A Survey of Firefox Extension API Use

Adrienne Porter Felt

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-139

http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-139.html

October 16, 2009

Copyright © 2009, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Thanks to Adam Barth and Prateek Saxena, who helped compile and categorize the list of interfaces.

A Survey of Firefox Extension API Use

Adrienne Porter Felt
UC Berkeley
apf@berkeley.edu

October 16, 2009

Abstract

Mozilla Firefox provides third-party developers with a framework for writing *extensions* to add functionality to the browser. Extensions have unfettered access to browser privileges: extensions can snoop on web content, delete files from the hard drive, and even launch new processes from arbitrary binaries. Extensions might be intentionally malicious (i.e., a user unknowingly installs browser malware) or they might accidentally leak privileges to malicious web sites. It would be desirable to limit the powers of extensions, but we also do not want to cripple the extension framework too severely. Here, we review twenty-five "recommended" Firefox extensions to provide a basis for the discussion of legacy extensions' interface needs. Notably, we find that very few extensions need access to the file system or system calls despite the fact that all extensions have this ability.

1 Introduction

Mozilla Firefox provides third-party developers with an extensive *extension* framework for adding functionality. Extensions can change the visible browser chrome or access browser resources through an API. Examples include ad blocking and sharing websites with social contacts. These third-party tools have proven extremely popular, with the most downloaded extension in the Firefox directory having nearly 50 million total downloads [7].

The Mozilla policy thus far has been to give extensions complete browser privileges. Extensions can change chrome, look at user's web content, make web requests on behalf of a user, and access password and cookie managers. They can read, write, and execute files on the local file system. Writing spyware or malware is trivial since the code is so highly privileged. Malicious extensions (or unauthorized, criminal versions of legitimate extensions) do exist in the wild [6, 3]. Additionally, a vulnerability in a well-intentioned extension can lead to a browser or system compromise. Mozilla attempts to mitigate the malware potential of extensions by providing an official directory of approved Add-ons, all of which must go through a code review process. This is a slow and time-consuming process, especially since future updates must also be reviewed. Despite the review process, approved extensions may still feature bugs that can lead to the privilege escalation of malicious web sites [5].

Giving all extensions full privileges appears excessive given the simple nature of many extensions. Let us consider two simple extensions:

- **Answers 2.2.48.** This extension helps users look up words from web sites on `Answers.com`. To use it, the user highlights some text and then alt-clicks on the selected text. A styled pop-up balloon appears with `Answer.com`'s dictionary entry for the text. The extension works by fetching the page from `Answer.com` with a normal GET request.
- **Fission 1.0.3.** This extension makes the URL bar color load proportionally to the load state of the URL, like in Safari. It checks to see what platform is running to decide on the appropriate chrome styling, and it lets the user choose a custom image for the progress bar.

Why does an encyclopedia lookup tool need to be able to launch processes? We hypothesize that many Firefox extensions can be satisfied with a significantly reduced set of privileges. On the other hand, we acknowledge that some number of extensions do actually need powerful abilities. The goal of this project is to substantiate the discussion of extension security with information about the privilege requirements of real extensions. Our case study is intended to motivate the design of new extension platforms that better reflect extension security needs.

We review twenty-five popular extensions from the "recommended" directory. First, we examined extension behavior to determine what privileges an extension needs to accomplish its functionality. We then collected the set of interfaces that an extension uses to see what privileges it receives from the API in pursuit of its functionality. We also characterized extension complexity based on the number of interfaces used by an extension and identified the most popular interfaces. Section 3 describes our methodology, and Section 4 presents the results of the survey.

2 Background

Mozilla Firefox resources are available through the XPCOM (Cross-Platform Component Object Model) API. Internal Mozilla code and extensions alike use the XPCOM API. The interfaces include services such as download, cookie, window, and password managers. It also provides network connectivity, file system access, and generally utility tools such as DOM parsing. The XPCOM API is implemented in C++ and can be accessed through a set of interfaces described by a Corba-like Interface Description Language (IDL).

Extension authors commonly create their own components, which may implement XPCOM interfaces. An extension component can replace a default Mozilla-implemented component. Alternately, a component can simply be used to pursue an object-oriented singleton pattern. Extensions without components are essentially a set of GUI event handler functions; components are used to maintain global state and functionality for the extension. Components can be written in either C++ or JavaScript; when written in C++, typically only a DLL is provided.

Plug-ins and extensions are often confused; we only consider extensions in this work. Plug-ins are wholly binary applications that make extensive use of direct operating system access, unlike extensions which are largely written in JavaScript and primarily rely on the XPCOM API for resources. Plug-ins act as platforms for rich media types like Flash and Java, which makes policies for plug-ins more complex; permissions need to be determined for each application a plug-in runs, rather than for as plug-in as a whole. Extensions are self-contained, so each extension will only need one set of privileges. Although plug-in security is an important area of research, plug-ins require different policies and confinement techniques [4].

3 Methodology

The set of extensions we examined came from the thirteen categories[1] on the "recommended" section of the Mozilla Add-on directory. We randomly chose two from each category for a total of twenty-six extensions[2]. One (ColorZilla) included a C++ component, so we had to elide it from review. We therefore considered twenty-five extensions (the set of 26 minus ColorZilla). All of the extensions are ranked highly in the "popular" directory as well.

First, we manually tested each extension and reviewed its code to determine what behavior the extension wants to accomplish. Next, we collected the set of interfaces each extension uses to implement its behavior. We used a `grep` wrapper to search for all mentions of XPCOM interface names in the code. Once we had the set of interfaces used by an extension, we manually matched up the interfaces with the extension's functionality. Our search strategy may have missed some XPCOM interfaces because we only find explicitly named interfaces, but we do not believe this amounts to a large number of interfaces because we augmented it with manual review. Additionally, during the manual review process, we noticed that some extensions include libraries with interfaces and functionality that are never used in the main body of the extension; we identified this code to the best of our ability and removed it from consideration in our survey.

In order to examine extension behaviors and interfaces from a security perspective, we assigned security ratings to each of the behaviors and interfaces. The ratings are based on Firefox security severity ratings [1] and Chromium's severity guidelines for security issues [2]. These guidelines provide descriptions for *critical, high, medium,* and *low* vulnerabilities. We rate an interface by answering the following question: if a malicious party obtains a reference to that functionality, what degree security breach has occurred? We do not rate an interface based on the kinds of interfaces it could possibly return, but rather by the actions that interface itself can be prompted to do.

- *Critical.* Can run arbitrary code on the user's system. Includes file I/O, system calls.
- *High.* Access to broad private information (e.g., the cookie and password stores), the DOM of all web pages, or data that belongs to other extensions.
- *Medium.* Pertains to limited data, such as recent history or the DOM of specific web pages.
- *Low.* At worst, can annoy the user.
- *None.* Completely trivial (e.g., strings) or limited to an extension's own data.

We can use these ratings to see how many extensions actually need critical or high privileges. We can also look at whether extensions use excessively powerful interfaces to implement their functionality.

---

[1] The thirteen categories are: Alerts & Updates; Appearance; Bookmarks; Download Management; Feeds, News & Blogging; Language Support; Photos, Music & Videos; Privacy & Security; Search Tools; Social & Communication; Tabs; Toolbars; and Web Development.

[2] Adblock Plus 1.0.2, Answers 2.2.48, AutoPager 0.5.0.1, Auto Shutdown (InBasic) 3.1.1B, Babel Fish 1.84, ColorZilla 2.02, CoolPreviews 2.7.4, Delicious Bookmarks 4.3, docked JS-Console 0.1.1, DownloadHelper 4.3, Download Statusbar 2.1.018, File and Folder Shortcuts 1.3, Firefox Showcase 0.3.2009040901, Fission 1.3, Glue 4.2.18, GoogleEnhancer 1.70, Image Tweak 0.18.1, Lazarus: Form Recovery 1.0.5, Mouseless Browsing 0.5.2.1, Multiple Tab Handler 0.9.5, Quick Locale Switcher 1.6.9, Shareaholic 1.7, Status-bar Scientific Calculator 4.5, TwitterFox 1.7.7.1, WeatherBug 2.0.0.4, Zemanta 0.5.4

4 Study Results

Here, we give and discuss the results of our study. First, we present the privilege requirements of extensions and show that extensions are overprivileged. We then present information about extension complexity and the popularity of certain interfaces.

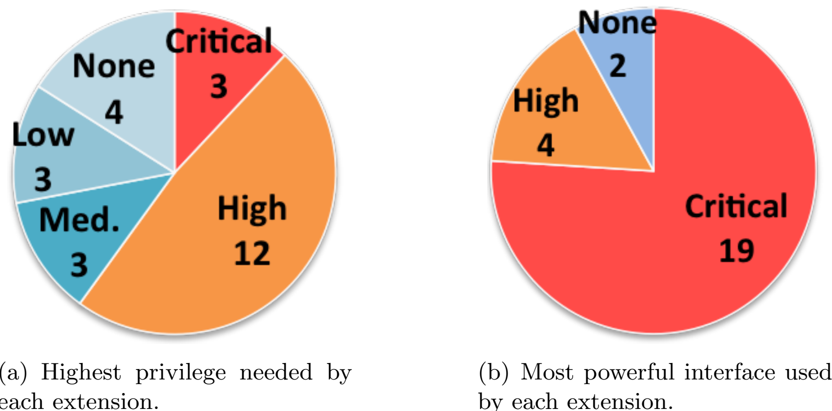
4.1 Privilege Gap

Of the 25 subject extensions, only 3 require critical privileges for their functionality. This means that the remaining 22 extensions are overprivileged because all extensions have the ability to perform critical tasks. Figure 1(a) shows a summary of the highest privilege required by each extension in our study. It should be noted that all three extensions that require critical privileges are download managers. DownloadHelper converts file types using system utility files like `/usr/bin/ffmpeg`. Download Statusbar asks the user to supply a path to a virus scanner and then starts the virus scanner for each downloaded file. AutoShutdown lets users delay the shutdown of their computer until after all downloads have completed; it launches a new process to use the operating system's API for shutdown. A future extension system might remove critical privileges altogether and then accommodate download managers as a special, high-need category of extensions.

We also discovered that individual interfaces are more powerful than they need to be. Only 3 extensions need critical privileges to function, but 19 use a critical-rated interface in their implementation. An additional 3 use high-rated interfaces despite needing only medium or lower privileges, meaning that a total of 19 extensions use interfaces that have broader privileges than they require. Figure 1(a) shows the distribution of the most powerful interface used by each extension.

To account for the disparity between extension behavior and implementation, Figure 2 shows the most popular security-relevant behaviors. The relevant interfaces should be considered primary targets for a future API redesign. Here, we delve into further detail:

File access. None of the extensions we studied require arbitrary file access, which is what `nsIFile` and related interfaces provide. User file choosing can be accommodated with a limited file handle. Extension-specific files are used to store information that is too complex for the preferences system, and this behavior can be safely handled with file pointers that are only valid for files in the



(a) Highest privilege needed by each extension.

(b) Most powerful interface used by each extension.

Figure 1: This figure shows the privilege gap between extension functionality and implementation.

Behavior	Interface	Disparity?	Frequency	
Process launching (C)	Process launching (C)	No	3	(12%)
User chooses a file (N)	Arbitrary file access (C)	Yes	11	(44%)
Extension-specific files (N)	Arbitrary file access (C)	Yes	10	(40%)
Extension-specific SQLite (N)	Arbitrary SQLite access (H)	Yes	3	(12%)
Arbitrary network access (H)	Arbitrary network access (H)	No	8	(40%)
Specific domain access (M)	Arbitrary network access (H)	Yes	2	(8%)
Arbitrary DOM access (H)	Arbitrary DOM access (H)	No	9	(36%)
Page for display only (L)	Arbitrary DOM access (H)	Yes	3	(12%)
DOM of specific sites (M)	Arbitrary DOM access (H)	Yes	2	(8%)
Highlighted text/images (L)	Arbitrary DOM access (H)	Yes	2	(8%)
Password, login managers (H)	Password, login managers (H)	No	3	(12%)
Cookie manager (H)	Cookie manager (H)	No	2	(8%)
Same-extension prefs (N)	Browser & all ext prefs (H)	Yes	21	(84%)
Language preferences (M)	Browser & all ext prefs (H)	Yes	1	(4%)

Figure 2: The frequency of security-relevant behaviors. The security rating of each behavior is abbreviated in parentheses. If the interface's privilege is greater than the required behavioral privilege, there is a disparity.

appropriate extension folder. SQLite access could be limited to the portions of the database that are relevant to a given extension with standard database techniques.

DOM access. 9 of our studied extensions need full page DOM access, either for user-navigated pages or content retrieved via independent network access. None distinguish between HTTP and HTTPS pages, so any DOM-reliant extension would break if we removed HTTPS DOM access. Two are satisfied with the DOM of specific sites (e.g., Zemanta adds GUI functionality to about a dozen specific pages). Limiting DOM access to specific sites could prevent extensions from snooping on sensitive web site data.

Network access. Network access includes observing network data or issuing HTTP requests (either through the XPCOM API or with `XMLHttpRequest`). 10 extensions require network access in some capacity; of those, 2 only want specific domain access. Network access is typically accompanied by DOM access, but 3 extensions issue content requests for display only and do not need DOM access to the result.

Password, login, and cookie managers. A number of extensions make legitimate use of credential-related managers. If an extension is associated with a specific site, it might want to authenticate users. Twitterfox (a third-party program not officially affiliated with Twitter) handles Twitter account and login information this way. Lazarus requires a password to read the data that the extension has stored on behalf of the user, and CoolPreviews will save e-mail and password information to facilitate the easy sending and sharing of links. The Delicious extension checks `delicious.com` and `de.li.cio.us` cookies, and DownloadHelper checks a MP3 website. All of these are site-specific credentials; it could be beneficial to limit access to credential managers by site. This would have little effect, however, if an extension has full DOM access because the extension could simply pull a site password out of the DOM.

Preferences. 20 of the extensions use the preference system to store preferences related to themselves. Each of these extensions could be given its own preferences branch and denied access to browser-wide or other extensions' preferences. One extension (Quick Locale Switcher) needed access to browser-wide settings to change the language preference.

Of the 228 interfaces used by the twenty-five extensions, 17 (7.4%) are rated critical, 52 (22.8%) are high, 24 (10.5%) are medium, 18 (7.9%) are low, and 117 (51.3%) are none. We can also weight these by prevalence: we found a total of 735 interface references, of which 73 (9.9%) are critical, 161 (21.9%) are high, 57 (7.7%) are medium, 28 (3.8%) are low, and 416 (56.6%) are none. By either metric, use of critical interfaces makes up a small fraction of overall interface usage. Since 19 extensions use at least one critical interface, this implies that a small number of critical interfaces are disproportionately popular.

4.2 Popular Extension Behavior

In addition to the security-relevant functionality in the previous section, we noticed five particularly popular behaviors with no notable security ramifications. Consequently, supporting these behaviors might be of interest to a new extension platform.

Behavior	**# Extensions**
Logs to the console	13 (52%)
Registers a component	11 (44%)
Clipboard access	10 (40%)
JSON	8 (32%)
RSS/RDF/XML reading and writing	6 (24%)

Figure 3: The relative popularity of different interesting behaviors.

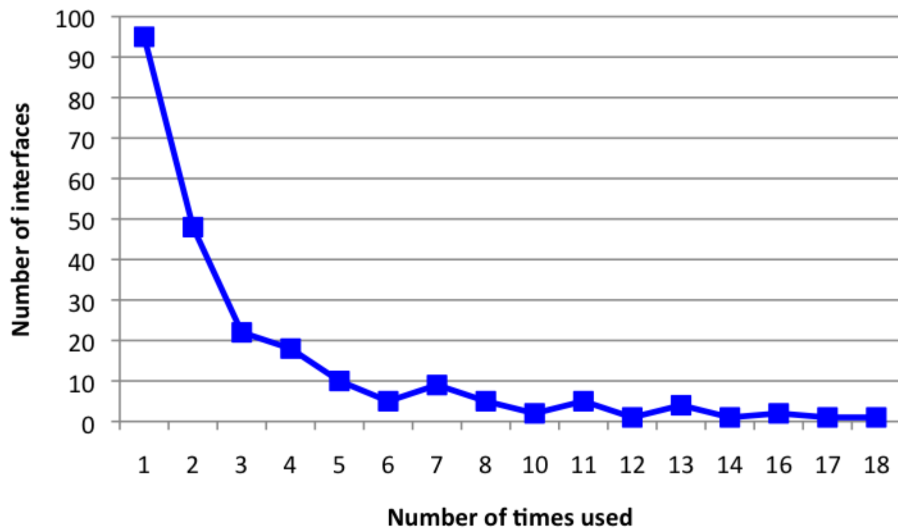
Notably, developers register components so that they can implement a singleton design pattern. Even if the component-based API disappears, extensions should still be given a way to do this. Additionally, an API might want to include a JSON parser to help extensions avoid the use of `eval` on input data. The popularity of RDF interfaces is partially caused by extensions reading Firefox's metadata (e.g., the list of extensions slated for uninstallation).

4.3 Popular Interfaces

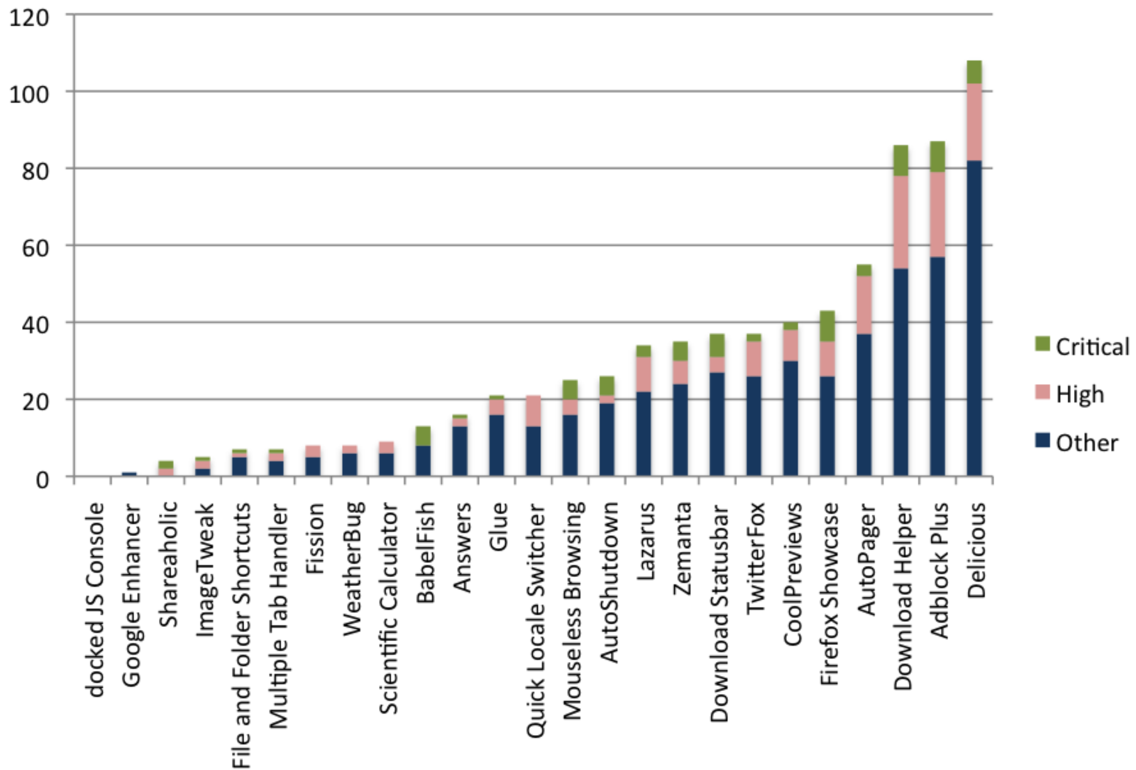
From the aggregate list of interfaces used by extensions, we can identify the interfaces that are particularly popular. This is a good indication of which interfaces are important for a new API, although it is not a perfect gauge of important features (for example, there are four different interfaces relating to preferences, and they are scored separately). When relevant, we consider the security ramifications of each popular interface.

Figure 4(a) shows the popularity of each of the 228 interfaces in our study. The majority of interfaces (nearly a hundred) are only used by one extension. Only ten are used by twelve or more extensions:

1. `nsIWindowMediator` (18) keeps track of open windows and can be used to get window objects.



(a) Particularly popular interfaces. A large number of interfaces (nearly 100) were only used once, and a very small number of interfaces were used by 50% or more of the extensions.



(b) Interface usage breakdown, showing how many interfaces were used by each extension and how many of those were security-relevant.

Figure 4: Results of the interface survey.

2. `nsIIOService` (17) provides I/O helper functions, like URI parsing and getting protocol schemes. It can be used for local URIs (`file://` and `chrome://`) as well as network access.
3. `nsIPrefBranch` (16) is used for preferences. Severity: High. It can be used to edit any preference settings, whether they belong to other extensions or the browser itself. This could be limited as discussed in Section 4.1.
4. `nsIPrefService` (16) is also used for preferences. Severity: High.
5. `nsISupports` (14) is the base class from which all XPCOM interfaces inherit. Extensions' components must explicitly inherit from it in their definitions. It also provides the commonly used `QueryInterface` method, which provides runtime type discovery.
6. `nsIConsoleService` (13) provides logging services.
7. `nsIExtensionManager` (13) provides general extension management functions, such as checking for upgrades and incompatible extensions. Severity: Critical. It can add things to the download manager or install a new extension. Simple extension management functions (e.g., checking for an upgrade) could be separated from the more dangerous ones.
8. `nsIFile` (13) represents a file and has related methods. Severity: Critical. This could be limited as discussed in Section 4.1.
9. `nsILocalFile` (13) is a platform-independent file handler and its related methods. Severity: Critical. Same analysis as `nsIFile`.
10. `nsIPromptService` (12) displays dialogs similar to `window.alert` and `window.confirm`.

Figure 4(b) shows the distribution of interfaces across extensions. Nearly half (eleven) of the extensions use fewer than twenty total interfaces, and only a handful (three) use more than eighty interfaces. Delicious is the most complex with 109 interfaces; it replaces and enhances the entire bookmarking functionality, authenticates users, and interacts with the `de.licio.us` website. The graph also shows how many interfaces were rated critical or high for each extension. Notably, the number of critical interfaces holds relatively steady across all extensions, even as the total number of interfaces used by each extension differs greatly.

5 Conclusion

We present a study of 25 Firefox extensions. After examining their behavior, we determine that only 3 of the 25 extensions actually need access to the most powerful capabilities of the Firefox extension system. We then compare extension behavior to the interfaces used to implement it and find a privilege gap between the desired functionality and the actual interfaces. Reducing the power of file handles and the preference system would make a significant difference in improving this discrepancy. We suggest that domain-specific DOM and network access should also be considered as an option.

6 Acknowledgements

Thanks to Adam Barth and Prateek Saxena, who helped compile and categorize the list of interfaces.

References

[1] L. Adamski. Security Severity Ratings. https://wiki.mozilla.org/Security_Severity_Ratings, 2008.

[2] A. Barth. Severity Guidelines for Security Issues: http://dev.chromium.org/developers/severity-guidelines.

[3] BitDefender. Trojan.pws.chromeinject.b. urlhttp://www.bitdefender.com/VIRUS-1000451-en–Trojan.PWS.ChromeInject.B.html.

[4] C. Grier, S. T. King, and D. S. Wallach. How I Learned to Stop Worrying and Love Plugins. In *Web 2.0 Security and Privacy*, 2009.

[5] R. S. Liverani and N. Freeman. Defcon17. Abusing Firefox Extensions, July 2009.

[6] McAfee. Form Spy. http://vil.nai.com/vil/content/v_140256.htm.

[7] Mozilla. Add-on Directory. https://addons.mozilla.org/en-US/firefox/addon/1865.