Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Effective detection of vulnerable and malicious browser extensions

Hossain Shahriar [c], Komminist Weldemariam [a,b,*], Mohammad Zulkernine [a], Thibaud Lutellier [a]

[a] School of Computing, Queen's University, Kingston, Ontario, Canada K7L 3N6
[b] IBM Research – Africa, CUEA, Langata Road, Nairobi, Kenya
[c] Department of Computer Science, Kennesaw State University, Kennesaw, GA 30144, USA

ARTICLE INFO

Article history:
Received 2 January 2014
Received in revised form
20 May 2014
Accepted 8 June 2014
Available online 19 June 2014

Keywords:
Browser extensions
Web security
Malware
Hidden Markov Model
JavaScript

ABSTRACT

Unsafely coded browser extensions can compromise the security of a browser, making them attractive targets for attackers as a primary vehicle for conducting cyber-attacks. Among others, the three factors making vulnerable extensions a high-risk security threat for browsers include: *i*) the wide popularity of browser extensions, *ii*) the similarity of browser extensions with web applications, and *iii*) the high privilege of browser extension scripts. Furthermore, mechanisms that specifically target to mitigate browser extension-related attacks have received less attention as opposed to solutions that have been deployed for common web security problems (such as SQL injection, XSS, logic flaws, client-side vulnerabilities, drive-by-download, etc.). To address these challenges, recently some techniques have been proposed to defend extension-related attacks. These techniques mainly focus on information flow analysis to capture suspicious data flows, impose privilege restriction on API calls by malicious extensions, apply digital signatures to monitor process and memory level activities, and allow browser users to specify policies in order to restrict the operations of extensions.

This article presents a model-based approach to detect vulnerable and malicious browser extensions by widening and complementing the existing techniques. We observe and utilize various common and distinguishing characteristics of benign, vulnerable, and malicious browser extensions. These characteristics are then used to build our detection models, which are based on the Hidden Markov Model constructs. The models are well trained using a set of features extracted from a number of browser extensions together with user supplied specifications. Along the course of this study, one of the main challenges we encountered was the lack of vulnerable and malicious extension samples. To address this issue, based on our previous knowledge on testing web applications and heuristics obtained from available vulnerable and malicious extensions, we have defined rules to generate training samples. The approach is implemented in a prototype tool and evaluated using a number of Mozilla Firefox extensions. Our evaluation indicated that the approach not only detects known vulnerable and malicious extensions, but also identifies previously undetected extensions with a negligible performance overhead.

* *Corresponding author*. School of Computing, Queen's University, Kingston, Ontario, Canada K7L 3N6.
E-mail addresses: weldemar@cs.queensu.ca, k.weldemariam@ke.ibm.com, komminist@gmail.com (K. Weldemariam).
http://dx.doi.org/10.1016/j.cose.2014.06.005
0167-4048/© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Browser extensions have become an integral part of Web browsers (e.g., Mozilla Firefox, Google Chrome) to enrich the browser with various functionalities. Extensions are becoming popular and are the main presentation point for all of the web contents. For instance, as of July 2 of 2013, more than three billions[1] extensions have been downloaded only for Mozilla Firefox browser (Mozilla Firefox, 2012a) with over 60 million daily extension users (Lerner et al., 2013). At the same time, every web user relies on these pieces of software for everyday tasks.

Unfortunately, extensions are frequently targeted by attackers. As a result, attacks such as Cross-Site Scripting (XSS) and SQL injections are still common in browser extensions. One of the reasons for this is the presence of potential vulnerabilities in extensions and some of them are also malicious by design (Bandhakavi et al., 2011; Louw et al., 2008; Kirda et al., 2006; Barth et al., 2010; Wang et al., 2012; OWASP, 2013). In addition, todays' exploitation strategies are remarkably effective as they exploit vulnerable extensions to deploy malicious code, infect new victims, join botnets, or systematically compromise entire netblocks using automated attack kits deployed by the blackhats (see, e.g., in Householder et al., 2002; Dagon et al., 2008; Provos et al., 2008). The common aspects of all these attacks are that they are carried over the web. More importantly, these attacks attempt to penetrate into the victims' computer by taking advantage of vulnerabilities exposed by their browsers or installed browser extensions (Wang et al., 2006; Stuart Schechter et al., 2007; MSISAC, 2013; Ford et al., 2009; Seo and Lam, 2010).

In addition, most extensions are thoroughly checked by a team of security professionals before they are hosted on trusted websites for distribution. However, various reports confirmed that the prevalence of vulnerable and malicious browser extensions is on a continuous rise (Symantec, 2011, 2012; Bandhakavi et al., 2011; Chufeng and Qingxian, 2011; Chen et al., 2011; Grier et al., 2012; Eshete et al., 2012). We also observed an increased number of security breaches in industry and government organizations—e.g., see Symantec (2011, 2012), McCarthy (2013), and MailOnline (2013). These trends show that extensions often go through checking mechanisms without being detected. Note that once an extension has been installed, it can enjoy the same privilege level (e.g., read, write, and/or modify) as the browser itself (Barth et al., 2010). This way extensions can get access to local filesystem and other sensitive resources through critical APIs.

While extensions (be them benign, vulnerable or malicious) interact extensively with arbitrary webpages, it is important to ensure that they are checked for vulnerabilities and maliciousness before installing them to mitigate (some of) the unwanted consequences. For this purpose, so far a number of automated analysis and detection techniques have been proposed. These include static and dynamic information flow analysis to check suspicious data flows from sources to sinks in vulnerable extensions (Bandhakavi et al., 2011; Dhawan and Ganapathy, 2009; Djeric and Goel, 2010). Some approaches restrict the privilege of APIs that could be invoked by malicious extensions (Barth et al., 2010), generate and validate digital signatures for benign extensions so that they can be checked at runtime for the presence of malicious extensions (Louw et al., 2008), and monitor process and memory level activities against a set of behaviors of benign extensions (Kirda et al., 2006). Barua et al. (2013) presented an approach to differentiate between legitimate and malicious JavaScript code supplied through unsanitized user inputs to Firefox extensions using a code randomization and point-to analysis techniques. A recent work that allows Firefox users to specify policies for extensions and offers runtime enforcement of those policies is discussed in Onarlioglu et al. (2013). A user could specify that extensions are allowed to read from the filesystem and password manager but not allowed to write to either. Additionally, the user can use predefined policies or specify a policy per extension, giving a great deal of control up to the user.

This article presents our approach for detecting browser extension types by widening and complementing prior works. Our hypothesis is that the type of a browser extension can be identified by thoroughly analyzing its distinguishing features while in operation. These features help determine the behaviors of the extension and thereby detect its type automatically. Benign, vulnerable, and malicious extensions can create, read, and write to local machine and browser specific resources based on a set of API calls. An API invocation may or may not be related to user interactions. We assume that a benign extension sanitizes user supplied inputs, and performs actions based on events from users. Our specific attention is to find out the presence of any API that can access sensitive resources of the browser (e.g., cookie, password manager) and local system (e.g., file, memory, process) between event handler invocation and content generation process. The major difference between malicious extension and the others is usually in the visibility of the user interface and actions that are performed without user supplied events. Benign and vulnerable extensions can also be differentiated based on the presence of input filtering mechanisms.

To verify our hypothesis, in this article, we examined a set of common and distinguishing functionalities that benign, vulnerable, and malicious extensions can perform. Three independent models for each extension type were built based on Hidden Markov Model (HMM) constructs (Poritz, 1988). The essential entities (e.g., state, observation sequence) of the models are built by utilizing the identified characteristic of benign, vulnerable and malicious browser extensions and our prior experience. We then used a set of extensions (training samples), which are collected from various extension sources (such as Mozilla Add-ons repository, Bugzilla reports (Bugzilla, 2013), other websites that report malicious extensions and related literature) to train the three models. Vulnerable and malicious extension samples are specifically difficult to find (also noted elsewhere (Barua et al., 2013; Onarlioglu et al., 2013)). Hence, we defined rules and applied them to generate additional training samples such that the detection would be more accurate and efficient. The models were trained and evaluated using randomly selected set of benign, vulnerable, and malicious Firefox browser extensions (test samples). Our approach detected most of the extension types successfully by monitoring features related to user activities (e.g., click operation), visibility of operation source (e.g., button, menus

[1] Note that we cannot verify the number of downloads are unique.