



More ▾ Next Blog»

Create Blog Sign In

A Renaissance Security Professional

**Alex**

Bay Area, CA, United States

I'm a computer security professional, most interested in cybercrime and computer forensics. I'm also on Twitter @bond_alexander All opinions are my own unless explicitly stated.

[View my complete profile](#)Showing posts with label **firefox 4**. [Show all posts](#)

Thursday, May 5, 2011

Firefox 4 Browser Forensics, Part 5



Recommend this on Google

We're nearing the end of my series on Firefox 4 forensics ([click here](#) for the full list). Media coverage has finally started to make people aware of how much their online behavior is tracked, and the addition of "Private Browsing" modes in all major browsers is making browser anti-forensics easier than ever. This means we'll probably encounter it in our investigations.

First, I'll cover actions that prevent the creation of artifacts: turning "Remember History" off and using "Private Browsing" mode. Then I'll cover various some methods of destroying artifacts that have been created. I won't be covering third-party products.

Preventative antifoensics

To test "Private Browsing" mode, I activated private browsing, searched for Nmap and downloaded the latest version and then closed Firefox. First, I wanted to see if the page was listed in the browser history, so I opened places.sqlite and queried: "select * from moz_places where url like '%nmap%';" No result. Same with searching for 'input' in typed_urls, no cookies from the domain and nothing in the download history either. However, the google search for "nmap", many nmap images, the websites <http://nmap.org>, and <http://nmap.org/download.html> all appear in the browser cache with the appropriate timestamps and fetch count. This, plus having the creation time of the downloaded file, tells us exactly what the user did and when.

Turning off browsing history is pretty easily done, it's front-and-center on the "Privacy" tab in options. Default is "Remember history", but there are custom history settings as well as just "off". To test the artifacts, I turned off browsing history, googled "metasploit", and downloaded the latest version. As is expected, nothing is appearing in moz_places (browsing history). Nothing's showing up in the cache or the download history, so oddly enough turning off browsing history protects privacy better than "Private Browsing" mode. That means the only possibilities for detection are outside of Firefox, such as using the operating system to track who was logged in when the downloaded file was created and who executed it.

Note, this is accurate at the time of writing, for the current version of Firefox (4.0.1). Once this is made known, it's entirely possible that any of these behaviors will change. You should always run your own testing to confirm behavior before trusting it in a case.

My Blog List



SANS Internet Storm Center, InfoCON: green
ISC StormCast for Wednesday, February 4th 2015
<http://isc.sans.edu/podcasts/detail.html?id=4341>,
(Wed, Feb 4th)



Threat Level
Silk Road Defense Makes Its Final Pitch: Don't Trust Internet Evidence



Ars Technica - Law & Disorder
Silk Road trial closes: "It's a hacker! It's a virus! It's ludicrous."



Cybersecurity Report
'Connect the Dots' -- An Alternative to NSA Bulk Surveillance?



Krebs on Security
Banks: Card Thieves Hit White Lodging Again



Law of Data Security & Investigations
Blockchain Smart Contracts | Fraud, Taxes & Evidence



Graham Cluley's blog
"Exploit This": Evaluating the exploit skills of malware groups

Evidence destruction

But what if the target of our investigation didn't know in advance that he needed to cover his tracks? Firefox has several options to remove recorded data, from the selective to the blunt.

The most selective way to remove data is through the history pane. If you open the history pane and right-click on a history item, you can select "forget this site". Let's imagine this is a "violation of policy" case: browsing porn at work. I browsed to www.pornhub.com and started a video streaming to get a good cache. Opening up the history, it looks like Pornhub connected to several other porn sites, so if our suspect didn't make sure to forget all of the relevant sites there would still be evidence of their illicit browsing. In this case, however, I'm going to make sure and forget about all of them. After "forgetting" all the sites, there are no traces left in `places.sqlite`. There's evidence that sites were forgotten because of the gap in `id` numbers, but no indication of what was formerly there. Interestingly, using "forget this site" completely destroys the cache, but only removes the selected site(s) from the browsing history. This is a clear sign of evidence destruction, and the deleted cache files could likely be recovered from unallocated space or from backups (such as Volume Shadow Copy).

If any of the databases are deleted, Firefox will automatically create a new empty copy of it the next time it's run. Normally, the databases will have a modified date of the last browsing event, but a creation date of when Firefox was originally installed. The creation date is not even modified when Firefox is upgraded or the history is "forgotten" through the browser options. Therefore, if the creation date of the tables is more recent than the creation date of core Firefox files (such as `firefox.exe`), it's a clear indication that the table was deleted around the creation date of the existing table. It may be recoverable through standard means.

Directly modifying the databases would be somewhat more difficult to detect. The databases are modified constantly through regular browsing, so the timestamps wouldn't be a clue. However, like "forgetting this site", there will be a gap in the normally sequential `ID` numbers that could indicate that something was deleted, and examining the `last_visit_date` of the sites surrounding the gap might allow you to determine when the missing sites were visited. If backups of the databases exist, they might have the missing data. Also, the cache isn't nearly as user-friendly to edit as a `sqlite` database so if the cache isn't cleared it could provide a clue for what was lost. Even if the cache had been cleared, the deleted files might be recoverable through standard methods.

This isn't meant to be a complete overview of all possible methods of antiforensics with Firefox, just a quick highlight of some possibly relevant issues and how to detect and overcome them. This is the end of my Firefox 4 forensics series, I hope it'll be a useful reference for your investigations. If any of this information turns out to be incorrect or changes in future versions, please let me know and I'll edit the appropriate post.

Posted by [Alex](#) at 9:43 PM No comments:  [Links to this post](#)

Labels: [anti-forensics](#), [browser](#), [firefox 4](#), [forensics](#)

Friday, April 29, 2011

Firefox 4 Browser Forensics, Part 4

  Recommend this on Google

 www.symantec.com/connect/blogs/feed/srblog/2261/all/en

New Adobe Flash zero-day is being exploited in the wild

 [TaoSecurity](#)

A Word of Caution on Fraudulent Routing

 [A Day in the Life of an Information Security Investigator](#)

Videos: BSides Columbus Ohio, USA 2015

 [Google Online Security Blog](#)

Security Reward Programs: Year in Review, Year in Preview

 [Uncommon Sense Security](#)

But Jack, community and stuff...

[Overhack](#)

Daystar Suspension

[SANS Computer Forensics, Investigation, and Response](#)

"What is New in Windows Application Execution?"

[Rational Survivability](#)

Incomplete Thought: The Time Is Now For OCP-like White Box Security Appliances

 [flyingpenguin](#)

Gov Fumbles Over-Inflated Sony Hack Attribution Ball

[Microsoft Malware Protection Center](#)

System Center Endpoint Protection support for

When I started this series, I had no idea it would go on this long. There are more forensic artifacts in FF4 than I thought. It seems like every time I turn around I find another database to mine for artifacts. For example, I was just about to start tearing apart cookies.sqlite when I saw there was a search.sqlite. It didn't turn out to have any significant artifacts, just the listing of search engines in the quick search box. Much more interesting are browser cookies.

Cookies

Browser cookies are a notable and well-known source of browsing history. In IE, each cookie is a separate text file, but in Firefox, they're stored in yet another sqlite database: **cookies.sqlite**. This database has just one table, **moz_cookies**, with several columns:

- **id**: an index
- **name**: the variable being stored
- **value**: the value of "name"
- **host**: the website the cookie is for
- **pathmain**: the path the cookie is valid for.
- **expiry**: when the cookie should be purged
- **lastAccessed**: when the [website last accessed the cookie](#) in PRTIME
- **isSecure**: is HTTPS required to access the cookie?
- **isHttpOnly**: Can the only be accessed by HTTP, or can other methods (Javascript) access it?
- **baseDomain**: The site's base domain, without www or other subdomain
- **creationTime**: when the cookie was created in PRTIME

When you're drawing conclusions from cookies, remember to take into account the browser's cookie control settings as well as the cookie timestamps. Like browsing history, cookies can tell you when the user viewed a website and can be a source for usernames and passwords for insecure websites. However, if the user clears cookies regularly, the amount of data will be limited. It also doesn't mean the user visited the site directly, as many advertising cookies (such as doubleclick.net) will be downloaded for related sites.

Saved Sessions

One useful aspect of Firefox is it will automatically save the currently open browser tabs so you can reopen it the next session. The browser tabs are saved in sessionstore.js in Javascript object notation. These aren't just the raw URLs, possible fields include page title, referrer, formdata, and cookies. Any GET variables in the URL are preserved as well. As a result, if the user was logged into a website, they may still be logged in when the saved session is restored. If sessionstore.js has been damaged or deleted, it may be recoverable in the backup, sessionstore.bak. A quick test shows that sessionstore.bak isn't always a duplicate of sessionstore.js, opening a new browser session overwrote sessionstore.js but not sessionstore.bak, so you may be able to recover two different browser sessions under some conditions.

Bookmark backups

In the profile folder there's a folder called bookmarkbackups, which contains a series of JSON files storing the last [10 backups](#) of the user's bookmarks. The filenames are in the format of bookmarks-YYYY-MM-DD. These backup files include the bookmarks the user explicitly makes, but also Firefox's "Smart Bookmarks" which include items of forensic value like "Most Visited", "Recently Bookmarked", and "Recently Tagged". These backups also include timestamps in PRTIME for when the

Windows Server 2003



FBI — Cyber Crimes
Ransomware on the Rise



Journey Into Incident Response
Linkz for Detection and Response



Digital Forensics Solutions
Spesifikasi Lumia 532, Kekurangan dan Kelebihan Serta Harga Jualnya



Praetorian Prefect
Jimmy Kimmel Gets your Password



M-union
Looking Ahead: The State of Incident Detection and Response in 2015



Digital Forensics Blog
Nominations Open for 2014 Forensic 4cast Awards!



Windows Incident Response
A Smattering of Links and Other Stuff

Social-Engineer.Org

Social-Engineer
Newsletter – Vol 05 Issue 64



DFI News RSS Feeds
Digital Forensics Can Use Facebook to Solve Cases



catch²² (in)security
Taking out the Eurotrash



Assuming the breach
Our Gray Cyber Future

bookmark was added and last modified. As always in forensics, backups like these can provide valuable insight into detecting antifoensics (such as deleting bookmarks) and in behavior over time.

Downloads

downloads.sqlite is where Firefox 3+ stores information relating to downloaded files. There's just one table, **moz_downloads**, but it has quite a few useful artifacts.

- **id**: an index
- **name**: the local filename of the download
- **source**: the remote filename and path being downloaded
- **target**: where it's being downloaded to
- **tempPath**: if the file is complete, it will be blank. If not, it's where the incomplete file is being stored before moving to **target**
- **startTime**: Time the download started, in PRTIME
- **endTime**: Time the download finished, also PRTIME
- **state**: state of download, encoded as an **integer**
- **referrer**: the page containing the link to the file
- **entityID**: a value used for resuming downloads
- **currBytes**: Number of bytes downloaded.
- **maxBytes**: Total file size.
- **contentType**: MIME file type.
- **preferredApplication**: From the download dialogue box, if the user clicks run, this stores the program that will open the downloaded file. If the user clicks save, this will be blank.
- **preferredAction**: Action to take when download is complete. Default is 0, just save the file.
- **autoResume**: Can the download be resumed if broken?

extensions.ini records what Firefox extensions are installed, which may be useful if, for example, hacker tools like Hackbar or anonymizers like FoxyProxy are installed.

Form history

formhistory.sqlite is another good source for artifacts, although unfortunately it doesn't track what website the form was used on.


- **id**: Another numeric index
- **fieldname**: The field that contained **value**. This may be an HTML field in a website's form, or it may be a Firefox field, like searchbar-history.
- **value**: What was typed into **fieldname**. In addition to search history, this is a good source for email addresses and usernames connected to the user.
- **timesUsed**: How many times the user has typed **value** into **fieldname**.
- **firstUsed**: The first time the user typed **value** into **fieldname**, in PRTIME as always.
- **lastUsed**: The last time the user typed **value** into **fieldname**, in PRTIME as always.
- **guid**: A global id for the formhistory, in case syncing is enabled.

Random Thoughts of Forensics

File History, Research - Part 2 - Deconstructing the Catalog.

Lenny Zeltser on Information Security
Security Risks and Benefits of Docker Application Containers


threatpost - The First Stop for Security News
Most Targeted Attacks Exploit Privileged Accounts

 **Ex Forensics**
Cell Phone Tracking via Call Detail Records

 **The Digital Standard**
One Step Ahead Part 3


 **Tracking Cyber Crime**
Rescator Sept update

 **CyberCrime & Doing Time**
Spammers use Google redirection to sneak shady URLs through filters

 **wirewatcher**
Who are you?

 **Forensicaliente - because digital forensics is 'hot'**
It's a Groovy Kind of Risk


Forensic Methods
Registry Analysis with CrowdResponse

 **CounterMeasures**
Naked celebrities revealed by "iCloud hack"

Cache

The cache exists in two folders, for Windows 7 they are Users/[user]/AppData/Local/Mozilla/Firefox/Profiles/[profile name]/Cache and Users/[user]/AppData/Local/Mozilla/Firefox/Profiles/[profile name]/OfflineCache. The offline cache is used when Firefox is in offline mode, but it's the standard cache which will be more likely to be used for forensics. The cache can be viewed through the browser by navigating to about:cache, or it can be viewed directly on the file system. On the filesystem, the actual cache files are stored in a set of directories and subdirectories with names in hex. The files can be located by using the _CACHE_MAP_ and _CACHE_00X_ files. A full writeup on how the cache works has been done by Symantec's response team [here](#). As far as I can tell, the cache scheme hasn't changed since then. This layout is rather inconvenient to navigate by hand, so an automated tool like Firefox's cache browser or other tool is definitely the way to proceed here. Looking at about:cache, it appears Firefox stores useful information like the full URL cached (including GET parameters), cached file size, number of times the cached file was accessed, last time the file was modified and time the cached file expires, full hex dump of the file, and the full HTTP request issued to access it. Since these are individual files on the hard drive, the standard Modified-Accessed-Created timestamps can provide additional information.

Alright, we're finally through the forensic artifacts available in Firefox 4! There'll be just one entry left, anti-forensics in Firefox 4.

Posted by [Alex](#) at 1:22 PM 2 comments:  [Links to this post](#)

Labels: [browser](#), [firefox 4](#), [forensics](#)

Friday, April 22, 2011

Firefox 4 Browser Forensics, Part 3


     Recommend this on Google

In [Part 1](#), we covered typed URLs and the bookmark structure of Firefox 4. In [Part 2](#), we started delving into the moz_places table, ending with an introduction to frecency.

As I noted, frecency is a number generated by Firefox that combines different measures of user behavior with a URL, including bookmarking it, frequency and recency of visits. The result is a single number that tries to quantify how interested the user is in the URL. This means that with a simple SQL query (**select * from moz_places order by frecency desc limit 20** for example), we can get a snapshot of the user's current sites of interest. This could be particularly useful for an "acceptable use policy" investigation, where the issue may turn around how much of a user's browsing history is personal vs. work-related. If the FarmVille app is among a user's top 20 sites sorted by frecency, a violation should be pretty easy to prove. Since frecency is based on other [known factors](#), you could go through the user's full behavior to see what exactly they did to get there (type of visit, etc). That information is stored in **moz_inpuhistory** (which we already covered) and **moz_historyvisits**.


moz_historyvisits


As the names imply, moz_places records the urls that a user visits, and moz_historyvisits record the details of each visit to the url. Since it records each visit to each page in the domain, there will be a lot of records ... which makes a shorthand like frecency very useful. The columns are **id** (the index), **from_visit** (the

 **Computer Forensics, Malware Analysis & Digital Investigations**
EnCase v7 EnScript to find files based on MD5 hash values

Network Forensics Puzzle Contest
Network Forensics Puzzle Contest 2014 Walkthrough

 **A Fistful of Dongles**
AFoD Blog Interview With Andrew Case


 **Happy as a Monkey**
Monkeytown Secret Cyber Tools List

 **Integriography: A Journal of Broken Locks, Ethics, and Computer Forensics**
If You Are Doing Incident Response, You Are Doing It Wrong

 **Alex Levinson**
CCDC 2014 Year in Review

 **The Firewall**
DARPA-Funded Researchers Help You Learn To Hack A Car For A Tenth The Price

PandaLabs Blog
PowerLocker

 **Forensic Focus Blog**
Webinar: Accelerating Investigations Using Advanced eDiscovery Techniques

WebCase WebLog
New Book Investigating Internet Crimes Released

referrer), **place_id** (the connection back to the URL entry in moz_places), **visit_date** (timestamp in PRTime again), **visit_type**, and **session**.

Let's step through what this means by actually analyzing my browsing behavior. First, I ran **select * from moz_places order by frecency desc limit 20** to grab the top 20 websites from moz_places to look for something interesting. I'm arbitrarily picking <http://www.wired.com> as my URL of interest. moz_places.id for this url is 485, so I run **select * from moz_historyvisits where place_id=485** which yields:

```

C:\Windows\system32\cmd.exe - "c:\Program Files\sqlite3.exe" places.sqlite
sqlite> select * from moz_historyvisits where place_id=485;
9421941:485:1130091053617200001511518
9721970:485:11300914391145000015126
101114010:485:113009251965000015186
134411343:485:113016118334640000151562
1653101485:113026515075150000121420
167752101485:113028914597208000121212988
180842101485:113034039873998000121213102
sqlite>

```

seven visits. Now, for these results we can ignore the first column (id). Next is **from_visit**, which gives the id of the page I visited before visiting the current page. For the last three visits, it's 0 indicating that I opened a new tab and went straight to Wired. For the others it refers to the immediately prior id number, which is what you would expect if I was only browsing in one tab at the moment. If the numbers were not consecutive, that would imply multiple-tabbed browsing. In the first entry, I came to Wired after visiting the entry in moz_historyvisits.id=941. Looking up that entry, I see that was a visit to place_id=484, which is <http://wired.com>. Incidentally, that's true for the first four entries. To explain why, see that the visit type for all four www.wired.com entries is 5, indicating a [permanent redirect](#) (HTTP 301). For the respective entries for <http://wired.com>, the visit type is 2, indicating I typed the URL. So, at those times I typed wired.com in the address bar. Firefox logged the visit and took me to <http://wired.com>, which redirected me to <http://www.wired.com> and logged another visit. For the last three visits to Wired, I typed www.wired.com directly. Since those were the more recent and since I logged 7 visits to www.wired.com and only 4 to wired.com, www.wired.com was the one that was on top in frecency (8800 vs. 4813) instead of the non-www version, despite the fact that a typed URL gets a frecency bonus over a redirect URL. Also note that just because I came to www.wired.com via a redirect, doesn't mean I didn't intend to go there. Seeing what site took me there is essential to determine what my intentions were.

Obviously this is a trivial example, but it should be pretty apparent how it could be applied to a more interesting investigation.

More tables

There are a set of tables called **moz_annos**, **moz_anno_attributes** and **moz_items_annos**, but [as far as I can tell](#) they are used by Firefox extensions and are unlikely to be useful for forensics.

That's everything of interest out of **places.sqlite**, but there are other tables that have useful information: tables like **signons.sqlite**. This is where the logins and passwords are stored. If the user hasn't specified a master password for his Firefox profile, then the usernames and passwords will be clearly available through Options > Security > Saved Passwords. If there is a master password set and you don't have it, then you won't be able to get to the list of sites and passwords directly. Nothing stops you from accessing **signons.sqlite** directly, though.

signons.sqlite contains two tables, **moz_logins** and **moz_disabledHosts**.

The Security Shoggoth
Installing Yara into IDA
Pro 64-bit Linux

Righteous IT
Advice to Recruiters

Police-Led Intelligence
A New Hope...

Security Brindump
Finding Cryptolocker
Encrypted Files using the
NTFS Master File Table

JadedSecurity
ISC2 Board of Directors
2013

An Eye on Forensics
A Cold Day in E-
Commerce - Guest Post

Digital Detective
NetAnalysis v1.56 / HstEx
v3.10 Release

Digital Forensic Source
A Rubik's Approach:
Casey Anthony Google
Searches Overlooked &
Lessons Learned

Girl, Unallocated
Be Very Quiet... I'm
Tracking Emails Through
Headers

SecureArtisan
Alternative to Typed
URLs?

Practically Secure:
Don't forget the "A"

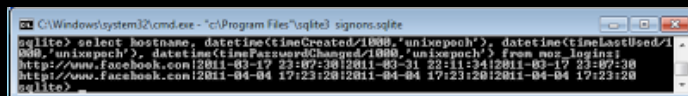
**News & Events and
Other Interesting Stuff**
InfraGard & FBI's Trident

moz_disabledHosts is a list of websites which the user doesn't want Firefox to ask to remember logins. **moz_logins** is where the sites, usernames and passwords are stored. The columns are **id**, **hostname** (the login page), **httpRealm**, **formSubmitUrl** (the actual page the username and password get sent to), **usernameField** (the identifier of the username field), **passwordField** (the identifier of the password field), **encryptedUsername**, **encryptedPassword**, **guid** (global ID, for login syncing), **encType**, **timeCreated**, **timeLastUsed**, **timePasswordChanged**, **timesUsed**.

Setting a master password in Firefox does not change any of the data here, it just blocks you from viewing the decrypted passwords IN Firefox. You can still get lots of useful browsing information from the file, everything except what their username and passwords are. Documentation on **signons.sqlite** is scanty, but from what I can tell all the timestamp fields are new in Firefox 4. Strangely, these timestamps aren't encoded in PRTime (microseconds since Unix epoch) like the rest of the timestamps. Instead it's milliseconds since epoch, so we get all the timestamps with:

```
sqlite> select hostname,
datetime(timeCreated/1000,'unixepoch'),datetime(timeLastUsed/1000,'unixepoch'),d
atetime(timePasswordChanged/1000,'unixepoch') from moz_logins;
```

For my test system, the results are:



```
C:\Windows\system32\cmd.exe - "c:\Program Files\sqlite\sqlite3.exe" signons.sqlite
sqlite> select hostname, datetime(timeCreated/1000,'unixepoch'), datetime(timeLastUsed/1
000,'unixepoch'), datetime(timePasswordChanged/1000,'unixepoch') from moz_logins;
http://www.facebook.com|2011-03-17 23:07:30|2011-03-31 22:11:34|2011-03-17 23:07:30
http://www.facebook.com|2011-04-04 17:23:20|2011-04-04 17:23:20|2011-04-04 17:23:20
sqlite>
```

This system has credentials saved for two Facebook accounts. For the first, the login was saved on March 17 and last used on March 31. For the second, it was only used once, on April 4. Neither has changed passwords. This can be useful if you need to prove that a user actually logged into a site, rather than just visiting it. If a user clears their browsing history but not saved passwords (which is default behavior), this will still show up as well. It's also yet another way to prove that a user actually intended to go to a site repeatedly instead of it being an accidental click or redirect.

The usernames and passwords are encrypted, but the master password and decryption keys are stored in key3.db (more info [here](#)). Since you have the decryption keys, decrypting the passwords should be possible if needed. There are [tools out](#) to do that, so I'll leave that as an exercise to the reader. :)

In [Part 4](#), I'll go into cookies and cached files.

Posted by [Alex](#) at 12:56 PM No comments:  [Links to this post](#)


Labels: [browser](#), [firefox 4](#), [forensics](#)

Monday, April 18, 2011

Firefox 4 Browser Forensics, Part 2

  Recommend this on Google

Breach investigation


 **High Tech Happenings**
Open source Android
forensics: An HTCIA
student charter project


Information Warfare Monitor
Big Data Meets Big Brother


Cyber Crime 101
Book Review: Windows
Forensic Analysis 2/e by
Harlan Carvey


inREVERSE
More on callbacks:
ObRegisterCallbacks


Network Forensics Blog
Finding injection attacks
by looking for injection
attacks is a fail

 **Post Humorous**
Sleuth Kit & Open Source
Forensics Conference

 **Packetstan**
Crafting Overlapping
Fragments Finally!

 **Forensic Incident Response**
On the sophistication of
attacks

 **Brian Baskin's Site (FWIW)**
Putting the Lightning up
for sale

 **CyberSpeak's Podcast**
CyberSpeak March 31
2008

Journal

Verizon Business Security Blog

moz_places

Blog Archive

[illegible]

2011

July

June

May

Reverse engineering a malicious PDF

Part 1

Firefox 4 Browser Forensics, Part 5

April

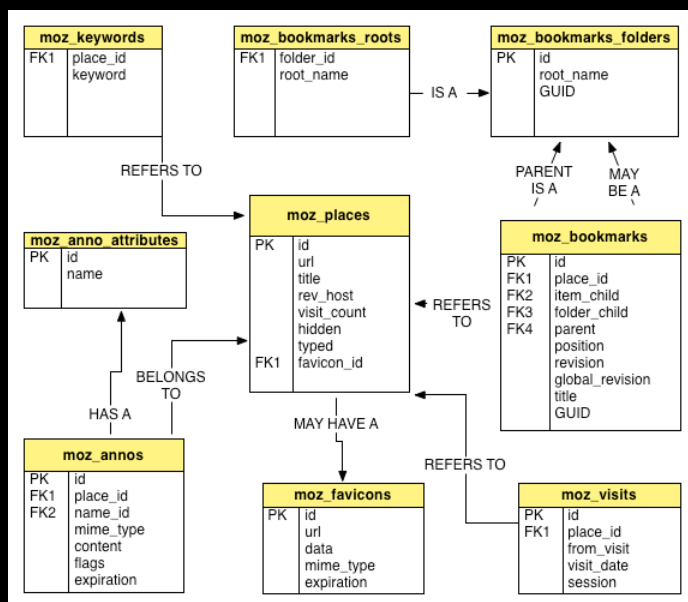
March

February

January

2010

As we've mentioned before, **id** is a simple numerical index that connects the links in **moz_places** to their references in the other tables. The Mozilla Wiki has a handy graphic for reference.



The Mozilla Wiki notes:

Bookmarks are basically pointers to that history entry, with a few properties to determine their placement in the bookmarks folder hierarchy. The design is such that most properties of a bookmark are derived from the `moz_history` entry, and not the bookmark. This is problematic as soon as the same URI is bookmarked more than once. Eg: If you change a bookmark's title, then the title changes anywhere you've bookmarked that URI.

This is no longer the case. As we already discovered, the **`moz_bookmarks`** table has it's own **`title`** field where the bookmark title is stored. Changing the bookmark title no longer overwrites the title in **`moz_places`**.

`rev_host` is the fully qualified domain name backwards, ending with a period. So, "`http://www.phishtank.com/phish_detail.php`" gets converted to "`moc.knathsihp.www.`" Firefox uses this so that they can index this and quickly search for subdomains of a primary domain. An investigator may want to use this field for the same purpose. See the [Firefox source code](#). Thanks to [FirefoxForensics](#) for this.

`visit_count` is another field very useful for an investigation. This is an integer that increments for every time the user visits the site. Since the `places.sqlite` is associated with the user logged into the computer, this only reflects the number of times that particular user account visited the site. Some additional notes from FirefoxForensics:

This counter is incremented when the associated `places.sqlite::moz_historyvisits::visit_type` is not 0, 4 (embedded), or 7 (download).

Research by the author shows that reloading a page by clicking the 'Reload current page' button, pressing CTRL-R or following a self-referring URL does not increment `visit_count`.

If the start-up option to 'Show my windows and tabs from last time' is selected, the `visit_count` of those pages will NOT be incremented when the browser is started and they are loaded.

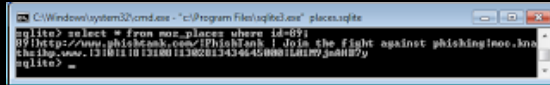
If the start-up option to 'Show my home page' is selected, the `visit_count` of the home page WILL be incremented each time the browser is started and they are loaded.

hidden

According to the [Mozilla devs](#), "Hidden uri's are ones that the user didn't specifically navigate to. Those tend to include embedded pages and images. It might include something else, but I'm not 100% sure." At first glance one might want to exclude this from analysis because the user didn't specifically navigate to them, but since it includes images that would be a mistake. If the user navigated to the page containing the image, the images entry here would still give you valuable info like how many times the image has been loaded, etc.

`typed` is different than the **`input`** field in **`moz_inpuhistory`** If the user started typing and then clicked the quick result, then it'll be listed under **`moz_inpuhistory.input`**. If the user typed the full url, then it'll be under **`moz_places.typed`**.

To give an example, the first time I went to `phishtank.com`, I typed "`www.phishtank.com`" into the address bar and it took me to the site. That generated this entry in the database:



The next time I visited, I just typed 'phishtank' before Firefox recognized the URL and I went to the site. That got stored in `moz_inpuhistory`.

favicon_id is an id number that connects to which image in **moz_favicons** is the **favicon** (the little graphic in the title bar) for the page.

The last field in `moz_places` is `frecency`. That's not a typo for "frequency", it's a combination of "frequency" and "recency". As you'd expect from the name, it's a value generated by both how often the user has visited a place and how recently they visited it. The full algorithm is [here](#). 0 is the default score, and the higher the number the higher it appears on autocomplete results. As the algorithm link shows, it draws on a number of elements of user behavior to determine the frecency. This means that you can use it as a shorthand for determining if a user was interested in the URL and then investigate the underlying behaviors (how often they visited, how often they typed the address vs. redirect vs. following links, if they bookmarked it, etc) to articulate what exactly they did that shows interest in the link.

Frequency turns out to be a pretty complicated topic, so I'll save it (and **signons.sqlite**) for [part 3](#). [Part 4](#) covers cookies, cache, downloads and more.

Posted by [Alex](#) at 10:19 AM No comments: [Links to this post](#)

Labels: browser, firefox 4, forensics

Wednesday, April 13, 2011

Firefox 4 Browser Forensics, Part 1



g⁺¹ +1 Recommend this on Google

Firefox 4 was released almost a month ago now, and was in open beta for a significant time before that. However, a cursory Google search doesn't reveal much documentation of it's features of forensic interest, or it's changes since the last version.

For this analysis, I'm using Firefox 4.0 (current version as of writing) running on a Windows 7 VM. To examine the sqlite databases, I'm using the sqlite3 command line tool. Sqlite3 runs under Linux and Windows, and you can feed it SQL commands to script common actions, such as searching history for keywords or for browsing during a time range of interest.

Just like Firefox 3, Firefox 4 stores the browser history in an SQLite database. For Windows Vista/7, it's located at

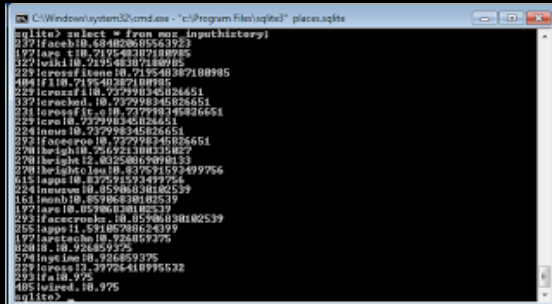
:Users\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\places.sqlite This database contains the tables **moz_anno_attributes** **moz_annos** **moz_bookmarks** **moz_bookmarks_roots** **moz_favicons** **moz_historyvisits** **moz_inpuhistory** **moz_items_annos** **moz_keywords** and **moz_places** These tables seem unchanged from version 3, as documented on the [MozillaZine](#).

moz_inpuhistory

When a person begins typing into the address bar, Firefox searches through its browser history to try and connect what the user is typing with the websites he has visited recently. It outputs the suggestions with the webpage title and full URL in the suggestions below the address bar. When the user clicks the suggestion, the text

they typed to find that url is stored in the **moz_inpuhistory** table. The moz_inpuhistory table is of particular interest to forensics because it can show that the person using the user account typed something into the address bar to access the site, rather than following a link or getting forced there by a pop-up.

The column names for this table are `place_id`, `input`, `use_count`, and `PRIMARY KEY`. Here's an example of the contents of the table, taken from my test system.



Note that none of these are full links. What you see here is what you get when someone starts typing a URL or keyword into the address bar and then clicks on one of the autocomplete terms. It does not seem to record full URLs. So, we get "ars t" and "arstechn" from two of the ways I accessed [Ars](#)

Technica. It does not store search terms from the search box.

place_id refers to the destination of the input. This number can be converted into the actual website visited via the **moz_places** table. So, for example, I see that when I typed "ars t" into my address bar, I clicked on the website with id 197. The command **> select * from moz_places where id=197;** returns the website URL (<http://arstechnica.com/>) and the website title (Ars Technica) as well as other information that we'll cover when we get to the moz_places table.

The last column, `use_count` is an odd one. [David Koepi](#) looked at Firefox 3.6 and suggested that use count refers to the number of times the same text was typed into the address bar. On my system in Firefox 4, that doesn't seem to be the case as most of the values in `use_count` are decimals (see screenshot above). Running some tests, it seems to be related to the number of times run ... for example, typing "sans.o" into the address bar the first time adds it to `moz_inputhistory` and sets `use_count` to 1. So far so good, but here's where it gets odd. Typing it in four more times changes it to 1.9, then 2.71, 3.439, and 4.0951. As you can see from the other links in my history, many of them end up with values less than 1. This is probably a bug in Firefox, so I wouldn't rely on the value of this for any reason until it's better understood.

This is why it's important to do your own testing, instead of just believing what you read on the Internet.

moz bookmarks and moz bookmarks roots

Bookmarks are also valuable for an investigator to examine because like `inputhistory` it implies that the user took an active action with the site, in this case marking it for later re-visit. The fields here are `id`, `type`, `fk`, `parent`, `position`, `title`, `keyword_id`, `folder_type`, `dateAdded`, `lastModified`, and `guid`. Comparing to the documentation by [David Koepi](#) and [firefoxforensics](#), it appears `guid` has been added for FF4. According to the [Mozilla wiki](#), `guid` was added in order to have a unique global identifier, rather than the simple incrementing index of `id`. This means that `gid`, not `id`, would be the proper link between bookmarks across computers using the new Sync service in FF4. Note, I haven't tested this.

type is an integer referring to the type of the bookmark. Type 1 is a normal bookmark, 2 is a tag (folders are really just a type of tag). Type 3 is a separator, and so has no useful info. **fk** is a foreign key, which links to the id in **moz_places** (where

you can find the actual url). **parent** refers to the id of the containing object (folder), which can tell you where the user categorized the bookmark. **position** refers to the listing order of the bookmarks, and is not likely to be interesting. **title** refers to the display name of the bookmark and **keyword_id** is the index number of the keyword (moz_keywords) associated with the bookmark. It may be useful to find associated bookmarks. folder_type is not likely to be useful, but dateAdded and lastModified would be. These values are exactly what they sound like, stored as [PRTime](#)

moz_bookmarks_root defines the root folders in the Firefox bookmark structure, and is not of forensic interest.

In [Part 2](#) I'll get into **moz_places** and more browsing history. [Part 3](#) has **moz_historyvisits** and **signons.sqlite**. [Part 4](#) covers cookies, cache, downloads and more.

Posted by [Alex](#) at 3:48 PM No comments:  [Links to this post](#)

Labels: [browser](#), [firefox 4](#), [forensics](#)

[Home](#)[Older Posts](#)

Subscribe to: [Posts \(Atom\)](#)

© 2010, Alex Bond. Awesome Inc. template. Template images by [RBFried](#). Powered by [Blogger](#).