A SUBSTITUTION BASED ENCODING SCHEME TO MITIGATE CROSS SITE SCRIPT VULNERABILITIES

By

BHARTI NAGPAL * NARESH CHAUHAN ** NANHAY SINGH ***

Assistant Professor, Department of Computer Science Engineering, Ambedkar Institute of Advanced Communication Technology and Research (AIACT&R), Delhi, India.
*** Chairman and Professor, YMCA University of Science and Technology, Faridabad, Haryana, India.*
**** Associate Professor, Department of Computer Science Engineering, Ambedkar Institute of Advanced Communication Technology and Research (AIACT&R), Delhi, India.*

ABSTRACT

Most of the attacks made on the web, target the vulnerability of web applications. These vulnerabilities are researched and analyzed at OWASP [1]. The Open Web Application Security project, OWASP, tracks the most common failures. Cross Site Scripting (XSS) is one of the worst vulnerabilities that allow malicious attacks such as cookie thefts and web page defacements. Testing an implementation against XSS vulnerabilities can avoid these consequences. Obtaining an adequate test data set is essential for testing of XSS vulnerabilities. These inputs are interpreted by browsers while rendering web pages. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting website. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. Cross-site scripting attacks essentially compromise the trust relationship between a user and the website. XSS occurs when a web page displays user input typically via JavaScript that is not properly validated. This paper uses an encoding scheme that scans the starting tag present in a HTML tag and encodes it such that, the script written inside the starting and closing tags will not work as a HTML element thus, rendering the attack useless.

Keywords: XSS Attack, Vulnerability, XSS Types, Prevention.

INTRODUCTION

Cross-site Scripting (XSS) is a technique that is used by the attacker to supply code into a user's browser instance. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. A Cross-site scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the website they are visiting. As a result, the intended behavior of generated web pages alters through visible (e.g., creation of pop-up windows) and invisible (e.g., cookie bypassing) symptoms. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. Cross-site Scripting allows a hacker to insert malicious JavaScript, VBScript, ActiveX, HTML, or Flash into a dynamic web page which is vulnerable. The purpose of executing the script is to collect confidential data. The hacker gathers private information as well as manipulates or steal cookies.

1. Types of XSS

There are three types of XSS attacks which are mentioned below:

- Non-persistent
- Persistent
- DOM-based

1.1 Non-persistent

Non-persistent attacks, probably the most common type of cross-site scripting exploit is the reflected exploit. It targets vulnerabilities that occur in some websites when data submitted by the client is immediately processed by the server to generate results that are sent back to the browser on the client system. An exploit is successful if it can send code to the server that is included in the web page results sent back to the browser, and when those results are sent, the code is not encoded using HTML special character encoding thus being interpreted by the browser rather than being displayed as inert visible.

1.2 Persistent

Persistent attacks occur when the malicious code is submitted to a website where it is stored for a period of time. Examples of an attacker's favorite target often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.

1.3 DOM-based

DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. It is a special variant of reflected XSS, where logic errors in legitimate JavaScript and careless usage of client-side data result in XSS conditions. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.

2. Vulnerabilities

Cross Site Scripting can be used to Steal cookies, which is also known as cookie hijacking, redirecting the users to different websites, displaying completely different content on your website, performing port scans of the customer's internal network which may lead to full intrusion attempt, denting the reputation and goodwill of the organization, can lead to huge penalty amount which can affect the continuity of the business and of course steal sensitive user data. Cross-Site Scripting stems from the notion that, a malicious website has the ability to load another website into another frame or window. This is accomplished by JavaScript which is used to read or write data on the other website.

For example:

- The following query when injected in the login form of the website, the results will be recorded as follows-

<ScRipt>ALeRt("XSS");</sCRipT>

This technique is also called Obfuscation. It is used when there is no proper filter for filtering different use case letters in the script. The starting and closing tags are converted to &lt and &gt respectively making them unusable as HTML tags and gives the following output:

<ScRipt>ALeRt("XSS");</sCRipT>

- The following query when injected in the login form of the website and the results will be recorded.

<script><IMG SRC="javascript:alert('XSS');"></script>

Image XSS using the JavaScript directive is also used as an XSS attack. The starting and closing tags are converted to &lt and &gt respectively making them unusable as HTML tags and gives the following output:

<script><IMG SRC="javascript:alert('XSS');"></script>

3. Related Work

There has been a plenty of research going on in order to detect and prevent cross site scripting attacks. Following are some detection and prevention mechanism proposed by the researchers.

3.1 "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing" (Fabien Duchene, Roland Groz, Sanjay Rawat, and Jean-Luc Richier) (2012)

[2] In this paper, they have proposed an approach to detect web injection vulnerabilities by generating test input using a combination of model inference and evolutionary fuzzing. Model inference is used to obtain a knowledge about the application behavior. Here inputs are generated using GA (Genetic Algorithm). GA uses the learned formal model to automatically generate input with better fitness values towards triggering an instance of the given vulnerability. They have proposed an

automated type-1 XSS search technique which is based on model inference and evolutionary fuzzing to generate test cases.

3.2 "Model Checking for the Defense against Cross-site Scripting Attacks" (Yu Sun, and Dake He) (2012)

[3] In this paper, they have proposed a model checking method for the defense against Cross-site Scripting attacks. Bugs of the e-commercial website are found and counter examples are shown by model checking. The automatic modeling algorithm for the HTML code is proposed and the case of the performance of the algorithm is presented.

3.3 "Mining Input Sanitization Patterns for Predicting SQL Injection and Cross Site Scripting Vulnerabilities" (Lwin Khin Shar and Hee Beng Kuan Tan) (2012)

[4] In this paper, they have proposed various input sanitization methods and proposed a set of static code attributes. They have used data mining method to predict SQL injection and cross site scripting vulnerabilities in web applications. Here the classification schemes are based on CFG (Control Flow Graph) of a web application program. They implemented a proof-of-concept tool called PhpMiner. It is used to extract the data and attributes from PHP programs.

3.4 "Defending against Cross-Site Scripting Attacks" (Lwin Khin Shar and Hee Beng Kuan Tan) (2012)

[5] The authors have proposed various XSS exploit techniques that are similar to SQL injection, an original form of code injection. This type of attack exploits an application's output function that references poorly sanitized user input. They have also proposed various types of XSS Defense Techniques like (defensive coding practices, XSS testing, vulnerability detection, and runtime attack prevention.

3.5 "A Cross Platform Intrusion Detection System using Inter Server Communication Technique" (Ms. R. Priyadarshini, D. Jagadiswaree, A. Fareedha, and M. Janarthanan) (2011)

[6] In this paper an IDS sever has been developed which analyse and detect the input interaction from the web application via an API (Application Programming Interface) using Curl library and identifies whether the intrusion occurred or not and prevents it from attacking the web application. Web application tested in this paper is message application developed in the programming language like .NET, PHP, JSP etc. and sends its input file to IDS Server via inter server communication mechanism and thereby detects and prevents the intrusion. In addition to this, a log file is generated over which the behavioural pattern is analysed and plotted to generate the report dynamically.

3.6 "Security Testing of Web Applications: A Search Based Approach for Cross-Site Scripting vulnerabilities" (Andrea Avancini and Mariano Ceccato) (2011)

[7] In this paper, they proposed a search based approach for security testing of web applications. They took advantage of static analysis to detect candidate cross-site scripting vulnerabilities. Input values that expose these vulnerabilities are searched by a genetic algorithm and, to help the genetic algorithm escape local optima, symbolic constraints are collected at run-time and passed to a solver. Search results represent test cases to be used by software developers to understand and fix security problems. They implemented this approach in a prototype and evaluated it on a real world PHP code.

3.7 "Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique" (Rattipong Putthacharoen and Pratheep Bunyatnoparat) (2011)

[8] This approach aims to change the cookies in such a way that, they will become useless for XSS attacks. This technique is called "Dynamic Cookie Rewriting" implemented in a web proxy where, it will automatically replace the cookies with the randomized value before sending the cookie to the browser. In this way, the browser will keep the randomized value instead of original value that is sent by the web browser. At the web server end, the return cookie from the browser again is rewritten to its original form at the web proxy before being forwarded to the web server. So in case if XSS attacks steal the cookies from the browser's database, the cookies cannot be used by the attacker to impersonate the users. This technique is not tested with HTTPs connections.

4. Proposed Solution

In this paper, the authors proposed a substitution based encoding scheme to mitigate cross site scripting attack. The effectiveness of the proposed method is tested and validated to mitigate XSS attack. The proposed solution is explained below.

4.1 Flowchart

The prevention of XSS attack can be explained with the help of flowchart as mentioned below in Figure1.

The attack starts when a user injects the string in the text box. This input is scanned for special characters (like <,>,&,'and "). If a character is found, then it is encoded and if not the string is stored in the database and the specified string is executed on the website.

4.2 Algorithm

The proposed algorithm to prevent XSS attack is shown in Figure 2.

If a web page is vulnerable to XSS attack, the attacker can inject his code in the comment area. If there is any HTML tag present in the string, then the browser will render it as HTML tags and the script tag is executed. By using this algorithm, whenever the user enters some text and press submit, the text is then analyzed character by character and if there exist any HTML tag, then it is encoded like '<' is encoded to '&lt' '>' is encoded to '&gt' and so on. The string entered by the user is the input. Every character of input string is checked whether it is a HTML tag or not, if it is so, then the character is replaced by its corresponding encoded tag and if it is not a HTML tag the character is placed as it is. The new created string i.e. encoded String is sent to the database.

5. Implementation

If a web page is vulnerable to XSS attack, the attacker can inject his code in a search bar or at the url. When checking for vulnerability, the attacker injects the search bar or url with a script tag.

For example :

<script>alert ("Check for Vulnerability"); </script>

If the browser responds with a pop-up with the message inside the alert, this means that the web page is vulnerable to XSS attack and the attacker may use different attacks for stealing sensitive data accessible only to the admin of

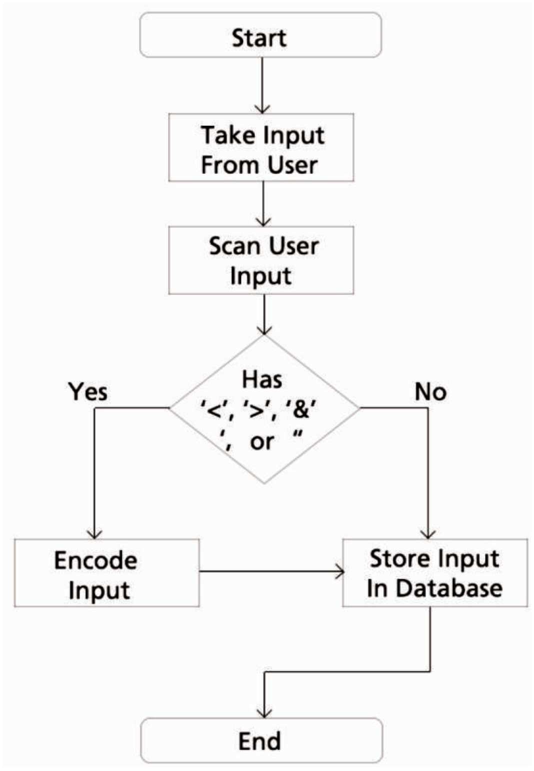

Figure1. Flow Chart to Prevent XSS Attack

```
functionencode Input{
int i=0;
charencodedInput [500];
while(input[ i]!='/0'){
if(input[ i]=='<')
Concatenate  encodedInput  with '&lt';
Else if(input[ i]=='>')
Concatenate  encodedInput  with '&gt';
Else if(input[ i]=='&')
Concatenate  encodedInput  with '&amp';
Else if(input[ i]==' " ')
Concatenate  encodedInput  with '&quot';
Else if(input[ i]==' ' ')
Concatenate  encodedInput  with '&apos';
Else
Concatenate  encodedInput  with input[ i];
}
returnencodedInput ;
}
```

Figure 2. Algorithm to Prevent XSS Attack

Query	Detected	Prevented
<script>alert("XSS");</script>	Yes	Yes
<script>String. from Char Code (97, 108, 101, 114, 116, 40, 34, 88, 83, 83, 34, 41)</script>	Yes	Yes
<script><IMG SRC="javascript:alert('XSS);"></script>	Yes	Yes
<ScRipt>ALeRt("XSS");</sCRipT>	Yes	Yes
<ScRipt>ALeRt("XSS");</sCRipT>	Yes	Yes

Table1. Experimental Analysis of XSS Queries

Moneycontrol.com	<script>alert(/Moneycontrol.com is Vulnerable/);</script>	Site was vulnerable	
Deephousepage. com	<script>String.fromCharCode (97, 108, 101, 114, 116, 40, 34, 88, 83, 83, 34, 41)</script>	Site was vulnerable	
Dogspot.in	<script><IMG SRC="javascript:alert('Dogspot.in is Vulnerable');"></script>	Site was vulnerable	
Bankgorodov.ru	<script><iframe/onload=alert (/XSS/)</script>	Site was vulnerable	
Sidereel.com	<script>alert("XSS");</script>	Site was vulnerable	

Table 2. Experimental Analysis of XSS Queries on Different Websites

the web page. These attacks can be prevented using the encoding scheme which will change the meaning of the injected script by the attacker. If there is any HTML tag present in the string, then the browser will render it as HTML tag and the script tag is executed.

For example :

Consider the following text,

<i>Computer

The above text will be printed in italics.

Now if we substitute the starting tag '<' and closing tag '>' with a replacement &lt and &gt respectively, the script inside the tags will be rendered useless.

For example, if an attacker injects his code in the login form of the website like:

<script>alert ("Check for Vulnerability");</script>

The browser will read the above query as:

&ltscript& gtalert ("Check for Vulnerability");&lt/script&gt

And hence the output would be:

<script>alert ("Check for Vulnerability");</script>

6. Experimental Analysis

Table 1 shows the experimental analysis of various XSS queries.

Table 2 shows the experimental analysis of various XSS queries on different websites:

Conclusion

A variety of programming practice guidelines and web application security testing tools and scanners have been proposed by the research community to detect and prevent XSS attack. Inspite of implementing the known preventive techniques, attackers are still able to successfully perform XSS attack on web applications and get access to the confidential user information.

The proposed approach prevents the attacker from attacking vulnerable websites by encoding the incoming input given by the user. The efficiency of the proposed method is successfully tested on different websites and validated to prevent XSS attack.

References

[1]. **The Open Web Application Security Project**, "OWASP TOP 10 Project", Retrieved from http://www.owasp.org/

[2]. **Fabien Duchene, Roland Groz, Sanjay Rawat, and Jean-Luc Richier, (2012).** "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing". *IEEE Fifth International Conference on Software Testing, Verification and Validation.*

[3]. **Yu Sun and Dake He, (2012).** "Model Checking for the Defense against Cross-site Scripting Attacks". *International Conference on Computer Science and Service System.*

[4]. **Lwin Khin Shar and Hee Beng Kuan Tan, (2012).** "Mining Input Sanitization Patterns for Predicting SQL Injection and Cross Site Scripting Vulnerabilities". *IEEE.*

[5]. **Lwin Khin Shar and Hee Beng Kuan Tan, (2012).** "Defending against Cross-Site Scripting Attacks". *IEEE.*

[6]. **R. Priyadarshini, D. Jagadiswaree, A. Fareedha, and M. Janarthanan, (2011).** "A Cross Platform Intrusion Detection System using Inter Server Communication Technique". *International Conference on Recent Trends in Information Technology, IEEE.*

[7]. **Andrea Avancini and Mariano Ceccato, (2011).** "Security Testing of Web Applications: A Search Based Approach for Cross-Site Scripting vulnerabilities". *IEEE.*

[8]. Rattipong Putthacharoen and Pratheep Bunyatnoparat, (2011). "Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique". *ICACT*.

ABOUT THE AUTHORS

Bharti Nagpal is currently working as an Assistant Professor in the Department of Computer Science Engineering at Ambedkar Institute of Advanced Communication Technology & Research, Delhi, India. She obtained her M.Tech in Information Systems from N.S.I.T Dwarka, Delhi in 2009 and B.Tech in Computer Science Engineering from NIT Kurukshetra, in the year 1999. She has about 14 years of teaching experience. Her research interest includes Web Technologies, Information Security, Web Mining, Data Mining and Data Warehousing.

Dr. Naresh Chauhan is currently the Chairman and Professor in the Department of Computer Science Engineering at YMCA University of Science and Technology, Faridabad, Haryana, India. He received his Ph.D. in Computer Science Engineering from MD University, Rohtak (Haryana) in 2008, M.Tech in Information Technology from GGS Indra Prastha University, Delhi in 2004 and B.Tech (Computer Science Engineering) from NIT Kurukshetra, in the year 1992. He has about 22 years of experience in teaching and Industries. He served Bharat Electronics Ltd. and Motorola India Ltd. His research interest includes Internet Technologies, Software Engineering, Software Testing and Real Time Systems. He has published one book on Software Testing published from Oxford University Press, India (2010).

Dr. Nanhay Singh is currently working as an Associate Professor in the Department of Computer Science Engineering at Ambedkar Institute of Advanced Communication Technology & Research, Delhi, India. He received his Ph.D in Computer Science Engineering from Kurukshetra University, Kurukshetra (Haryana) in 2011, M.Tech in Computer Science Engineering from Kurukshetra University, Kurukshetra in 1998. He has about 16 years of teaching experience. He served as Assistant Professor in various prestigious institutes like HBTI, Kanpur (Uttar Pradesh) etc. His research interest includes Web Mining, Web Security, Web Applications and Data Mining.