**CSE 545: Software Security**

**Cross-site Scripting and SQL Injection Prevention in PHP based websites**

Maitrayee Navnath Pingale

Ira A Fulton School of Computing, Informatics and Decision Systems Engineering

Arizona State University

1211206968 **|** [mpingale@asu.edu](mailto:mpingale@asu.edu)

**Abstract**

The project focuses on web insecurities and points to cross-site scripting and SQL injection attacks which are prominent vulnerabilities in most of the web applications. This project talks about what all ways these attacks can be carried out and what all can be done to prevent them from getting hold of any sensitive information or even playing with the application.

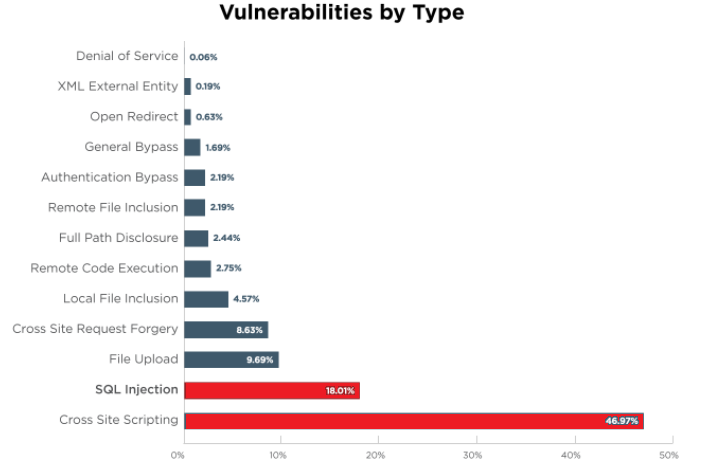


Figure: SQL Injection and Cross Site Scripting are the most common occurring vulnerabilities

**Introduction**

A type of attack where users insert or inject malicious scripts into web sites is known as cross-scripting attack or XSS. This attack is nothing but a form in which the attacker inserts server-side scripts into usually safe looking websites and is done mostly with a web request. It is rather very difficult to differentiate the malicious scripts inserted among the server code which is trusted. Other users won’t be able to distinguish codes which are safe to execute and code which aren’t safe. Executing the attackers malicious script can result in giving out the any valuable information. These malicious scripts are also known as malicious payloads. XSS is on of the widest spreading web attacks now days. The common reason of this attack is unvalidated or unencoded user inputs. By doing this the attacker takes advantage of any vulnerabilities in the web application of the victim.

User inputs of any kind (for e.g. comments, form fields or pictures) on a website can be a malicious input and cause a XSS attack. Hence, any user input must be sanitized and validated with established rules. This attack can easily be prevented by sanitizing the inputs coming into the web system from an external source.

Another type of attack that we are talking in this project is SQL injection attack. This another web-based attack or a hacking technique in web applications. What it does is inject some malicious code into your database and destroy it. It usually takes place in a data driven application, using malicious queries to infiltrate the database and gain valuable information. SQL injection attack is one of the oldest, most common and even more dangerous web application attacks as this could give out authentication information of users. The attacker can now have unauthorized access to valuable/sensitive data for e.g. trade secrets, customer data etc. The attacker can tamper the data and cause unsolvable issues. PHP and ASP applications are commonly vulnerable to this kind of attack.

Using some of the established rules and regulations, this project aims at building a wrapper or a library which will prevent XSS attack in a PHP website using the guidelines given by OWASP – Open Web Application Security Project. Also, the wrapper will prevent any SQL injection attacks by sanitizing the input from the user.

**Solution**

The project aims at creating a basic wrapper which will take the user input, before processing, sanitize and validate it and the forward it to the next level. This will help in CTF to counter any of the XSS or SQL injection attacks.

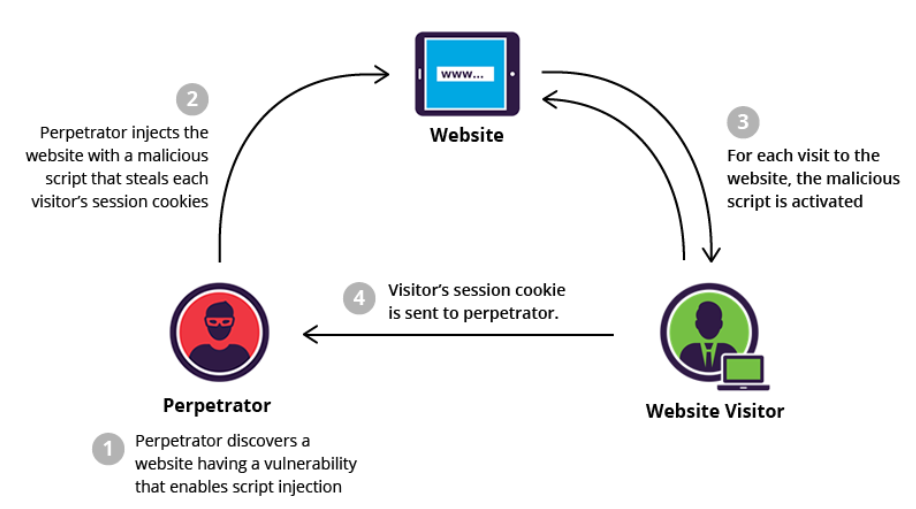


Figure: High level overview of XSS attack.

For XSS we have referred to the OWASP’s XSS prevention cheat sheet rules. These are some string sanitization rules which will help prevent XSS. They are as follows:

* RULE 0:

This rule makes sure that no untrusted data gets inserted except in the locations where it can (allowed locations by the system). If the system says no string is to be inserted in some field, it will simply replace the input with a blank or an empty string.

* RULE 1:

While inserting user input data into any HTML element content, special characters are escaped. This is done so that the user’s input doesn’t try to modify the HTML tags and put in its own script.

* RULE 2:

This rule takes care of the user inputs which will be put directly into HTML as HTML common attributes, for e.g. width, name. Here all the characters with ASCII value less than 256 with &#xHH.

* RULE 3:

This rule applies while inserting data into javascript data values. It is like rule 2, but that here we have to escape all the non-alphanumeric characters with ASCII less than 256 with \xHH format.

* RULE 4:

While inserting data into HTML style property tags, we need to escape all the characters with ASCII value less than 256 and format as \HH.

* RULE 5:

This rule investigates HTML URL parameters, that is the value that goes into URL parameters. Here we escape all the non-alphanumeric characters with ASCII value less than 256 and escape format as %HH.

These are the six input sanitization rules that are incorporated into our project to prevent XSS attacks.

Now let’s talk about SQL injection attack. For overcoming SQL injection attack, we have implemented three functions each of which sanitizes different kind of inputs.

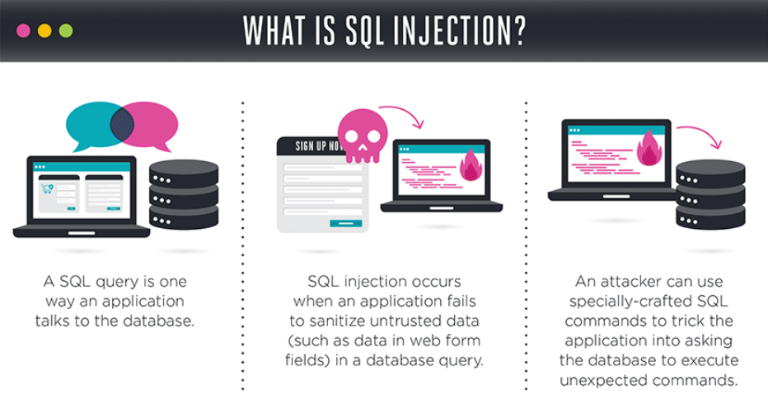


Figure: SQL injection

Function SQL\_Escaping, escapes all the special characters in the string which is used as a SQL query statement. This function works like the mysql\_real\_escape\_string function in PHP.

The next function is to avoid any second order SQL injection attacks. What are SQL injection second order attacks? Here the payload or the malicious data is already stored into the database. So, we sanitize the strings which have special characters and needed string is numeric. It will escape all non-alphanumeric characters.

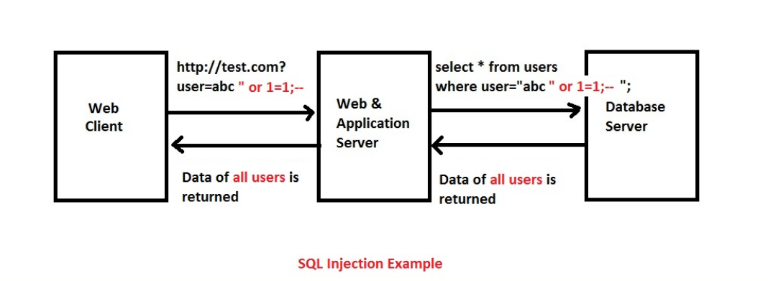


Figure: Example of SQL injection attack

The last function checks if the input needed was numeric and the input given in by the user is numeric. If the input given by the user isn’t numeric, it will make the value as 0.

For the implementation of the wrapper, the admin needs to run and configure it to sanitize strings. As and when, the wrapper is configured and running, the admin can direct the incoming traffic of user inputs to the wrapper rather than going directly to the web application. As the wrapper starts receiving the inputs, it parses them and sends it to appropriate functions to sanitize them. All the rules are defined in separate functions which takes in the untrusted or malicious input, checks for the faulty characters or strings, removes them and returns the validated string back to the web application.

Working and Demonstration

1) All required files should be kept in the same folder as the web service that is wrapped.

* Required files:
* pctf-filter.php, xss\_sql\_filter.php, ,driver.php, .htaccess
* To configure the wrapper: ./pctf-filter.php <command> [<args>]
* For a list of available commands: './pctf-filter.php commands'
* For more info about a specific command: './pctf-filter.php help <command>'
* Possible commands: commands, help, set, unset
* For using set to add a variable to filter: ./pctf-filter.php set <page> <type> <variable> <rule-id>

where type is the request type and page is the php page of the original website where the request is headed.

2) By importing the xss\_sql\_filter.php, one can use the library directly and then call correct functions to sanitize inputs before putting them into the source code.

Results:

Query fetched: http://localhost/Test/?var=\x00%20\n%20\r%20%27%20"%20<script>alert(%27Hello%27)</script>%20SELECT%20\*%20FROM"

Outputs as per displayed by the browser:

* Output of filter\_Disallow:
* Output of filter\_HTMLEntityEncode: \x00 \n \r ' " <script>alert('Hello')</script> SELECT \* FROM"
* Output of filter\_For\_Attributes: \x00 \n \r ' " <script>alert('Hello')</script> SELECT \* FROM"
* Output of filter\_For\_Javascript: \x5cx00\x20\x5cn\x20\x5cr\x20\x27\x20\x22\x20\x3cscript\x3ealert\x28\x27Hello\x27\x29\x3c\x2fscript\x3e\x20SELECT\x20\x2a\x20FROM\x22
* Output of filter\_For\_CSS: \5cx00\20\5cn\20\5cr\20\27\20\22\20\3cscript\3ealert\28\27Hello\27\29\3c\2fscript\3e\20SELECT\20\2a\20FROM\22
* Output of filter\_For\_URL: %5cx00%20%5cn%20%5cr%20%27%20%22%20%3cscript%3ealert%28%27Hello%27%29%3c%2fscript%3e%20SELECT%20%2a%20FROM%22
* Output of filter\_SQL\_Escaping: \\x00 \\n \\r \' \" SELECT \* FROM\"
* Output of filter\_SQL\_CompleteEscape: \\x00\ \\n\ \\r\ \'\ \"\ \alert\(\'Hello\'\)\<\/script\>\ SELECT\ \\*\ FROM\"
* Output of filter\_Char2Num\_id: 0

**Conclusion and Recommendation**

The project helped us to gather knowledge about web vulnerabilities, importantly cross-site scripting and SQL injection, both of which are the most common occurring vulnerabilities in web applications now days. We learnt reasons for such vulnerabilities to occur in the web applications and how can they be prevented. Also using this project in CTF made us understand the limitations it has and how much more we can improve on this.

**Individual Contribution**

* Worked on gathering information to understand reasons for XSS attacks and SQL injection attacks.
* Wrote code to prevent XSS attacks with untrusted data in HTML element content, HTML common attributes, JavaScript data values and HTML style properties.
* Worked on understanding reasons for SQL injection second order attack.
* Coded function to prevent SQL injection second order attack.
* Helped other team members to configure and run wrapper in CTF,
* Worked on documenting the process and presenting it in a form of report.
* Documented limitations of the project when implemented in CTF

**Knowledge Acquired**

The project gave great understanding of what web insecurities are. It made us dig deeper into cross-site scripting and SQL injection attacks which gave us a key lesson on how important it is to secure any web application so as not to leak any data or not let any untrusted source get in. Even though the program ran perfectly in most of the test cases during the CTF, but many a times the service was marked down when the wrapper was active in certain configurations. This made us realize the loop holes that we missed and need to be covered. This made me understand that thorough testing is important before deploying it.

**Team Members**

Group Name: t34m \_r06u3

1. Christopher Hansen
2. William Montgomery
3. Shantanu Bhusari
4. Pratik Iyer
5. Maitrayee Pingale

**References**

1. “Cross-site Scripting”, <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>
2. “XSS Prevention Cheat Sheet”, <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>
3. “Cross-site scripting”, <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>
4. <https://www.acunetix.com/websitesecurity/cross-site-scripting/>
5. “SQL injection”, <https://www.w3schools.com/sql/sql_injection.asp>
6. <https://en.wikipedia.org/wiki/SQL_injection>
7. <https://www.owasp.org/index.php/SQL_Injection>
8. <https://www.incapsula.com/web-application-security/sql-injection.html>