XSSGuard

Team Members:
Varun Chandra Jammula 1207940275
Azhaku Sakthi Vel Muthu Krishnan 1210299529

Describe the problem that you are solving

In this project, we developed a security library in PHP that prevents web applications from Cross-Site Scripting (XSS) vulnerabilities. This library does not add any overhead to the application. This library can be used while developing a new application as well as with an existing application. Very minor changes would be required in the existing code to use the library. The library is called XSSGuard.

Describe the approach you took to implementing the project

Cross-Site scripting is a very critical issue that needs to be handled while developing secure web applications. There are various ways in which XSS can be performed. There are discussed below:

1. **Stored XSS (Persistent or Type I)**

   Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim's browser, such as an HTML5 database, and never being sent to the server at all.

2. **Reflected XSS (Non Persistent or Type II)**

   Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and

without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser.

3. **DOM based XSS (Type 0)**

   DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write)."

Hence it is important to provide various options for developers with our library. In this project, we provide two options to the developer. Depending upon the usage of the user input and how it is used, developers can either sanitize the data or embed it into the browser safely.

Describe what a developer or security expert must do to use your approach

There are two ways of using this tool. We discuss the details below:

1. Sanitize the user input based on the usage.

```php
/** This function takes an unsanitized string and
*  returns the sanitized string, based on flags set.
* @param $input
* @param $flag string containing the flag
* @param $min
* @param $max
* @return bool|float|int|mixed|string
*/
function sanitize($input, $flag, $min, $max) {

  if (strcmp($flag, 'PARANOID') == 0) $input = $this->sanitize_paranoid_string($input, $min, $max);
  if (strcmp($flag, 'INT') == 0) $input = $this->sanitize_int($input, $min, $max);
  if (strcmp($flag, 'FLOAT') == 0) $input = $this->sanitize_float($input, $min, $max);
  if (strcmp($flag, 'HTML') == 0) $input = $this->sanitize_html_string($input);
  if (strcmp($flag, 'LDAP') == 0) $input = $this->sanitize_ldap_string($input, $min, $max);
  if (strcmp($flag, 'SHELL') == 0) $input = $this->sanitize_shell_string($input, $min, $max);

  return $input;
}
```

This function takes in malicious user input along with flag. Depending upon where this input is used, we set the flag. For example, if the user input is embedded in a HTML page, we set the flag 'HTML' in the flag variable. If the input is stored in database, then the developers have to use the following:

```
/** This function sanitizes strings that will inserted in to the
*  query or the entire string as query
* @param $link MySQL Connection object
* @param $string Query string or variable
* @return string
*/
function sanitize_sql_string($link, $string)
{
  return mysqli_escape_string($link, $string);
}
```

2. Directly embedding the data into HTML

   More often developers use ***echo, print, print_r*** features of PHP to embed content into HTML page. If the user input contains malicious content, then the application is vulnerable. To avoid this, we provided the developer with our own custom ***XSSecho, XSSprint, XSSprint_r*** functions that have the same functionality as that of the PHP's default functions. These functions internally use ***htmlentities()*** to prevent executing any malicious code in the user input.

Method 1 can be used in scenarios when there are restrictions on the input that is being processed. If the application requires the user input to be stored in a database, displayed to other users such as in a public blog then method 1 would be very useful.

Usage:

```
1)  include "XSSGuard.php";
2)  $xssguard = new XSSGuard();
3)  echo $xssguard->sanitize($_GET["Data"],"HTML","","). "\n";
4)  $xssguard->XSSecho($_GET["Data"]). "\n";
5)  $xssguard->XSSprint($_GET["Data"]). "\n";
6)  $xssguard->XSSprint_r($_GET). "\n";
```

Firstly, include the library in the developer's code. Next create an object to the class. If the developer wishes to sanitize the user input, it is done similar to line 3 and the flag is to be set based on where the input is being used. If the developer just wishes to embed the data directly to HTML, line 4,5 or 6 can be used instead of the PHP's default. This library can be used for preventing various other attacks such as ***SHELL, LDAP, SQL*** attacks successfully by setting the flags for ***sanitize()*** properly.

Describe the limitations of your tool: What types of applications can it be applied to?

This library can only be used with web applications that are developed on PHP. This library also supports applications that are deployed on server running latest version of PHP (7.0.5). This library cannot prevent Client Side XSS (DOM based XSS).

Describe the future work that you can see which would improve your tool.

Using the approach that we followed in this project, we plan to extend the tool and develop libraries in other popular server side languages such as Node.js and also using JQuery or JavaScript on the client side. We also plan to release the library as an open source project.