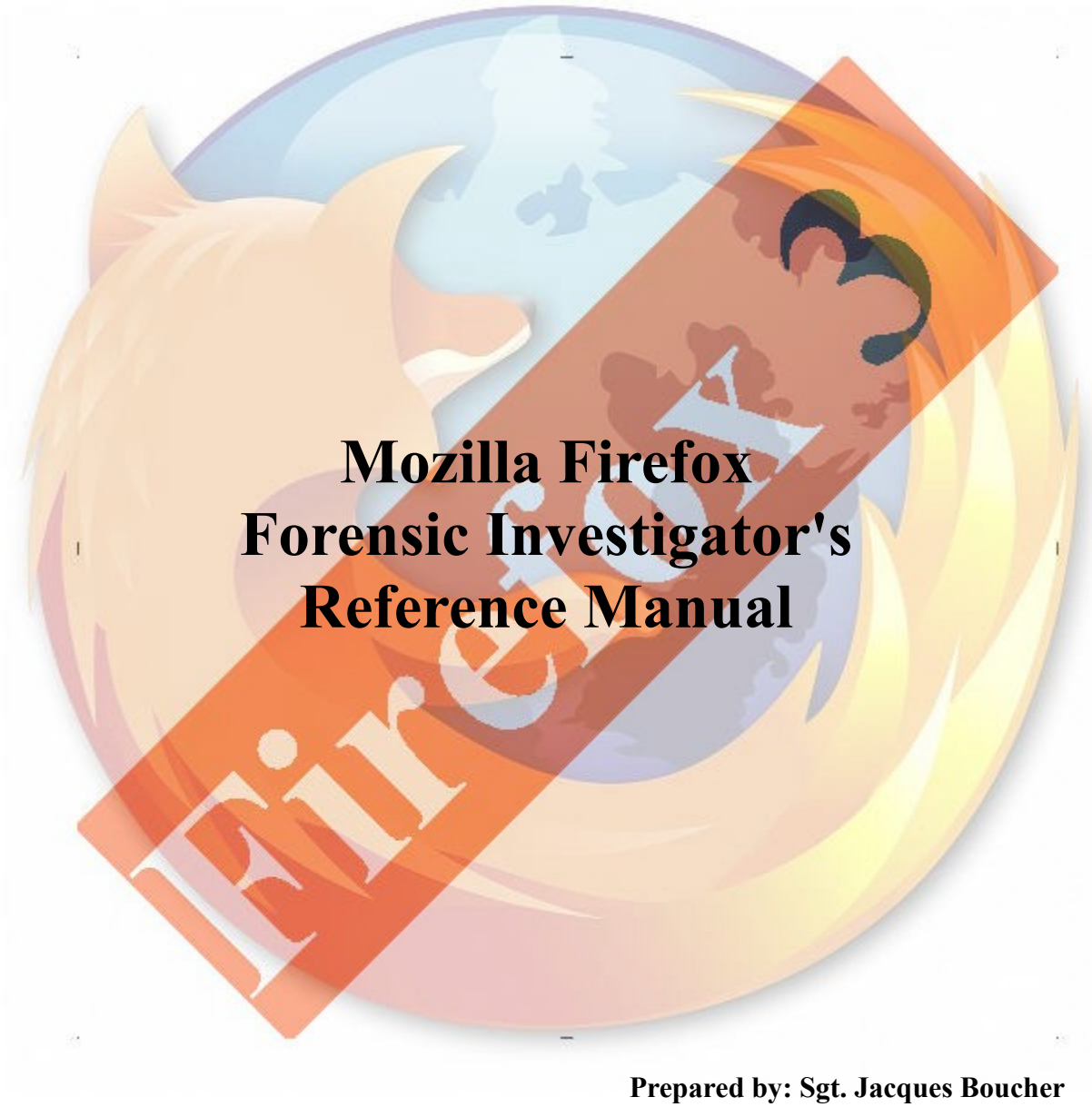


Mozilla Firefox Forensic Investigator's Reference Manual

Prepared by: Sgt. Jacques Boucher
jboucher@cpc.gc.ca
Revision Date: March 26, 2009
Canadian Police College
Ottawa, Ontario, Canada

Canada

Foreword

This reference manual contains an alphabetical index at the end thus allowing an investigator to quickly locate the information he/she requires. It is designed to be used as a teaching aid for the browser session on the Internet Evidence Analysis Course taught at the Canadian Police College, and also designed to be a reference manual out in the field after having completed the course. It is unlikely that an analyst would have to exhaust all aspects of analysis covered in this manual. However given the uniqueness of each case this manual will hopefully address the varied situations encountered by an analyst.

As with everything else in the field of computers and computer forensics, the only constant is change. This manual was created using Firefox 3.0.3 on a Windows XP SP3 system as the test environment. Should you identify any consistencies or inconsistencies with other versions of Firefox or other operating systems (i.e. Vista) please forward them so that they may be tested and added to this reference manual. Every effort was made to ensure accuracy in the content of this manual. However should you note any errors in this manual please advise us so that we may validate your findings and make necessary corrections.

The beauty of Firefox is that it is open source. Because of this you will find extensive documentation on Mozilla's website (see Links section in this reference manual for useful links). This manual was possible in great part thanks to the efforts of the open source community and the excellent documentation maintained online. You are encouraged to check out these resources for more indepth details on the topics covered herein in addition to many other topics not covered in this manual.

Although many things have changed with Firefox 3, some remain the same as with Firefox 2. Therefore some sections may be verbatim out of the Firefox 2 guide (once validated to be applicable to version 3) possibly with additional information either as a result of further testing or due to some added features in version 3.

Table of Content

Additional TO DO upon completion of above – table of content and alphabetical index

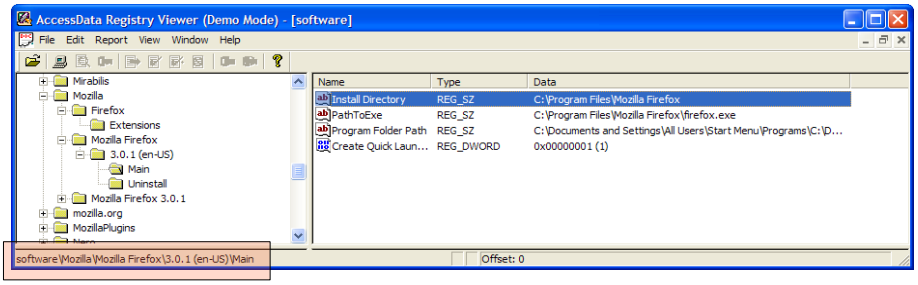
Chapter 1

Install & Version Information, Updates

Knowing what version of Firefox the user has on their system and the updates/patches applied can be of value if you are attempting to rule out browser malware found on the system as being responsible for browser artifacts.

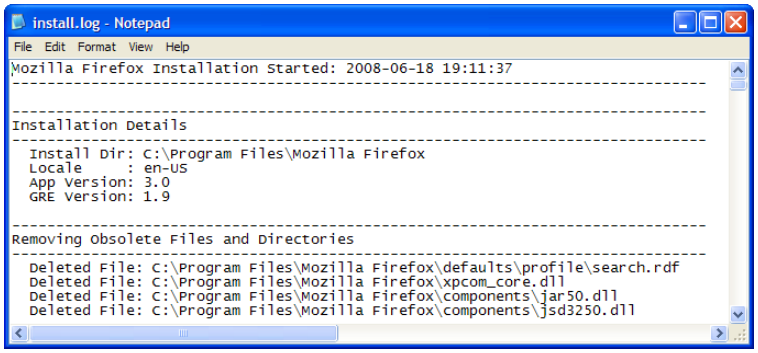
Install Information

Firefox Menu	Nil
Install Log	C:\Program Files\Mozilla Firefox\install.log (default path)
Associated Registry Key	HKLM\software\Mozilla\Mozilla Firefox\3.0.1 (en-US)\Main ● *This key contains the install path for Firefox, which is where you will find the install log.* ● *Note that the version # within the registry key path will vary.*



Note that some who were running Firefox 2 may have chosen to install Firefox 3 (beta version or otherwise) in a new folder in order to retain version 2 pending being satisfied with the new version of Firefox. If that is the case, you may see a folder for a previous version of Firefox that may or may not contain an install.log file from that version of Firefox.

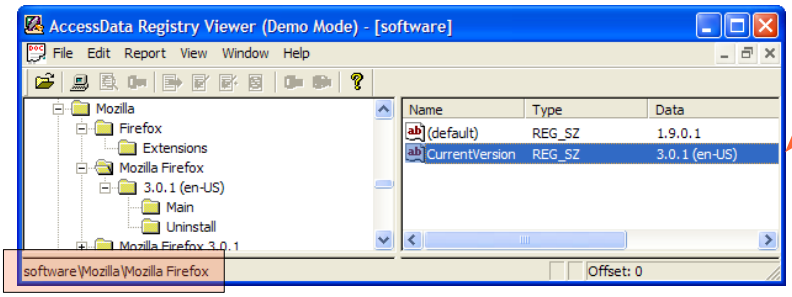
The install.log file is a plain text file that contains information about the install (date/time of install and version of Firefox, files deleted, folders created, files created, etc). Whether or not you have a need to examine it will depend on the circumstances of your analysis.

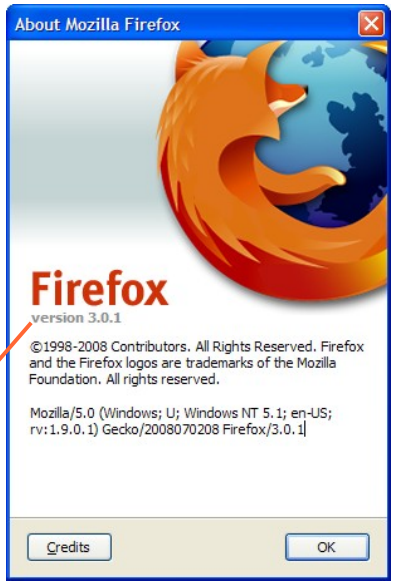


To the right you see an excerpt of an install.log file from a Firefox install.
The Install Directory noted in the file should match where the install.log file was found.
The Application Version could differ slightly from the installed version due to an upgrade after install.
GRE most likely stands for Gecko Runtime Environment or Gecko Rendering Engine (both acronyms have been found online, although neither found referencing the install.log file specifically).

Version Information

Firefox Menu	Help, About Mozilla Firefox
Live Registry Entry	HKLM\Software\Mozilla\Mozilla Firefox\Current Version
Registry File	c:\windows\system32\config\software

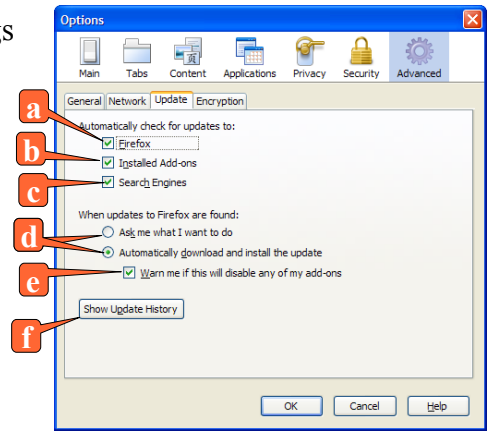




Updates

Firefox Menu	Tools, Options – **Advanced** tab - **Update** tab
about:config Entries	a app.update.enabled (default -True) b extensions.update.enabled (default – True) c browser.search.update (default – True) d app.update.auto (default – True) e app.update.mode (default – 1)
Path of Config File	C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\prefs.js (settings for a specific user if set will be found in user.js)
Path of Update File	f C:\Documents and Settings\{user}\Local Settings\Application Data\Mozilla\Firefox\Mozilla Firefox\updates.xml

There are several configuration options that allow a user to configure Firefox' automatic update feature. If the default settings are selected, you will not see them in the prefs.js file. In most cases you likely will not have to worry about this. However it is being provided in the event you do require to establish this fact.



- a "This preference determines whether the application will auto-update." Here the "application" refers to Firefox itself.
  - **true** - Auto-update the application. (Default)
  - **false** - Do not auto-update.

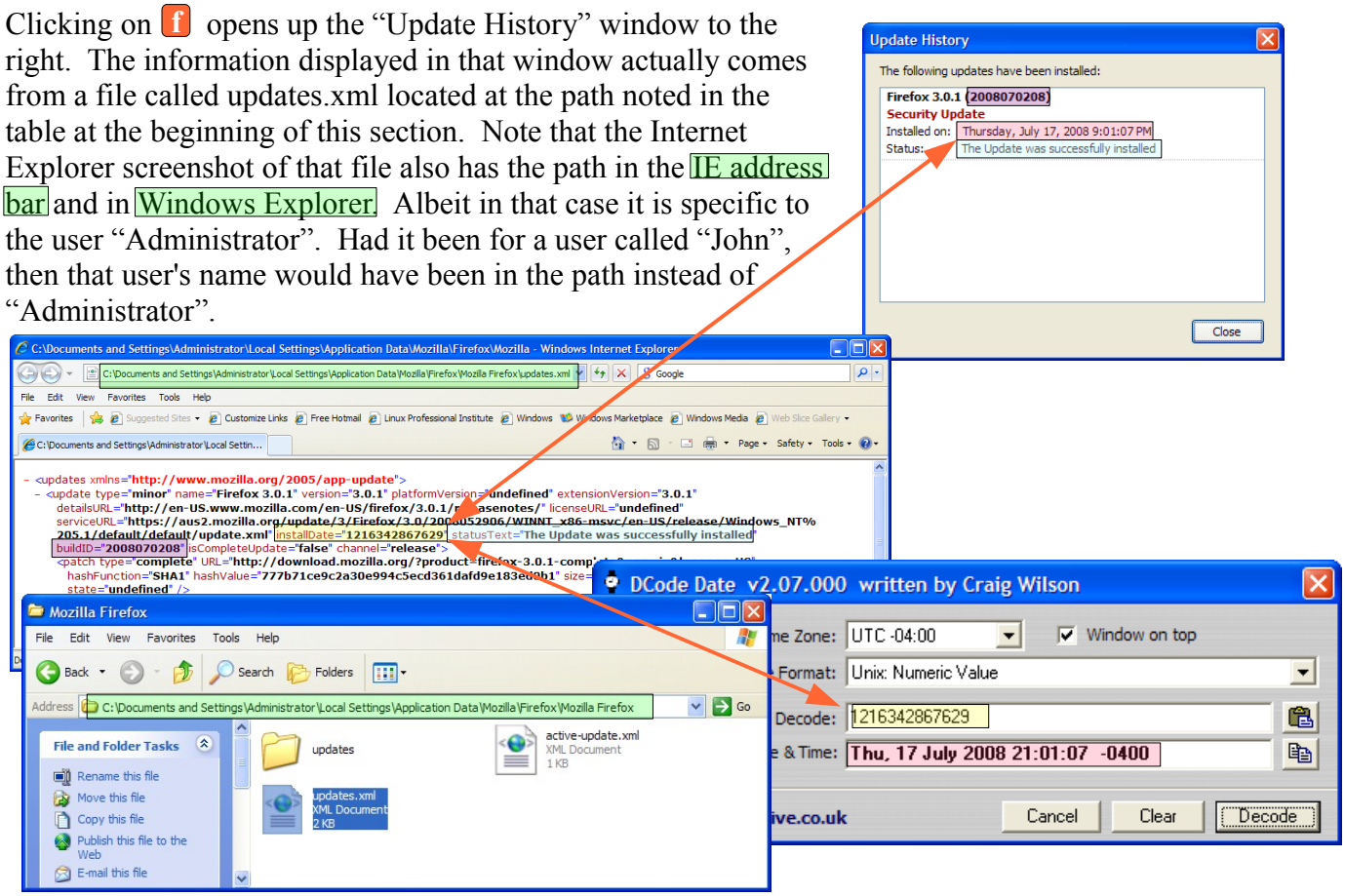
b. This preference determines whether the installed extensions will auto-update.
   - **True** (default): Allow checking for updates
   - **False**: Opposite of the above
     - Can be overridden on a per-extension basis by setting **extensions.{GUID}.update.enabled"**

c. This preference determines whether the search plugins will auto-update (i.e plugins used in Firefiox' search bar).
   - **"True (default):** Automatically check for updates to search plugins.
   - **False:** Opposite of the above."

d. This option works hand in hand with app.update.enabled found in a.
   - **True (default):** Automatic update of the Firefox application (providing app.update.enabled is also set to True). Which means "Automatically download and install the update" radio button is selected.
   - **False:** Prompts the user to download an update to the Firefox application. Which means "Ask me what I want to do." radio button is selected.

e. "This preference determines which updates can be downloaded in background, and which ones requires an user prompt to be applied.
   - **0** - download all updates without any prompt
   - **1** (default) - download all update only if there are no incompatibilities with enabled extensions, prompt otherwise
   - **2** - download minor updates only, prompt for major updates, regardless of whether or not all enabled extensions are compatible.
     - Caveats - app.update.enabled and app.update.auto must be set to true for this preference to take effect."
   - Note that a value of 1 is when the box is checked. A value of 0 is when it is not checked. A value of 2 can be manually entered via about:config. In the GUI the box will still show as checked same as for a value of 1.

Clicking on f opens up the "Update History" window to the right. The information displayed in that window actually comes from a file called updates.xml located at the path noted in the table at the beginning of this section. Note that the Internet Explorer screenshot of that file also has the path in the IE address bar and in Windows Explorer. Albeit in that case it is specific to the user "Administrator". Had it been for a user called "John", then that user's name would have been in the path instead of "Administrator".



Notice that the "Installed on" date corresponds to the Unix Numeric Value in updates.xml. This is verified by using DCode and convering that Unix Numeric Value to a date format. That decoded date matches exactly to the second the date displayed in the Firefox Update History window.

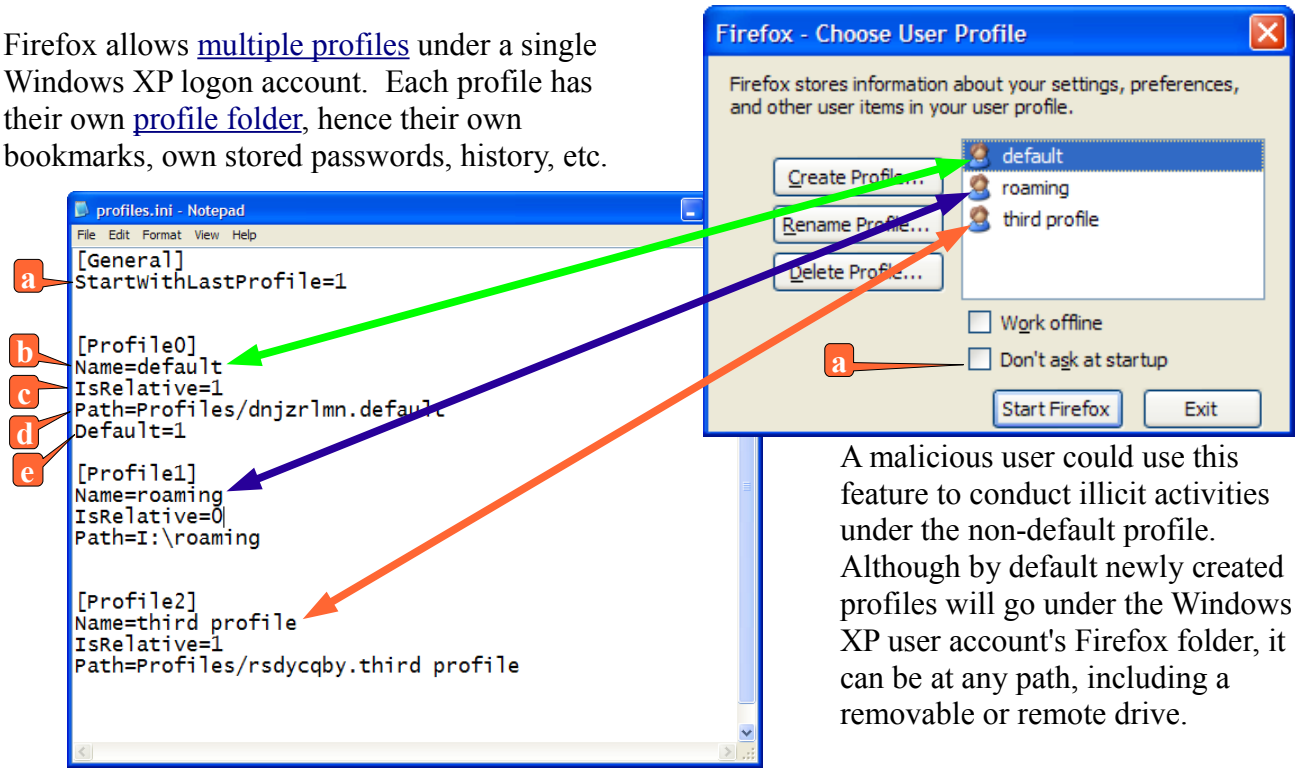
Chapter 2

User Profile

profiles.ini

How To Access	Access from Windows START, Run. Then type firefox -p, or firefox -profilemanager
about:config Entries	N/A
Path of Config File	C\Documents and Settings\{user}\Application Data\Mozilla\Firefox\profiles.ini Default profile is [Profile0]

Firefox allows multiple profiles under a single Windows XP logon account. Each profile has their own profile folder, hence their own bookmarks, own stored passwords, history, etc.



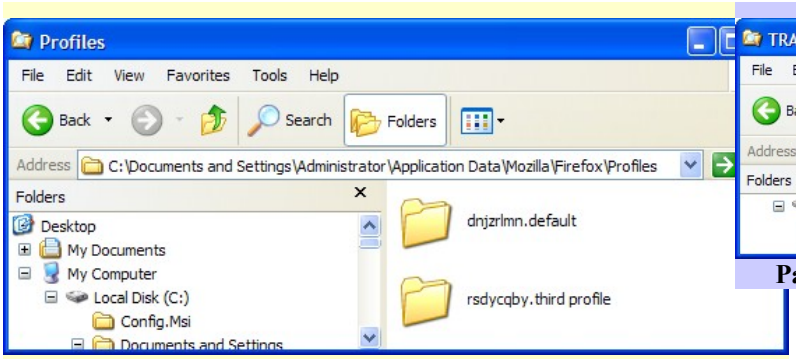
A malicious user could use this feature to conduct illicit activities under the non-default profile. Although by default newly created profiles will go under the Windows XP user account's Firefox folder, it can be at any path, including a removable or remote drive.

a StartWithLastProfile can have a value of 0 or 1.
- If it's 0, you will see the menu above displayed prompting you to select a profile to use for this session of Firefox (the menu is displayed every time Firefox is started).
- If it's set to 1, this means to use the Profile in this profiles.ini file which has the entry Default=1. In that case it will start Firefox with that profile without prompting. A user can still start Firefox from the Run menu with the switch -p or -profilemanager to access the above profile selection menu to select a non default profile.

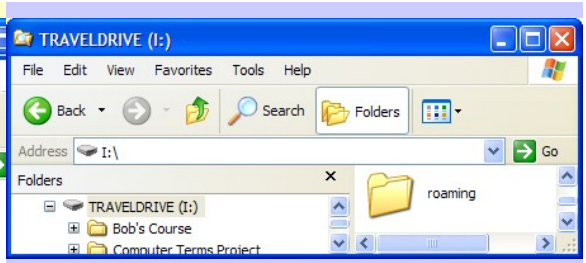
b This is the profile name which matches what you see in the menu and by default will be appended to the name of the profile folder created by Firefox in the default location.

c. IsRelative tells you if the path for that profile is a relative path (relative to the default Firefox application data folder) or if it's an absolute path.
   - A value of 1 tells you the profile folder is under the default location, so a relative path.
   - A value of 0 tells you the path for the profile folder is an absolute path, so not at the default location.

d. Path tells you where to find the profile folder for this profile. A relative path will start with Profiles/ (see Profile0 and Profile2) whereas an absolute path will be just that, complete with the drive letter (see Profile1). This absolute path can be on the same drive, or on a different drive, **even a removable thumb drive as is the case for profile "roaming"...**



Path within Windows Explorer of [Profile0] and [Profile2]



Path within Windows Explorer of [Profile1]

e. If a profile has this entry, Default=1, it will be the default profile that will be used by Firefox if StartWithLastProfile is set to 1 – so no selection menu would be displayed even if multiple profiles existed. "Default=1" denotes the last selected profile. Otherwise it will be ignored and the multiple profiles will be displayed.

IMPORTANT: Examine Profile1 a bit closer. Note that it's stored on a removable drive (I:). You'd want to go in the registry and restore points to try and identify a what device was assigned that drive letter. Even if the thumb drive is not present, Firefox will display that profile in the menu because it is in the profiles.ini file for that Windows XP user account. However if a user selects it while the thumb drive is not connected it will produce an error stating that it's in use. However in the profiles.ini setting above the alternate profiles wouldn't even be displayed as it would start to the default profile. A malicious user could simply plug in the thumb drive and start Firefox with the -p option to access the stored profile.

Multiple Profiles

In a multi-profile configuration Firefox will automatically set "Default=1" to whichever profile you start up. In the above example Profile0 (Default) is currently the default. If you start Firefox via the profile manager and select an alternate profile, Firefox will edit profiles.ini and put the line "Default=1" in whichever profile is selected. Which means if a user has Firefox to automatically start with the default profile, the act of starting Firefox with an alternate profile will make that profile become the default one. In order to prevent Firefox from changing the default to whichever profile was last used, a person can make profiles.ini read-only. By doing that, Firefox will not be able to change

which profile has "Default=1". Firefox does not make profiles.ini read-only. If you find this file read-only, clearly a user has taken extra steps to ensure that the default profile does not change even if the user starts Firefox using a different profile.

Deleting profiles.ini

If a user deletes the profiles.ini file, upon restart of Firefox it will create a new profiles.ini file as well as a new profile. Thus the old profiles will no longer be in the menu (but will still reside on the disk thus a user could easily manually edit profiles.ini and add those previous profiles to the list). **The old profile will still contain all the browser artifacts (bookmarks, history, cookies, downloads, etc).**

Overriding Profile Manager

A user can explicitly select a profile at startup, overriding the profile menu regardless if "Don't check at startup" was selected or not. This is accomplished from the Start, Run menu by typing firefox.exe -p "profile name". Note that profile names are case sensitive. Typing firefox.exe -p "work" will not start Firefox with the profile "Work". If the profile does not exist, it will simpy present the user with the profile menu. A profile can be renamed changing what the GUI displays as a listed profile. However the folder name will not change. The new name will point to the same folder bearing the original name.

During an analysis, providing the additional profiles were created at the default location they would be readily discovered and analyzed. However if one of the profiles were stored on a removable drive, **the analyst would never become aware of this fact** without examining profiles.ini. And even if the new profiles were stored on the same drive it could demonstrate a degree of intent if a profile with illicit activity was hidden by having Firefox start with a default profile as in the example we examined herein.

Location of Cache Files

All profiles that are relative will have their cache under "C:\Documents and Settings\{user}**Local Settings**\Application Data\Mozilla\Firefox\Profiles\{ff profile}". This is not to be confused with the remainder of the profile information found at "C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{ff profile}" (note we do not have "Local Settings" as part of the profile path).

If the profile is at an absolute path (i.e. stored somewhere other than under the "Profiles" folder noted in the previous paragraph), the cached files will be located in a folder called "cache" found under the profile folder.

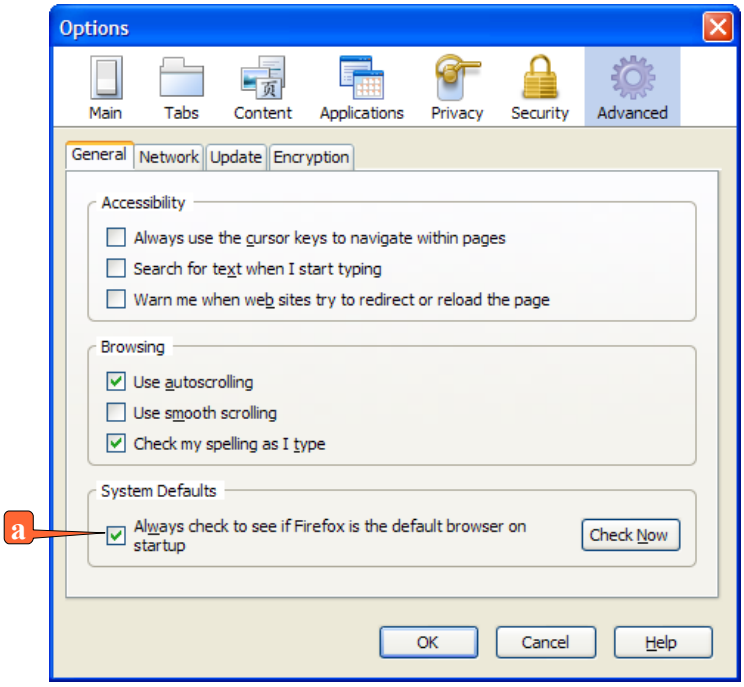
Analyzing cache is covered in a later chapter.

Default Browser

Most browsers are configured by default to check at startup to see if that browser is the default browser. Up until summer of 2008, on a Windows XP system it stored this information in the software hive. However at some point that seems to have changed and XP is now in keeping with MS Vista. Browser preference is now tracked at the individual user level in that user's NTUSER.DAT. A quick test of having a different default brower for the user vs for the machine and then deleting the registry entry for the user's default browser & logging off and back on indeed changed the browser displayed on that user's start menu to that of the default browser at the system level.

Firefox Menu	Nil
about:config Entries	• browser.shell.checkDefaultBrowser (default - true)
Files	• {Firefox profile folder}\prefs.js • c:\documents and settings\{user}\NTUSER.DAT • c:\windows\system32\config\software

Associated Registry Key	• HKCU\Software\Clients\StartMenuInternet
	  ○ Default
	• HKLM\Software\Clients\StartMenuInternet
	  ○ Default



Chapter 3

Application Settings

Firefox 3 shares many commonalities with Firefox 2 when it comes to storing its preferences/application settings. They may contain more settings given the added functionalities. But they are otherwise the same. Per site preferences is something else that Firefox 3 now tracks.

Firefox Menu	• Type "about:config" in the address bar (for a only)
Path of Relevant Files	a C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\prefs.js (settings for a specific user if set will be found in user.js) b C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profile\{profile name}\content-prefs.sqlite c C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profile\{profile name}\permissions.sqlite

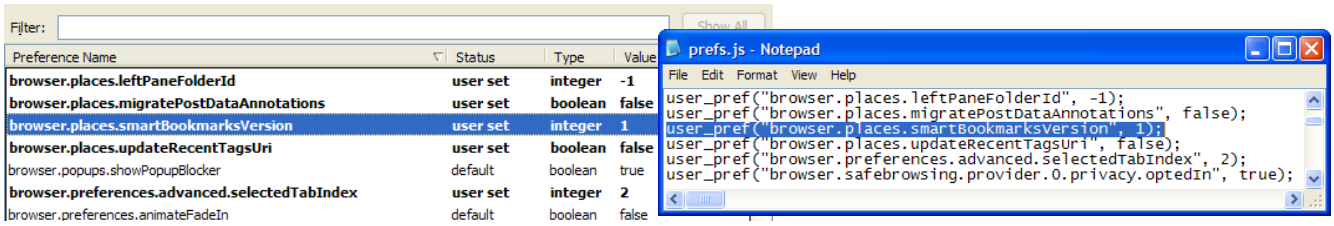
about:config

a Within Firefox you can change settings from the regular menu (i.e. Tools, Options). You can also change settings via "about:config". The difference being that within about:config you can change values of settings to options other than what the GUI will allow. You can also create new preferences that do not exist by default. Within here you may also find settings for plug-ins.

prefs.js and user.js

The prefs.js file located in the user's profile folder contains preferences (i.e. Settings) for Firefox. However it will only contain non-default settings. In other words, default settings are not saved in prefs.js. Which means if you are looking for a particular setting and do not find it in prefs.js it means that the default setting is used for that preference.

As well when viewing these preferences from within Firefox using about:config URL, default preferences will be in regular text whereas any non-default settings will show up in bold. Only the ones that you see in bold (non-default settings) will be written to the prefs.js file (see screenshots below comparing snip from about:config and content of prefs.js).



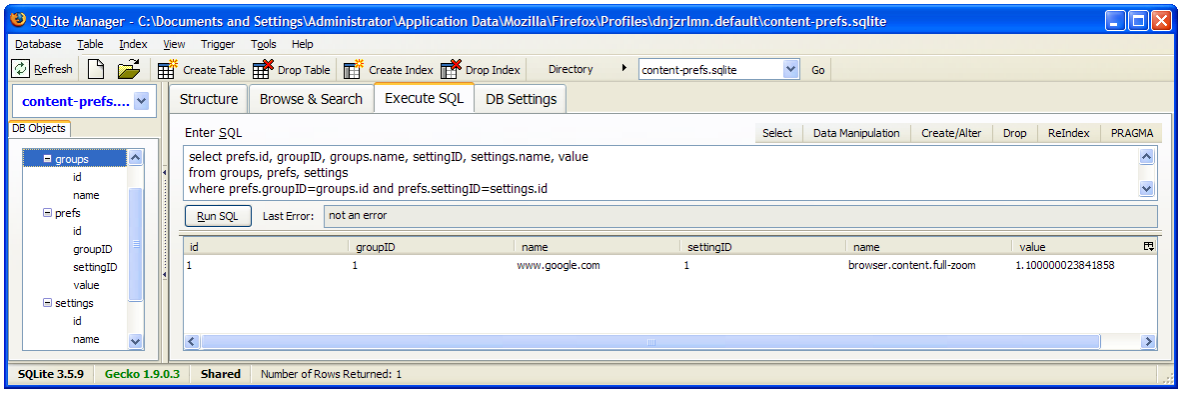
The file user.js is a user specific preference file. It overrides the settings in prefs.js. So if in prefs.js you do not find an entry for popup blocking, the default setting is assumed. However in examining user.js you find an entry for popup blocking. The settings in user.js will take precedence over settings in prefs.js.

One possible value for this would be in a corporate environment where they may have a standard preference template for all employees that gets rolled out with any standard install. But for certain employees who require different settings the system administrator simply applies those settings in that user's user.js file rather than editing the prefs.js. This way if corporate policy changes and a new prefs.js gets rolled out it will not interfere with that particular user's specific settings required for their job.

Why this is done is not as important as the fact that it can be done. Always check the content of user.js as well. In a home or single use environment it is highly unlikely that a person would bother with user.js. But the fact that they can is reason enough to examine it.

Per Site Preferences

Firefox allows you to set preferences on a per site basis. Those settings are saved in content-prefs.sqlite (see appendix "I" for database schema). The settings could possibly include any of the following attributes: text zoom, character encoding, page style, language preferences. Other attributes specific to a site could potentially be stored at this location as well. From a forensic perspective, most of this would likely be of little or no value. With the exception of the URI of a web page for which preferences were set. This is of great value to us because it shows knowledge of a site/web page as a user would have visited that page and set some preferences while viewing that page. What makes it even more valuable is that **these entries are not cleared when a user chooses to "Clear Private Data...".** These settings are only cleared if a user resets the settings back to default for that particular web page (or they manually remove the entries from the database using an add-on or an application capable of manipulating sqlite files). **Which means that for most users this information will remain for quite some time as few are likely to reset their preferences for a particular web page, or manually edit this database file.** It does not however provide any date/time information. The date/time stamp content-prefs.sqlite would provide some timeline information, keeping in mind that it can represent the addition or the deletion of an entry in this database. The SQL statement below provides a joined output from all three tables within the database for a scenario where the text size was increased once for the webpage www.google.com.



Permissions

Useful sqlite statements	● SELECT host, permission AS "1=Allow" FROM moz_hosts WHERE type="**popup**" ● SELECT host, permission AS "1=Allow, 2=Block, 8=Allow for session only" FROM moz_hosts WHERE type="**cookie**" ● SELECT host, permission as "1=Allow" FROM moz_hosts WHERE type="**install**" ● SELECT host, permission as "1=Allow, 2=Block" from moz_hosts WHERE type="**image**"

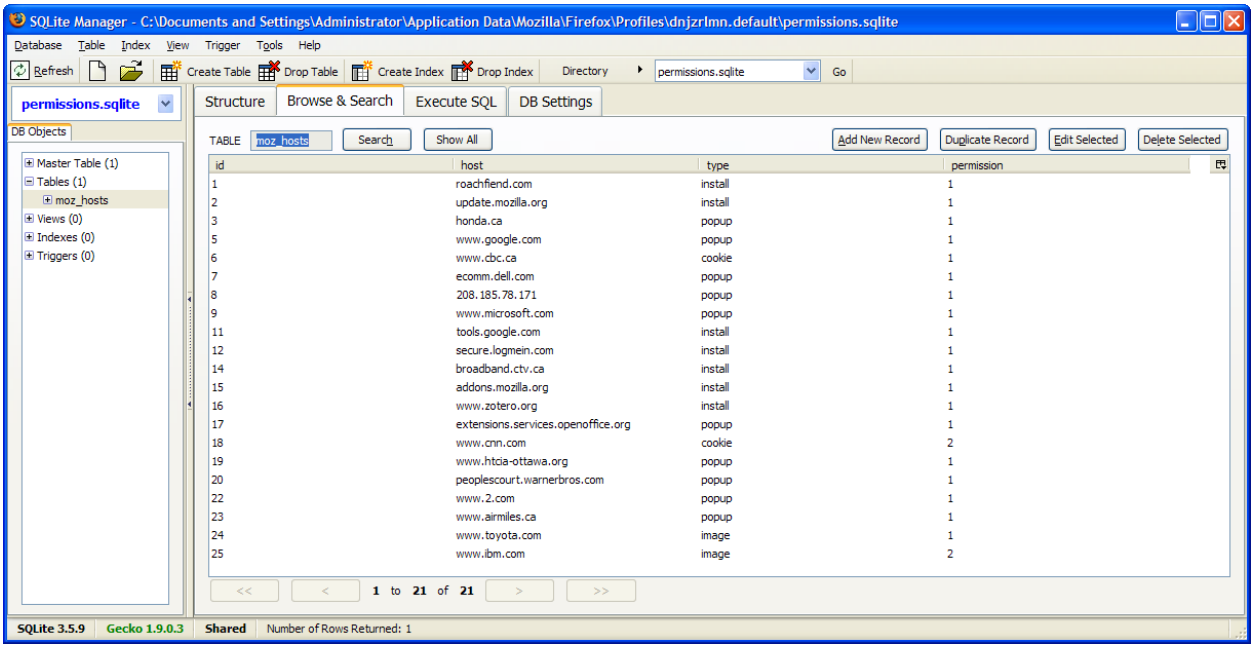
c Permissions that were previously stored in the hostperm.1 file are now stored in the database file permissions.sqlite found in the user's profile directory.

According to http://kb.mozillazine.org/Profile_folder_-_Firefox, this file is the "Permission database for passwords, cookies, popup blocking, image loading and add-ons installation."

Possible types: popup (1 = allow)
cookie (1 = allow, 2 = block, 8 = allow for session only)
install (1 = allow – do not warn when a site tries to install an add-on)
image (1 = allow – load images automatically even if the check box "Load images automatically" is not set {about:config entry – permissions.default.image},
2 = block images for particular sites even if the option is selected to load images automatically)

Exceptions for passwords are stored in signons3.txt (see password section for more on same). Likewise cookies and popup covered in greater detail in each of their respective sections.

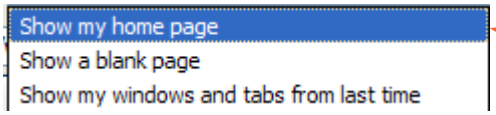
Much like per site preferences noted earlier, this is another valuable place to look for evidence. Because even if a user clears his/her private settings, it does not clear this database. A user may unknowingly leave evidence of URL activity via popup, images, and cookie exceptions.



Home page/Start Page

Firefox Menu	• Tools, Options, "Main" Tab
about:config entries	a browser.startup.page (default – 1) b browser.startup.homepage (default – resource:/browserconfig.properties)

a browser.startup.page



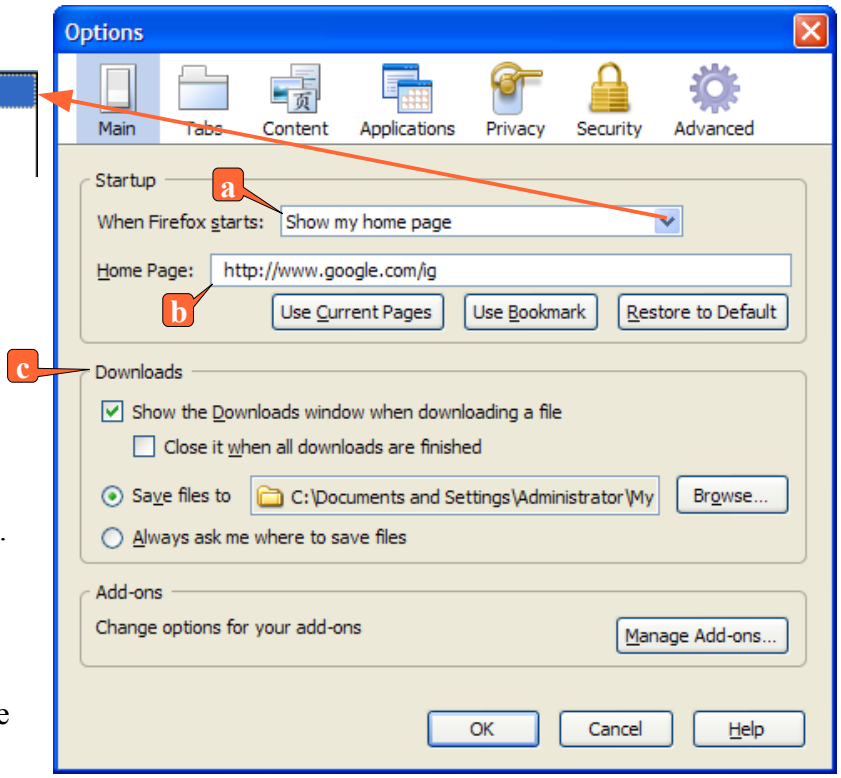
- **0** – Show a blank page
- **1** - Show my home page (default).
- **3** – Show my windows and tabs from last time.

b browser.startup.homepage

- resource:/ browserconfig.properties (default)

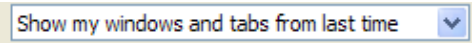
Contains URI for homepage. A pipe | can be used to separate multiple home pages. When used, it will cause each to be opened in a separate tab when starting the browser or clicking on the home page icon.

c See chapter on downloads.



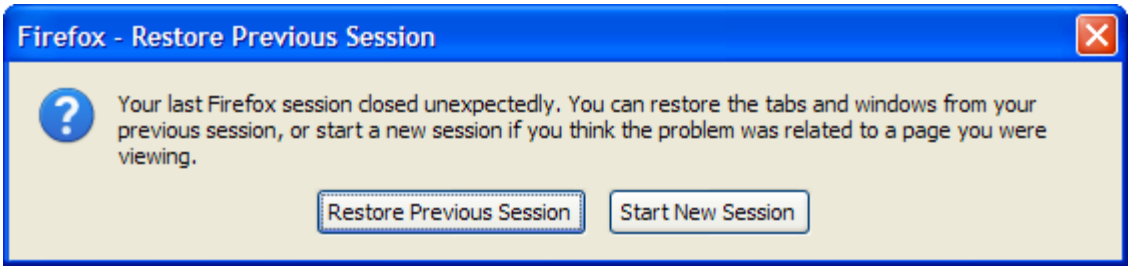
sessionstore.js

During a browser session Firefox keeps track of the open tabs. If a use closes a tab they can re-open it by pressing SHIFT-CTRL-T (CTRL-T is to open a new tab). That will open the last tab that was closed during the current browsing session. You can repeat this to re-open other tabs closed during the session. The file sessionstore.js located in the user's profile folder contains a list of open tabs for that session (including tabs that have been closed). It is suspected that this is what Firefox uses to track tabs that have been closed during the current session.

In the above above for homepage, if a user chooses 

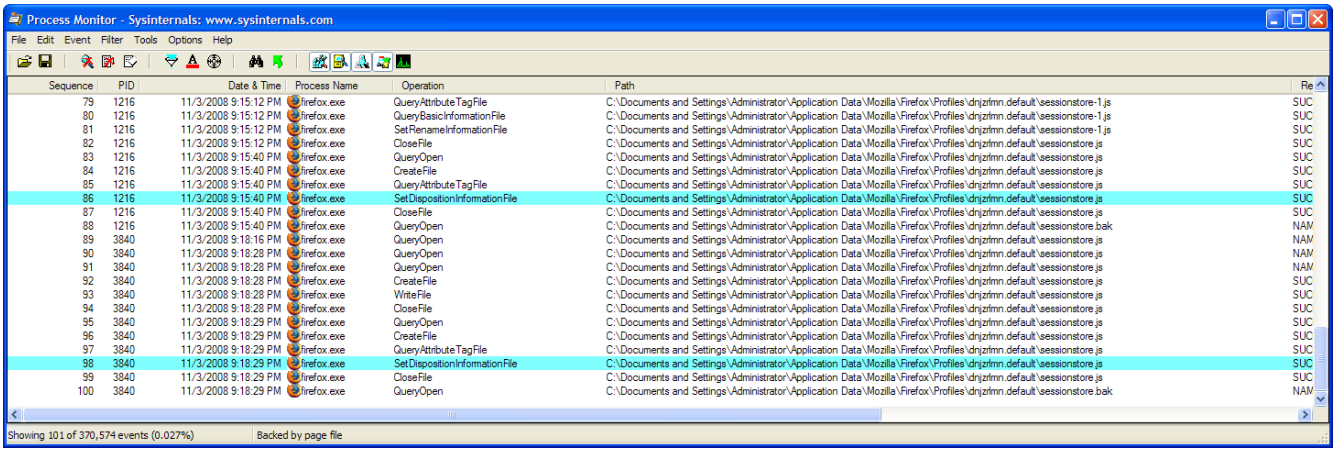
Firefox will track this through the file sessionstore.js. While a session is active, the file will have session:{state:"running"} at the end of the file. However once Firefox shuts down using the above option, providing it shuts down properly it will have session:{state:"stopped"} at the end of the file.

If Firefox does not shut down properly, it will not be able to change the state within sessionstore.js from running to stopped. If this happens, the next time Firefox starts it will recognize that it did not shut down properly the last time and will prompt the user with the screen below.



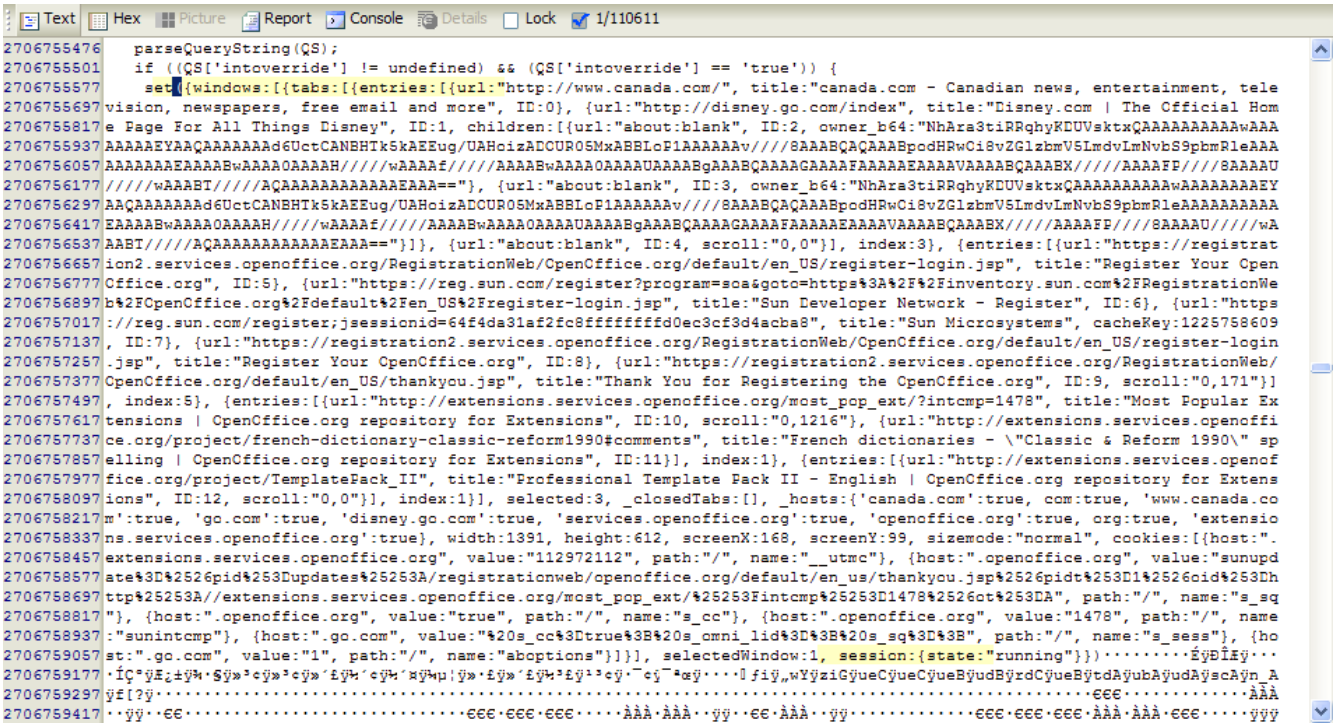
Firefox can restore the previous browsing session from the sessionstore.js file. If Firefox properly exits and the user did not elect to save the windows and tabs from the last browsing session, the file session store will be deleted.

The ProcMon screen capture below (with a filter to only show path which contains sessionstore) shows in the first instance where the user elects to save the windows and tabs from the last session. In that case the session information is written to sessionstore-1.js first and then that file is renamed to sessionstore.js (Sequence #81). In Sequence #86 during the next session Firefox sets the file to delete=true (SetDispositionInformationFile) after having previously saved it from the last session, resulting in sessionstore.js being deleted upon closing in Sequence #87. This is what happens when a user exits Firefox properly and does NOT elect to save their session. **What is important to note** is that the old sessionstore.js file gets deleted. **Which means it ends up in unallocated space.** Sequence #98 shows what happens from session to session when a user does not save it. **Each time** the file sessionstore.js gets deleted thus **ending up in unallocated space**.



As a forensic investigator, this means you can search unallocated space for evidence of prior Firefox browsing sessions. You do this with a keyword search for the start of the file:
({windows:[{tabs:[{entries:[{url:
or the end of the file:
session:{state:
The best thing to do is to do a keyword search for both of them. This will visually mark the beginning

and the end of a full sessionstore.js file. The screenshot below from EnCase demonstrates this.



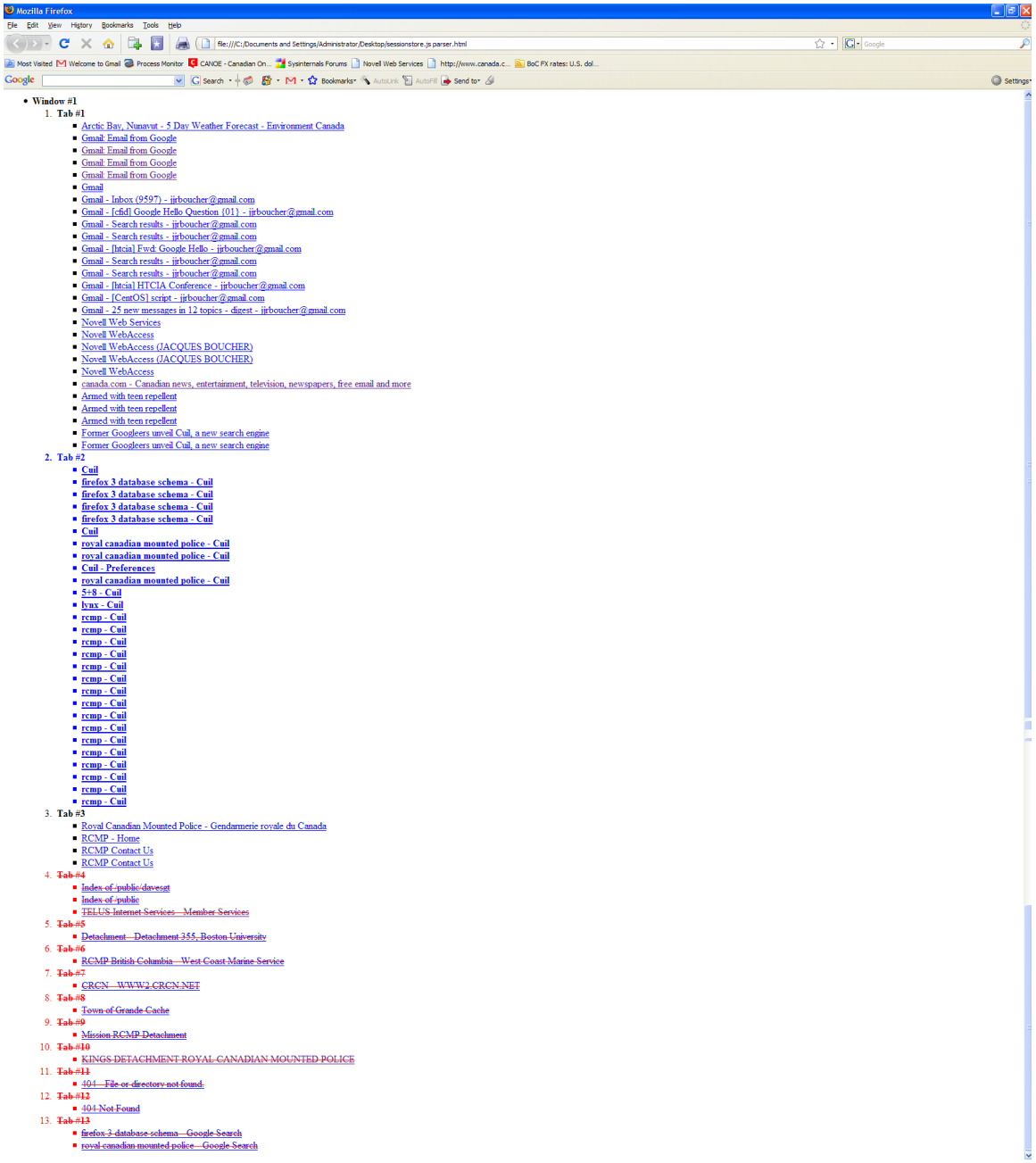
You can then select the entire sessionstore.js file from unallocated space and export it out to a file. Copy it over to your Firefox profile folder on your sandboxed system and start it up. Firefox will think that it did not shut down properly the last time because of the state:"running" and will therefore prompt you with the option to restore your last session. Choose to restore it and you will see the tabs that the user had open in that particular session.

If the state was "stopped", you can either configure Firefox to start with the windows and tabs from the last time and then overwrite the sessionstore.js file from your local machine with the exported one, or edit the exported one to say state:"running" instead of "stopped".

When the above two keywords were run against unallocated space on a system where Firefox is the primary browser, it yielded 72 hits for the start of the file and 219 hits for the end of the file. So the maximum number of intact sessionstore.js files would be the lower of the two, 72.

VERY IMPORTANT: Once you've recovered a sessionstore.js file and open it in Firefox, make sure you do SHIFT-CTRL-T as many times as necessary to open up tabs that were closed during that session but that still exist in the sessionstore.js file waiting to be resurrected.

Another alternative (perhaps an even better one than above) is to copy out the deleted session and paste it into the following HTML document at the noted location. Start off by copying/pasting the html document in Appendix "P" to a file called sessionstore-parser.html. Then copy out the deleted session from your forensic tool and past it where you see **<PUT CONTENTS OF sessionstore.js HERE!>**. Overwrite the opening less than and closing greater than as well as the text in between with the recovered sessionstore.js file. Then open that file in your browser where you will see something like below. Information from each of the tabs, including tabs that were closed but still recoverable via SHIFT-CTRL-T.

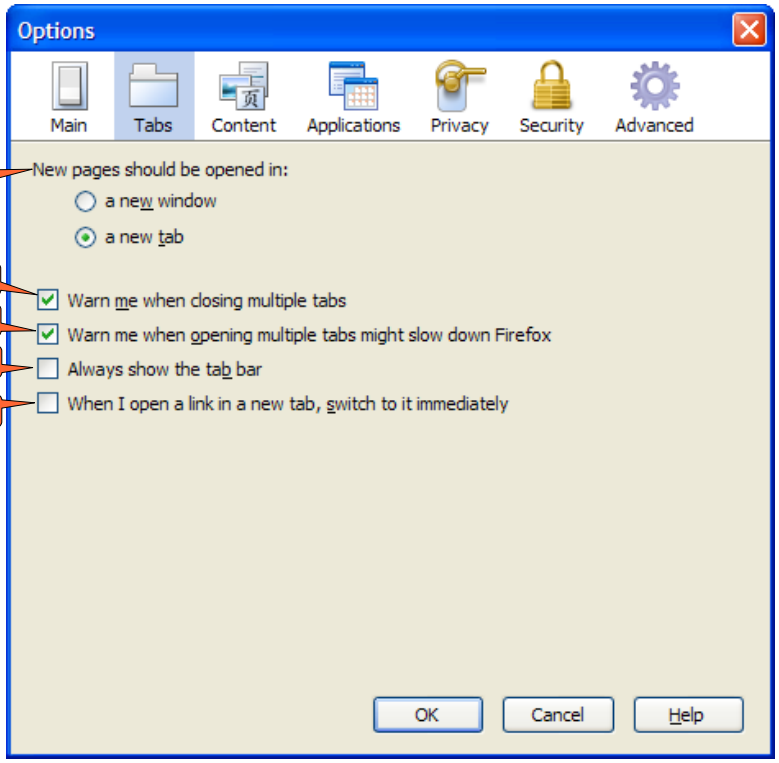


In the even the sessionstore.js is not complete you can still analyze it manually if necessary.

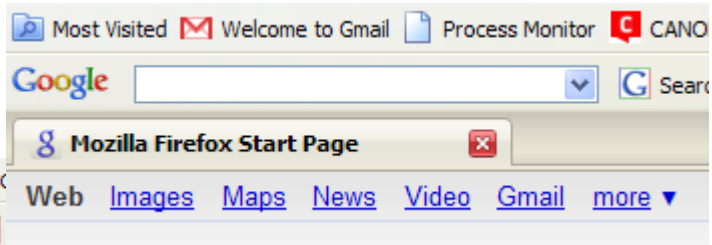
Tabs Settings

Firefox Menu	● Tools, Options, "Tabs" Tab
about:config entries	a browser.link.open_newwindow (default – 3) b browser.tabs.warnOnClose (default – true) c browser.tabs.warnOnOpen (default – true) d browser.tabs.autoHide (default – true) e browser.tabs.loadInBackground (default - true)

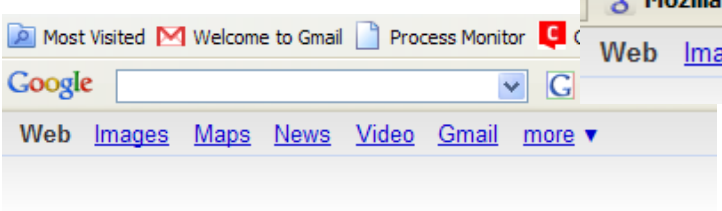
- a browser.link.open_newwindow
  - default - 3 – open in a new tab.
  - 2 – open in a new window.
- b browser.tabs.warnOnClose
  - default - true, warn when closing multiple tabs.
- c browser.tabs.warnOnOpen
  - default - true, warn when opening multiple tabs might slow down Firefox.
- d browser.tabs.autoHide
  - default – true, will auto hide the tab bar
- e browser.tabs.loadInBackground
  - default – true, will load tabs in background, remaining on the current tab.
  - False – will switch to the new tab.



d When set to false, tab bar displayed even if only tab open.



d When set to true, no tab displayed if only one open.



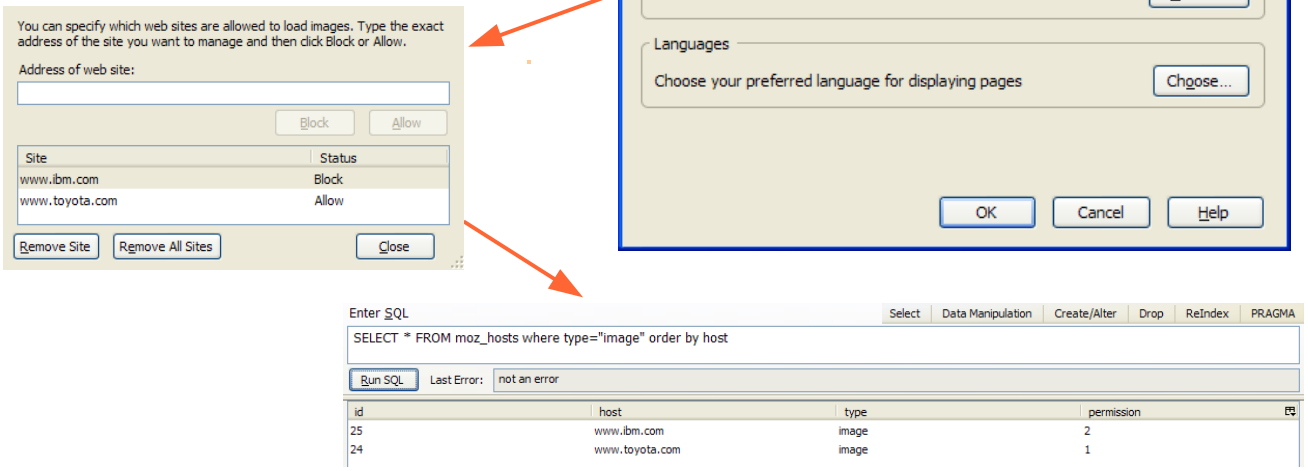
Tools, Option, Content

Firefox Menu	● Tools, Options, "Content" Tab
about:config entries	b permissions.default.image (default – 1, value of 2 means not checked) c javascript.enabled (default - true) d security.enable_java (default - true)

a See chapter on pop-ups.
b When checked, will load images automatically.
  - Exceptions will result in entries in permissions.sqlite.

c When checked, enables JavaScript.
  - The advanced settings appear to have no forensic value therefore not documented.

d When checked, enables Java.

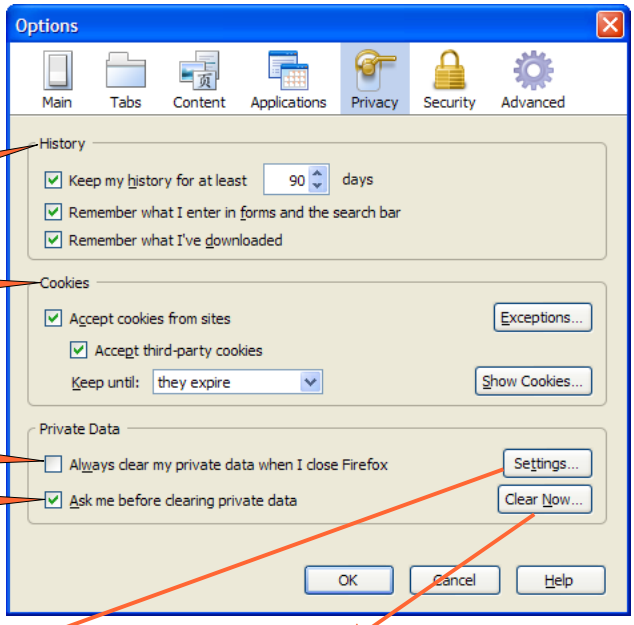


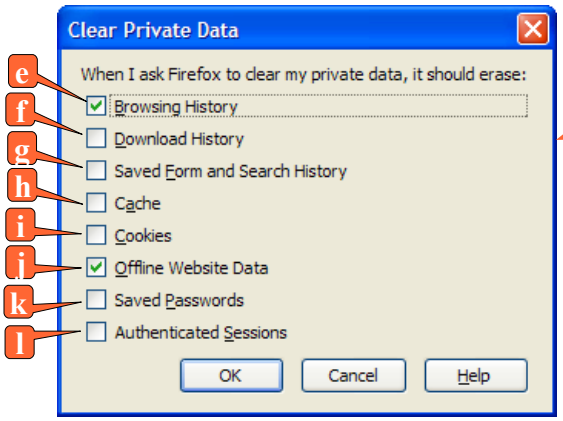
In the above screenshots we see the GUI view within Firefox 3, and below it the corresponding entries in permissions.sqlite. Notice that a status of Block for www.ibm.com yields a permission value of 2. And a status of Allow for www.toyota.com yields a permission value of 1.

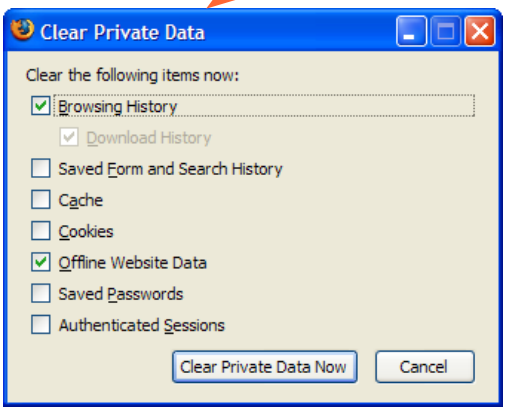
Privacy Settings

Firefox Menu	• Tools, Options, "Privacy" Tab
about:config entries	c privacy.sanitize.sanitizeOnShutdown (default - false) d privacy.sanitize.promptOnSanitize (default - true) e privacy.item.history (default - true) f privacy.item.downloads (default – true) g privacy.item.formdata (default – true) h privacy.item.cache (default – true) i privacy.item.cookies (default – false) j privacy.item.offlineApps (default – false) k privacy.item.passwords (default – false) l privacy.item.sessions (default - true)

a See chapter on history.
b See chapter on cookies.
c If checked (true), will clear privacy settings on shutdown of Firefox.
d If checked (true), will prompt the user before clearing privacy settings on shutdown. Otherwise will clear without prompting (if box c is checked).
e to l: If checked, that particular info will be cleared when clearing privacy settings.





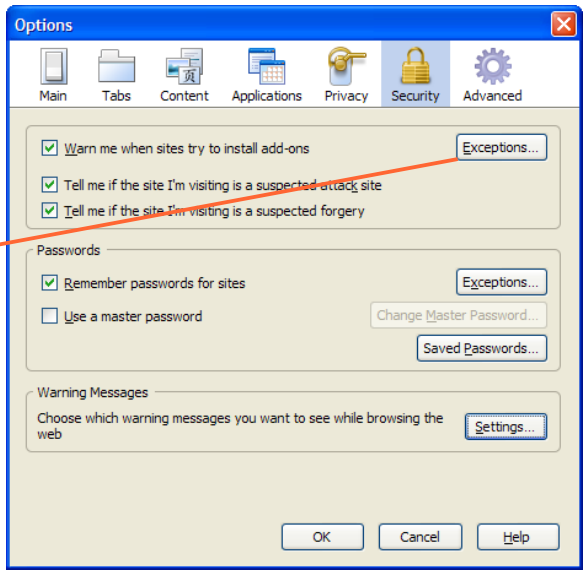


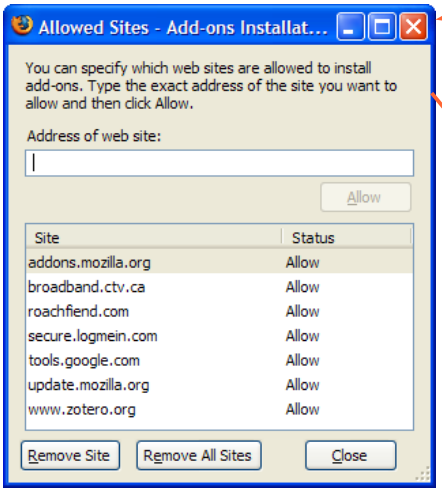
If a user clicks on "Clear Now", the dialog window to the right will open up allowing the user to "Clear Private Data Now". The boxes checked off by default match those that were selected under Settings above. A user can override these choices at that time. So although a user may configure Firefox to clear certain privacy settings, they can easily change that on the fly.

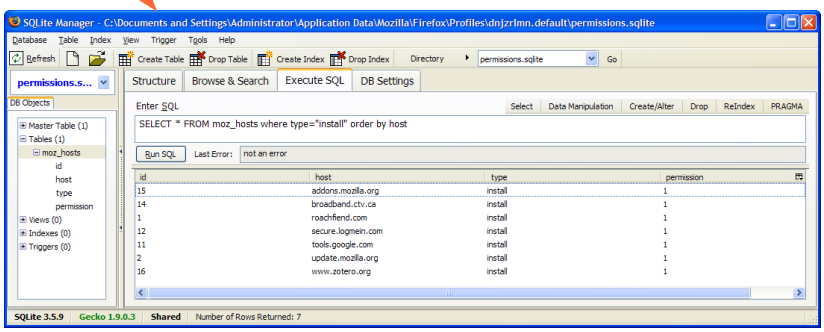
Security Settings

Firefox Menu	• Tools, Options, "Security" Tab
about:config entries	a xpinstall.whitelist.required (default - true) b browser.safebrowsing.malware.enabled (default - true) c browser.safebrowsing.enabled (default - true)

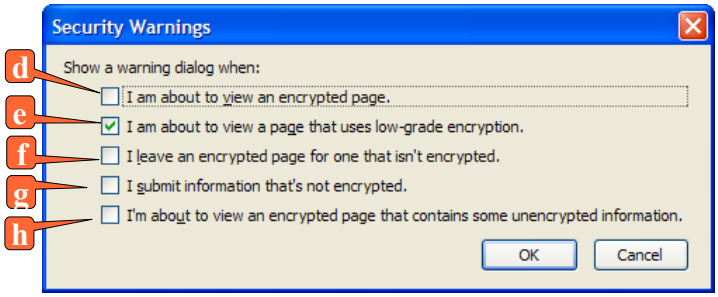
- If checked, will get a dialog window warning when a site attempts to install an add-on.
- If checked, will get a dialog window warning when a site is a suspected attack site.
- If checked, will get a dialog window warning when a site being visited is a suspected forgery (i.e. phishing site).







The window above is the Firefox GUI window when clicking on exceptions. The window to the right is the entries in permissions.sqlite resulting from the above.



about:config entries	d security.warn_entering_secure (default – false) •

Chapter 4

Bookmarks

Firefox 3 has significantly changed how it stores bookmarks. Previously they were stored in bookmarks.hml located in the user's profile folder. Now they are stored in places.sqlite. The tables within that database that deal specifically with bookmarks are moz_bookmarks, moz_bookmarks_roots, moz_keywords, and moz_items_annos. See places.sqlite database schema in Appendix "D" for more info on the relationship between the various tables within places.sqlite.

Firefox Menu	• Bookmarks, Organize Bookmarks, as well as CTRL-B, or the star in the address bar.
Path of Relevant Files	• C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\places.sqlite a C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\bookmarkbackups\bookmarks-yyyy-mm-dd.json • C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\bookmarks.html & bookmarks.bak
about:config entries	b browser.bookmarks.max_backups (default - 5) c browser.bookmarks.autoExportHTML (default - false) d browser.places.importBookmarksHTML (default – true - but changed to false after FF started for first time and bookmarks imported?)
Useful sqlite statements	• select url, moz_bookmarks.title as "Title in moz_bookmarks", moz_places.title as "Title in moz_places" from moz_bookmarks, moz_places where fk=moz_places.id order by moz_bookmarks.title

a Firefox 3, as with Firefox 2, maintains a backup of the bookmarks. In Firefox 3 it does so in the bookmarkbackups folder under the user's profile, in files named bookmarks-yyyy-mm-dd.json where yyyy-mm-dd is the date when the backup was made. **This is a valuable place to look for evidence of past bookmarks that a user may have deleted from the current set of bookmarks to attempt to hide it from authorities.** Firefox 3 has a feature to backup or restore bookmarks to/from .json files. Although the file is a plain text file therefore easily viewable, it is much easier to simply restore it to a sandboxed system running Firefox 3 where ALL bookmarks from that profile have been removed. You can then simply view them through the Organize Bookmarks GUI within Firefox. And if you wish you can certainly use the sqlite manager add-on to query the databases or export the data out to a CSV file for further processing using another tool.

b The preference browser.bookmarks.max_backups controls how many backups will be retained. The default value is 5 and they are stored in the folder bookmarkbackups in the related profile folder.

c & d are options to either automatically export bookmarks from the sqlite database file to bookmarks.html within the user's profile, or automatically import the content of bookmarks.html within the user's profile into the sqlite database. **This means that a user that upgraded from Firefox 2 to Firefox 3 will have their old bookmarks.html.** Or if they ever enabled the option to export to bookmarks.html, even if they later disable that option the last exported version of bookmarks will remain in bookmarks.html.

Understanding Bookmarks

When a user clears FF 3's history, it is re-populated automatically with the entries from bookmarks. This means that when a user starts to type something in the address bar that matches something in the bookmarks, it will populate the pull-down menu accordingly. Initially this can give the wrong impression that the history was not cleared. Bookmarked entries will have a yellow star next to them in the pull down menu whereas non-bookmarked items in history will not have a star. In the screen capture below where "proc" was typed in the address bar, we see the first three entries are from bookmarks, while the next three are not.

Firefox does not automatically place bookmarks at the top. Rather it is based on frecency (with those weighted equally displayed in apparent random order however insufficient testing done to date to make any conclusions on how it decides which one to list first when frecency are equal).

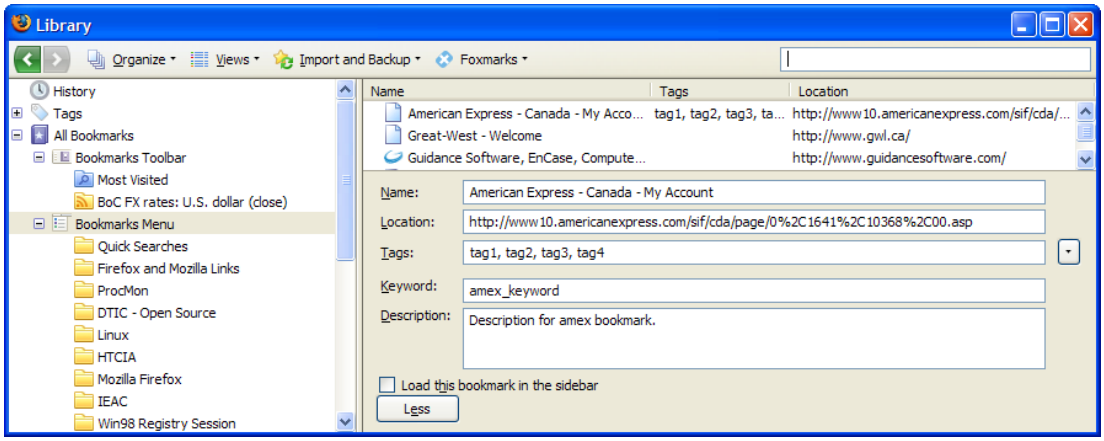
Tags

In Firefox 3 you can assign tags to bookmarks. You can assign as many tags as you wish for a bookmark. When you start typing in the address bar, Firefox will search all bookmarks, all history, and all tags and display those that match that string. In the example above, Firefox lists entries from bookmarks and from history where either the URL contains the string "proc", the title contains the string "proc", or the tag contains the string "proc". If you are gmail user, think of tags as labels in gmail. It allows you to take a bookmark and assign several tags such as automotive, hobby, and travel. The next bookmark might have the tags hobby, woodworking, education. And the next bookmark could have education, ontario. If you want to list all bookmarks relating to hobby, simply type hobby in the address bar. Thus all bookmarks with the tag hobby will be listed in the drop down list (along with those with "hobby" in the URL or in the description). This way you can place a bookmark in a particular folder, but assign it tags to associated it to other categories.

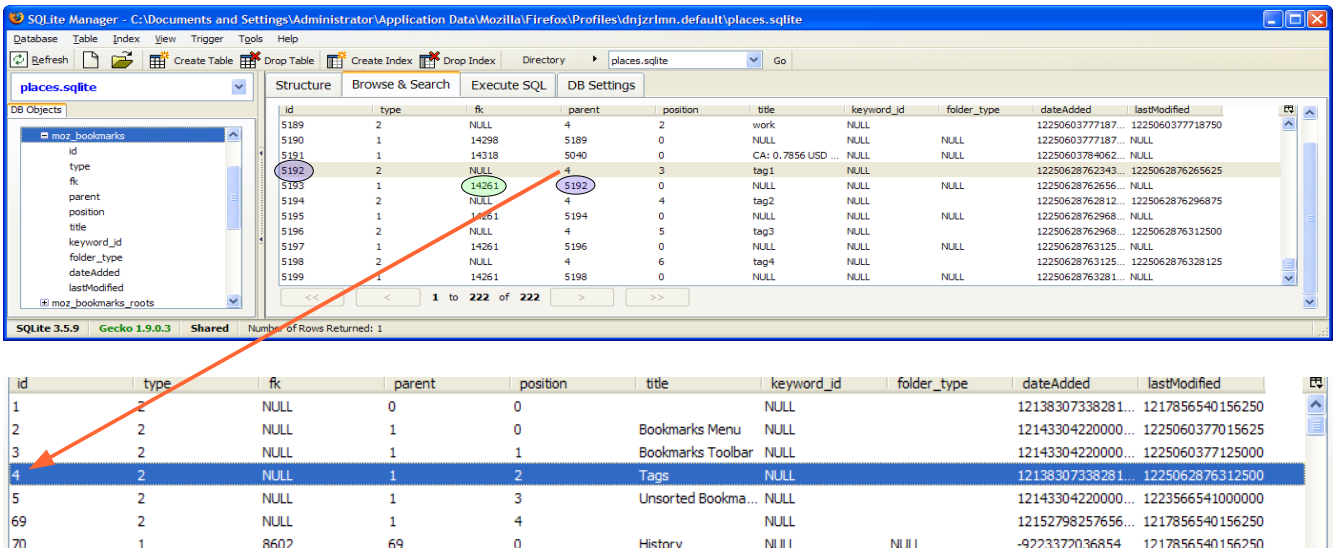
From a forensic perspective, tags are stored in moz_bookmarks of places.sqlite along with regular bookmarks. When a tag is added to a bookmark, **two entries are created in moz_bookmarks for each tag**. The field title (moz_bookmarks.title) will contain the tag string. moz_bookmarks.fk which is the field that contains the id number that points to the appropriate entry in moz_places will have a NULL value. The field moz_bookmarks.parent will have a numerical value which points to the parent entry within the bookmarks hierarchy. At the time of this writing, tags have a value of 4 in this field. And entry # 4 within moz_bookmarks is the tags subfolder. A second entry is created immediately following the tag entry. That entry has the tag entry as its parent, but its fk value points to the appropriate entry within moz_places for the bookmarked entry to which that tag is associated with.

A more graphical representation of this association follows:

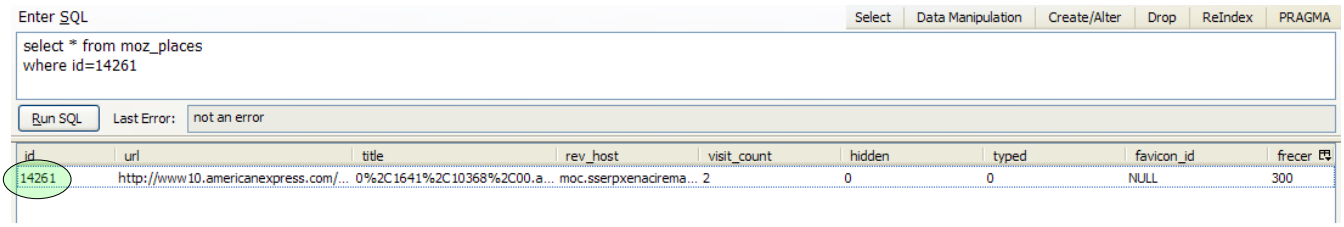
Bookmarked entry for American Express. Note the tags assigned to it. Also note in the left pane we have History, then Tags, followed by All Bookmarks.



Note the entries in moz_bookmarks for the tags above. We see that its parent is 4 (which is tag in the left pane in the above screenshot), and that the fk value is NULL. So this entry does not point to the URL to which it is associated. The parent value is used to properly structure the hierarchal layout of the menus.
The entry that follows that one has a parent of 5192, which is the tag. And it has a fk value of 14261, which is the entry in moz_places for the URL in question.



Below we see that indeed entry 14261 in moz_places is for Amex, the bookmarked entry which contains the tags in this example.



Keywords

Firefox 3 also supports keywords when bookmarking a page. Contrary to tags, you can only assign one keyword to a bookmark. If you assign the keyword "hobby" to a bookmark, it will not cause that bookmark to display in the drop down list when you type "hobby" (unless that string exists in the URL or description). However if you press ENTER, it will insert the URL for that bookmark that has hobby as its keyword.

If you have more than one bookmark with the same keyword, typing in that keyword will hit on the first bookmarked entry that has that keyword assigned to it (respecting the order in which bookmarks are being displayed in your bookmarks menu when searching for the first bookmark with a particular keyword). Because of this you do not want to assign the same keyword to more than one bookmark contrary to tags, where you can assign the same tag to multiple bookmarks (and typically you would). Keywords can be used for example for your router. Bookmark the IP for your router, then assign it the keyword "router". In the future you need only type router in the address bar to navigate to it.

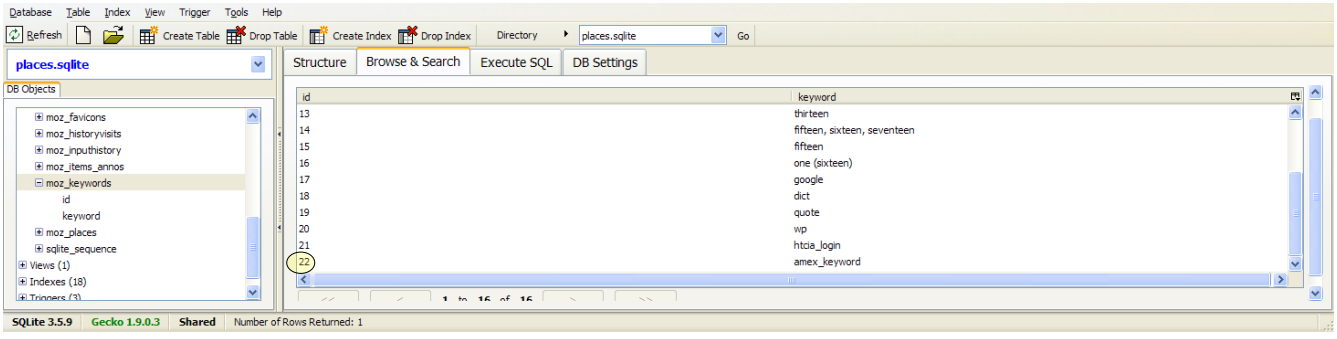
Impact of keywords on history – when you type a keyword in the address bar to navigate to a bookmark which has that keyword, it will cause an entry in places.sqlite with the actual URL in the history, not the keyword. Which means from a forensic perspective keywords will likely have minimal value other than to perhaps show greater knowledge/intent by a user if a keyword has been assigned to a bookmark.

From a forensic perspective, we can find these keywords that a person can enter in the address bar to quickly access a bookmark. **Remember that the resulting entry in the history is the actual URL visited, not the keyword.**

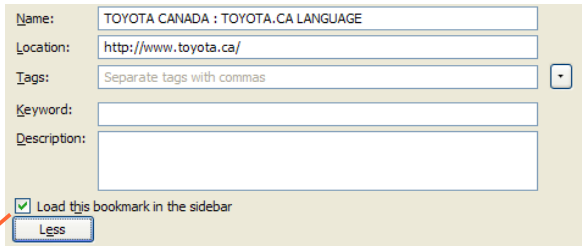
If we examine moz_bookmarks we note the keyword_id is 22 for American Express, and the value of fk is 14261 (same as earlier naturally seeing it's the same bookmark we analyzed above in the tags scenario). As you recall that points to entry 14261 in moz_places, the URL for that bookmark.

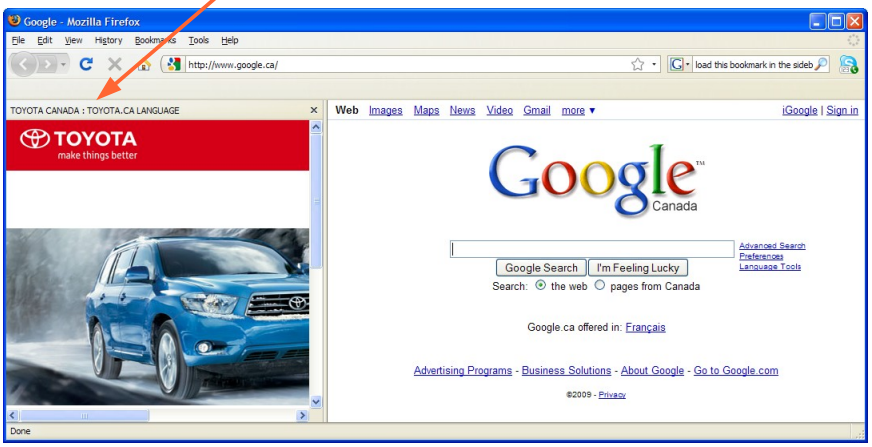
id	type	fk	parent	position	title	keyword_id	folder_type	dateAdded
75	1	8606	72	2	Unsorted Bookmarks	NULL	NULL	1213830733828125
4989	1	14261	2	0	American Express - Canada - My Acc...	22	NULL	1096069907000000
4990	1	14262	2	1	Great-West - Welcome	NULL	NULL	1096069907000000
4991	1	12735	2	2	Guidance Software, EnCase, Comput...	NULL	NULL	1096069907000000
4992	1	14263	2	3	Men & Mice's DNS Glossary	NULL	NULL	1096069907000000

If we examine moz_keywords for entry #22, we see the keyword "amex_keyword" which is indeed what was set for that bookmark (see screenshot of same in the tags example).



When you choose Organize Bookmarks and select one of the bookmarks, you can click on a button to see more details (you can collapse the expanded view by clicking on Less as in the screenshot). One of the options is to "Load this bookmark in the sidebar". If a user selects this option, when that bookmark is selected it will open up in a sidebar as per the screenshot below it.





Forensically this is likely of no concern to us . But the act of enabling/disabling that option within the Organized Bookmarks view will change the date moz_bookmarks.lastModified. (but not the other ones). As with other dates in Firefox, they are stored in PRTime (UTC).

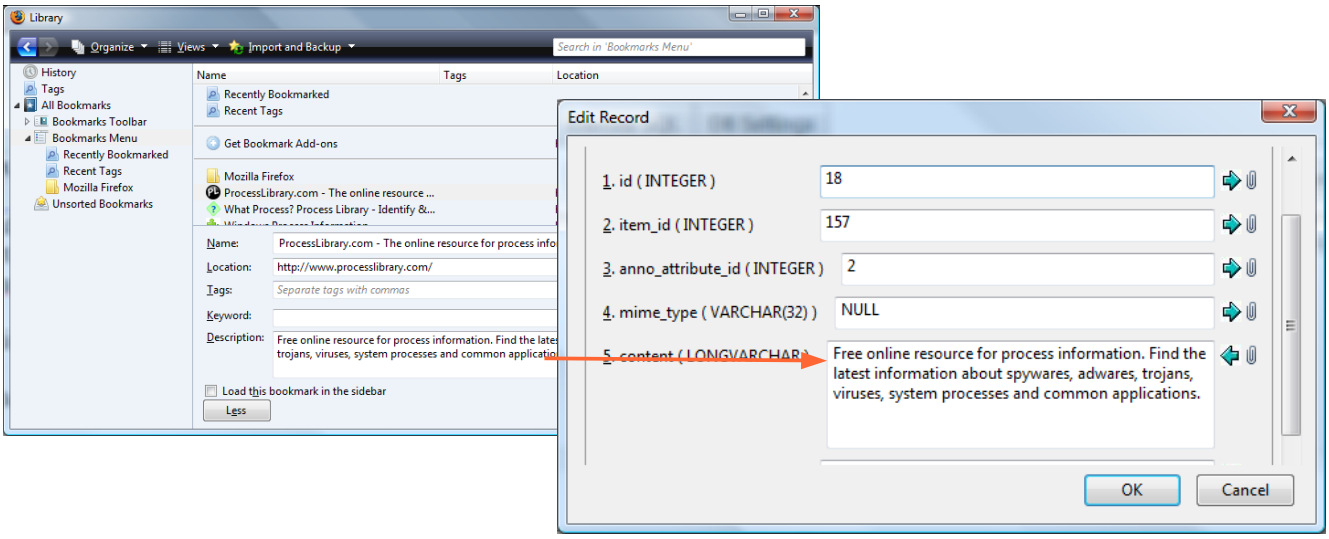
Next time Firefox is started, that sidebar and associated bookmark will open with it (until the user closes that sidebar).

As a forensic examiner you may need to make some timeline analysis. To do this you must understand what causes these dates to be updated. So let's summerize actions and effects of actions on these dates/ times. Keep in mind that these dates have to do with bookmarking activity, not necessarily surfing activity. Although typically a person will navigate to a page to bookmark it so the original dateAdded for moz_bookmarks would be very suggestive that the user navigated to that page on that date/time – but keep in mind that I could bookmark toyota.ca, then later edit that to navigate to ford.ca without ever having had to navigate to ford.ca.

IMPORTANT: remember that add-ons can potentially interact with bookmarks. And in doing so those actions may impact the associated dates/times.

Action	moz_bookmarks		moz_annos	
	dateAdded	lastModified	dateAdded	lastModified
Creating a new bookmark	Inserts current date	Inserts current date	Inserts current date	0
Selecting a bookmark	No change	No change	No change	No change
Editing name	No change	Inserts current date	No change	No change
Editing URL	This yields some interesting results. A new entry is created in moz_places (assuming the new URL is not already in there). The bookmark entry in moz_bookmarks.fk is changed to point to the new entry in moz_places for the new URL. The entry moz_annos.place_id continues to point to the old entry in moz_places. No entry results in moz_historyvisits (which is the behaviour we'd expect as the site was not visited).			
Adding, editing, or removing a description	No change	Inserts current date	No changeNo change to original entry. No associated entry for the secondary moz_bookmarks entry resulting form the tag.	No change
Adding a tag	No change to original entry. As explained earlier in the chapter, a second entry is created in moz_bookmarks for a tag, however no associated moz_annos entry is created. This second entry will have Current Date	No change to original entry. As explained earlier in the chapter, a second entry is created in moz_bookmarks for a tag, however no associated moz_annos entry is created. This second entry will have Current Date	No change to original entry. No associated entry for the secondary moz_bookmarks entry resulting form the tag.	No change to original entry. No associated entry for the secondary moz_bookmarks entry resulting form the tag.
Adding a keyword				

Removing a bookmark description



If you remove the Description from a bookmark, it removes the entry in moz_items_annos. If that entry was in the middle of others, you will note a break in the order of the id numbers as a result of this. In the example above, it was id #18. Deleting the description from the Organize Bookmarks screen resulted in entry 18 being deleted from moz_items_annos resulting in id numbers going 17, 19, 20 (so 18 no longer there).

Re-adding that description via the Organize Bookmarks menu results in it being assigned the next available id #, being 21 in the example above (so it does not re-assign it the previous id #). However when deleting the description once again of this bookmark which now occupies the very last entry in moz_items_annos, and then re-creating it, it does assign it 21 again. So it appears that it will assign it the next available number even if that number was previously used but then deleted.

D-R-A-F-T

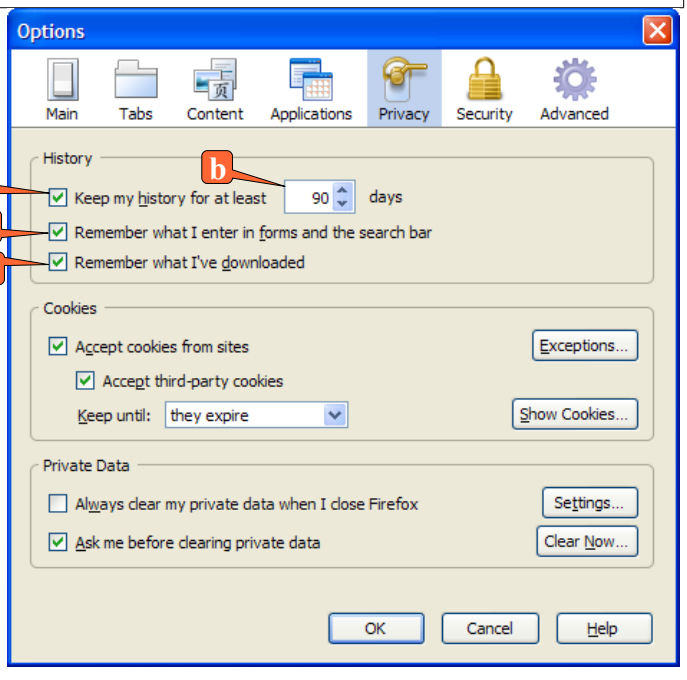
Add info on bookmark backups here... Do the old ones end up in unallocated? Can we restore a suspect one to a new profile in order to analyze? What about presence of bookmarks.html and possibly bookmarks.bak

Chapter 5

History and Typed URIs

In Firefox 3 history and typed URIs are all stored in the database places.sqlite. The history is stored in the moz_historyvisits table with the noted database, and the flag to indicate that something was typed is stored in the moz_places table. See places.sqlite database schema in Appendix "D" for more info on the relationship between the various tables within places.sqlite.

Firefox Menu	• History, Show All History or CTRL-H
Path of Relevant Files	• C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\places.sqlite
about:config entries	a browser.history_expire_days (default - 180) b browser.history_expire_days_min (default - 90) c browser.formfill.enable (default - true) d browser.history_expire_days.mirror (default - 180) e browser.download.manager.retention (see chapter on downloads for info on this preference).
Useful sqlite statements	*View history excluding hidden URIs, reverse sort by visit_count (largest to smallest)* • SELECT url, title, visit_count, typed AS "typed URL? (1=yes, 0=no)" FROM moz_historyvisits, moz_places WHERE place_id = moz_places.id AND NOT hidden=1 order by visit_count desc *View all typed URIs reverse sort by visit_count** • SELECT url, title, visit_count FROM moz_historyvisits, moz_places WHERE place_id=moz_places.id AND typed=1 order by visit_count desc



a # of days before a history entry expires.
b Minimum number of days to retain history.
c See chapter 10 on form data for more info on this preference.
d There does not appear to be a menu option to change this value. If changed manually, it resets itself to the default (180) as soon as you effect a change to option a.
e See chapter 8 on downloads for more information on this preference.

*In the case of the typed URI query, an alternative would be SELECT url from moz_places where typed=1. This could

produce different results than the previous SELECT statement for typed URIs. An example of this would be if a user visits www.toyota.ca by typing it in, and then bookmarks it. The individual clears his history. Which means moz_historyvisits is now empty. The first SELECT statement will make an association between entries in moz_historyvisits and moz_places and then list those that typed=1 within that list of associated entries. But where history has been cleared, there will be no entries in moz_historyvisits thus the SELECT statement would yield no hits. The second select statement looks at moz_places only for typed=1. Remember that we typed www.toyota.ca earlier on and bookmarked that value. Which means that there will be an entry in moz_places for that site by virtue of it being bookmarked, and it will have typed=1. Thus the second statement would hit on the bookmarked entry for www.toyota.ca whereas the first one would not as previously explained.

Impact of sites that open within a frame...

Favicons

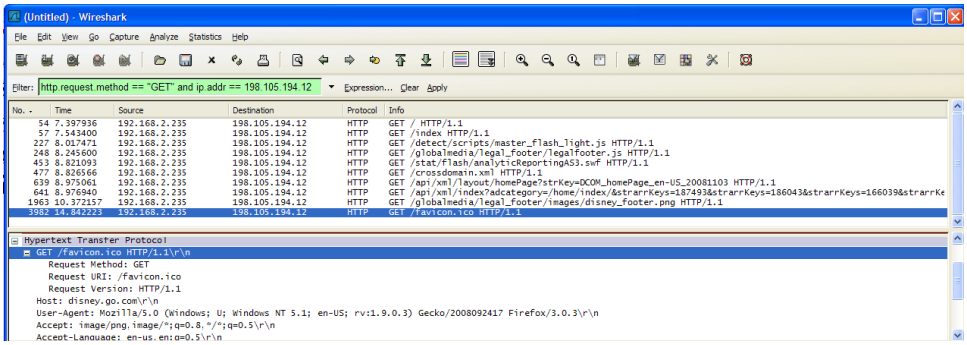
Firefox Menu	• Nil
Path of Relevant Files	• C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\places.sqlite
about:config entries	a browser.chrome.favicons (default - true)
Useful sqlite statements	*View all typed URIs reverse sort by visit_count* • SELECT url, expiration FROM moz_favicons

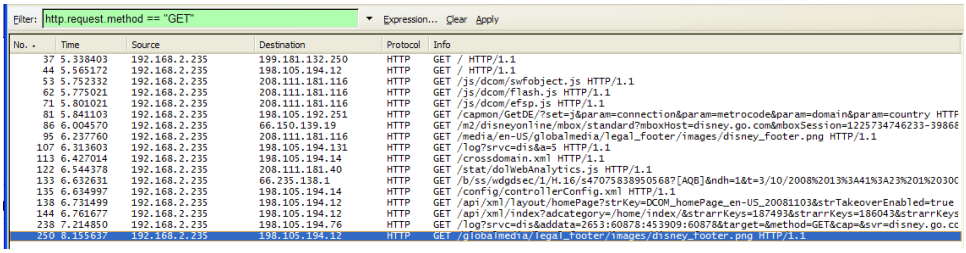
When you navigate to a web site, your browser will request from the web server (GET) the icon it should display in the address bar, favicon.ico. If there is an icon set for it, the server will send that to the browser. Firefox saves information about that icon in places.sqlite in the table moz_favicons. This favicon applies to the browser's history and to bookmarks. Remember that both moz_bookmarks and moz_historyvisits points to moz_places where we find the actual URL. It is moz_places that then points to moz_favicon. See appendix "A" for the database schema that demonstrates this.

a This preference determines if Firefox will request the favicon.ico or not from the server. By default it will (True). If you set this preference to false, it will not request the favicon, thus will not store it either.

When a favicon is saved in moz_favicons, it is assigned an expiry date. This expiry date is in PRTime/ Epoch Time same as all other dates/times we've seen in Firefox 3 sqlite files. This expiry date is set to 24 hours after the local machine's time, in UTC/GMT time. This is easily observed by changing your local machine's date/time before visiting a site that will yield an entry in moz_favicons and noting that the expiry date is 24 hours after the local machine's time.

In the Wireshark screenshot to the right we see that on our first visit to this page, it does several GETs, including the one for favicon.ico.





When we revisit the site after closing the browser (but not clearing privacy settings) we notice in the screenshot to the left that we do not see a GET for favicons.ico.

If we clear the history entry relating to that icon and then exit and restart Firefox, it will clear the entry in moz_favicons as long as no bookmark exists for it. Likewise if you delete the bookmark associated to the icon and then exit and restart Firefox we observe that the entry is removed from moz_favicons (as long as no entries exist in moz_historyvisits). However if an entry persists in either moz_bookmarks or moz_historyvisits then the entry in moz_favicons will persist.

If the expiry date for favicons lapses, the next time the browser visits that site it will request (GET) favicon.ico once again. And a new 24 hour expiry date will be set.

Potential forensic value: Because the icon has an expiry of 24 hours, any visit within that 24 hour period will not yield another get of favicon.ico. Which means that with the exception of the user turning off the preference for favicon or the favicon.ico being removed from the server during later visits, the date in favicon will reflect the last 24 hour period that the user visited the domain associated to that favicon. In other words if the expiry date in moz_favicon for a particular icon is November 4th, 2008 at 01:24 hrs GMT, then the last visit to that domain would have been in the 24 hours preceding that date/time.

Of course we have a visit date in moz_historyvisits that would provide us with the exact date/time of a particular visit. However if a user was to clear his history but still have that site bookmarked so that the favicon was preserved, we could rely on the expiry date set for its favicon to reach a conclusion of approximate last activity for that domain.

(Draft)
previously closed tabs
Where is it
format
select statements

pre-populated with bookmarks
moz_inputhistory
unallocated? Impact of privacy settings

Sessions field -
The field session under moz_historyvisits:

Keeps track of surfing sessions. Starts at 1 after history has been cleared and FF restarted. Increments by 1. As long as you keep following links from a page to the next, remains within the same session and the from visit will be the previous page from which the user navigated from. Clicking on a banner ad on a site that results in a new tab opening while previous tab remains open retain the same session number as the original tab which contained the banner ad. Clicking on a bookmark starts a new session. Opening a link in a new tab retains the same session number. Opening a new tab and navigating to a new page from a bookmark starts a new session number. Typing in a URL in the address bar starts a new session #. Navigating to a new bookmark (or new typed URL) starts a new session. However if you click on the back button to return to the previous page from the previous session and then continue navigating from it, then all historyvisits will retain that prior session number same as if you had never left that page. If you start a second instance of FF browser, it will create a new session (incremental as if a new page was opened within the existing browser instance). Dyamic content appears to create an unusually large session number. That very large session # is retained in traffic that flows from that original page same as other session numbers. However when I tried to re-create it by navigating to the same page that created it in the first place (but from a new tab), it resulted in the expected incremental session number, not a large number as previously observed. A network timeout does not appear to result in an entry in historyvisits. If upon exiting Firefox you choose to save the sessions, next time you start Firefox those sessions are restored. It does not result in any entries in historyvisits at that point. If you navigate from those pages, the appropriate prior session numbers are retained for that surfing session. Therefore it is possible that activity from one session number resulted from a new start of Firefox where the prior session was restored. Shift-Ctrl-Tab re-opens a closed tab and retains the session number that was assigned to that tab (no new entry in historyvisits when re-opening a closed tab in this fashion). If you click on a link to open a PDF document, it results in the following in historyvisits:

entry for the resulting .html page after the doc was opened – with session 0 – visit type 1
entry for the URL of the pdf itself – with session 0 – visit type of 7
entry for the resulting .html page (again – same as previous URL) with proper session # and visit type 1.
Note that this also results in an entry in the downloads.sqlite

If you do not open the PDF, but rather click on cancel, it only results in a single entry with session 0 and a visit type of 7 with the URL for the PDF itself, no other entries in historyvisits (whereas opening it yielded: resulting URL with 0, PDF with 0, resulting URL with proper session #).

When doing a timeline of sessions, overlaps means that the sessions overlapped (thus has to be in another tab or another instance of FF).

A Google search from the search bar results in a new session. Following a search hit will have the same session # as the Google search itself.

If you clear history (private data), it starts back at 1. Noted however some strange session #s in very high range ("69756660314138220000", "6975666031398487000", "6975666031386691000", "6975666031377319000") in amongst what appears to be otherwise normal numbering. Another anomaly noted, doing the following:
select * from moz_historyvisits
order by visit_date desc

Here are some entries with those very large numbers including the entries on either side of it:
C:\Documents and Settings\Administrator\My Documents\IEAC\FF3\Very large session numbers.txt

Does FF keep track of which tab a particular history activity takes place in?

A hidden URL will have a value of 0 under visit_count.

Chapter 6

Cache

related about:config entries
Where is it
format
select statements
pre-populated with bookmarks
unallocated? Impact of privacy settings
How do we ascertain the likelihood of a legitimate trojan defense?

Chapter 7

Cookies

related about:config entries
Where is it
format
select statements
unallocated? Impact of privacy settings

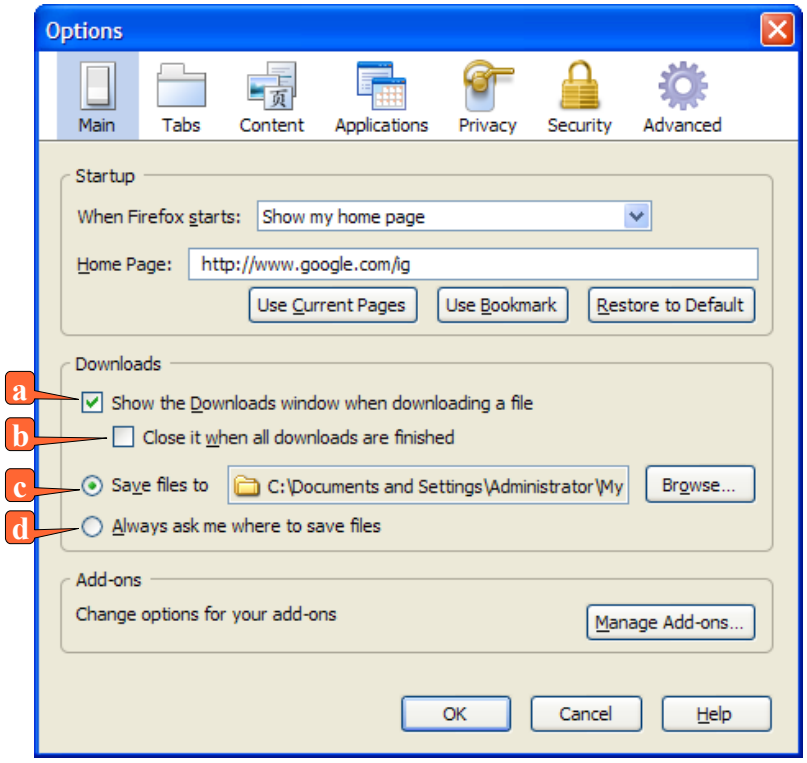
http://www.bewley.net/perl/cookie-test.html
http://www.thesitewizard.com/javascripts/cookies.shtml

id = creation date of cookie in unix time – PRTime
name = name of the cookie as given by the host
value – value of the cookie as given by the host
host – host that created the cookie
path – path on the host for which the cookie applies (frequently / meaning the root of the host)
expiry – expiry date of the cookie (PRTime)
lastAccessed – last time the cookie was accessed (PRTime)
isSecure – all have a value of 0 so far
isHttpOnly – most have a value of 0, a smaller number have a value of 1

Chapter 8

Downloads

Firefox Menu	• Tools, Options, "Main" Tab
Path of Relevant Files	• C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\downloads.sqlite • C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\prefs.js (settings for a specific user if set will be found in user.js)
about:config entries	a browser.download.manager.showWhenStarting (default – true) b browser.download.manager.closeWhenDone (default – true) c browser.download.dir (default – no value) d browser.download.useDownloadDir (default – true) e browser.download.manager.retention (default - 2)
Useful sqlite statements	• SELECT name, target, state AS "0=In Progress, 1=Complete, 3=Stopped, 4=Paused" FROM moz_downloads • SELECT name, target FROM moz_downloads WHERE startTime BETWEEN ################## AND ##################



a If set to true, will display the download dialog window when downloading.

b If set to true, will close the download dialog window when finished downloading.

c Location where downloads will be saved if d not selected.
- browser.download.lastDir will contain the path of the last download location if a user right clicked and chose Save As rather than accepting the default download location.

d If selected, will cause the associated preference to have a value of false (i.e. Do not use download directory).

e This preference controls whether or not Firefox remembers downloaded files. By default it is checked and thus the preference has a value of 2. If unselected the preference will have a value of 0 and will therefore be found in the prefs.js file. A value of 0 means the downloaded file will not be noted in downloads.sqlite (but any record of downloads already in downloads.sqlite will remain there and will display in the download dialog window until the user clears that privacy

setting). This preference is actually selected from Tools, Option, Privacy, and then under the History section (top third of that window). The third option in that section reads "Remember what I've downloaded". You can see the screen shot with that option in the chapter on History.

- browser.download.downloadDir – doesn't appear to be used...

When clearing private data via Tools menu, all downloads will be cleared from the database EXCEPT paused downloads. Because we are dealing with transactional activity in a database we will not find evidence of the cleared downloads in unallocated. A test revealed that the database was zero'ed out and no evidence was found in unallocated of a deleted download entry.

To decode the dates/times stored in this database or create queries for date ranges, see the appendix on PRTime/Epoch Time.

Forensic Process

During the course of a forensic examination you can copy downloads.sqlite out of your forensic tool to your forensic machine and then bring it into sqlite manager on your forensic machine (see appendix on sqlite manager if you require further instructions on how to use that tool). You can then query the database or browse it and copy out rows as required to paste into your report. As long as you are simply querying the database and copying out information you will not change the hash value of the file. This can be validated by hashing the file with a third party utility such as Hashcalc (http://www.slavasoft.com/hashcalc/index.htm) after you've finished your examination of the database and then comparing it to the hash value reported by EnCase for that same file within the forensic image.

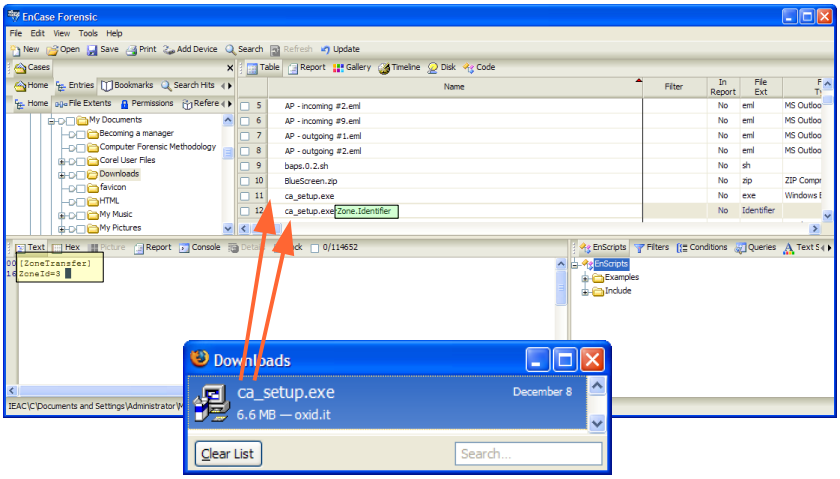
As for the settings for the relevant preferences, you can easily examine the prefs.js file from within your forensic software to note if any of the preferences relating to downloads are present or not. Those that are not present in prefs.js use their default value.

Alternate Data Stream

When you download a file from the Internet using Firefox 3 to an NTFS drive, it will create an Alternate Data Stream (ADS) associated to that file called Zone Identifier. This ADS will contain evidence that the file originated from the Internet as it will contain:

```
[ZoneTransfer]
ZoneId=3
```

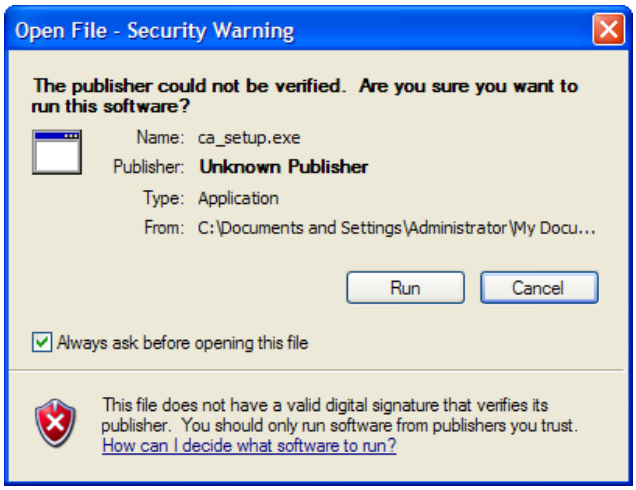
as noted in the screen capture to the right.



The ADS Zone Identifier will be follow the original file if the downloaded file is moved/copied to

another NTFS formatted partition. If you move the original file to a file system that does not support Alternate Data Streams (such as any of the FAT file systems) you will lose the Zone Identifier ADS.

Microsoft Windows XP will use this ADS to determine if a file originated from an untrusted zone (i.e. the Internet). When you double click on a file for which an ADS exists as in the example herein, you will get the following warning screen:



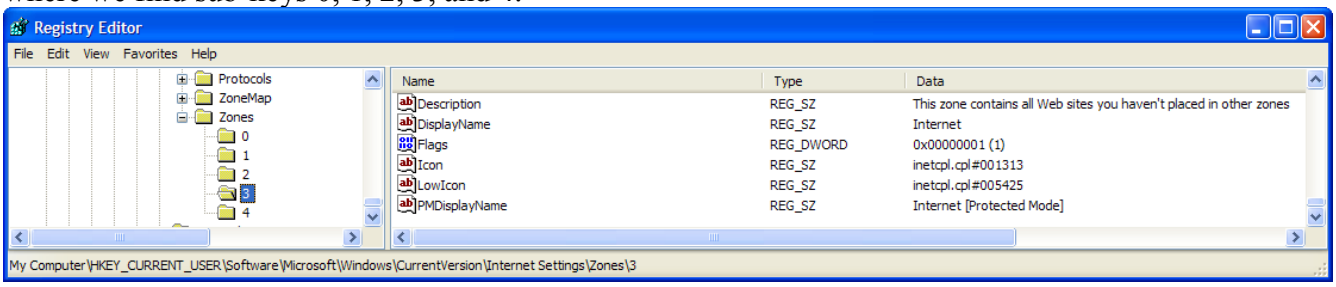
If you un-check the option to "Always ask before opening this file", the Zone Identifier ADS will be deleted. From that point on you will no longer see that warning window because the ADS will no longer exist. The file will be considered as trusted.

If you are using ProcMon to monitor this activity, it will pick up the creation of the ADS when the file is downloaded, and the deletion of it when you un-check the noted check box.

Forensic Value

You can search through your case for all Zone Identifier ADS by either sorting on the extension, or searching for an appropriate keyword such as ZoneTransfer. This will allow you to determine which files originated from the Internet (ZoneId=3). Combine that with downloads.sqlite, and/or its location (downloads folder or in a folder with a number of other files with this ADS) and you have pretty strong evidence that a file came from the Internet (or an Intranet in the case of a corporate investigation where the file came from the internal network).

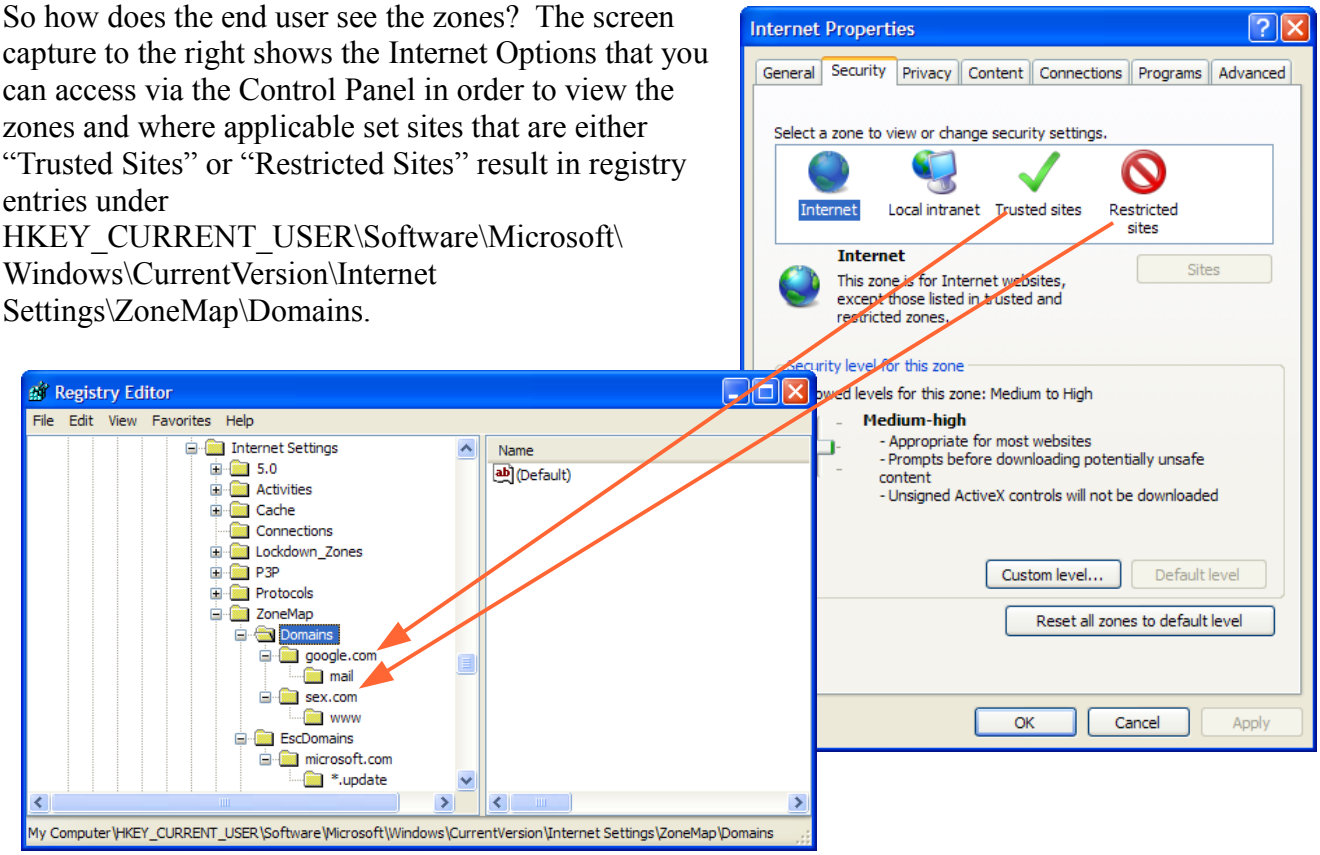
But how do we know Zone 3 is the Internet? The Registry will tell us this. We can find this at: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones where we find sub-keys 0, 1, 2, 3, and 4.



We can also find evidence of these zones for the system (if absent from the user as above) at: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones

The other zones are described as follows:
0 - Your computer
1 - This zone contains all Web sites that are on your organization's intranet.
2 - This zone contains Web sites that you trust not to damage your computer or data.
4 - This zone contains Web sites that could potentially damage your computer or data.

So how does the end user see the zones? The screen capture to the right shows the Internet Options that you can access via the Control Panel in order to view the zones and where applicable set sites that are either "Trusted Sites" or "Restricted Sites" result in registry entries under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\Domains.



NOTE: The use of the Zone Identifier ADS appears to have started in Firefox 3 (or perhaps one of the last versions of Firefox 2 such as 2.0.016-18 or thereabouts – not tested by writer). Internet Explorer (starting with at least version 7, likely even before that) also uses this Zone Identifier ADS for files downloaded from the Internet and populates it identical to Firefox 3.

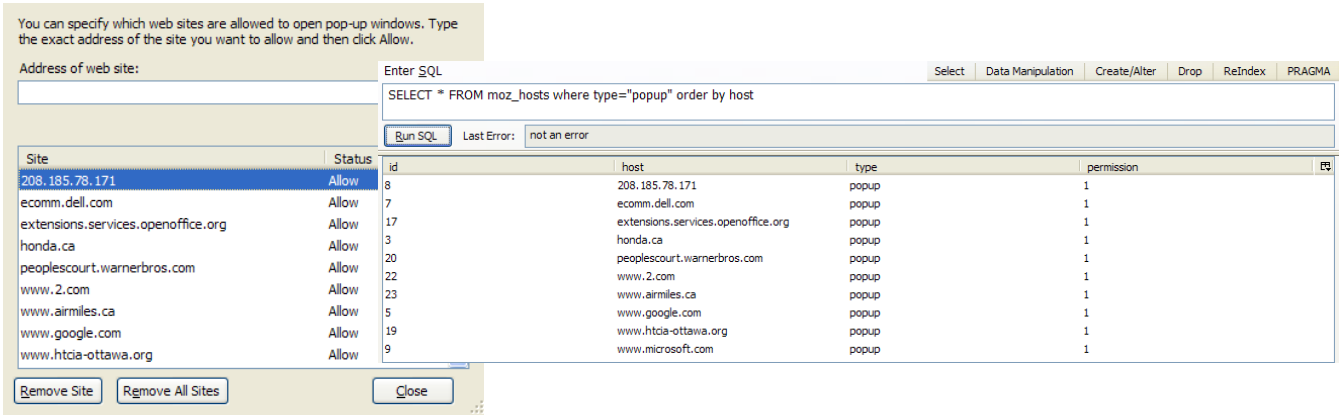
Chapter 9

Popups

Firefox Menu	• Tools, Options, "Content" Tab
Path of Relevant Files	• C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\permissions.sqlite • C:\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\prefs.js (settings for a specific user if set will be found in user.js)
about:config entries	a dom.disable_open_during_load (default – true) b privacy.popups.showBrowserMessage (default – true)
Useful sqlite statements	• SELECT host, permission AS "1=Allow" FROM moz_hosts WHERE type="**popup**"

a If set to true (default), popups will be blocked.
b Used to determine if the bar appears atop your browser to advise of a popup attempt. If false, will only have the icon in the lower right corner.

In the screenshots below we see the popup exceptions in Firefox (on the left) and the associated entries in permissions.sqlite (on the right).



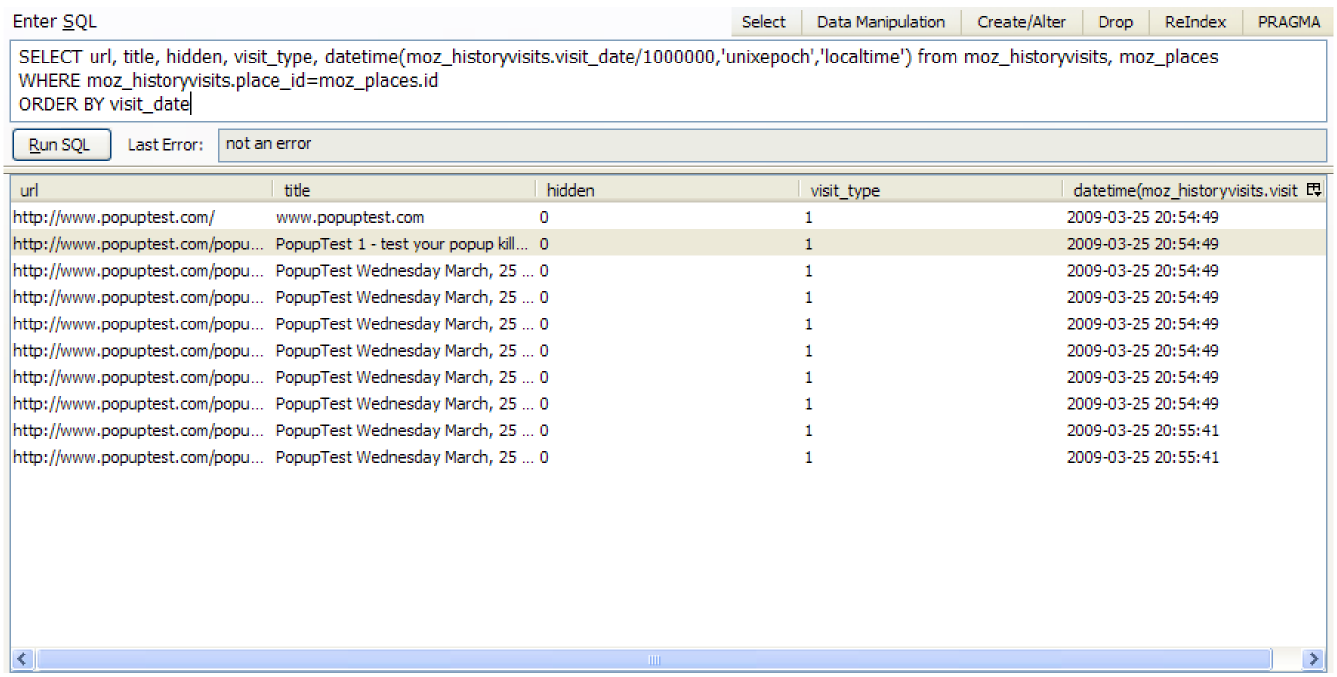
Popup Defense

One possible defense that a suspect can avail of when it comes to images in cache is the popup defense. Claiming that popups were responsible for those images. A forensic examiner must determine if that is a legitimate defense.

Evidence of popup at work
what is created in places.sqlite? They appear as normal traffic in moz_places (not type=hidden). Also appears as normal traffic in moz_historyvisits (type=1) however identical times. Identical times can

also be produced if a user navigates to a bookmark folder and elected to open all bookmarks from that folder to multiple tabs.

Set your local machine timezone to the suspect's timezone.
SELECT url, title, hidden, visit_type,
datetime(moz_historyvisits.visit_date/1000000,'unixepoch','localtime') from moz_historyvisits,
moz_places
WHERE moz_historyvisits.place_id=moz_places.id
ORDER BY visit_date



url	title	hidden	visit_type	datetime(moz_historyvisits.visit
http://www.popuptest.com/	www.popuptest.com	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest 1 - test your popup kill...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:54:49
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:55:41
http://www.popuptest.com/popu...	PopupTest Wednesday March, 25 ...	0	1	2009-03-25 20:55:41

Chapter 10

Form Data

(DRAFT)

related about:config entries
Where is it
format
select statements
unallocated? Impact of privacy settings

Chapter 11

Passwords

(DRAFT)

related about:config entries
Where is it
format
select statements
tools
unallocated? Impact of privacy settings

Chapter 12

Private Browsing

(DRAFT)

Private browsing will be a standard feature as of Firefox 3.1 (currently in beta). Testing with the beta confirms that when using private browsing, there appears to be nothing at all written to the hard drive (not even sessionstore.js information on open tabs).

A RAM dump & crude analysis yielded evidence of the URLs in RAM along with other URLs. Further testing will have to be done in a more controlled/documented fashion to confirm the presence of same in RAM and hopefully with the help of existing memory analysis tool be able to effectively extract it to a meaningful format.

Appendix "A"

Summary of Practical sqlite Statements

(DRAFT)

SELECT * from moz_historyvisits, moz_places
where visit_date between 1217706724812500 and 1220212324000000
and moz_historyvisits.place_id = moz_places.id

Typed URLs

SELECT * from moz_historyvisits, moz_places
where typed=1
and moz_historyvisits.place_id = moz_places.id

to only see fields of interest (i.e. URL and typed)...
SELECT url, typed as "typed URLs" from moz_historyvisits, moz_places
where typed=1
and moz_historyvisits.place_id = moz_places.id

as above, but sorted in descending order by visit count...
SELECT url, typed as "typed URLs", visit_count from moz_historyvisits, moz_places
where typed=1
and moz_historyvisits.place_id = moz_places.id
order by visit_count desc

as above, but only how sites visited at lest 10 times, and group by URL (so we don't see the multiple visits to the same URL).
SELECT url, typed as "typed URLs", visit_count from moz_historyvisits, moz_places
where typed=1
and moz_historyvisits.place_id = moz_places.id
and visit_count>9
group by url

Including a constant for the preceeding text to the url.
SELECT 'This is a typed URL', url from moz_places where typed=1

However all of the above simply looks in moz_places which will only contain one entry per URL and no associated date. Actual visits are stored in moz_historyvisits therefore to see all instances where a URL was typed, you need to query both tables as follows:
select moz_places.id, url, title, visit_count, visit_date from moz_places, moz_historyvisits
where typed=1
and moz_historyvisits.place_id=moz_places.id

With the above you find all history entries in moz_historyvisits where its corresponding moz_places record shows that it was typed.

BIG CAVEAT!! You can navigate to a site a dozen times without typing it, resulting in a dozen history entries and the one places entry. However if on the 13th time you type it in, the record in moz_places for that URL will now be flagged as typed=1. The above query will give the impression that all 13 entries in history were typed. Meanwhile it was just the 13th one.

Sites visited between 5 and 10 times:
SELECT * FROM moz_historyvisits, moz_places
where visit_count between 5 and 10
and moz_historyvisits.id=moz_places.id

to find all URLs with the word canada in it (not case sensitive)
select * from moz_places
where url like "%canada%"

Find all entries in moz_historyvisits, moz_places with the word Canada in the url (not case sensitive).
SELECT * FROM moz_places, moz_historyvisits
where url like "%canada%"
and moz_historyvisits.place_id=moz_places.id

moz_places has no dates in it. No URLs repeat.
moz_historyvisits has dates associated to it. A reference to a URL in moz_places can be in moz_historyvisits more than once as a user can re-visit a site several times. Each time resulting in an entry in moz_historyvisits but not necessarirly a new entry in moz_places because that URL may already exist in there.

Creating a view:
create view "typed_URLS" AS select id, url, title, visit_count from moz_places where typed=1
using a view:
select * from typed_URLS
From within sqlite manager you can then export the result of that view to a file.

Appendix “B”

Glossary of Terms

Appendix "C"

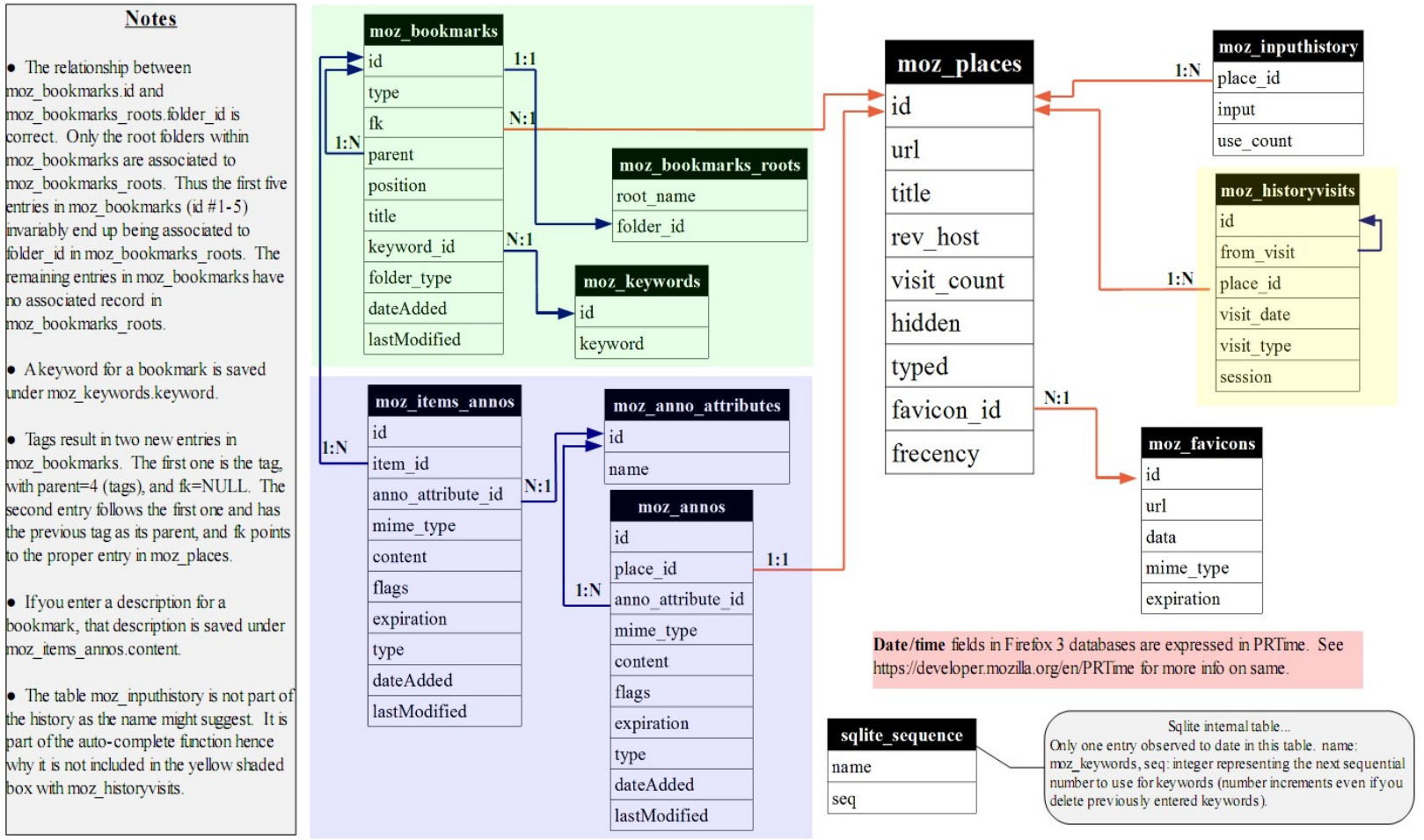
Add-ons & Extensions

Google toolbar (DRAFT)– Has an option to generate thumbnails for each page visited.
If that option is selected (default?), will result in a lot of entries in the table moz_annos in places.sqlite.
See http://forums-test.mozillazine.org/viewtopic.php?f=38&t=1144415&start and
http://groups.google.ca/group/mozilla.support.firefox/browse_thread/thread/d585ec58ec0b9354 for info about it. Impacted by privacy settings?

Foxmarks – evidence of same being used, username & password for same stored in signon3.txt (same place as all other usernames/passwords saved by Firefox), last sync time (may be kept by server).

Appendix “D”

places.sqlite



See https://wiki.mozilla.org/Places:Design_Overview#Models for online version of this document.

Appendix “E”

cookies.sqlite

Appendix “F”

downloads.sqlite

moz_downloads
id
name
source
target
tempPath
startTime
endTime
state
referrer
entityID
currBytes
maxBytes
mimeType
preferredApplication
preferredAction
autoResume

- id – sequential number. Will grab the next available number at the end of the sequence. If there is a gap in the middle from removing a file from the list, that number does not get used. If an entry is removed at the end of the list, that number will be re-used.
- name – name of the file downloaded.
- source – full URI/filename of file's origin.
- target – full path/filename where file was saved.
- tempPath – temporary path where a file gets downloaded when selecting to open it (i.e. A large .pdf file) pending the entire file being fully downloaded (so you'd see .part at the end of the temp file).
- startTime – PRTime (https://developer.mozilla.org/en/PRTime) when download was started. This time is expressed in UTC time and is obtained from the client's machine that is doing the downloading. If a download is canceled but then re-started from the download dialog window, the startTime will be updated with the re-start time.
- endTime – Much like start time, except it's when the download was completed, paused (if download not completed), or canceled. Where a download is paused, and then later canceled without ever first resuming it, the pause date will be reflected, not the canceled date. A re-started download will change the endTime to the new paused, canceled, or completed download time.
- state – 0 (download in progress), 1 (download complete), 3 (download stopped), 4 (download paused)
- referrer – appears to be the URI that directed the browser to the downloaded file. Some have a NULL value. More testing required to make assertive statements on this field.
- entityID – only observed an entry in this field if a download was in a paused state. More testing required to determine the significance of this field.
- currBytes – current number of bytes downloaded
- maxBytes – logical size of the source file in bytes
- mimeType -
- preferredApplication – application that was used to open the downloaded file (typically will be Firefox unless a user selects to open the file instead of save it, and specifies a particular application to use to open it with.
- preferredAction – values noted to date are 0, 2, and 4. More testing required to determine significance.
- autoResume – only value noted for this field was 0 even for a paused download. More testing required.

Appendix “G”

formdata.sqlite

Appendix “H”

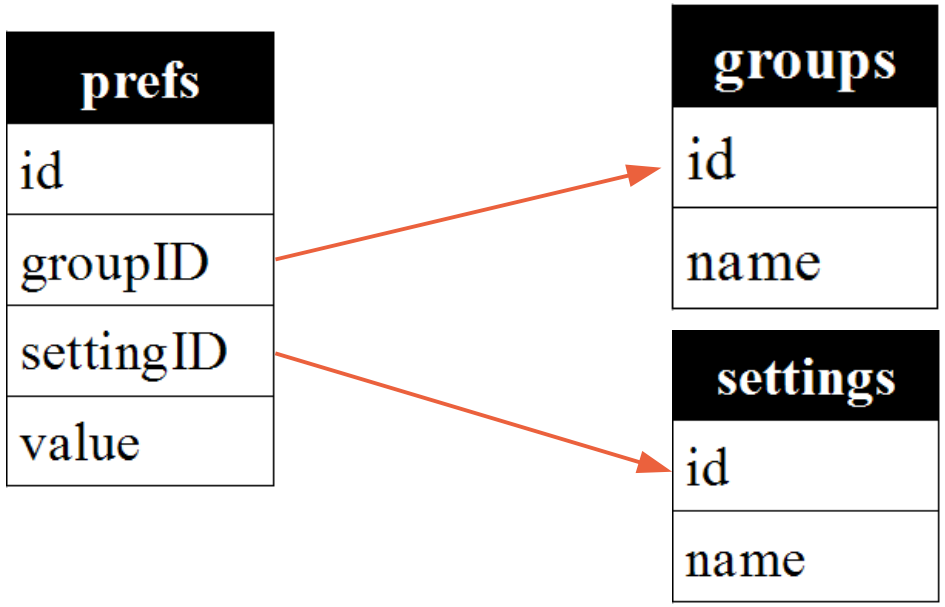
permissions.sqlite

moz_hosts
id
host
type
permission

This database only has one table. It contains permissions for installing of add-ons (type=install), cookies (type=cookie), pop-up blocking (type-popup), and images (type-image).

- popup - 1 = allow
- cookie - 1 = allow
  2 = block
  8 = allow for session only
- install - 1 = allow (do not warn when a site tries to install an add-on)
- image - 1 = allow (load images automatically even if the check box “Load images automatically” is not set {about:config entry - permissions.default.image}
  2 = block (block images for particular sites even if the option is selected to load images automatically)

content-prefs.sqlite



This database contains preferences specific to a site (i.e. size of text via View, Zoom, ...).

- **groups.name** contains the URI of the site for which a site specific preference is set.
- **settings.name** contains the description of the setting.
- **prefs.value** contains the value for the setting in question.

These entries do not get cleared when a user clears their private settings. Therefore good place to look for evidence that a user may not even realize they left behind.

Appendix “J”

search.sqlite

Appendix "K"

PRTime/Epoch Time

(DRAFT)

Firefox 3 stores its dates/times within the various sqlite databases in PRTime/Epoch time.

From: https://developer.mozilla.org/en/PRTime:

"This type is a 64-bit integer representing the number of microseconds since the NSPR epoch, midnight (00:00:00) 1 January 1970 Coordinated Universal Time (UTC). A time after the epoch has a positive value, and a time before the epoch has a negative value.
In NSPR, we use the more familiar term Greenwich Mean Time (GMT) in place of UTC. Although UTC and GMT are not exactly the same in their precise definitions, they can generally be treated as if they were."

To search for a range of dates, use the converter at www.epochconverter.com to convert from standard date to epoch date. Note that they also have a batch converter option, and a quick converter option for daily, monthly, or yearly time frames. You must then add six 0's after the epoch time produced by the online converter (except for cookie expiry date, then use as converted). This will allow you to determine the epoch times to use in your select statement when seeking activity for a given date range.

You can also use DCode (http://www.digital-detective.co.uk/freetools/decode.asp) to convert from Epoch time to regular time. If converting from epoch time to standard time, you must either use DCode, or truncate the last six digits from the value in Firefox in order to have it convert properly by the online converter. In comparing the values returned by the online converter when providing the full 16 digit number from Firefox vs truncating the last six digits we note that they produce different results. However when using DCode with the full 16 digit number it does produce the same date as the online converter when using 10 digits. As well if you convert a 16 digit epoch number using the online converter and then convert the date/time it returns back to epoch, it does not match the starting number. But if you truncate the last six digits off then it produces consistent results as you convert to and from epoch, as well as consistent results with DCode.

To query for history entries between Sat, 02 August 2008 15:52:04 -0400
and Sun, 31 August 2008 15:52:04 -0400. When using the epoch converter above, make sure to check **GMT** box, and enter the first 10 digits of the stored date/time to convert. Or when converting to epoch time, add 6 digits (all 0's) to make it 16 digits in length as per the stored value. And where Firefox 3 stores these dates/times in GMT format you must convert your date/time range to GMT before converting it to Epoch. In other words for the above example where the time zone is -0400, the GMT time is 19:52:04 (19:52:04GMT – 0400 Time Zone offset= 15:52:04 local time).

Most of the dates in the Firefox 3 databases have six digits beyond the standard Epoch time. Those dates do not properly decode using www.epochconverter.com unless you remove the last six digits.

Remember to pad the epoch times you obtain from www.epochconverter.com with six 0's at the end for

your sqlite statements where applicable (for the fields that use the longer epoch time). Failing to do this will not produce the expected results as www.epochconverter.com returns a 10 digit number but Firefox uses a 16 digit number. Thus looking for entries between the two 10 digit numbers returned by epoch converter will never match the 16 digit numbers in Firefox. But by padding those 10 digit numbers with six 0's at the end will give you a sixteen digit number without altering the intended date range thus properly hitting on entries between the desired dates.

CAUTION – if you reverse the date codes in your "between" statement, putting the later date (larger number) first followed by the earlier date, it will not match anything but will not produce an error.

Sample select statement to do conversion on the fly (important caveat – set your local machine to the suspect's time zone so that the "localtime" will be calculated according to the suspect's timezone).

```
SELECT url, title, hidden, visit_type,
datetime(moz_historyvisits.visit_date/1000000,'unixepoch','localtime') from moz_historyvisits,
moz_places
WHERE moz_historyvisits.place_id=moz_places.id
ORDER BY visit_date
```

Appendix “L”

Intro to sqlite

sqlite date/time functions - http://www.sqlite.org/lang_datefunc.html

Appendix “M”

sqlite Browser

Appendix "N"

sqlite Manager

Firefox has an extension called sqlite Manager. It is available at https://addons.mozilla.org/en-US/firefox/addon/5817. It was developed as a result of an initiative by Google. You can get the source code for it at http://code.google.com/p/sqlite-manager/.

Versions of Firefox 3 prior to 3.0.6 allowed the sqlite manager add-on to view the contents of all the databases (and even make changes to them) for the current profile. As of version 3.0.6 (possibly even earlier but first noted in 3.0.6) Firefox something changed which prevented the add-on from viewing some of the databases for the current profile while Firefox was running. However this was corrected during a routine update of Sqlite Manager add-on.

From a forensic analyst perspective that is not really an issue as you'd be examining suspect's sqlite databases copied out of the image file, not those of the currently running Firefox profile. However when trying to test things out in Firefox, if such a limitation returns it will require an examiner to either use the external Sqlite Manager application (as described later in this appendix) or a second Firefox profile with the add-on (also described in this appendix).

This means you must examine these database files when Firefox that owns the sqlite files to be examined is not running. This is not a problem in your forensic environment obviously because you'll be copying out the suspect's sqlite files. In that case you can still use the add-on in Firefox and then open the sqlite files that you exported as those would not be locked.

Therefore you have a few options open to you if this problem returns in a later upgrade. You can do whatever actions you want in Firefox and then exit it to unlock the files which you can then examine using either Sqlite Browser from Sourceforge, using Sqlite Manager via Xulrunner (a Mozilla runtime package that is also used for Firefox 3), or examining them from within an second Firefox profile that has sqlite manager installed as an add-on.

Sqlite Browser from Sourceforge is covered in another appendix so you can see it if you want to use that solution. Personally I prefer Sqlite Manager so my solutions are to install Xulrunner to run the tool outside of the browser, and to create a second profile as well.

Xulrunner/Sqlite Manager

To use Sqlite Manager as a standalone application, you must install Xulrunner. The main website for Xulrunner is at https://developer.mozilla.org/en/XULRunner. You download the tool from http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases/, selecting the latest release (1.9.0.7 at the time of this writing). Click on the folder for the latest version, and then on the folder called runtimes. You will see one called xulrunner-1.9.0.7.en-US.win32.zip (of course the version number will vary) for MS Windows, and versions for Linux as well. Download it and extract it to a folder (it

will run from that location, it does not install).

Next download Sqlite Manager from Google's link provided above. Download the version that has XR in its name (again, this is as of the time of writing – read the page to see if Google changes this current naming convention), which currently is SQLiteManager_XR_0.4.7.zip. Extract that to a folder and the application will then run from that folder (it does not install).

The last step is to then create a shortcut on your desktop (or your quick launch or in your Start menu, as you wish). The target for the shortcut will look something like below (assuming you extract the zip files in the respective folders):
"C:\Documents and Settings\Administrator\xulrunner-1.9.0.7.en-US.win32\xulrunner\xulrunner.exe"
"C:\Documents and Settings\Administrator\Sqlite Manager\application.ini"

Although the above is on two lines, it's really one line that wrapped because of the width of this page. Notice that the path for xulrunner.exe is in quotes, and the path for the sqlite manager application is in quotes (because of spaces in the path). Of course you'll have to change your paths to match where you unzipped the respective downloads. The above reflects where I extracted them.

You are now good to go. Selecting that shortcut should start sqlite manager outside of Firefox. Here again if you try and view a database from a running version of Firefox that locks the databases you'll get the same error. So you still have to shut down Firefox.

Using a Second FF Profile

Another way to circumvent this problem (and possibly an easier way to do it) is to create a second Firefox profile (see section on profiles to accomplish this). I created one called “forensics”. But I didn't start it from within the profile manager because I didn't want to set it as my default profile. I wanted to retain my main profile as my default. So after I created it I canceled out of the profile manager.

The next step is to create a shortcut (again this can be on your desktop, on your quick launch, or in your Start menu). The target for that shortcut will then be
"C:\Program Files\Mozilla Firefox\firefox.exe" --no-remote -p forensics
assuming you have Firefox installed at the same location as me, and you also called your profile “forensics” (give it a different icon so you'll notice the difference). Notice the –no-remote option. What this allows you to do is to start this profile while another profile is also running. So you'll have two copies of Firefox running.

In your main one you can have the sqlite manager add-on. You start up your main profile as you normally do, and then start up the other profile by clicking on the new shortcut. The new profile will have its own bookmarks and other artifacts. So you can do some testing within it and then exit out of that copy of Firefox and then examine the sqlite files from it using the add-on from your main Firefox profile. Because as I indicated they are two separate profiles each with their own profile folder and all associated database files. When you exit out of that profile, the database files are unlocked and can therefore be properly opened.

You can leave that open and re-start the forensic profile to do more testing. While that profile is open, the databases are again locked. But when you shut down, you can go back to the other profile in sqlite manager and refresh the view for that database and you are good to go.
When using multiple profiles running at the same time it can get confusing to remember which window is which profile. Of course your bookmarks are different so looking at those could tell you which is which. Another way to do this is to go in the address bar and type about:cache. This page will give you the path to the cache files. In the path you can see the profile name. A third option is to add some style to Firefox as explained at:
http://groups.google.ca/group/mozilla.support.firefox/browse_thread/thread/2186b29b041a356/9813ba0b168a26a2?lnk=gst&q=multiple+profiles#9813ba0b168a26a2

Forensically Sound Process

Of course as a forensic examiner we want to make sure that our process is forensically sound. In any case where you copy out files from the suspect's image and then analyze the data using another tool you will want to ensure that the analysis did not change the content of the files being examined. In other words, you want to be able to state in court that examining the suspect's Firefox database files using a tool such as Sqlite Manager outside of your forensic tool did not affect the integrity of the data.

This can be easily accomplished by hashing the exported files after you've examined them and comparing their hash values with the hash values of those respective files within your forensic software. By establishing that the hash values did not change as a result of examining the files using Sqlite Manager or some other third party tool you can easily defend your evidence. Failing to do so opens up the possibility of an argument that the tool affected the integrity of the data thus the results cannot be relied upon in court.

Appendix “O”

Safe Mode

Appendix "P"

sessionstore.js parser

The html code below was found at http://forums.mozillazine.org/viewtopic.php?t=622036 under a posting dated March 26, 2008 by a person using the nickname ZeePrime. There is a posting after that one where the poster advises that he has a sessionstore parser on his site. However that page no longer exists. The html code below however works very well. A minor modification was made to not only display the name displayed for a URL, but the URL itself as well. Thanks to Kelvin BEAR as well as Dan

```
<html><head>
<style type="text/css">
a.selected, li.selectedtab { color: blue; font-weight: bold; }
li.closedtab { color: red; text-decoration: line-through; }
</style>
</head><body>
<ul id="windows">
<script type="text/javascript">
function dowindow(wind,wname,wclass) {
document.write("<li class=\"" + wclass + "\">");
document.write("<b>" + wname + "</b>");
document.write("<ol class=\"tabs\">");
var sel = wind["selected"];
var tabs = wind["tabs"];
for (var j = 0; j < tabs.length; j++) {
var tabclass = "tab"
if (j + 1 == sel) tabclass = "selectedtab"
dotab(tabs[j],"Tab #" + (j+1),tabclass);
}
var tablen = tabs.length;
tabs = wind["_closedTabs"];
for (var j = 0; j < tabs.length; j++) {
dotab(tabs[j]["state"],"Tab #" + (tablen + j + 1),"closedtab");
}
document.write("</ol>");
document.write("</li>");
}
function dotab(tab,tname,tclass) {
document.write("<li class=\"" + tclass + "\">");
document.write("<b>" + tname + "</b>");
document.write("<ul class=\"urls\">");
var sel = tab["index"];
var urls = tab["entries"];
for (var j = 0; j < urls.length; j++) {
```

```
var urlclass = "url"
if (j + 1 == sel) urlclass = "selectedurl"
dourl(urls[j],urlclass);
}
document.write("</ul>");
document.write("</li>");
}
function dourl(url,uclass) {
document.write("<li class=\"" + uclass + "\">");
var uname = url["title"];
if (!(uname) || uname == "") uname = url["url"];
document.write("<a href=\"" + url["url"] + "\">" + uname + "</a>" + "<br>" + url["url"]);
document.write("</li>");
}
var data = <PUT CONTENTS OF sessionstore.js HERE! (replace <> as well)>;
var windows = data["windows"];
for (var i = 0; i < windows.length; i++) {
dowindow(windows[i],"Window #" + (i+1),"window");
}
</script>
</ul>
</body></html>
```

Appendix "Q"

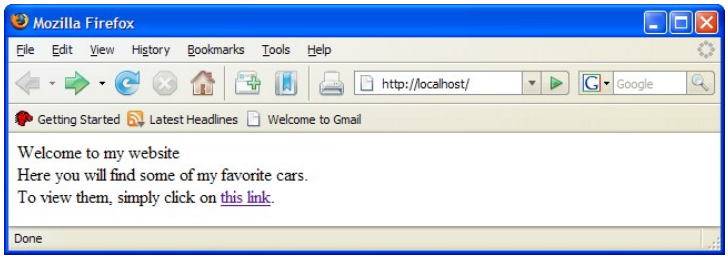
Prefetching

Prefetching

Firefox Menu	Nil. Not currently configurable via options menu. Only via about:config menu.
about:config Entries	network.prefetch-next (default - True)
Path of Config File	C\Documents and Settings\{user}\Application Data\Mozilla\Firefox\Profiles\{profile name}\prefs.js (settings for a specific user if set will be found in user.js)
Possible Live Registry Entry	Nil

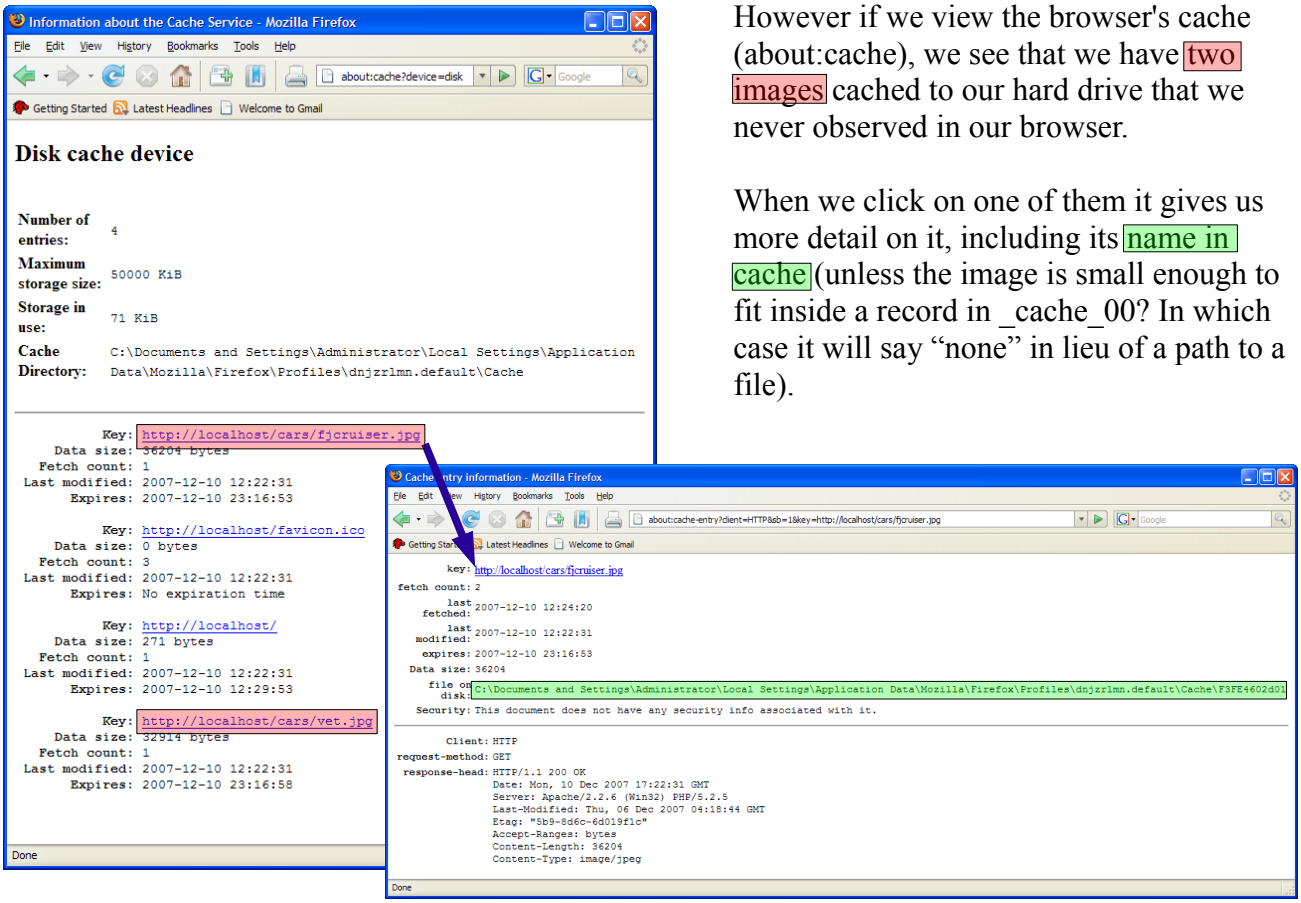
- Prefetching can be used by a web server to send information to a browser client that supports prefetching (such as Firefox) before it's even requested. An example of this would be if you navigated to a site which had a PDF. The server could send a prefetch link to your browser causing the PDF document to be prefetched by your machine during idle time in anticipation that you will be clicking on it. This way when you do click on that PDF it will appear to load much faster whereas in actual fact it will already be in your browser's cache thanks to prefetching.

This feature presents forensic examiners with a challenge. How can you be certain that an image or other document found in Firefox's cache is not the result of a prefetch. To determine this, we must first explore prefetching a bit more and better understand how it works. To illustrate this example I tested this on a webpage on my localhost. I open Firefox and clear the cache, then navigate to localhost as below:



In navigating to this site our browser does a GET for the html page (index.html). If we view the source for this page we note that there are two prefetch commands.

In the source code (to the right) for the above we see the prefetch commands for the two .jpg files located in a sub-folder called cars. Those images are now in our cache but as a end user we have never viewed them.



However if we view the browser's cache (about:cache), we see that we have two images cached to our hard drive that we never observed in our browser.

When we click on one of them it gives us more detail on it, including its name in cache (unless the image is small enough to fit inside a record in _cache_00? In which case it will say "none" in lieu of a path to a file).

This means that a forensic examiner who runs signature analysis will properly see those photos in cache and could conclude (incorrectly) that the user has viewed those photos when in fact he has not.

But let's not get ahead of ourselves. First things first...

In a typical investigation your will want to search through the three cache files (_001-3) for one of the following three terms (so use them as a keyword list) that can be used to cause prefetching.

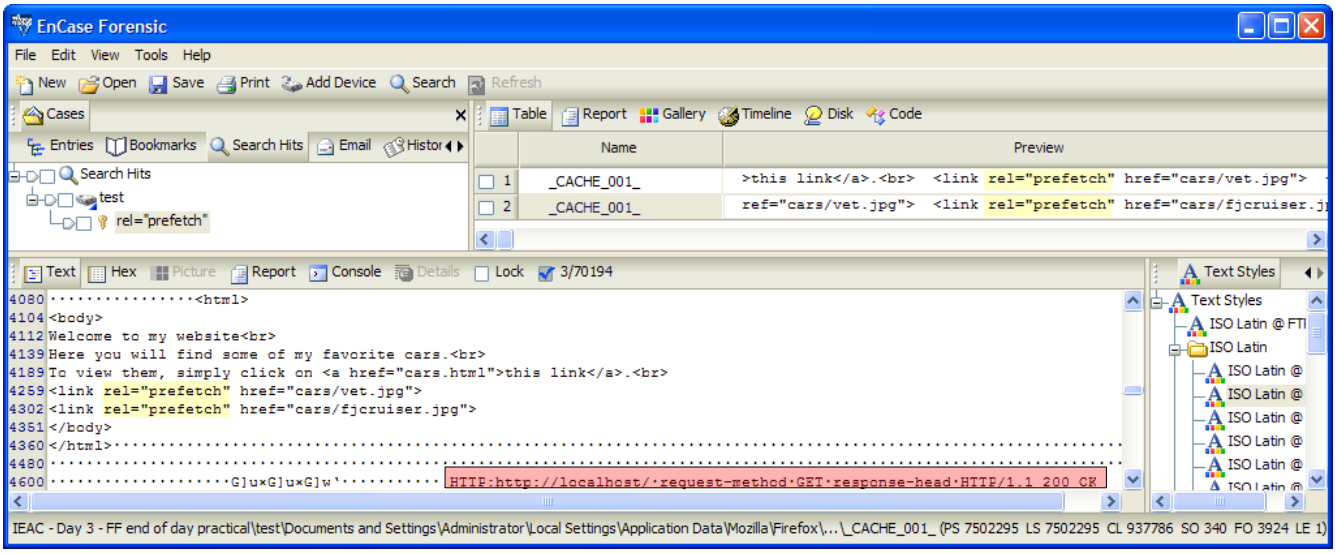
- rel="prefetch"
- rel=prefetch
- rel="next"

If you do get a hit, then you proceed to the next step, determining which images were prefetched (to see if any of them are the ones of concern to you.

If you don't get a hit for prefetching, it means it was not used in relation to the contents of cache (unless prefetching can be accomplished differently than noted herein, recognizing that programming languages evolve and embedded languages can be used in an HTML document thus availing of the abilities of that language). You can therefore reasonably conclude that none of the images in cache were created as a result of prefetching.
Furthermore prefetched items will not result in history activity. Therefore if you find a URL for an item in cache then that item was not the result of prefetching.

What was prefetched?

We run our search and hit on the term (see below). Now what.



Well now that we know we have to deal with prefetching, we need to determine which cached entries relate to those prefetched images. We see above the HTML page with the prefetch in it. Below that is the information on the GET that was used to grab that HTML page. There is yet further info just below that on the GET, but what we are concerned with is at to bottom of that lower pane. We can see from which host that page came from (http://localhost/) and that the GET was successful (200 OK).

In the HML code we see that the two prefetched items are .jpg images, and both are in a sub-folder called cars. So they are located at http://localhost/cars/ and bear the names vet.jpg and fjcruiser.jpg.

Following that we can see the GETs for the prefetched images (and you'd see GETs for the non-prefetched images as well but since this page has none, the only GETs we see are for prefetched images). Note that we see the full path to those files and that the GETs were successful (200 OK), the date, file size (Content-Length), and Content-Type.

```
5055 ·····G]u×G]u×G^·:··€'···‡···HTTP:http://localho
5175 st/cars/vet.jpg·request-method·GET·response-head·HTTP/1.1 200 OK
5241 Date: Mon, 10 Dec 2007 17:22:31 GMT
5278 Server: Apache/2.2.6 (Win32) PHP/5.2.5
5318 Last-Modified: Thu, 06 Dec 2007 04:17:57 GMT
5364 Etag: "5ae-8092-6a2c1d21"
5391 Accept-Ranges: bytes
5413 Content-Length: 32914
5436 Content-Type: image/jpeg
5462 ·····
5582 ·····▯·····G]u×G]u×ÿÿÿÿ·····-···MissedIconCache:http://localhost/f
5702 avicon.ico·Icon·Missed·····
5822 ·····
5942 ·····
6062 ·····'·····G]vDG]u×G^·5··▯l···)····HT
6182 TP:http://localhost/cars/fjcruiser.jpg·request-method·GET·response-head·HTTP/1.1 200 OK
6271 Date: Mon, 10 Dec 2007 17:22:31 GMT
6308 Server: Apache/2.2.6 (Win32) PHP/5.2.5
6348 Last-Modified: Thu, 06 Dec 2007 04:18:44 GMT
6394 Etag: "5b9-8d6c-6d019f1c"
6421 Accept-Ranges: bytes
6443 Content-Length: 36204
6466 Content-Type: image/jpeg
6492 ·····
```

So we know that the images were properly prefetched. And the file size will match what EnCase has for those temp files. The Date: information matches the file created time in EnCase for the associated cache file.

What are the cached files relating to those prefetched images?

So how do I determine what cached entries relate to those? Well you could do some manual decoding or use a third party tool. However an easy solution is to use Firefox itself.

Start with a machine which has no Internet access and ensure that the cache is already cleared. Export the suspect's cache out of EnCase and into the cache location of your sandbox system. Start up Firefox and type about:cache and click on the disk cache which will give you a screen same as on the previous page.

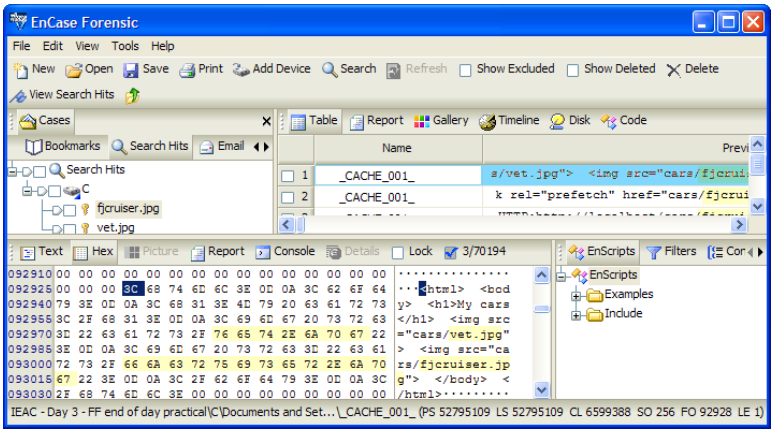
Search for the prefetched file names (fjcruiser.jpg and vet.jpg in this example). For each of those click on the entry to bring up the more detailed screen (also on the previous page) telling you the file name of the cached file. If those prefetched files in cache don't match the images you've bookmarked as evidence then you are done. You are not concerned about prefetching of stuff that is not being relied upon as evidence. Just that what you are relying upon was NOT prefetched.

What if one of those images is considered evidence?

If one of those prefetched files contain is considered evidence then you must proceed to the next step, determining if the suspect subsequently navigated to the web page where the prefetched images would have been viewable on the screen.

We accomplish this by building a keyword search list using the prefetched files. For our example that would be fjcruiser.jpg and vet.jpg. We then run a search on the three cached files (_001-3) and examine the hits. If the only hit we have for an image is the one from the prefetch command it tells us that the individual did not navigate to view the images that were in all likelyhood unknowingly prefetched (short of a user checking their cache entries or viewing the source of the page to see that prefetching was being done). If we do hit on the same filename elsewhere in any of the three cache files we must examine it to determine if it's our prefetched file now being properly accessed or if it's another file bearing the same file name but coming from another host.

To illustrate this we have a hit where we did navigate to the page. Note the HTML code. We have the same sub-folder and file names as the prefetched images. Now we simply need to confirm from which host this came. It must have come from localhost (same as the prefetched one) in order to be the previously prefetched images now being viewed by the user.



The challenge

The previous EnCase screen capture looks promising, but without being able to demonstrate that this came from localhost as did the prefetched ones we cannot yet say with 100% certainty that the user subsequently navigated to the page that displayed the previously prefetched images.

One point to remember when dealing with cache. If something is in cache (be it an image, a pdf, whatever), then so is the HTML code for the webpage responsible for that cached entry. When you clear cache, you cannot selectively do so (at least not from within FF). Therefore if a person clears the cache, all of it is cleared – HTML pages, jpg, pdf, etc...
It is because of the fact that the HTML code will exist along with a cached item that we can effectively examine cache to account for possible prefetching.