1#
'or 1=1--
"or 1=1#
"or 1=1--
"or 1=1
"or 1=1-- -
```

Payload using Burp Suite



Filter: Showing all items

Requ...	Position	Payload	Status	Error	Timeout	Length	Comment
0			302	☐	☐	253	
1	1	'or 1=1	302	☐	☐	373	
2	1	'or 1=1#	200	☐	☐	6212	
3	1	'or 1=1--	302	☐	☐	373	
4	1	"or 1=1#	302	☐	☐	253	
5	1	"or 1=1--	302	☐	☐	253	
6	1	"or 1=1	302	☐	☐	253	
7	1	"or 1=1-- -	302	☐	☐	253	
8	2	'or 1=1	302	☐	☐	373	
9	2	'or 1=1#	200	☐	☐	6212	
10	2	'or 1=1--	302	☐	☐	373	

John Smith (test)

On this page you can visualize or edit you user information.

Name:	John Smith
Credit card number:	1234-5678-2300-9000
E-Mail:	hacked@email.com
Phone number:	2323345

Testing output for different lengths

Chapter 6: NEW LEARNINGS FROM THE TOPIC

During the course of this seminar the key takeaways have been:

- What an SQL Injection Attack is and how it works under the hood. A SQL injection attack is when a third party is able to use SQL commands to interfere with back-end databases in ways that they shouldn't be allowed to. A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. SQL Injection is an attack that poisons dynamic SQL statements to comment out certain parts of the statement or appending a condition that will always be true. It takes advantage of the design flaws in poorly designed web applications to exploit SQL statements to execute malicious SQL code.

- The different types of SQL Injection Attacks based on which channel is used. Besides this a lot of information regarding in band and inferential SQLIA was executed practically on XAMPP. The execution of the queries beforehand in mariaDB made it easier to understand the attack in the browser on the intentionally vulnerable site http://testphp.vulnweb.com.

- The use of a semi-automated tool(Burp Suite) in order to test the payloads against a vulnerable website. The semi-automated tool is basically testing multiple different combinations at once without the user having to manually type it out, thereby saving time. Based on difference in length the different output rendered finally implies which attack was successful.

- The different types of preventive techniques used against SQL Injection Attacks which include the parameterized queries and stored procedures which are the universally used basic defense techniques.

- Understanding novel approaches to preventing SQL Injections from recent research papers including code based analysis an approach which takes into consideration complex numbers to detect malicious code.

- use of LATEX typesetting tool for generating reports.

REFERENCES AND ANNEXURES

CITATIONS:

[1] D. A. Kindy and A.-S. K. Pathan, “A survey on sql injection: Vulnerabilities, attacks, and prevention techniques,” in 2011 IEEE 15th international symposium on consumer electronics (ISCE). IEEE, 2011, pp. 468–471.

[2] J. Clarke-Salt, SQL injection attacks and defense. Elsevier, 2009.

[3] A. Jana and D. Maity, “Code-based analysis approach to detect and prevent sql injection attacks,” in 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, 2020, pp. 1–6.

[4] L. Ma, D. Zhao, Y. Gao, and C. Zhao, “Research on sql injection attack and prevention technology based on web,” in 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA). IEEE, 2019, pp. 176–179.

[5] Z. Chen, M. Li, X. Cui, and Y. Sun, “Research on sql injection and defense technology,” in International Conference on Artificial Intelligence and Security. Springer, 2019, pp. 191–201.

SOURCES:

1.https://www.javatpoint.com/

2.https://www.acunetix.com/

3.https://www.imperva.com/

4.https://www.geeksforgeeks.org/

5.https://owasp.org/

6.https://portswigger.net/