2014 24th International Conference on Computer Theory and Applications (ICCTA) | 978-1-4799-7476-4/14/$31.00 ©2014 IEEE | DOI: 10.1109/ICCTA35431.2014.9521633

PREVENTION OF DOM BASED XSS ATTACKS USING A WHITE LIST FRAMEWORK

Khaled Ali Ayman Abdel-Hamid Mohamed Kholief

College of Computing and Information Technology
Arab Academy for Science and Technology (AAST), Egypt
khaldov@yahoo.com, hamid@aast.edu, kholief@aast.edu

ABSTRACT

Web applications are not guaranteed to be safe for both clients and servers since many vulnerabilities can be exploited in a web application to reach a malicious goal. One such vulnerability is Cross Site Scripting or XSS that has many types but in general is aimed at executing malicious scripts at the client's machine when exploiting vulnerabilities in the server side. Another type of client side XSS vulnerability is called DOM (Document Object Model) Based XSS which can be achieved at client side only without putting any script in the server side. In this paper, we propose a DOM XSS prevention technique that protects the clients from web pages that contain such scripts in the HTML DOM tree source. This is an anti-DOM XSS framework that stops DOM XSS scripts and prevents it at client side. In addition, a prototype tool was implemented which has demonstrated the validity and viability of the proposed framework

Keywords - *Web Application Vulnerabilities, Cross Site Scripting (XSS), DOM.*

I. INTRODUCTION

Nowadays, web applications are inherently dynamic adopting technologies such as DHTML allowing execution of scripts written in scripting programming languages to customize a web application according to the clients' preference. However, such advantages of a dynamic web application may be converted to threats against the visitors -clients- when visiting dynamic vulnerable sites or vulnerable web applications. Web vulnerabilities can be exploited by attackers with attacks against the server side, the client side, or even both server and client side.

The OWASP foundation provided statistics depicting the top 10 web vulnerabilities in 2013 [1]. The top 3 vulnerabilities are identified as injection attack, broken authentication and session management, and Cross Site Scripting (XSS) [1]. XSS occurs when an attacker inserts malicious code in a web application and then the server sends such code to the client browser. For example, the URL field of a web site can be used to insert executable scripts. When the browser executes such malicious code, an attacker can hijack a client session data such as cookies, redirect the client to other web sites, or deface the web site [1]. Static web applications are immune to this type of attack because XSS targets dynamic web sites [2].

XSS attack types can be classified as Reflected (non persistent), Stored (persistent), or DOM-based XSS [6]. In reflected XSS, data containing malicious code can be reflected to the user by a web application without proper sanitization. In stored XSS, the malicious code is persistently stored on the server, for example in a database or guestbook, to be later retrieved by unaware clients. The Document Object Model (DOM) is an application programming interface that represents HTML and XML documents as a hierarchical collection of objects and provides methods for their manipulation [5]. In DOM-based XSS, the tainted data flow is within the browser (client side) causing malicious code to be executed without being retrieved from the web server.

Related research has shown that 56 of Alexa top 100 sites are vulnerable to DOM-based XSS attacks [3]. Server side XSS detection and prevention tools are not applicable for DOM-based XSS as is the case with reflected or stored XSS. Tools for detection and prevention of DOM-based XSS are a few, e.g. [9] [12].

In this paper, a methodology is proposed to prevent DOM-based XSS attacks at the web client side. The approach adopts a white box analysis strategy and hinges on a while list framework. The white list is populated by allowable commands in scripts. The prevention strength and depth is controlled by a user-provided security level (low, medium, and high). The security level controls the inspection level of DOM elements. A prototype tool is implemented to demonstrate the validity and viability of the proposed prevention framework for DOM-based XSS. Performance evaluation results have shown accuracy and efficiency of the proposed methodology.

The rest of this paper is organized as follows. Section II provides background for DOM-based XSS. In addition, section II explores related work for detection and prevention tools regarding DOM-based XSS. Section III details the requirements, analysis, design, and implementation of the DOM-based XSS Prevention technique and prototype. Section IV illustrates the prototype's performance evaluation experiments and analyses results. Finally, section V concludes the paper along with discussion of future work.

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

II. Background And Related Work

Section II.A highlights the danger of DOM-based XSS along with exploitation concepts. Section II.B discusses related work regarding detection and prevention tools of DOM-based XSS.

A. DOM-based XSS: Danger and Exploitation

Figure 1 depicts a simple example of DOM-based XSS attack. Any malicious message within the alert function is considered as a DOM-based XSS Attack. Similar attacks can be used deface a web application.

```
<script> alert('DOM XSS Attack says hi to you!') </script>
```

Figure 1. A simple example of DOM-based XSS

Such a script can be executed after putting it in a vulnerable input form, URL or any vulnerable input and then executing it at the client's browser machine without interfering or noticing from the server. Such attacks target Sources –which are the input sources or initiated codes from outside the page- or Sinks –which can be any functionality that create HTML or interpret string as a JavaScript. In general, sources are points in the program to read untrusted information while sinks are security sensitive operations that may lead to unsecure actions [8].

The DOM can be manipulated via scripting programming languages such as JavaScript [7]. It has to be noted that languages such as Action Script, to some extent Acrobat Script, and especially Visual Basic Script (VBS) can interact with the DOM as well and provide similar features. Accessing the DOM via JavaScript allows application of the JavaScript language features for DOM elements [4]. JavaScript can update the DOM and new data can also contain JavaScript and that makes DOM-based XSS a very dangerous attack [16]. The DOM tree hierarchy structure can be manipulated and modified through DOM-based XSS.

B. Related Work

XSS vulnerabilities, and many other vulnerabilities, can be detected through a black box Approach, a white box Approach, or a hybrid approach.

In a black box approach, the test payload is not included in the response, but rather gets embedded in the DOM as the JavaScript code executes. Such approach tries to solve this challenge by either injecting test payloads into the URL and parameter values, and then observing the side effects of the sandboxed java script execution or by performing dynamic taint analysis via instrumentation of the client code [8].

In a white box Approach it is common for JavaScript content to be loaded dynamically from independent resources that may reside outside. If an attacker embeds a source code script from outside (by e.g. URL or href) it will be hard for the static white box approach to detect it. Another limitation of static analysis is imprecision due to client-side frameworks, such as jQuery and Dojo, whose runtime behavior is hard to model statically due to their inherent complexity, proprietary syntax, and tight interaction with the DOM [8].

The advantages of the white box approach over the black box approach is that the former is very good at finding the flaws if it was well implemented and it is very fast while the latter uses a native interpreter so it has no problem with obfuscation and/or compression issues. A white box approach lacks run time analysis, access to knowledge bases, lots of false negatives and mainly false positives.

DOM-based XSS Detection

There are many offline web vulnerabilities detection scanner tools that are not dedicated to detecting DOM-based XSS, but can do so efficiently, e.g. Acunetix web vulnerabilities scanner [14].

There are a few specialized tools and efforts for the detection of DOM-based XSS vulnerabilities as follows [10].

1) **DOMinator:** A commercial tool based on the Firefox browser with modified Spidermonkey Javascript engine that helps testers identify and verify DOM based XSS flaws [11]. This modified version of the Javascript engine adds dynamic tainting and performs taint propagation tracing. The taint propagation tracing is added to DOM attributes.

2) **DOM XSS Wiki:** The start of a knowledge base for defining sources of attacker controlled inputs and sinks which could potentially introduce DOM-based XSS issues. This project aims also to identify sources and sinks methods exposed by public, widely used Javascript frameworks (see [15]).

3) **DOM Snitch:** An experimental Google Chrome extension that enables developers and testers to identify insecure practices commonly found in client-side code. DOM Snitch is able to identify DOM modification and collect debugging data about such modifications [20].

4) **Ra.2:** is a new black box tool and a FireFox add-on and uses FireFox's Javascript Engine to dynamically execute vectors injected into possible sources to locate most exploitable DOM XSS issues [9]. Ra.2 is basically a lightweight Mozilla Firefox add-on that uses a very simple yet effective and unique approach to detect most DOM-based XSS vulnerabilities [21].

DOM-based XSS Prevention

There are some live prevention methods like sandboxing the script and prevent the attacks from occurring. For example, a browser can limit the amount of damage that malicious code can cause by providing a sandbox in which scripts can only

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

perform web-related actions. Although this sandbox helps to prevent the browser from revealing other information stored on a user's computer, it does not extend that protection to data, such as login credentials and credit card numbers, which users willingly and directly supply.

Another prevention technique is the Same Origin Policy (SOP) which limits a script's access to information. This policy allows scripts from the same origin to access each other's data and prevents access for scripts of different origins. The intent is to prevent a malicious script from breaching the confidentiality and integrity of information pertaining to a different domain [12].

There are some specialized tools for the prevention of DOM-based XSS vulnerabilities as follows.

1) **ConDOM:** When the browser loads HTML or JS, it registers the code's domain of origin in the DomainRegistry before processing. The DomainRegistry maps every domain to a unique bit in a 64 bit label [12]. The request from clients must match any of the registered values in the server of DomainRegistry. ConDOM is based on webkit browser. ConDOM Can stop form data exfiltration and key logging [12].

2) **IceShield:** IceSheild is a live analysis tool which can detect DOM-based XSS by freezing DOM after detecting a malicious script and it can prevent it as well. Based upon a security library –Mario Heidrich Prototypic Library- which is similar to ECMA 5 security sandbox but instead of sealing objects with the Sealed() function, it is freezing DOM objects. IceShield can mitigate detected attacks by changing suspicious elements [13].

3) **NoScript** is a general tool to protect users against many vulnerabilities including DOM-based XSS [17]. It is similar to (NotScript) which provides a high degree of 'NoScript' like control of javascript, iframes, and plugins on Google Chrome [18].

III. DOM XSS PREVENTION METHODOLOGY

This section overviews the proposed DOM XSS prevention methodology (section III.A). In addition, it presents the methodology's design implications (section III.B). Moreover, it contrasts the proposed methodology versus related work (section III.C).

A. overview

The proposed methodology adopts a white box approach and relies on white list database which contains allowable commands in scripts. The white list is consulted to determine the possibility of execution of any commands. A 3-level user-provided security preference is relied upon to determine the strictness of the prevention methodology and the level of the DOM tree inspection. The ability to crawl the DOM tree and inspecting specific nodes are assumed. The objective is to provide live protection for the client. In addition, the ability to replace any identified malicious scripts with a benign version from the white list based on user choice is offered.

The white list Database consists of useful and commonly used scripts by the clients when visiting web sites. Most of these useful scripts are for showing desired information, e.g. using Date() in Javascript to show the current date for the user or using Get attributes in DOM and showing results, e.g. document.readyState to get the status of the page. The white list database wisely includes allowable scripts that show information or use Get attributes, e.g. it does not include showing cookies or any sensitive information. This will cut off several exploitable scripts commands that are made for setting attributes which make the client put values, thus these scripts can be exploited if the values were containing malicious data. However, that does not mean that the white list database cannot contain any scripts to let the user set and put some values, for example changing the background color of the page. This can be allowed but care is needed to add a regular expression check process and consulting the white list. The white list concept allows many useful scripts to be provided by the client. Thus, a client can browse pages and put several useful scripts for his/her preferences. Such process will serve to reduce the number of false positives reported by the prevention methodology.

Table I illustrates the prevention methodology's functional requirements. In general input scripts entered by the client and the DOM tree content of a page are checked through white box analysis and by consulting a white list database for allowable scripts. The possibility of fixing the malicious or suspicious scripts by replacing these scripts by safe scripts from the white list database based on user choice. If the scripts were not malicious, the page's HTML DOM Tree will be crawled through to check if the page itself is malicious or not by getting sensitive nodes which are the fired events nodes to prevent consequent actions when browsing the page. These events nodes are checked versus the white list database contents according to the user-selected security level (Low (default – Medium – High). The security level determines which DOM tree nodes to check and how to check them. In the Low level setting it only checks the onClick & onLoad events because they are usually available in many web sites. Such events are checked versus a tiny black list database to decide whether to approve execution or not. In the Medium level setting the HTML events are checked versus the white list database. Finally, the High level setting is the most defensive level among the three levels because it checks the (HTML events + HTML5 events + some very sensitive nodes tags like <script> and <Iframe>) versus the white list database. The security level setting will provide the client the freedom and the strictness choice according to the client security awareness when navigating web pages and in the same time it can reduce the amount of false positives.

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

The prevention process relies on a linear search through the white list while checking the events nodes in the DOM tree source. This can prove to be costly based on the DOM tree complexity. In the future, we plan to investigate techniques to speed up such search process.

Table I. Prevention Methodology Functional Requirements.

Input	1. URL Address
Processing	1. Take the full URL address and extract the script(s) if there is some script(s) in it and will do o Save the extracted script(s) o Process every single script as a one script o Check the regular expression of the script if that script can set values o Check the script(s) safety according to the allowable white list database o Flag to allow the script if the checking processes verified its safety 2. Crawl the HTML DOM tree for the page and o Get the specified events nodes to check according to the selected security level o Check the specified events nodes according to the selected security level method in checking the nodes o Flag to allow the page if the checking processes for the events nodes verified the safety of these events nodes in the DOM tree source
Output	1. Show the desired link page if it is secure 2. Show to the client that this link or this page is malicious and Prevent it 3. show the possibility of fixing the unsafe script(s) to safe script(s) by letting the client chooses from the allowable white list database to replace the unsafe script(s) o If the client chooses safe script(s) from the white list database to replace the unsafe script(s), the web page will be rendered with these new safe script(s)

B. Design Implications

The prevention methodology pseudocode is illustrated in figure 2.

```
1 Input URL is required for the testing
2 Check Input URL
3 IF URL contains malicious script(s)
4     Prevent browsing it
5     Show (link is unsafe) & only choose to browse it if the client approves so despite warning
6     Show the web page & Show Script(s) Fixation with Replacement option
7 ELSE
8     Check the web page's DOM tree events' nodes
9         IF the page's DOM tree events nodes contain malicious script(s) –according to the selected security level-
10            Prevent to browse it
11            Show (the page is malicious) & only chooses to browse it when if the client approves so despite warning
12            Show the web page & Show Script(s) Fixation with Replacing
13        ELSE
14            Allow to browse the web page
```

Figure 2. Prevention methodology pseudocode

The tables (data structures) maintained during operation are white list, white list for script replacement, and tiny balack list table as follows.

1) **White list:** the main storage of all the allowable commands that can be used in scripts.
2) **White list for script replacement:** Similar to white list and used to let the client select the benign script(s) the client wants to replace instead of the malicious or the suspicious scripts.
3) **Tiny Black List:** This table was designed to be used in conjunction with the "low" security level. The table will provide the prohibited black list commands to be in effect.

C. Proposed Methodology versus Related Work

There are specialized DOM XSS Prevention tools which operate in a similar manner to the proposed methodology in preventing DOM-based XSS from occurring. Table II contrasts such tools versus the proposed work. The proposed methodology can produce a large number of false positives based the chosen security level. However, it attempts to provide flexibility to the security-savvy user through the security level and the manipulation of the white list.

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

Table II. Prevention Tools operation Comparison versus proposed methodology

Prevention Technique	**Advantages**	**Disadvantages**
ConDOM	-Low overhead results and can be extended to include additional browser subsystems [12].	-Before this approach can be adopted, web site authors will need a policy specification language for white listing trusted domains and expressing allowed information flow[12].
IceShield	-The runtime overhead is low [13].	-High false-positive rate which could be lowered by using more elaborated machine learning techniques [13].
NoScript	-Provides the most powerful anti-XSS and anti-Clickjacking protection ever available in a browser [17].	-produces a huge rate of false positive unless the client is aware of the link to allow it.
Proposed methodology	-Ease of use and configuration -It will allow some useful javascripts for the user through a provided decision, based on the White list. -can replace unsafe scripts with safe scripts by selecting some safe script(s) from the white list database. -Can reduce the high rate of false positive by showing to the client the warning message to stop browsing this malicious page or not, if the client thinks it is not a dangerous link.	-Though the white list database makes some reduction for the false positive rates because it allows many useful scripts but the methodology can produce a high rate of false positives. -The more complex the HTML DOM Tree, the slower the processing.

IV. PERFORMANCE EVALUATION

A prototype tool was implemented based on the DOM XSS prevention methodology presented in section III. The tool was implemented in C#. The tool functions as a specialized web browser and in the future can be modified to act as a browser add-on.

In order to have a controlled testing environment and in order to validate obtained results, a testbed was built by creating a malicious HTML web page from scratch with a specific number of vulnerabilities. This page contains 1000 points which in general can lead to malicious purposes. 160 of these points are safe points, but the remaining 840 points are unsafe points (unavailable in the white list). An example of a safe point is for example showing the current date. An example of an unsafe point is to show the cookies for the client –which may end up with such data being sent to an attacker. Figure 3 depicts an example of a safe and unsafe point.

alert(Date()) –This code is an example of a safe point-

alert(document.cookie) –This code is an example of unsafe point-)

Figure 3. Examples of both safe an unsafe points

Safe points, leading to safe actions, need to be stored in the white list. However, There are some safe points in the testbed not saved in the white list and the prototype identifies them as being unsafe points. For example, enlarging an image in a HTML page is a safe action that is not stored in the white list as shown in Figure 4.

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

```
<script>
function bigImg(x)
{
x.style.height="16px";
x.style.width="64px";
}
</script>
```

Figure 4. An example of safe point code not stored in white list

Performance evaluation was conducted against all security levels. Table III depicts performance evaluation results. Note that in this case, the following definitions apply:

1. True Positive (TP) when correctly detected unsafe point(s).
2. True Negative (TN) when correctly rejected safe point(s).
3. False positive (FP) when wrongly detected safe point(s).
4. False Negative (FN) when wrongly rejected unsafe point(s).
5.

Based on the highest security level, the following metrics are calculated.

1. Sensitivity
2. Positive Predictive Value
3. Specificity
4. Negative Predictive Value

The Sensitivity for the highest security level is given by equation 1 as follows.

Sensitivity = [TP / (TP + FN)] * 100 (1)

[838 / (838 + 2)] * 100 = 99.8%

The Positive Predictive Value for the same security level is given by equation 2 as follows.

Positive Predictive Value = [TP / (TP + FP)] * 100 (2)

[838 / (838 + 4)] * 100 = 99.5%

The Specificity for the same security level is given by equation 3 as follows.

Specificity = [TN / (FP + TN)] * 100 (3)

[156 / (4+ 156)] * 100 = 97.5%

The Negative Predictive Value for the same security level is given by equation 4 as follows.

Negative Predictive Value = [TN / (TN + FN)] * 100 (4)

[156 / (156 + 2)] * 100= 98.7%

The results indicate that the proposed methodology achieved promising results. In addition, the highest security level can be reliable and efficient for clients even if they have no experience in security.

For comparison purposes, an online web-based tool called Dom XSS Scanner [19] is run against the built testbed. It was able to detect only (40.11%) of these 1000 points (including the safe points). In addition, it is prone to a higher percentage of false positives (reporting 160 safe points). Such results show the superiority of the proposed methodology with the offered flexibility due to the white list data structure which can store a useful Javascript that a visitor may use.

Table III. Performance Evaluation Results

Security Level	Can Prevent	True Positive	True Negative	False Positive	False Negative
Low Level	Yes	Detected 273 unsafe points (32.5%)	Not detected 5 safe point (3.1%)	Detected 155 safe points (96.9%)	Not detected 567 unsafe points (67.5%)
Medium Level	Yes	Detected 552 unsafe points (65.7%)	Not detected 158 safe points (98.8%)	Detected 2 safe points (1.3%)	Not detected 288 unsafe points (34.3%)
High Level	Yes	Detected 838 unsafe points (99.8%)	Not detected 156 safe points (97.5 %)	Detected 4 safe points (2.5%)	Not detected 2 unsafe points (0.2%)

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

V. Conclusions and Future Work

In this paper, a methodology was presented for the prevention of DOM-based XSS. The methodology relies on white box analysis and adopts a white list framework to allow or disallow execution. In case of unsafe script(s) detection, the client is given the option to replace with safe script(s). A security level is client-provided to dictate the strictness of the prevention methodology and consequently reduce the number of resulting false positives. A prototype tool was implemented which acts as a specialized web browser. Performance evaluation and validation results indicate the validity and viability of the proposed approach. However, a high percentage of false positives can be produced due to the nature of the white box –static analysis- approach.

In the future, the white list database is planned to be expanded to contain more useful Get attributes scripts and adding more useful Set attributes scripts with their regular expressions as discussed earlier in the paper. In addition, the reduction of false positives needs to be further investigated. Moreover, understanding the semantics of a script to recommend a safe replacement for malicious code is an idea for future work

References

[1] OWASP -The Open Web Application Security Project-The Ten Most Critical Web Application Security Risks. [online] at https://www.owasp.org/index.php/Top_10_2013-Top_10 (Last accessed May 2014)

[2] Ismail, O , Etoh M , Kadobayashi Y. ; and Yamaguchi, S., A Proposal and Implementation of Automatic Detection /Collection System for Cross Site Scripting Vulnerability, *in Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA)*, 2004

[3] Stefano Di Paola (Minded Security), Analysis and Identification of DOM Based XSS Issues. [online] at http://www.nds.rub.de/media/attachments/files/2012/01/AnalyzingDOMXss_StefanoDiPaola_Bochum.pdf (Last access October 2014)

[4] Mario Heiderich, Towards Elimination of XSS Attacks with a Trusted and Capability Controlled DOM. PhD Thesis, Ruhr-University Bochum, May 2012.

[5] Philippe Le Hegaret, Lauren Wood, and Jonathan Robie (Editors) What is the Document Object Model, in W3C DOM Level 2 Core Specifications, November 2000.

[6] OWASP Foundation – Types of Cross Site Scripting, [online] at https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting, last accessed May 2014.

[7] GovCertUK , CESG National Technical Authority for Information Assurance Cross Site Scripting Techniques and mitigation.Version 1.October 2007.

[8] Omer Tripp, Omri Weisman, Hybrid Analysis for JavaScript Security Assessment, *in proceedings of the 8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Hungary, 2010.

[9] Ra.2 Black Box DOM detection tool (Presentation) – International Security Conference- www.nullcon.net Last accessed May 2014

[10]OWASP Open Web Application Security Project: DOM Based XSS. Available [online] a https://www.owasp.org/index.php/DOM_Based_XSS. Last accessed May 2014

[11] DOMinator, Tool for analysis of DOM based XSS issues. [online] http://www.net-security.org/secworld.php?id=11057. Last accessed May 2014

[12] Christoph Kerschbaumer, Eric Hennigan, Stefan Brunthaler, Per Larsen, and Michael Franz, ConDOM: Containing the DOM for Safe Browsing. Techincal Report 12-01, Department of Information and Computer Science, University of California Irvine, October 2012.

[13] Mario Heiderich, Tilman Frosch, Thorsten Holz, IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM, *in proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, pp. 281-300, 2011

[14] Acunetix Web Vulnerability Scanner. [online] at http://www.acunetix.com/. Last accessed May 2014.

[15] DOM XSS Wiki, [online] at http://code.google.com/p/domxsswiki/ Last accessed May 2014

[16] Dave Wichers, Unraveling some of the Mysteries around DOM-based XSS, research presentation [online] at http://www.aspectsecurity.com/research-presentations/unraveling-some-of-the-mysteries-around-dom-based-xss. Last accessed May 2014

[17] NoScript Firefex Extension, Available [online] at http://noscript.net/. Last accessed May 2014

[18] NotScript the security extension tool in Google Chrome web store. Available [online] at : https://chrome.google.com/webstore/detail/notscripts/odjhifogjcknibkahlpidmdajjpkkcfn. Last accessed June 2014

[19] DOM XSS scanner online detection tool. [online] at http://www.domxssscanner.com/. Last accessed May 2014

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.

[20] DOMSnitch: A passive reconnaissance tool inside the DOM (experimental). [online] at https://code.google.com/p/domsnitch/wiki/QuickIntro. Last accessed May 2014

[21] Ra.2 – Blackbox DOM XSS Scanner. [online] at https://code.google.com/p/ra2-dom-xss-scanner/. Last accessed May 2014

Authorized licensed use limited to: ANNA UNIVERSITY. Downloaded on August 16,2023 at 09:57:34 UTC from IEEE Xplore. Restrictions apply.