Leaks in Web Browser Privacy Modes

Matt Burrough
Burrogh2@illinois.edu
CS 461
November 2012

Abstract

This paper provides an examination of five possible circumvention techniques for browser privacy modes. These include residual files, malicious extensions, browser crashes, extension crashes, and non-terminated browsers. The success of these techniques is evaluated across Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Opera, and Apple Safari running on a Windows 7 system.

Disclosure: I have been employed by Microsoft full time for the past 56 months. That said, I do not work on the Internet Explorer team and have no direct contact with them. I have not used any internal Microsoft resources in the preparation of this report, and have not intentionally favored Internet Explorer or omitted any findings. Additionally, I have not reverse-engineered any of the products described in this report. This paper's contents are the sole work and ownership of the author, and do not represent any opinions of his employer. Microsoft has not reviewed or approved its content in any way.

Introduction

Browser "privacy" modes have been included in the last several releases of every major web browser on the market. While this functionality does nothing to protect the identity of a user to the web sites visited, nor to hide browsing habits from network providers (Soghoian, 2010), users have come to trust these modes to keep confidential their web history from others with access to their computers. Is it possible that this trust is misplaced? In spite of the prevalence of privacy modes, little research has been done to validate if ones web history truly is protected when using them.

Two academic papers on the topic are "Forensic Analysis of Private Browsing Mode in Popular Browsers" (Mahendrakar, Irving, & Patel, 2011) and "An Analysis of Private Browsing Modes in Modern Browsers" (Aggarwal, Bursztein, Jackson, & Boneh, 2010). The former mainly examines system memory using forensic analysis tools during and immediately after private browsing. The later takes more of a survey approach, examining many facets of browser privacy.

This paper aims to determine if a 3rd-party could determine the browser history of a user utilizing private browsing by examining five areas where browser history may be exposed in spite of running in a privacy mode. This is not meant to be a fully exhaustive list of tests, but rather to highlight some potential areas of concern and validate if that concern is warranted. The tests are:

1. Confirm that files are not left behind by the browser itself when private mode is deactivated. This will be accomplished by reviewing process monitor logs to observe which files are modified during private browsing and validating that they have been scrubbed or removed when the session is ended.
2. Creating a collection of browser extensions that covertly leak a user's browsing history to a 3rd party. Browsers will be tested to see if they allow the extension to load in private mode, and if so, if they at least provide some warning or make the loading of the extension difficult for the user.
3. Determine what information is saved and reported to browser vendors when a browser crashes in privacy mode. In order to facilitate a crash, a debugger will be used to induce an unhandled exception.
4. Validate if a crashing browser extension results in information disclosure to the browser vendor.
5. Investigate what information can be obtained from a browser in privacy mode if a user closes any open tabs but does not exit the process.

Each of these are either novel when compared to areas examined in existing literature, or are approached in a different way than in the previous papers. For example, where other papers look at popular extensions to see if they disclose any history from private sessions,

the papers do not attempt to create a malicious extension. Additionally, this paper seeks to include Opera where possible, which was omitted from other papers.

Test Machine Configuration

To analyze each browser, a virtual machine was created in VirtualBox 4.2.2 running on a Windows 7 x64 SP1 host on an Intel i7-950 with 12 GB of RAM and VT-x enabled. The virtual machine was built with 2 GB of allocated memory, two virtual cores, a 50 GB VHD disk file, and Windows 7 Home Basic x86 SP1. Windows 7 was chosen as it is both widely used and supports a wide variety of browsers. The x86 architecture was chosen to avoid the complication of WOW64 when using debugging tools.

The VM was configured with each of the following:

- All updates offered on Windows Update as of 10/19/2012 except Bing Desktop and Silverlight 5
- VirtualBox guest additions 4.2.2
- Process Monitor v3.03
- Process Explorer v15.23
- ProcDump v5.0
- Microsoft Network Monitor v3.4
- Microsoft Security Essentials
- NotMyFault
- The Debugging Tools for Windows package v6.2.9200.16384
- Windows 8 SDK (as much as required to install the debugging tools package)
- OS Set for complete memory dumps, page file set to 2.5 GB min/4 GB max
- 1280x1024 resolution (single screen)
- The system environment variable "_NT_SYMBOL_PATH" set to "srv*c:\symbols*http://msdl.microsoft.com/download/symbols;SRV*c:\symbols*http://symbols.mozilla.org/firefox;srv*c:\symbols*http://chromium-browser-symsrv.commondatastorage.googleapis.com"

Otherwise, the VM was installed with all default options. The VM NIC was bridged to the author's home network and the VM's network policy was set to "Home". One account was created during Windows installation named "testuser". It had administrative rights, though UAC was enabled so the account typically ran with least privilege.

With these programs and settings in place, the VM was snapshotted so it could be reverted to this state between each test to ensure a clean testing environment. From here, five sub-snapshots were created, one for each browser. (Note that all sections are written with browsers in alphabetical order.)

Chrome
Google Chrome 22.0.1229.94m was installed in this snapshot and set as the default browser and with usage and crash reporting enabled. A reboot was performed after the installation but before taking the snapshot.

Firefox
This snapshot had Firefox 16.0.1 installed using the standard install option and set as the default browser. The homepage was set to a blank page, and Mozilla usage and error reporting was enabled (opted-in) during installation. Hardware acceleration was set to disabled to avoid any issues with the VirtualBox vGPU. A reboot was performed after the installation but before taking the snapshot.

Internet Explorer
This snapshot was upgraded to Internet Explorer (IE) 9. IE was configured to use recommended security and compatibility settings. The home page was set to about:blank. Software rendering was set to enabled to avoid any issues with the VirtualBox vGPU. A reboot was performed before taking the snapshot.

Opera
In this snapshot, Opera 12.02 was installed as the default browser. The home and startup pages were set to about:blank. A reboot was performed after the installation but before taking the snapshot.

Safari
Finally, Safari 5.1.17 was installed with all installation options enabled. The start page was set to an empty page. A reboot was performed after the installation but before taking the snapshot.

Test 1: Monitoring for residual files

The most obvious violation of the concept of private browsing is traces of websites visited while in private mode left behind on the user's disk. For this test, each browser was started in its VM snapshot, and put into private mode. Process Monitor was then launched and filtered to just processes related to that browser. The first tab in private browsing was navigated to http://www.burrough.org while a second tab was created and navigated to http://en.wikipedia.org/wiki/Cockapoo. (The dog breed Cockapoo was chosen as the article is relatively short but contains several images, and the author is partial to that breed.) Once both pages finished loading, the browser was exited and process monitor stopped. The process monitor log was then reviewed for file and registry

entries that were created during the session, and those items were then searched for in the registry and file system.

Chrome

Chrome primarily used one directory and two registry keys while operating in incognito mode:

HKEY_LOCAL_MACHINE\SOFTWARE\Google\Update\ClientState
HKEY_CURRENT_USER\Software\Google\Update\ClientState
C:\Users\testuser\AppData\Local\Google\Chrome\User Data\Default\

Reviewing the registry keys, these appear to mostly contain information about Chrome itself like installation path, version number, language, and update check information, none of which appears to be sensitive.

Within the Default folder, there were nine files that were modified during incognito browsing, as well as one subfolder, Cache, which contained one modified file. The files each came in pairs, with one file being a journal version for the other, likely for consistency. The files were History, Web Data, Cookies, and Preferences, plus a file data_01 in the cache directory. Using tools from NirSoft (Sofer, 2012), each file could be decoded.

ChromeHistoryView from NirSoft revealed that no entries were left in the web history from the incognito session.

The Web Data files were in binary form, but contained a SQLite header, so SQLite Database Browser from http://sourceforge.net/projects/sqlitebrowser/ was used to examine the files. The database contained a series of tables for form auto-complete information. The only one populated was the list of search engines to use – Bing, Google, or Yahoo. Otherwise, all of the tables were blank.

Nirsoft did not make a Cookies viewer for Chrome, but the file was again a SQLite database, so again SQLite browser was used to examine the file. It appears that all of the cookies were for a previous session and none were from visiting Wikipedia. It was confirmed that, under normal browsing mode, Wikipedia would place a cookie on the system.

The preferences file was not binary, and was human readable. It appeared to contain the user's browser preference information, and did not contain anything private.

Using NirSoft's ChromeCacheView tool, the files in the cache subdirectory were examined. Although one file had a modified timestamp reflecting the time of private browsing, no entries in the cache were from that session.

Firefox

Process monitor revealed a number of files that were modified at the time of private browsing. There were no modified registry keys/values. All of the modified files were located in two folders in the user's profile:
C:\Users\testuser\AppData\Local\Mozilla\Firefox\Profiles\(Mozilla Profile ID)\
C:\Users\testuser\AppData\Roaming\Mozilla\Firefox\Profiles\(Mozilla Profile ID)\

Reviewing the files, many were specific to the Firefox application, such as last window size, timestamp of last successful exit, and the like. These pose no risk of information disclosure, other than the fact that the browser was run at this specific time.

One subfolder of the user's profile that was modified was the cache directory. This initially was cause for some concern, as browser caches should not contain data resulting from private browsing. Mozilla keeps its cache in a non-human viewable format, so MozillaCacheView from NirSoft was used to view the contents of the cache. This revealed that the cache only contained the initial Firefox start page that was opened before the browser could be put into private browsing mode. No other content was contained in the cache.

Another file that was updated was cookies.sqlite. This file contains cookie information for the user. NirSoft's MozillaCookiesView was used to open the file. It contained three cookies, one from Mozilla.org, one from google.com, and one from webtrendslive.com. It appears that all three were placed by the initial start page, and not by subsequent private browsing.

The only other file in the profile that was updated was urlclassifier3.sqlite. It is a binary file so its contents could not be directly viewed. According to MozillaZine, this file is used to contain a list of known malware and phishing sites and is updated from a feed from Google (Barnabe, 2007). Thus, it is unlikely that this file contains any personal information, though this was not confirmed given the inability to read the file.

Internet Explorer

There was considerable registry access during the use of InPrivate mode in Internet Explorer. While many of the other browsers seem to rely largely on configuration and preference files, many of these settings are found in the registry for IE, mainly under HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\. Examination of this key and searching the registry as a whole showed that none of the private browsing history was stored in the registry.

In terms of file system write access, it was mostly concentrated in C:\Users\testuser\AppData\Local\Microsoft\Windows\Temporary Internet Files\Low and

C:\Users\testuser\AppData\Local\Microsoft\Windows\History. There was also some access to C:\Users\testuser\AppData\Roaming\Microsoft\Windows\IEDownloadHistory and C:\Users\testuser\AppData\Roaming\Microsoft\Windows\Cookies\Low. These are the folders responsible for IE's cache, history, download history, and cookies respectively. One benefit of IE is that it stores all of these files as actual individual files, instead of in different databases. This made visually inspecting the folders to confirm that no traces of any of the files from the InPrivate session were still present possible.

IE also uses index.dat files in these locations to catalog the files present. Most of these files showed no evidence of any of the files or URLs from InPrivate browsing, however the index.dat located in C:\Users\testuser\AppData\Local\Microsoft\Windows\Temporary Internet Files\Low\Content.IE5\ did contain the burrough.org URL, though not Wikipedia's (Figure 1). Once this entry was discovered, an Index.dat viewer (Gould, 2008) was installed to gain more insight into the record. This utility revealed that the entry was a REDR record (Figure 2), which according to (Kornblum & Metz, 2007), is a redirected URL. This was confirmed through further testing, which revealed that entering the shorter "burrough.org" redirected the browser to "http://www.burrough.org/pages/index.aspx." Other redirection pages like fwlink pages on go.microsoft.com (used to create permalinks/URL shortening in Microsoft documentation) also appear in this index.dat as REDR records.

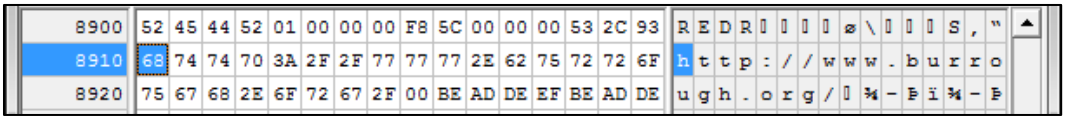


Figure 1: IE Cache Index.dat

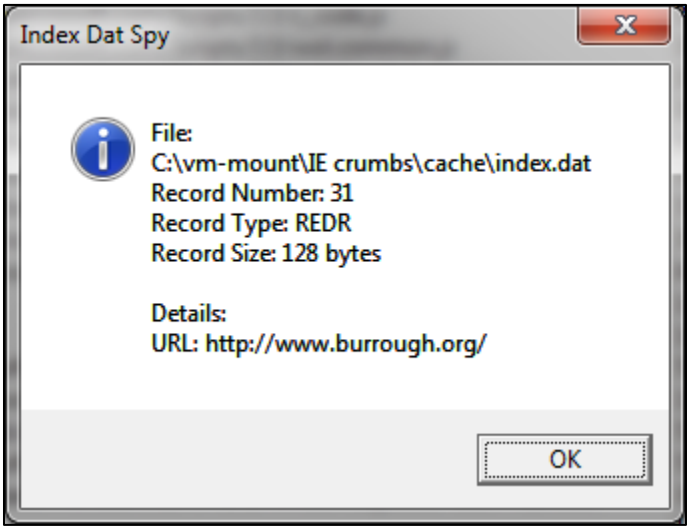


Figure 2: Index Dat Spy Entry

Opera

Opera seems to store all of its cookie, history, and cache information, as well as browser settings in two directories in the user's profile:
C:\Users\testuser\AppData\Roaming\Opera\Opera
C:\Users\testuser\AppData\Local\Opera\Opera

The majority of the files in these directories are either DAT files that are largely ASCII, INIs, or XML. Reviewing every file with a modified timestamp from the time of the private session, none contained any sensitive information. NirSoft's OperaCacheView showed the same. Opera did not appear to store anything of interest in the registry.

Safari

Reviewing entries for the Safari and WebKit2WebProcess processes, once accesses for things like font lookups, default web browser settings, and DLL loads were excluded, Safari actually made relatively few calls to the file system and essentially none to the registry. Those that it did make went to the same two folders seen for other browsers:
C:\Users\testuser\AppData\Local\Apple Computer\Safari
C:\Users\testuser\AppData\Roaming\Apple Computer\Safari

Most of the files therein are SQLite databases, with a few other binary files and some XML. Reviewing each, none appeared to contain any information from the private browsing session. NirSoft's SafariCacheView further confirmed this.

Test 2: Spying Browser Extension

In this test, a custom browser extension was created for each browser tested. The extension generally tried to monitor for page navigation and provide this information to a third party, without notice to the user.

Chrome

Chrome offers a powerful extension system based on JavaScript. The WebNavigation sample code (West, 2012) from Chrome's sample area was used as a base for this extension. The sample code was first modified to remove the UI portion of the code that is exposed to the user – this code originally provided a history-type view to the user in a pop-up menu, but that would hinder the covert nature of add-in.

Next, code was added to post the URL of any completed page navigation to pastebin.com. This site was chosen as it provides an API that allows anonymous submissions via HTTP POST messages, which can later be viewed. This would not be an ideal site for

implementing a history-recording add-on in practice, as it limits the number of posts submitted per day from a given user/computer. However, it would trivial for an attacker to set up a simple web server that accepted POSTs and stored them in a simple database.

The extension was packaged in to a Chrome Extension (.crx) file. When loading the extension, Chrome presented a dialog box alerting the user that the extension would have permission to their browsing activity and to pastebin.com (Figure 3).

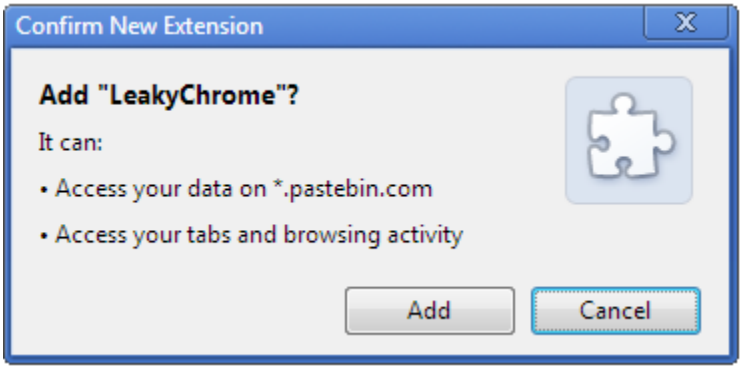


Figure 3: Chrome Extension Permission Dialog

Once the add-in was loaded, it still will not run in Chrome's "incognito" mode by default. For it to do so, the user must check the option in Chrome's Extension list (Figure 4). Upon selecting it, Chrome displays a warning message (Figure 5).

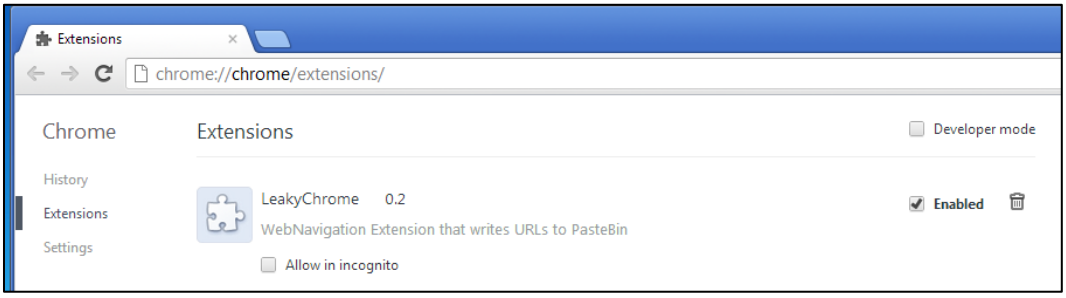


Figure 4: Chrome extension list

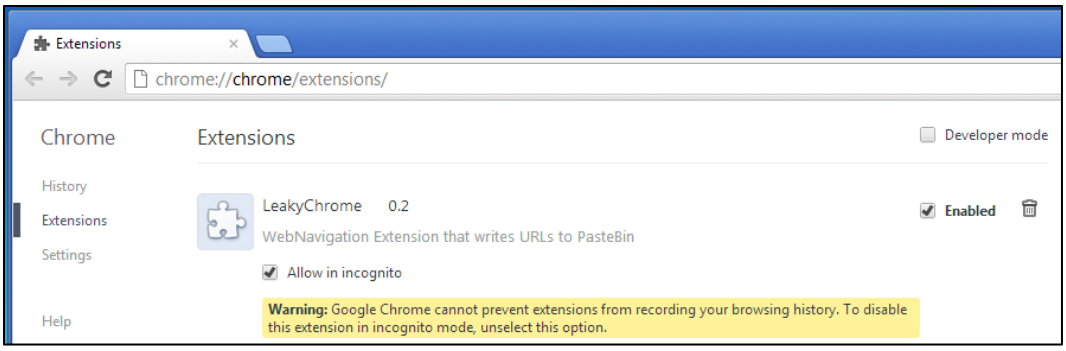


Figure 5: Chrome Extension Warning

After the plugin was installed and checked to allow it to run in incognito mode, it functioned as expected. There were no further warnings that it was submitting URL history to PasteBin, and the URLs appeared on the PasteBin site. Further launches of Chrome in incognito mode provided no warning that any extensions were enabled, though the start page of chrome does mention that loaded extensions could violate their privacy:

> **"You've gone incognito**. Pages you view in this window won't appear in your browser history or search history, and they won't leave other traces, like cookies, on your computer after you close **all** open incognito windows. Any files you download or bookmarks you create will be preserved, however.
>
> **Going incognito doesn't affect the behavior of other people, servers, or software. Be wary of:**
>
> - Websites that collect or share information about you
> - Internet service providers or employers that track the pages you visit
> - Malicious software that tracks your keystrokes in exchange for free smileys
> - Surveillance by secret agents
> - People standing behind you
>
> Learn more about incognito browsing.
> Because Google Chrome does not control how extensions handle your personal data, all extensions have been disabled for incognito windows. You can reenable them individually in the extensions manager."

This message only loads if the user has not specified their own home page, and the warning about extensions at the end of the message appears regardless if the user has any extensions enabled in incognito mode.

Firefox

Firefox has a very strong extension API that, like Chrome, allows developers to create extensions in JavaScript. Using the same method as in Chrome, an extension was written

that reports URLs to pastebin.com upon page load. The code between the two browsers was similar, though the page change notification handling was a bit different. In Chrome, the webNavigation API allows access to page change notifications, while Firefox allows onPageLoad DOM event listeners to be registered. Once fired, the code to perform the POST was identical aside from debug prints for testing. Two resources that were invaluable for preparing a Firefox plugin were (Ben, 2005) and (Jung, 2007).

One interesting aspect about Firefox is that it is very trusting of the extensions that users load. When dropping the .xpi extension file onto Firefox, it presented a warning that the extension was not signed, but allowed the installation to proceed (Figure 6). The UI did not give any indication about what types of access or operations the plugin might use.

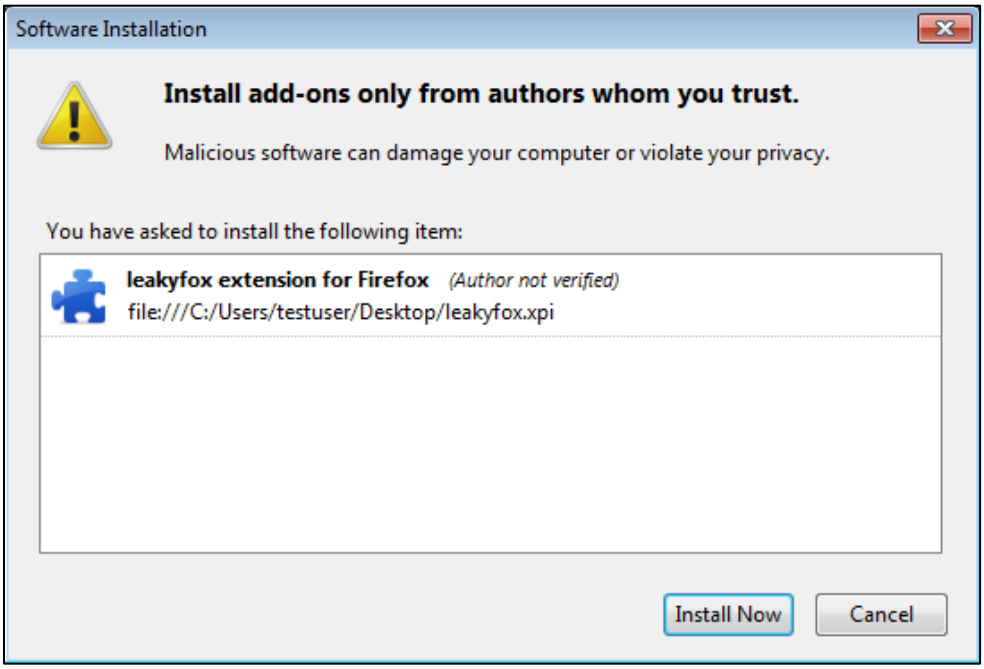


Figure 6: Firefox Extension Installation

After the plugin was installed, Firefox made no mention about the plugin or its activity when switched into private browsing mode. In fact, even the private browsing warning page did not mention extensions (Figure 7). The linked "Learn More" page (http://support.mozilla.org/en-US/kb/private-browsing-browse-web-without-saving-info?redirectlocale=en-US&as=u&redirectslug=Private+Browsing) also failed to mention extensions. The only documentation stating that extensions remained enabled in private browsing mode was from a moderator on the Mozilla forums (cor-el, 2012).

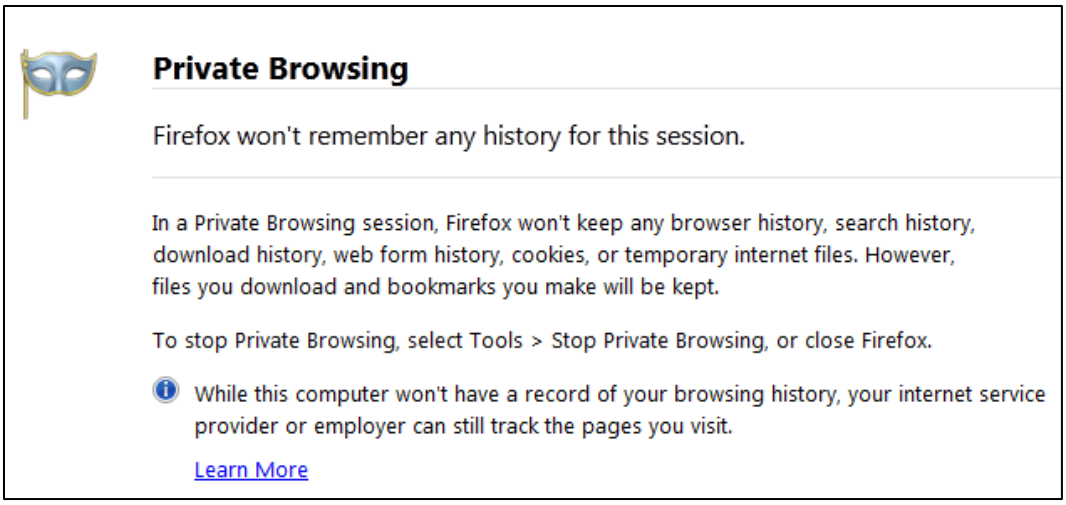


Figure 7: Firefox Private Browsing Start Page

Internet Explorer

Internet Explorer's extension interface is quite different from Chrome's. It relies on C/C++ DLLs to be loaded which interact with IE's COM interface. Recently, several examples of .NET (managed) extensions have appeared online. These samples leverage .NET's platform invocation (p/invoke) feature. A particularly good sample and explanation of interacting with IE from a .NET library can be found in (Jones, 2010). Using Jones' example as a starting point, the message box call was removed and additional code was added to create a WebClient object that submits the URL of the current page to PasteBin's POST interface as the Chrome extension did. The extension submits the URL as soon as a page finishes loading.

A major difference from Chrome's implementation is that Internet Explorer cannot detect the permissions/capabilities of extensions because they are native binaries. As such, the prompt to enable an Internet Explorer extension simply asks the user whether to enable it (Figure 8).

Figure 8: Internet Explorer Extension Installation

Upon loading the extension and restarting Internet Explorer, the extension began reporting the URLs of loaded pages in normal IE operation. However, after switching into IE's "InPrivate" mode, no pages were reported. Further investigation revealed that, by default, IE does not load any extensions while in InPrivate mode. This behavior can be altered by changing a setting in Internet Options (Figure 9). (IE does not allow individual

extensions to be opted-on/off in its privacy mode as Chrome does.) Once changed, the extension worked as expected.

Much like Chrome, the default start page of IE when in InPrivate mode is a static message that does not indicate if any extensions are loaded:

"*InPrivate Browsing* helps prevent Internet Explorer from storing data about your browsing session. This includes cookies, temporary Internet files, history, and other data. Toolbars and extensions are disabled by default. See Help for more information.

To turn off InPrivate Browsing, close this browser window."

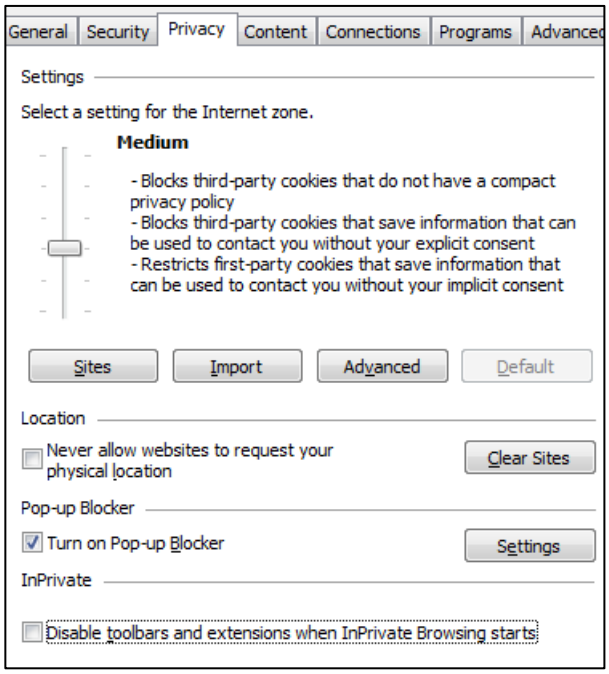


Figure 9: Internet Explorer Privacy Settings

Opera

After hours of reviewing Opera extension documentation, sample extensions, and testing multiple iterations of code, the author was unable to create an Opera Extension ("Widget") that would reliably fire when a page completed loading. Even using Opera-provided sample extensions that were supposed to function on page load without any changes, the extensions did not fire. With Opera's extension developer community seeming to be smaller than that of other browsers, and its less advanced plug-in infrastructure, an attempt to get a prototype working was deemed futile.

In testing extension code, it was noted that installing any add-in that accesses sensitive information triggers Opera to include two check boxes on the Extension installation window (Figure 10). By default, extensions that access any private information by default do not load in Opera's private tabs. A user would need to opt-in explicitly to enable the extension.

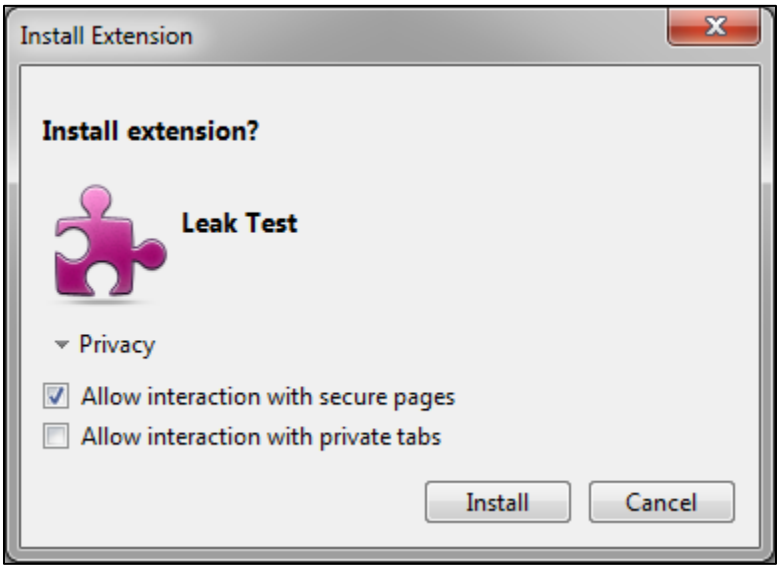


Figure 10: Opera Extension Installation Prompt

Safari

This test was not performed on Safari because Apple requires developers to sign an explicit agreement that states that they will not produce extensions that intentionally violate its users' privacy. The other browsers tested in this section did not have a formal EULA and registration process required to develop an extension.

Safari's extension platform appears to be JavaScript-based like Chrome, so it is conceivable that a similar attack here would have similar success as in Chrome. Additionally, in testing publically available extensions in Safari, they continue to load in Private mode without any warning or indication of possible privacy violations. The extensions load by default in Private mode, without any reconfiguration or settings change required.

Test 3: Crashing Browser

This test involves simulating a code defect in a browser by attaching a debugger to the browser process when in private mode, breaking in, replacing an instruction with an unhandled illegal operation, and detaching the debugger. Upon detach, the process will resume and crash. Any resulting crash dump or error reporting logs will be examined to see if information about the browsing session can be determined.

For each test, the browser was put into private mode and two tabs were opened – one to www.burrough.org and one to wikipedia.org/wiki/cockapoo. Windbg.exe was then attached to the main browser process, and an instruction was changed to cause a crash. Process Monitor was used to look for any dump files or other crash info written to the file system.

Chrome

For Chrome, with two tabs open in incognito mode, there were three Chrome processes (one for each tab, plus one parent process), as well as a GoogleChrashHandler.exe instance. The debugger was attached to the parent Chrome process. After breaking in, the debugger was used to unassembled the topmost Chrome function on the main thread from the point where user32.dll would return to it. About 10 instructions in, the code contained a call instruction. Using the debugger's memory editing command (eb), the command was changed to call a null pointer (0000000). The debugger was then detached gracefully so the process would resume, but not be running under the windbg debugger. This sequence of commands is illustrated in Figure 11.

```
Microsoft (R) Windows Debugger Version 6.2.9200.16384 X86
Copyright (c) Microsoft Corporation. All rights reserved.
*** wait with pending attach
Symbol search path is:
srv*c:\symbols*http://msdl.microsoft.com/download/symbols;SRV*c:\symbol
s*http://symbols.mozilla.org/firefox;srv*c:\symbols*http://chromium-
browser-symsrv.commondatastorage.googleapis.com
Executable search path is:
ModLoad: 00c20000 00d57000   C:\Program
Files\Google\Chrome\Application\chrome.exe
...
 (82c.bc0): Break instruction exception - code 80000003 (first chance)
eax=7ffa0000 ebx=00000000 ecx=00000000 edx=779bf17d esi=00000000
edi=00000000
eip=7795410c esp=072cf954 ebp=072cf980 iopl=0   nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
```

Figure 11: Chrome Crash Setup

```
ntdll!DbgBreakPoint:
7795410c cc              int     3
0:030> ~0s
eax=00000000 ebx=0031f8b4 ecx=00000000 edx=003d8600 esi=00000001
edi=0031f8d4
eip=77967094 esp=0031f864 ebp=0031f900 iopl=0         nv up ei pl zr na
pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
ntdll!KiFastSystemCallRet:
77967094 c3              ret
0:000> k
ChildEBP RetAddr
0031f860 77966a04 ntdll!KiFastSystemCallRet
0031f864 75bc6a8e ntdll!NtWaitForMultipleObjects+0xc
0031f900 77aabd66 KERNELBASE!WaitForMultipleObjectsEx+0x100
0031f948 75f862f9 kernel32!WaitForMultipleObjectsExImplementation+0xe0
0031f99c 6984b9c2 USER32!RealMsgWaitForMultipleObjectsEx+0x13c
0031f9d8 6984b5f8
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x24
0031fa08 696e27c7 chrome_696b0000!base::MessagePumpForUI::DoRunLoop+0xb8
…
0031fd70 00000000 ntdll!_RtlUserThreadStart+0x1b
0:000> * //Let's look at the return point into the first chrome
function...
0:000> u 6984b9c2
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x24:
6984b9c2 85c0            test    eax,eax
6984b9c4 7546            jne
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x6e
6984b9c6 6a06            push    6
6984b9c8 8945e4          mov     dword ptr [ebp-1Ch],eax
6984b9cb 8945e8          mov     dword ptr [ebp-18h],eax
6984b9ce 8945ec          mov     dword ptr [ebp-14h],eax
6984b9d1 8945f0          mov     dword ptr [ebp-10h],eax
6984b9d4 8945f4          mov     dword ptr [ebp-0Ch],eax
0:000> u
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x39:
6984b9d7 8945f8          mov     dword ptr [ebp-8],eax
6984b9da 8945fc          mov     dword ptr [ebp-4],eax
6984b9dd ff152416896b    call    dword ptr [(6b891624)]
6984b9e3 c1e810          shr     eax,10h
6984b9e6 a806            test    al,6
6984b9e8 7422            je      (6984ba0c)
6984b9ea 6a00            push    0
6984b9ec 680e020000      push    20Eh
0:000> * //The call looks like a good candidate to replace with a call
to 0
0:000> eb 6984b9dd
6984b9dd ff ff
ff
6984b9de 15 15
15
```

Figure 11: Chrome Crash Setup (continued)

```
6984b9df 24 00
00
6984b9e0 16 00
00
6984b9e1 89 00
00
6984b9e2 6b 00
00
6984b9e3 c1
0:000> u 6984b9c2 L10
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x24:
6984b9c2 85c0            test    eax,eax
6984b9c4 7546            jne
chrome_696b0000!base::MessagePumpForUI::WaitForWork+0x6e
6984b9c6 6a06            push    6
6984b9c8 8945e4          mov     dword ptr [ebp-1Ch],eax
6984b9cb 8945e8          mov     dword ptr [ebp-18h],eax
6984b9ce 8945ec          mov     dword ptr [ebp-14h],eax
6984b9d1 8945f0          mov     dword ptr [ebp-10h],eax
6984b9d4 8945f4          mov     dword ptr [ebp-0Ch],eax
6984b9d7 8945f8          mov     dword ptr [ebp-8],eax
6984b9da 8945fc          mov     dword ptr [ebp-4],eax
6984b9dd ff1500000000    call    dword ptr ds:[0]
6984b9e3 c1e810          shr     eax,10h
0:000> * //The change looks good. Time to detach and wait for the crash.
0:000> qd
```

Figure 11: Chrome Crash Setup (continued)

Once Chrome resumed, it immediately crashed as evidenced by the dialog shown in Figure 12. This resulted in a dump and text file being saved to C:\Program Files\Google\CrashReports by GoogleCrashHandler.exe. Both files had the same name, which was a GUID, only distinguished by their respective file extensions. After GoogleCrashHandler completed its operation, it deleted the text file and renamed to dump file to Chrome-last.dmp, overwriting any existing Chrome-last.dmp.

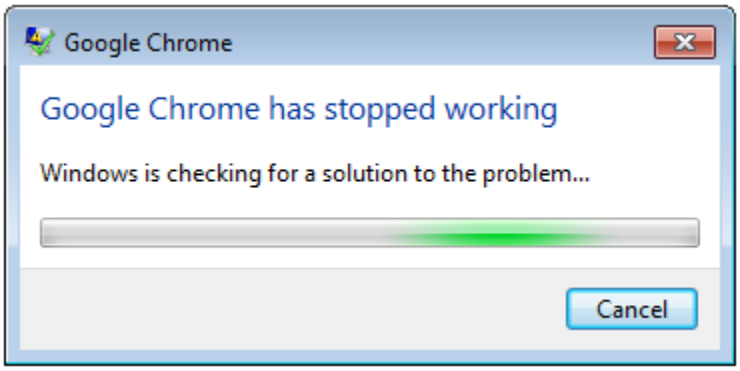


Figure 12: Chrome Crash

The dump file was not a complete userdump, however it did contain the crashing stack, registers, and portions of memory. Unfortunately for the user, in addition to being able to determine the username, hostname, domain, domain controller, and the fact that the user was using incognito mode from the process environment block (PEB) (Figure 13), the dump contained the URL of one of the incognito tabs (Figure 14). Presumably this dump was uploaded to Google. It was also left on the disk as the Chrome-last.dmp file. The text file contained mostly unintelligible hex values, though it did indicate that the browser was running in incognito mode, and listed the Chrome version number.

```
0:000> !peb
PEB at 7ffde000
    CommandLine:  '"C:\Program Files\Google\Chrome\Application\chrome.exe"  --
incognito'
...
    Environment:  001481d0
        COMPUTERNAME=BROWSERTEST
        LOGONSERVER=\\BROWSERTEST
        USERDOMAIN=browsertest
        USERNAME=testuser
      ...
```

Figure 13: Chrome PEB from dump

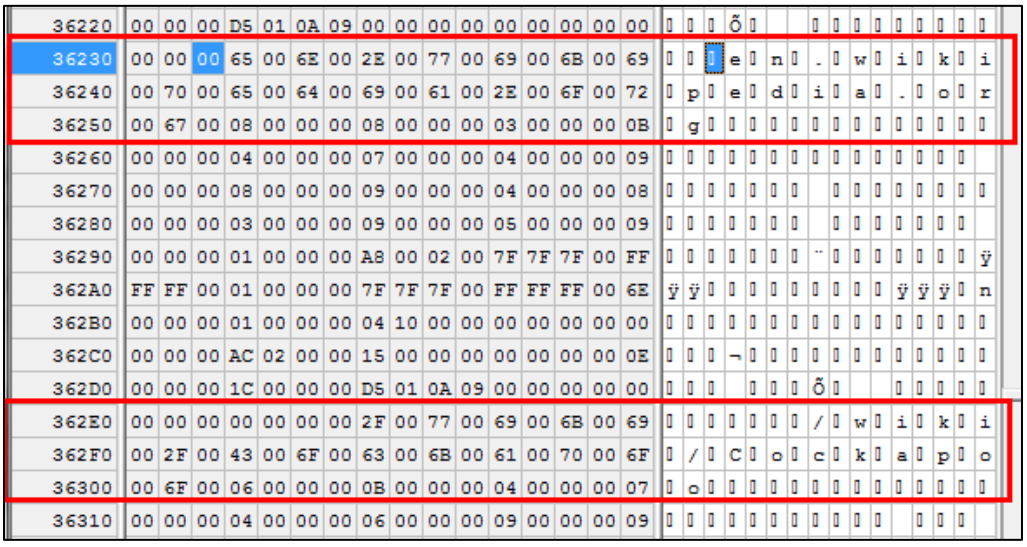


Figure 14: URL in Chrome memory dump

Firefox

Firefox could be crashed in much the same was as Chrome. The main thread's most recent Mozilla-provided function was unassembled from its return address and the first call was changed to a null reference (Figure 15). Like Chrome, Firefox has its own crash-reporting agent, crashreporter.exe, which launched as soon as the process crashed. Their crash handler wrote a dump and an .extra file to C:\Users\testuser\AppData\Roaming\Mozilla\Firefox\Crash Reports\pending. Unlike Chrome's crash reporting, Mozilla prompts the user for how they would like to proceed concerning reporting (Figure 16).

```
0:034> ~0s
eax=00000001 ebx=00000001 ecx=75e2cc37 edx=00000030 esi=75e2634a
edi=00000000
eip=77397094 esp=0024c998 ebp=0024ca54 iopl=0         nv up ei pl nz na po
nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00200202
ntdll!KiFastSystemCallRet:
77397094 c3              ret
0:000> k
ChildEBP RetAddr
0024c994 75e266c9 ntdll!KiFastSystemCallRet
0024c998 68948f6d USER32!NtUserWaitMessage+0xc
0024ca54 689497d2 xul!nsAppShell::ProcessNextNativeEvent+0x35d
…
0024ecf8 01351742 xul!XRE_main+0x30
0024f7d0 01351a64 firefox!wmain+0x742
0024f814 7700ed6c firefox!__tmainCRTStartup+0x122
0024f820 773b377b kernel32!BaseThreadInitThunk+0xe
0024f860 773b374e ntdll!__RtlUserThreadStart+0x70
0024f878 00000000 ntdll!_RtlUserThreadStart+0x1b
0:000> u 68948f6d
xul!nsAppShell::ProcessNextNativeEvent+0x35d:
68948f6d 84db            test    bl,bl
68948f6f 0f85dbfcffff    jne     (68948c50)
68948f75 e997fdffff      jmp     (68948d11)
68948f7a 8b4c2418        mov     ecx,dword ptr [esp+18h]
68948f7e 8b11            mov     edx,dword ptr [ecx]
68948f80 8b4234          mov     eax,dword ptr [edx+34h]
68948f83 ffd0            call    eax
68948f85 8a44240e        mov     al,byte ptr [esp+0Eh]
0:000> eb 68948f83
68948f83 ff ff
ff
68948f84 d0 15
15
68948f85 8a 00
00
68948f86 44 00
00
```

Figure 15: Firefox Crash Setup

```
68948f87 24 00
00
68948f88 0e 00
00
68948f89 5f
0:000> u 68948f6d
xul!nsAppShell::ProcessNextNativeEvent+0x35d:
68948f6d 84db            test    bl,bl
68948f6f 0f85dbfcffff    jne     (68948c50)
68948f75 e997fdffff      jmp     (68948d11)
68948f7a 8b4c2418        mov     ecx,dword ptr [esp+18h]
68948f7e 8b11            mov     edx,dword ptr [ecx]
68948f80 8b4234          mov     eax,dword ptr [edx+34h]
68948f83 ff1500000000    call    dword ptr ds:[0]
68948f89 5f              pop     edi
0:000> qd
```

Figure 15: Firefox Crash Setup (continued)

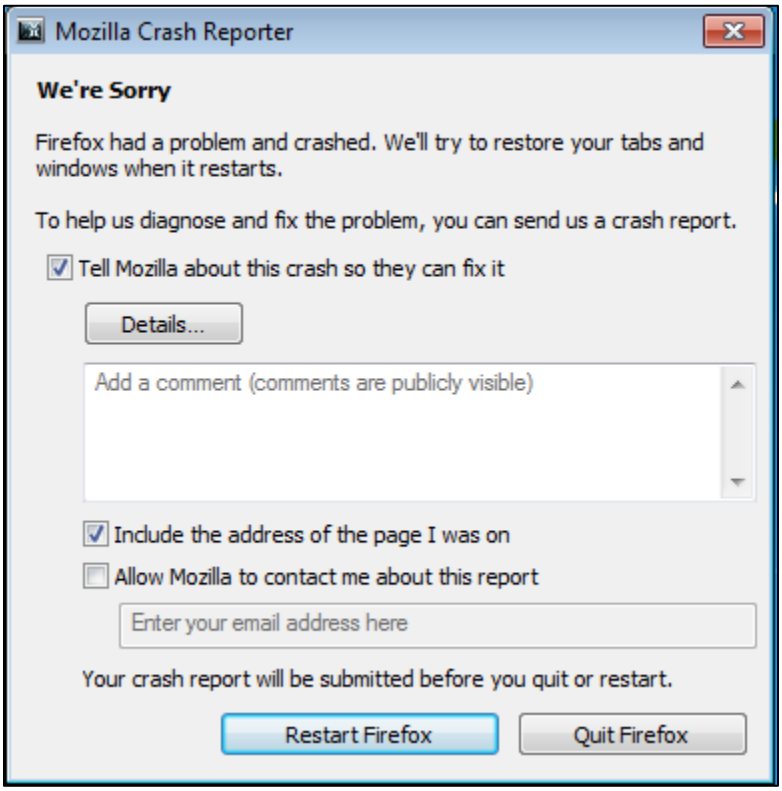


Figure 16: Mozilla Crash Reporting

Reviewing the data that Firefox would submit if the user left "Tell Mozilla about this crash…" checked (default), the .extra file was found to contain the time of the crash,

Firefox version, crashing page URL, and report server URL, as well as Winsock version information. Presumably if the user unchecked the include page address option, the crashing page URL would be removed.

The dump was in some ways better at protecting information than Chrome. For example, it did not contain a PEB, which means the dump excluded environment variables like machine name and username. The dump did contain one of the tab's URL (Figure 17), as well as a path to the user's profile, which contained the username (Figure 18). Presumably, the crash reporter process does not scrub the dump for any residual sensitive strings before submitting the report, regardless of the include address checkbox setting.

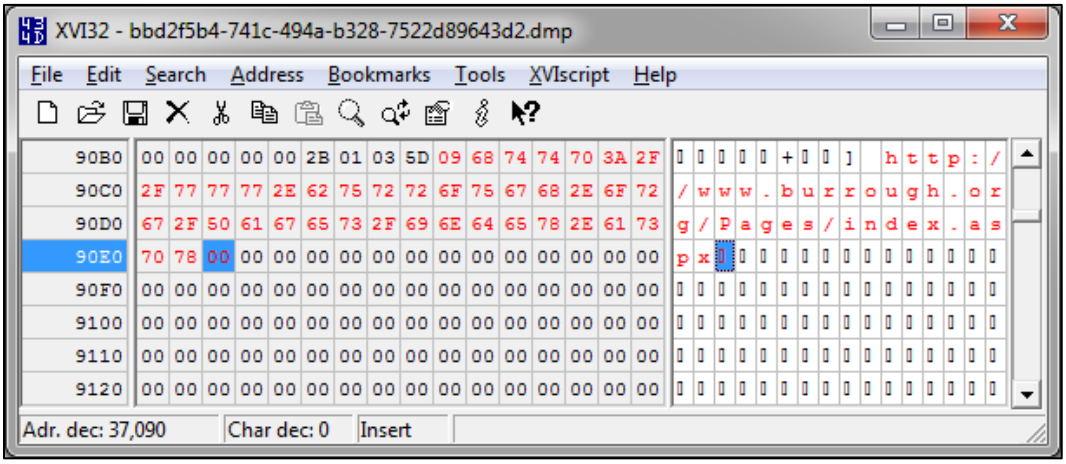


Figure 17: Firefox Dump Leaked URL

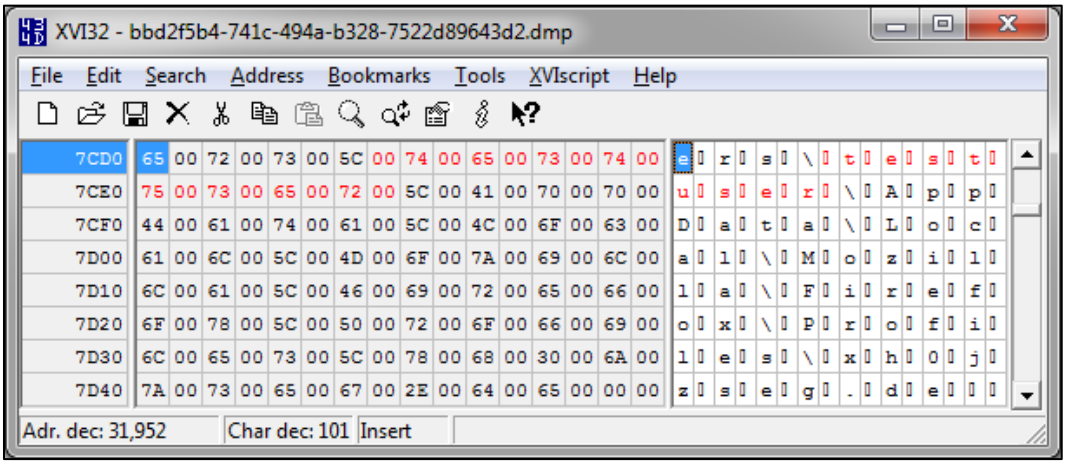


Figure 18: Firefox Dump Leaked Username

Internet Explorer

Using the same call pointer nullification method as in the previous browsers, IE crashed and was automatically restarted by werfault.exe, the Windows Error Reporting program (Figure 19). Because the first code unassembled in IE did not contain a call, an existing compare instruction was intentionally replaced with a null call (Figure 20).

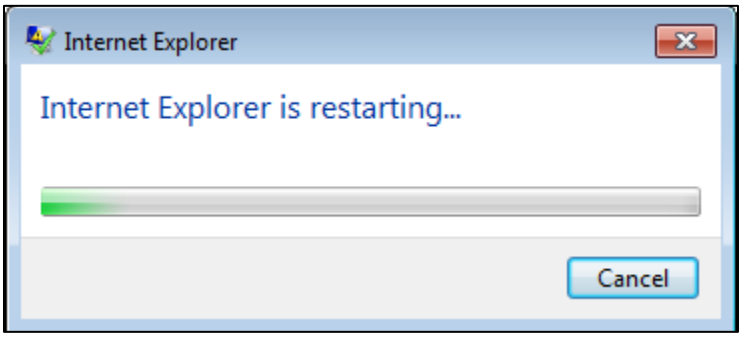


Figure 19: IE Crash Auto-Restart

It should be noted that when IE automatically restarted, it started out of InPrivate mode and did not recall any of the previously open sites, which is a good design in terms of protecting user privacy. Additionally, WER fault did not collect or upload a dump file during testing. Instead, it first created an XML file at C:\Users\testuser\AppData\Local\Temp\WERD8EC.tmp.WERInternalMetadata.xml, which was deleted as soon as WER finished. WERFault.exe then generated a Report.wer file at C:\Users\testuser\AppData\Local\Microsoft\Windows\WER\ReportArchive\AppCrash_iexplore.exe_e5177260de2b2649acbc1073ca731d7f07aa98a_05321e81\Report.wer. This file contained the crashing module, offset, and version information, as well as the loaded module names from the IE process, as well as a few error message strings. It did not contain any identifying information about the user or their browsing session. A copy of the report can be seen in Figure 21 (loaded module list omitted for brevity).

```
0:011> ~0s
eax=000000c0 ebx=00000113 ecx=7ffd4000 edx=00000030 esi=00000002 edi=001de4ac
eip=77457094 esp=001de460 ebp=001de480 iopl=0         nv up ei ng nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000297
ntdll!KiFastSystemCallRet:
77457094 c3              ret
0:000> k
ChildEBP RetAddr
001de45c 75a24473 ntdll!KiFastSystemCallRet
…
001de864 75a262f9 kernel32!WaitForMultipleObjectsExImplementation+0xe0
001de8b8 6f8d2006 USER32!RealMsgWaitForMultipleObjectsEx+0x13c
001de8dc 6f8d239f IEUI!CoreSC::Wait+0x50
…
001dfd38 00e81226 iexplore!wWinMain+0x391
…
001dfe30 00000000 ntdll!_RtlUserThreadStart+0x1b
0:000> u 6f8d2006 L3
IEUI!CoreSC::Wait+0x50:
6f8d2006 3dc0000000      cmp     eax,0C0h
6f8d200b 74e7            je      IEUI!CoreSC::Wait+0x3e (6f8d1ff4)
6f8d200d 5f              pop     edi
0:000> eb 6f8d2006
6f8d2006 3d ff
ff
6f8d2007 c0 15
15
6f8d2008 00 00
00
6f8d2009 00 00
00
6f8d200a 00 00
00
6f8d200b 74 00
00
6f8d200c e7
0:000> u 6f8d2006 L3
IEUI!CoreSC::Wait+0x50:
6f8d2006 ff1500000000    call    dword ptr ds:[0]
6f8d200c e75f            out     5Fh,eax
6f8d200e 3bc6            cmp     eax,esi
0:000> qd
```

Figure 20: IE Crash Setup

```
Version=1
EventType=APPCRASH
EventTime=129982065798324231
ReportType=2
Consent=1
UploadTime=129982065801917981
ReportIdentifier=48701bee-35f3-11e2-9ffd-0800275e6093
IntegratorReportIdentifier=48701bed-35f3-11e2-9ffd-0800275e6093
Response.BucketId=3168233462
Response.BucketTable=1
Response.type=4
Sig[0].Name=Application Name
Sig[0].Value=iexplore.exe
Sig[1].Name=Application Version
Sig[1].Value=9.0.8112.16450
Sig[2].Name=Application Timestamp
Sig[2].Value=503723f6
Sig[3].Name=Fault Module Name
Sig[3].Value=IEUI.dll
Sig[4].Name=Fault Module Version
Sig[4].Value=9.0.8112.16450
Sig[5].Name=Fault Module Timestamp
Sig[5].Value=503721ca
Sig[6].Name=Exception Code
Sig[6].Value=c0000005
Sig[7].Name=Exception Offset
Sig[7].Value=00002006
DynamicSig[1].Name=OS Version
DynamicSig[1].Value=6.1.7601.2.1.0.768.2
DynamicSig[2].Name=Locale ID
DynamicSig[2].Value=1033
DynamicSig[22].Name=Additional Information 1
DynamicSig[22].Value=0a9e
DynamicSig[23].Name=Additional Information 2
DynamicSig[23].Value=0a9e372d3b4ad19135b953a78882e789
DynamicSig[24].Name=Additional Information 3
DynamicSig[24].Value=0a9e
DynamicSig[25].Name=Additional Information 4
DynamicSig[25].Value=0a9e372d3b4ad19135b953a78882e789
UI[2]=C:\Program Files\Internet Explorer\iexplore.exe
UI[3]=Internet Explorer has stopped working
UI[4]=Windows can check online for a solution to the problem and try to restart the program.
UI[5]=Check online for a solution and restart the program
UI[6]=Check online for a solution later and close the program
UI[7]=Close the program
LoadedModule[0]=C:\Program Files\Internet Explorer\iexplore.exe
…
ConsentKey=APPCRASH
AppName=Internet Explorer
AppPath=C:\Program Files\Internet Explorer\iexplore.exe
```

Figure 21: WER Report

Opera

Again, the same method to induce a crash in the other browsers was used here (Figure 22). One interesting feature of Opera is it appears to handle its crashes by attaching another instance of Opera to the crashing instance instead of using a separate crash reporting utility. This new process not only handles the crash, but also prompts the user if they want to restore their session. Like IE, it a restored session does not include private browsing tabs.

When Opera crashed, an approximately 1.2 MB crash.txt file was written to C:\Users\testuser\AppData\Local\Temp\opera-20121124005901. It contained mostly sections of stack memory from Opera, including things like environment variables, revealing the username and domain of the user (Figure 24). No URLs or page contents from the session were observed in the file. Opera did not present any options or ask the user if information about the crash could be uploaded.

```
0:011> ~0s
0:000> k
ChildEBP RetAddr
0014e6e0 76e8cde0 ntdll!KiFastSystemCallRet
0014e6e4 76e8ce13 USER32!NtUserGetMessage+0xc
0014e700 6ae090af USER32!GetMessageW+0x33
WARNING: Stack unwind information not available. Following frames may be wrong.
0014e774 6afbf45b Opera_6ac40000!OpSetLaunchMan+0x1c7e19
...
0:000> u 6ae090af L3
Opera_6ac40000!OpSetLaunchMan+0x1c7e19:
6ae090af 85c0            test    eax,eax
6ae090b1 0f8529ffffff    jne     Opera_6ac40000!OpSetLaunchMan+0x1c7d4a (6ae08fe0)
6ae090b7 5f              pop     edi
0:000> eb 6ae090b1
6ae090b1 0f ff
ff
6ae090b2 85 15
15
6ae090b3 29 00
00
6ae090b4 ff 00
00
6ae090b5 ff 00
00
6ae090b6 ff 00
00
6ae090b7 5f
0:000> u 6ae090af L3
Opera_6ac40000!OpSetLaunchMan+0x1c7e19:
6ae090af 85c0            test    eax,eax
6ae090b1 ff1500000000    call    dword ptr ds:[0]
6ae090b7 5f              pop     edi
0:000> qd
```

Figure 22: Opera Crash Setup

Figure 23: Opera Restart

```
OPERA-CRASHLOG V1 desktop 12.02 1578 windows
Opera.exe 1578 caused exception C0000005 at address 67C190B1 (Base: C80000)

Registers:
EAX=00000001   EBX=76E8CDE8   ECX=0000042C   EDX=77BD7094   ESI=02109FB8
EDI=0210A044   EBP=0016E634   ESP=0016E5D8   EIP=67C190B1 FLAGS=00010202
CS=001B   DS=0023   SS=0023   ES=0023   FS=003B   GS=0000
FPU stack:
00000000000000000000 00000000000000000000 00000000000000000000
00000000000000000000 00000000000000000000 00000000000000000000
00000000000000000000 00000000000000000000 SW=0127 CW=027F

Stack dump:
0016E5D8  7773E868  00000400  00000000  0016FC24  hèsw.......$ü.
0016E5E8  7265704F  614D2061  57206E69  00646E69  Opera Main Wind.
…
00210FB0  55 00 53 00  45 00 52 00  44 00 4F 00  4D 00 41 00  U.S.E.R.D.O.M.A.
00210FC0  49 00 4E 00  3D 00 62 00  72 00 6F 00  77 00 73 00  I.N.=.b.r.o.w.s.
00210FD0  65 00 72 00  74 00 65 00  73 00 74 00  00 00 55 00  e.r.t.e.s.t...U.
00210FE0  53 00 45 00  52 00 4E 00  41 00 4D 00  45 00 3D 00  S.E.R.N.A.M.E.=.
00210FF0  74 00 65 00  73 00 74 00  75 00 73 00  65 00 72 00  t.e.s.t.u.s.e.r.
…
```

Figure 24: Opera crash log

Safari

Unlike Chrome and Firefox, which have their own out of process crash handlers or Opera with its self-contained handler, Safari does not provide its own crash reporting mechanism. After inducing the crash (Figure 25), Windows Error Reporting captured the crash (Figure 26) and uploaded a report to the Windows Error Reporting site. Apple can later access these reports if they have registered as an ISV (Microsoft, 2012). The WER Report did not contain any sensitive information – only the module, version information, and exception record (Figure 27; loaded module information omitted for brevity).

```
0:026> ~0s
0:000> k
ChildEBP RetAddr
0030f508 7602cde0 ntdll!KiFastSystemCallRet
0030f50c 7602ce13 USER32!NtUserGetMessage+0xc
0030f528 6a7af4de USER32!GetMessageW+0x33
WARNING: Stack unwind information not available. Following frames may be wrong.
00000000 00000000 Safari_6a750000+0x5f4de
0:000> u 6a7af4de  L2
Safari_6a750000+0x5f4de:
6a7af4de 83f8ff          cmp     eax,0FFFFFFFFh
6a7af4e1 74ba            je      Safari_6a750000+0x5f49d (6a7af49d)
0:000> eb 6a7af4de
6a7af4de 83 ff
ff
6a7af4df f8 15
15
6a7af4e0 ff 00
00
6a7af4e1 74 00
00
6a7af4e2 ba 00
00
6a7af4e3 85 00
00
6a7af4e4 c0
0:000> u 6a7af4de L2
Safari_6a750000+0x5f4de:
6a7af4de ff1500000000    call    dword ptr ds:[0]
6a7af4e4 c074548b07      sal     byte ptr [esp+edx*2-75h],7
0:000> qd
```

Figure 25: Safari Crash Setup

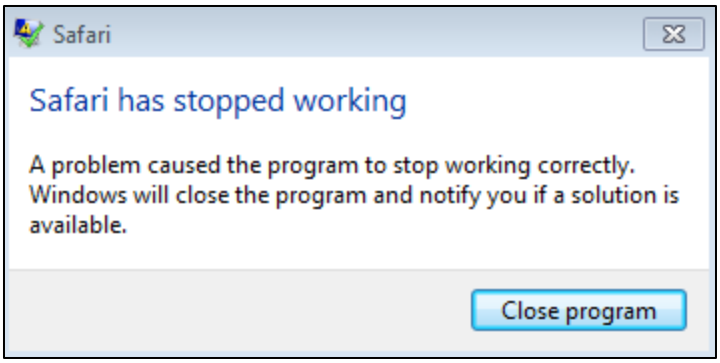


Figure 26: WER Dialog for Safari

```
Version=1
EventType=APPCRASH
EventTime=129982488858088750
ReportType=2
Consent=1
UploadTime=129982488859963750
ReportIdentifier=c8e9f6b5-3655-11e2-9f71-0800275e6093
IntegratorReportIdentifier=c8e9f6b4-3655-11e2-9f71-0800275e6093
Response.BucketId=3272016251
Response.BucketTable=1
Response.type=4
Sig[0].Name=Application Name
Sig[0].Value=Safari.exe
Sig[1].Name=Application Version
Sig[1].Value=5.34.57.2
Sig[2].Name=Application Timestamp
Sig[2].Value=4f982b5e
Sig[3].Name=Fault Module Name
Sig[3].Value=Safari.dll
Sig[4].Name=Fault Module Version
Sig[4].Value=7534.57.2.4
Sig[5].Name=Fault Module Timestamp
Sig[5].Value=4f982b22
Sig[6].Name=Exception Code
Sig[6].Value=c0000005
Sig[7].Name=Exception Offset
Sig[7].Value=0005f4de
DynamicSig[1].Name=OS Version
DynamicSig[1].Value=6.1.7601.2.1.0.768.2
DynamicSig[2].Name=Locale ID
DynamicSig[2].Value=1033
DynamicSig[22].Name=Additional Information 1
DynamicSig[22].Value=0a9e
DynamicSig[23].Name=Additional Information 2
```

Figure 27: WER Report for Safari

```
DynamicSig[23].Value=0a9e372d3b4ad19135b953a78882e789
DynamicSig[24].Name=Additional Information 3
DynamicSig[24].Value=0a9e
DynamicSig[25].Name=Additional Information 4
DynamicSig[25].Value=0a9e372d3b4ad19135b953a78882e789
UI[2]=C:\Program Files\Safari\Safari.exe
UI[3]=Safari has stopped working
UI[4]=Windows can check online for a solution to the problem.
UI[5]=Check online for a solution and close the program
UI[6]=Check online for a solution later and close the program
UI[7]=Close the program
LoadedModule[0]=C:\Program Files\Safari\Safari.exe
...
State[0].Key=Transport.DoneStage1
State[0].Value=1
State[1].Key=DataRequest
State[1].Value=Bucket=-1022951045/nBucketTable=1/nResponse=1/n
FriendlyEventName=Stopped working
ConsentKey=APPCRASH
AppName=Safari
AppPath=C:\Program Files\Safari\Safari.exe
```

Figure 27: WER Report for Safari (continued)

Upon restarting Safari, it did not provide any indication that it had crashed. A new, empty, normal session was created.

Test 4: Crashing Plug-In

Another possible attack vector for browser privacy modes is to leverage an extension that intentionally crashes in the hopes of crashing the browser. As in test three, this could result in a memory dump being saved to disk or uploaded to a crash submission site. This memory dump could contain sensitive user information.

Chrome

As described in test 2, Chrome's extension architecture consists of package JavaScript/JSON/HTML files. Given that no native code is running in the extension, it seems the only way an extension could cause a crash would be if it found and exposed a bug in Chrome itself. This has occurred, such as the one documented at (Dunn, 2011), however that bug has since been fixed. Attempting to find a bug in Chrome's API is outside the scope of this paper, so this test was not attempted on Chrome.

Chrome does support NPAPI plugins (see Firefox section of this test for a description and more information); however, Chrome places additional restrictions on these plugins, so this is a less-likely attack vector on Chrome than on Firefox (Google, 2012).

Firefox

One of the aspects of Firefox that makes it so extensible is the variety of add-ons it accepts. In addition to the script-based extensions as was used in test 2, Firefox can also run native plugins built with the Gecko SDK. This includes native C++ code written using the Netscape PlugIn Application Programming Interface (NPAPI). While other browsers offer differing levels of support for NPAPI plugins, Mozilla fully supports them, due to its ties to Netscape.

Firefox also uses a novel plugin-container process to host native plugins outside the main Firefox process. This prevents a single native plugin's crash from disrupting the user's browser session. There are two problems with this approach, as it relates to protecting a user's privacy in crash reporting. First, Mozilla may collect the URL of the page being visited when a plugin crashes (Verdi, 2010). Second, Firefox may only silos select plugins into external processes, not all of them (Wyman, 2010).

To test Firefox using a crashing plugin, the NPAPI sample code served as the basis for a native Firefox plugin (Mozilla, 2007). Once loaded into the browser, the Firefox did create a plugin-container process for the plugin. Upon triggering the crash, Firefox offered to send a crash report to Mozilla (Figure 28).

Figure 28: Firefox Crash Report Prompt

A process monitor log revealed that when a crash occurs, Firefox stores a usermode process minidump and an .extra file in %APPDATA%\Mozilla\Firefox\Crash Reports\pending\. These files are left even if the user dismisses the send crash report dialog.

In reviewing a dump and .extra file taken when the test NPAPI plugin crashed while loaded in private browsing mode, it appears that no discernible user information is present. At the time of the crash, two tabs were loaded – one on yahoo.com and the other on the NPAPI text page bundled with the sample code that exposes the plugin's functionality. The .extra file was a plain-text document containing the URL of the crash report, the Winsock version, the Firefox version, several timestamps, and the name of the crashing plugin (Figure 29). The dump file was not a complete user mode memory dump, and only contained stacks, registers, and small portions of memory. Searching the dump with both a debugger and a hex editor, it contained no trace of either page URL, or any of the pages' content or markup.

```
Winsock_LSP=MSAFD Tcpip [TCP/IP] : 2 : 1 :  \n MSAFD Tcpip [UDP/IP] : 2 : 2 :
%SystemRoot%\\system32\\mswsock.dll \n MSAFD Tcpip [RAW/IP] : 2 : 3 :  \n MSAFD Tcpip [TCP/IPv6] :
2 : 1 : %SystemRoot%\\system32\\mswsock.dll \n MSAFD Tcpip [UDP/IPv6] : 2 : 2 :  \n MSAFD Tcpip
[RAW/IPv6] : 2 : 3 : %SystemRoot%\\system32\\mswsock.dll \n RSVP TCPv6 Service Provider : 2 : 1 :  \n
RSVP TCP Service Provider : 2 : 1 : %SystemRoot%\\system32\\mswsock.dll \n RSVP UDPv6 Service
Provider : 2 : 2 :  \n RSVP UDP Service Provider : 2 : 2 : %SystemRoot%\\system32\\mswsock.dll
AdapterVendorID=0x10de
EMCheckCompatibility=true
ProductName=Firefox
Vendor=Mozilla
InstallTime=1352338926
Theme=classic/1.0
Notes=AdapterVendorID: 0x10de, AdapterDeviceID: 0x0e23, AdapterSubsysID: 13663842,
AdapterDriverVersion: 9.18.13.697\nD2D? D2D+ DWrite? DWrite+ D3D10 Layers? D3D10 Layers+
Version=16.0.2
ReleaseChannel=release
ServerURL=https://crash-reports.mozilla.com/submit?id={ec8030f7-c20a-464f-9b0e-
13a3a9e97384}&version=16.0.2&buildid=20121024073032
AdapterDeviceID=0x0e23
Add-ons={972ce4c6-7e08-4474-a285-3208198ce6fd}:16.0.2
BuildID=20121024073032
ProductID={ec8030f7-c20a-464f-9b0e-13a3a9e97384}
CrashTime=1353508819
StartupTime=1353508768
ProcessType=plugin
PluginVersion=
PluginName=
PluginFilename=npapitest.dll
```

Figure 29: Firefox .extra file

This is reassuring, since anyone is able to lookup information from any crash on Mozilla's crash reporting site at https://crash-stats.mozilla.com/. The site even allows people to filter by plugin name, so an attacker could reveal just the information from their plugin.

Internet Explorer

Internet Explorer loads extensions into its own process, which means a crashing extension will also crash the browser. IE does launch multiple processes depending on the number of tabs in use, and can recover if one of its instances crashes (Zeigler, 2008). However, this still means that it is possible that a crashing extension could result in private information being saved in a crash dump and potentially uploaded to Microsoft's Windows Error Reporting (WER) servers.

To test Internet Explorer's handling of extension crashes, a slight modification was made to the IE Extension used in test 2. In the function that is called when a page completes loading, a test was added to check if the loaded URL contains "yale.edu". If it does, the extension calls Debugger.Break(). If a debugger is not attached, the process will crash.

Reviewing the data collected by Windows Error Reporting, it appears that only a text file containing the crash reason and loaded modules was submitted (Figure 30). While WER does have the ability to collect process mini-dumps, triggering this particular crash did not lead to a submission. It is possible other classes of crashes could result in a dump being upload, though that was not observed in testing.

It should also be noted that Microsoft's System Center Desktop Error Monitoring product allows IT administrators to collect crash information from their machines, so in a corporate environment it is possible an IT administrator would receive crash dumps in this situation (Vel, 2008). Even without System Center, an administrator can still configure WER to save crash dumps (Microsoft Corporation, 2012).

```
Version=1
EventType=CLR20r3
EventTime=129978175308259746
ReportType=2
Consent=1
UploadTime=129978175352039043
ReportIdentifier=7540b50d-3269-11e2-a5ab-0800275e6093
IntegratorReportIdentifier=7540b50c-3269-11e2-a5ab-0800275e6093
Response.BucketId=50
Response.BucketTable=5
Response.type=4
Sig[0].Name=Problem Signature 01
Sig[0].Value=iexplore.exe
Sig[1].Name=Problem Signature 02
Sig[1].Value=9.0.8112.16455
Sig[2].Name=Problem Signature 03
Sig[2].Value=507284ba
Sig[3].Name=Problem Signature 04
Sig[3].Value=IEAddOn
Sig[4].Name=Problem Signature 05
Sig[4].Value=1.0.0.0
Sig[5].Name=Problem Signature 06
Sig[5].Value=50a939d5
Sig[6].Name=Problem Signature 07
Sig[6].Value=4
Sig[7].Name=Problem Signature 08
```

Figure 30: WER Report from IE Extension Crash

```
Sig[7].Value=4f
Sig[8].Name=Problem Signature 09
Sig[8].Value=Debugger.Break
DynamicSig[1].Name=OS Version
DynamicSig[1].Value=6.1.7601.2.1.0.768.2
DynamicSig[2].Name=Locale ID
DynamicSig[2].Value=1033
DynamicSig[22].Name=Additional Information 1
DynamicSig[22].Value=301c
DynamicSig[23].Name=Additional Information 2
DynamicSig[23].Value=301cf6e2b3f77c41ce76c3c05aac9deb
DynamicSig[24].Name=Additional Information 3
DynamicSig[24].Value=4330
DynamicSig[25].Name=Additional Information 4
DynamicSig[25].Value=4330a527e81099ec1a054fe374ced8bf
UI[2]=C:\Program Files\Internet Explorer\iexplore.exe
UI[3]=Internet Explorer has stopped working
UI[4]=Windows can check online for a solution to the problem.
UI[5]=Check online for a solution and close the program
UI[6]=Check online for a solution later and close the program
UI[7]=Close the program
LoadedModule[0]=C:\Program Files\Internet Explorer\iexplore.exe
…
LoadedModule[104]=C:\Windows\system32\FaultRep.dll
State[0].Key=Transport.DoneStage1
State[0].Value=1
State[1].Key=DataRequest
State[1].Value=Bucket=50/nBucketTable=5/nResponse=1/n
FriendlyEventName=Stopped working
ConsentKey=CLR20r3
AppName=Internet Explorer
AppPath=C:\Program Files\Internet Explorer\iexplore.exe
```

Figure 30: WER Report from IE Extension Crash (Continued)

Opera

Like with Chrome, Opera's extensions generally utilize Opera's JavaScript architecture, which makes the browser more crash resistant to misbehaving plug-ins. Without finding a flaw in Opera's JavaScript processing engine or APIs, it is unlikely to cause a browser crash from an extension.

Opera also supports NPAPI plugins, though in testing Opera did not pick up a NPAPI plug-in installed on the system. Additionally, Opera's NPAPI documentation implies that some security boundaries are enforced for NPAPI plugins (Opera Software, 2008), and that

Opera is working on an Out-of-Process plugin model that may help protect user information in the event of a crash (Mills, 2012).

Safari

This test was skipped on Safari, as Apple requires developers to sign an explicit agreement that states that they will not produce extensions that intentionally interfere with the operation of Safari. The other browsers tested in this section did not have a formal EULA and registration process required to develop an extension.

Again, it should be noted that Safari's extension platform appears to be JavaScript-based like Chrome. As such, it is likely that a misbehaving extension would probably not result in a browser crash. This has not been tested in the course of this paper, however.

Test 5: Abandoned Browser Reconnaissance

A final possible attack surface is an abandoned browser. This scenario could easily occur in a library or web café setting. Consider a user browsing in private mode who subsequently closes any tabs containing sensitive sites, but fails to close the browser before stepping away.

A complete memory dump of the process will be created using Task Manager's "Create Dump File" option in the processes tab. This dump file could easily be copied to a thumb drive and examined later by an attacker. In this test, the dump will be searched for references to URLs and page content from the victim's sensitive sites.

Chrome

For this test, Chrome was started directly in Incognito mode via its jump list from its pinned taskbar icon. From the start page, two new tabs were created, yielding three tabs all in Incognito mode. The first tab was left blank while the second tab was navigated to burrough.org and the third tab to wikipedia.org/wiki/cockapoo. Once both pages completed loading, both tabs were closed. After about 30 seconds, Task Manager was used to create a dump file of the main Chrome process (Figure 31). Once completed, the dump was saved to C:\Users\testuser\AppData\Local\Temp\chrome.DMP. It was 135 MB.

The dump was then copied off the VM and run against strings.exe (Russinovich, 2012) on a different machine with these parameters: "strings.exe -n 10 chrome.dmp > chrome-strings.txt", which found all Unicode and ASCII strings 10 or more characters in length and

saved them to a text file, which was 17.8 MB. Grep was then used to find any lines that contained burrough or cockapoo (case insensitive).

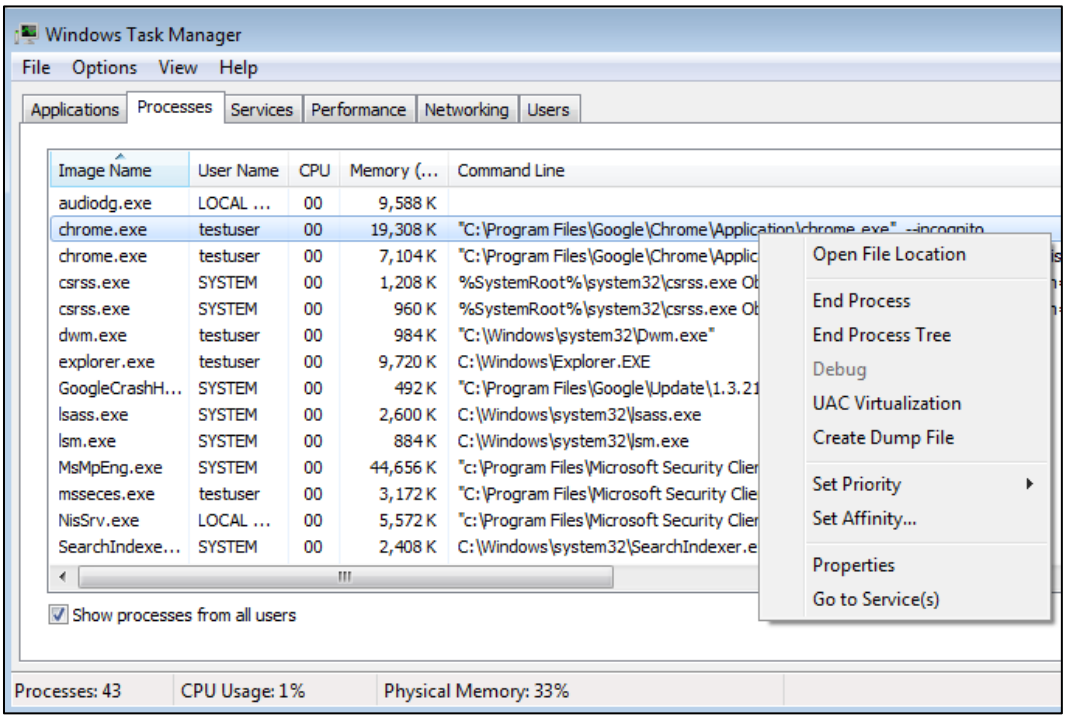


Figure 31: Dumping Chrome via Task Manager

The dump contained 104 strings of 10 or more characters with burrough in them and 162 strings containing cockapoo. Figure 32 and Figure 33 contain a sampling of the strings found. An additional search for the Word Rochester (page contents from Burrough.org contain the author's CV, including Rochester Institute of Technology where he completed his undergraduate degree), found seven hits:

grep -U -i Rochester chrome-strings.txt
Rochester, NY
Rochester Institute of Technology, Rochester, NY
Rochester Institute of Technology, Rochester, NY
Rochester Institute of Technology, Rochester, NY
Nathaniel Rochester Society Scholar, Rochester Institute of Technology, 2005
Presidential Scholar, Rochester Institute of Technology, 2003
rochester.museum

```
Matthew Burrough's CV - Google Chrome
http://www.burrough.org/pages/index.aspx
http://www.burrough.org/SiteImages/mcitp.png
burrough.org
```

Figure 32: Sample of Burrough Strings in Chrome

```
Cockapoo - Wikipedia, the free encyclopedia - Google Chrome
http://wikipedia.org/wiki/cockapoo
A Cockapoo is a cross breed dog. It is the cross of an American Cocker Spaniel or English Cocker Spaniel
    and a poodle (in most cases a miniature poodle or toy poodle), or of two cockapoos.
Cockapoos are often active and agile.
^ "Cockapoo Coat Colors". Cockapoo Club of GB. Retrieved 2012-03-09.
```

Figure 33: Sample of Cockapoo Strings in Chrome

Firefox

Firefox was started in normal mode, and then immediacy placed into private browsing mode from the Firefox menu. As with Chrome, two tabs were opened in addition to the first blank tab, and these tabs were navigated to burrough.org and Wikipedia.org/wiki/cockapoo. Once loaded, these two tabs were closed. After 30 seconds, the firefox.exe process was dumped with Task Manager. The 140 MB dump was run through strings.exe on a different machine, and looking for strings ten characters long or longer.

The resulting strings output was 10 MB. It included 111 lines containing Burrough, 455 lines containing Cockapoo, and 13 lines containing Rochester. Reviewing the text, it was interesting to see that although each of these keywords had more hits than Chrome, there were many more repeats of the same strings, and many of the hits were page URLs and hyperlinks. There was very little page content in the strings. Examples can be seen in Figure 34, Figure 35 and Figure 36. This illustrates a difference in how Chrome and Firefox's memory allocation/deallocation patterns.

```
Matthew Burrough's CV
http://www.burrough.org/Pages/index.aspx
Matthew Burrough's CV - Mozilla Firefox (Private Browsing)
http://blogs.msdn.com/b/ntdebugging/archive/tags/burrough/
Transferring data from www.burrough.org
http://www.burrough.org/SiteImages/mcses.png
```

Figure 34: Sample of Burrough Strings in Firefox

```
http://en.wikipedia.org/wiki/Cockapoo
Cockapoo - Wikipedia, the free encyclopedia - Mozilla Firefox (Private Browsing)
Like many floppy-eared breeds, Cockapoos can be subject to ear infections, and it's important
    to keep their ears clean and dry.
kwww.americancockapooclub.com
kwww.cockapoo-owners-club.org.uk
/w/index.php?title=Cockapoo&action=history
```

Figure 35: Sample of Cockapoo Strings in Firefox

```
Rochester
Rochester
Rochester
Nathaniel Rochester Society Scholar, Rochester Institute of Technology, 2005
Rochester,1
Rochester
, Rochester, NY
, Rochester, NY
, Rochester, NY
, Rochester, NY
<a class="style5" href="http://www.rit.edu/"><span class="style6">Rochester
Institute of Technology</span></a>, Rochester, NY</span></p>
rochester.museum
```

Figure 36: All Rochester Strings in Firefox

Internet Explorer

Internet Explorer was also started directly into InPrivate mode via its jump list. After two tabs had been created and fully loaded the Burrough.org and Wikipedia Cockapoo pages, they were closed and a dump was created of the main iexplore.exe process 30 seconds later. The dump was 99 MB, and yielded 9.8 MB of 10+ character strings.

Reviewing the grepped outputs for Burrough, Cockapoo, and Rochester, there were 80 hits for Burrough, 79 for Cockapoo, and none for Rochester. Possibly due to IE's spate-process-per-tab architecture, there were no strings for any of the page content from either burrough.org or Wikipedia.org/wiki/cockapoo. All of the strings were URLs, links, or the page title/title bar text. Samples are in Figure 37 and Figure 38.

```
Matthew Burrough's CV - Windows Internet Explorer - [InPrivate]
http://www.burrough.org/Pages/index.aspx
burrough.org
www.burrough.org
http://www.burrough.org/favicon.ico
```

Figure 37: Sample of Burrough Strings in IE

```
http://wikipedia.org/wiki/cockapoo
Cockapoo - Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Cockapoo
Cockapoo - Wikipedia, the free encyclopedia - Windows Internet Explorer - [InPrivate]
ttp://en.wikipedia.org/w/index.php?title=Special:UserLogin&returnto=Cockapoo&type=signup
```

Figure 38: Sample of Cockapoo Strings in IE

Opera

Since Opera is unique in that its tabs – and not the window or process – are either in private or normal mode, it was launched normally, followed by the creation of two private tabs. Once the Burrough and Wikipedia pages loaded in those tabs, they were closed, leaving just the normal mode blank tab. After 30 seconds, a 133 MB dump of opera.exe was created with Task Manager. 10 MB of strings were then saved using strings.exe as in the other browser's tests in this section.

Of the strings, 39 contained Burrough, 210 cockapoo, and 6 Rochester. Samples are in Figure 39, Figure 40, and Figure 41. The strings contained a mix of URLs and page contents.

```
Address: http://www.burrough.org/Pages/index.aspx
www.burrough.org
burrough.org
Completed request to www.burrough.org
atthew Burrough's CV
http://blogs.msdn.com/b/ntdebugging/archive/tags/burrough/
```

Figure 39: Sample of Burrough Strings in Opera

```
/w/index.php?title=Cockapoo&action=history
tp://en.wikipedia.org/wiki/cockapoo
A healthy 12-week-old cockapoo.
of toy poodles, miniature poodles, cocker spaniels and cockapoos, using AKC standards and
    other information.
Characteristics of the Cockapoo"
While some Cockapoos appear more similar to Cocker Spaniels, others will exhibit more Poodle
traits, creating a variation in Cockapoo appearance and temperament.
itle: Cockapoo - Wikipedia, the free encyclopedia
Address: http://en.wikipedia.org/wiki/Cockapoo
```

Figure 40: Sample of Cockapoo Strings in Opera

Rochester, NY
Rochester, NY
Rochester, NY
Presidential Scholar, Rochester Institute of Technology, 2003
Nathaniel Rochester Society Scholar, Rochester Institute of Technology, 2005
Rochester, NY

Figure 41: All Rochester Strings in Opera

Safari

Safari was launched in normal mode then immediately switched to private browsing mode. As before, two new tabs were created, navigated to burrough.org and Wikipedia.org/wiki/cockapoo, and then closed when loading finished. A dump of Safari was captured using Task Manager when 30 seconds had elapsed with just the blank original tab open. The 133 MB dump contained 13.5 MB of 10+ character strings. Among these were 308 Burrough strings, 532 Cockapoo strings, and 8 Rochester strings.

What was interesting about Safari was that, in spite of the large number of strings, none was for page content. Each was a URL, link, or page title. In fact, the only reason Rochester appeared at all was Safari seems to pull news RSS feeds automatically for several sites, and included in these were a few stories about Rochester. None of the Rochester hits was related to the Burrough.org page.

http://www.burrough.org/Pages/index.aspx
http://burrough.org/
http://www.burrough.org/SiteImages/SBE-Certified.jpg
www.burrough.org
Matthew Burrough's CV

Figure 42: Sample of Burrough Strings in Safari

http://wikipedia.org/wiki/cockapoo
http://en.wikipedia.org/wiki/Cockapoo
http://upload.wikimedia.org/wikipedia/commons/thumb/2/20/Twelve_%2812%29_Week_Old_Cockapoo.jpg/220px-Twelve_%2812%29_Week_Old_Cockapoo.jpg
Cockapoo - Wikipedia, the free encyclopedia

Figure 43: Sample of Cockapoo Strings in Safari

Conclusion

Based on the results of the five tests, it is clear that no browser is without flaws when it comes to information disclosure about private browsing sessions. Generally, browsers were good about not leaving files from the private session on the disk, but did a poor job protecting users when tabs were closed but the browser was not. The most varied behavior came in the form of how browsers handle plugins during private browsing. At best, the browser provides warnings that plugins may cause data leaks during private browsing, and allow the plugins to be enabled or disabled individually for privacy mode, like Chrome and Opera. At worst, they run plugins by default in privacy mode with no warning, such as with Firefox and Safari. In between is Internet Explorer's design where plugins are loaded all-or-nothing in privacy mode.

In terms of crash reporting, the best behavior is when a browser only submits text files about a crash with limited information and no memory dump cached on the disk. Almost as good is when always asks if the user would like to submit a report, and offers to scrub some information, as Firefox does. However, Firefox's offer to not include the current page URL may give users a false sense of security, since that data can probably be retried from the dump file. Additionally, some of the crash data submitted to Mozilla is available publically on their website. Chrome's design is particularly lacking in that the dump contains a fair amount of information, is submitted without prompting the user, and the last dump collected is preserved in the user's profile on disk.

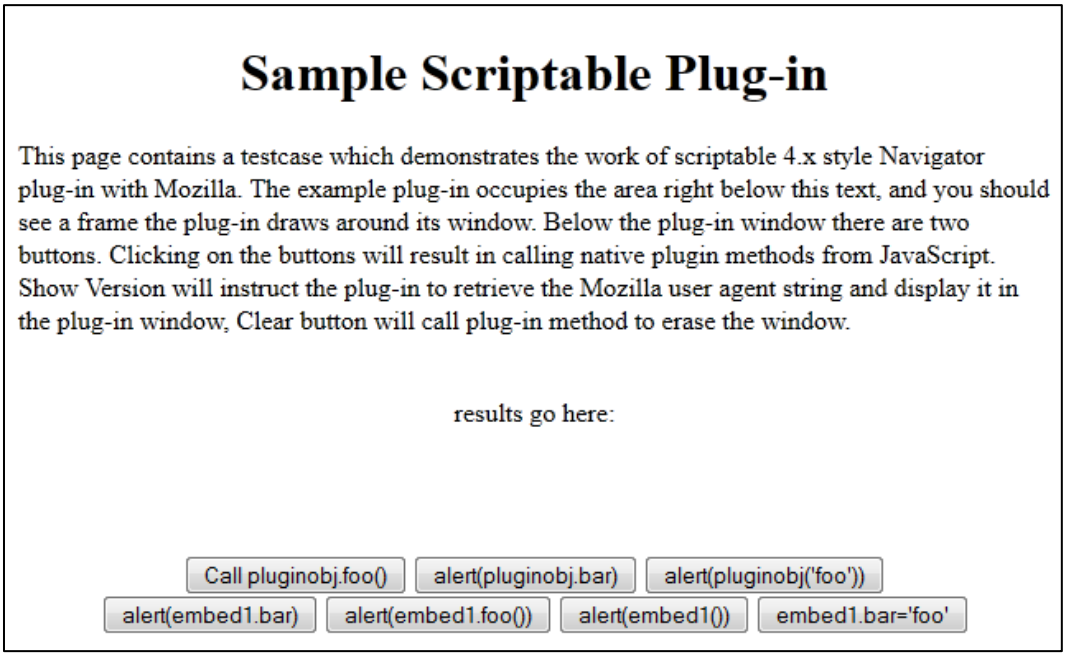
Figure 44 gives a subjective assessment of how each browser did for each test, where green indicates best performance, yellow indicates some troubling behavior, red indicates a strong concern, and black indicates a test was not performed or was inconclusive.

Browser	Residual Files	Spying Plugin	Browser Crash	Plugin Crash	Orphaned Session
Chrome	●	●	◆	■	◆
Firefox	●	◆	▲	●	◆
IE	◆	▲	●	●	▲
Opera	●	●	●	■	◆
Safari	●	◆	●	■	▲

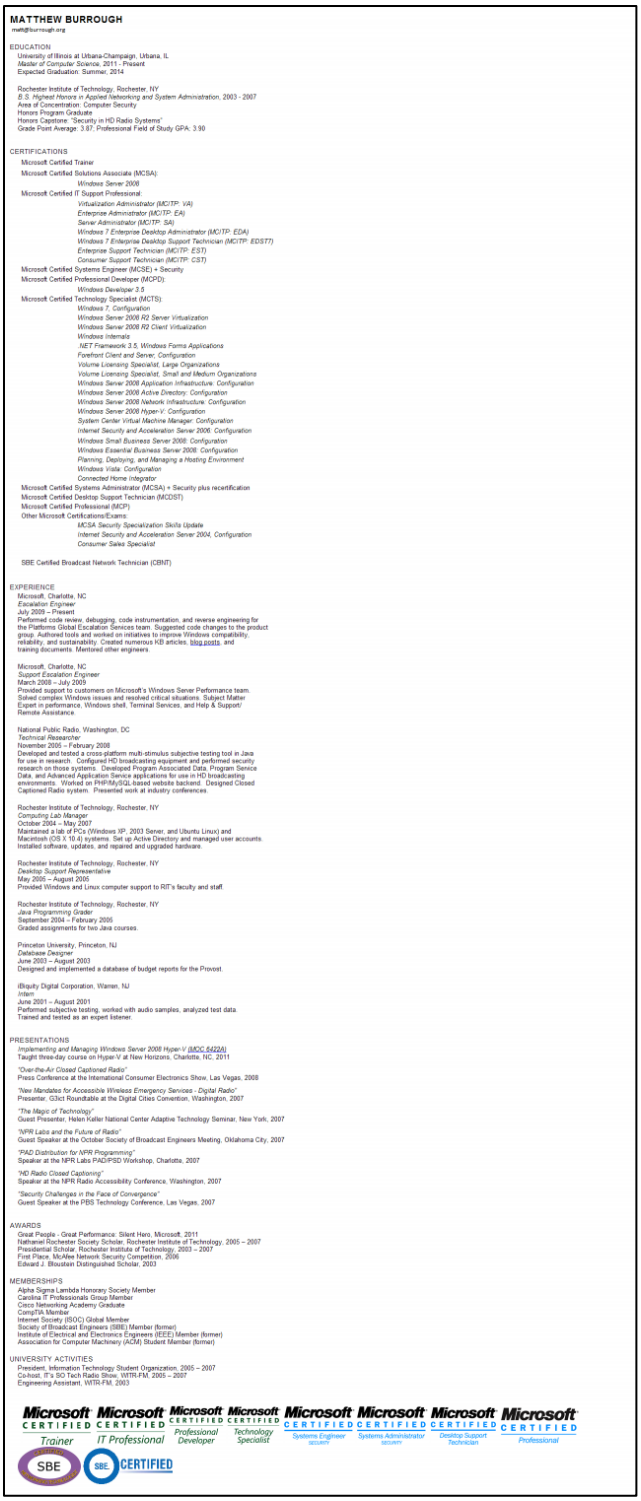
Figure 44: Subjective Assessment of Browser Performance

Appendix A – Screenshots of Testing Webpages

This section contains screenshots of each page used for testing as it appeared during the tests.



NPAPI Test Page



MATTHEW BURROUGH
matt@burrough.org

EDUCATION

University of Illinois at Urbana-Champaign, Urbana, IL
Master of Computer Science, 2011 - Present
Expected Graduation: Summer, 2014

Rochester Institute of Technology, Rochester, NY
B.S. Highest Honors in Applied Networking and System Administration, 2003 - 2007
Area of Concentration: Computer Security
Honors Program Graduate
Honors Capstone: "Security in HD Radio Systems"
Grade Point Average: 3.87; Professional Field of Study GPA: 3.90

CERTIFICATIONS

Microsoft Certified Trainer
Microsoft Certified Solutions Associate (MCSA):
Windows Server 2008
Microsoft Certified IT Support Professional:
Virtualization Administrator (MCITP: VA)
Enterprise Administrator (MCITP: EA)
Server Administrator (MCITP: SA)
Windows 7 Enterprise Desktop Administrator (MCITP: EDA)
Windows 7 Enterprise Desktop Support Technician (MCITP: EDST7)
Enterprise Support Technician (MCITP: EST)
Consumer Support Technician (MCITP: CST)
Microsoft Certified Systems Engineer (MCSE) + Security
Microsoft Certified Professional Developer (MCPD):
Windows Developer 3.5
Microsoft Certified Technology Specialist (MCTS):
Windows 7, Configuration
Windows Server 2008 R2 Server Virtualization
Windows Server 2008 R2 Client Virtualization
Windows Internals
.NET Framework 3.5, Windows Forms Applications
Forefront Client and Server, Configuration
Volume Licensing Specialist, Large Organizations
Volume Licensing Specialist, Small and Medium Organizations
Windows Server 2008 Application Infrastructure, Configuration
Windows Server 2008 Active Directory: Configuration
Windows Server 2008 Network Infrastructure, Configuration
Windows Server 2008 Hyper-V: Configuration
System Center Virtual Machine Manager: Configuration
Internet Security and Acceleration Server 2006: Configuration
Windows Small Business Server 2008: Configuration
Windows Essential Business Server 2008: Configuration
Planning, Deploying, and Managing a Hosting Environment
Windows Vista: Configuration
Connected Home Integrator
Microsoft Certified Systems Administrator (MCSA) + Security plus recertification
Microsoft Certified Desktop Support Technician (MCDST)
Microsoft Certified Professional (MCP)
Other Microsoft Certifications/Exams:
MCSA Security Specialization Skills Update
Internet Security and Acceleration Server 2004, Configuration
Consumer Sales Specialist

SBE Certified Broadcast Network Technician (CBNT)

EXPERIENCE

Microsoft, Charlotte, NC
Escalation Engineer
July 2009 – Present
Performed code review, debugging, code instrumentation, and reverse engineering for the Platforms Global Escalation Services team. Suggested code changes to the product group. Authored tools and worked on initiatives to improve Windows compatibility, reliability, and sustainability. Created numerous KB articles, blog posts, and training documents. Mentored other engineers.

Microsoft, Charlotte, NC
Support Escalation Engineer
March 2008 – July 2009
Provided support to customers on Microsoft's Windows Server Performance team. Solved complex Windows issues and resolved critical situations. Subject Matter Expert in performance, Windows shell, Terminal Services, and Help & Support/ Remote Assistance.

National Public Radio, Washington, DC
Technical Researcher
November 2005 – February 2008
Developed and tested a cross-platform multi-stimulus subjective testing tool in Java for use in research. Configured HD broadcasting equipment and performed security research on those systems. Developed Program Associated Data, Program Service Data, and Advanced Application Service applications for use in HD broadcasting environments. Worked on PHP/MySQL-based website backend. Designed Closed Captioned Radio system. Presented work at industry conferences.

Rochester Institute of Technology, Rochester, NY
Computing Lab Manager
October 2004 – May 2007
Maintained a lab of PCs (Windows XP, 2003 Server, and Ubuntu Linux) and Macintosh (OS X 10.4) systems. Set up Active Directory and managed user accounts. Installed software, updates, and repaired and upgraded hardware.

Rochester Institute of Technology, Rochester, NY
Desktop Support Representative
May 2005 – August 2005
Provided Windows and Linux computer support to RIT's faculty and staff.

Rochester Institute of Technology, Rochester, NY
Java Programming Grader
September 2004 – February 2005
Graded assignments for two Java courses.

Princeton University, Princeton, NJ
Database Designer
June 2003 – August 2003
Designed and implemented a database of budget reports for the Provost.

iBiquity Digital Corporation, Warren, NJ
Intern
June 2001 – August 2001
Performed subjective testing, worked with audio samples, analyzed test data. Trained and tested as an expert listener.

PRESENTATIONS

Implementing and Managing Windows Server 2008 Hyper-V (MOC 6422A)
Taught three-day course on Hyper-V at New Horizons, Charlotte, NC, 2011

"Over-the-Air Closed Captioned Radio"
Press Conference at the International Consumer Electronics Show, Las Vegas, 2008

"New Mandates for Accessible Wireless Emergency Services - Digital Radio"
Presenter, G3ict Roundtable at the Digital Cities Convention, Washington, 2007

"The Magic of Technology"
Guest Presenter, Helen Keller National Center Adaptive Technology Seminar, New York, 2007

"NPR Labs and the Future of Radio"
Guest Speaker at the October Society of Broadcast Engineers Meeting, Oklahoma City, 2007

"PAD Distribution for NPR Programming"
Speaker at the NPR Labs PAD/PSD Workshop, Charlotte, 2007

"HD Radio Closed Captioning"
Speaker at the NPR Radio Accessibility Conference, Washington, 2007

"Security Challenges in the Face of Convergence"
Guest Speaker at the PBS Technology Conference, Las Vegas, 2007

AWARDS

Great People - Great Performance: Silent Hero, Microsoft, 2011
Nathaniel Rochester Society Scholar, Rochester Institute of Technology, 2005 – 2007
Presidential Scholar, Rochester Institute of Technology, 2003 – 2007
First Place, McAfee Network Security Competition, 2006
Edward J. Bloustein Distinguished Scholar, 2003

MEMBERSHIPS

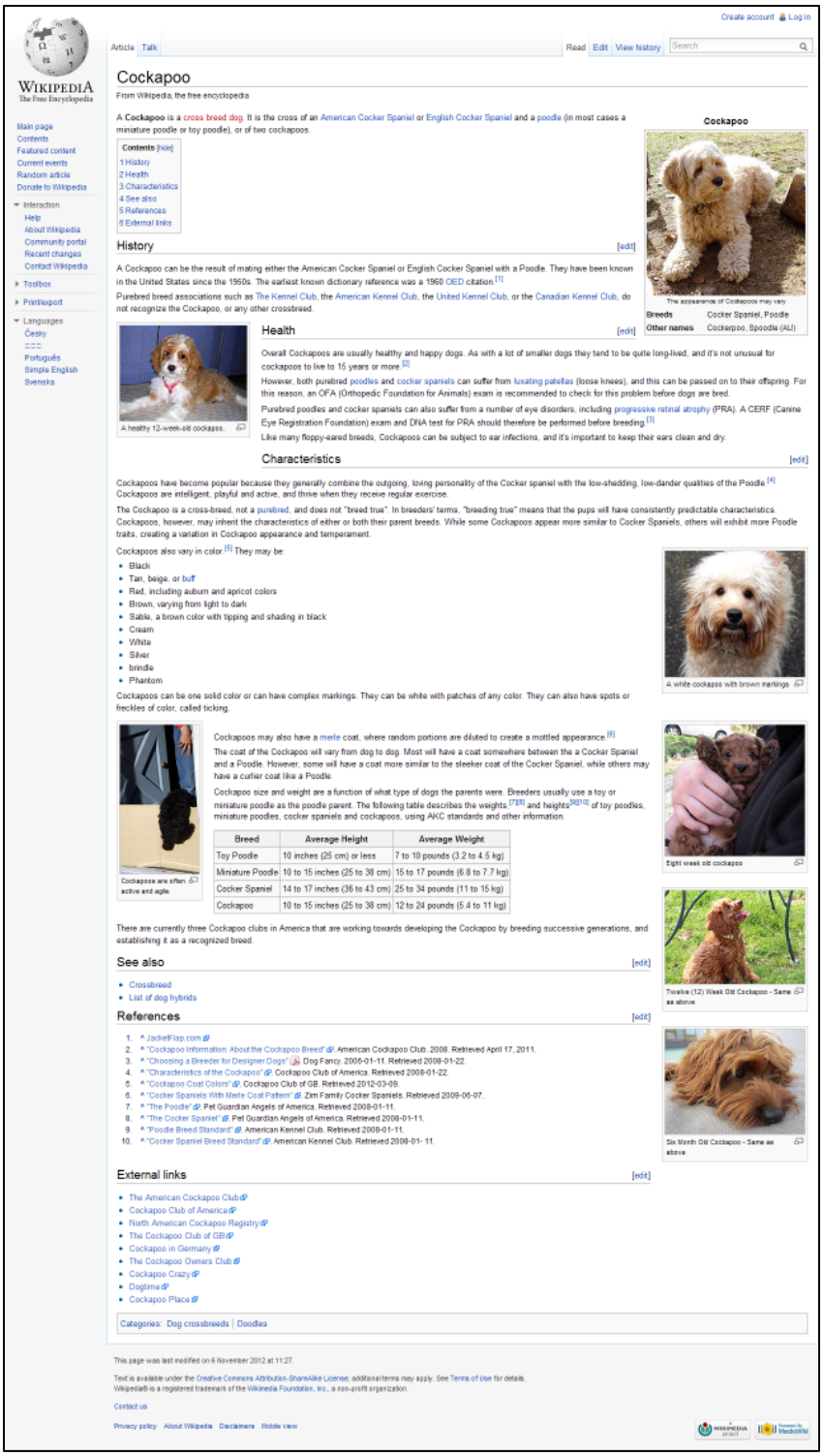
Alpha Sigma Lambda Honorary Society Member
Carolina IT Professionals Group Member
Cisco Networking Academy Graduate
CompTIA Member
Internet Society (ISOC) Global Member
Society of Broadcast Engineers (SBE) Member (former)
Institute of Electrical and Electronics Engineers (IEEE) Member (former)
Association for Computer Machinery (ACM) Student Member (former)

UNIVERSITY ACTIVITIES

President, Information Technology Student Organization, 2005 – 2007
Co-host, IT's SO Tech Radio Show, WITR-FM, 2005 – 2007
Engineering Assistant, WITR-FM, 2003

Microsoft CERTIFIED Trainer
Microsoft CERTIFIED IT Professional
Microsoft CERTIFIED Professional Developer
Microsoft CERTIFIED Technology Specialist
Microsoft CERTIFIED Systems Engineer security
Microsoft CERTIFIED Systems Administrator security
Microsoft CERTIFIED Desktop Support Technician
Microsoft CERTIFIED Professional
SBE CERTIFIED

Burrough.org Page



Create account Log in

WIKIPEDIA
The Free Encyclopedia

Article Talk Read Edit View history Search

Cockapoo

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export
Languages
Česky
Português
Simple English
Svenska

A **Cockapoo** is a cross breed dog. It is the cross of an American Cocker Spaniel or English Cocker Spaniel and a poodle (in most cases a miniature poodle or toy poodle), or of two cockapoos.

Contents [hide]
1 History
2 Health
3 Characteristics
4 See also
5 References
6 External links

Cockapoo

The appearance of Cockapoos may vary

Breeds	Cocker Spaniel, Poodle
Other names	Cockerpoo, Spoodle (AU)

History [edit]

A Cockapoo can be the result of mating either the American Cocker Spaniel or English Cocker Spaniel with a Poodle. They have been known in the United States since the 1950s. The earliest known dictionary reference was a 1960 OED citation.[1]

Purebred breed associations such as The Kennel Club, the American Kennel Club, the United Kennel Club, or the Canadian Kennel Club, do not recognize the Cockapoo, or any other crossbreed.

A healthy 12-week-old cockapoo.

Health [edit]

Overall Cockapoos are usually healthy and happy dogs. As with a lot of smaller dogs they tend to be quite long-lived, and it's not unusual for cockapoos to live to 15 years or more.[2]

However, both purebred poodles and cocker spaniels can suffer from luxating patellas (loose knees), and this can be passed on to their offspring. For this reason, an OFA (Orthopedic Foundation for Animals) exam is recommended to check for this problem before dogs are bred.

Purebred poodles and cocker spaniels can also suffer from a number of eye disorders, including progressive retinal atrophy (PRA). A CERF (Canine Eye Registration Foundation) exam and DNA test for PRA should therefore be performed before breeding.[3]

Like many floppy-eared breeds, Cockapoos can be subject to ear infections, and it's important to keep their ears clean and dry.

Characteristics [edit]

Cockapoos have become popular because they generally combine the outgoing, loving personality of the Cocker spaniel with the low-shedding, low-dander qualities of the Poodle.[4] Cockapoos are intelligent, playful and active, and thrive when they receive regular exercise.

The Cockapoo is a cross-breed, not a purebred, and does not "breed true". In breeders' terms, "breeding true" means that the pups will have consistently predictable characteristics. Cockapoos, however, may inherit the characteristics of either or both their parent breeds. While some Cockapoos appear more similar to Cocker Spaniels, others will exhibit more Poodle traits, creating a variation in Cockapoo appearance and temperament.

Cockapoos also vary in color.[5] They may be:

- Black
- Tan, beige, or buff
- Red, including auburn and apricot colors
- Brown, varying from light to dark
- Sable, a brown color with tipping and shading in black
- Cream
- White
- Silver
- brindle
- Phantom

A white cockapoo with brown markings

Cockapoos can be one solid color or can have complex markings. They can be white with patches of any color. They can also have spots or freckles of color, called ticking.

Cockapoos are often active and agile.

Cockapoos may also have a merle coat, where random portions are diluted to create a mottled appearance.[6]

The coat of the Cockapoo will vary from dog to dog. Most will have a coat somewhere between the a Cocker Spaniel and a Poodle. However, some will have a coat more similar to the sleeker coat of the Cocker Spaniel, while others may have a curlier coat like a Poodle.

Cockapoo size and weight are a function of what type of dogs the parents were. Breeders usually use a toy or miniature poodle as the poodle parent. The following table describes the weights,[7][8] and heights[9][10] of toy poodles, miniature poodles, cocker spaniels and cockapoos, using AKC standards and other information.

Breed	Average Height	Average Weight
Toy Poodle	10 inches (25 cm) or less	7 to 10 pounds (3.2 to 4.5 kg)
Miniature Poodle	10 to 15 inches (25 to 38 cm)	15 to 17 pounds (6.8 to 7.7 kg)
Cocker Spaniel	14 to 17 inches (36 to 43 cm)	25 to 34 pounds (11 to 15 kg)
Cockapoo	10 to 15 inches (25 to 38 cm)	12 to 24 pounds (5.4 to 11 kg)

Eight week old cockapoo

Twelve (12) Week Old Cockapoo - Same as above

Six Month Old Cockapoo - Same as above

There are currently three Cockapoo clubs in America that are working towards developing the Cockapoo by breeding successive generations, and establishing it as a recognized breed.

See also [edit]

- Crossbreed
- List of dog hybrids

References [edit]

1. ^ JacketFlap.com
2. ^ "Cockapoo Information: About the Cockapoo Breed". American Cockapoo Club. 2008. Retrieved April 17, 2011.
3. ^ "Choosing a Breeder for Designer Dogs". Dog Fancy. 2006-01-11. Retrieved 2008-01-22.
4. ^ "Characteristics of the Cockapoo". Cockapoo Club of America. Retrieved 2008-01-22.
5. ^ "Cockapoo Coat Colors". Cockapoo Club of GB. Retrieved 2012-03-09.
6. ^ "Cocker Spaniels With Merle Coat Pattern". Zim Family Cocker Spaniels. Retrieved 2009-06-07.
7. ^ "The Poodle". Pet Guardian Angels of America. Retrieved 2008-01-11.
8. ^ "The Cocker Spaniel". Pet Guardian Angels of America. Retrieved 2008-01-11.
9. ^ "Poodle Breed Standard". American Kennel Club. Retrieved 2008-01-11.
10. ^ "Cocker Spaniel Breed Standard". American Kennel Club. Retrieved 2008-01- 11.

External links [edit]

- The American Cockapoo Club
- Cockapoo Club of America
- North American Cockapoo Registry
- The Cockapoo Club of GB
- Cockapoo in Germany
- The Cockapoo Owners Club
- Cockapoo Crazy
- Dogtime
- Cockapoo Place

Categories: Dog crossbreeds | Doodles

Wikipedia Cockapoo Page

References

Aggarwal, G., Bursztein, E., Jackson, C., & Boneh, D. (2010). An analysis of private browsing modes in modern browsers. *Proc. of 19th Usenix Security Symposium.*

Barnabe, J. (2007, May 28). *Urlclassifier2.sqlite*. Retrieved November 21, 2012, from mozillaZine: http://kb.mozillazine.org/Urlclassifier2.sqlite

Ben. (2005, August 4). *Building an extension.* Retrieved November 20, 2012, from Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Building_an_Extension

cor-el. (2012, May 8). *Are extentions disabled in private browsing in firefox???* Retrieved November 20, 2012, from Mozilla Support: https://support.mozilla.org/en-US/questions/926986

Dunn, N. (2011, May 3). *Issue 81400: Extension API causing crash. chrome.windows.create causes crash if only a single app mode window is open.* Retrieved November 18, 2012, from Chromium Issues: http://code.google.com/p/chromium/issues/detail?id=81400

Google. (2012). *NPAPI Plugins.* Retrieved November 20, 2012, from Google Chrome Extensions: http://developer.chrome.com/extensions/npapi.html

Gould, S. (2008). *See inside index.dat files.* Retrieved November 21, 2012, from stevengould.org: http://www.stevengould.org/index.php?option=com_content&task=view&id=47&Itemid=88

Jones, K. (2010, May 23). *Writing a Managed Internet Explorer Extension: Part 1.* Retrieved November 18, 2012, from Random Agile Thoughts: http://msmvps.com/blogs/vcsjones/archive/2010/05/23/writing-a-managed-internet-explorer-extension-part-1.aspx

Jung, E. (2007, January 30). *On page load.* Retrieved November 20, 2012, from Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Code_snippets/On_page_load

Kornblum, J., & Metz, J. (2007, March 10). *Internet Explorer History File Format*. Retrieved November 21, 2012, from Forensics Wiki: http://www.forensicswiki.org/wiki/Internet_Explorer_History_File_Format#REDR_Records

Mahendrakar, A., Irving, J., & Patel, S. (2011). Forensic analysis of private browsing artifacts. *Innovations in Information Technology (IIT)* (pp. 197-202). IEEE.

Microsoft. (2012, February 13). *Windows Error Reporting: Getting Started*. Retrieved November 24, 2012, from Dev Center - Hardware: http://msdn.microsoft.com/en-us/library/windows/hardware/gg487440.aspx

Microsoft Corporation. (2012, October 16). *Collecting User-Mode Dumps (Windows).* Retrieved November 19, 2012, from Windows Dev Center - Desktop: http://msdn.microsoft.com/en-us/library/windows/desktop/bb787181%28v=vs.85%29.aspx

Mills, C. (2012, February 9). *64-bit Opera, and out-of-process plug-ins.* Retrieved November 20, 2012, from DEV.Opera: http://dev.opera.com/articles/view/64-bit-opera-and-out-of-process-plug-ins/

Mozilla. (2007, September 21). *NpRuntime.* Retrieved November 20, 2012, from Mozilla Samples: http://mxr.mozilla.org/firefox/source/modules/plugin/samples/npruntime/

Opera Software. (2008, January 18). *The Opera plug-in interface.* Retrieved November 20, 2012, from DEV.Opera: http://dev.opera.com/articles/view/the-opera-plug-in-interface/#security

Russinovich, M. (2012, May 14). *Strings v2.5.* Retrieved November 24, 2012, from Windows Sysinternals: http://technet.microsoft.com/en-us/sysinternals/bb897439.aspx

Sofer, N. (2012). *Web Browser Tools Package*. Retrieved November 21, 2012, from NirSoft: http://www.nirsoft.net/web_browser_tools.html

Soghoian, C. (2010). Private Browsing Modes Do Not Deliver Real Privacy. *IAB Internet Privacy Workshop.* Boston.

Vel, S. (2008, May 23). *Troubleshooting Agentless Exception Monitoring and Desktop Error.* Retrieved November 19, 2012, from TechNet: http://blogs.technet.com/cfs-filesystemfile.ashx/__key/communityserver-components-postattachments/00-03-06-00-25/Troubleshooting-AEM-and-DEM.docx

Verdi, M. (2010, June 30). *Send plugin crash reports to help Mozilla improve Firefox.* Retrieved November 20, 2012, from Mozilla Support: http://support.mozilla.org/en-US/kb/send-plugin-crash-reports-help-improve-firefox#w_what-information-is-sent-in-a-crash-report

West, M. (2012). *WebNavigation Tech Demo.* Retrieved November 18, 2012, from Google Chrome Extensions - Sample Extensions: http://developer.chrome.com/extensions/examples/api/webNavigation/basic.zip

Wyman, A. (2010, July 14). *Plugin-container and out-of-process plugins.* Retrieved November 20, 2012, from mozillaZine: http://kb.mozillazine.org/Plugin-container_and_out-of-process_plugins

Zeigler, A. (2008, July 28). *IE8 and Reliability.* Retrieved November 18, 2012, from IEBlog: http://blogs.msdn.com/b/ie/archive/2008/07/28/ie8-and-reliability.aspx