**AIM:** **To study simulation of Cross-Cite Scripting attack and SQL injection attack.**

**THEORY: Cross-Cite Scripting attack**

**o What is Cross-Cite Scripting attack?**

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. Cross-site Scripting (XSS) is a client-side code injection attack. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user’s browser. Vulnerable vehicles that are commonly used for Cross-site Scripting attacks are forums, message boards, and web pages that allow comments.

**o How to prevent this attack?**

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.

Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.

Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

**Attack Snapshots:**

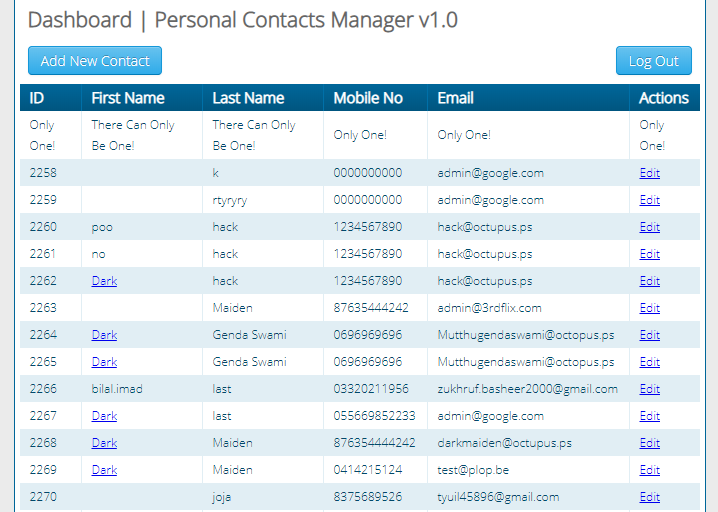


Figure 1:Dashboard

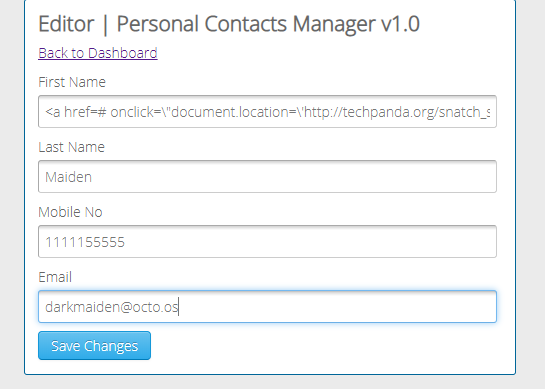


Figure 2:Adding the user for Cross-Cite Scripting Attack

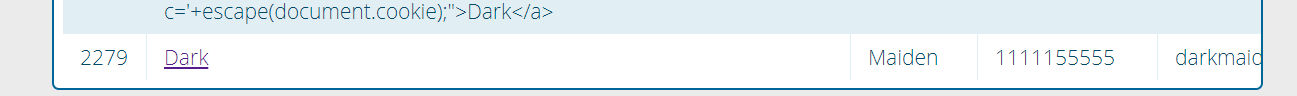


Figure 3:User Added

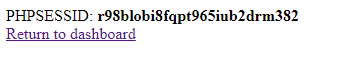


Figure 4:PHP SESSION ID found

**o How SQL Injection works.**

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more.

**o Preventive measures.**

1.Trust no-one: Assume all user-submitted data is evil and validate and sanitize everything.

2. Don't use dynamic SQL when it can be avoided: used prepared statements, parameterized queries or stored procedures instead whenever possible.

3.Update and patch: vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it's vital to apply patches and updates as soon as practical.

4.Firewall: Consider a web application firewall (WAF) – either software or appliance based – to help filter out malicious data. A WAF can be particularly useful to provide some security protection against a particular new vulnerability before a patch is available.

5.Reduce your attack surface: Get rid of any database functionality that you don't need to prevent a hacker taking advantage of it.

6.Use appropriate privileges: don't connect to your database using an account with admin-level privileges unless there is some compelling reason to do so.

7.Keep your secrets secret: Assume that your application is not secure and act accordingly by encrypting or hashing passwords and other confidential data including connection strings.

8.Don't divulge more information than you need to: hackers can learn a great deal about database architecture from error messages, so ensure that they display minimal information. Use the "RemoteOnly" customErrors mode (or equivalent) to display verbose error messages on the local machine while ensuring that an external hacker gets nothing more than the fact that his actions resulted in an unhandled error.

9.Don't forget the basics: Change the passwords of application accounts into the database regularly.

10.Buy better software: Make code writers responsible for checking the code and for fixing security flaws in custom applications before the software is delivered.

**Attack Snapshots:**

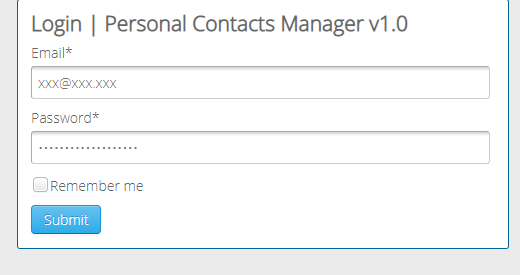


Figure 5:Trying SQL Injection

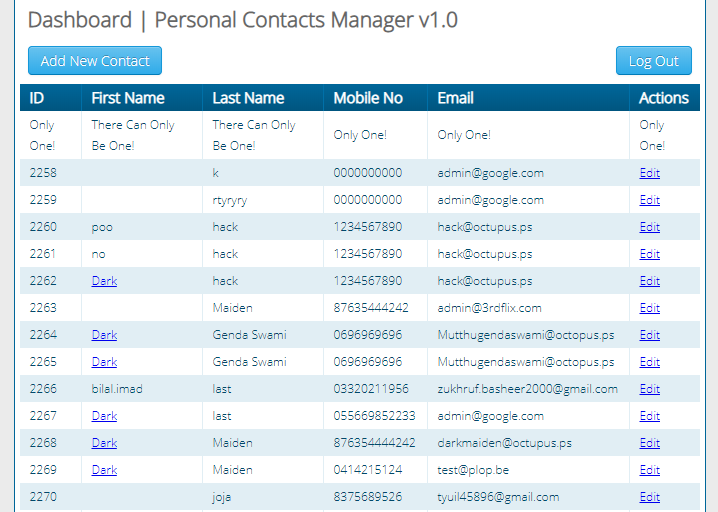


Figure 6:Attack Successful

**CONCLUSION:**

From this experiment we learnt two new attacks namely cross cite scripting attack and SQL injection attack. In cross cite scripting attack which is client side code injection we add malicious scripts which are legitimate for websites and get important information whereas in SQL injection we will execute malicious SQL statements and try to get access of the database. Once we get into the database we can add ,insert ,update and delete records too. We implemented both attacks on techpanda.org and saw how vulnerable these attacks are for unsecured websites.