Analysis of Vulnerabilities of Web Browser Extensions

Charlie Obimbo
School of Computer Science
University of Guelph
Guelph, ON, Canada
cobimbo@uoguelph.ca*

Yong Zhou
School of Computer Science
University of Guelph
Guelph, ON, Canada
yzhou09@uoguelph.ca

Randy Nguyen
School of Computer Science
University of Guelph
Guelph, ON, Canada
nguyenr@uoguelph.ca

***Abstract*—Web browser extensions are popularly used to add extra functionality to web-browsers such as Safari, Chrome, Firefox, and Edge. These functionalities may include activity, entertainment, and security. However, many of these web extensions are not coded by security experts and thus they are targets for hackers when conducting cyber-attacks.**

This paper is based on a security review of 50 chrome extensions, 25 popular and 25 random. A study is done on their vulnerabilities through privilege granting and escalation, and suggestions are given on how to ameliorate the development and use of extensions, while reducing the risk-factors.

***Index Terms*—Web-browser extensions, Privilege Separation, Cyber-attacks.**

I. Introduction

The most popularly used web browsers are: Google Chrome, Mozilla FireFox, Edge and Apple Safari [4]. Across these browsers, there are thousands of extensions that could help users in many different ways such as ad blocking, converting youtube videos into .mp3, and remembering usernames and passwords for various sites. Web browser extensions take the vanilla-plain web-browsers and add extra functionality, like activity, entertainment, and security to one's browsing experience. Some examples of extensions and their use are found in Table 1 below.

Although, these browser extensions can help an everyday user with ease of use and productivity, many of these extensions could be coded in an unsafe manner which could actually compromise the security of these browsers [3]. Many of these web extensions are not coded by security experts these web browser extensions are a target for hackers when conducting cyber-attacks [3]. The reason for most of these attacks is that some browsers use full access privileges and hackers may exploit this fact. Many of the common extensions do not need full privilege access and looking into this issue may be a solution.

A. Recent Incidences

In May, 2018 Dan Goodin of Ars Technica reported that criminals infected more than 100,000 computers with browser extensions that stole login credentials, covertly mined cryptocurrencies, and engaged in click fraud. To make matters worse, the malicious extensions were hosted in Google's official Chrome Web Store [9].

TABLE I
Examples of Web-browser Extensions.

1.	annotate-page	Displays a sidebar that lets you take notes on web pages.
2.	beastify	Adds a browser action icon to the toolbar. Click the button to choose a beast. The active tab's body content is then replaced with a picture of the chosen beast.
3.	commands	Demonstrates using the commands API to set up a keyboard shortcut.
4.	latest-download	Shows the last downloaded item, and lets you open or delete it.
5.	list-cookies	This extensions list the cookies in the active tab.
6.	emoji-substitution	Replaces words with emojis.
7.	LastPass	Easily Access All Passwords in a Secure Place! Generates Secure Passwords. Autofills Passwords.
8.	Evernote Web Clipper	keeps your notes organized. Memos are synced so they're accessible anywhere, and searchable so you always find what you need.
9.	Save to Pocket	lets you save web pages and videos to Pocket in just one click

Developers may also be approached by companies, offering to buy their extensions for a large sum of money. The new buyers can then add malware to the automatically updating extensions, and then use it to as an adware, as happened to *Particle*, a popular Chrome extension for customizing YouTube videos [10] in July 2017. Other vulnerabilities have been pointed out in [11, 12, 13].

This paper analyzes browser extension vulnerability, and security measures that could be taken in order to secure browsers from high-risk attacks. The main subject of it's interest is the google chrome extension security system and its different levels of permission partition, content script, extension core, and native binary. It also suggest a remedy.

One effective way that has been suggested in literature of remedying is by curbing the privileges the extension have. Instead of running with the user's full privileges all the time, the extensions system are limited to a set of privileges when the extension is installed. If an extension later becomes

978-1-7281-1360-9/18/$31.00 ©2018 IEEE
DOI 10.1109/CSCI46756.2018.00029

infected, the extension will be unable to increase this set of privileges [5].

The research focuses on extensions that are created by regular programmers: it is assumed that the developer is not a web-security expert. The attacker attempts to hijack the extension and upgrade its privileges. For example, the attacker might be able to install malicious software on the user's machine if the extension has been given privileges for arbitrary file access. It is also assumed that the attacker is unable to convince the user into downloading or running executable programs. Further, it is assumed the browser itself is vulnerability-free [5].

The doing of this research is of paramount importance due to the fact that most people with access to the Internet do web browsing daily with the use of extensions that contain important information pertaining to banking information, emails, social media and gaming accounts, and many more. It is very interesting how using something as simple a web extension can lead to the risk of malware, hijacking, and manipulation of programs and data [1].

II. Related Work

The current extension system protect users from unsecure extensions by designing least privilege, privilege separation, and strong isolation into the extension system. Instead of the extension itself running with the user's full privileges all the time, extensions in current system are limited to a set of privileges chosen at the time when they are installed. If an extension later become infected by malware, the extension will be unable to increase this set of privileges and gaining increased access to the host's machine. The permission to execute foreign code will often be unavailable to an attacker who compromises an extension in our system. In addition to limiting the overall privileges of each extension, our system further reduces the attack surface of extensions by forcing developers to divide their extensions into three components: content scripts, an extension core, and a native binary [5].

- Native Binary is an optional component of the our extension system that may access the machine with user full privileges. It uses the Netscape Plugin Application Programming Interface (NPAPI), an API that allows the use of web plugins. [5]
- The extension core contains the majority of the extension privileges, the extension core can only interact with web content via `XMLHttpRequest`, this is an object type that allows the retrieval of data from URL without the need to do a full page refresh. The core also interaction content scripts. However the extension core does not have access to the host machine. [5]
- Content Script has direct access to the Document Object Model (DOM) of a single web page, because of this, it is exposed to potentially malicious input. However, content scripts have no other privileges except for the ability to send messages to the extension core. [5]

To access the user's full web site privileges, a hacker would need to get the extension to forward malicious injections from the content script to the extension core and from the extension core to the native binary, where the input would need to exploit a vulnerability. We argue that exploiting such a multi-layer vulnerability is more difficult than exploiting a simple cross-site scripting hole in a Firefox extension.[5]

III. Methodology

Fifty google chrome extensions were selected from the official chrome store kust u (refer to appendix A). Only extensions with were "open source", or ones that were easily accessible through chrome developer inspection were chosen from these extensions.

A number of extensions were manually reviewed, using the following security review process:

- **Black-box testing**. Each extension's user interface was run and the network traffic and behaviour were observed. Note was taken for any instances of network data being inserted into the DOM of a page. This was done using ip4 config and Wireshark to monitor traffic. After monitoring the extension for awhile, malicious data was inserted into its network traffic (including the websites it interacts with) to test the extension's robustness and correct functionality
- **Source code testing**. The extensions' source code was studied to see whether data from an untrusted source could be used all the way to the execution sink. This was done in combination with the black box testing. The source code of the extension was found from open source projects, Github and Chrome developer, and the code was copied into into a text viewer and manually review the source code, grep was used to search for any additional sources or sinks that might have been missed. For sources, one looked for static and dynamic script insertion, XMLHttpRequests, cookies, bookmarks, and reading websites' DOMs, and for sinks – uses of eval, setTimeout, document.write, innerHTML.
- **Holistic testing**. The behaviour from of the source code was observed for each extension programs in black-box testing. With the combined knowledge of an extension's source code, network traffic, and the extensions intended described functionality, an attempt was made to identify any additional behaviour that had previously been missed.

From the analysis types of vulnerability were grouped into categories, *extension add vulnerability, content script vulnerability* and *core vulnerability*. Since content script interact with the extension core closely in the chrome security model some vulnerabilities in the content script may be extended to the core as well. If the vulnerability was detected in the core it was considered a high priority threat since the core would direct access the hosts machine and would have the highest level of privilege.

A. Vulnerabilities Found

Table II below, categorizes the type of vulnerability by the location and how it may be exploited. A pattern found is that

TABLE II
VULNERABILITIES FOUND

Vulnerable Component	Popular Extension	Random Extension
Core Vulnerability	6	7
Content Script Vulnerability	0	1
Extension Add Vulnerability	6	3
Any (Holistic)	11	9

these vulnerability may be exploited by a type of network attack rather then web attack this may suggest that the security factor in place already by chrome may be focused on detecting web attacks.

IV. EVALUATION

Content scripts are JavaScript files that run in the context of web pages inside of a special environment called an isolated world. This mechanism preserves the integrity of content scripts from malicious sites. In the security study of isolated worlds it was found that only 3 of the 50 extensions have content script security vulnerabilities, with 2 extensions only able to execute arbitrary code. When interacting with websites, programmers are faced with four key security challenges: click injection, eval, data as HTML, and prototypes and capabilities. The relationship between the isolated worlds mechanism against vulnerability divisions is evaluated.

- **Click Injection**. Extensions are able to listen and handle DOM elements of a website. An example of click injection, for example, is an extension that listened and handled an onClick event from a malicious website instead of from the user, compromising the behaviour of content scripts. The isolated worlds mechanism does not prevent this sort of attack whatsoever. We find that there were no vulnerabilities within 10 Google extensions, because most buttons simply change the state of the user interface.
- **Eval**. An eval is able to execute code within a content script that will run inside an isolated world using untrusted information. Through this method, the isolated worlds mechanism does not protect against the flaws using eval. It was found that only 4 extensions use eval inside their content scripts that manipulate strings into executable code.
- **Prototypes** and **Capabilities**. Javascript prototype poisoning and capability leaks was a big cause of vulnerabilities within many Firefox extensions and bookmarklets in the past [6, 7, 8]. Through heap separation, the isolated worlds mechanism can prevent both types of attacks which are no longer possible using Google Chrome if the mechanism is fully functional. It was found that there were 2 extensions that were open to prototype poisoning or capability leaks from our manual review of the extensions, but it was hard to conclude whether many vulnerable extension may have been missed or there may have been an improper evaluation of the effectiveness of isolated worlds and these vulnerabilities.
- Data as HTML. Inserting data as HTML is a common web development error as it allows for untrusted data to be injected into a page and running as code. In content scripts, the isolated worlds mechanism prevents this attack by placing the content scripts inside its own environment, allowing extensions to read a websites DOM elements, modify it, and place it back into the page without compromising the integrity of the content script itself. As a common error, it was expected that many extensions had this kind of security vulnerability, but it was found that only 4 extensions had this type of flaw.

Privilege Separation is a technique used to divide the privileged core extension from the rest of its program that each has its own specific privileges and tasks to mitigate the possibility of malicious attacks. This technique is backup defensive system in case of isolated worlds failing. Even if a content script is rendered vulnerable to attack, privilege separation should stop an attacker from executing code using the permissions granted by the core privileges.

In one extension, it was found that the core extension was able to send messages to its vulnerable content script, requesting to exercise the privileges granted. An `XMLHttpRequest` (`XHR`) is an API whose methods allow the transfer of data between a web server and browser. This request was run from the content script to the core and back, triggering an arbitrary XHR that creates vulnerability. This means that the privilege separation technique is only partially successful on stopping the attack, since the attacker could not get full privileges from the core.

It was found that 35 out of 50 extensions used content scripts that were not vulnerable to code injection. In the ones where attacks were possible to successfully manipulate the content scripts, permissions were granted to the attacker through passing messages with the core extension. Table III depicts the number of vulnerable content-scripts in the extensions and the severity of the vulnerabilities, in terms of what permissions were granted by the scripts to the attackers.

TABLE III
VULNERABILITIES FOUND

Permissions	Number of Content Scripts
Success: All permissions of the extension	2
Partial: Cross-Origin Resource Sharing	4
Partial: Tab Control	3
Other	3

Since the isolated worlds mechanism is highly effective against malicious attacks overall, only 21 extension required the protection of privilege separation as a backup defensive system. Therefore, a scenario was instigated which answered the question of “would privilege separation mitigate the vulnerabilities within content scripts that contained vulnerability

flaws?" Using the list of 50 extensions, 35 used content scripts. Next, the message-passing channel between the core extensions and content scripts was explored, and it was determined that 35% of the scripts can communicate with their core, allowing them to abuse their privileges. Table III shows the number of content scripts and their accesses to permissions with their core.

As a result, we can say that the privilege separation mechanism is an effective backup defensive mechanism that would prevent core privilege access if isolated worlds were to fail using this scenario.

V. Conclusion

A security review of 50 chrome extensions was performed, some popular while others were random.

Around 40% of the extensions were found to have security compromises. Based on the found vulnerability studies were done on how Chrome's security machines function in order to study their effectiveness.

The isolated world security system is effective, it prevents regular programmers making common errors that may cause security flaws, this effective system lessens the need for privilege separation. However, privilege separation is still a needed mechanism in case that isolated world fails or is not present. Permission privileges also has positive impact on the overall system security, developers can use permissions to reduce the scope of their vulnerability compressed.

Although Google has already put in place good systems that can mitigate security breaches, web developers may not always use them effectively, there are cases from the source code analysis where developers would expose their systems accidentally in spite of the benefits of the security measures. Programs would also sometimes make their extension ask more permission than required, automated tools in certain APIs may help developers develop better security programming practices.

References

1) Louw, M. T., Lim, J. S., & Venkatakrishnan, V. N. (2008). Enhancing Web Browser Security Against Malware Extensions. Journal in Computer Virology, 4(3), 179-195. doi:10.1007/s11416-007-0078-5 https://doi.org/10.1007/s11416-007-0078-5
2) Bandhakavi, S., Tiku, N., Pittman, W., King, S. T., Madhusudan, P., & and, M. (2011). Vetting browser extensions for security vulnerabilities with VEX. Communications of the ACM, 54(9), 91. doi:10.1145/1995376.1995398 http://static.usenix.org/event/sec10/tech/full_papers/ Bandhakavi.pdf
3) Shahriar, H., Weldemariam, K., Lutellier, T., & Zulkernine, M. (2013). Effective detection of vulnerable and malicious browser extensions. 2013 IEEE 7th International Conference on Software Security and Reliability. doi:10.1109/sere.2013.32 https://journals-scholarsportal-info.subzero.lib.uoguelph.ca/pdf/01674048/v47icomplete/66_edo vambe.xml
4) Liu, L., Zhang, X., Yan, G., Chen, S. (2012). Chrome Extensions: Threat Analysis and Countermeasures. In 19th Annual Network & Distributed System Security Symposium. " https://pdfs.semanticscholar.org/fde1/a4806a8c fef90f1cc913156a13de81063f62.pdf.
5) Adam, B., Adrienne Porter Felt., Prateek Saxena. Protecting Browsers from Extension Vulnerabilities. http://webblaze.cs.berkeley.edu/papers/Extensi ons.pdf
6) Adida, A. Barth, and C. Jackson. Rootkits for JavaScript Environments. In Web 2.0 Security and Privacy (W2SP), 2009.
7) R. S. Liverani and N. Freeman. Abusing Firefox Extensions. Defcon17.
8) S. Willison. (2005) Understanding the Greasemonkey vulnerability. http://simonwillison.net/2005/Jul/20/vulnerability.
9) Dan Goodin CHROME'S ACHILLES' HEEL – Malicious Chrome extensions infect 100,000-plus users, again. May 10, 2018. https://arstechnica.com/information-technology/2018/05/malicious-chrome-extensions-infect-more-than-100000-users-again.
10) Catalin Cimpanu. "Particle" Chrome Extension Sold to New Dev Who Immediately Turns It Into Adware. July 13, 2017. https://www.bleepingcomputer.com/news/security/-particle-chrome-extension-sold-to-new-dev-who-immediately-turns-it-into-adware/.
11) Agarwal, Chinmay, Medhavini Kulshrestha, and Himanshu Rathore. "Security Verification in Web Browser Extensions." (2018).
12) Sagar, Deepika, et al. "Studying Open Source Vulnerability Scanners For Vulnerabilities in Web Applications." IIOAB JOURNAL 9.2 (2018): 43-49.
13) Sanchez-Rola, Iskander, Igor Santos, and Davide Balzarotti. "Extension breakdown: security analysis of browsers extension resources control policies." 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, Vancouver, BC. https://www. usenix.org/conference/usenixsecurity17/technicalsessions/presentation/sanchez-rola. 2017.