Research Article

PXSS: Framework to Prevent Cross Site Scripting Attacks

Sonika* and Yogesh Kumar

#Computer Science and Engineering, Ganga Technical Campus, Surdha, India

Accepted 01 Sept 2014, Available online 01 Oct 2014, Vol.4, No.5 (Oct 2014)

Abstract

Web applications are being developed at faster pace. Fast pace development comes with the risks of failure to protect from various cyber-attacks prevalent today. Input validation based attacks has been the leader since long time. One of the user input validation based attack is Cross Site Scripting attack. Caring for need of developers to cope with requirement to protect against XSS attacks we have proposed a framework called PXSS (Prevent Cross Site Scripting). Our focus mainly over the mitigation of all types of XSS attacks like Persistent and non-persistent attacks. Our framework has been fully evaluated to prove it significance and is available in the form of library code that can be easily integrated with all ASP.NET based web applications. We have also taken care of performance overhead that causes slow loading of web applications.

Keywords: XSS, Input vulnerability, security.

1. Introduction

Web applications have become an important part of our day to day life. Web application have also proved their dominance over standalone applications to provide access to online services like E-commerce, Internet banking, online study tutorials, e-governance, social media, online storage services and many more. As our dependence over web increases so does the interest of intruders attack relays on the injection of a malicious code, in order to compromise the trust relationship between one user and the web application’s site. If the vulnerability is successfully exploited, the malicious user who injected the code may then bypass, for instance, those controls that guarantee the privacy of its users, or even the integrity of the application itself.

One reason for the popularity of XSS vulnerabilities is that developers of web-based applications often have little or no security background. Moreover, business pressure forces these developers to focus on the functionality for the end user and to work under strict time constraints, without the resources (or the knowledge) necessary to perform a thorough security analysis of the applications being developed. The result is that poorly-developed code, riddled with security flaws, is deployed and made accessible to the whole Internet. Currently, XSS attacks are dealt with by fixing the server side vulnerability, which is usually the result of improper input validation routines. While being the obvious course of action, this approach leaves the user completely open to abuse if the vulnerable web site is not willing or able to fix the security issue. For example, this was the case for e-Bay, in which a known XSS vulnerability was not fixed for months. A complementary approach is to protect the user’s environment from XSS attacks. This requires means to discern malicious JavaScript code downloaded from a trusted web site from normal JavaScript code, or techniques to mitigate the impact of cross-site scripting attacks.

Two main classes of XSS attacks exist: stored attacks and reflected attacks. In a stored XSS attack, the malicious JavaScript code is permanently stored on the target server (e.g., in a database, in a message forum, in a guestbook etc.). In a reflected XSS attack, on the other hand, the injected code is reflected off the web server such as in an error message or a search result that may include some or all of the input sent to the server as part of the request. Reflected XSS attacks are delivered to the victims via e-mail messages or links embedded on other web pages. When a user clicks on a malicious link or submits a specially crafted form, the injected code travels to the vulnerable web application and is reflected back to the victim’s browser. The reader is referred to for information on the wide range of possible XSS attacks and the damages the attacker may cause. There is a number of input validations and filtering techniques that web developers can use in order to prevent XSS vulnerabilities. However, these are server-side solutions over which the end user has no control.

Types of XSS

Primarily XSS attacks are categorized into two types namely

1. Stored XSS
2. Reflected XSS

*Corresponding author: Sonika
Reflected XSS

When user inputs accepted through HTTP request and are echoed as HTTP response in HTML for browser rendering. It involves all temporary issues. Stored XSS

Stored or persistent XSS denotes to all such vulnerabilities where intruder is able to permanently inject the malicious JavaScript code in web application. The malicious code is saved in databases or files used to frame runtime HTTP response to user.

XSS attack scenario

Cross site scripting can be triggered to exploit various vulnerabilities in web applications.

- User’s Cookie Theft
- Modifying the DOM of web page
- Statistics Collection
- Exploit Browser Vulnerability
- Misusing Clipboard content
- History and Search queries theft
- Post scanning and Privacy concern

XSS attacks can also be exploited without using <script></script> tags. For example

<body onload=alert('Hello Hackers')

As well as using other attributes like onmouseover, Onerror.

To mitigate the impact of XSS attacks we have developed a .Net library PXSS which mainly includes four components User input validator, Database Sanitizer, URL Sanitizer, Output Sanitizer which is server side solution layer and whose runtime overhead is negligible.

2. Proposed work

Despite being intense research in security testing of web application, there are still issues relate to security in web applications.

User Input Validator

User input validation is one of the main issue for vulnerability towards XSS attack. Properly validating the user input can prevent all of the XSS attacks. We have developed a function in our code library to protect web application against the XSS attack vectors. Our code library is can be easily referenced by any web application developer to validate the user inputs fields against the malicious user input that may cause harm to integrity of web applications. User input validation involves checking each and every user input character to make sure that character entered by user acceptable for the web application use as per web application policies.

Each user input field web application that accept input from anonymous user or registered user must validate the data before saving to server side. Our code library is well acquainted with method to verify each user input field for security of web application.

As an example, the following code snippet shows how to use the user input validation function:

```csharp
public static bool InputSanitize(string userInput)
```

This function accepts the user input as its parameter to validate against the specified list of keywords which must be prevented to reach the server side. It matches each character of input string with the list of character in blacklist specified and return true if none of the blacklisted character is found in the user input string. Single occurrence of even one blacklisted character in user input string will return false and whole of the string is rejected by server and user is presented with the custom error reporting the issue detected with the input field.

We are mainly focused on XSS attack vectors so we are protecting our web application against JavaScript code that is the main source of attack. A list of blacklisted character is as follows:

- '
- ...
- --
- ;,
- /*
- */
- @
- script,
- <
- <javascript>
- javascript,
- <script>,

Figure: 1 PXSS (Prevent Cross Site Scripting) Framework

We have analyzed some of the key fundamentals that are making implementation of secure system infeasible and leading to regular security breaches. We have devised a framework of activities that must be implemented in order to make the web application secure against XSS attacks. Main cause of XSS is user input validation. User input validation is the first step towards securing any application from malicious inputs that may lead to hijack of web applications. Our framework PXSS (Prevent Cross Site Scripting) has 4 major components

1. User input validator
2. Database Sanitizer
3. URL Sanitizer
4. Output Validator
Sonika et al

PXSS: Framework to Prevent Cross Site Scripting Attacks


Database Sanitizer

Database sanitizer is specially designed to protect web application against persistent XSS attack i.e Stored type XSS. Using database sanitizer we can cleanup our database tables to remove all the infected content. Database sanitizer works by checking each and every field of database table against the presence of malicious content and whenever any malicious content is found; entire row of data is removed permanently to prevent the data from being used to craft on the fly user client page and for making further infection when served to users through database access in upcoming user request to database. Due care has been taken to sanitize the database because database may lose some useful content if removed due to some false positive.

We have implemented our database sanitizer also as part of our code library. Database sanitizer can be accessed using the function name DatabaseSanitizer(). Database Sanitizer has following form when presented in library.

public static bool DatabaseSanitizer(string DatabaseKeys)

It is publicly accessible to every developer using our namespace as:

using PreventXSS;

Under the class name PXSS

boolean check=PXSS.DatabaseSanitizer(DatabaseKeys);

Similar to InputSanitize(), it also accept string as the value from database table column data and verity it against set of allowable characters and return Boolean value to denote the presence and absence of malicious content.

For Example- If a user inputs the script(Java script)
from persistent storage like database or flat files is used to generate dynamic web page content. Output validator checks this dynamically generated content against XSS vulnerabilities. Output validator can result in false positive if applied over the complete web page so it is restricted to verify only the web page content that is supplied by user or third party. It does not check the JavaScript code if applied over the complete web page so it is restricted to verify only the web page content that is supplied by user or third party. It does not check the JavaScript code provided by web application owner in static form as part of web application.

3. Results and discussion

With the web applications having a prevalent role in our day to day life. We are all socially connected over social websites like Facebook, Twitter, Google Plus, LinkedIn and many more. Most of our daily activities involve Internet banking, emails, web search, e-governance, jobs, e-health, business, stock trading, auctions, electronic reading, online tutorials etc. All these come equipped with challenges to keep all such daily usage web application active and healthy. But there are some third parties who are actively working to gain financial benefits and trying to put this application down.

XSS is among the top security threat for long time as per various security agencies reports like SANS, MITRE, CVE etc. One of major reason for being it so popular is the simplicity to craft and execute. Any novice user with little knowledge of JavaScript may practice this attack. For such simple attack vectors various techniques has been proposed but are not actively used by developer, either developers are not aware of security issues or management is not spending sufficient funds towards the security and does not have dedicated security team. Lack of security teams is mostly the problem for small scale organization which does not have enough budgets.

Solution Layer

Solution Layer defines the web application tier architecture targeted to mitigate the XSS attacks. It defines the layer the tool designed work actively and implemented. Solution layer could be Client side, server side, database layer in case of 3-tier architecture. Most of the XSS prevention tools have focused over the server side to protect user.

Table 1: Outline of XSS techniques in literature survey

<table>
<thead>
<tr>
<th>Paper Ref.</th>
<th>Tool</th>
<th>Solution Layer</th>
<th>Runtime Overhead</th>
<th>Coverage</th>
<th>Implementation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Noxes</td>
<td>Client Side</td>
<td>N/A</td>
<td>All</td>
<td>Web Proxy</td>
</tr>
<tr>
<td>2.</td>
<td>X.509 Certificates</td>
<td>Server side</td>
<td>N/A</td>
<td>All</td>
<td>Firefox Browser Extension</td>
</tr>
<tr>
<td>3.</td>
<td>Smask</td>
<td>Server side</td>
<td>N/A</td>
<td>PHP,Perl and java</td>
<td>PHP Extension</td>
</tr>
<tr>
<td>5.</td>
<td>ForceHTTPS</td>
<td>Client Side</td>
<td>N/A</td>
<td>All</td>
<td>Firefox Browser Extension</td>
</tr>
<tr>
<td>6.</td>
<td>JaSPin</td>
<td>Client Side</td>
<td>0.8-4.1 s</td>
<td>All</td>
<td>Firefox Browser Extension</td>
</tr>
<tr>
<td>7.</td>
<td>Memento</td>
<td>Server side</td>
<td>28%</td>
<td>Apache Web Server</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>XSSDS</td>
<td>Server side</td>
<td>0</td>
<td>All</td>
<td>Firefox Browser Extension</td>
</tr>
<tr>
<td>10.</td>
<td>BLUEPRINT</td>
<td>Server side</td>
<td>N/A</td>
<td>All</td>
<td>Web Application Encoding</td>
</tr>
<tr>
<td>11.</td>
<td>Masibty</td>
<td>Server side</td>
<td>N/A</td>
<td>All</td>
<td>Reverse Proxy</td>
</tr>
<tr>
<td>12.</td>
<td>Noncespaces</td>
<td>Both</td>
<td>&lt;2%</td>
<td>All</td>
<td>Template at Web server and Proxy for Client side</td>
</tr>
<tr>
<td>15.</td>
<td>SWAP</td>
<td>Server side</td>
<td>&lt;200%</td>
<td>All</td>
<td>Reverse Proxy</td>
</tr>
<tr>
<td>16.</td>
<td>Alhambra</td>
<td>Client Side</td>
<td>200%</td>
<td>All</td>
<td>Browser</td>
</tr>
<tr>
<td>18.</td>
<td>L-WMxD</td>
<td>Webmail Server</td>
<td>N/A</td>
<td>All</td>
<td>Standalone application</td>
</tr>
<tr>
<td>21.</td>
<td>WebShield</td>
<td>Client Side</td>
<td>1-5 s</td>
<td>All</td>
<td>Proxy Based</td>
</tr>
<tr>
<td>22.</td>
<td>FlowFox</td>
<td>Client Side</td>
<td>20%</td>
<td>All</td>
<td>Firefox Browser Extension</td>
</tr>
<tr>
<td>23.</td>
<td>PXSS</td>
<td>Server side</td>
<td>&lt;.08ms</td>
<td>.Net</td>
<td>Library</td>
</tr>
</tbody>
</table>

Figure 4: Classification of tool on basis of Solution Layer

Runtime Overhead

Runtime overhead is the performance overhead caused by the tool designed to protect the application or user from XSS attacks. It is the increase in response time due to extra processing being executed by the tool. Performance overhead causes delay in loading of web applications and rendering by browser. Excess overhead may even let the user to switch to alternate resource of information.

We have carried out our analysis over 13 tools that also prevent against XSS but there has a little attention towards the runtime overhead evaluation by author. Unavailability of runtime overhead information from tool developer raises a question to developer and management team, whether to use the tools or not. Runtime overhead details the efficiency of the tools, which make it the best
choice for developer to use. Some tools have provided the performance overhead ranges from 2% to 200%. Since average overhead is too large. Excess runtime overhead will lead to longer loading time of web application and hence will not attract potential users.

**Principle Technique**

Principle technique is the core basis of how the XSS attack mitigation is carried out by the tool so developed. It defines the internal structure and logic behind the working of the tool. Some of the prevention schemes used is sandboxing, server side input filters, cryptography based mechanism, secure coding, strong typing, instruction set randomization etc.

**Implementation Type**

Implementation type defines the details about how the whole tool has been implemented to counter attack the XSS attacks. It involves various type like Proxy based software, Browser extension to force the policies against XSS, standalone applications to test web application before going live, source code based application of input filters in web application during the web development life cycle, configuring the web server to mitigate the XSS attack vectors.

How the tools have been implemented is important from developer point of view for security enforcement. As shown in Table 1, most of the tools are implemented as browser extension which put the security an issue to client shoulder instead of developer. Developer either needs to force user to use particular browser extension for better and safe browsing or let it to user's awareness about security. Binding web applications to browser specific itself is birth to new paradigm of problem. A few others have developed Reverse proxy based tools which also requires awareness from client side to make the browsing secure. 4 out of 13 tools have server side implementation of security.

![Classification of tool on basis of Implementation Type](image)

**Conclusion and future scope**

We have proposed a framework to prevent XSS attacks over web applications. Our framework has separate components to defend against each type of XSS related security issue. PXSS (Prevent Cross Site Scripting) system prevents reflected XSS, Stored XSS as well as Cookie related problems due to XSS attacks. Our framework has been implemented as code library for .NET based web applications. Our code library can be easily integrated with web application to utilize its fruitfulness to protect against XSS. We have also evaluated significance of our framework against some of real world web applications. It provides total coverage of all types of XSS related problems with negligible runtime overhead. It will be available for open access to all developer to integrate with web application for security enforcement.

In future we will be adding the support for other web technology and also working to developer a solution that does not require technology expertise to integrate and with easy to use user interface.

**References**


Ter Louw, Mike, and V. N. Venkatakrishnan (2009.) Blueprint: Robust prevention of cross-site scripting attacks for existing browsers, *Security and Privacy, 30th IEEE Symposium on. IEEE.*


Van Gundy, Matthew, and Hao Chen (2009.) Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks. *NDSS.*


