Investigation of Defences Against Cross Site Scripting Attacks

Chris Mills
BE Undergraduate, Software Engineering
University of Auckland

October 26, 2004

Abstract

Cross-site scripting attacks are used to run scripts on users computers, utilizing a poorly coded web server or application. These attacks can be used to run malicious code on or steal personal information from the users computer.

There are different solutions to the problem, based in to two broad categories; server and client based defences. On each there are multiple solutions to the problem with their own advantages and disadvantages.

This paper gives a comparison of each solution and then compares the two categories. The client-side defences are preferred because of the disassociation from the web servers and therefore minimal reliance on a web servers defences.

1 Introduction

Web applications provide rich multimedia and enable user configuration through dynamic web pages that can make web sites highly usable. Unfortunately, these types of applications can be susceptible to attack because of a lack of quality programming during development [1]. Attackers are able to embed their own scripts within dynamic content on unsecure web pages that can be used against users, including the revealing of private information contained in cookies. These types of attacks are known as code injection or cross-site scripting (XSS) attacks.

The potential dangers of dynamic content and these kinds of attacks have been known for many years [2]. This sort of attack has, on occasion, been successfully defended against, especially in on-line message boards [3]. A server-side solution that has been recommended and implemented before is HTML filtering and encoding [2]. Client side solutions have been generally over-looked and most attention has focused on changing a users habits [1].

Potential solutions for web servers are; high-level abstractions of security from the web service, HTML encoding and filtering, and validating proper HTML grammar. Potential client-side solutions include the request change and response change solutions and user education in 'safe' browsing. These solutions all incur a certain amount of time overhead, apart from the extreme example of disabling all scripts from executing on a users browser, which instead limits the usability of a web application.

This report attempts to provide a comparison of the current solutions to the problem of XSS attacks. This includes a broad high level overview of server and client side solutions.

The next section (Section 2) provides a background and examples of cross-site scripting. Section 3 covers descriptions and examples of server and client side protection against those attacks respectively. Section 4 provides a comparison and discussion of the solutions. Section 5 concludes the article.

2 XSS Overview

A XSS attack relies on the web browser and its allowed functionality. In HTML some characters are treated specially and they can affect the formatting or can introduce programs, or scripts, to the page. These characters and tags are listed below [3]:

<SCRIPT> Adds a script to the document.

<OBJECT> Places an object (such as an applet, media file, e.t.c) in a document.

<APPLET> Used to place a Java applet on a document. This is deprecated in the HTML 4.0 specification.

<EMBED> Embeds an object in to the document. This tag was dropped in favour of the <object> tag in the HTML 4.0 specification.

<FORM> Indicates the beginning and end of a form.

The tags listed above are the ones most likely to be used in a XSS attack [3], especially the first four. The <FORM> tag could be used to modify existing data within the form, thereby tricking users in to giving away their personal information. Another form of attack not listed is in-line scripting elements in JavaScript, for example:

```
JavaScript:alert('executing script')
```

As mentioned briefly in the introduction, bulletin boards have been exposed to these attacks for several years. In the past, bulletin boards allowed messages with HTML tags to be inserted. This allowed users to insert different coloured and formatted text. For example, the following code would insert bold face text in to a document; "`<b>text</b>`" formatted in HTML is the same as "**text**" [4]. This same technique, used in conjunction with the above tags can give an attacker an opportunity to run malicious code on a user's computer. This method forms the basis of a XSS attack.

XSS attacks use a similar technique to the one described above, but instead of formatting text it runs scripts using the permissions of associated web servers. By using a poorly coded web application, an attacker is able to insert code in to links or search fields that include malicious code. As the code is submitted as a legitimate link or entry, the server may end up delivering the code back to the client browser, where it runs under the same privileges as the originating server. Below is an example of such a link:

```
<A HREF="http://trusted.org/search.cgi?criteria=<SCRIPT
SRC='http://evil.org/badkarma.js'></SCRIPT>Go to trusted
web site.</A>
```
[3]

These types of attacks can be used to run programs on the client or the web server (if given sufficient privileges), but the attack with the most potential

to do damage is the harvesting of cookie information. Cookies are simple files that contain information about a user's interaction with a website. Because a website generally does not have enough space to store all the information about its users in a database, cookies can solve the problem by installing user-names and passwords on the user's computer.

This practice allows an attacker to access these cookies using a XSS attack. For example, by investigating a site and using social engineering, an attacker can insert a malicious link, containing cookie stealing scripts, in to an email and then send it to a large mailing list. If the email is plausible enough or, in other words, the recipient believes it is a valid email from the originating server, she may click on a link and activate those malicious scripts. Worse, if the email client is not protected, these links can be embedded in images or frames that are executed when loaded. The cookies gathered by the user can be used to authenticate the attacker as a user, fooling the server.

Some common examples of XSS attacks follow [3]:

In-line Scripting The malicious code is passed as a parameter

e.g. http://trusted.org/search.cgi?criteria=<script>code</script>

Forced error response This attack uses the error handling of the associated server. If a link that causes an error goes to the default error page (404), then scripts that are included with the link may cause the script to run because the error page is unprotected (i.e. allows the scripts to run). This vulnerability is found on earlier servers which can be patched with updates.

e.g. http://trusted.org/<script>code</script>

JavaScript entities The special character '&' is sometimes interpreted as a new JavaScript code segment (entity).

e.g. <img src="&(alert('CSS Vulnerable'));">

Figure 1 shows a XSS scripting attack in motion. A user browses to a web site or is shown an email message in HTML that contains a malicious link to a trusted site. If the user clicks this link, the trusted server is sent a request containing the malicious code. If the server does not check for XSS attacks,

the script that is sent will be delivered to the user with the malicious link still attached. Once the page is loaded on the user's machine the script will then execute and run with the same permissions as that of the trusted server.

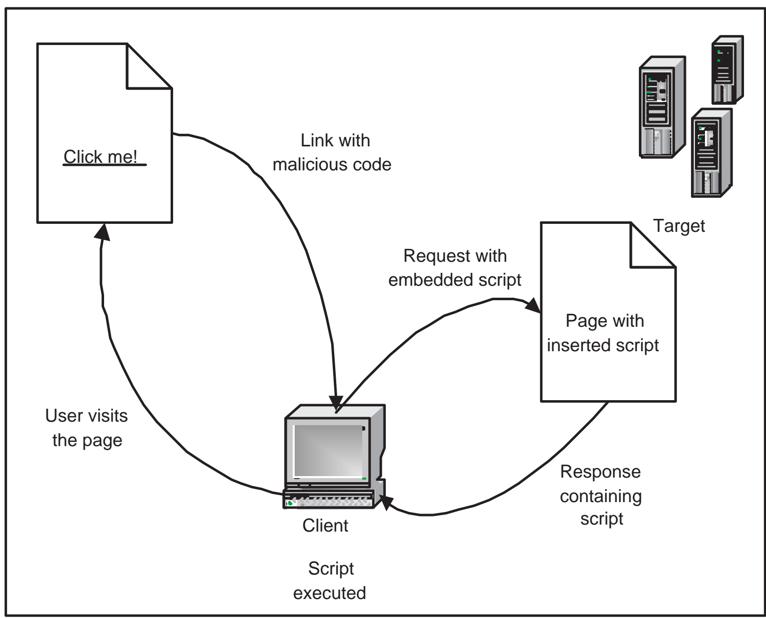

Figure 1: Example of an XSS Attack [5]

3 Solutions

The primary solutions to a XSS scripting attack are either server or client based. With a server based solution, the responsibility protecting users is up to the web developers. On the client side, it is up to the web application (i.e. browsers) developers or another application (e.g. validation proxy) to provide protection.

3.1 Server-side

XSS attacks only affect dynamic pages. Static pages are immune because they are fixed and do not rely on scripted programs to create dynamic user output. However, this limits the users interaction with a web site. The below solutions cover protection for dynamic web applications.

3.1.1 HTML Encoding and Filtering

The simplest way of protecting a client, and the web server, is to encode and filter user input. This involves filtering through HTML and replacing special characters, as mentioned above (see Section 2). This would mean, following on from the previous examples that the HTML, "`<s>test</s>`", would now appear as "<s>test</s>".

By replacing the following characters with the alternatives, according to CERT/CC recommendations, this is achieved [4]:

- '<' to <
- '>' to >
- '&' to &
- ' " ' to "

Disadvantages of this solution include complexity and reduced performance. The complexity correctly changing HTML means that errors may occur, such as incorrectly formatted HTML documents. This method also reduces performance because it is necessary to filter the HTML input for special characters and then change them to the above translations.

Another disadvantage is that because most web servers in use are closed source, developers are unable to take advantage of these filtering techniques. Almost 80% of e-commerce sites in Japan are susceptible to XSS attacks because the application software does not allow a third party to patch security holes [4].

3.1.2 Security Assessment Strategies

A security assessment is an in-depth view of an applications code, searching for bugs that may allow attacks on the application. The assessment includes the identification of bugs and the insertion of runtime checks.

One such application is WebSSARI (Web application Security by Static Analysis and Runtime Inspection) [6]. This program statically checks code on PHP pages, PHP being a common web scripting language and WebSSARI's default type to check. The application also inserts runtime checks to secure potentially insecure pieces of code.

An advantage of this strategy, using WebbSSARI as an example, is it has less overhead then when running assessments at runtime. WebSSARI has shown that it can reduce the potential runtime by 98.4% [6]. WebSSARI has also been used to check 230 web applications on SourceForge.net. Of the 230 tested, 69 projects were shown to have major security vulnerabilities (XSS and others) [6]. This proves the capability of the tool.

The disadvantages with assessment tools are they do not detect new attacks until they have been specifically identified. For instance, if a new XSS attack was attempted that was not covered in the assessment tool, then the tool would not pick it up. Apart from WebSSARI, another disadvantage is that most assessment tools do not have runtime checks inserted.

3.1.3 Abstraction of security concerns

Abstracting security concerns from the web application was introduced by Scott and Sharp [7]. It involves removing security responsibility from the developers and instead provides a set of rules and allowed interactions with different Uniform Resource Locaters (URLs). In Scott and Sharps paper [7], they review an abstraction of a security gateway from the web application and its database.

Figure 2 shows the essential parts of the system. The security policy description language (SPDL) specification is a set of validation constraints and transformation rules for incoming data. The transformation rules are important for defending against XSS attacks. The security policy compiler outputs code based on the SPDL specifications that enforce the validations and transformation. The security gateway processes input based on the compiled code.

The validation and transformation rules are simple rules that define; URL's and their associated parameters including the allowed values, and validations (which include transformations) that allow complex constraints to be placed on input.

The advantage of this approach is that the security responsibilities are removed from web developers. The SPDL allows for tight constraints to be placed on a web servers input. The major advantage for XSS defence is the transformation of HTML so that it is properly encoded.

The disadvantage of this approach is the initial cost, configuration and performance degradation. The cost of implementing this solution involves a pur-

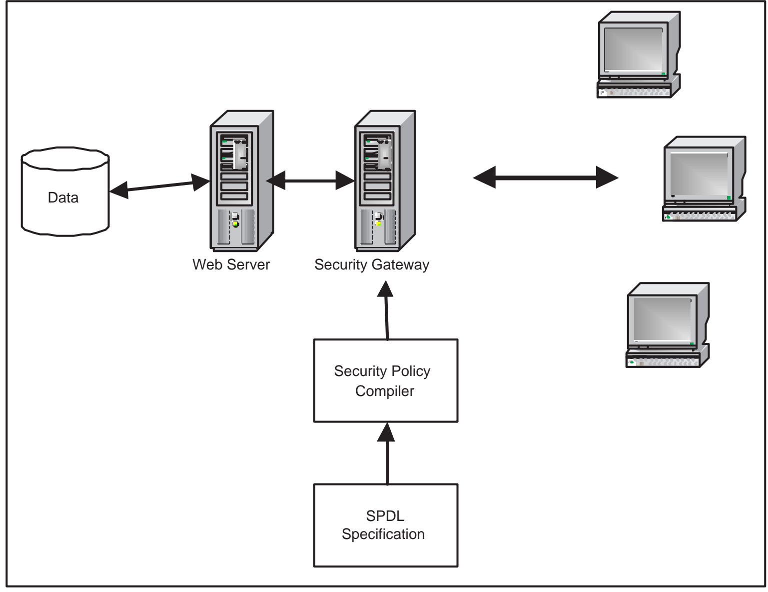

Figure 2: Overview of Scott and Sharps System

chase of a security gateway (though this can be saved by moving the logic to the web server) and man hours required to configure the SPDL. The configuration is intensive because for each URL a related configuration script needs to be produced. With test runs of the system, the latency of the system was high, approximately 10 times slower than a HTTP-proxy [7]. The author's state that this can be improved, however, as their development effort went in to getting a working application and not in to performance.

3.2 Client-side

There are few client-side systems that check for the possibility of XSS attacks. The majority of defence relies in the education of the users; documenting correct browsing procedures, listing vulnerabilities and reiterating common attacks that attackers may use (i.e. emails from seemingly legitimate sources such as banks). This approach relies on clients being well informed and constantly vigilant. This is not always the case, and generally users are not concerned with such precautions. This section introduces a few different methods of approach in defending users against XSS attacks.

3.2.1 Disabling execution of scripts in browser

The easiest solution to prevent XSS attacks is to disable the execution of scripts within the browser. This can be disabled in most popular browsers and prevents any scripts running, avoiding XSS attacks.

The advantage of this is clear; no scripts will run on the client side preventing any XSS attack from succeeding. The disadvantage is that this will disable any user enhancement features within a web page. This is cumbersome and some web pages, Gmail for instance, will not run at all unless scripts are enabled.

3.2.2 Response Change method

This method involves sending a request to a server, keeping a copy of HTML tags on a client side proxy and then forwarding the request to the web server. On receipt of the response, the HTML tags are checked for the tags that were sent within the request. If any tags match, the response is marked as XSS vulnerable.

This method is advantageous as it is simple to implement and configure on the client side. There is little overhead because the checking of HTML is done on the client side, therefore removing overhead on the server side and increasing server response.

The major disadvantage of this technique is that it is not very smart. It will mark any HTML that is returned that matches the request as XSS vulnerable, even if it is safe. For instance, the request might contain the `<HTML>` tag, which is also returned. This will be marked as vulnerable, even though in nearly all cases it is not [5]. A technique to fix this error is to apply a length constraint to tags checked, but this is still not a full proof technique to prevent the incorrect XSS vulnerability indication.

3.2.3 Request Change method

In this method, an identifier, usually a number is inserted into the tags of the request. This creates an identity for the tag. Each successive tag is also given an identifier, incremented by one. This request is then forwarded to the web server which sends the response. The response will contain the tags with the associated identifiers, quickly identifying malicious scripts.

The advantage extends the response change methods and eliminates the major disadvantage (i.e. identification of malicious scripts). However, the overhead of this method is greater because initially a request with embedded identifiers is sent to receive a response identifying malicious scripts. This is sent additionally to the actual request.

4 Discussion

The methods above are all incomplete solutions to the XSS threat. Some provide security above and beyond this threat as well. The solutions are divided in to two categories, server and client.

Within the server solutions, the assessment tools, those that statically check web applications provide the best performance and error identification. In some cases, specifically WebSSARI, they are able to insert runtime checks in to the code to protect insecure areas of the application. However, assessment tools need constant updating in order to protect an application over time. Also, whenever a new part of the application is implemented the tool needs to be run again to identify new errors and insert new checks.

Scott and Sharps system architecture provides a good software engineering approach to protecting web applications. By removing the security concerns from the developers and creating a security gateway the system effectively protects the web server, and therefore its users, from the effects of XSS attacks. The configuration of the security gateway is intensive and requires a constraint for every URL on the web server, leading to a large set of rules. These configuration files will also need updating every time a new part of the application is added or modified. Using this method also degrades the performance of the web server, taking ten times as long as a HTTP proxy.

HTML filtering and encoding is by the far the easiest method to prevent XSS filtering, but not the easiest implementation. It requires the developers to change HTML special characters to safe characters for return to the user. This can be hard to do but there are various components that can help with the processing. This does involve a performance overhead, but not as great as Scott and Sharps methods.

With respect to the server side technologies, the author believes that the

method that provides the best level of protection is the HTML filtering and encoding. It provides the best performance and protection against XSS attacks. Its performance, though not as good as assessment tool performance, is acceptable. It does require effort on the development side, but most of that can be mitigated by using components that already filter the input / output stream. The filtering and encoding method also works against most kind of attacks and does not require extra configuration.

Client solutions include disabling scripts, the response change method and the request change method. Disabling scripts in a browser adversely affects the usability of most dynamic sites. Disabling scripts removes such functionality such as drop down menus, some images and multimedia. This severe restriction is not really a solution, but more of a strict adherence for safe browsing. It does protect users against all XSS attacks though.

The response change method protects users against most forms of XSS attacks. However it also incorrectly identifies some web servers as XSS vulnerable due to its non-complex processing. This can be solved partially by setting the minimum number of characters in tags before they are marked for checking. The performance of this method is good, requiring only one request being sent to the web server.

The request change method sends a request, where each HTML tag is embedded with an identifier number. On receipt of the response, it is checked for those identifiers. If any are the same the site is marked as XSS vulnerable. This solves the disadvantage of the previous method, request response, by providing a means to identify parameters. The major disadvantage is the required sending of two messages to the server, adding to the time taken to retrieve a web page. This also causes problems on the network as twice as many messages are sent.

The author believes that the best method is the request change method for clients, even though the performance of this method is poor, it better identifies the XSS attack and still lets the user have full use of a site.

Comparing the two major defences, server and client based, the author believes that the client based attack is the better approach of the two. Although web development teams are encouraged to perform security checks on their systems, a decentralized security system based in the clients would be a better approach to securing users against XSS attacks. The amount of servers that are

susceptible to XSS attacks is large and increasing [5]. A client based XSS defence would offer better protection against this form of attack. It could also be highly configurable by the user, rather than the web application, allowing them to identify sites that are insecure and the amount of dynamic content allowed when browsing. This gives the client more protection against sites that do not have good security processes in place.

5 Conclusion

This paper has introduced the dangers of XSS attacks and various methods to defend against those attacks. The dangers of XSS are large and include the theft of personal information and the running of malicious code on the user's computer.

The methods introduced are placed in to categories, server and client based. Within server based defences are methods such as; assessment of security holes, abstraction of security responsibility and HTML filtering and encoding. HTML filtering and encoding is considered the best solution because of its performance and ease of implementation using components that provide the filtering functions.

Client based methods of defence included; disabling scripts, response change method and request change method. Disabling scripts was quickly abandoned as it reduces the functionality of a web application. Request change method is considered the best solution as it can accurately identify XSS vulnerabilities although it has a cost in performance.

Client based methods were considered to be the better out of the two categories. There are many web servers that are insecure, and may always be insecure. Instead of web developers taking the responsibility of protecting the client, clients can use programs and systems to protect themselves from XSS attacks. This allows the client full configuration and they can then safely browse the Internet.

References

[1] M. Morana, "Make it & Breakit Defending Against Cross-Site Scripting Attacks," September 2004. http://www.securitypipeline.com/showArticle.jhtml.

[2] "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests," tech. rep., Computer Emergency Response Team, February 2000.

[3] G. Ollmann, "HTML Code Injection and Cross-site scripting." http://www.technicalinfo.net/papers/CSS.

[4] K. Ohamki, "Open source software research activities in AIST towards secure open systems," in *High Assurance Systems Engineering, 2002. Proceedings. 7th IEEE International Symposium on, Vol., Iss., 2002*, pp. 37– 41, 2002.

[5] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability," in *Advanced Information Networking and Applications, 2004. AINA 2004*, pp. 145– 151, 2004.

[6] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *Proceedings of the 13th international conference on World Wide Web*, pp. 40–52, ACM Press, 2004.

[7] D. Scott and R. Sharp, "Abstracting application-level web security," in *Proceedings of the eleventh international conference on World Wide Web*, pp. 396–407, ACM Press, 2002.