SECURING WEB APPLICATIONS AGAINST CROSS-SITE SCRIPTING

[1]AMMAR ALDALLAL, [2]KASHIF SHABBIR

[1]Computer Engineering Department, [2]Information Technology Department, Ahlia University, Bahrain
E-mail: [1]aaldallal@ahlia.edu.bh, [2]kashif.sn@gmail.com

Abstract— Using Web applications becomes more popular, and this raises many security threats against them. One of the most common and dangerous threats is the Cross Site-Scripting (XSS). XSS scripts can be executed on web browsers as a result of a request generated by the user and processed by the server. The main objective of this research is to identify if the defensive mechanism against XSS is provided by popular browsers or not. In addition, it proposes a technique to prevent XSS attacks. This technique is based on validating the user input against malicious string used by the attackers. Results show that some browsers have defensive mechanism against simple cross site scripts while complex cross site scripts cannot be blocked by the browsers. Moreover, the proposed technique verifies the feasibility and practicality of the protection mechanism against persistent XSS attacks.

Keywords— Cross-Site Scripting (XSS) Attack, Web Applications, Web Security, Security Threats.

I. INTRODUCTION

Technology has become the foundation of basic human needs. Every individual need to connect to different providers for different needs through Facebook, Instagram, and other technological advances or to need to get some service such as banking, e-commerce or other web applications.

The dynamic web applications developed using innovative technologies are quite complex in nature. Traditional web applications only involved server-side communications but the modern technologies like .NET framework, AJAX, and JavaScript have enabled web developers to design applications that implement client side as well. Using web applications becomes more and more popular on a daily basis, and this motivates the hackers to commit cyber-crimes such as identity theft. It is argued that this connectivity has raised a major security threat since attacker will be able to access personal and sensitive information. These advanced services are provided using some kind of web or mobile application. [1].

[2] Explained that threats to the web applications include un-validated redirects and forward, component vulnerabilities, cross-site forgery, and missing function-level access control. The threats contain the other top threats which are listed by Open Web Application Security Project (OWASP). These threats pose a great security breach in terms of infrastructure and personal damage. Analyzing and trying to minimize or prevent these types of attacks is the major focus of several researchers. The Open Web Application Security Project (OWASP) is a non-profit organization working for enabling organizations to implement, buy, and maintain trustworthy applications. The OWASP Top 10 list contains the most vulnerable risks from information security perspective. Detailed information about each of these threats is available on the official website of OWASP [3].

Many security threats against Web Applications are investigated by researchers. One of the most common and dangerous threat is the Cross Site-Scripting (XSS). XSS scripts can be executed on web browsers as a result of request generated by the user and processed by the server. The main objectives of this research are to identify if defensive mechanism against XSS is provided by different browsers and to propose data validation mechanism against XSS attacks.

The rest of this paper is organized as follows: Section two provides background on the problems faced by XSS and how previous researchers have acknowledged the need to study this threat in details and proposed their solutions. Section three explains one type of XSS attack which is persistent XSS attack and provides example of it. Section four shows how different browsers reacts against XSS scripts which are executed on them and also provides information on how XSS attacks are successful and what type of dangers can XSS possess. The techniques proposed to prevent XSS are explained in Section five. Finally, conclusion and recommendations for future works are presented in Section six.

II. RELATED WORK

Various works in the literature have been analyzed for XSS attacks. Some may have critical gaps that can be filled through this research in conjunction with other resources. Web applications are prone to web attacks. Lots of work has been done in the literature to find a solution for this problem. The following highlights the summary of the work done by various researchers in the field of Web Application Security.

Since computers have been prone to cyber abuse since the beginning of the internet services, extensive research has been made to prevent the online malicious activities. [4] Focused on the effective measures such as VPN usage, access and

authentication control, and error page cleaning to create a secure platform that can prevent cyber-attacks.

[5] Have investigated the fact that web applications have opened the paths for organizational vulnerabilities. The research is based on evaluation of dependability attributes using Vulnerability Assessment Tools. There are six dependability measures, and the study involves the techniques to evaluate them. The study also focused on the dangers of ignoring the dependability attributes while developing new software products.

[6] Conducted a research on the faults in the web systems which cause vulnerabilities. Almost every web application has some faults, or bugs, which contributes to the organized cyber-crimes. Their research reviewed two programmed scripts of security patches of various web applications, which are: SQL Injection and XSS. In addition, it explained how hackers take advantage of these bugs to enter the system for malicious activities.

The topmost website vulnerability is Cross-Site Request Forgery (CSRF) as per [7]. It is used by hackers for their cyber-crimes. The research focuses on a special case of CSRF, known as Login CSRF. Login CSRF uses a cross site script to access the login form of a website, and forge the user into attempt for login. Three major prevention mechanisms against such attacks are proposed along with their pros and cons. HTTP referrer can be helpful in identifying the attacks but the results indicate that it is difficult to analyze the HTTP referrer for privacy concerns. The results provide the HTTP headers as a temporal alternative for detection of CSRF Login attacks. [8] Also analyzed CSRF attacks and the standard defensive procedures applied against them. Multiple mechanisms are reviewed, and it is found that the most powerful techniques are the CSRF guard and CSRF detector.

Ponnavaikko and Shanmugam have stated that more than 80% attacks on websites are using cross-site scripting. Web developers have to build user friendly environments for users while keeping in mind that the features are not misused by the hackers for their malicious activities. The authors have gathered around 2800 vulnerable Cross Site Scripting (XSS) websites, which were used for detailed study on the subject to draw conclusions. The research indicated that XSS is used for introducing worms into the system. The surveyed solutions are categorized based on attributes like location, analysis type, technique, and intrusion detection [9].

Moodle is a popular open source software system that provides users with virtual learning environment for e-learning purposes. From security perspective, Moodle often falls prey to cyber-crime vulnerabilities, which make it important to implement solutions that address its security bugs. [10] Addressed the details that most commonly cause flaws in the system, and the best possible prevention mechanisms against them. The authors categorized the authentication vulnerability into session attack and design attack and then proposed ways for effectively learn how to mitigate security threats and risks.

The authors of [11] have shown that the major concern for any web application is its security aspect. They discussed the two types of XSS vulnerabilities called Persistent XSS attacks and Non-Persistent XSS attacks. They explained how these two types of threats can be harmful to a web application if not taken care of properly.

Since attack (XSS for short) is usually used to steal the cookies from a browser's database [12] proposed a technique which is called "Dynamic Cookies Rewriting". This technique rewrite the name attribute of the cookie with the randomized value then send it to the browser. This value is kept in its database instead of the original value sent by the web server. Because the original values of the cookies are not stored in the browser's database, the cookies cannot be used later to impersonate the users by XSS attackers.

The mentioned literature reviews different security threats against web based applications. Each has solved the XSS from certain perspective. However, more information is needed to be able to evaluate their significance. Protection from intruders practicing cross-site scripting is a gap in the various researches that needs to be filled. Researches listed above provided information regarding detection and possible ways to solve each related risk or problems but could have discussed more specific ways in preventing and alleviating issues such as vulnerability to XSS attacks. Proper coding and security practices should also be discussed more thoroughly. These endeavors need to be done to avoid such attacks. This paper would fill part of this gap. It would provide more technical and detailed prevention techniques to mitigate cross-site scripting attacks that target the codes used to build a website or a web application.

III. PERSISTENT XSS ATTACK

The Cross-site scripting attacks compromise the relationship of trust between a user and a website. Since the main idea behind a successful XSS attack is the lack of proper coding, programmers need to practice writing proper coding by strictly following the agile software development methodologies.

Among three well known types of XSS, which are Persistent XSS, Non- Persistent XSS and DOM-based XSS, this work concentrates on Persistent XSS Attack. It can be summarized in the following steps:

- The attacker uses one of the website's forms to insert a malicious string into the website's database.
- The victim from his personal computer unknowingly requests the infected page from the website.

- The website includes the malicious string from the database in the response and sends it to the victim.
- The victim's browser executes the malicious script inside the response, sending the victim's cookie to the attacker's server. These steps are illustrated in **Fig. 1.**

IV. TESTING FOR XSS VULNERABILITY

Three browsers are used to examine the effects of cross site scripting on a simple webpage containing one input field and a submit button. These browsers are: Internet Explorer, Firefox and Google chrome. The response of these browsers to the XSS attack is tested using the following script:

<script> alert ("XSS WORKS!") ;</script> (1)

Script (1) is executed by the browser without providing any built-in prevention mechanism against XSS attacks. In this case the only prevention against XSS can be achieved using the user input validation at client side as well as the server side which will be demonstrated later.

4.1. Browser Responses against XSS Attacks

Script (1) is executed in three browsers which are Internet Explorer v8, Firefox v45 and Google Chrome v49. The script is injected in the input field provided by the website application and submitted to the server. A simple webpage is created using PHP that contains an input text field and a submit button. The page is loaded in the above mentioned browsers and Script (1) is executed. The following results are obtained in each browser.

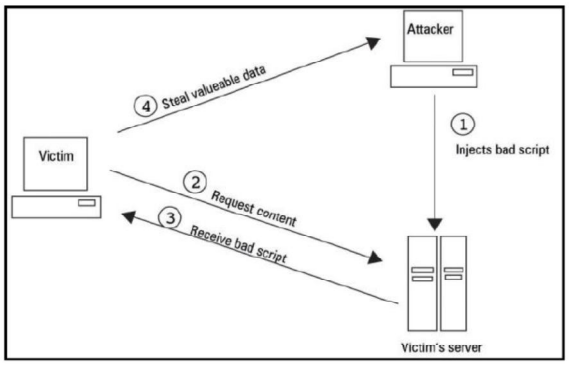

Fig.1. Persistent XSS [13]

Internet Explorer is one of the widely used web browsers. When executing Script (1), Internet Explorer generates a warning stating that the browser has modified the page to prevent XSS. When executing same script on Google Chrome it fails to run because it found the source code within the request. By trying to execute same script on Mozilla Firefox, the browser fails to execute and does not return any value. However, if the XSS script is modified to be: <scipt> alert (0) ;</script>, and used instead of the one in (1), the attack becomes successful. Firefox tries to prevent XSS attacks by encoding the quotation marks with the ASCII character code in order to prevent the script from execution.

All browsers provide some form of protection against basic XSS attacks. However, using complex XSS scripts such as encoded text cannot be blocked by browsers; instead, the web application has to rely on the developers programming capability and validation.

4.2. Simulating Persistent XSS Attacks

Cross-site scripting (XSS) is an attack technique where the attacker runs executable code through a website running in user's browser. The code itself is usually written in HTML/JavaScript, but VBScript, ActiveX, Java, Flash or any other Browser-supported technology can also be used. If an attacker succeeds, the code in the browser is executed in the security context (or zone) of the visited site. The code is able to read, edit and send sensitive data over the browser. Cross-site scripting attacks can be directed to a specific user by forging a URL or by sending an email to the victim or directing the user to a specific section of the website. Moreover, the victim can also be targeted easily using the URL shortening services

In this research a website representing a forum is developed using PHP and MYSQL. This site is used to demonstrate the effects of XSS when a successful XSS attack has been carried out. Also the same website will be implemented using a proper data validation where a controlled user input will be designed to demonstrate how better programming practices and thorough data validation which provides better security against XSS.

Persistent XSS Attacks are stored on the vulnerable website server. These attacks are not directed to any specific victim; instead, they can be executed directly without the knowledge of the user. The Following experiments demonstrate different types of stored XSS attacks executed in a forum. The XSS vulnerability is tested by executing the script in (1). Once the modified script is posted, the vulnerability is revealed. If the browser executes the script on the client side, an alert box is generated which displays the proposed messaged "XSS". This clearly states that the website is vulnerable to XSS attack and JavaScript can be executed on the server. Next step is to steal the user session and cookies for any victim who will use the forum and the post will provide a link to another website which will direct the victim to.

As an example, on clicking the link shown in **Fig.2** the user is directed to the website which claims to be from outlook.com but in fact it is from another malicious looking website which resembles the outlook.com website. In fact it is redirecting the victim to "*http://www.prinkfashions.com/sigin msn. him"* page where a novice victim is unfamiliar with this type of attacks thinking that it is a real login page

and he will enter his credentials, therefore compromising his credentials**.** After entering his credentials, the victim is redirected to any other bogus website or can be redirected to the original login page of authentic website.

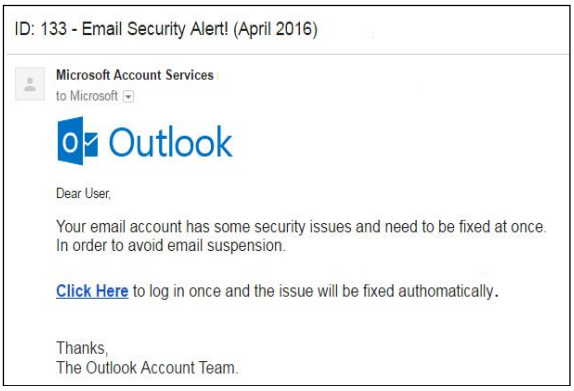

Fig.1. XSS Redirection Attack

V. XSS PROPOSED DETECTION AND VALIDATION TECHNIQUE

As mentioned earlier, XSS attacks are the result of lack of input validation. This section proposes input validation and XSS detection techniques to minimize the XSS attacks.

The following sections explain the proposed methods and functions to be used to defend against XSS attacks. Although these methods do not provide a 100% protection from XSS attacks, applying the following procedures can greatly reduces XSS attacks and in some cases prevents any serious damage.

5.1. HTML5 and Data validation

HTML5 is recommended by the World Wide Web Consortium since 2014. It contains newer and more interesting elements for web application design control such as header, footer, article and section. In addition, HTML5 provides form elements such as textbox, password and email with new attributes such as required and pattern. Using the *required* attribute, HTML5 forces the user to enter some text into the provided field. Moreover, using *pattern* attribute together with required attribute provides an extra protection against XSS attacks. Patterns are required as a regular expression string to match the type of input. For example, *Name* field can only contain string of alphabets followed by one space. The following is an example of the *required* and *Pattern*: <input type=”text” value=”” required pattern= ”[A-Z|a-z]{3,}*” name =”searchstring” title=”Firstname SecondName LastName” />.

This example shows an HTML5 input element of type text, *values* are required and the pattern of capital letters or small letters followed by a space need to be used. If the pattern is not matching the defined string and if the special characters such as “, “or “< “or ASCII value are inserted, HTML5 generates an error message requesting the user to enter correct format. HTML5 input element can contain *title* attribute. *Title* attribute provides hints to the users on what type of information is required in the input element along with its required format. When the attacker tries to inject a script in the *name* text box, the attacker is shown a warning message including the type of input format that is required.

However, using HTML5 safety features does not guarantee that the web application is safe against XSS attacks; instead, it is just forming a first line of defense. In case the attacker manages to bypass HTML5 validation, the web application is required to validate the data before sending it to the server and this is illustrated in the next section.

5.2. Client Side Validation

User input requires that the user enters a value confined a specific data set. For example, name consists of first name, middle name and last name. Similarly, phone number follows a specific pattern of country code followed by area code then the phone number. Fixed input can be restricted to its specific domain and the programmers can set standards or use readymade data validation libraries such as JQuery. JQuery comes with a pre-built powerful data validation classes allowing the developers to easily validate user input before passing it to the server.

Validating user input in the case of a blog or a comment box is hard because the user can use special characters such as quotation marks to emphasize a keyword or use dangerous character or symbol such as “!”, “<” or “>”. The safe way for a web application to handle these special characters is to encode them. Encoding converts the special characters into its equivalent ASCII value. HTML tags should be encoded before they are saved in the database at the server or retrieved from the database and displayed to the user. Encoding string or special characters prevents the browser from executing malicious code.

5.3. Server Side Validation

Once the user has submitted the form the server receives the information sent by the HTTP header. The server then extracts each variable and processes it as required. The server mostly takes the data and executes SQL queries.

Each attribute or value submitted has to be validated again at the server side. Data submitted by the user should never be trusted and must be validated. The validation can be done by matching each form element value against its expected pattern required by the database field. If there are any special characters that need to be converted into ASCII notation, the database query should use specific functions such as mysql_real_escape_string() method to escape any special characters that can result in SQL injection attacks.

To provide protection against XSS attacks, proper data validation and input handling should be used. Writing validation classes and methods is a time

consuming effort, instead frameworks can be used which provides data validation methods built in it. No matter the system is being built, using a proper framework can save time for developers by relying on the validation functions provided by the framework. Single and double quotes are one of the most dangerous characters and when encoded improperly properly XSS attacks occur. Single and Double Quotations can be properly encoded as the following: htmlentity (maliciousScipt, EN_QUOTES | ENT_HTML5, 'UTF-8'). This script converts the quotations into equivalent HTML entities before displaying it on the client side. Similarly, a developed method: *XSS_CLEAN()* is used to clean the output text from any malicious codes. This method takes the user data and matches for special characters that can result in XSS attacks, searches for these patterns available in *$data* variable and replaces them with ASCII equivalent text. Its code is shown in **Fig.4.**

Another function used to render useless script tags is to use cleanInput() function as defined in **Fig.3**. This method cleans the user input from all malicious codes such as JavaScript, HTML tags, any CSS style tags and comments.

```
function cleanInput($input) {

  $search = array(
    '@<script[^>]*?>.*?</script>@si',    // Strip out javascript
    '@<[\/\!]*?[^<>]*?>@si',             // Strip out HTML tags
    '@<style[^>]*?>.*?</style>@siU',     // Strip style tags properly
    '@<![\s\S]*?--[ \t\n\r]*>@'          // Strip multi-line comments
  );

    $output = preg_replace($search, '', $input);
    return $output;
  }
```

Fig.3. CleanInput Method

CONCLUSIONS

Cross site scripting poses a great threat to web applications since it runs malicious code on the server and returns back with sensitive information. Validating user input is one way to prevent XSS attacks but is not guaranteed as the browsers may be infected with malicious malware.

XSS attacks can be minimized by following proper programming and data validation process. This research proposed proper data validation and encoding to the scripts to convert them into HTML entities to notify the server and the browser that the user input is to be treated as a text and not as a code.

XSS attacks can be fairly reduced by properly validating the user input according to the requirements and converting any special characters into equivalent HTML entities. Converting into entities preserves the significance of the special character.

On the server side, data validation can make use of Object Oriented (OO) software development methodology. Using OO user input can be defined as classes and restrict the variables to their domain such as string, integer, float etc. and at the same time keeping the length of the user data provided under control by specifying the maximum number of characters a string can accept thus adding additional security against malicious user input.

Future work could be to assess the effectiveness of popular frameworks currently being used for web application development. Performing analysis on what security measures are provided by these frameworks against XSS attacks and evaluating their efficiency against detection of XSS attacks. Many web application development frameworks exist such as Codeigniter, CakePHP, and Laravel, etc. All these frameworks are based on Model View Controller pattern (MVC). These frameworks provide some form of filtering mechanism against XSS which need to be evaluated. Most MVC frameworks come with built-in methods which can be used to clean the data from any malicious codes inserted by the user. However, these methods need to be analyzed to come up with a simpler and faster mechanism that can be used across all platforms.

```
function xss_clean($data)
{
// Fix &entity\n;
$data = str_replace(array('&amp;','&lt;','&gt;'), array('&amp;amp;','&amp;lt;','&amp;gt;'), $data);
$data = preg_replace('/(&#*\w+)[\x00-\x20]+;/u', '$1;', $data);
$data = preg_replace('/(&#x*[0-9A-F]+);*/iu', '$1;', $data);
$data = html_entity_decode($data, ENT_COMPAT, 'UTF-8');

// Remove any attribute starting with "on" or xmlns
$data = preg_replace('#(<[^>]+?[\x00-\x20"\'])(?:on|xmlns)[^>]*+>#iu', '$1>', $data);

// Remove javascript: and vbscript: protocols
$data = preg_replace(
'#([a-z]*)[\x00-\x20]*=[\x00-\x20]*([`\'"]*)[\x00-\x20]*j[\x00-\x20]*a[\x00-\x20]*v[\x00-\x20]*a[\x00-\x20]*s[\x00-
\x20]*c[\x00-\x20]*r[\x00-\x20]*i[\x00-\x20]*p[\x00-\x20]*t[\x00-\x20]*:#iu', '$1=$2nojavascript...', $data);
$data = preg_replace(
'#([a-z]*)[\x00-\x20]*=([\'"]*)[\x00-\x20]*v[\x00-\x20]*b[\x00-\x20]*s[\x00-\x20]*c[\x00-\x20]*r[\x00-\x20]*i[\x00-
\x20]*p[\x00-\x20]*t[\x00-\x20]*:#iu', '$1=$2novbscript...', $data);
$data = preg_replace('#([a-z]*)[\x00-\x20]*=([\'"]*)[\x00-\x20]*-moz-binding[\x00-\x20]*:#u',
'$1=$2nomozbinding...', $data);

// Only works in IE: <span style="width: expression(alert('Ping!'));"></span>
$data = preg_replace('#(<[^>]+?)style[\x00-\x20]*=[\x00-\x20]*[`\'"]*.*?expression[\x00-\x20]*\([^>]*+>#i', '$1>',
 $data);
$data = preg_replace('#(<[^>]+?)style[\x00-\x20]*=[\x00-\x20]*[`\'"]*.*?behaviour[\x00-\x20]*\([^>]*+>#i', '$1>',
$data);
```

Fig.4. XSS_CLEAN code

REFERENCES

[1] Barth, A., Jackson, C., & Mitchell, J. (2008). 'Robust defenses for cross-site request forgery'. Proceedings of the 15th ACM conference on Computer and communications security, pp. 75-88. ACM.

[2] Gupta, B. B., Gupta, S., Gangwar, S., Kumar, M., & Meena, P. K. (2015). 'Cross-site scripting (XSS) abuse and defense: Exploitation on several testing bed environments and its defense'. Journal of Information Privacy and Security, 11(2): 118-136.

[3] Ionescu, P. (2015). The 10 Most Common Application Attacks in Action Retrieved from:

[4] <https://securityintelligence.com/the-10-most-common-application-attacks-in-action/> [Accessed on 14th February, 2016].

[5] OWASP Foundation, 2002, Top 10 2013-Top 10, Accessed from:

[6] <https://www.owasp.org/index.php/Top_10_2013-Top_10> [Accessed on 15th February, 2016]

[7] Brain, R. (2015). Preventing and detecting security vulnerabilities in Web applications. Retrieved from <http://www.computerweekly.com/tip/Preventing-and-detecting-security-vulnerabilities-in-Web-applications> [Accessed on 10th February, 2016].

[8] Kahtan, H, Bakar, N, Nordin, R, & Abdulgabber, M 2014, 'Evaluation Dependability Attributes of Web Application using Vulnerability Assessments Tools', Information Technology Journal, 13(14): 2240-2249

[9] Fonseca, J, Seixas, N, Vieira, M, & Madeira, H 2014, 'Analysis of Field Data on Web Security Vulnerabilities', IEEE Transactions On Dependable & Secure Computing, 11(2): 89-100

[10] Barth, A., Jackson, C., & Mitchell, J. (2008). 'Robust defenses for cross-site request forgery'. Proceedings of the 15th ACM conference on Computer and communications security, pp. 75-88. ACM.

[11] Kombade, R., and Meshram, B. 2012. 'CSRF Vulnerabilities and Defensive Techniques'. International Journal Computer Network and Information Security, 1: 31-37.

[12] Putthacharoen R. and Bunyatnoparat P., (2011) "Protecting cookies from Cross Site Script attacks using Dynamic Cookies Rewriting technique," *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, Seoul,, pp. 1090-1094.

[13] Kumar, S., and Dutta, K. 2011, 'Investigation On Security In Lms Moodle', International Journal of Information Technology and Knowledge Management, 4(1): 233-238.

[14] Garcia-Alfaro, Joaquin and Guillermo Navarro-Arribas. "A Survey On Detection Techniques To Prevent Cross-Site Scripting Attacks On Current Web Applications". Critical Information Infrastructures Security (2008): 287-298.

[15] Jovanovic, N., Kirda, E. and Kruegel, C., 2006, August. 'Preventing cross site request forgery attacks'. In Securecomm and Workshops, 2006. pp: 1-10. IEEE.

[16] http://opensourceforu.com/2010/09/securing-apache-part-2-xss-injections/ [Accessed 14 Feb. 2016].

★★★