Malicious Browser Extensions

Author: Nicholas Teleky
Nicholas.Teleky@tufts.edu

Supervisor:
Ming Chow

December 11, 2013
COMP116 – Computer Security

Table of Contents

1 Abstract

With well over a billion users combined, Google Chrome and Mozilla Firefox web browsers are among the most popular in the world. A contributing factor to that popularity is that users have the ability to install third-party browser extensions, usually to streamline a user's experience with that browser. Unfortunately, black-hat hackers have exploited the fact that millions of these extensions are downloaded every day, and can easily build malicious extensions that can track browser usage, intercept cookie information, and even obtain logon credentials for websites.

Here, I'll examine how extensions are built, with a focus on the permissions and security model of those extensions. I'll focus on the Google Chrome browser, discussing what security features they have added to make their extensions more secure, but demonstrate how easily an extension could be built to steal user information.

2 Introduction

The Google Chrome Web Browser is one of the most popular browsers used today – in October 2013, over 54% of all web-browsing consumers used Chrome to browse the internet world-wide [1]. With the introduction of Chrome OS and an increasing consumer reliance on web browsers, users are increasingly making use of “Browser Extensions” – essentially ‘apps’ for the browser. Considering the popularity of Chrome and its extensions, the security measures Chrome puts in place are becoming increasingly relevant. While Chrome does have security measures in place to help prevent attacks via browser extension, attackers still have the ability to break the security model to get access to personal consumer information.

2.1 Chrome Extension Architecture

A typical extension has several parts, and includes content scripts (usually JavaScript), an extension core, and a native binary. A content script is written in JavaScript and can be injected into a web page when it is loaded. These extensions inject the content script into a tab when the extension is activated on that tab, which thus gives the extension access to DOM objects on the page. The extension core includes one or more background web pages written in HTML and JavaScript and run in separate renderer processes [2]. While the extension core contains most of the privelges, it runs in a sandboxed environment. Therefore, it cannot access network information directly; it must use `XmlHttpRequest` to access external web resources.

2.2 Chrome Extension Security Model

Chrome extensions follow three security principles: privilege separation, least privilege, and strong isolation.

The content script of an extension can directly interact and work with web contents. But by default, it cannot access browser modules, and can only communicate with the extension core via `postMessage`. This core has the most assigned privileges, but is separate from the actual content core. If applicable, the native binary of an extension has the most privileges, as it can run any arbitrary code or access any file. This follows the principle of privilege separation.

Extensions also make use of the *least privilege* principle. By default, Chrome extensions are given access to only a few privileges. Chrome defines a set of other privileges, however, ranging from accessing bookmarks, cookies, history, location, and even other tabs or extensions. These can be accessed using the Chrome API [2, 4]. For the least privileges purpose, extensions are required to have a manifest.json file, which declares a list of permissions an extension needs to access in order to run.

Finally, Chrome makes heavy use of *strong isolation*, by making use of three different layers. On the first level, permissions to access web sites are defined based on origins, as defined in the Same-Origins Policy (SOP) [5]. The second level of isolation is on the multi-process architecture of the browser. This works because each component of an extension runs in a different process. Finally, the third level of isolation deals with how scripts are run. Instead of just running a JavaScript file, JavaScript code is run in a separate JavaScript engine called `isolated world`. Each isolated world has its own copy of the DOM, making it impossible to pass JavaScript pointers around to malicious code.

Unfortunately, even with this rather comprehensive security model, various attacks can still be launched via Chrome extensions.

3 To The Community

This paper is aimed primarily at users of modern browsers, notably Google Chrome, who have installed browser extensions before. Many users who make use of Chrome and similar browsers are unaware of the security vulnerabilities that can be opened by giving a malicious browser extension too many permissions [7]. Hopefully users will gain a better understanding of how these extensions work within the browser environment and will realize the need for caution when installing extensions onto a browser.

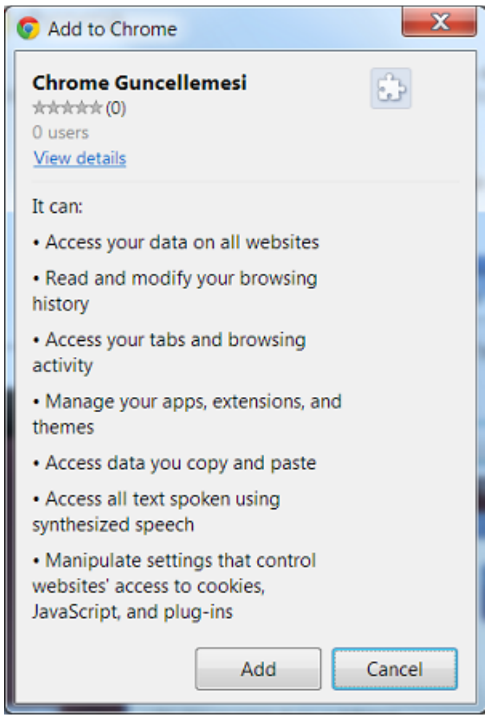
4 Action Items

The following are a selection of security vulnerabilities that a malicious Chrome extension could make use of, given the proper permissions. They include how extensions can extract user information, launch cross-site request forgery (XSRF) attacks, and sniff for passwords. I'll then discuss the measures Google takes to prevent malicious extensions.

4.1 Extracting Browser Information

When a Chrome extension is first run, like an Android application, it requests permission to access various parts of the browser experience. As discussed earlier, this request can range from asking to access your geolocation, the browser cookies, the browser history, and more [4]. Google provides developers the ability to access these aspects of your browser in able to give third-party developers the chance to enhance the experience of browsing the web. Many extensions, such as AdBlock, use the permissions granted to it for good – in this case, to access the DOM and extract all advertisements [8]. However, these permissions can be abused to collect information about a user, attempt to spoof cookies, or otherwise interact maliciously with a user.

In January of last year, one group of Turkish cyber-criminals created an extension that obtained permissions and then used them maliciously [9]. If a user clicked an infected link (which was spread through Facebook), they would be taken to a site that promised a “Chrome update.” This “update” was actually a malicious extension, which requested permission for the following:


Add to Chrome

Chrome Guncellemesi
(0)
0 users
View details

It can:
- Access your data on all websites
- Read and modify your browsing history
- Access your tabs and browsing activity
- Manage your apps, extensions, and themes
- Access data you copy and paste
- Access all text spoken using synthesized speech
- Manipulate settings that control websites' access to cookies, JavaScript, and plug-ins

Using these permissions, the extension attempted to use XSRF attacks to steal personal information, and then, upon logging into Facebook, would auto-post the links to spread itself around.

Other similar apps gain these permissions for the purpose of selling personal user information to advertisers [7]. Once the extension has access to a user's browsing history and cookies, the extension can send all that information to a central server, and the potentially sell to interested parties. The extension also could feasibly upload its own cookies, perhaps for the purpose of tracking users. Once the extension has permission to access those parts of the browser, actually collecting information is a trivial API call [4]. For example, if an extension wanted to search a user's history after gaining permission, the only call needed would be `chrome.history.search("foo")`.

4.2 Cross-site Request Forgery (XSRF)

Through its content script, an extension is given access to a web page's DOM. While a content script cannot make cross-site requests without authorized permissions, a cross-site request forgery attack is still possible. A content script does have access to the origin of the associated web page, so it can make HTTP requests to that web page. Since all requests are regarded to have the same origin, any user credentials, such as cookies, can be included. Since many web sites, such as banks, use cookies as an authentication mechanism, an XSRF attack is possible if the content script abuses the trust between the browser and the server [10].

For example, let's say a malicious extension wanted to make use of an XSRF vulnerability in a bank website to withdraw money from a victims account. When activated, the extension might load a well-designed "image" element that withdraws money from the bank and deposits it in another account (e.g. '`<img src=http://www.thebank.com/transfer?acnt=nick\&amnt=999999999\&recipient=ming>`'). If the bank's server keeps the authentication information in a cookie and that cookie hasn't expired, the attempt by the browser to load that image will result in a transfer from the victim to the attacker in the listed amount, without the user's approval. These sorts of attacks can also be used to automatically post to social media sites like Facebook without a user's permission or knowledge [6].

4.3 Extracting User Input

A content script that is given access to the DOM can also be used to steal logon credentials for other seemingly secure websites. Since sensitive information, like passwords, usernames, credit card numbers, or account numbers are frequently stored in the web browser (whether temporarily or permanently), browsers are major targets of spyware and attempts to retrieve that information.

With a malicious extension, this task may be trivial. Since an extension is given access to the DOM, it can read the content that is loaded and entered by the user. First, the user must give the extension permission to access the site and insert the content script when the page is loaded. Once given access, the content script can read the values of the user name and password elements when the user inputs the information to the web page. If the extension is then given cross-site access (`http://*/*`), the content script can then also send that information to an outside network or server.

4.4 Google Security Patches and Policies

Overall, Google is very good about releasing security patches or changing policy to respond to security threats. Over 2013, they have made many changes to the way Chrome Extensions are installed, released, and updated. Now, users are not permitted to install extensions to the browser that are not downloaded directly from the Google Chrome Store. This gives Google a new layer of control, by forcing all developers to upload their extensions directly to Google, and allows Google to be the ultimate arbiter when it comes to deciding if an extension follows their policies or not. Extensions now undergo a code review process, which allows Google to filter out applications that hint at being malicious. As more threats pop up, researchers try to build scanning software that will find most malicious applications before they are given the opportunity to distribute themselves. Google has also banned ‘silent installs’ from occurring on your browser, meaning that an extension must be granted specific authorization by the user to be installed in the first place. With all these checks, it’s more difficult for a malicious extension to be installed on the browser than it was in the past.

But this does not curb all threats. Extensions still update silently and in the background every time the browser is loaded. What this means is that, potentially, an attacker could build a seemingly legitimate extension that passes Google’s code review, and then, after it has been installed on many browsers, “update” the code by adding in the malicious code [7]. This update would instantly be seen by all users, who would not know an update occurred unless they were to explicitly check. Further, there is no definitive list of what makes an extension malicious and what does not. As a result, it’s impossible to create a fail-safe list of all the different permissions and types of code that could be used maliciously, making it impossible to filter out all malicious extensions during a code review [3].

5 Conclusion

Google Chrome overall is a secure browser, and does make efforts to protect it’s users. But despite those attempts, there are still ways that malicious extensions can steal your information, hijack your web sessions, or even grab authentication credentials. Extensions can be given access to a wide variety of browser information, ranging from cookies to browsing history, and accessing that information is trivial for an extension. Through XSRF attacks, an attacker could use authentication cookies to trick a website into making changes, like automatically posting on social media or even withdrawing money from a bank. Given access to the DOM, an extension could potentially read user-input fields to steal usernames, passwords, credit card numbers, or anything else that a user might input in a field on a page.

The one thing all these different types of attacks have in common is that they depend on being given permission by the user to access those parts of a browser. While Google itself attempts to curb these sorts of malicious extension, the only fail-safe way to not be attacked is for the user to have the knowledge of how extensions work, and what permissions they are granted. It is ultimately up to the user whether to install and grant permissions to a potentially malicious extension, so it is important that a user understands how malicious extensions work, and only installs extensions that he or she trusts. The Google Chrome browser has many extensions that are not malicious, and ultimately it’s up to the user to separate between the two.

6 References

[1] *W3 Browser Statistics,* Accessed Dec. 2013 http://www.w3schools.com/browsers/browsers_stats.asp

[2] *Chrome Extension Developer Overview,* Accessed Dec 2013 http://developer.chrome.com/extensions/overview.html

[3] Liu, Zhang, et al. *Chrome Extensions: Threat Analysis and Countermeasures.* George Mason University, 2011

[4] Osborn, Johansen; *Hacking Google Chrome OS;* WhiteHat Security August 2011 http://www.slideshare.net/kosborn/hacking-google-chromeos-blackhat-whitepaper

[5] *Chrome Extension Developer: Cross-Origin XMLHTTPRequests* http://developer.chrome.com/extensions/xhr.html Accessed Dec 2013

[6] Schrwartz, *Malicious Chrome Extension Poses As Facebook Video,* Information Week, August 30 2013. http://www.informationweek.com/attacks/malicious-chrome-extension-poses-as-facebook-video/d/d-id/1111354?

[7] Seltzer, *Malicious Chrome Extensions on the* Rise. ZDNet. August 27 2013. http://www.zdnet.com/malicious-chrome-extensions-on-the-rise-7000019913/

[8] *About AdBlock Plus* https://adblockplus.org/en/about

[9] Assolini. *Malicious Chrome Extensions: A Cat and Mouse Game*. January 31 2013. http://www.securelist.com/en/blog/208194095/Malicious_Chrome_extensions_a_cat_and_mouse_game

[10] *Auto-Translate Extension*
https://chrome.google.com/extensions/detail/obgoiaeapddkeekbocomnjlc
kbbfapmk