Web Application Security: XSS Attacks

Matthew A. Smith
Kansas State University
CIS 726

***Abstract*— Cross-Site Scripting, XSS, is the most common method of attack. It is simplistic in nature and utilizes both web application faults and bugs in browsers. It can be used either completely transparent to the user, or by relying on the user's lack of understanding. Three recent incidents are described involving XSS and major websites. Two prevention methods include filtering and web application gateways.**

I. Introduction

Web security must be a primary focus in a time when a person can make purchases, pay bills, and even access banks accounts all from the convenience of a web browser. There is a lot of sensitive information transmitted over the internet, due to websites enable people to conduct all business from a web browser. According to the Web Application Security Consortium's website, in 2005 there were 58 reported incidents ranging from vulnerabilities in email, to online gambling, to Universities' applicants data, to charity contributers' credit card information [4]. The abundance of incidents in a range of contexts should raise concern as to how, and why these incidents are happening.

In this paper, one of the most common class of attacks will be focused on as well as steps towards its prevention. A few recent incidents involving this type of attack with be will be discussed.

II. Cross Site Scripting

Cross-Site Scripting, referred to from here forward as XSS, is a method to trick a web page into displaying or altering the behavior a web page. XSS involves an interaction with active server content [6]. In essence, it allows an attacker to manipulate pages, to collect data, and to take control of the user's browser.

A. How does it work?

There are two basic types of XSS attacks, active and passive [2]. Both can be used to breach security if a site has not properly protected against it. The difference is in an active attack a user must initiate the action by following a link, or by submitting a form. The much more dangerous, is passive, which the user need only view a page containing the script in order for it to execute. In which case the user wouldn't have the chance to know what had happened. Both attacks utilize injecting a script into a page, so it will be executed on a visitor's machine.

1) Active XSS: Active XSS attacks are much more simple to use. Simply writing a well crafted URL and coaxing a user into following it, can be done as simply as using it in a link falsely labeled. This relies on the user being unaware XSS and unobservant to the URLs they follow. Unfortunately, this is the majority of users.

The example of a single URL requires the use of query strings to display the query string, unfiltered into the webpage. Take for example a site that takes a query such the following.

```
view.php?age=<script>alert('XSS')</script>

-- view.php
<?php
   $age = $_GET['age'];
   echo('You are '.$age.' yrs. old.');
?>
```

In this example, we can see how simple an XSS vulnerability can be. Let say an attacker was able to coax others into following the link. This would cause the JavaScript alert window to pop up on the user's browser. Sure an alert box is quite harmless, but what if instead the attacker redirected the user to a cookie stealing script on another server then redirected them to there original target? The original hosting site would likely have a user login on it, in which case the session id or email address is likely stored in a cookie. In this case the user unknowingly gave away their data, simply by following this link.

This form of XSS is extremely easy to perform. However, it is far from optimal for the attacker, as once the a user recognizes something out of ordinary about the link. Then the attacker can be reported and caught. It really does rely on users to overlook the details. More discrete are passive attacks.

2) Passive XSS: Passive XSS attacks are how large attacks can be conducted. The chance of exposure is much less and the potential for victims are much greater as the user does not need to consciously do anything except view the page. The most vulnerable targets for such an attack are guest books, HTML chatrooms, and discussion forums [6]. Many of these are often overlooked, or not promptly patched.

In this example, we will assume the attacker is using a message forum. On the forum the developer had the foresite to prohibit HTML in the body of messages, but they forgot to filter the title. Assume the attacker has the same cookie stealing script as above, named cs.php which queries for 'c'. Now, all the attacker needs to do is to title the name of his/her thread as follows:

```
<script>document.location=
   'attackhost/cs.php?c='
   +document.cookie)
</script>
```

Since the thread topic now has the script injected into it, any user who views the thread topics will be redirected to the cookie stealer, having their cookie data copied. This took no action from the user. They were just attempting to view the page. If the attacker placed the script on another compromised host, then this action would be extremely difficult to track down. On an active board, any users online could be victim in a matter of seconds from the injection of the script.

Now suppose, the user control panel for this forum had an image upload. Let's say the user wrote a script, named it 'evil.jpg', and uploaded it to there (or a victims) avatar. If this image was not validated then the script is now on the host. Now, this script can be used in a fashion as follows:

```
<script src='path/evil.jpg'></script>
```

Now, the script can be executed on the server itself. Data obtained from it will need to be sent out another way, but now the script's domain matches the host. Imagine if this site also contained an SSL encrypted store. Now the user's credit card and other sensitive data is jeopardized, as there will not be any warning from the browser saying that there are insecure items on the page. The user is none the wiser [6].

B. Examples

In order to further emphasize the risk of XSS attacks, a few real world examples will be discussed. These three vulnerabilities were discovered over the past year, and chosen from presentation because the targets are very well known and two of the three were breaches. The other was a disclosure, although it was not properly disclosed.

1) Google Mail: The first example was found in Google's Gmail web application in early March 2006. A teenage blogger claims that he discovered a flaw that allows JavaScript to run. The blogger identifies himself as a 14 year old by the name of Anthony, and uses the alias ph3rny [7].

He states that he was attempting to email himself a bit of JavaScript from his Yahoo! account to his Gmail account when he discovered the vulnerability. The JavaScript ran while the email was viewed in the preview pane (in Gmail where the list of email, their subjects, and a piece of their body are shown) stated ph3rny in his blog [10].

He goes on to describe what the conditions were that produced the vulnerability. It is very similar to how the active example work in the example I gave. The subject of his message was short as to allow more space for the script to run. Then the body begin with a small bit of text so the code wasn't treated as quoted text. This was followed by a short piece of code. Here was his content:

```
Subject: a
Body: asdfasdf<script>alert("asdF");</script>
```

This script simply displayed the JavaScript alert window. If ph3rny had had unethical intentions, he could have sent out a worm through this XSS hole that could DoS a server, steal other's email or logins or any number of other things.

Ph3rny posted this vulnerability on his blog, rather than taking a more responsible route and notifying Google about it in private. From an ethical stand, this was immature. From a security stand, this was very dangerous. However, as a testament to Google's alertness, the blog was posted at 2:19PM and from the comments to the post, Google had fixed the problem by 7 PM that evening. Leaving a gap between disclosure and resolution of under five hours [10].

As a further note, this vulnerability was only open to non-Gmail accounts. Gmail to Gmail correspondence had the script filtered appropriately as to not be executed.

2) MySpace: "But most of all samy is my hero." This was the mark of the XSS breach used against MySpace, the fifth largest website on the internet, last year. In under 24 hours, Samy had amassed over one million friend requests and had been added on the 'hero' list of each. Although the creater's intentions were for a laugh, a joke to his friends of having himself as one of their heros, it soon grew. As the plan for hero recognition amassed, he soon found that he was able to inject the script into each visitor of his page's own page. A worm was born. The exponential growth affected such a portion of MySpace that the entire site had to be shut down for its removal [8]. Note that this was in fact a breach and not a disclosure as in the previous report.

This incident involved more than simply an XSS exploit. MySpace was in fact filtering the 'javascript' string out of user content. The exploit was made possible because Microsoft's Internet Explorer has very lenient rules which allowed the string 'javascript' to be separated by a line break, yet still be executed. Since the string was broken, the MySpace filters did not catch it.

The script, with the addition of the line break, separating 'javascript', was inserted within a `<div>` tag, as the background URL. Another testament to the danger of allowing users to manipulate image content. To hide the addition of Samy as a hero was masked from the user by the use of an XMLHTTPRequest. This is used in Ajax to allow the page to load content without the need for it to refresh the page. The source code is available online along with explanation of its content[1].

3) Ebay: So far in the examples, the exploits have used passive XSS and have been on of a relatively low security concern. Yet, it is important to note that both were repaired in a matter of hours. The next incident, like the last, is a breach, meaning when the hole was found, the host site was not notified, nor the public, rather it was exploited for gain. This incident was conducted against the online auctioneer eBay in early March of 2005 [9].

Phishing is the classification of an attack involving theft of sensitive data from the victims. Typically, these attacks involve an email that appears to be a legitimate message from an online vendor, whom the sender hopes the victim has previous transactions with. These email typically have a link that the victim is requested to follow to update their account or

[1]http://namb.la/popular/tech.html

something similar. On arrival the website typically is identical to the host site in appearance with the same login forms. Upon loging in, the user might be asked for credit card information. As long as everything appears legitimate and the user takes the bait, then the attacker has acquired the information they wanted.

Back to the eBay incident. Although, little information was acknowledged about the incident itself, there was some information on the hole itself. Using an actual eBay link an attacker could send a link such as this:

```
http://cgi4.ebay.com/ws/eBayISAPI.dll?
  MfcISAPICommand=RedirectToDomain&
  DomainUrl=http://www.cis.ksu.edu
```

Although, rather than directing to K-State's CIS department website, an attacker could create a mock site of eBay and request the user information. Though the details of the phishing attack are not available, this incident could have been very large scale. Brian McWilliams, the author of Spam Kings, commented, ”It certainly adds some credibility to phishing e-mails. But scammers have used other types of URL redirection for a long time.” [9].

Worth noting, is that while the previous examples were of relatively low risk, eBay is certainly a host to large amounts of sensitive data. That being noted, an eBay spokesman told BetaNews, which published a story about the incident, days after it was found, ”We are aware of it and we have a fix rolling out in the next few days.” [9]. So, when the security of an email was at stake, Google responded with a fix within hours. When having Samy as your hero was at stake on MySpace, the problem was solved within hours. When a major marketplace containing user data was at stake, eBay dealt with it as much as a week later.

III. Prevention

Preventing XSS from exploiting a system is centered around filtering user input. According to David Zimmer, proper filtering alone can prevent about 95% of XSS holes [6]. The remaining 5% he suggests can not be filtered due to variances in browsers, such as Internet Explorer's ruleset that was used in the MySpace worm, the continual change in web technologies and simply human error in overlooking them due to the fact that most web sites are quite large and built upon legacy software. In addition to filtering, there are many commercial solutions available to protect web applications.

A. Filtering

The key component that XSS requires is that raw HTML may be displayed somewhere on a page. The second component is that a user must be able to manipulate the content of a page. These two components are what filtering will focus on. If the data is cleaned before it is stored, and it is clean when it is displayed, then an XSS hole has been avoided.

The question is: How can we filter user input to allow the user flexibility, yet still keep it restrict enough to protect the other users from XSS? Sure, the site could completely restrict HTML no matter what. This works for some applications. However, in a blog or a message board or another similar application, the user may desire certain formating or options that are only available by inserting raw HTML. These selective applications are where the problem lies.

A strict 'No HTML' filtering is quite easy to implement. Many scripting languages have built in methods to convert HTML encoded strings into safe strings that may be displayed without worry. In JavaScript, this function is used as follows:

```
var clean_str = escape(dirty_str);
```

If the value one is requesting is known to be an Integer, then this is just as simple. In ASP one could use the line:

```
x=cInt(Request.QueryString("num"))
```

Similarly in PHP one can simply add `+0` to the end of the assignment [6].

To allow the use of some HTML to be input or displayed, we first must decide on the rules for what can be used. In general the safe tags are `<font>`,`<img>`,`<b>`, `<i>`, and `<a>`. No other tags are suggested for input according to Zimmer [6]. I would include `<br>` in the list.

The simple tags, bold and italics, can easily be filtered. When scanning the user's input for '`<`' then having the next two characters '`b>`' or '`i>`' it can simply be allowed. In this filtering it would be wise to go ahead and make sure the closing tags are included.

Although commonly used, the image, link, and font tags are complex to filter in that they contain attributes. It is recommended when handling these to only accept certain attributes. Such as when an image tag is located, to create a new tag in the cleaned text to insert the scanned source attribute into. The other attributes can be ignored and with a new tag, any extra attributes can be removed without special rulesets to remove them on an individual basis. Links can be dealt with the same way, by allowing only the href attribute. The font by only accepting the size and color. Additional attributes could be applied to the filter as well such as alt tags to maintain accessibility.

It is important to know when the input should be filtered. For highest security, it is a good rule to keep the filtering on the serverside right before it is stored, and perhaps also before it is displayed, incase the storage mechanism has be contaminated. It should be done on the server side, because any client side script can be manipulated or circumvented as it is fully accessable to the client side. For scalability, it would be beneficial to have it run on the client side, but a balance would have to be made on the importance of security.

B. Web Application Gateways

In order to aide companies in their compliance to recent legislations, many commercial solutions are available. Of these there are two general types: Web Application Gateways and tools to supplement or replace a analysis by a security consultant. Only the former is applicable to this document.

The general idea of a Web Application Gateway, which I will refer to simply as a gateway hereforth, is that it terminate both inbound and outbound HTTP and HTTPS traffic [5]. Very similar to a standard network gateway, except that in this application it focuses on the web site specifically.

In Imperva's SecureSphere lineup, the gateway is 'smart' in that it learns from how users are interacting with the site and constructs a policy according to these actions. Security personnel can modify or create new policies. The gateway then can automatically tell when transactions deviate from policy, and further, which ones are an evolution of the user behavior or a change in the application [1]. Filtering can be conducted at this level for such items as the eBay incident which no data is stored, and such it does not make it to the server side filtering.

Assessment Management Platform (AMP), by SPI Dynamics implements both a web application gateway and an analysis scanner to aide in secure web transactions. While it focuses less on blocking improper access, it combines scanning with auditing of all transactions to inform security personnel what problem exist and where the problems are located. This can be used in development to help insure that the application is developed to be a secure product [3]. This type of application could be most useful in a University type environment where the web applications are not all managed by a single department and are constantly updated or fail to be maintained, audits could be used to notify the appropriate administrators.

In plug and play simplicity, Applicure's dotDefender product analyzes incoming HTTP requests and handles them appropriately based on a predefined ruleset of known attacks. These rules are automatically updated, so at to require a minimal amount of effort from the administrative staff. The mentioned list of attacks it protects against are DoS, SQL injection, cookie tampering, path traversal, probing, session hijacking, access from blacklisted sources, and of course XSS [2]. This product is focused towards the smaller company who may not have the resources to hire and train additional administrative personal.

There are obviously many more commercial solutions, these are only a sample of a few. However, they illustrate the idea of what this type of product can accomplish. With the starting price of Imperva's SecureSphere starting at $42,500 per unit, this solution isn't viable for a small startup or a personal website [1]. These products are still not a replacement for security minded development.

IV. Conclusion

Over 90% of websites contain XSS vulnerabilities making it the most common security issue [8]. In addition to being the most common vulnerability the examples, both theoretical and actual incidents, show how simple an attack can be. One does not need to have years of programming experience to be able to exploit a site to obtain user data. A simple cookie stealer can be written in 6 lines of basic PHP to log a user's IP, all their cookie information from the host site and redirect them to their desired location.

Even the fifth largest website, the highly successful online auction agent, and the leader in search engine technology have overlooked vulnerabilities in their production applications. Active attacks are common in phishing, through use of spam email. Passive attacks can completely circumvent SSL connections without the user knowing. Proper and thorough filtering can reduce the number of XSS vulnerabilities by ninety-five percent. Commercial products are available to help reduce this risk and even automate some of the tasks. Cross-Site Scripting vulnerabilities can be prevented when focused upon early in development.

References

[1] ”Security Appliances Aid Compliance. (Imperva),” eWeek, Feb 2, 2006.

[2] ”Software prevents Web attacks from known sources. (Web Application Security Provider Applicure Prevents Damages Related to Cyber Crime with the Release of dotDefender 2.1),” Product News Network, Feb 7, 2006.

[3] ”Software analyzes web applications for security risks. (SPI Dynamics Automates Comprehensive Enterprise-Wide Web Application Security Risk Management with AMP(TM) 2.0).” Product News Network, Feb 8, 2006.

[4] ”The Web Hacking Incidents Database,” Web Application Security Consortium, http://www.webappsec.org/projects/whid/whid.shtml, 2006.

[5] Conray-Murray, A., ”Web Application Security For All – New Web application security products are packaging the expertise of consultants into software tools and appliances,” Network Magazine, Feb 1, 2005.

[6] Zimmer, D., ”Real World XSS,” http://www.net-security.org/dl/articles/XSS-Paper.txt, 2003.

[7] Kirk, J., ”Teenager claims to find code flaw in Gmail,” NetworkWorld, http://networkworld.com/news/2006/030206-teen-flaw-gmail.html, 2006.

[8] Mook, N., ”Cross-Site Scripting Worm Hits MySpace,” BetaNews, http://www.betanews.com/article/CrossSite_Scripting_Worm_Hits_ MySpace/1129232391, 2005.

[9] Worthington, D., ”eBay Redirect Becomes Phishing Tool,” BetaNews, http://www.betanews.com/article/eBay_Redirect_Becomes_Phishing_ Tool/1109886753 , 2005.

[10] Anthony, ”Vulnerability in Gmail,” http://ph3rny.blogspot.com/2006/03/vulnerability-in-gmail.html, 2006.