A Survey on Cross Site Scripting Attack Detection

Mr. Sudhir. S. Dhekane
Email:sudhirdhekane1987@gmail.com
Computer dept, Terna Engineering College
Navi Mumbai, India

Prof. V. B. Gaikwad
Email: vb2k@rediffmail.com
Computer dept, Terna Engineering College
Navi Mumbai, India

Abstract

In the today's world most of the electronic transactions, web services run via website with stateful reliable communication by transmitting, storing, retrieving essential data on network, client and webserver. The major problem with such reliable communication is that they become vulnerable to an XSS type of attack which mostly carried out to steal essential and confidential information, gain control of stateful communication, to change browser settings and cause fraudulent activities for financial gain.

This paper introducing basic types of XSS attack, the survey of the XSS detection systems to find out advantages and disadvantages in existing XSS detection systems and, understanding their importance with respect to security for web applications.

INTRODUCTION

This paper is organized in the following way. In the first section we discussed basics of the cross site script attack (XSS). In the second section, we studied available systems designed and implemented for XSS attack detection with their advantages and disadvantages. In third section we have conclude the paper with discussion of the available systems and methods of XSS detection.

Keywords: Cross site script (XSS)

What is XSS?

Cross-Site Scripting (XSS) attacks occur when:

1. Data input in Web application through a non trusted source, mainly a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious script.

The malicious content sent to the web browser is a piece of JavaScript, but it may also include HTML or any other type of code that the browser may able to execute. The variety of attacks based on XSS is very vast, but commonly they include transmitting confidential data like cookies or other essential session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the appearance of the vulnerable site.

Classification of XSS Attacks

It is too difficult to categorize XSS attacks. Generally they are categorized into two categories, stored and reflected. There is a third, much less well known type of XSS attack called DOM Based XSS.

Stored XSS Attacks:
These are the attacks where the injected script is permanently stored on the targeted servers, such as in a database, in a message forum, visitor log, comment field, etc. The injured party then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes called Persistent XSS or Type-I XSS. The Figure 1 showing how this type of attack is carried out.

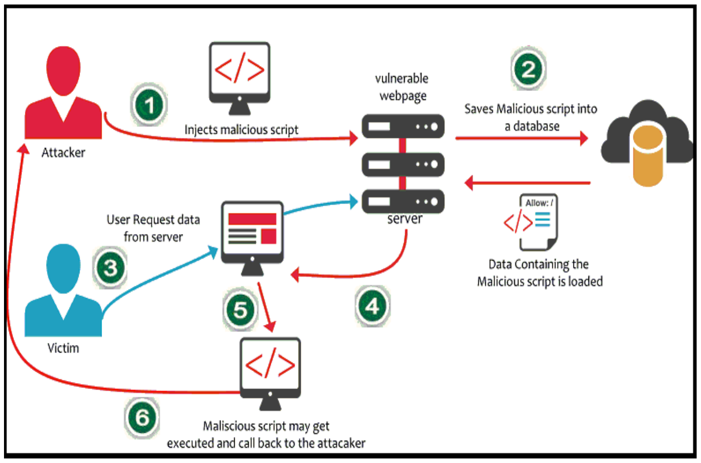
Attacker
Injects malicious script
vulnerable webpage
server
Saves Malicious script into a database
User Request data from server
Victim
Allow: /
Data Containing the Malicious script is loaded
Maliscious script may get executed and call back to the attacaker

Figure 1: Working of Stored XSS Attack [8]

Reflected XSS Attacks

These are the attacks where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another path, such as in an e-mail message, or on some other web site. When a user will attempt click on a malicious link, submitting a specially crafted form, or even just browsing to a malicious web site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code because it assumes that came from a reliable server. Reflected XSS is also sometimes known as Non-Persistent or Type-II XSS. The Figure 2 showing how this type of attack is carried out.

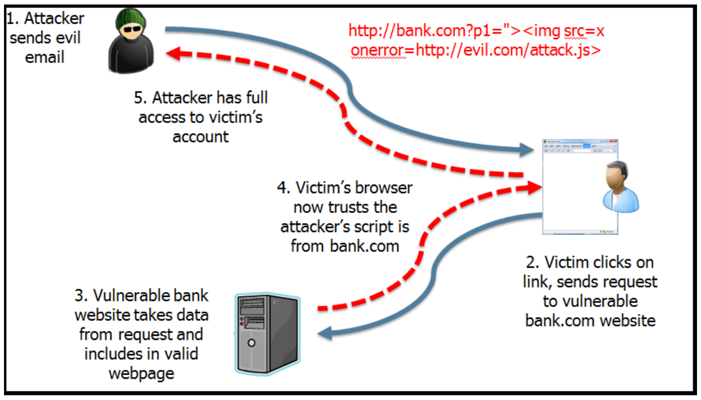
1. Attacker sends evil email
http://bank.com?p1="><img src=x onerror=http://evil.com/attack.js>
5. Attacker has full access to victim's account
4. Victim's browser now trusts the attacker's script is from bank.com
2. Victim clicks on link, sends request to vulnerable bank.com website
3. Vulnerable bank website takes data from request and includes in valid webpage

Figure 2: Working of Reflected XSS Attack [9]

XSS Attack Consequences

- The end result of an XSS attack is the same apart from whether it is stored (Persistent), reflected (Non-Persistent). The difference is in how the payload arrives at the server.
- The most severe XSS attacks involve leak of the user's session cookie, which allows an attacker to take control on user's session and get hold of the account.
- Other injurious attacks include the disclosure of end user documentations, installation of Trojan horse programs, redirect the user to some other page or site, or modify presentation of content.
- An XSS vulnerability allow an attacker to modify a press release or news item could affect a company's stock price or lessen consumer confidence.

THE PRESENT XSS DETECTION SYSTEMS

A Proposal and Implementation of XSS Automatic Detection/Collection System By, Omar ISMAIL, Masashi ETOH, Youki KADOBAYASHI, Suguru YAMAGUCHI [2]

Author proposed a client-side system that automatically detects an XSS vulnerability by manipulating either request or server response. The system also shares the indication of vulnerability via a central repository. The purpose of the proposed system is dual:
To protect users from XSS attacks
To inform the web servers with XSS vulnerabilities.
As the detection part, author has implemented two different detection mechanisms, response change mode and the response change mode.
Response Change Mode:
Fig. 3 illustrates a series of steps taken to accomplish the detection and collection procedures in response change

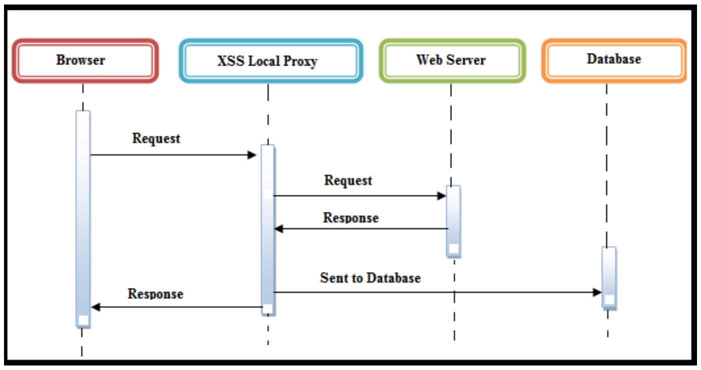

Figure 3: Response Change Mode

1. Request Check
The proxy checks whether its parameters include special characters. If there are, the detection/collection system will save a copy of the request in the proxy side and forward the

original request. Otherwise the system just forwards the request or response between the clients and servers.

2. Response Check

Followed by sending the request, the server generates its response. If the request is detected of containing the special characters, the detection/collection proxy compares the response message with the corresponding request message stored in the proxy server to see whether the same special characters are still included in the response message. If no special characters are found, the detection/collection proxy servers simply forward the response to the client. Otherwise, the system marks the server as XSS vulnerable and sends the alert messages to the client. Meanwhile, the escape encoded response message will be sent to the client.

Request Change Mode:

Fig.4 illustrates a series of steps taken to accomplish the detection and collection procedures in request change mode. Every step is explained below.

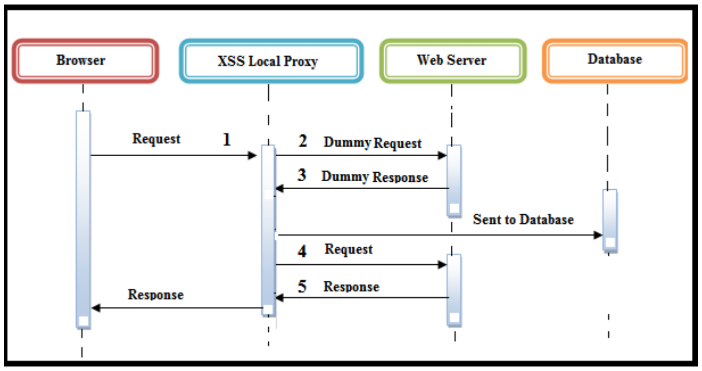

Figure 4: Request Change Mode

1. Request Check

Check whether the request message containing special characters.

2. Sending Dummy Request

If the request message contains special character, the detection/collection server will save the copy of original request message and then to differ parameters in request message, random generated numbers are inserted to every parameter for identification purpose before sending to the requested web server.

3. Dummy Response Check

At this stage, the system investigates the server generated response message to see whether the Web server is XSS vulnerable. If the Web server is found vulnerable, the information about the Web server will be send to the database.

4. Sending the Request

If the web server is XSS vulnerable, the special characters in original request are escape encoded before sending to the web server. Otherwise, the detection/ collection system simply forwards the original request to the server

5. Response Check

Alert the user by embedding the alert HTML message in the response page.

The Information Collection for XSS Vulnerability

Fig. 5 presents the system overview of the Automatic Detection/ Collection system for XSS vulnerability. After the proxy server detects vulnerabilities, it sends those collected information such as host names, the parameter name, the path name etc. to the collection database server and such that the collected information can be shared between the proxy servers.

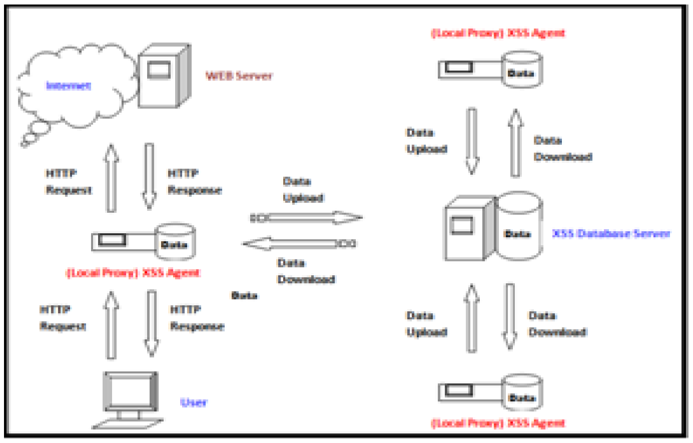

Figure 5: Automatic detection/collection systems for XSS

Advantages:

Their approach is an effective way to detect and collect XSS vulnerabilities. In this paper, author has presented a user-side proxy approach for automatically detecting and collecting Cross-Site Scripting Vulnerability. Two different detection modes, the response change mode and the request change mode, are discussed and evaluated with real-world examples respectively.

Disadvantages:

There are many challenges to be addressed, especially, utilization of the collected XSS information in the central database, and to make the system deployment universal.

Cross Site Scripting Attacks Detection Algorithm Based on the Appearance Position of Characters By, Takeshi Matsudat, Daiki Koizumi, Michio Sonoda [3]

In this paper, the Author's proposed detection algorithm against cross site scripting attacks by extracting an attack feature of cross site scripting attacks by considering the

appearance position and frequency of symbols. Their proposed algorithm learns the attack features from given attack samples. They prepared, and find the samples for learning and testing from various websites and books, to show the effectiveness of their proposed XSS detection algorithm.
Their work includes following tasks:Word Extraction Algorithm
Detection Algorithm of Cross Site Scripting
Classification Rule:Calculation of important Degree of Symbols
Detection of Cross Site Scripting
Word Extraction Algorithm:

Word extraction algorithm expands the search query when users use web search engines. This algorithm pays concentration to the distance between sentences to look for related word of key search query. In this algorithm, a word nearby key word is treated as important word.

Detection Algorithm of Cross Site Scripting:
Classification Rule:

Here there focus is on the characters which are included in cross site scripting attacks.

Calculation of important Degree of Symbols:

Here, they defined the calculation method for calculation of the important degree of characters.

Detection of Cross Site Scripting:

Here author's calculated the attack feature vector value and the threshold of all possible symbols occurred in a script to determine the XSS script.

Consequences:

As the result, the proposed detection method of author's was successfully detected 99.5% attack test samples and 97.5% normal test samples

Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities by Jayamsakthi Shanmugam, M.Ponnavaikko [4]

Author's research aims to use the positive security model to reduce the processing time and by introducing the application level attributes. Their proposed solution comprises of four components namely analyzer, parser, verifier and white listed tag cluster and the interactions between them.
Proposed Solution Procedure:
At an application level they have defined the following attributes in table 1:

Severity Level	Character Set	Encoding	Maximum No. Of Characters
High	ISO-8895-1	Nil	20
Medium	UTF-8	Yes	3000
Low	ISO-8895-1	Yes	10000

Table 1: Application Level Attributes [4]
Following components and their interaction shown in figure addressing the XSS vulnerability at server side the components are:

1. Analyzer
2. Parser
3. Verifier
4. Tag Cluster (White Listed Clusters)

The following definitions are made to define the tags with respect to the group of tag clusters and are used to form the rules to identify the vulnerability.

Let I= {I1, I2, I3… In} be a finite set of tags in the input. Let W = {W1, W2, W3… Wn} be the finite set of white listed tags. {MS1, MS2, MS3… MSn} be the corresponding set of security classes for the tag Wi to identify the attribute or the value of the tag content to determine whether the input provided is malicious.

Rules to conclude an input as untainted input is defined as follows:

If Ii is not as per the application level parameters set. Ii is untainted, only if it is a subset of { W1, W2, W3…Wn}where Ii is the tag in the input and if security classes identify the attribute's value as untainted.
Figure 6 describes the flow of the system. The execution sequence is numbered in the above diagram for better understanding of the process. Analyzer reads application level parameters first, and checks whether the input meets the maximum character rule and encoding rule. Then the input is checked for the special character existence in the input and if it exists then it forwards the request to the parser. The parser splits input to tokens and sends it to the verifier. The verifier accesses the white listed cluster and checks for its vulnerability. If there is no vulnerability detected then the verifier returns the status to parser. The parser then returns the status to analyzer.

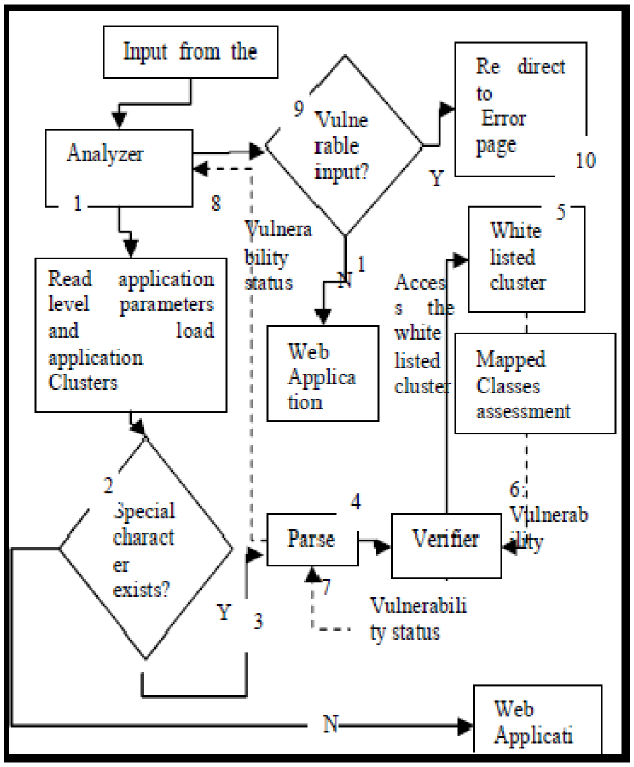

Figure 6: Flow of input through the Web Application [4]

Based on the status returned, analyzer either redirects the request to the error page or forwards the request to the web application.

Result Analysis:

Around 2500 lines of code has been developed by author's and also 108 unique XSS test cases are created to test this approach. This approach is also tested in about 2000 vulnerable input data collected from various research sites and in the white hat hackers' site where the proof of code is provided for XSS vulnerability. These web pages with vulnerable input are categorized based on the severity level parameters defined above. Out of 2000 XSS vulnerable pages found, around 160 web sites are SSL protected banking applications.

In this the observation is that, there is an increase in the processing time to process a single vulnerable input request from 0.29 to 0.33 milliseconds after the implementation of the security mechanisms, which is a 0.04 millisecond increase per request, which is a very minor increase in the processing time. To process a non-vulnerable input, on an average the proposed system takes .006 milliseconds higher than the system without the security mechanisms implemented as the application parameters are introduced.

The following are the advantages of this approach:

1. This approach allows tags to be entered in the web application and at the same time provide security for the web application.
2. The research work uses the positive security model to reduce the processing time. In the negative security model, the processing time of the server increases for every new threat introduced, since the input should be matched with the larger number of signatures as the XSS attack surface is very high. In the authors approach, the attack surface is minimized using the positive security model.

The following are the Disadvantages of this approach:

1. This approach needs an updation in the white listed cluster XML data, when a new tag needs to be permitted. As of now the Behavior-based anomaly detection on the server side approach does not address all the encoding patterns,

Optimized Client Side Solution for Cross Site Scripting By Siddharth Tiwari, Richa Bansal, Divya Bansal [5]

The solutions on server side result in considerable degradation of web application and are often unreliable, whereas the client side solutions result in a poor web browsing experience, there is need of an efficient client side solution which does not degrade the performance. The proposed system by Autho's has designed in order to provide effective security against the Cross Site Scripting attack, keeping the concept of usable security with optimized web browsing. This approach uses a three step process, described in Fig 7.

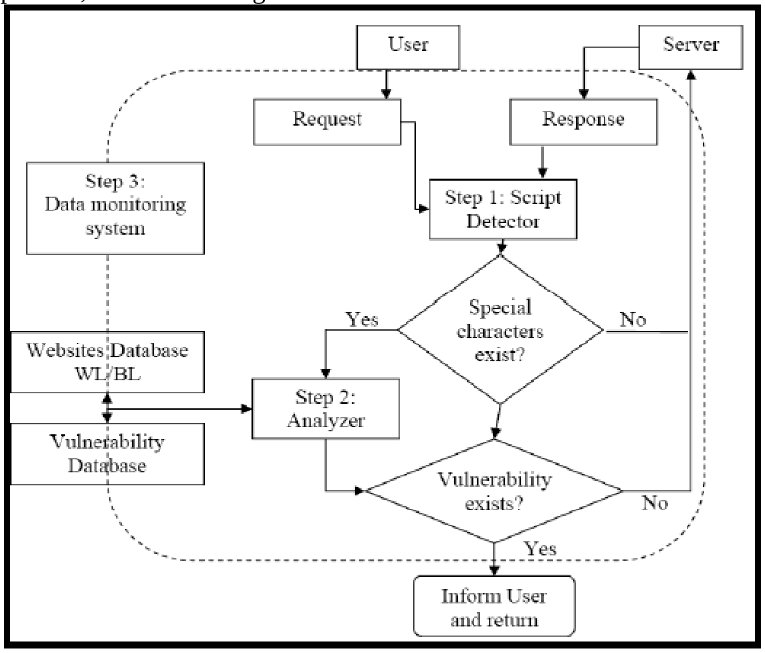

Figure 7: A three step process to detect XSS[5]

The first step is to check for scripts tags in the input. When the HTTP request is received, it is passed through the script detector. It reads the application level parameters and applies the rules on the input. First, it checks for the maximum number of characters, and if the input exceeds the number of characters, then the input is rejected without processing the input further. The second condition checked by the analyzer is the existence of special characters. This is because the scripts can only be executed when it is embedded using the tags and special characters. If special character exists in the input, then the input is passed to the parser. Otherwise the request is forwarded to the web application.

The second step is performed by an analyzer which uses both these databases to detect vulnerability, and decision is made by user.

The third step is above the whole system, which is performed by a data monitoring system. The flow of data is passively monitored by the system. The operations processing sensitive information are marked along with the results of those operations. If the marked data is about to be transferred over the network, user is asked to allow or disallow the transfer based on the information in the dialogue box provided.

Implementation and Result Analysis:

This solution was implemented using open source Mozilla Firefox 1.5 web browser from Mozilla foundation

Security Evaluation:

The proposed solution has been tested with thousands of malicious inputs, non vulnerable input with white listed tags and vulnerable websites. They compared the proposed browser with Firefox without security implemented, Microsoft's Internet Explorer, Apple's Safari Web Browser and other available web browsers on the same platform and environment. It has been observed that there are more than 100 variants of XSS attacks exist and the approach is tested with the data collected from various research sites, white hat and black hat sites.

Firefox2:14%
Safari 2:17%
Implemented web Browser: 5%
Opera: 20%
IE 6 with SP2:44%

Performance Evaluation:

It is important that after the application of security by the proposed model, the user's web browsing experience is not affected seriously. The proposed browser's performance was compared with several available browsers. The performance has been observed by logging the time of processing. The approach is tested on 2.0 GHz Intel Core2duo machine, with 1 GB RAM. Each browser's speed response was logged by putting them through a number of tests. To get unbiased results, it is important that the internet connection speed should be uniform during the experimentation. The page load time can be calculated by writing a small script on a locally hosted webpage, or freely available website load time and speed checker.

Each test was done with a default browser install, without changing any settings.

SWAP: Mitigating XSS Attacks using a Reverse Proxy By, Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, and Christopher Kruege [6]

In this paper, Author's introduce SWAP (Secure Web Application Proxy), a server-side solution for detecting and preventing cross-site scripting attacks. SWAP comprises a reverse proxy that intercepts all HTML responses, as well as a modified Web browser which is utilized to detect script content. SWAP can be deployed transparently for the client, and requires only a simple automated transformation of the original Web application. Using SWAP, they were able to correctly detect exploits on several authentic vulnerabilities in popular Web applications.

Working of SWAP:

SWAP operates on a reverse proxy, which relays all traffic between the Web server that should be protected and its visitors as shown in below Figure 8. The proxy forwards each Web response, before sending it back to the client browser, to a JavaScript detection component, in order to identify embedded JavaScript content. In the JavaScript detection component, SWAP puts to work a fully functional, modified Web browser, that notifies the proxy of whether any scripts are contained in the inspected content.

In order to differentiate between benign and malicious JavaScript, previously to enabling the proxy with the JavaScript detection component, the hosted Web application is modified. All legitimate script calls in the original Web application are encoded into unparsable identifiers, so
called script IDs, and thus, hidden from the JavaScript detection component. Consequently, it is safe to assume that each script that is still found must have been injected, either via the preceding Web request (reflected XSS), or via the Web application's database (stored XSS).

If no scripts are found, the proxy decodes all script IDs, effectively restoring all legitimate scripts, and delivers the response to the client. If the JavaScript detection component, on the other hand, detects a script, SWAP refrains from delivering the response, but instead notifies the client of the attempted XSS attack.

The main components of SWAP are:

1. A JavaScript detection component, which, given the Web server's response, is capable of determining whether script content is present or not.
2. A reverse proxy installed in front of the Web server, which intercepts all HTML responses from the server and subjects them to analysis by the JavaScript detection component.

3. A set of scripts to automatically encode/decode scripts/script IDs.

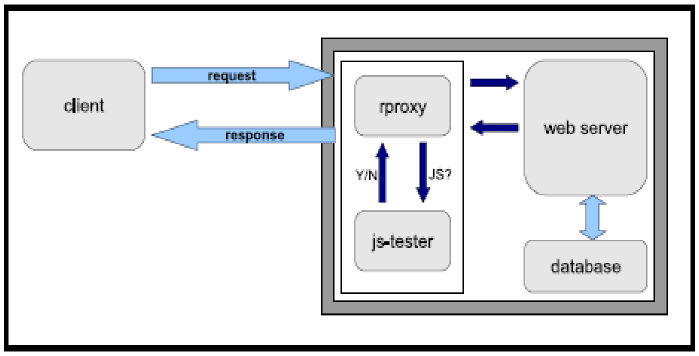

Figure 8: Working of SWAP [6]

Performance Evaluation:

Due to the additional requirements for processing power introduced by SWAP, clearly, a performance detriment is introduced, meaning that the client will experience higher latency when requesting content from a SWAP protected Web server, as compared to a server that does not feature SWAP protection. SWAP adds to the latency two-fold: First, by putting an additional stepping stone between client and server, namely the reverse proxy, all traffic is relayed instead of a direct transmission, and thus, takes longer to arrive at its target. Second, and more importantly, the JavaScript detection component effectively has to render each page before it can be delivered to the client. We have conducted experiments to measure the magnitude of the performance penalty inflicted by our SWAP prototype implementation.

Advantages:
1. The proposed approach works well to detect and alert about XSS malicious script
2. The JS detection component is able to distinguish between benign and malicious scripts.

Disadvantages:
1. SWAP introduces performance overhead
2. SWAP is experimentally not suitable for high-performance web services
3. In this proposed and implemented technique the processing speed is not been considered which is a important factor necessary to be consider

Injecting Comments to Detect JavaScript Code Injection Attacks By, Hossain Shahriar and Mohammad Zulkernine[7]

In this review, Author's addresses the issues like an injection of malicious scripts by third parties in the form of overriding the available methods but with different implementations, injection of bad inline functions, injection of additional methods which may be malicious or not by developing a server side JavaScript code injection detection approach. They pre and postpend each legitimate JavaScript code block with comment statements that include identical random token then they identify the expected features of a JavaScript code block (*e.g.*, method call, method definition), save the features in policies, and embed the policy information into comments. During the deployment phase, they performed a number of checks to detect injection attacks. These include (i) code without comment, (ii) code with correct and duplicate comment, and (iii) code with correct and non-duplicate comment; however, the actual code features are not matching with the intended features specified in a policy.

They apply the proposed approach for server side programs implemented in Java Server Pages (JSP). They developed a prototype tool in Java to inject JavaScript comments and generate policies based on legitimate code features then they had deployed the injected code detector as a server side filter.

Evaluation:

They evaluated their approach with three real world JSP programs.

Advantages:

-Their evaluation indicates that the proposed approach can mitigate many types of injected JavaScript code that might contain arbitrary and legitimate method call injection, and method definition overriding.

-The result showed that the approach suffers from zero false negative rates.

CONCLUSION

The first approach is an effective way to detect and collect XSS vulnerabilities by two different detection modes, the response change mode and the request change mode [2]. The Second approach detected successfully 99.5% attack for real world test samples and 97.5% normal test samples [3]. The third approach comprises of four components namely analyzer, parser, verifier and white listed tag cluster and the interactions between them and the observation is that, there is an increase in the processing time to process a single vulnerable input request from 0.29 to 0.33 milliseconds after the implementation of the security [4]. The fourth approach present mechanisms that tested with thousands of malicious inputs, non vulnerable input with white listed tags and vulnerable websites and compared proposed mechanism with other types of browsers[5].In fifth approach SWAP (Secure Web Application Proxy) the mechanism developed to server-side solution for detecting and preventing cross-site scripting attacks in which client will experience higher latency when requesting content from a SWAP protected Web server, as compared to a server that does not feature SWAP protection, and the last approach inject comment for valied blocks in response page and by extracting policies the comparison is done to detect any malicious injection of a code.

References:
[1] OWASP: Cross-site Scripting (XSS) Last revision (mm/dd/yy): 09/13/2013
[2] Omar ISMAIL, Masashi ETOH, Youki KADOBAYASHI, Suguru YAMAGUCHI,"A Proposal and Implementation of XSS Automatic Detection/Collection System",Proceedings of the 18th IEEE International Conference on Advanced Information Networking and, Application, 2004
[3] Takeshi Matsudat, Daiki Koizumi, Michio Sonoda, "Cross Site Scripting Attacks Detection Algorithm Based on the Appearance Position of Characters" The 5th International Conference on Communications, Computers and Applications (MIC-CCA2012); Istanbul, Turkey: 12-14 October 2012
[4] Jayamsakthi Shanmugam, M.Ponnavaikko,"Behavior-based anomaly detection on the server side to reduce the effectiveness of Cross Site Scripting vulnerabilities", Third International Conference on Semantics, Knowledge and Grid, IEEE 2007
[5] Siddharth Tiwari, Richa Bansal, Divya Bansal Optimized Client Side Solution for Cross Site Scripting, 16th IEEE Conference On Degital Object Identifier, 2008
[6] Peter Wurzinger, Christian Platzer, Christian Ludl, Engin Kirda, and Christopher Kruege " SWAP: Mitigating XSS Attacks using a Reverse Proxy", IEE May 19, 2009, Vancouver, Canada
[7] Hossain Shahriar and Mohammad Zulkernine, "Injecting Comments to Detect JavaScript Code Injection Attacks", 35th IEEE Annual Computer Software and Applications Conference Workshops, 2011.
[8] Blind XSS:http://www.acunetix.com accessed on date: 11,Oct.2013
[9] Reflected XSS Attackstaging.arstechnica.com accessed on date: 14,Oct.2013