ISSN:0975-9646

A Brief Study on Defining Templates to Avoid XSS Vulnerabilities Using Auto Escape Templates for Web Applications

Ch Rajesh
Asst. Professor, IT
ANITS, Visakhapatnam

K S V K Srikanth
Asst. Professor, IT
ANITS, Visakhapatnam

I S L Sarwani
Asst. Professor, IT
ANITS,Visakhapatnam

G Sankara Rao
Asst. Professor, CSE
GVPCEW, Visakhapatnam

Abstract: **Cross-site scripting (XSS) is a type of computer security vulnerabilities typically found in Web applications. XSS enables attackers to inject client-side script into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same origin policy (refer Fig:1). Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantecas of 2007 [1]. Their effect may range from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.**

***General Terms:*Web Security, Vulnerability, Templates, Security Attack, Web Application Security, Attacks from Third Parry Web Sites.**

***Keywords:*XSS, Cross Site Scripting, Auto Escape Templates, C Templates, Template Parsers.**

I. INTRODUCTION:

Avoiding Cross-site scripting (XSS) problems occurred in web applications with the help of Auto Escape templates to filter malicious code injected into the web applications by defining contexts and modifiers with proper parsers [2]. This paper gives you how to define auto escape templates to avoid XSS in your web applications.

II. XSS ATTACK:

XSS is possible because of internet security weakness of client-side scripting languages. HTML, Java Script, VB Script, ActiveX and Flash are prime culprits for this exploit. Exploited XSS is commonly used to achieve the following malicious results [3].

- Identity theft
- Accessing sensitive or restricted information
- Gaining free access to otherwise paid for content
- Spying on user's web browsing habits
- Altering browser functionality
- Public defamation of an individual or corporation
- Web application defacement
- Denial of Service attacks

III. AUTO ESCAPE TEMPLATES:

In the Template System, Auto Escape is an optional mode of execution evolved to defence cross-site scripting (XSS) attacks in web applications. In this optional execution mode the template system takes the responsibility of defining proper escaping modifiers for each variable in the template. So, the template developer no need to apply the escaping modifiers manually to each variable, which is a recursive and error-prone method mostly for big applications.

A built-in HTML-aware parser helps in determining the escaping modifiers required to apply on each variable, for the sake of the Template System during template initialization. The HTML-aware parser scans the template to determine the context of each variable such as to emit the unwanted variable. This scanning process doesn't require any processing time as it is done during initialization-time.

Auto Escape currently supports well HTML and Java script templates along with proper parsing. It also provides support a wide range of escaping modifier for other contexts like (XML, JSON and etc.) but without defining a parser. Parsers may exist specific to these contexts and modifiers for those that are fine-grained in the future.

In case of scams related to Phishing, a harmful web page (fake / malicious web page) is loaded instead of the original web page which we wants to access, but in cross site scripting the real web page is loaded with malicious scripts, it is very tough to identify the attack.

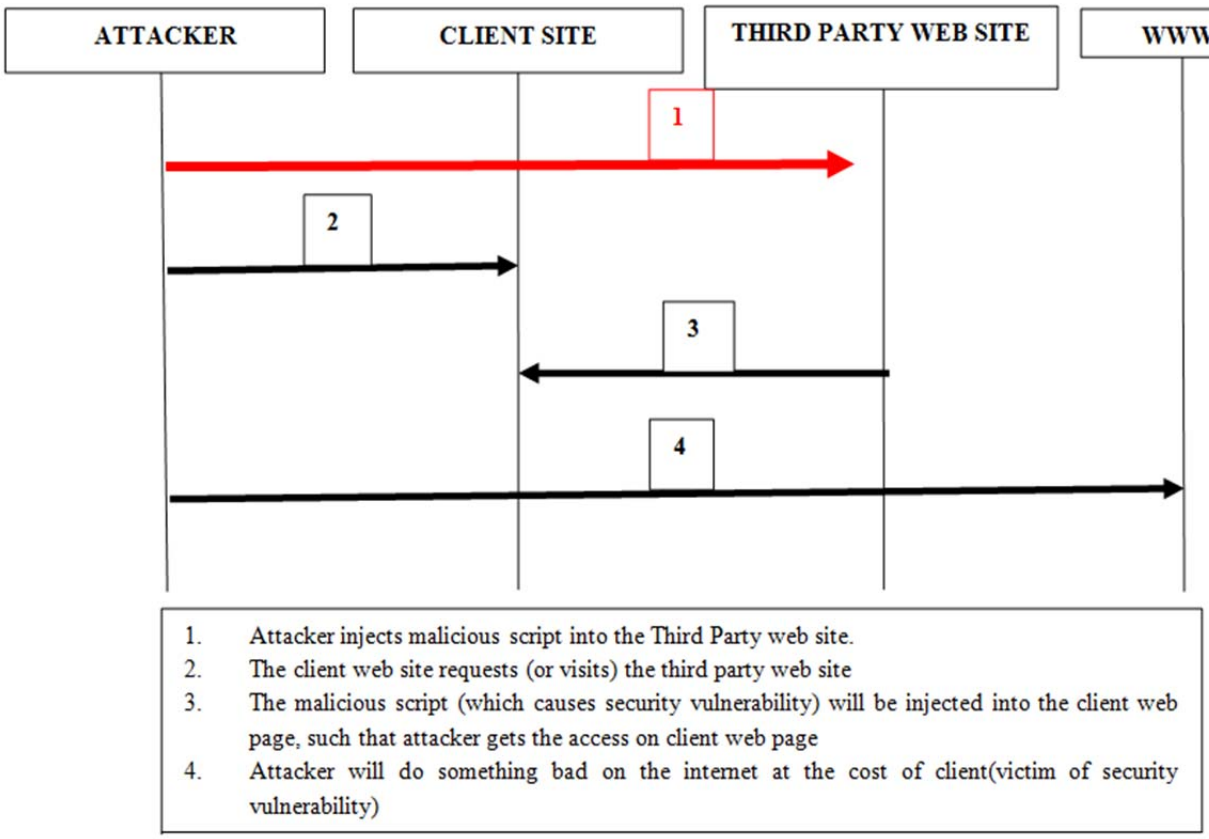

Fig 1 : A High Level View of Cross Site Scripting Attack

IV. SYNTAX RULES TO DEFINE TEMPLATES:

The HTML-aware parser takes care about filtering the unwanted variables or code, if we provide perfect template system.

There are two major parts in C-template System, they are:

- Template
- Dictionary

A Template defines an element on the web page, the element can be a string, image URL, Script, CSS or etc., when the HTML-aware parser finds a template it verifies the Dictionary for its value. If the template element is found in the dictionary then it is processed, otherwise the element will be treated as a malicious code and eliminated. The syntax to define a template is as follows:

Syntax: {{Template_Element_Name}}

A Dictionary is collection of key value pairs, as we already familiar, here the key is Template element (dictionary name), the template element is also called as marker, and the value is the description (dictionary value) of the template element or marker.

Expanding is a process which is done by the HTML-aware parser, in which each template element (marker) is been substituted with its dictionary value [4].

{“URL” : “./Mydirectory/mypic.jpg”,
“DATE” : “17-07-2015”,
“TITLE” : “My First Template”,
“BODY” : “This is an example of Auto escape template, \n which depicts how to use template and dictionary” }

For example: See, how the following web page is parsed by the HTML-aware parse.

```
<html>
        <head>
                <title>{{TITLE}}</title>
                {{META_TAGS}}
        </head>
        <body>
                {{BODY}}
                <img                    src=”{{URL}}”
alt=”logo”></img>
                {{DATE}}
        </body>
</html>
```

The output of the above web page would be as below, once after the instantiation of the template with the dictionary [6] (which is called “expanding”).

```
<html>
        <head>
                <title>My First Template</title>
        </head>
        <body>
                This is an example of Auto escape
template,
```

which depicts how to use template and dictionary

```
        <img src="./Mydirectory/mypic.jpg" alt="logo"></img>
        17-07-2015
    </body>
</html>
```

Here the markers {{TITLE}},{{BODY}},{{URL}} and {{DATE}} have been replaced by their respective dictionary values. Whereas, the marker {{META_DATA}} doesn't have an entry in the dictionary, so it is replaced by an empty string. Even like the marker {{META_DATA}} if any malicious code is injected it is also filtered in the same fashion.

V. TYPES OF TEMPLATES:

There are six major types of markers, that are been used mostly in template language [7]. They are as follows in Table 1:

VI. CONCLUSION:

Whenever we wish to get rid of these XSS attach then we need to deploy a strong template system on the client machine. The template system filters all the malicious code which is been injected from the third party website. But, proper care is to be taken before using the template system, because sometimes due to wrong template definition actual content of the web page may be filtered out by our template system. So, by following the above syntax rules template definitions must be made with utmost care.

Table 1: Types of Templates

Marker	**Description**
Variable Marker	The markers are replaced by the text defined by the value of respective dictionary. In the example earlier, all are variable markers only. Ex**:{{VAR_MRKR}}**
Section Marker	There are Start Section and end section markers, which specifies a section in the template. There may be 0, 1 or N section markers in the output. Section markers look like **{{#STRT_SEC}} ... {{/END_SEC}}**
Template include marker	It delegate other template to be expanded and inserted instead of this marker, wherever it appear. Template include can be thought of as a section marker, whose content is specified in a file rather than inline. Just like sections it can appear -, 1 or N times in the output. It look like **{{>MARK_INCL}}**
Comment Marker	Which may explain the template but it is removed completely from the o\utput once after parsing. Its syntax is - **{{! Your comment comes here ?}}**
Set-delimiter Marker	The default delimiters for markers are double open curl braces ({{) and double close curl braces (}}). If we wish to define our own custom delimiters for markers then we can use set-delimiter marker. Ex: ||**MRK-TAG**||, assume here '|' (pipe symbol is a delimiter)
Pragma Marker	To invoke additional template features we use pragma. The only pragma defined in the template system is **AUTOESCAPE** pragma.

REFERENCES:

[1] A Survey on Cross Site Scripting, Mr. Shailendra M. Pardeshi, International Journal of Advance Foundation and Research in Science & Engineering (IJAFRSE) Volume 1, Issue 5, October 2014. Impact Factor: 1.036, Science Central Value: 10.33

[2] Meixing Le and Angelos Stavrou ," Double Guard: Detecting Intrusions in Multitier Web Applications" IEEE Transactions On Dependable And Secure Computing, Vol. 9, No. 4, July/August 2012, pp. 512-525

[3] M. James Stephen, P.V.G.D. Prasad Reddy, Ch. Demudu Naidu and Ch. Rajesh," Prevention of Cross Site Scripting with E-Guard Algorithm" International Journal of Computer Applications (0975 – 8887) Volume 22– No.5, May 2011,pp. 30-34.

[4] https://www.netsparker.com/web-vulnerability-scanner/vulnerabilitysecurity-checksindex/ crosssite-scripting-xss/

[5] Cross site scripting techniques and mitigation by CESG revision 1.0, October 2007.

[6] S. Christey and R.A Martin. Vulnerability type distributions in cve, version 1.1. [online], http://cwe.mitreorg/documents/vuln-trends/index.html ,(09/11/07), may 2007.

[7] A. Klien. Cross site scripting explained. White paper, sanctum security group, http://crypto.stanford.edu/cs155/css.pdf, June 2002.