International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 5 May 2022, pp: 2593-2596 www.ijaem.net ISSN: 2395-5252

Google Chrome Extension Development: Security, Analogy and Technologies Used

Jainisha Chauhan[1], Avinash Gupta[2], Alka Singh[3], Preeti Singh[4], Mradul Dixit[5]

[1,5]*Student, Babu Banarasi Das Engineering College, Lucknow, Uttar Pradesh*
[2]*Professor and Head, Babu Banarasi Das Engineering College, Lucknow, Uttar Pradesh*
[3,4]*Assistant Professor, Babu Banarasi Das Engineering College, Lucknow, Uttar Pradesh*

Submitted: 20-05-2022 Revised: 28-05-2022 Accepted: 30-05-2022

ABSTRACT: Google Chrome extensions are apps that can be installed in Chrome to enhance its functionality. This can include adding new capabilities to Chrome or changing the program's existing behaviour to make it more user-friendly. Chrome extensions are usually built using technologies such as HTML, CSS and JavaScript. Because extensions have specific privileges within the browser, they are a tempting target for hackers. In designing its own extension architecture, Google Chrome integrates security concepts thus overcoming the limitations of legacy extensions.

KEYWORDS: Google Chrome Extension, Architecture, Security, HTML, JavaScript

I. INTRODUCTION

A web browser extension enhances browsing experience by adding functionality to the browser usually in the form of additional toolbars, context menus or user interface customization. Executable plugins to interpret certain MIME formats (e.g., PDF readers, ActiveX, Flash players), browser helper objects, and scriptable extensions created largely in JavaScript, HTML, and CSS are all examples of extensions.

Extensions provide a wide range of capabilities, including presenting specific data based on the user's preferences, customising the generated Web page, accessing and even changing security and privacy sensitive data, developing and debugging Web apps, and much more. Figure 1 shows an extension displaying a scientific calculator upon clicking a particular icon. Some extensions grow so popular that they are eventually included as standard features in the main browser. Third-party developers create browser extensions, which are widely available and improve the browsing experience for end-users by allowing them to personalise the available features by installing multiple extensions. Extensions contribute to the development of a developer community for the concerned browser platform, and hence to the popularity of the Google Chrome browser.

Most modern Web browsers export privileged APIs that allow extension developers to access sensitive resources such as file systems, passwords, cookies, networks, and more. Because of this unlimited access to critical resources, JavaScript(JS)-based extensions can run with the privileges of hosting principals, such as a Web browser. As a result, browser extensions are fundamentally different from Web applications, which are restricted by concepts such as the same-origin policy and have limited authority. Therefore, these extensions offer a greater risk to end users than Web 16 programmes, because the benign-but-bugged extensions can be exploited by remote attackers to seize control of the entire Web browser.

Figure 1. A real world google chrome extension of a scientific calculator

Browser makers are required to maintain authorised sites for hosting extensions because of the high security risk. The Chrome Web Store is where users can find and install extensions for Google Chrome while the official site for Firefox extensions is the addon gallery. Developers make their extensions accessible for download in the

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 5 May 2022, pp: 2593-2596 www.ijaem.net ISSN: 2395-5252

addon gallery and web shop, just like they do in other app stores for iOS and Android. In addition to these authorised places, extensions can also be installed manually by a user or an external program. These extensions, unlike those on official sites, are not subjected to a thorough security evaluation. Due to potential security risks, browser providers normally prohibit end-users from obtaining and installing extensions from untrusted sources.

II. SECURITY MODEL OF CHROME EXTENSIONS

Anticipating browser security issues, the Google Chrome extension platform was created to safeguard users from bugs in otherwise harmless addons [14]. It includes three security mechanisms:

A. **Privilege separation :** Every Chrome extension comprises two types of components: content scripts (zero or more) and core extension (zero or one). Content scripts read websites and make changes as needed. The main extension includes browser UI components, long-running background jobs, an options page, and other functionality that are not directly related to websites. Separate processes run content scripts and core extensions, and they communicate by sending structured clones across an authorised channel. Each website gets its own isolated instance of a certain content script. Chrome's extension API is accessible to core extensions, but not to content scripts.

Content scripts are the most vulnerable to attack since they interact directly with webpages, making them low-privilege. Higher-privilege is the protected core extension. As a result, a content script compromise that does not extend beyond the message-passing channel to the higher-privilege core extension does not constitute a substantial threat to the user.

B. Isolated worlds : The separated worlds approach is designed to keep web attackers away from content scripts. A content script can read or edit the DOM of a website, but the content script and the website both have their own JavaScript heaps and DOM objects. As a result, web pages and content scripts never exchange pointers. This should make tampering with content scripts on websites more difficult.

C. Permissions : Extensions can't access sections of the browser API that affect users' privacy or security by default. A developer must declare the required permissions in a file packed with the extension in order to have access to these APIs. An extension, for example, must request permission to read or change the user's bookmarks. Extensions' use of cross-origin XML Http Requests is similarly limited by permissions; extensions must declare the domains with which they intend to interact. Permissions are only available to the core extension. Browser APIs and cross-originXHRs are not available to content scripts. A content script can only access the website it is executing on and deliver messages to its core extension.

Permissions are used to protect against vulnerabilities in core extensions. An attacker cannot request new rights for a hacked extension since it is limited to the permissions that its developer requested. As a result, the severity of an extension's vulnerability is confined to the API calls and domains that the permissions permit.

Google Chrome was the first browser to provide features like privilege separation, isolated worlds, and extension permissions. These safeguards were designed to make Google Chrome extensions safer than Mozilla Firefox extensions or Internet Explorer browser helper objects. [14].

III. ARCHITECTURAL FEATURES OF LEGACY EXTENSIONS AND CHROME EXTENSIONS

The architectural elements of legacy framework and google chrome extension clearly illustrate why legacy framework has security issues and how the architectural design of chrome extensions overcome them.

Legacy Extension Architecture : Open technologies like HTML, CSS, JavaScript, and XUL are commonly used to create legacy extensions. To access system resources and perform valuable functions, these extensions frequently leverage privileged browser APIs like XPCOM. In terms of API interaction, there are two types of JavaScript code in extensions: privileged JavaScript code (chrome script) that accesses XPCOM and unprivileged JavaScript code (content script) that interacts with untrusted Web content on Web pages. However, Firefox's original extension design contains several flaws that make it vulnerable. Some of these are briefly discussed below:

- **Unified JavaScript heap :** Both unprivileged content scripts and privileged chrome scripts execute in the same heap in Mozilla's legacy extensions, increasing the risk of shared references. The interface has been attacked [8,

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 5 May 2022, pp: 2593-2596 www.ijaem.net ISSN: 2395-5252

12] by attackers due to the limitations of the isolation mechanism that seeks to segregate untrusted references of the content JavaScript from the chrome JavaScript. In such circumstances, the attacker could modify shared object references and influence the execution of the privileged code within the extension if the user navigates to a malicious Web page. Privilege escalation scenarios like this have already been utilised to exploit insecure extensions [13].

- **Chrome DOM :** The chrome DOM stores the visual representation of the browser's user interface, which includes toolbars, menus, the status bar, and icons. Chrome scripts, like JavaScript code on a Web page that can access the page DOM, can access the chrome DOM and programmatically manipulate the browser's complete UI.
- **Privileged objects :** The global window object and its properties are available to all chrome scripts by default. The Components object is a window-specific property that grants access to the browser's private XPCOM APIs. An attacker who obtains a reference to the Components object has complete control over the browser and can access any system resources. As a result, in a shared heap environment, the availability of Components to all scripts by default dramatically enhances the probability of vulnerability exploitation.

JavaScript is used to write parts of the browser and addons. With a monolithic heap and no isolation primitives in the language, legacy extension security is primarily dependent on the discretion and competence of extension developers. Many previous studies [7, 8, 9, 10] have demonstrated the downsides of legacy extensions and highlighted design flaws in legacy architecture.

Google Chrome Extension Architecture:

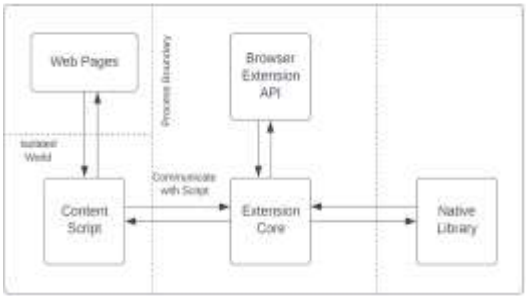

Figure 2. Architecture of Google Chrome Extension

Google Chrome integrates security concepts when creating its own extension architecture [8], addressing the security vulnerabilities of legacy Firefox extension architecture. By dividing the extension into three separate processes: content scripts, extension core, and native binaries and it seeks to protect users from vulnerabilities in benign-but-bugged extensions using three security principles: POLA, privilege separation, and strong isolation.The Chrome extension architecture is depicted in Figure 2 at a high level. It comes with three main security features which have already been discussed in section II (Security Model of Chrome Extensions).

IV. TECHNOLOGIES FOR DEVELOPING GOOGLE CHROME EXTENSIONS

HTML/CSS parsers, layout and rendering engines, JavaScript interpreters, network protocol stacks, and storage layers are all important components of Web browsers. The following is a rundown of some of the most important Web technologies that are at the heart of browser extensions and Web apps.

1. HTML/CSS : All Web documents are primarily written in HTML [6] and CSS [5]. While HTML, a declarative markup language, describes the structure of the Web document, CSS dictates its presentation and style. Elements in the HTML document form a tree structure that is internally represented and can be manipulated by a programming API known as Document Object Model(DOM) [4]. A node in the DOM tree represents each HTML tag on a Web page.Each DOM node additionally contains any application-defined event handlers for GUI activity, as well as the accompanying CSS data.

2. JavaScript : JavaScript [3] is a dynamic, lightweight, interpreted language that has become the de facto standard for Web and, in particular, browser client side scripting. It's prototype-based and object-oriented, and it lets you handle functions as first-class objects. It allows Web pages to change their HTML DOM structure, CSS style attributes, and displayed content dynamically.

REFERENCES

[1] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In Proceedings of the USENIX Security ’12. USENIX Association.

[2] Guanhua Yan Lei Liu, Xinwen Zhang and Songqing Chen. Chrome extensions: Threat analysis and countermeasures. In

Proceedings of the 19th Network and Distributed System Security Symposium.
[3] David Flanagan. JavaScript: The Definitive Guide. O'Reilly Media Inc.
[4] Document object model. www.w3.org/DOM.
[5] World Wide Web Consortium. CSS. http://www.w3.org/Style/CSS/ Overview.en.html.
[6] World Wide Web Consortium. HTML. http://www.w3.org/html/.
[7] Sruthi Bandhakavi, Samuel T. King, P. Madhusudan, and Marianne Winslett. Vetting browser extensions for security vulnerabilities with VEX. CACM, 54(9), September 2011.
[8] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. In Proceedings of 13th Network and Distributed System Security Symposium, NDSS '10. The Internet Society, 2010.
[9] Mohan Dhawan and Vinod Ganapathy. Analysing information flow in javascript based browser extensions. In ACSAC, 2009. 96 Vladan Djeric and Ashvin Goel. Securing script-based extensibility in web browsers. In Proceedings of USENIX Security 10.
[10] Google Chrome APIs. http://developer.chrome.com/extensions/api_ index.html.
[11] Google. Web APIs. http://developer.chrome.com/extensions/api_ other.html.
[12] Addon SDK. Content proxy. https://addons.mozilla.org/ en-US/developers/docs/sdk/latest/dev-guide/guides/content-scripts/accessing-the-dom.html.
[13] Greasespot: The weblog about Greasemonkey. http://www.greasespot.net.
[14] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting Browsers from Extension Vulnerabilities. In Network and Distributed System Security Symposium (NDSS), 2010.