Vol.5. No. 1, July, 2016
111 - 118, ISSN : 0975-5446

Preventing DOM-based Cross-Site Scripting Attacks by Traversing Rendering Tree

DESAI Purva N.
JJT University, Jhunjhunu, Rajasthan, India
purva.desai79@gmail.com

PANDYA Jagdish G.
JJT University, Jhunjhunu, Rajasthan,
india pandyajg@yahoo.com

Abstract

Now a day, Internet has become handy and most advanced useful technology due to use of various electronic gadgets. This online system helps the present human civilization to such a greater extend that life without internet seems to be impossible. Due to its omnipresence, Internet has started attracting hackers/attackers who keep looking for new techniques to create maliciousness in web application. According to researchers and industry experts (like CENZIC, CISCO, OWASP, WhiteHat Security), the Cross-Site Scripting (XSS) is the one of the top most vulnerability in the web application. Here, injected malicious code executes on the browser's site which affects victims badly. This paper focuses on DOM-based XSS attacks i.e. 3rd type of XSS attacks based on how attacks take place during browser's display process. This paper further discusses on the proposed algorithm called as the DOM tracker that protects web application against DOM-based XSS attacks.

Keywords: XSS, Stored XSS, Reflected XSS, DOM-Based XSS, render tree

1. Introduction

In this current age of information and technology, most of the tech-savvy population of the world became heavily dependent on the Internet. Various services provided by Internet has changed present civilization to climb its new and wider dimensions. It is bitter truth that without Internet our day-to-day life seems to be impossible.

Today thousands of web sites including customized services have become key part of most of the human activities. But on the other side unfortunately, it has attracted hackers or attackers too who always keep eyes and tries to create new techniques to sabotage fruitful-skilled advance technology by intruding in to web application. XSS is one of the top most vulnerability in the web application.

XSS is one kind of application layer web attack in which they try to inject malicious scripts to perform malicious actions on any trusted web sites to fulfill only their nominal or bigger self-interest. According to Shalini & Usha (2011) [1], it is called "cross-site" because it involves interaction between two web sites to achieve attacker's goal. In XSS, malicious code executes on web browser side which badly affects users.

In normal case, XSS executes when the web page is loaded or associated event occurs. XSS is not only embedded in JavaScript and HTML, but also in VBScript, ActiveX, AJAX,

action script like flash or any other browser executable scripting language and mark-up language.

For many reasons XSS can be used such as Take over user' account, Spread worms, Trojan horse, Control access of browser, Phishing, Expose of the user's session cookie, Redirect the user to some other page or site, Modify presentation of content, Bypass restrictions, Malware attacks & DoS attack, Fake advertisement, Click fraud, etc.

2. Types of XSS Attacks

Before 2005 there were only two types of XSS attacks that were identified i.e. stored XSS & reflected XSS. Later Amit Klein defined third type of XSS called Dom-Based XSS. These three types of XSS are defined as follow:

2.1 Stored XSS (persistent XSS)

When XSS vectors in user input is stored on the target server such as in database using message form, visitor log, bulletin board, comment field, etc. stored XSS attack generally occurs. In that case whenever users opens that web page in the browser, script executes automatically.

2.2 Reflected XSS (non-persistent XSS)

In reflected attacks, injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as a part of the request. This type of attack normally is done through another route such as in an e-mail message or on some other website.

Here, when a user is tricked in to clicking on a malicious link, submitting a specially crafted form or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user browser. The browser then executes the code because it came from a trusted site.

2.3 DOM-Based XSS

This attack is only possible when improper handling of data related with DOM (document object model) present in HTML page. In fact, every HTML document has related DOM and it represents document properties related to the browser. These HTML entities present in page can be accessed and modified by using DOM object properties such as document.referrer, document.url and document.location.

Attacker can manipulate or access DOM properties to execute such type of attack. In DOM-Based XSS the malicious script does not reach to the web server. It is executed in web browser only. This type of attack is only possible when untrusted data which is provided by the user is interpreted as JavaScript using method such as eval(), document.write() or innerHTML.

By handling certain DOM objects, the attacker can manipulate and generate cross site scripting condition. Out of many objects most vulnerable and popular objects of documents can be listed as: (i) URL of document, (ii) Location and (iii) Reference objects of the document.

3. How browser displays web pages?

Basically browser is application software which is used by web user who wants to surf web sites. Browser displays contents of a web page which is requested by user. It contains seven main components which is shown in figure 1 and its functionality is described in table 1.

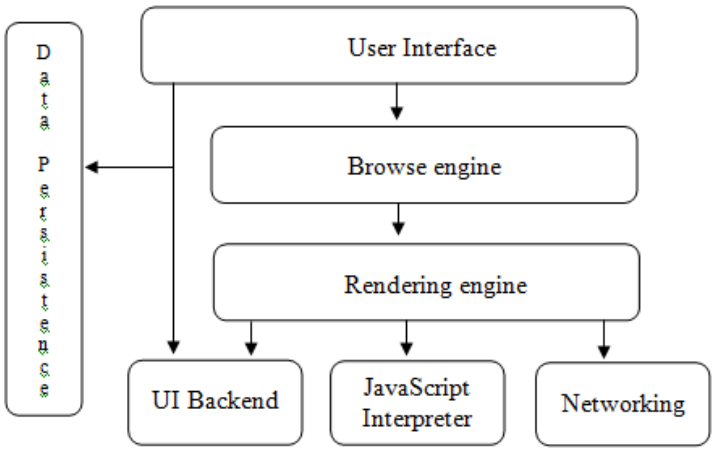


Figure 1: Modules of web browser

Table 1: Modules Of Web Browser

Modules	**Role**
User Interface	User interface includes the address bar, back button, forward button, bookmark option menu, refresh button etc. Content part of the browser i.e. the main window enables you to see the requested page.
Browser Engine	It provides interaction between user interface and rendering engine.
Rendering Engine	Browser displays requested contents because of this main component i.e. using rendering engine. For example if user' request HTML contents then rendering engine is responsible for parsing the HTML and CSS, and displaying the parsed content on the screen.
Networking	It works with network calls like HTTP requests. It has platform independent interface for different implementations.
UI Backend	It is use to draw basic web page tool design like combo boxes and windows on the browser's screen. It describes generic interface which is platform independent. It uses methods which is defined by operating system to define user interface.
JavaScript Interpreter	It is used to parse and execute the JavaScript code snippet.
Data Persistence	It presents layer which is persistence. The browser needs to store data like cookie on the hard disk.

Rendering engine is main component of browser. It fetches contents of user's requested document from network layer and creates content tree by parsing HTML document and by converting HTML contents to DOM nodes in the tree. It also creates separate tree called render tree or frame tree for visual commands of HTML file. This content tree and render tree is connected to gather and is refer as attachment. Now each nodes it extract from coordinates at which position output of tag is going to appear on browser screen. This process is known as layout or reflow. In next step, render engine traverse each and every node of tree and draws it on browser's screen using UI backend layer. Drawing nodes on screen is known as painting. These workout sequences described in figure 2:

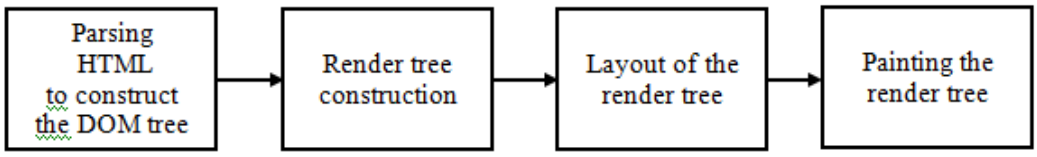
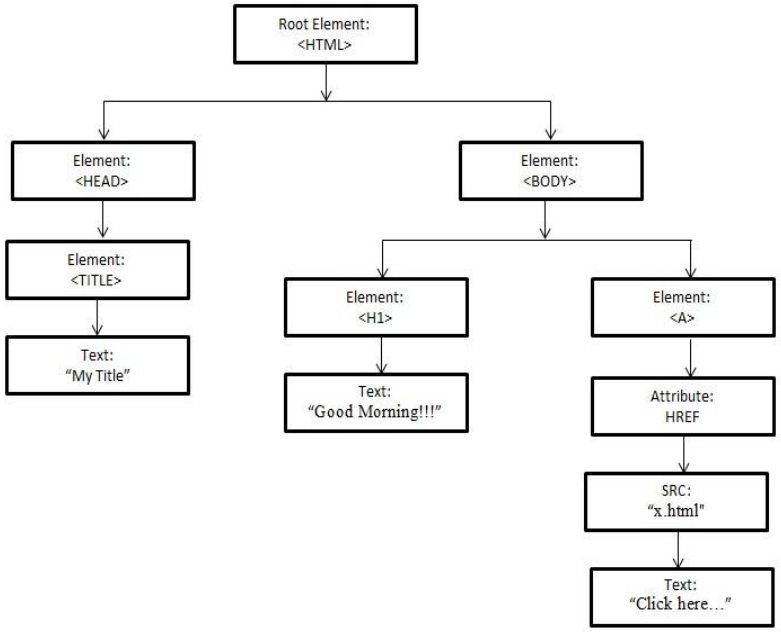


Figure 2: General flow of rendering process

For example, if your HTML page code is as follow:

```
<html>
  <head>
        <title> My Title </title>
  </head>
  <body>
        <h1> Good Morning!!! </h1>
        <a href="x.html"> Click here… </a>
  </body>
</html>
```

hen the DOM tree of this code would be defined as figure 3



:

Figure 3: DOM tree

4. Reviews of Literatures

Patil and Patil (2015) [2] explained working scenario of web application. They also stated different types of cross-site scripting attacks and existing sanitizer solutions. Their system architecture contains DOM module which creates DOM tree for each web page, input field capture module fetches user inputs, input analyzer module categorized given user input, link module maintains list of links of current web page, text are module maintains inputted text data of current web page, these list of links and texts are passes to sanitizer module which detects XSS attacks and XSS notification module gives message to web user regarding XSS vulnerabilities. This approach implemented as browser extension using Jetpack framework.

Tiwari and Jeysree (2015) [3] discussed about new class of XSS called mutation-based XSS that occur in innerHTML related properties. This paper stated various examples of mutated XSS attack vectors. It basically presents as inactive markup which turns into active attack after transformation occur and bypass client site XSS filters. Authors also denoted that generally mutation occurs in web application to perform vulnerabilities due to serialization and deserialization of data in DOM-tree. Authors described automation technique to search vulnerable code in innerHTML properties. If mutants are available, it reported to tester module.

Acker *et al.* (2014) [4] **s**tudied about Grease monkey browser extension and community-driven script market. Here created scripts are run on HTTP as well as HTTPs. This paper further describes about DOM-based XSS occurred through Grease monkey API. Here script can easily added to any web page using @include directive which results in global XSS or privileged XSS. Authors also developed proof-of-concept exploit to verify these attacks.

Kaur (2014) [5] proposed two-tire implementation to remove limitations of XSS attacks. One tire is for detecting reflected and stored XSS attack and second tire for avoiding DOM-based XSS attack. In first tire, script guard logic is implemented in MVC2 architecture controller that receives all parameters of client request and scans it to detect XSS. It matches parameters with predefine expressions. If attack is found, page request is not further proceed. For second tire, author creates JavaScript code that send to client with each response. This code runs at client site to detect and prevent DOM attack.

Krishnaveni and Sathiyakumari (2013) [6] presents method that works on features extraction. Authors identify features like script based features, core contents, DOM objects, some special tags and methods. These features are extracted from URL and web pages. They traced variables and functions from the web page and examines attack factors.

Saha *et al.* (2011) [7] presents static taint analysis that detects loop holes of DOM-based XSS. Here authors mentioned entry point of DOM-based XSS through URL parameter like document.location, document.URL or document.referrer. Proposed system uses detector that parses JavaScript embedded HTML input file and generates abstract syntax tree. Detector than analyze data source and path flow. This make sure that URL must sanitize.

Barnett (2011) [8] analyze various defensive strategies like blacklist filtering, generic attack payload detection, whitelist filtering, identifying improper output handling flows and application profiling of ModSecurity. Author introduced client-site approach to avoid DOM-based XSS called Active Content Signature (ACS). Here JavaScript code is created as temporary DOM that executes on browser and removes DOM type malicious code. Here ACS is needed to be added at Document_Root of web pages without affecting rest of source code of web pages. ACS does not

allow execu ting inline scripts so source code of web page needed to be change. This system does not work for stored cross-site scripting attacks and reflected cross-site scripting attacks.

Sharath and Selvakumar (2011) [9] present approach where XSS sanitizer parses HTML contents provided by user to find out static tags. This approach is known as BIXAN. Here, static tags are remain as it is and non-static tags are filtered out. If static tags contains dynamic content than tags are sent to JavaScript tester. It checks out whether these contents are harmful or not. If it is than it will again filter out. Rest of the safe code is converted into DOM API and is executed safely at client site. This way it will filter out malicious XSS content i.e. added as dynamic property value of script tag.

5. Research Objectives

The main objectives of this research paper are as follow:

- Identifying DOM-based XSS attack vectors.
- Defend web application from DOM-based XSS attacks.

6. Methodology

To protect against DOM-based XSS attacks, proposed algorithm DOM tracker can be implemented as client-site solution (as plug-ins or browser modification) wherein DOM properties can be modified either directly or via accessing HTML. Before requested web page is appear or visible in client browser, browser's rendering engine creates render tree which is combination of DOM tree and CSSOM tree. To find DOM attack, this render tree needs to be traverse. Flow of DOM tracker module is shown in figure 4:

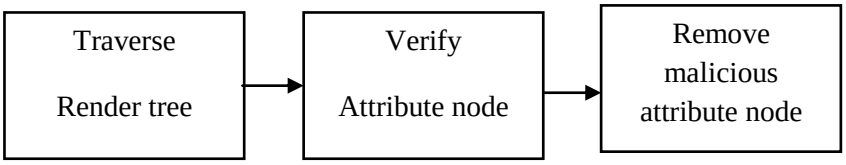


Figure 4: Flow of DOM tracker module

6.1 Algorithm for DOM tracker module:

Following section defines working flow of DOM tracker algorithm. To fix up DOM attacks, render tree needs to be traverse and verify properly before it started drawing web page layout on browser's screen. Here, SiteURLTable is data table, which contains list of URL supported by given web site; HTML tag is considered as an element node; attribute of HTML tag is considered as an attribute node; if web page contains comment, it is considered as comment node.

Step 1: Fetch the render tree and process its root node.

Step 2: Check whether the root node of the tree contains a left side element node or not.

Step 3: Check whether this element node is comment node or not. If the given node is a comment node then remove the comment node (because it contains attack vector) and fetch next element node. If given node is not comment node then fetch its attribute node.

Step 4: Check the contents of attribute node. If attribute node contains 'href' or 'src' attribute then fetch its attribute value and compares it with URL entries mentioned in SiteURLTable. If its attribute value does not match with SiteURLTable data entries, remove this attribute node from tree. Otherwise, fetch another attribute node, if any and repeat this process.

Step 5: Fetch the next element node, if any. Repeat step 3 and step 4 until the left side element' leaf node or last child node is arrived.

Step 6: If the root node of the tree contains right side element node, follow same verification process for attribute node until right side element's leaf node or last child node is arrived.

Step 7: If the tree node verification process is completed, process of drawing the nodes on browser's screen take place.

Here it is important to note that, in overall scenario of DOM-based attack, attacks are generally occurred by:

- changing URL redirection,
- modifying current HTML element nodes,
- add new HTML element node

This can be done by using either document object, location object, windows object or using element directly. In all above cases, attacker/hacker mostly tries to redirect victim to some malicious web page. All the modifications of attacker/hacker are interpreted and resulting elements are added to render tree which in turn is painted on browser's screen. So in above algorithm, only those attribute nodes are veriferd which contains 'href' or 'src' attribute.

This module only works with comment node and attribute node, it does not work with text node (text node contains static html text string) i.e. if attacker used text node then this module fails to detect injected vector.

7. Conclusion

Cross-Site Scripting is the simplest way for an attacker to gain user's sensitive information. This paper presents technique that identifies as well as prevents DOM-based XSS attacks by traversing render tree at browser site. If render node contains any comment node or attribute node which contains malicious links then this module removes detected node and performs further processing. This paper presents conceptual approach towards DOM-based XSS attacks. Further, we are planning to develop techniques which detect and prevent Cross-site request forgery attacks which is sub category of XSS attack.

References

[1] Shalini, S., & Usha, S. (July, 2011). Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side. *IJCSI International Journal of Computer Science, Vol. 8*(4), 650-654.

[2] Patil, D. K., & Patil, K. R. (July, 2015). Client-side Automated Sanitizer for Cross-Site Scripting Vulnerabilities. *International Journal of Computer Applications, 121*(20), 1-8.

[3] Tiwari, A., & Jeysree, J. (April, 2015). Automation of Mutated Cross Site Scripting. *International Journal of Science and Research (IJSR), 4*(4), 1274-1277.

[4] Acker, S. V., Nikiforakis, N., Desmet, L., Piessens, F., & Joosen, W. (June, 2014). Monkey-in-the-browser: Malware and Vulnerabilities in Augmented Browsing Script Markets. *ASIA CCS'14.* Kyoto, Japan: ACM.

[5] Kaur, G. (2014). Study of Cross-Site Scripting Attacks and Their Countermeasures. *International Journal of Computer Applications Technology and Research, Vol. 3*(10), 604 - 609.

[6] Krishnaveni , S., & Sathiyakumari , K. (July, 2013). Multiclass Classification of XSS Web Page Attack using Machine Learning Techniques. *International Journal of Computer Applications, 74*(12), 36-40.

[7] Saha, S., Jin, S., & Doh, K.-G. (2011). Detection of DOM-based Cross-Site Scripting by Analyzing Dynamically Extracted Scripts. *Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / National Research Foundation of Korea (NRF)*, (pp. 487-491). Korea.

[8] Barnett, R. (January, 2011). XSS Street-Fight: The Only Rule Is There Are No Rules. *Trustwave*.

[9] Sharath, C. V., & Selvakumar, S. (September, 2011). BIXSAN: Browser Independent XSS Sanitizer for prevention of XSS attacks. *ACM SIGSOFT Software Engineering Notes, Volume 36*(5).