**CSE 545 | Spring 2017**

# Cross-site Scripting & SQL Injection Prevention in PHP-based Websites

**Group: t34m_r06u3**

Christopher Hansen 1000865741 | William Montgomery 1210448314 | Shantanu Bhusari 1211213728
Pratik Iyer 1211366348 | Maitrayee Pingale 1211206968

## The Problem

Cross-site scripting (XSS) is a web-based form of attack where users inject malicious scripts into web sites. Attacks typically involve an attacker inserting server-side scripts into otherwise safe websites, typically through a web request. Since the malicious code is inserted among trusted server code, other users are unable to distinguish which code is safe and which code is not safe, causing the attacker's script to be executed on other users' machines. Cross-site scripting attacks vary greatly with respect to methods of insertion and severity and most commonly access information stored by the user's machine that used with that site such as cookies and session information. With this information, an attacker can impersonate the victim on the website and have access to any of their account data. Cross-site scripting is arguably the most prevalent type of security vulnerability in the software industry.

Any user input that is published on a website (e.g. user comments, blog entries, username fields, or even files such as pictures) could potentially be vulnerable to malicious input in a XSS attack, so each given input must be sanitized according to the rules which correspond to its input type and publish location. Cross-site scripting is most easily prevented by validating user input on the server before publishing it. There are established rules and guidelines to prevent malicious code from being inserted into a website based on the location in the code that the user's input is inserted into. The aim of this project is to create a usable library which will prevent cross-site scripting in up to five potential types of locations in a given PHP website based on the recommendations by the Open Web Application Security Project (OWASP) (https://www.owasp.org/index.php/Cross-site_Scripting_XSS).

SQL injection is a type of attack like XSS in that it is web-based and is typically executed through user input fields through web requests to attack a webpage. SQL injection differs, however, in that it is an attack pointed at the database of a webpage. SQL injection can be used to get control of a SQL database to collect sensitive information such as passwords or to manipulate the database such as dropping tables. While this project was originally intended to protect from just XSS attacks, the groundwork laid to sanitize user input made it simple to add some protection against SQL injection as well

## The Approach

This project is designed to be a basic wrapper which sanitizes user input before it is forwarded to be processed by the intended website. This project will be usable during CTF competitions in which a XSS vulnerability is identified in a php web service. String sanitization rules are implemented for the following possible destinations of user input:

XSS:

Rule #0 – Admin decides when locations are not allowed. If it is decided that no user-created string should be entered into a particular field, this rule can prevent that by simply setting the user's input to an empty string.

Rule #1 – Inserting Data into HTML Element Content – Special characters need to be escaped so that an attacker cannot modify the HTML to switch to an execution, such as in a script tag.

Rule #2 - Inserting Data into HTML Common Attributes – In this case, all characters with ASCII values less than 256 with the &#xHH; format that are not alphanumeric are escaped.

Rule #3 - Inserting Data into JavaScript Data Values – Similar to rule #2, all non-alphanumeric characters with ASCII values less than 256 must be escaped, but this time with the \xHH format.

Rule #4 - Inserting Data into HTML Style Property Values – All non-alphanumeric characters with ASCII values less than 256 with the \HH escaping format should be escaped.

Rule #5 - Inserting Data into HTML URL Parameter Values – All non-alphanumeric characters with ASCII values less than 256 with the %HH format should be escaped.

SQL Injection:

We have implemented 3 functions for handling SQL injections. The function, filter_SQL_Escaping will behave like mysql_real_escape_string function in php, where all the special characters will be escaped in a string which is used as an sql statement.

The second function, filter_SQL_CompleteEscape help us sanitize the strings which will have a numeric input where we need not necessarily a special character. It will append a \ to all the non-alphanumeric characters. By making use of this function we avoid second order SQL injection.

The third function written for SQL injection prevention is filter_Char2Num_id. This function checks if input expected is numeric and checks if the input given by user is not numeric, it will straight away make input value equal to 0.

For a more detailed list of XSS prevention rules, see
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

Implementation begins when the website admin runs and configures the wrapper to sanitize strings based on name and destination within the page in addition to the destination page itself. Once the wrapper is active and properly configured, the admin can point incoming requests at the wrapper rather than the original website. After being initialized and receiving the list of strings to secure, the wrapper waits to receive requests from users seeking to access the website. Once a request is received, the server parses the strings which need to be protected and sends each of them to their appropriate sanitization function. Each rule has

a function which takes in the untrusted string, searches for potentially malicious code, removes suspicious characters or sets of characters, and returns the safe string. Once all the strings have been sanitized, the wrapper passes on the safe strings to the website (which has been included by the wrapper).

## Usage

1) All files should be placed in the same folder as the web service which is being 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.

Example: ./pctf-filter.php set /index.php get msg 1

For using unset to remove a variable from being filtered: ./pctf-filter.php unset <page> [<type> <variable>]

Example: ./pctf-filter.php unset index.php get msg

The configuration of the wrapper is stored in the file filter.json.

2) The library can also be used directly with the source code by importing ‘xss_sql_filter.php’ and using appropriate function to sanitize inputs before placing them in the source code.

## Demonstration

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

## Limitations

1. This solution is PHP-based and requires the protected web service to be in PHP as well (PHP version 5.6 or newer is required).
2. Specific vulnerabilities need to be identified by the administrator as well as the appropriate prevention tool. This wrapper must also be properly configured and all variables properly identified by the administrator.

## PCTF Performance and Improvement

After participating in the class CTF competition, many lessons were learned, including how to improve the performance of this program. Although the program ran perfectly with many test cases that we implemented, we are uncertain whether it helped the group during competition or not, as the service was marked as being “down” when the wrapper was activated with some configurations. Too much time was spent trying to configure the .htaccess file to direct traffic at the wrapper. If the group were to compete in another CTF, the highest priorities would be to practice configuring the server to direct requests to the wrapper (the practice server was taken down during the time we had scheduled to practice) and to improve network traffic monitoring. By logging the HTTP requests that were coming and going, the group could make the necessary to fixes to make sure the wrapper was functioning properly. This would also help the group recognize how other groups were performing successful attacks, and then potentially use those inputs on other groups as well.

If the group were to continue working on this wrapper, it could be improved by:

1. Adding additional rules for string inputs
2. Adding support for protecting against XSS attacks in user-submitted files
3. Automating parsing to identify vulnerabilities and send strings to appropriate sanitization functions