International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 04 | April -2019 ISSN: 1847-9790 || p-ISSN: 2395-0126

DETECT AND PREVENT CODE INJECTION FLAWS USING SCANCIF TOOL

Kanchan Jundare, Suraj Deshmukh, Shweta Patade

Assistant Professor Mr. Swapnil Waghmare

Mahatma education society, Pillai HOC College of Engineering, Rasayani
kanchanjundare7@gmail.com
Mahatma education society, Pillai HOC College of Engineering, Rasayani
surajdeshmukh034@gmail.com
Mahatma education society, Pillai HOC College of Engineering, Rasayani
shwezzpatade10@gmail.com

Abstract: In HTML5-based apps, attackers can deploy attacks. The previous approaches aimed at analysis the data flow to detect sensitive information flows from such call back functions. Besides, analyzing information flows in JavaScript is challenging. We found that the previous approaches are not able to analyze various contexts of callback functions. In this paper, we developed a static analysis tool called SCANCIF to scan code injection flaws. SCANCIF identifies the sensitive callback functions or sensitive keywords and present the report to the users and prevent it by disabling the code from execution.

Keywords: Code injection attacks, HTML5-based website, and data flow analysis.

I. INTRODUCTION

For HTML5-based Web apps, adversaries can take advantage of inject malicious code. When the malicious message is displayed on HTML5-based app the malicious JavaScript code can get executed on a browser. To solve this problem we introduce an automated analysis tool called SCANCIF (SCAN Code Injection Flaws) to detect code injection flaws by scanning sensitive information flows [3]. First, SCANCIF identifies sensitive keywords or callback functions. We picked up possible code injection tags, including verbs, nouns, synonyms and abbreviations to identify sensitive keywords. Moreover, we did not aim at proposing a new approach to data flow analysis to analyze sensitive information flows in JavaScript [1].

Once the sensitive keywords or callback functions are detected it can be prevented from execution.

II. LITREATURE SURVEY

The existing automatic detection tools for code injection flaws in HTML5-based mobile apps presented sensitive plug-in APIs implemented for code injection channels such as Contact, SMS, and Camera. Subsequently, they developed an automatic tool to detect dangerous information flows from such sensitive plug-in APIs to unsafe rendering APIs. By presenting a new text box injection channel and introducing an automatic approach called Droid CIA to scan code injection faults [5]. Droid CIA applied the depth first search (DFS) algorithm to slice sensitive data flow from the text box

Injection channel to sensitive sinks. The existing approaches detected code injection faults based on modelling exactly known sensitive plug-in APIs implemented for code injection channels. Developers can write different plug-in APIs, so their approaches can miss unknown sensitive plug-in APIs. In addition, analyzing data flow in JavaScript is challenging. We manually analyzed source code of HTML5-based mobile apps in and we found that the previous approaches were not able to detect various contexts of call-back functions. PhoneGap could be a framework that's supported the open standards of HTML5 and permits developers to use common internet technologies (HTML, CSS, and JavaScript) to build applications for multiple mobile platforms from a single code base Wang Ruilong [2].

Cross-Site Scripting (XSS) and SQL injection are the highest vulnerabilities found in internet applications. Attacks to these vulnerabilities could have been minimized through placing a good filter before the web application processes the malicious strings. Anyway, enemies could make minor departure from the assault strings so as to not get separated. Checking through all of the

potential attack strings was tedious and causes the online application performance to degrade. In this paper, we implement the use of a hash map as a data structure to address the issue [3].

HTML5-based websites are designed by victimization customary internet technologies like HTML5, JavaScript and CSS Because of their cross-platform support, HTML5-based versatile applications are getting increasingly prevalent. However, kind of like ancient internet apps, they're usually liable to script-injection attacks. It ends up in new threats to code integrity and information privacy. Compared to ancient internet apps, HTML5-based mobile apps have a lot of attainable channels to inject code, e.g., contacts, SMS, files, NFC, and cameras. Even worse, the injected scripts could gain rather more powerful privileges from the mobile apps than those within the ancient internet apps in this paper, we propose a way to deal with distinguish injected practices in HTML5-based Android applications [6].

Our methodology screens the execution of applications, and creates conduct state machines to portray the applications' runtime practices dependent on the execution settings of applications.

When code injection occurs, the injected practices will be recognized dependent on deviation from the conduct state machine of the first application. We prototyped our approach and evaluated its effectiveness mistreatment existing code injection examples. The result demonstrates that the proposed method is effective in code injection detection for real-world HTML5-based Android apps [7].

Realizing that JavaScript is liable to code injection assaults, we have directed a precise report on HTML5-based portable applications, endeavoring to assess whether it is sheltered to depend on the web technologies for mobile app development. Our discoveries are quite surprising. We got wind that if HTML5-based mobile apps become popular–it looks to travel that direction supported this projection–many of the items that we have a tendency to unremarkably do nowadays could become dangerous, including reading from 2D barcodes, scanning Wi-Fi access points, playing MP4 videos, pairing with Bluetooth devices, etc. This paper describes however HTML5-based apps will become vulnerable, however attackers will exploit their vulnerabilities through a spread of channels, and what injury may be achieved by the attackers. In addition to demonstrating the attacks through example apps, we have studied 186 PhoneGap plugins, used by apps to achieve a variety of functionalities, and we found that 11 are vulnerable. We also found two real HTML5-based apps that are vulnerable to the attacks [8].

III. IMPLEMENTED SYSTEM

Our system aims at introducing an automated analysis tool called SCANCIF (SCAN Code Injection Flaws) to detect code injection flaws by scanning sensitive information flows from website. We manually selected the code injection tags from code injection channels that the previous work reported. In addition, SCANCIF is able to analyze various contexts of call-back functions are passed in function.

IV. SYSTEM ARCHITECTURE

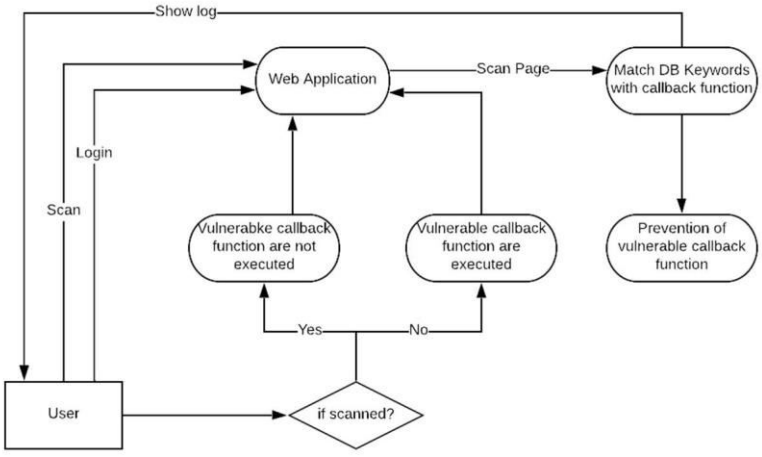

Fig1: Flow of Detection and Prevention of Code Injection Flaws.

Architecture Flow:

Login to the web site.

Run SCANCIF tool.

SCANCIF tool scans the website in order to find tag that is already saved in the dataset.

SCANCIF tool also scans the callback function which can be vulnerable.

Once the vulnerable tags and callback functions are found in order to prevent code injection, the vulnerable tags and callback functions are sliced out.

v. CONCLUSION AND FUTURE SCOPE

SCANCIF identifies sensitive plugin APIs based on code injection tags and we manually select the code injection tags from code injection channels which is further been sliced out. The verification of code injection faults in vulnerable apps is executed by human analysis. In future we can make this tool for various websites where functions can be automated.

REFERENCE

[1] M. Georgiev, S. Jana, V. Shmatikov, "Breaking and fixing origin-based access control in hybrid web/mobile application frameworks," In Network and Distributed System Security Symposium (NDSS), 2014.

[2] PhoneGap: Build amazing mobile apps powered by open web tech. https://phonegap.com.2017.

[3] HTML code injection and cross-site scripting: Understand the cause and effect of XSS. http://www.technicalinfo.net/papers/CSS.html.2014.

[4] A. Charland, B. Leroux, "Mobile application development: web vs. native," In Communications of the ACM, 54(5), 2011, pp. 49-53.

[5] X. Jin, L. Wang, T. Luo, W. Du, "Fine-grained access control for HTML5-based mobile web applications in Android," In Proceedings of16th Information Security Conference (ISC), 2015, pp. 309-318.

[6] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, G. N. Peri, M. Young, "Code injection attacks on HTML5-based mobile apps: characterization, detection, mitigation," In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2014, pp. 66-77.

[7] X. Jin, T. Luo, D. G. Tsui, W. Du, "Code injection attacks on HTML5- based mobile apps," In arXiv preprint arXiv:1410.7756, 2014.

[8] J. Mao, R. Wang, Y. Chen, Y. JIA, "Detecting injected behaviors in HTML5-based Android applications," In Journal of High Speed Networks, 2016, 22(1), pp. 15-34.