International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Developing Firefox add-on for DOM vulnerability Assessment

Saranraj Ilangovan[1], Geogen George[2]

Master's in Information Security and Cyber Forensics Cyber security Research Center, SRM University SRM Universirty.Chennai, India

Abstract: ***Cross-site scripting (XSS) is a type of security vulnerability typically found in web applications which allows the attackers to inject malicious script into web pages/servers. XSS is the main cause of DOM attack.This add-on is a penetration testing tool to detect DOM vulnerabilities in Web Applications. This tool detects the DOM vulnerabilities based on xss vulnerabilities in the web page. It provides a penetration tester the ability to test all kinds of xssvulnerabilities. This add-on will be useful for web application developers in detecting DOM vulnerabilities early in the development process will help protect a web application from unnecessary flaws.***

Keywords: Vulnerability Testing; Cross-site scripting; Web Applications; Security; Code Insertion;

1. Introduction

Most Web sites today add dynamic content to a Web page making the experience for the user more enjoyable. Dynamic content is content generated by some server process, which when delivered can behave and display differently to the user depending upon their settings and needs. Dynamic Web sites have a threat that static Web sites don't, called "cross-site scripting," also known as "XSS."

"A Web page contains both text and HTML markup that is generated by the server and interpreted by the client browser. Web sites that generate only static pages are able to have full control over how the browser user interprets these pages. Web sites that generate dynamic pages do not have complete control over how their outputs are interpreted by the client. The heart of the issue is that if untrusted content can be introduced into a dynamic page, neither the Web sites nor the client has enough information to recognize that this has happened and take protective actions," according to CERT Coordination Center, a federally funded research and development center to study Internet security vulnerabilities and provide incident response.
Cross-site scripting is gaining popularity among attackers as an easy exposure to find in Web sites. Every month cross-site scripting attacks are found in commercial sites and advisories are published explaining the threat. Left unattended, your Web site's ability to operate securely, as well as your company's reputation, may become victim of the attacks.

2. Literature Review

The web technologies were invented for the betterment of world, but with its advancement threats like spam, malware, hacking, phishing, denial of service attacks, invasion of privacy, defamation, frauds, etc. started taking place. Web server's vulnerabilities were the target for the attackers in the early days of dot com boom. Microsoft's IIS was vulnerable to those attacks. Attacks on web server were mainly performed on core server code and on supporting library. Those attacks were basically buffer overflow, input validation attacks, format string attack, canonicalization attack, encoding attacks, privilege escalation, form tampering and user generated content. But the improvement in security aspects of network and servers reduced the successful attacks on well-configured web servers. Like other technologies web applications also faced attacks. The development of server side languages exposed the web server for security vulnerabilities. With the popularity of blogging and web services, forums attackers started taking interest in web applications. This resulted in some new attacks like cross site scripting (XSS), SQL injection, and insecure direct object reference, remote malicious file inclusion, cross site request forgery, access control weaknesses, data confidentiality failures and poor error handling.

Among those attacks XSS is the most common security vulnerability in today's web applications. It is more dangerous as it provides surface for other type of attacks. XSS causes danger for victim by the insertion of a piece of script on client side. This code can be written in any scripting language. JavaScript is mostly used but use of other languages is also possible. This attack can also be deployed through a link in an email or on a web page that appears to be originated from the hacker's site. This work provides an overview of classification of XSS, threats, detection methods.

Cross-Site Scripting

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. [4]

A High Level View of a typical XSS Attack

Classification of Cross Site Scripting [1]

XSS attacks are generally categorized into following categories:

- Stored (or persistent)
- Reflected (or nonpersistent).
- DOM based XSS and
- Induced XSS attacks.
- mXSS or Mutation XSS

A. Stored or Persistent XSS

In stored or persistent XSS attacks the injected malicious code is permanently stored on the target servers. In this type of attack, attacker first tries to find vulnerability in web application. If such vulnerability is present in web application, he injects a malicious script that will be able to steal user's confidential information or cause other damages. This script then resides permanently on the server. When any user access this information via web application, the malicious script gets executed and the confidential information becomes accessible to attacker. Stored XSS attacks are generally performed on web applications that takes input from user in the form of text and store it in the database of web application. Some examples of these applications are blogs, forums, comments or profile.

B. Reflected or Non-persistent XSS

As opposed to stored XSS attacks in reflected XSS attacks the injected code doesn't reside on the web server. In reflected attacks malicious links are sent to victims using email, or embedding the link in a web page residing on another server. When user clicks on this link, the injected code goes to attacker's web server, which sends the attack back to victim's browser. Now browser executes the code because it comes from a trusted server. In this way an attacker bypass the same origin policy. When this code executes on browser, it performs the malicious work like stealing the confidential information of victims.

C. DOM based XSS

DOM Based XSS is an XSS attack where the DOM environment in the victim's browser is modified by the original client side script, so that the client side code runs in an "unexpected" manner. In this kind of attack the page doesn't change but the client side code gets executed in a different manner because of the modification in the DOM environment. It is different from the other two XSS attacks as the attack is executed at the client side.

D. Induced XSS

Induced XSS are possible in the web applications where web server has HTTP Response Splitting vulnerability. As a result of this vulnerability an attacker can manipulate the HTTP header of the server's response. These types of XSS are not very common but it is mentioned here for the completeness of classification. The DOM based and induced XSS attacks are more severe as they can also affect static HTML pages.

E. mXSS or Mutation XSS [2]

mXSS or Mutation XSS is a kind of XSS vulnerability that occurs when the untrusted data is processed in the context of DOM's innerHTML property and get mutated by the browser, resulting as a valid XSS vector. In mXSSan user specified data that appears harmless may pass through the client side or server side XSS Filters if present or not and get mutated by the browser's execution engine and reflect back as a valid XSS vector. XSS Filters alone won't protect from mXSS. To prevent mXSS an effective CSP should be implemented, Framing should not be allowed, HTML documents should specify the document type definition that enforce the browser to follow a standard in rendering content as well as for the execution of scripts.

Threats Due to XSS [2]

XSS Tunneling:

XSS Tunnel is the standard HTTP proxy which sits on an attacker's system. Any tool that is configured to use it will tunnel its traffic through the active XSS Channel on the XSS Shell server.

Client side code injection:

Client-side attacks, and particularly code injection at the client, might be the first thing a layperson thinks of when they hear about mobile security threats. Client-side injection in mobile applications works in a way similar to certain server-side security risks.

DOS:

A denial-of-service (DoS) or distributed denial-of-service (DDoS) attack is an attempt to make a machine or network resource unavailable to its intended users.

Cookie Stealing:

A cookie stealer is used to steal the login information of any unsuspecting victim. Once the link is visited, the cookie of the user is taken and stored in a text file. They are then redirected to another page without knowing what has just happened

Malware Spreading:

Malware includes computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other malicious programs. As of 2011 the majority of active malware threats were worms or trojans rather than viruses.

Phishing:

The fraudulent practice of sending emails purporting to be from reputable companies in order to induce individuals to reveal personal information, such as passwords and credit card numbers, online.

Defacing:
Website defacement is an attack on a website that changes the visual appearance of the site or a webpage.

Detecting Cross-Site Scripting [3]

- Insecure JavaScript Practice
- Malicious code between Static Scripts
- Browser-specific Problems
- DOM-based Problems
- Multi-Module Problems

Insecure Javascript Practice

Yue et al. characterize the insecure engineering practice of JavaScript inclusion and dynamic generation at different websites by examining severity and nature of security vulnerabilities. These two insecure practices are the main reasons for injecting malicious code into websites and creating XSS vectors. According to their survey results, 66.4% of measured websites has insecure practice of JavaScript inclusion using src attribute of a script tag to include a JavaScript file from external domain into top-level domain document of a web page. Top-level document is document loaded from URL displayed in a web browser's address bar. Two domain names are regarded as different only if, after discarding their top-level domain names (e.g., .com) and the leading name "www" (if existing); they don't have any common sub-domain name. For instance, two domain name are regarded as different only if the intersection of the two sets
{ d1sub2.d1sub1}and { d2sub3.d2sub2.d2sub1}is empty .
1. www.d1sub2.d1sub1.d1tld
2. d2sub3.d2sub2.d2sub1.d2tld

Almost, 79.9% of measured websites uses one or more types of JavaScript dynamic generation techniques. In case of dynamic generation techniques, document.write(), innerHTML, eval() functions are more popular than some other secure methods. Their results show 94.9% of the measured website register various kinds of event handlers in their webpage. Dynamically generated Script (DJS) instance is identified in different ways for different generation techniques. For the eval() function, the whole evaluated string content is regarded as a DJS instance. Within the written content of the document. Write () method and the value of the innerHTML property, a DJS instance can be identified by from three source .

- Between a pair of <SCRIPT> and </SCRIPT> tags
- In an event handler specified as the value of an HTML attribute such as onclick or onmouseover;
- In a URL using the special JavaScript: protocol specifier.

To eliminate this risk, developers have to avoid insecure practice of JavaScript, such as they need to avoid external JavaScript inclusion using internal JavaScript files, eval() function need to be replaced with some other safe function

Malicious Code between Static Scripts

User input between any existing scripting codes is vital issue while detecting XSS. It's really hard to find any method from existing systems that can solve this dilemma appropriately. There are two types of scripting code in any webpage. Some of them are static and some of them are dynamic (composed during runtime). Let's begin the discus on this issue with one example.

<SCRIPT>var a = $ENV_STRING; </SCRIPT>

User given data between static script code In the above example, both starting both starting and ending tag of script are static and the user input is sandwiched between them that make the scripting code executable. But problem is that any successful injection in this context may create XSS vector. All strong filters of the existing systems try to find malicious code from the user input. This kind of situation in static code may help attackers to circumvent any detecting filter. For instance, the Samy MySpace Worm introduced keywords prohibited by the filters (innerHTML) through JavaScript code that resulted the output as the client end (eval('inner'+'HTML')). On the other hand we cannot eliminate any static scripting code while filtering because they are legitimate and there may be a safe user input between those legitimate codes. So it is hard to isolate and filter input that builds such construct without understanding the syntactical context in which they used . So meaning of the syntax is a vital concern while filtering.

Dom-Based Problems

One of the crucial problems of most existing systems is they cannot detect DOM-based XSS. So only identifying stored and reflected XSS is not sufficient for preventing all of XSS domain and according to Amit Klein's article, DOM based is one of the upcoming injection problems in web world because nowadays, most of the issues related to other type of XSS problems are being cleaned up on major websites. So, bad guys will try for third type of XSS vulnerability. We already know, DOM-based XSS vector does not need to appear on the server and it's not easy for a server to identify. So, attackers get extra advantage with this type of XSS vulnerability. DOM-based XSS is introduced by Amit Klein in his article and this type XSS can be hidden in the JavaScript code and many strong web application firewalls fail to filter this malicious code. In the eXtensible Markup Language (XML) world, there are mainly two types of parser, DOM and SAX. DOM-based parsers load the entire document as an object structure, which contains methods and variables to easily move around the document and modify nodes, values, and attributes on the fly. Browsers work with DOM. When a page is loaded, the browser parses the resulting page into an object structure. The getElementByTagName is a standard DOM function that is used to locate XML/HTML nodes based on their tag name. Let's start to discuss about on this topic deeply with Amit Klein given example.

Say, the content of http://www.vulnerable.site/welcome.html as follows:

```
<HTML>
<TITLE> Welcome! </TITLE>
<SCRIPT>
varpos =document.URL.indexof("name=")+5
document.write(document.URL.substring(pos,
document.URL.length));
</SCRIPT>
<BR>
Welcome to our System
</HTML>
```

If we analyze the code of the example, we will see that developer has forgotten to sanitize the value of the "name" get parameter, which is subsequently written inside the document as soon as it is retrieved. The result of this HTML page will be "***Http://vulnerable.site/welcome.html?name=Joe"***(if user input is 'Joe'). However, if the user input is any scripting code that would result in an XSS situation. e.g.;

"http://vulnerable.site/welcome.html?name=<SCRIPT>alert(document.cookie);</SCRIPT>"

Many people may disagree with this statement and may argue that still, the malicious code is sending to the server, and any filter can be used in the server to identify it. Let's see an update version of previous example.

http://vulnerable.site/welcome.html#name=<SCRIPT>alert(document.cookie)</SCRIPT>

Here sign (#) right after the file name used as fragment starter and anything beyond this is not a part of query. Most of the well-known browsers do not send the fragment to server. So actual malicious part of the code is not appeared to the server, and therefore, the server would see the equivalent of

http://www.vulnerable.site/welcome.html

More scenarios on DOM-based XSS are in Amit Klein's article. He suggests that minimizing insecure JavaScript practice in code may reduce the chances of DOM-based XSS. Web developer must be very careful when relying on local variables for data and control and should give attention on the scenarios wherein DOM is modified with the user input. Automated testing has only very limited success at identifying and validating DOM based XSS as it usually identifies XSS by sending a specific payload and attempts to observe it in the server response. If we exclude the idea of (#) sign but may not work in the following contrived case:

```
<SCRIPT>
varnavAgt = navigator.userAgent;
if (navAgt.indexOf("MSIE")!=-1)
{
document.write("You are using IE and visiting site"
+document.location.href+".");
}
else
{
document.write("You are using an unknown browser.");
}
</SCRIPT>
```

For this reason, automated testing will not detect areas that may be susceptible to DOM based XSS unless the testing tool can perform addition analysis of the client side code. Manual testing should therefore be undertaken and can be done by examining areas in the code where parameter are referred to that may be useful to attackers. Examples of such Areas include places where code is dynamically written to the page and elsewhere where the DOM is modified or even where scripts are directly executed.

Multi-Module Problems

The vulnerability of a server page is necessary condition for the vulnerability of web application, but it isn't a necessary condition. That means protecting any single page from a malicious code never guarantees the protection of entire web application. Server page may send user data to ot her page or to any other persistent data store instead of client browser. In these situations, XSS may occur through another page. Most of the existing systems don't provide any procedure to handle this difficulty. In the multi-module scenario, data may be passed from one module to another module using some session variables and those session variables status are stored in cookies. Let's see the above example. In the above example, we can see user input is stored into session variable and later it is stored into $name variable. The session variable is echoed through different page. So, any filtering process on $name variable will not effect to session variable. In this case, any malicious code can create XSS vector using session variable and can bypass any filtering process. Bisht, Venkatakrishnan and Balzarotti, Cova, Felmetsger, Vigna solved Multi-module problem in their work but most of other tools are not having any technique to handle it

```
<HTML>
<HEAD>
<TITLE> Enter User Name:</TITLE>
</HEAD>
<BODY>
<? php
//connect to the existing session
session_start();
// create a session variable
session_register("ses_var");
// set ses_var with php variable
$HTTP_SESSION_VARS["ses_var"] = $name
if (isset($_POST["user"])){
$name = addslashes($_POST["user"]);
exit;
}
?>
<FORM action = "create.php" method ="POST">
UserName :
<input name = "user" type = "text">
<input name = "OK" type = "submit">
</FORM>
</BODY>
</HTML>
<?php
        echo $_SESSION ["ses_var"];
?>
```

3. Design

I am proposing an add-on that will detect reflected, stored, DOM based cross-site scripting in a webpage. This add-on will help pentesters and web developers to detect XSS vulnerability in a webpage. We want to check whether a website is vulnerable to cross-site scripting attack by add-on created. The add-on works by submitting the HTML forms and substituting the form value with strings that are representative of an XSS strings. If the resulting HTML page sets a specific JavaScript value (document.vulnerable=true) then the tool makes the page as vulnerable to given XSS string.

A report is generated based on the vulnerabilities in the element.

Modules

Four modules are designed, they are
- Extracting Forms.
- Requesting the server with XSS Strings.
- Analyzing the response.
- Report Generation.

Extracting Forms
All the forms in the web pages are choosed.

Requesting the server with XSS Strings
The HTML forms are submitted with XSS payloads, then wait for the response from the server.

Analyzing the Response
If the resulting HTML page sets a specific JavaScript value (document.vulnerable=true) then the tool makes the page as vulnerable to given XSS string.

Report Generation
A report is generated on the basis of vulnerabilities in the webpage.

References

[1] Vikas K. Malviya, SaketSaurav, Atul Gupta, ”On Security Issues in Web Applications throughCross Site Scripting (XSS)” 2013 20th Asia-Pacific Software Engineering Conference.
[2] Ajin Abraham, (Author of OWASP Xenotix XSS Exploit Framework) ”Ultimate xss Protection Cheat Sheet For Developers”
[3] SumanSaha , “Consideration Points: Detecting Cross-Site Scripting”, (IJCSIS) International Journal of Computer Science and Information Security,(2009)
[4] OWASP