XSS (Cross-site scripting) prevention

	Group members	ASU ID
1	Abhishek Rao	1210425135
2	Ravi Nihalani	1210448145

1. XSS Attack: Problem that we are solving

Facebook’s bug bounty program has received 2,400+ valid submissions and awarded more than $4.3 million to 800+ researchers around the world. Majority of the issues are discovered in the traditional vulnerability called XSS. It was always an interesting topic to research on for us as every now and then we hear about XSS vulnerabilities revealed in all the top websites.

Cross-site Scripting (XSS) is a client-side code injection attack wherein an attacker can execute malicious scripts into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of invalidated or unencoded user input within the output it generates. XSS enables 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. Cross-site scripting effect may range from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.

By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim’s browser. While XSS can be taken advantage of within VBScript, ActiveX and Flash, unquestionably, the most widely abused is JavaScript – primarily because JavaScript is fundamental to most browsing experiences.

2. Approach used for implementing the prevention of XSS attack

The goal is to implement a security layer that prevents XSS vulnerabilities. We have chosen to define a prevention library in PHP. The PHP library created is named "XSSPrevention". The library provides the flexibility to the developers in terms of three functions which have different functionalities as per the developers need. We have explained the working of each of the functions in detail below. The three methods under "XSSPrevention" are:

a. **The protectXSS($input) Method**:

The protectXSS($input) function takes in a String input defined in $input. This is the most vital function in this library. The function doesn't add/remove any content but consumes the data from the web application user as it is without executing any intended malicious code (javascript, CSS, HTML). Some characters are reserved in HTML. If we use the less than (<) or greater than (>) signs in our text, the browser might mix them with tags. Character entities are used to display reserved characters in HTML. The protectXSS($input) function converts the input to HTML entities such that even if the malicious code is inserted in the form input (retrieved by $_POST method) or through the URI parameters (retrieved by $_GET method), it is not executed but just converted into html entities. In nutshell this function will translate not only the most useful everyday programming HTML characters but all the HTML character entities making it 100% secure.

The most important aspect is that the developers who want to protect their website from XSS attack and are using our library, must use this function wherever the user **inserts** data in the application **AND** wherever the data is **executed** in the application logic **AND** wherever the data is **read** from the database/file **AND** wherever the user inserted data is echoed/printed on the web page. In nutshell, whenever the data is referred across the application, the function should be prefixed before the text.

E.g. : echo $x->protectXSS($_POST['input_name']);

The malicious input could be inserted into the server application via the URL parameters (using the $_GET method) or could be via the form textarea input (retrieved using the $_POST method). Note that after the data has passed through this function, the HTML code is not executed on the browser if the application logic is echoing the data OR the HTML code is not executed in the application wherever the code is referred across the application. We have described the various test cases in the test case document.

Usage eg. 1 : echo $x->protectXSS($_POST['input_name']);

Usage eg. 2 : echo $x->protectXSS($_GET['input_name']);

Usage eg. 3 : print_r(protectXSS($_GET['input_name']));

b. The protectXSSBlackListTags($input) Method

The protectXSSBlackListTags($input) method takes in a String input $input and returns a string without the BlackListed HTML tags. The returned String can be HTML but the returned string is safe to be executed. We have assumed certain Blacklisted tags which if not filtered might cause an XSS attack resulting into malfunctioning of the application or unintended access to the server.

A PHP web developer might want to use this function in scenarios where he wants to give the user an option to input text in the textarea (which will be retrieved via $_POST) with an intention of **executing** it. An example of this is commenting a text on the stackoverflow.com where basic html tags like Bold (<b></b>), etc. have been whitelisted and allowed for the user to execute.

The list of assumed blacklisted tags for this function are:

- "<style>"
- "<div>"
- "<script>"
- "<a>"
- "<h>"
- "<body>"
- "<img>"
- "<marquee>"

- "<iframe>"
- "<input>"
- "<svg>"
- "<bgsound>"
- "<br>"
- "<link>"
- "<xss>"
- "<frameset>"
- "<table>"
- "<base>"
- "<embed>"
- "<xml>"
- "<span>"

The above tags once encountered in the user input are removed and rest of the code is executed (which is safe).

Usage eg.: echo $x->protectXSSBlackListTags($_POST['input_name']);

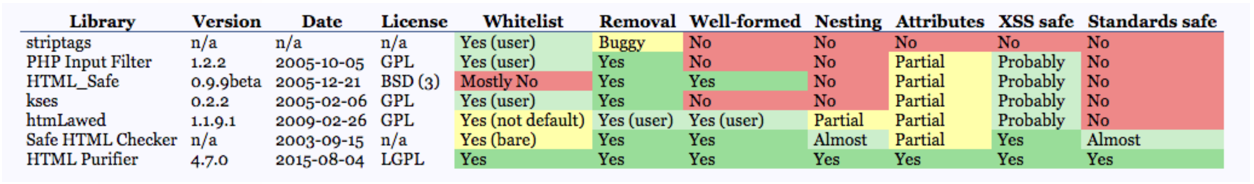
Usage eg.: echo $x->protectXSSBlackListTags($_GET['input_name']);

Usage eg.: print_r(protectXSSBlackListTags($_GET['input_name']));

c. The protectXSSpurifier($input) Method

Similar to the first two functions, even this function takes in a single input parameter which needs to be sanitized. If the developer wants to allow the user to allow execution of HTML in the input, this function is the ideal function to be used. When accepting any input from your users, we can follow the approach of using the first method protectXSS($input) which is 100% secure and will never inject or execute anything malicious, but doesn't give the user any freedom to add and execute any basic HTML, for example for **bolding** text , underlined text, etc. protectXSSpurifier($input) function uses the HTML Purifier library and allows a middle ground: it allows the user to inject some HTML, but not malicious HTML. So, if the developer wants to allow users to inject some html tags so as to modify the context of the text, then he can use our protectXSSpurifier function . protectXSSpurifier function actually parses the HTML, has secure yet permissive whitelist that filters out the nasty stuff that can be present in user submitted HTML. It will also clean your HTML and make sure it's standards compliant.

Below is a comparison table showcasing the benefits of protectXSSpurifier() function which is mainly calling the HTML Purifier library for preventing the XSS attack.



Library	Version	Date	License	Whitelist	Removal	Well-formed	Nesting	Attributes	XSS safe	Standards safe
striptags	n/a	n/a	n/a	Yes (user)	Buggy	No	No	No	No	No
PHP Input Filter	1.2.2	2005-10-05	GPL	Yes (user)	Yes	No	No	Partial	Probably	No
HTML_Safe	0.9.9beta	2005-12-21	BSD (3)	Mostly No	Yes	Yes	No	Partial	Probably	No
kses	0.2.2	2005-02-06	GPL	Yes (user)	Yes	No	No	Partial	Probably	No
htmLawed	1.1.9.1	2009-02-26	GPL	Yes (not default)	Yes (user)	Yes (user)	Partial	Partial	Probably	No
Safe HTML Checker	n/a	2003-09-15	n/a	Yes (bare)	Yes	Yes	Almost	Partial	Yes	Almost
HTML Purifier	4.7.0	2015-08-04	LGPL	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Usage e.g.: echo $x->protectXSSpurifier($_POST['input_name']);

Usage e.g.: echo $x->protectXSSpurifier($_GET['input_name']);

Usage e.g.: print_r(protectXSSpurifier($_GET['input_name']));

3. What does a Developer or Security expert must do to use this XSS Prevention library?

The library "XSSPrevention" created in this project can be used by any developer or Security expert developing web application in PHP with minimum modifications to his code.

In nutshell, all the developer needs to do is the following in his php file:
Step 1: Place the developers php file (index.php in our case) inside the XssPrevention folder provided in the attachment

Step 2: Now inside the developers php file, put the following code in the beginning of the file:

```
include './XssPrevention.inc.php';    // imports the library
$x = new XssPrevention();             // creates an object
```

Step 3: Now the library has been imported and all we need to do is that we need to use any of our three functions (protectXSS(), protectXSSBlackListTags(), protectXSSpurifier()) at all the places where we are retreiving data using $_GET['input_name'] and $_POST['input_name']

Case 1: If the developer wants to allow the user to input any text in the web application and doesn't want to execute it on the browser so that it is 100% secure in terms of XSS attack, he must use the function **protectXSS($input)** wherever ${_GET['input_name']} and ${_POST['input_name']} are used across the php pages of the web application. It could be while echoing or while using the data in the application logic, etc.

Case 2: If the developer wants to allow the user to input any text in the web application and wants to give him the privilege to execute it on the browser, he must use the function **protectXSSBlackListTags ($input)** or **protectXSSpurifier($input).** Except for the assumed blacklisted tags in case of **protectXSSBlackListTags**, the rest of the code will be executed on the browser. The function must be used wherever ${_GET['input_name']} and ${_POST['input_name']} are used across the php pages of the web application. It could be while echoing or while using the data in the application logic, etc.

4. Limitations of the tool / Types of applications supported

The library can only be used by developers developing web applications in PHP and not in any other server side language.
Also this library is only preventing the applications from Reflected XSS attack and Stored XSS on the server side. It doesn't support DOM based XSS attacks.

The third method **protectXSSpurifier**($input) uses HTML Purifier library extensively. Though the library is very powerful in terms of functionality but it loses the game in execution time. It might take some time for some of the inputs to be sanitized using this function

5. Future work that can be implemented to improve this prevention tool

As this library has been designed in PHP, currently it can only be used by PHP developers. Since we now know the cases to be handled for any kind of Reflected XSS and Stored XSS attack, we can extend this library to other server side technologies like Python, ASP.NET, Node.js etc.

Also, currently we have just focused on Reflected XSS and Stored XSS attacks handled at the server side. We can also implement DOM based XSS in the client side technologies like Javascript/JQuery.

REFERENCES:

1. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
2. http://security.stackexchange.com/questions/100769/if-using-htmlentities-or-htmlspecialchars-functions-is-enough-for-block-xss-atta
3. https://wiremask.eu/articles/xss-filter-evasion-cheat-sheet/
4. http://www.sitepoint.com/php-security-cross-site-scripting-attacks-xss/
5. https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
6. https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001)
7. http://stackoverflow.com/questions/46483/htmlentities-vs-htmlspecialchars