Virginia Journal of Science
Volume 70, Issue 3
Fall 2019
doi: 10.25778/bx6k-2285

A Study of Existing Cross-Site Scripting Detection and Prevention Techniques Using XAMPP and VirtualBox

Jalen Mack, Yen-Hung (Frank) Hu, and Mary Ann Hoppa

Department of Computer Science
Norfolk State University, Norfolk, Virginia, USA

ABSTRACT

Most operating websites experience a cyber-attack at some point. Cross-site Scripting (XSS) attacks are cited as the top website risk. More than 60 percent of web applications are vulnerable to them, and they ultimately are responsible for over 30 percent of all web application attacks. XSS attacks are complicated, and they often are used in conjunction with social engineering techniques to cause even more damage. Although prevention techniques exist, hackers still find points of vulnerability to launch their attacks. This project explored what XSS attacks are, examples of popular attacks, and ways to detect and prevent them. Using knowledge gained and lessons-learned from analyzing prior XSS incidents, a simulation environment was built using XAMPP and VirtualBox. Four typical XSS attacks were launched in this virtual environment, and their potential to cause significant damage was measured and compared using the Common Vulnerability Scoring System (CVSS) Calculator. Recommendations are offered for approaches to impeding XSS attacks including solutions involving sanitizing data, whitelisting data, implementing a content security policy and statistical analysis tools.

Keywords: Cross-Site Scripting (XSS) Attack, XAMPP, VirtualBox

1. INTRODUCTION

Today, millions of users rely on web applications for bank information, education, and social media. However, the presence of security vulnerabilities creates risk when they use these applications. Malicious users can take advantage of these vulnerability to steal sensitive information, send illegal Hypertext Transfer Protocol (HTTP) requests, redirect unsuspecting users to harmful websites, install malware, and perform other malicious operations (Gupta, Govil, & Singh, 2015). XSS is a common cyber-attack typically found in web applications (Cross-site scripting). XSS attacks are a type of injection that occurs when hackers exploit a weakness in an otherwise benign and trusted webpage to insert their own malicious code. That code can be implemented to steal users' personally identifiable

information (PII) such as credentials, session cookies, sensitive data; and it even can live persistently on a site to continue attacking multiple users (Vigliarolo, 2018).

XSS attacks can cause significant damage to individuals, businesses, and other enterprises. According to a Ponemon Institute study on the *Cost of a Data Breach*, the mean time to identify a cyber-attack is 197 days, and the average total cost per breach is US$3.86 million (IBM, 2018). XSS attacks can negatively impact a company's reputation too, which leads to loss of productivity and revenue. XSS attacks have targeted social networks such as MySpace, Orkut, LinkedIn, Twitter and Facebook, exposing hundreds of millions of users to potential PII theft and other nefarious actions.

There are two main approaches to inserting malicious code into a webpage: reflected-XSS and stored-XSS attacks (Vogt, 2006). There is a third, less well-known type of XSS attack called DOM-based XSS that is beyond the scope of this paper. A reflected-XSS attack is delivered to the victim by an indirect means, such as an e-mail message or another website. The user may be tricked into clicking on a link, submitting a form, or even just browsing to a website. This provides an opportunity for the hacker to send malicious code as part of a server request that travels to a vulnerable website, then back to the user's browser. The user's browser executes the code because it appears to be coming from a trustworthy source.

A stored-XSS attack involves malicious content stored on the target server. If a user requests stored information from that server, such as from a webpage that contains a malicious script, the code is returned as part of the message. For example, an attack executed in the victim's web browser might transfer cookies to a web server that is controlled by the attacker. Cookies are the easiest way to locate and verify users and are used on most web browsers. This makes them an attractive target for attackers. If an attacker can steal the valid cookies from a victim's session, then the attacker can hijack the victim's session (Gupta & Sharma, Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense, 2012). This also could give the hacker the ability to login to a user's social networks such as Twitter, Facebook and Instagram, or other accounts.

The goals of this research are: to develop an understanding of the many risks of XSS attacks; to identify some companies affected by XSS attacks and the damages they suffered; to launch XSS attacks on a virtual machine to measure the damages they can cause; to categorize and compare potential defensive mechanisms; to identify solutions or proactive approaches to securing vulnerabilities exploited by XSS attacks; and to summarize best-practice solutions and recommendations based on the examined vulnerabilities.

The remainder of this paper is organized as follows. Section 2 introduces vulnerabilities and impacts related to XSS attacks. Section 3 describes exploitation and detection of XSS vulnerabilities. Section 4 documents XSS attack case studies. Section 5

explains the research methodology. Section 6 describes the research experiment in detail. Section 7 gives research results, discussion and recommendations. Section 8 concludes this paper.

2. VULNERABILITIES AND IMPACTS RELATED TO XSS ATTACKS

When attackers penetrate a victim's system, they have the ability to examine the system as well as to use other intranet applications. A few things a successful attacker might be able to do is take over an account, spread malicious worms, control web browsers remotely, exploit applications, and install a keylogger.

A webpage includes text and HTML markup that are available on the server and read by the client browser. Web sites that generate only static pages are able to have full control over how the client interprets these pages. Web sites that generate dynamic pages do not have complete control over how their outputs are interpreted by the client (Shanmugam & Ponnavaikko, 2008). XSS attacks could occur at the application-level when a server program (i.e., dynamic webpage) uses unrestricted input via an HTTP request, database, or files in its response without any validation, which allows malicious code injection (Mukesh Kumar Gupta, 2015). Exploitation of such vulnerabilities allows hackers to steal confidential information and execute other malicious actions. Examples of XSS vulnerabilities include failing to encode HTML outputs to the browser, and failing to validate inputs to web applications.

The effects of an XSS attack normally depend on the type of application, the functionality and data, as well as the affected user's privileges. The consequences of an XSS attack can be severe, including identity theft, confidential information retrieval, denial of service, changing the way the web browser operates, and even spreading worms that access the user's computer and view the user's browser history or remotely control the browser (Shanmugam & Ponnavaikko, 2008).

XSS attacks are so popular because they are fairly easy to launch and don't require a lot of technical skill. Some XSS attacks can be launched with merely basic knowledge of JavaScript and HTML. This makes it quite simple for attackers to learn how to carry out XSS attacks.

The capabilities of an attacker who launches an XSS attack can be quite broad. It is difficult for companies to track an XSS attack because there are so many ways an attacker can launch an XSS attack and exploit an XSS vulnerability. An attacker who exploits an XSS vulnerability typically is able to (Cross-site scripting):

- Impersonate or masquerade as the victim/user.
- Carry out any action that the victim/user is able to perform.
- Read any data that the victim/user is able to access.
- Capture the victim/user's login credentials.
- Perform virtual defacement of the website.

- Inject Trojan functionality into the website.

3. EXPLOITATION AND DETECTION OF XSS VULNERABILITIES

There has been a lot of prior research focused on determining ways XSS attacks are used to control and adjust how a webpage operates. Multiple platforms offer ways to test or exploit vulnerabilities of XSS attacks. A few such websites are Web Goat, Acunetix (Acunetix, 2014), Pentest Tools and Burp (Sarmah, Bhattacharyya, & Kalita, 2018). These options allow users to enter website addresses and have them checked for vulnerabilities. Since most XSS attacks involve JavaScript, all detection tools should be able to detect malicious JavaScript (Vonnegut, 2017). However, the security testing they can provided still will be limited to only systems the user owns or has permission to work with (Gupta & Sharma, Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense, 2012).

To minimize XSS attacks, organizations must assess their web application code and eliminate any XSS vulnerabilities. To successfully identify potential XSS attacks, organizations should adopt a few measures including (Laing, 2017):

- Evaluate any object that a browser may open, or that may launch a browser. This includes email messages, attachments, downloads, webpages, and any other document that contains HTML links.
- Perform rapid static analysis of each object, evaluating them for malicious capabilities and links, known attack signatures, structural deviations, and other anomalies.
- Perform full behavioral analysis by completely executing each object and testing it for evasion techniques and malicious actions.
- Monitor the network for side-effect activity created by malware operating on a network, such as code injection, malware communicating with command and control servers, and other anomalous activity.

There are additional ways to detect XSS attacks, but the methods listed above have been deemed the most effective. The lack of more efficient solutions is one of the reasons that XSS attacks are still so common today.

3.1 Cross-site Scripting Actors

Typically, there are three actors involved in an XSS attack: the victim, the hacker, and the website. The victim is the user of the website who requests pages from it using their browser. Hackers are the malicious users who find a vulnerability to exploit in a website to target users. The website delivers HTML pages to users who request them (Kallin & Valbuena, 2013). In some XSS attack cases, the victim doesn't have to request information from the webpage to be attacked. This makes XSS attacks more dangerous, especially XSS worms that self-propagate. When all three actors are implemented, it can result in a loss of confidential information, money, reputation, etc.

3.2 Preventing XSS attacks

A few methods are known to be effective against XSS attacks (Sarmah, Bhattacharyya, & Kalita, 2018). Five are summarized here.

- The first technique is ***escaping input***, which is the concept of ensuring the data an application has received is secure (i.e., cannot be inadvertently interpreted as code) before rendering it for further processing (Vonnegut, 2017). This technique also is used to encode special characters. As shown in Table 1, escaping changes specific characters that might otherwise be deciphered as harmful code by prefixing or replacing them with other characters. This helps control the information that goes to the webpage, which in turn reduces the chance of XSS attacks. If users are not allowed to add their own code or information to a webpage, a good rule of thumb is to escape all HTML, URLs, and JavaScripts (Vonnegut, 2017). If the webpage allows users to enter code or information (e.g., Facebook), it is best to use a similar approach that escapes all HTML input.

Table 1. Ways to escape special character attacks

Replace…	with
<	&lt
>	&gt
(	&#40
)	&#41
#	&#35

- Another technique is known as ***input validation***. This refers to the process of making sure input data is benign and contains no unexpected characters or malicious values that might otherwise attack a database, site or user. Input validation is an effective technique against XSS attacks because it prevents users from entering special characters into the fields altogether, instead of trying to intercept or deny resulting requests (Vonnegut, 2017).

- ***Sanitizing input*** prevents XSS attacks by combining other techniques such as escaping and validation. User input is analyzed and scrubbed clean of potentially harmful markup, effectively changing unacceptable user inputs to acceptable formats (Vonnegut, 2017). Sanitizing user input is especially helpful on sites that allow HTML markup.

- Another way to prevent XSS attacks is ***whitelisting values***. If a particular dynamic data item should only accept a handful of valid values, it is best practice for the rendering logic to permit only known allowed values (Protecting Your Users Against Reflected XSS). An example is for a webpage that expects

an "eye color" parameter to make sure only alphabetic letters – or a limited list of prescribed values – are accepted, instead of digits or other special characters. While whitelisting and input validation are more commonly associated with SQL injection attacks, they also can be used for preventing XSS attacks (Vonnegut, 2017).

- XSS attacks rely on the attacker being able to run malicious scripts on a user's webpage either by injecting inline <script> tags somewhere within the <html> tag of a page, or by tricking the browser into loading the JavaScript from a malicious third-party domain. So, the last method to prevent XSS attacks is implementing a ***content-security policy***. A content-security policy allows the creator of the webpage to specify where JavaScript and other potentially harmful methods can be launched and implemented (Protecting Your Users Against Reflected XSS). In this way a content-security policy can ensure that inline JavaScript isn't executed, which could prevent some XSS attacks.

4. CASE STUDY OF XSS ATTACKS

Members of the security community have researched numerous cases of XSS attacks. While there are many techniques to detect and prevent them, XSS attacks still affect companies and millions of their users. According to the report *Web Application Attack Statistics 2017 in Review*, XSS is used in 31% of all web attacks (Staff, 2018). The websites of many popular companies such as Google, eBay, Yahoo, Facebook and PayPal have been shown to have vulnerabilities that leave their users defenseless against XSS attacks. This project examined five documented XSS attacks as instructive use cases. Each case explains the organization that was targeted, the nature of the attack, and some suggested measures that could have prevent the attacks.

4.1 XSS Attack on MySpace (2005)

In October 2005, an XSS worm attacked a popular social networking website known as MySpace. Samy Kamkar, a 19-year-old hacker created the first known XSS worm to exploit MySpace's blacklist-based validation mechanism (Dabirsiaghi, 2008). What made this attack so important is that it didn't need user input – it spread on its own – and it consequently popularized XSS attacks. Within 24 hours, the attack affected over one million MySpace users. Although Kamkar's worm was harmless in theory, MySpace had to briefly shut down to fix the problem that allowed the worm to self-propagate, resulting in hours of lost production and MySpace becoming a victim of the attack right along with all the affected users.

Kamkar was able to penetrate MySpace's system by uploading an infected JavaScript to his profile, which then retrieved the user identity of the victim from the HTML source using the DOM (Richie, 2007). The attack itself was possible because an HTTP GET parameter was accepted without proper input validation checks and then echoed back to the user. If a secure Hypertext Transfer Protocol (HTTPS) had been implemented, this attack could have been prevented. (Richie, 2007). In retrospect, there

were many opportunities to prevent this attack. MySpace was capable of filtering the JavaScript, but failed to do so. Whitelisting and output escaping also could have prevented this attack. Implementing a content security policy could have blocked Kamkar from altering the code for his profile.

4.2 XSS Attack on PayPal (2006)

In June 2006, PayPal fell victim to an attack that had the potential to affect over 200 million users. Although the code from the attack was never released, it was said to be an XSS attack. The attacker inserted malicious code to retrieve confidential user information (Borg, 2006) (Seals). The attacker targeted users by sending an email stating their PayPal account had been disabled, and providing a link that allegedly would forward them to a solution. Instead, the link pointed to a malicious URL hosted on the legitimate PayPal website that asked for the user's social security number, credit card number, PIN, and other personal information (Borg, 2006).

This attack hinged on the attacker's malicious code being saved into the web application repository by the server, and then launched on the victim's browser (Kour, 2016). This was possible because PayPal's Web Application Firewall (WAF) was outdated, and they were not filtering for malicious JavaScripts. PayPal never revealed the amount of revenue it lost or the number of customers affected by this XSS attack. To prevent similar attacks in the future, PayPal could invest in an Acunetix Web Vulnerability Scanner which checks websites for exploitable vulnerabilities.

4.3 XSS Attack on Orkut (2010)

In 2010, an XSS vulnerability was exploited on Google's social media platform Orkut. It was a fast-moving malicious JavaScript that forced users to post specific content. This attack affected a victim's profile, then spread through all their friends, who spread it to all their friends, and so on, ultimately affecting over five million users all over the world. It spread overnight and infected users who viewed emails or Orkut messages carrying the malicious payload (Higgins, 2007). The email addresses of all victims were made available to the attacker which left them vulnerable to further attacks. The vulnerability was fixed within a few hours and the affected profiles were repaired. Since the accounts were connected to Google, all users were instructed to reset their passwords.

Persistent XSS vulnerabilities like the one exploited in this attack are the result of failing to properly sanitize input into forms. This allows attackers to insert malicious code into pages (Constantin, 2010). Validating input could have been used to prevent this XSS attack by making sure only legitimate data was being input into the webpage forms. Another effective measure would have been a content security policy that could have prevented the malicious JavaScript from being loaded and executed in the first place. In addition, Orkut didn't use a secure protocol; if they had, the breach also may have been prevented.

4.4 XSS Attack on Amazon (2013)

In December 2013, Amazon became a victim of a persistent XSS attack that left their customers vulnerable to their information being stolen. The vulnerability affected Kindle e-book readers. The malicious code was injected through e-book metadata. For example, the attacker could add a book title containing code such as *"<script src="https://www.example.org/script.js"></script>"* (Kovacs, 2014). This allowed cookies to be accessed by the attacker, which could lead to personal information being compromised, such as usernames and passwords. This vulnerability affected everyone who used the Kindle library to keep their e-books.

Amazon took a little over a month to respond to this vulnerability. The attack damaged Amazon's reputation, and likely affected their revenue: users were afraid to download the Kindle application due to all the bad press, which meant they weren't buying eBooks for the reader either. This vulnerability could have been prevented by using intrusion detection systems, which wouldn't have allowed the attacker to insert malicious data via e-book metadata. Likewise, validating input could have prevented the malicious code injection.

4.5 XSS Attack on Twitter (2014)

In 2014, an XSS vulnerability was found in TweetDeck, an application within Twitter (Cross-Site Scripting (XSS) Found in Tweetdeck, 2014). The attacker simply tweeted malicious JavaScript to make users automatically retweet tweets, and it began to regenerate. At the time, Twitter had over 50 million users, and over 15 percent of them were affected. Users were concerned that their accounts had been hijacked. The vulnerability remained on the site for so long that some users began to use it to implement harmful JavaScript and possibly steal other users' credentials. It is very likely that many users' information was stolen, although this was never confirmed.

This attack was possible because Twitter didn't have an updated WAF to filter code before it is processed to the webpage. To prevent this attack, Twitter could have practiced sanitizing input, which would have prevented the attacker from implementing the code in the browser. Twitter also could have protected their restricted servers by implementing separation of duty and access so that third parties would not be able to access them.

5. RESEARCH METHODOLOGY

This research explores the relationship between web application vulnerabilities and XSS attacks. The first four sections of this paper introduced details about the elements of typical XSS attacks – including the actors – as a basis for understanding the motivation and importance of this research. The experimental component of this study includes launching XSS attacks modeled after known attacks in a virtual environment to gain insight about ways they could have been prevented, and finally proposing a list of alternative solutions to prevent them. The attacks included were reflected XSS attack, persistent XSS attack, stealing cookies, and keylogging. Each of these attacks is related in some way to the attacks explored in the XSS use cases examined earlier.

The following methodology is followed in conducting the research and subsequent experimentation:

- Study and analyze previous XSS attack cases.
- Build fundamental virtual systems including web servers, SQL and several other related servers if needed, and some web users to mimic real-world systems.
- Introduce/inject well-known XSS attacks to the fundamental systems.
- Measure damages caused by the XSS attacks.
- Suggest solutions to prevent such XSS attacks.

6. CONDUCTING THE EXPERIMENT

6.1 Studying and Analyzing Previous XSS Attack Cases

There have been a few cases in which an XSS has affected thousands of people by exploiting vulnerabilities. The knowledge and experience gained from these cases can help implement XSS attacks in the virtual environment. Every XXS attack explored in this research exploits a vulnerability within a website. Most of the discussed XSS attacks could have been prevented if HTTPS were used instead of HTTP. HTTP doesn't encrypt data, leaves users open to attacks, and can present altered data to end users. Systems that use HTTP transmit data on port 80 and are vulnerable to information being intercepted.

HTTPS provides an authenticated server along with protection from hackers and data encryption. HTTPS transmits data on port 443 and uses a Secure Sockets Layer (SSL), which establishes encryption between the server and web browser. Transport Layer Security (TLS) is a cryptographic protocol that provides end-to-end communications security over networks and is widely used for internet communications and online transactions (Kerravala, 2018). The PayPal and Twitter attacks described earlier might have been prevented if the TLS protocol had been implemented (Kerravala, 2018).

The attacks discussed in this research are exploited by an attacker targeting the server and bypassing validation mechanisms. Each company used a server to process JavaScript after performing input filtering and other XSS prevention techniques; however, the attacks still occurred. The use of separation of duties in these attacks would have ensured that the attacks were detected quickly and protection mechanisms were enacted to protect the server. Separation of access would have ensured that no third parties could access the server, thereby keeping user information secure.

Table 2 shows the targets of XSS attacks from the previous section, the types of attacks that occurred, and security protocols that might have prevented them. None of the attacks listed in Table 2 were detected immediately. If these companies had used appropriate security protocols/tools, these attacks might have been prevented, or perhaps less effective.

Table 2. XSS attacks and protocol to possibly prevent them

Targeted Website	Type of XSS attack	Brief Description	Vulnerabilities	Protocols/tools to possibly prevent attack
MySpace (2005)	Stored XSS Worm	Worm exploited a flaw in MySpace's filter, which allowed hacker to inject code into a user's profile.	Located within OS of the server, HTTP, and web application	HTTPS, WAF
PayPal (2006)	Stored XSS attack	Vulnerability located in PayPal allowed an attacker to steal confidential information from users.	Outdated Web Application Server	HTTPS, WAF
Orkut (2010)	XSS worm	Self-propagating worm affected users by spreading malicious code to each profile who viewed the affected profile.	Vulnerable Web Server, Use of HTTP	HTTPS, WAF
Amazon (2013)	Stored XSS attack	Vulnerability allowed hackers to steal cookies, and user credentials.	Vulnerable Web Server, Outdated WAF	HTTPS, WAF
Twitter (2014)	Stored XSS attack	An XSS attack on Twitter caused users to post things without their permission.	Within the web server, Outdated Web server, No SSL Certificate	HTTPS, WAF

6.2 Building XAMPP on Virtual Machine for Simulating XSS Attacks

The following testbed was built for the purposes of this experiment: a virtual machine hosted in VirtualBox using XAMPP (XAMPP Apache + MariaDB + PHP + Perl, n.d.) under the domain "http://localhost". VirtualBox is software that allows users to run multiple operating systems in a simulated environment. This is beneficial when deploying XSS attacks so that the host operating system will not be damaged. VirtualBox allows users to select the storage and memory size needed for a virtual machine.

To implement a VirtualBox system loaded with Windows 10 to test the XSS attacks, download VirtualBox for Windows from the VirtualBox website (Mac, Linux, and Solaris downloads also are available). Next download the Windows 10 ISO file directly from the Microsoft website, and install it to emulate the Windows environment. Once Windows is successfully installed onto the virtual machine, download XAMPP for Windows from the Apache Friends website.

When XAMPP is installed, the user can select which components to install to test XSS attacks. Only Apache, PHP and MYSQL were needed for this project. The default ports remain the same for launching attacks, and the XAMPP control panel is right-clicked

to run the program as an administrator. Webpages can be added to the XAMPP folder to add files to the local host. For this project HTML, PHP and JavaScript files were used to implement the attacks.

6.3 Create Simulated XSS Attacks on an XAMPP System

In this research, several well-known XSS attacks were introduced into a web server. To implement these attacks, a vulnerable webpage is created and launched on the local host. Knowledge of JavaScript and MySQL are needed to implement these attacks. Four XSS attacks are launched on the XAMPP system to mimic damaging XSS attacks:

- **Reflected XXS Attack**: Reflected XSS is the most common XSS attack method. When a reflected XSS attack occurs, malicious code is reflected off the web server. The attacker injects code into the web server and the victim's browser executes the code. Common reflected XSS tactics include stealing cookies, redirecting to a phishing site, and making the user complete a task. For this experiment, code is injected into a webpage to show that the webpage is vulnerable. When the following code is injected into the text box

  "<script>alert("XSS") </script>".

  an alert text box pops up showing that the webpage is vulnerable. After determining that the webpage is vulnerable, hackers are able to launch almost any XSS attacks into the search bar. Figure 1 shows the URL string when a malicious script was injected into a search box implemented on the server.

localhost/xss/2/index.php?search=%3Csecipt%3Ealert%28%22XSS%22%29%3C%2Fscript%3E

Figure 1. URL of a malicious string inserted into a search box

- **Persistent XSS Attack**: In a persistent XSS attack, the malicious code comes from the database. These attacks often occur on blogs, forums, and web browsers. The code forces the webpage to redirect the user to another website. Following this command, the JavaScript will return the user to the webpage containing the script. Attackers use this technique to redirect users to fake websites to ask for user information such as credit card, social security numbers, and other confidential information. Figure 2 shows a JavaScript being stored on the webpage. If users visit the webpage after the implementation of this code, they will be directed to a Google search of Norfolk State University.

```
<script>
window.location='https://google.com/search?q=norfolk+state+university'
</script>
```

Figure 2. A malicious JavaScript inserted into the webpage

- **Stealing Cookies XSS Attack**: A cookie is a tiny piece of information that is sent from a website and stored on the user's computer by the web browser. Websites use cookies to remember certain details about a user, and in other situations such as adding items to a shopping cart. Attackers can use this data maliciously to steal sensitive information like credit card numbers, browsing history and email information. Figure 3 shows a JavaScript to capture the cookies of a user who views the webpage. When this code is implemented, the attacker can view confidential information about a user and perform any actions for which the user has permissions (Fake WordPrssAPI Stealing Cookies and Hijacking Sessions, 2017).

```
<script>
window.location='http://localhost/cookiemonster.php? cookie=' +eacape(document.cookie.)
</script>
```

Figure 3. A malicious JavaScript to capture the cookies from a user

- **Keylogging XSS Attack**: Keylogging often is used in XSS attacks to capture the user's keystrokes to steal usernames, passwords, social security numbers, addresses, etc. Keylogging attacks are so successful because they are difficult to detect. JavaScript, PHP, and HTML code can be used to implement this attack. Figure 4 shows a JavaScript used to launch a keylogging attack on the user.

```
var keys=''
document.onkeypress = function (e) {
   get = window.event?event: e;
   key = get.keyCode?get.keyCode:get.charCode;
   key = Sring.formCharCode(key);
   keys+=key;
}
window.setInterval(function() {
   if(keys != '') {
      new Image().src =  'http://27.0.0.1/xss/7/exploit/exploit.php?keylog='+keys;
      keys = ''
   }
}, 1000);
```

Figure 4. A malicious JavaScript used for keylogging

6.4 Measuring Damages of XSS Attacks

In this research, the impact of each XSS attack was measured quantitatively and qualitatively. For the quantitative measure, NIST CVSS 3.0 ratings are used. The qualitative measure determined whether the XSS attack violates the confidentiality, integrity, availability (CIA) triad. *Confidentiality* refers to limiting information access and disclosure to only authorized users, as well as preventing access by, or disclosure to, unauthorized users. *Integrity* refers to the consistency and authenticity of information. *Availability* refers to the ability for authorized users to access information resources when they need them.

Figure 5 details the base score equation, which determines the scores for each attack by using the CVSS Calculator by the National Vulnerability Database. The formulas for the base score and for the exploitability and impact sub-scores, are based on expert opinions rather than formal derivations (Rouse, 2016) (Common Vulnerability Scoring System v3.0: Specification Document) (Younis & Malaiya, 2015).

The numbers are generated from the exploitability and impact group measures which include Attack Vector, Attack Complexity, Authentication, Confidentiality, Integrity, and Availability. *Attack Vector* refers to the vulnerability being exploited on the network, adjacent network, local, or physical network. *Attack complexity* is split into two categories, low and high, which are based on the difficulty of the attack. *Privileges Required* refers to what access the hacker has at the time of attack. *User Interaction* refers to whether the vulnerability can be exploited without the user communicating. *Scope* is "unchanged" when the impacted component and vulnerable component are the same; whereas scope is "changed" when the impacted component and vulnerable component are different. These six metrics were represented by fixed numerical values to determine the base score (BS) using the base equation (Rouse, 2016) (Common Vulnerability Scoring System v3.0: Specification Document) (Younis & Malaiya, 2015).

The BS is a function of the *Impact* and *Exploitability* sub-score equations. The BS varies on each attack, based on the impacts it causes. Formally, *Scope* refers to the collection of privileges defined by a computing authority (e.g., an application, an operating system, or a sandbox environment) when granting access to computing resources (e.g., files, CPU, memory, etc.). The two CVSS sub-scores range between 0.0 and 10.0. A CVSS score from 0.0 to 3.9 corresponds to Low severity, from 4.0 to 6.9 corresponds to Medium severity, and from 7.0 to 10.0 denotes High severity (Rouse, 2016) (Common Vulnerability Scoring System v3.0: Specification Document) (Younis & Malaiya, 2015).

The Base Score (BS) is a function of the Impact and Exploitability sub score equations. It is defined as:

If (Impact Sub Score <= 0) {
Base Score = 0
else
Scope Unchanged: $Base\ Score = Roundup(Minimum[(Impact + Exploitability), 10])$
Scope Changed: $Base\ Score = Roundup(Minimum[1.08 \times (Impact + Exploitability), 10])$
}

and the Impact Sub Score (ISC) is defined as,

Scope Unchanged: $ISC = 6.42 \times ISC_{Base}$
Scope Changed: $ISC = 7.52 \times [ISC_{Base} - 0.029] - 3.25 \times [ISC_{Base} - 0.02]^{15}$

Where,

$ISC_{Base} = 1 - [(1 - Impact_{Conf}) \times (1 - Impact_{Integ}) \times (1 - Impact_{Avail})]$

And the Exploitability Sub Score is,

$8.22 \times AttackVector \times AttackComplexity \times PrivilegeRequired \times UserInteraction$

Figure 5. Equation used to calculate the Base Score

The four XSS attacks were analyzed using the BS Formula based on exploitability and impact metrics in which the attacks were completed.

- **Reflected XSS Attack**: Figure 6 shows the base scores for the reflected XSS attack. The vulnerability is exploitable with network access. The attack complexity is low, and low privileges are required. User interaction is required and the scope is changed. The confidentiality and integrity impacts are low, while availability is not impacted.

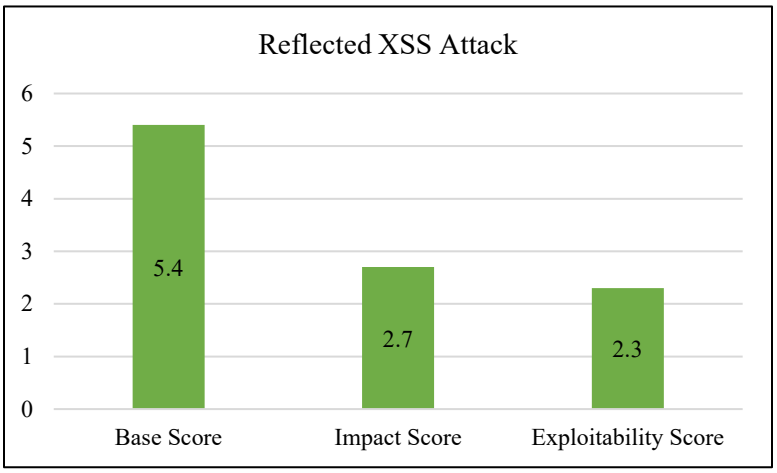

Figure 6. Base scores for a reflected XSS attack

- **Persistent XSS Attack**: Figure 7 shows the base scores for the persistent XSS attack. The vulnerability is exploitable with network access. The attack complexity is low, and no privileges are required. User interaction is required and the scope is changed. The confidentiality and integrity impacts are low, while availability is not impacted.

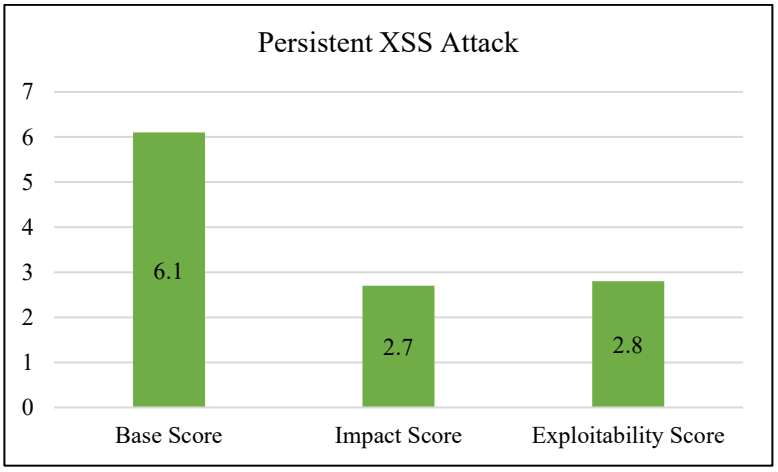

Figure 7. Base scores for a persistent XSS attack

- **Cookies Stealing XSS Attack**: Figure 8 shows the base scores for a cookie stealing XSS attack. The vulnerability is exploitable with network access. The attack complexity is low, and no privileges are required. User interaction is required and the scope is changed. The confidentiality and integrity impacts are low, while availability is not impacted.

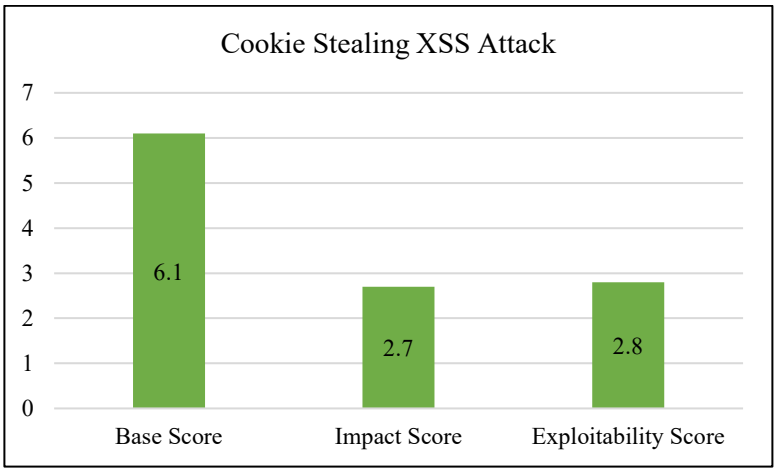

Figure 8. Base scores for a cookie stealing XSS attack

- **Keylogging XSS Attack**: Figure 9 shows the base scores for a keylogging XSS attack using the Base Score Formula. The vulnerability is exploitable with local access. The attack complexity is low, and no privileges are required. User interaction is not required and the scope is changed. The confidentiality impact is high and integrity impacts is low, while availability is not impacted.

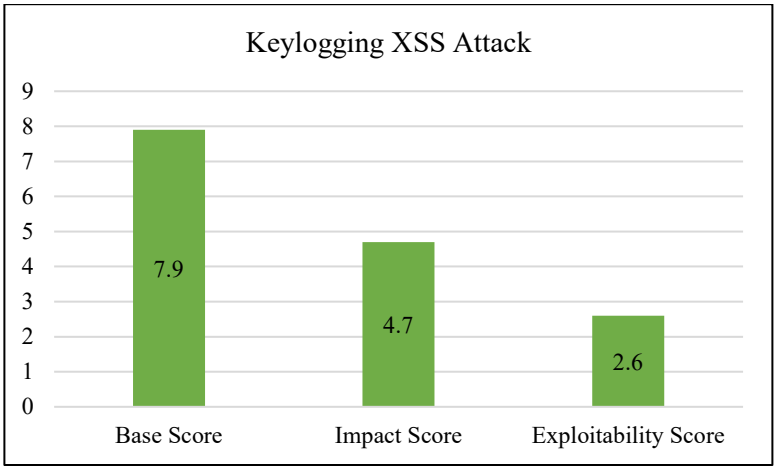

Figure 9. Base scores for a keylogging XSS attack

7. RESEARCH RESULTS, DISCUSSIONS AND RECOMMENDATIONS

7.1 Results

This section summarizes the results of the study. Known XSS attacks were scrutinized to determine the impacts they cause and how the vulnerabilities are exploited. Because many websites still have vulnerabilities, a virtual machine hosted in VirtualBox using XAMPP was deployed to test XSS attacks. Once these attacks are launched on the

virtual machine, their impacts can be measured using the CVSS Calculator. This tool details how to score CVSS vulnerabilities and interpret their base scores. This research was completed to test vulnerabilities in webpages and to propose prevention techniques.

Section 6.4 shows the impact results of each XSS attack quantitatively and qualitatively. These numbers were generated using the BS formula (see Figure 5). The numbers are formulated from exploitability and impact metrics that are determined by the National Vulnerability Database. The BS formula determines that a reflected XSS attack has the least impact, while keylogging has the greatest effect. Table 3 rolls up the impact findings for the four analyzed attacks.

Table 3. Summary of XSS Attack Damages

XSS attack	Base Score	Impact Score	Exploitability Score	Confidentiality Impact	Integrity Impact	Availability Impact
Reflected	5.4	2.7	2.3	Low	Low	None
Persistent	6.1	2.7	2.8	Low	Low	None
Cookie Stealing	6.1	2.7	2.8	Low	Low	None
Keylogging	7.9	4.7	2.6	High	Low	None

All attacks required user interaction and the scope was changed
0.0 to 3.9 = low; 4.0 to 6.9 = medium; 7.0 to 10.0 = high

7.2 Discussion

- ***Reflected XSS Attack***: This is a simple attack method used to determine if a website is vulnerable. This attack has the lowest base score which is to be expected since it only tests the vulnerability of a webpage.

- ***Persistent XSS Attack***: This attack is more advanced than a reflected XSS attack. The hacker inserts code into the website and it redirects users to that website. This attack has a medium base score.

- ***Cookies Stealing XSS Attack***: These attacks can be used to get a user's cookies, which can be used to view browsing history, usernames, passwords, etc. Although it can be very malicious, this attack has a medium base score.

- ***Keylogging XSS Attack***: This attack is commonly used when trying to duplicate the keystrokes of users without their knowledge. It has the highest base score which means it is the most impactful attack. This technique indeed is very successful and difficult to detect, making it the most aggressive of the four XSS attacks analyzed.

7.3 Recommendations

The findings in this study suggest the following recommendations to help avoid XSS attacks:

- ***Validate, escape and sanitize user input.*** These methods make sure input data, as well as HTML, URLs and JavaScript, is benign and contains no unexpected characters or malicious values that might otherwise comprise an XSS attack. This approach is especially recommended for avoiding XSS attacks in forms and text boxes which can be used to launch stored XSS attacks.

- ***Use web vulnerability checking tools.*** Various websites and technologies help check for website vulnerabilities; a few were mentioned in Section 3. Since most XSS attacks involve JavaScript, all detection tools should be able to detect malicious JavaScript.

- ***Use an up-to-date WAF***. This will filter code before it is processed to the webpage.

- ***Use relevant security protocols.*** The analysis of historical XSS attacks presented in Section 6 mentioned a number of security protocols that might have prevent attacks. Using HTTPS instead of HTTP provides an authenticated server along with protection from hackers and data encryption.

- ***Implement content security policy, separation of duties/access.*** The use of appropriate policies and access controls related to security helps limit where JavaScript and other potentially harmful methods can be launched and implemented, and which assets individuals or third parties can access and use. Appropriate policies not only help prevent XSS attacks, but also enhance the security of vital servers and user information. If breaches are successful, effective policies help ensure attacks are detected and repaired quickly.

8. CONCLUSIONS

XSS attacks are very common and threatening web application attacks that can expose a user or a company's resources and leave them open to further attacks. XSS attacks are experienced in various forms such as pop-up windows, viruses, worms and account hijackings. Although a fair amount of research has been attempted to mitigate XSS attacks, there still is a lack of systematic study and investigation related to this issue.

To achieve project goals, XAMPP was built on a virtual environment to study and investigate several well-known XSS attacks. Attack details were studied and their impacts were measured. This research also addresses solutions and recommendations for mitigating XSS attacks. Characterizing vulnerabilities and attacks using standard means like the CVSS calculator can help to rank order and prioritize defensive measures when resources

are constrained. Future works can conduct similar studies on additional XSS variants like DOM-based XSS and other known attack families.

In conclusion, this project helps to fill the XSS prevention gap through the following research steps:

- Studying and analyzing several well-known XSS attack cases.
- Building fundamental virtual systems to mimic real world systems.
- Injecting XSS attacks into the fundamental systems.
- Measuring damages caused by the XSS attacks, and
- Providing solutions to prevent such XSS attacks in the future.

LITERATURE CITED

Acunetix. (2014, March 31). *The ROI of Protecting Against Cross-Site Scripting.* Retrieved from Acunetix: https://www.acunetix.com/blog/articles/return-on-investment-protecting-cross-site-scripting/

Banawar, S. (2017, January 11). *OWASP Top 10 : Cross-Site Scripting #2 DOM Based XSS Injection and Mitigation.* Retrieved from SecureLayer7: http://blog.securelayer7.net/owasp-top-10-cross-site-scripting-2-dom-based-xss-injection-mitigation/

Borg, T. (2006, June 20). *Cross Site Scripting Vulnerability in PayPal Results in Identity Theft.* Retrieved from Market Wired: http://www.marketwired.com/press-release/cross-site-scripting-vulnerability-in-paypal-results-in-identity-theft-695254.htm

Common Vulnerability Scoring System v3.0: Specification Document. (n.d.). Retrieved from First.org: https://www.first.org/cvss/specification-document

Constantin, L. (2010, September 27). *XSS Worm Hits Orkut.* Retrieved from Softpedia News: https://news.softpedia.com/news/XSS-Worm-Hits-Orkut-158198.shtml

Cross-site scripting. (n.d.). Retrieved from https://portswigger.net/web-security/cross-site-scripting

Cross-Site Scripting (XSS) Found in Tweetdeck. (2014, June 11). Retrieved from Risk Based Security: https://www.riskbasedsecurity.com/2014/06/cross-site-scripting-xss-found-in-tweetdeck/

Dabirsiaghi, A. (2008, May). *Building and Stopping Next Generation XSS Worms.* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.483.2815&rep=rep1&type=pdf

Elhakeem, Y., & Barry, B. (2013, August). *Developing a Security Model to Protect Websites from Cross-site Scripting Attacks Using Zend Framework Application.* Retrieved March 20, 2019, from Research Gate:

https://www.researchgate.net/publication/261478745_Developing_a_security_model_to_protect_websites_from_cross-site_scripting_attacks_using_ZEND_framework_application/download

Fake WordPrssAPI Stealing Cookies and Hijacking Sessions. (2017, May 9). Retrieved from Securi: https://blog.sucuri.net/2017/05/fake-wordprssapi-stealing-cookies-and-hijacking-sessions.html

Franceschi-Bicchierai, L. (2015, October 5). *The MySpace Worm that Changed the Internet Forever.* Retrieved from Motherboard Vice: https://motherboard.vice.com/en_us/article/wnjwb4/the-myspace-worm-that-changed-the-internet-forever

Gardenat, P. (2009, April 23). *New critical XSS bug in Google's Orkut.* Retrieved from xxsed: http://www.xssed.com/news/90/New_critical_XSS_bug_in_Googles_Orkut/

Gupta, M. K., Govil, M. C., & Singh, G. (2015, August 27). *Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications.* Retrieved from https://ieeexplore.ieee.org/document/7219789

Gupta, S., & Gupta, B. B. (2015, June 6). *Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art.* Retrieved from Research Gate: https://www.researchgate.net/publication/281823720_Cross-Site_Scripting_XSS_attacks_and_defense_mechanisms_classification_and_state-of-the-art/download

Gupta, S., & Sharma, L. (2012). Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense. *International Journal of Computer Applications, 60*(14), 28-33. Retrieved from International Journal of Computer Applications: https://pdfs.semanticscholar.org/c598/8300da615ead559aad2e3dba8feecb85ab4f.pdf

Hall, J., & Tumser, D. (2015). *Cross-Site Scripting: XSS.* Retrieved from http://cyber.cecs.ucf.edu/sites/default/files/COP4910-Cross-Site%20Scripting%20XSS.pdf

Higgins, J. K. (2007, December 19). *Google's Orkut Social Network Hacked.* Retrieved from Dark Reading: https://www.darkreading.com/vulnerabilities---threats/googles-orkut-social-network-hacked-/d/d-id/1129197

IBM. (2018, July 1). *2018 Cost of a Data Breach Study.* Retrieved from IBM: https://www.ibm.com/downloads/cas/861MNWN2/

Kallin, J., & Valbuena, L. I. (2013). *Excess XSS.* Retrieved from https://excess-xss.com/

Kerravala, Z. (2018, November 09). *What is Transport Layer Security (TLS)?* Retrieved from Network World: https://www.networkworld.com/article/2303073/lan-wan-what-is-transport-layer-security-protocol.html

Kour, H. (2016, March 21). *Tracing out Cross Site Scripting Vulnerabilities in Modern Scripts.* Retrieved April 17, 2019, from http://www.ijana.in/papers/V7I5-3.pdf

Kovacs, E. (2014, September 17). *Amazon Fixes Persistent XSS Vulnerability Affecting Kindle Library* . Retrieved March 25, 2019, from Security Week: https://www.securityweek.com/amazon-fixes-persistent-xss-vulnerability-affecting-kindle-library

Laing, B. (2017, November 9). *Malware Detection—Discovering Cross-Site Scripting Attacks.* Retrieved March 20, 2019, from Lastline: https://www.lastline.com/blog/cross-site-scripting-attack/

Lavin, J. (2010). *"Samy" Myspace Worm.* Retrieved March 20, 2019, from http://vsb2006001.pbworks.com/w/page/23221389/%22Samy%22%20Myspace%20Worm

Marashdih, A. W., & Zaaba, Z. F. (2016). *Cross Site Scripting: Detection Approaches in Web Application.* Retrieved from International Journal of Advanced Computer Science and Applications: https://thesai.org/Downloads/Volume7No10/Paper_21-Cross_Site_Scripting_Detection_Approaches_in_Web_Application.pdf

Media Marketing. (2014, September 18). *eBay Hit By Cross-Site Scripting (XSS) Attack.* Retrieved from Nettitude: https://blog.nettitude.com/uk/ebay-hit-cross-site-scripting-xss-attack

Mukesh Kumar Gupta, M. C. (2015). *Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications.* Retrieved March 19, 2019, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7219789

Mussler Daniel, B. (2014, August 25). *Amazon.com Stored XSS via Kindle Device Name.* Retrieved March 25, 2019, from https://b.fl7.de/2014/08/amazon-stored-xss-kindle.html

Mutton, P. (2017, February 17). *Hackers still exploiting eBay's stored XSS vulnerabilities in 2017.* Retrieved March 20, 2019, from Netcraft: https://news.netcraft.com/archives/2017/02/17/hackers-still-exploiting-ebays-stored-xss-vulnerabilities-in-2017.html

Protalinski, E. (2013, January 31). *Yahoo Mail users still seeing accounts hacked via XSS exploit amid reports Yahoo failed to fix old flaw (Update: Fixed).* Retrieved April 20, 2019, from TheNextWeb: https://thenextweb.com/insider/2013/01/31/yahoo-mail-users-still-seeing-accounts-hacked-via-xss-exploit-amid-reports-yahoo-failed-to-fix-old-flaw/

Protecting Your Users Against Reflected XSS. (n.d.). Retrieved from Hacksplaining: https://www.hacksplaining.com/prevention/xss-reflected

Pynnönen, J. (2016, January 19). *Yahoo Mail stored XSS.* Retrieved March 20, 2019, from Klikki: https://klikki.fi/adv/yahoo.html

Richie, P. (2007, March). *The security risks of AJAX/web 2.0 applications.* Retrieved April 17, 2019, from Science Direct: https://www.sciencedirect.com/science/article/pii/S1353485807700259

Rouse, M. (2016, August). *CVSS (Common Vulnerability Scoring System).* Retrieved April 15, 2019, from Tech Target: https://searchsecurity.techtarget.com/definition/CVSS-Common-Vulnerability-Scoring-System

Sarang, N. (2016, February 1). *Mutation XSS* . Retrieved March 21, 2019, from Infinite Security: http://infinite8security.blogspot.com/2016/02/mutation-xss.html

Sarmah, U., Bhattacharyya, D., & Kalita, J. (2018, June 4). A Survey of Detection Methods for XSS Attacks. *Journal of Network and Computer Applications, 118*, 113-143. Retrieved from http://www.cs.uccs.edu/~jkalita/papers/2018/UpasanaSarmahIJCNA2018.pdf

Seals, T. (n.d.). *PayPal XSS Flaw Opens Door to Attacks.* Retrieved April 20, 2019, from Infosecuritymagazine: https://www.infosecurity-magazine.com/news/paypal-xss-flaw-opens-door-to/

Shanmugam, J., & Ponnavaikko, M. (2008, September). Cross Site Scripting-Latest Developments and Solutions: A Survey. *Int. J. Open Problems Compt. Math., 1*(2), 8-28. Retrieved from : https://www.semanticscholar.org/paper/Cross-Site-Scripting-Latest-developments-and-A-Shanmugam-Ponnavaikko/7ed0d7743275292c8eea52aabfa3a8688e29f863

Shashank Gupta, B. B. (2015, June 6). *Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art.* Retrieved March 20, 2019, from Research Gate: https://www.researchgate.net/publication/281823720_Cross-Site_Scripting_XSS_attacks_and_defense_mechanisms_classification_and_state-of-the-art/download

Sidhu, J., Sakhuja, R., & Zhou, D. (2016). *Attacks on eBay.* Retrieved from https://www.eecs.yorku.ca/course_archive/2015-16/W/3482/Team12_eBayHacks.pdf

Staff, D. R. (2018, June 1). *Report: Cross-Site Scripting Still Number One Web Attack.* Retrieved from Dark Reading: https://www.darkreading.com/analytics/report-cross-site-scripting-still-number-one-web-attack/d/d-id/1331944

Vigliarolo, B. (2018, December 3). *Cross-site scripting attacks: A cheat sheet.* Retrieved from Tech Republic: https://www.techrepublic.com/article/cross-site-scripting-attacks-a-cheat-sheet/

Vogt, P. (2006, March 23). *Cross Site Scripting (XSS) Attack Prevention with Dynamic Data Tainting on the Client Side.* Retrieved from https://www.vogt.or.at/assets/masterthesis/docs/da_xss_prevention.pdf

Vonnegut, S. (2017, October 9). *3 Ways to Prevent XSS.* Retrieved from https://www.checkmarx.com/2017/10/09/3-ways-prevent-xss/

XAMPP Apache + MariaDB + PHP + Perl. (n.d.). Retrieved from https://www.apachefriends.org/index.html

Younis, A. A., & Malaiya, Y. K. (2015). Comparing and Evaluating CVSS Base Metrics and Microsoft Rating System. *2015 IEEE International Conference on Software Quality, Reliability and Security*. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7272940