Indian Institute of Technology Jodhpur

CSL6010 – Cyber Security
LAB 9 Report

Name- Aman Srivastava
Roll No.- B20CS100

Date-13th Apr 2023

AIM: Understanding SQL Injection attack.

Part 1:How SQL injection works.

```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`email` VARCHAR(45) NULL,
`password` VARCHAR(45) NULL,
PRIMARY KEY (`id`));
insert into users (email,password) values ('iit@j.com',md5('abc'));
```

```
select * from users;
```

The SQL query "SELECT * FROM users WHERE email = 'cybersecurity@iitj.ac.in' AND password = md5('CSL6010');" is susceptible to SQL injection attacks if the email and password values are not properly validated or sanitized. An attacker can take advantage of this vulnerability by injecting malicious SQL code into either the email or password field.

For instance, the attacker may input the following:

Email: “” or 1=1 --
Password: “” or 1=1 --

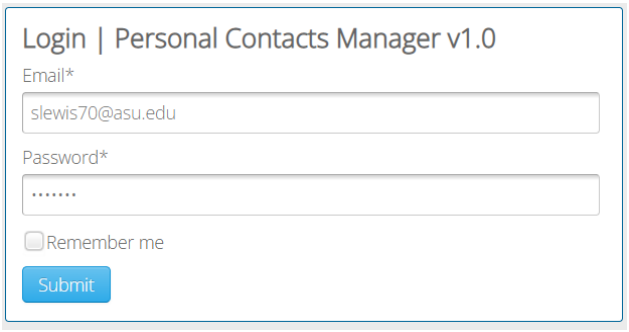
After injection, the resulting SQL query would be:

SELECT * FROM users WHERE email = "" or 1=1 -- AND password = md5(“” or 1=1 --);

```
SELECT * FROM users WHERE email = "" or 1=1 -- AND password = md5('' or 1=1 --);
```

The double-dash (--) is used to comment out the rest of the original query. The injection causes the query to return all rows from the "users" table since the condition "1=1" is always true. This allows the attacker to bypass the password authentication check and gain unauthorized access to the system.

Part 2: SQL inject in a web application



The dashboard is accessed with any random email and password.

Dashboard | Personal Contacts Manager v1.0

Add New Contact | Log Out

ID	First Name	Last Name	Mobile No	Email	Acti
1	mynams	jenefry	9898989898	admin@gmail.com	
62425	Dark	sjcdshgcd	7766863532	jimahsylvester12@gmail.com	Edit
62426	Bandi	Banda	06303690115	rajeshbandi106@gmail.com	Edit
62427	Dark	add	5554444	admin@xyz.com	Edit
62428	Dark	Dark		admin@xyz.com	Edit
62429	sasa	sasa	sasa	adddminn@gmail.com	Edit
62430	Dark	maiden	9878934543	maiden2008@gmail.com	Edit
62431	xoxox	xoxox	555555555	xxx@xxx.xxx	Edit
62432	tyuiof	ertyuiop	23456789	rtyu@1234.in	Edit
62433	Black	White	9532073332	ranulfoensoyjr@gmail.com	Edit
62434	Tmavý	bílí	123456789	ipadrip.cz@gmail.com	Edit
62435	ADAM	AZMI	01156677330	adamdanish227@gmail.com	Edit
62436	Ricardo	Rios	50457285576	adelaidegraham44@gmail.com	Edit
62437	Dark	Hacker	0987678897	polkjuhygtfdghjkl@gmail.com	Edit
62438	qwrt	yuio	5555555555	qwrt@yuio.com	Edit
62439	D	G	55	55@gmail.com	Edit
62440	john	doe	1234	john@axy.com	Edit
62441	Dark	Maiden	87635444242	darkmaiden@octpus.ps	Edit
62442	qedhqe	wegdfe	34802945	347572@gmail.com	Edit
62443	w	a	v	c@gmail.com	Edit
62444		a	b	c@gmail.com	Edit
62445	Dark	qwery	091321232123	admin@google.com	Edit

In SQL injection attacks, an attacker injects malicious SQL code into a web application's input fields, exploiting vulnerabilities in the application's code to access sensitive data or execute unauthorized actions. In the example provide, the attacker used the following input for the password field:

xxx') OR 1 = 1 --

This input is an example of a SQL injection attack that exploits a vulnerability in the SQL query used by the web application to validate user input. The attacker used a SQL injection technique known as a "Boolean-based blind" attack to bypass authentication and gain access to the system.

In this particular attack, the attacker appended a SQL comment (--) to the end of the injected code to prevent the rest of the original SQL query from executing. The injected code contains a Boolean expression that is always true (1=1), which means that the attacker can bypass the password check and gain access to the system.

For the password "xxx') OR 1 = 1 --", assuming that the email is "xxx@xxx.xxx", the SQL statement could look like:

SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = 'xxx\') OR 1 = 1 --'

This SQL statement is vulnerable to SQL injection attacks, as the attacker can use the injected code ("') OR 1 = 1 --") to bypass the password check and gain unauthorized access.

For the password "1234", assuming that the email is "xxx@xxx.xxx", a typical SQL statement for authentication could look like:

SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = '1234'

This SQL statement compares the user-provided password with the password stored in the database. If they match, the user is authenticated and granted access.

For the password "xxxx", assuming that the email is "xxx@xxx.xxx", a typical SQL statement for authentication could look like:

SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = 'xxxx'
This SQL statement compares the user-provided password with the password stored in the database. If they match, the user is authenticated and granted access.

Note that the second password ("1234") is not vulnerable to SQL injection attacks, as it does not contain any injected code.