University of Padova

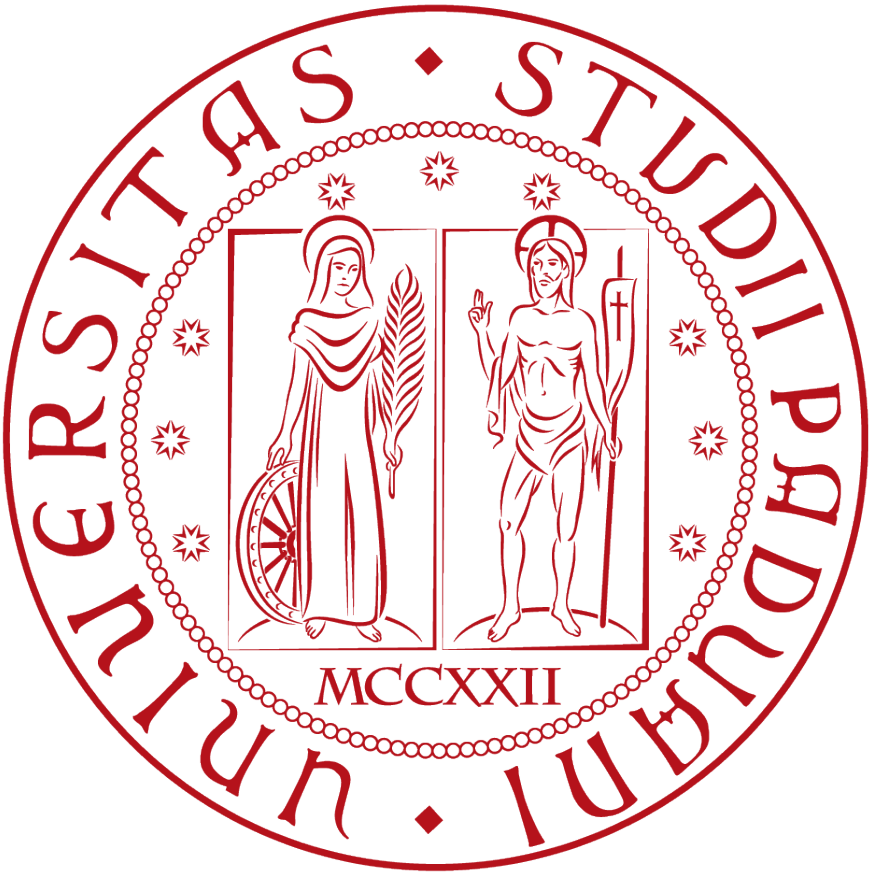
Computer and Network Security

Privacy issues on chrome extensions

Students:
Davide Trevisan
Andrea Multineddu

Registration number:
1070686
1049261

Friday 26 August 2016



Introduction

Nowadays the majority of browsers let users the possibility of customizing it in order to get a new feature o a new style of visualization to better match the taste of the single user. All of this is achievable through the existing extensions for modern browsers. But there is also the opposite situation in which a malicious user tries to get personal information using this extensions.

Abstract

Google Chrome browser is known for it huge amount of extensions that let users do almost everything they want, but there also malicious extensions that aim to steal personal information about the life, bank accounts and every other single piece of information of the users. In this paper we presents 3 extensions: ChromeLogger[2], Stealth Screenshot e Activity Logger;

ChromeLogger is a keylogger and form grabber for Google Chrome that runs as an extension. ChromeLogger works by injecting javascript into all loaded web pages.

Stealth Screenshot make screenshots of the active tab based on a timer and keep them until every opened tab is closed. Users can access with a combination of key to all screenshots taken during the session.

Activity Logger keep trace of all events related to extension and save them in Chrome storage. Users can check if there are installation, uninstallation, activation or deactivation of Chrome extension due to malicious code running in background.

How Chrome Extension works

Background

where to find an extension

The Chrome Web Store is the official way for users to find and install extensions, where the developers can publish it. They can also push out updates without any action by the end-user. In addition to the Chrome Web Store, extensions can also be installed manually by a user or an external program. Extension that are not downloaded through the official channel are flagged by Chrome when it starts and the user is asked to disable them. However, the user can whitelist the extension through the console.

Manually install an extension

Once you have all the source code of an extension in a folder, go to CHROME://EXTENSIONS (or find it in the menu in more tools > extensions), then activate the developer mode on the top right of the page. To install the extension, you just have to indicate where is this folder on your computer clicking on load unpacked extension. For .CRX packaged extension, you have to drag them on the CHROME://EXTENSIONS tab and then confirm that you want to install it (it's not directly possible because of the stronger security policy of chrome)

Permission

Chrome requires extensions to list the permissions needed to access the different parts of the extension API. The complete list of the permissions and their interaction can be found on google developer page[1].
Extensions must also specify a list of content scripts to indicate JavaScript files that will run inside of the web page, because this is a powerful feature that allows an extension to be indistinguishable to the web page behavior. Besides the content scripts that allow an extension to interact with a given page, Chrome also allows extensions to run scripts in a "background page" that often contain the logic and state an extension needs for the entirety of the browser session and do not have any visibility to the user.
For example this is the manifest of the stealth screenshot extension:

```
{
"name": "Stealth Screenshot Extension",
"version": "0.1",
"description": "This is a prototype
```

```
 of a background screenshot extension",
"background": {
"persistent": true,
"scripts": ["background.js"]
},
"commands": {
"toggle": {
"suggested_key": {
"windows": "Ctrl+Shift+Y",
"mac": "Command+Shift+Y",
"chromeos": "Ctrl+Shift+U",
"linux": "Ctrl+Shift+J"
},
"description": "My description"
}
},
"browser_action": {
"default_icon": "camera.png",
"default_title": "Take a screen shot!"
},
"omnibox": { "keyword" : "show" },
"permissions": [
"storage",
"tabs",
"<all_urls>",
"unlimitedStorage",
"activeTab"
],
"manifest_version": 2
}
```

as you can see, an extension has to declare in addition to all the permissions and all the interaction it has with the user such as an input in the omnibox or the click on the icon.

ChromeLogger

Scope of the extension

The extension is a form grabber for Google Chrome running natively in it without the need to install additional external software. It works on every loaded web pages acting on every form in the pages. The extension manage to let the user checks for every submitted form all the keystroke done and some additional data about the time of submission. This is done by an additional page where we can turn on some additional option for the form grabber and visualize all the data collected by the extension. The extension makes use of the following permissions in the manifest file:

- <all_urls>
- tabs
- storage
- unlimitedStorage

How the extension work

When a web page is completely loaded ChromeLogger injects javascript in it. The payload records keypresses using event listeners and saves them to Chrome's storage. Unlike other browser keyloggers, ChromeLogger runs natively in Chrome (on all OS's) without the need to install additional software. The form grabber works in a similar way. Javascript is injected and event listeners are added for all forms. When a form is submitted, its data is saved to ChromeLogger's storage. This allows form data transferred over SSL to be saved in plaintext. ChromeLogger's payload is written in pure JS and the log viewer is built using AngularJS. When a form is submitted, its data is saved to ChromeLogger's storage with the add of URL and time data about the submission of the form. The user has also access to an option page where he can turn on or off some optional features and check all the logs inside a table where can filtering and ordering them by the data and URL.

Injection

The injection is simple but effective:

```
chrome.tabs.onUpdated.addListener(
   function(tabId, changeInfo) {
if (changeInfo.status === 'complete') {
chrome.tabs.executeScript(tabId, {
allFrames: true,
file: 'src/inject/payload.js'
});
}
});
```

this allows the extension to add to the standard javascript of the page a malign javascript that allows the extension to receive and save all the form submitted by user and also the message submitted through social apps (for example whatsapp web)

Performance

There aren't expensive operations or recursive function calls that requires a lot of computation so we can state that the impact on the CPU is negligible.

Activity Logger Extension

Scope of the extension

The extension aims to keep traces about the activities of other extensions and let the user know about them through a dedicate page in which can check for every extension that are installed on the Chrome browser. This logger keep traces of logs even if some extensions are uninstalled.
The extension makes use of the following permission the manifest file:

- management
- storage
- unlimitedStorage

How the extension work

The extension at the browser start check the list of installed extension and updates the logs list with the one missing from it. After this first check and update we create listener for the events related to extensions which are:

- chrome.management.onInstalled
- chrome.management.onUninstalled
- chrome.management.onEnabled
- chrome.management.onDisabled

Every time the user or some malicious code install, uninstall, enable or disable an extension a log of the triggered event is saved in the specific event log for the specific extension in the chrome storage. The computation time is defined by the computation time of the Chrome search in his storage.

Performance

There aren't expensive operations or recursive function calls that requires a lot of computation so we can state that the impact on the CPU is negligible.

Limitations

The main limitation of this extension can be reconducted to the limitation imposed by or to the absence of Chrome APIs. At the moment we can just check the events described early in the paper, every other event defined in Chrome APIs and the presence or the absence of listener for a specific event. A better control on browser activity could be achieved by adding APIs, for example that return the name of extension from which is running codes in response to a specific event.

Future work

The final aims of this extension are to log every single activity of installed extensions and jasvacript scripts inside web pages in response to every event happening inside Chrome itself (like a modification inside the chrome storage area) or the web pages visited by the user (the pressure of a button) and let the user check these logs in order to let users check some strange and unwanted activities caused by some malicious code. At this time, for what we have find out from our research on chrome API documentation we can just check if there is some code waiting a specific events that trigger it. But this is also a double-edged weapon that can led to problems for privacy maintenance because let every user, capable of writing chrome extension the possibility to check every activity, but from the other hands let malicious users the possibility on customizable browsers to get knowledge about users routines, personal data and ad-hoc javascript code injection.

Stealth Screenshot Extension

The source code of the extension we developed is downloadable here

Scope of the extension

the scope of the stealth screenshot extension is to create a small prototype of an extension that catch screenshot in a regular interval with the minimum possible user interaction and obviously giving it back.

Activation and screenshot collection

The extension uses the relatively new tab APIs given by chrome too take screenshot. The extension makes use of the following APIs:

- storage
- tabs
- <all_urls>
- unlimited storage
- active tab

This is the core part of the extension:

```
var imglist=[];
.........................
// Listen for a click on the camera icon.
 On that click, take a screenshot.
chrome.browserAction.onClicked.
   addListener(function() {
chrome.windows.create({type: "popup",
   state:"minimized"})
interval=0;
while(interval<30000000)//print x times,
   where x is the number of milliseconds/
   milliseconds for screenshot
{
setTimeout(function(){Loop()},interval);
interval=interval+60000;
}
});

function Loop (){
chrome.tabs.captureVisibleTab(function(
   screenshotUrl) {
imglist[index]=screenshotUrl;
chrome.storage.sync.set({"data" : new
   Date().toJSON()}, function() {
index++;
if (chrome.runtime.error) {
console.log("Runtime error.");
}
});
}
);};
.........................
```

The extension works in a simple way: it triggers on a browser event, in this case the click on the icon of the extension, but there are other possibilities: we tested that there are no simple ways to take screenshot without no interaction, because the security policy of the API doesn't allow to take a screenshot without some kind of events, presuming that the user should be aware of what is happening; not triggering any event returns only null. Once triggered, the extension schedule all the future screenshot through the Javascript setTimeout method: this allows to take screenshot for hours, with only the need for a click to start. The screenshot are stored in an array visible to all the extension methods.

Screenshot retrieval

The screenshot can be collected in two ways in the extension we developed: through a combination of key for retrieving all screenshots (we programmed it on Ctrl+Shift+Y) or writing "show" in the omnibox, pressing "tab" and writing in the omnibox the number of screenshot to show (starting from the last one).

Performance

We tested that the extension is able to take screenshot with a interval of 1 minute for more than 2 hours without losing any screenshot. The application occupy less of 50MB of RAM for an hour of screenshots taken every minute. The impact on the CPU is negligible. Screenshot retrieval through the key combo however can crash chrome, although it never happened in our PC because of the great performance (we had an Intel i5 6400, with 8GB of DDR4 memory and SSD), but it got frozen for some seconds.

Limitations

The major limitation of this extension is that the screenshot only lives until the extension is active: this implies that closing all the windows of chrome completely deletes the screenshot taken. This is a

consequence of how the API and the chrome sandbox works. At the moment, we found no way to get around it, but we are pretty much confident that is possible to save those screenshots, but due to our lack of knowledge we are not able at the moment to demonstrate it.

Future work

Possible future works will focus on find a way to save those screenshot. The application should also be rewritten without the tab API, to avoid of the limitation of it. Javascript inject the code for the screenshot or use it for simulate the right events should do the work, but the time and the train needed to do it made impossible for us to test it for this paper.

Conclusions

Rank	Top 10 types of permissions	# ext.
1	tabs	16,787
2	notifications	12,011
3	unlimitedStorage	9,424
4	storage	5,725
5	contextMenus	4,774
6	cookies	2,872
7	webRequest	2,849
8	webRequestBlocking	2,102
9	webNavigation	1,623
10	management	1,533

Figure 1: The top 10 permissions found in the manifest files for all extensions HULK ran. Extensions can include more than one permission.

Chrome should improve their task manager, to give the users more knowledge about what is happening in their PC, because is possible to create a timed or a smart bomb with a simple WHILE(TRUE) cycle, and it can crash chrome, because the sandbox sometimes can't handle it. All the solution we have presented in this paper makes use of the most used chrome permission, as already stated in the HULK paper[3] As shown in the screenshot extension, taking screenshot without user consensus should not be allowed to the API (just a popup should be enough). The ChromeLogger and Stealth Screenshot show the presence of issues reguarding the maintenance of user's privacy, and the Activity Logger can be used as a little warden against unwanted modification of extensions list. We still find out ways to bring Google Chrome to crash in a really simple way using the basics keyword of Javascript.

References

[1] *Google developer page: permissions:* https://developer.chrome.com/extensions/declare_permissions

[2] Eric Zhang,
ChromeLogger,
A keylogger and form grabber for Google Chrome that runs as an extension.
ChromeLogger
referring site

[3] *Hulk: Eliciting Malicious Behavior in Browser Extensions*
Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel Giovanni Vigna, Vern Paxson, UC Santa Barbara, UC Berkeley, UC San Diego and International Computer Science Institute
23rd USENIX Security Symposium.
Paper