# UNIVERSITY OF CENTRAL FLORIDA

## FRONTIERS IN INFORMATION TECHNOLOGY

## COP 4910


# CLASS FINAL REPORT

**Abstract**

This report brings together the final papers presented by the students in the Frontiers in Information Technology class, COP 4910 during the Summer-2015 semester. In addition, it is worth mentioning that this semester the students attended 56 talks and each student or team gave 4 presentations. In each talk they had to present the technical aspects of the selected topic along with its social impact, ethical aspects, and professional impact.

# *Cross-Site Scripting: XSS*

Joni Hall and Daniel Tumser
University of Central Florida
Frontiers in Information Technology

***Abstract*** Cross-Site Scripting (or XSS) is a security attack that occurs when an attacker uses another's browser to run a malicious script. It is called "cross-site" because it involves the interactions of two or more sites. The "scripting" in the name comes from the injecting of malicious scripts. There are several types of Cross-Site Scripting attacks that can occur: reflected, stored, and DOM-based. With the growing technology, and creation of JavaScript in 1995, hackers began to discover the vulnerabilities of JavaScript. This is when the concept of Cross-Site Scripting was born. Cross-Site Scripting exploded in 2005 when Samy Kamkar's "Samy worm" attacked and subsequently shut down MySpace. Cross-Site Scripting continues to be quite prevalent in today's digital world. The Open Web Application Security Project (OWASP) has named XSS in the Top Ten security threats list starting in 2007; it remains #3 on the latest update in 2013. There are two perspectives to analyze Cross-Site Scripting from when it comes to its ethical impact, the offensive and the defensive, and are for the most part the same as all cyber-security ethics in those contexts. On the offensive side of cyber-security, both the ethics and the law are unequivocal: do not do it. The Bureau of Labor Statistics expects a 37% increase in Information Security jobs from now until 2022. As far as future expectations goes, the frequency of XSS attacks does not seem to be slowing down. One estimate is that 94% of web applications are vulnerable to XSS. Another estimates that roughly 10-25 XSS holes are found within commercial products per month. These numbers are high and expected to remain high.

## I. INTRODUCTION

Cross-Site Scripting would hardly be possible without JavaScript, because an attack requires some script injecting to be successful. With the growing technology, and creation of JavaScript in 1995, hackers began to discover the vulnerabilities of JavaScript. Shortly thereafter, Netscape introduced same origin policy to combat XSS attacks. The same origin policy roughly states that a web browser may run scripts in one web page to access data in a second web page, if and only if the two web pages come from the same origin. However, the policy was not effective since it proved too restrictive for many large websites and could be bypassed altogether in many cases.

A few years later, in 1999, David Ross (working for Microsoft), was working on security responses for Internet Explorer. Ross demonstrated server-side script injection and how it worked. In that same year, he published a Microsoft-internal paper titled "Script Injection". The paper discussed, among other things, how the threat was exploited, how a virus using this attack might work and prevention by I/O filtering techniques. Microsoft acknowledging the real threat of XSS opened the doors for other vendors to get involved.

In 2000, Microsoft began working with the Computer Emergency Response Team (CERT) Coordination Division at Carnegie Mellon University. The goal of the two teams was to not only recognize and prevent attacks in Microsoft's products but also help prevent XSS attacks in the industry in general. At the time, Cross-Site Scripting did not have a name. The threat was known and referred to by its attributes. The term "Cross-Site Scripting" was coined by Microsoft and CERT during 2000. Now something that was known to have existed before finally had a name.

Even though Cross-Site Scripting was known to exist and be a real threat to websites, not many large-scale attacks occurred. That changed in 2005 with an XSS attack known as the "Samy worm" and its attack on MySpace. This was the first attack of its kind that affected such a large number of people in just a short amount of time. This paper was written with the intention of raising awareness about Cross-Site Scripting and how it may be prevented.

To meet the goal of raising awareness about Cross-Site Scripting, how it may occur, and prevention, the paper is organized into sections: Technical Aspects, Professional Impact, Social Impact, Ethical Impact and Future Expectations. The Technical Aspects section is split into three main subsections: Reflected XSS, Stored XSS, and DOM-Based XSS. Following this section is the Professional Impact section of the paper. Directly following Professional Impact will be the Social Impact section. The section following that is called Ethical Impact. Lastly, the paper will address Future Expectations and a Conclusion.

## II. TECHNICAL ASPECTS

The core of the Cross-Site Scripting vulnerability and attack vector is JavaScript. The technical aspects that make JavaScript a powerful and convenient tool for Web Developers in order to outsource user data processing and render dynamic content for a web application in a client's browser are the same things being leveraged by a malicious user to attack other web application clients.

The vulnerability occurs in a manner much the same as other security vulnerabilities in computer systems--giving the user control over input, access to the processing, or both. Without proper input handling or HTTP parameter filtering by either the web server and its back-end components, or the client's web browser, all of the functionality of JavaScript becomes a method for a malicious user to execute arbitrary scripts on non-malicious users.

### *A. Reflected Cross-Site Scripting*

There are broadly three methods to exploit a Cross-Site Scripting vulnerability, the first and most common Cross-Site Scripting vulnerability, accounting for approximately 75% of real world application XSS vulnerabilities is the Reflected Cross-Site Scripting vulnerability. The attack gets the name

"reflected" from a malicious injection into, for example, an HTTP parameter is reflected off of the web server when the request is sent and back to the user. An example real world Reflected XSS vulnerability commonly found is a dynamically generated error message from a web application that includes user input or HTTP parameters without proper sanitizing.[1] An example of the URL injection is as follows:

http://www.VulnSite.com/Vuln?param=<script>iDrinkYourMilkshake(your.cookie);</script>

The process of the basic attack on a known Reflected XSS vulnerability is straight-forward. The key step in the attack, where the vulnerability truly exists is in Step 5: Malicious User M crafts JavaScript code into the URL for the vulnerable part of the web application.

1. M sends this URL to Non-Malicious User U.
2. U clicks on the link that has been sent to him.
3. U's browser sends an HTTP request to www.VulnSite.com/Vuln with M's malicious script as its parameter
4. The Web Server receives U's request, processes the traffic with the parameter and sends an HTTP response with M's script injected into the HTML document sent to U
5. U's browser receives the HTTP response from the web server and renders the HTML document with the included malicious script, which now executes within the Document context of the browser, and the User context of the web application.

The aims of the malicious user can be many and the complexity of the URL-injected script can be very sophisticated when the issue of filter/sanitization bypassing and phishing content come into play, but the most common aims are stealing a legitimate user's session token or stealing the user's credentials for the application.

Because code is executing within the Document Object for the vulnerable application it has access to other objects within the document allowing JS to manipulate or transmit these objects. A very simple method for an attacker to steal a user's cookie would be to craft their JavaScript to redirect the Location of the Document to a web server that they control, and attach the cookie for their current session to that redirect as a location on the web server, or as a parameter. If this is successful the attacker need only examine the server logs or console output and extract the victim's cookie and begin session hijacking.[1]

A more sophisticated attack would leverage JavaScript's ability to dynamically generate content in a web page. The script may add to or replace the server provided content with an attacker designed form for the targeted user to fill out. This form would then POST the user-provided information to wherever the attacker has chosen, and acquire the user data or credentials to do with as they please.

Such an attack was used as a Proof-of-Concept XSS exploitation against Google by Jim Ley in 2004, and effectively turned a Cross-Site Scripting attack into a Trojan Application[1]. The full URL for his XSS trojan is shown below:

http://www.google.com/custom?cof=L:javascript:javascript:document.appendChild(document.createElement('script')).src='http://jibbering.com/test.js' [6]

URL-encoding has been removed from Ley's XSS attack for readability. Ley explains how his attack works on his website Jibbering.com: "The problem is that Google fails to correctly check that the image you reference to cutomise[sic] the look is an actual image, and not some script, this is a well known problem for web authors . . . what it does is in the second case is create a hidden IFRAME containing a regular Google search for password (or the search term you used if you used one) and return part of the page to a page on my site which stores the data, the first replaces it with a form requesting a credit card number to buy access to Google, with all the details forwarded to my site."[6]

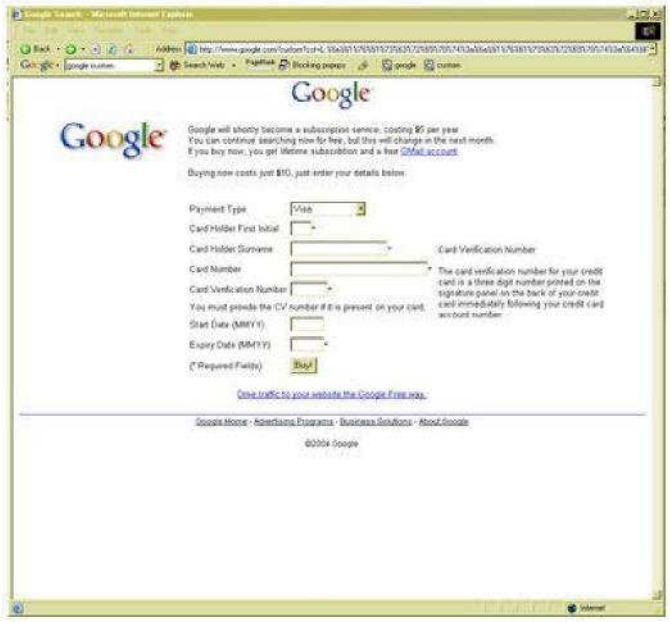


**Figure 1 - Google Registration Form**

The final result of this crafted URL would present users with the form shown in Figure 1, and as mentioned, submit all the user-supplied data to Ley's own site.[6]

One of the primary weaknesses of this attack vector is that if session hijacking is the aim, it relies on the targeted user currently having an active session to hijack. There are ways to get around this, but it is an inherent limitation on the Reflected Cross-Site Scripting attack, and one that isn't shared by Stored Cross-Site Scripting vulnerabilities.[1]

*B. Stored Cross-Site Scripting*

The other general method that Cross-Site Scripting attacks make use of is referred to as a Stored Cross-Site Scripting vulnerability or attack, and the name makes the details of the method obvious with some thought. If a Reflected Cross-Site Scripting attack "reflects" the injected

script off of the web server, then the Stored Cross-Site Scripting attack stores its injected script such that other users may find it and their browsers will execute the script when rendering the page.[1]

The steps involved in a Stored Cross-Site Scripting attack are similar to the Reflected XSS attack, with the exception that the attacker directly interacts with the web application, instead of directly communicating with the targeted user:

1. Malicious User M creates an account on the vulnerable application
2. M enters his/her script into a vulnerable fields for user input for display purposes
3. M's script is saved to the particular page for display to other users who browse to it.
4. Non-Malicious User U browses to the page.
5. The web server responds to U's request with an HTML document containing the malicious script.
6. U's browser renders this document and executes the script.

This method of Cross-Site Scripting attack overcomes the limitation of the Reflected attack, by way of ensuring affected users are logged in in order to have the script be viewed, thereby guaranteeing an active session, but has its own limitation on being used in spam. Because the attacker has the assurance of an active session by the user, some scripted behaviors are more reliable possibilities for writing into script execution.

One famous example of this is the case of Samy Kamkar's XSS exploit of MySpace known as the Kamkar "Samy Worm." His code, using AJAX would then make the attacked user's profile add Kamkar's as a friend, append "but most of all, Samy is my hero" to their pages, and finally copy the same code into the profile that has just been attacked. The result was a work created through Cross-Site Scripting that was spreading to more than one million profiles in 20 hours, and eventually crashed MySpace[7].

XSS vulnerabilities and attacks can be broadly divided between Reflected Cross-Site Scripting and the Stored Cross-Site Scripting, but there are subsets of these attack vectors worthy of note, as well, such as the DOM-Based Cross-Site Scripting method.

### *C. DOM-Based Cross-Site Scripting*

OWASP describes the Document Object Model, or DOM-Based, Cross-Site Scripting method as such: "DOM Based XSS is an XSS attack wherein the attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner. The page itself does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

Because this form of XSS is exploiting the client-side DOM environment, and not altering page content received from the server in the HTTP response with user input, the sanitization of user input is no longer the cause of the vulnerability. An example is given below:

```
Select your language:
<select><script>
document.write("<OPTION
value=1>"+document.location.href.substring(document.locati
on.href.indexOf("default=")+8)+"</OPTION>");
document.write("<OPTION value=2>English</OPTION>");
</script></select>[8]
```

This server-provided script makes references to the local Document object for the web page in the browser environment, specifically to the Location object within the Document. The Location contains the URL used for that page by the browser, including parameters used in the HTTP request, such as:

http://www.some.site/page.html?default=<script>alert(document.cookie)</script> [8]

The parameter in the HTTP request, named "default" and containing the alert script, does not alter the content of the HTTP response sent by the server containing the script for setting the default language, it is not processed by the server as user input. The parameter value is stored in Document.Location and when the server-provided script executes it accesses the client's locally stored Location for the parameter. In the case above, where the parameter contains a script, the script will then execute when the client-side processing of the script completes in rendering the webpage. The responsibility of developers shifts when it comes to addressing a DOM-Based Cross-Site Scripting vulnerability. Where Reflected and Stored XSS vulnerabilities are addressed with proper sanitizing user input and HTTP requests, DOM-based XSS is addressed with secure data handling in the JavaScript for your application.

## III. PROFESSIONAL IMPACT OF XSS

Overall, the Bureau of Labor Statistics groups the types of jobs relating to Cross-Site Scripting as Information Security Analyst jobs. Here is some information regarding the statistics for the job outlook of InfoSec positions in the period from

| Quick Facts: Information Security Analysts | |
|---|---|
| 2012 Median Pay | $86,170 per year<br>$41.43 per hour |
| Entry-Level Education | Bachelor's degree |
| Work Experience in a Related Occupation | Less than 5 years |
| On-the-job Training | None |
| Number of Jobs, 2012 | 75,100 |
| Job Outlook, 2012-22 | 37% (Much faster than average) |
| Employment Change, 2012-22 | 27,400 |

2012 - 2022.

| Job Title | Median Pay | Skills Required |
|---|---|---|
| Web Developer | $62,500 (in 2012) [4] | HTML, JavaScript, PHP, Java, C#, CSS |
| Platform Security Engineer | $90,000 (in 2015) | BS in CS, C++, JavaScript, verbal and written skills, privacy and security background |
| Web Application Security Engineer | $97,000 (in 2015) [3] | Bachelor's degree, developing web apps in Java, security certifications |
| Penetration Tester | $77, 274 (in 2015) [5] | web security and encryption, network security management, security testing and auditing |

**Table 1 - Job Examples in XSS**

### Iç. SOCIAL IMPACT OF XSS

Often with user training in the realm of security amounts to hovering the cursor over a link in the email to see if the link leads where it says. The URL that appears in the bottom-left of the window is naturally going to match the link text, because the purpose of a Reflected XSS attack isn't to scam you into a fake site. User training by an organization's IT security employees needs to be more thorough and needs to look at ways to build in incentives for proactive security behaviors from other employees, and this needs IT employees willing to invest the time to do this.

Even more important than user training, however, is the security consciousness of web developers themselves. Web Developers need start having security certification requirements at hiring and reviews for their code throughout development. The expectations and culture of developers needs to adjust to the modern ubiquity of cyber-threats, and their legal obligations to their clients.

### ç. ETHICAL IMPACT OF XSS

There are two perspectives to analyze Cross-Site Scripting from when it comes to its ethical impact, the offensive and the defensive, and are for the most part the same as all cyber-security ethics in those contexts.

On the offensive side of cyber-security, both the ethics and the law are unequivocal: do not do it. Title 18 of the US Code Section 1030 addresses the federal law with regards to cyber-crime, as well as the minimum and maximum sentencing. If you are convicted in a federal court on a violation of 1030 you may find yourself facing 5-20 years in prison, with the maximum looking more likely given the Silk Road conviction. It should be self-evident that having permission to execute these attacks negates the legal issue, so long as systems not outside the control of the tester or company are compromised.

Defensively the ethics are the burden of the web developers. Vulnerable code in a web application is always the responsibility of the organization running it, the developers who built it, and the IT professionals maintaining it and all its components. Organizations have an ethical obligation to their clients to do due diligence in maintaining the confidentiality of certain information entrusted to them, as well as a legal obligation. In the United States all but three states (Alabama, New Mexico, and South Dakota), have laws on the books requiring all companies operating in their borders to disclose unauthorized data breaches that have compromised customer data.

### çI. FUTURE EXPECTATIONS

In terms of future expectations for Cross-Site Scripting, you can expect it to be around for a long time. One estimate is that 94% of web applications are vulnerable to XSS. As stated before, roughly 10-25 XSS holes are found within commercial products per month [2]. With newer computer models and software, the risk of a Cross-Site Scripting attack is more likely as all the “bugs” have not been worked out of those types of machines yet. The Bureau of Labor Statistics expects a 37% increase in jobs in Information Security (in general) from now until 2022. As computers make their way into more and more job industries, we should expect the need for computer security to grow, given that all computers need security.

### çII. CONCLUSION

To summarize, Cross-Site Scripting is a usually malicious web application attack that involves injecting scripts in order to get another user’s browser to execute the malicious script. It is malicious because it is usually done to extract some data or information from an unknowing user (“victim”) by a hacker. The intent with what the hacker may do with the data is not always malicious.

There are three types of XSS that may be done. Reflected XSS is the most commonly used, accounting for 75% of Cross-Site Scripting attacks, and it is done by a malicious injection into, for example, an HTTP parameter that is “reflected” off of the web server when the request is sent and back to the user. The second type of attack is called Store XSS, which is done by storing an injected script on the web server, which is then navigated to by the unknowing user. The third type of Cross-Site Scripting attack is the least commonly used and it is called DOM-Based XSS.

Prevention of a Cross-Site Scripting attack can be summed up in a few key concepts. One is the sanitization of data input. A web site must sanitize all user input/ouput, on every page, ensure that any client-side code execution handles data securely to prevent injection and apply this input/output verification with boundary validation. Another key concept to prevention of XSS attacks is user/consumer education. Users must be trained what to look for and how to prevent an attack. In order to do this, employers need to have a general idea of what to look for and how to prevent attacks. Proper training in a workplace makes all the difference for the users expected to understand how these types of attack work and what can be done to combat them.

## REFERENCES


[1] Stuttard, Dafydd, and Marcus Pinto. The Web Application Hacker's Handbook Finding and Exploiting Security Flaws. 2nd ed. Indianapolis: Wiley, 2011. Print.

[2] "The Cross-Site Scripting (XSS) FAQ." 'Web and Application Security News' Web. 17 June 2015. http://www.cgisecurity.com/xss-faq.html.

[3] "Web Application Security Engineer Salary." Web Application Security Engineer Salary. Indeed. Web. 17 June 2015. http://www.indeed.com/salary?q1=Web Application Security Engineer&l1=.

[4] "Web Application Developer Salary (United States)." Web Application Developer Salary (United States). PayScale. Web. 17 June 2015. http://www.payscale.com/research/US/Job=Web_Application_Developer/Salary.

[5] "Penetration Tester Salary (United States)." Penetration Tester Salary (United States). PayScale. Web. 17 June 2015. http://www.payscale.com/research/US/Job=Penetration_Tester/Salary.

[6] Ley, Jim. "Google Desktop Exploit." Google Desktop Exploit. 1 Oct. 2004. Web. 23 June 2015. <https://web.archive.org/web/20050307212654/http://jibbering.com/2004/10/google.html>.

[7] Kamkar, Samy. "I'm Popular." I'm Popular. 4 Oct. 2005. Web. 23 June 2015. <http://namb.la/popular/>.

[8] "DOM Based XSS." - OWASP. Web. 24 June 2015. <https://www.owasp.org/index.php/DOM_Based_XSS>.