Prionti Nasir
SWEN 331

Overview of Vulnerabilities in Mozilla Firefox

Summary

Mozilla Firefox is a free web browser. Firefox 1.0 was released in 2004 and became a big success — in less than a year, it was downloaded over 100 million times. New versions of Firefox have come out regularly since then and keep setting new records.

According to NetMarketShare's report, Firefox has 8.02% usage share as a "desktop" web browser and 3.39% usage share across all platforms. Globally, the typical Firefox client averages around 5 hours of use per day. Among the top 10 countries, Americans and Russians are the heaviest users, with about 6.0 hours of daily use and 5.5 hours of daily use, respectively. On the opposite end of the spectrum, Italy and India show the lowest daily use, about 3.7 hours.

Mozilla Firefox went open-source in 1998, and has been a bastion of the open-source community ever since. For many years it was the default web browser on almost all Linux distros. It markets itself as a fast, private, and safe web browser. It does not send your private info to its servers or any third-party partners. It only sends statistics that are non-intrusive, anonymous and is used only to improve Firefox. It also offers features such as tracking protection to make sure that websites do not track your online activities across multiple sites, something that Facebook does very intrusively. Evidently, Firefox has been supporting and promoting the freedom of web users against being tracked and spied on.

Since Firefox takes privacy and security very seriously, the developers have historically been very prompt about patching any vulnerabilities that emerge out of the woodworks. The browser is particularly known for handling SSL certificate revocation extremely well, better than any other browser.

Although effective security measures already exist against many forms of potential attacks, there are still ample ways in which attacks can be made against users, against applications running within the browser, and against the browser itself. For instance, man-in-the-middle attacks are a popular way of extracting information from web communications. One way in which Firefox helps prevent it is by not allowing websites to track your activities. That can render man-in-the-middle attacks trying to fetch confidential information between the website and the user ineffective.

From 2019, Firefox on desktop and Android by default blocks third-party tracking cookies and cryptominers. Cryptominers access your computer's CPU, ultimately slowing it down and draining your battery, in order to generate cryptocurrency.

Firefox also offers protection against some forms of tracking that are more advanced than your usual cookie-based tracking. For instance, a type of script that one may not want to run in their browser are

Fingerprinting scripts. They harvest a snapshot of your computer's configuration when you visit a website. The snapshot can then also be used to track you across the web, an issue that has been present for years. To get protection from fingerprinting scripts Firefox users can turn on 'Strict Mode.' In a future release, they plan to turn fingerprinting protections on by default.

Product Assets

Product Resource	Description	Exploitable properties
Profile	All of the changes one makes in Firefox, like their homepage, what toolbars they use, extensions you have installed, saved passwords and your bookmarks, are all stored in the SQLite databases in a special folder, called a *profile.*	This hosts multiple important data points, but most valuable for exploitation are: Bookmarks, Downloads and Browsing History: ● *places.sqlite* This file contains all your Firefox bookmarks and lists of all the files you've downloaded and websites you've visited. ● *bookmarkbackups* This folder stores bookmark backup files, which can be used to restore your bookmarks. ● *favicons.sqlite* This file contains all of the favicons for your Firefox bookmarks. Passwords: ● *key4.db* ● *logins.json* Autocomplete history: ● formhistory.sqlite This file remembers what you have searched for in the Firefox search bar and what information you've entered into forms on websites. For more information, see Control whether Firefox automatically fills in forms. Cookies: ● cookies.sqlite

		A cookie is a bit of information stored on your computer by a website you've visited. Usually this is something like your site preferences or login status. Cookies are all stored in this file. This data storage can be hacked into using an open-source program called the SQLite Database Browser, and data can be browsed through and manipulated.
Telemetry data	Accessible by typing **about:telemetry** in the browser's URL address bar. The page shows deeply technical information about the user.	From this telemetry data, the user's browser settings, installed add-ons, OS/hardware information, browser session details, and running processes can be extracted for browser fingerprinting. Browser fingerprinting is an investigative method that involves matching browser activity to identify individual users. It can be used to trace online activities to a specific person, and as such, can represent a violation of privacy. Even using a VPN might not be enough to avoid browser fingerprinting.
Accessing malicious webpages	If a webpage is deemed to be in some way malicious by the Safe Browsing online reputational service, the user can choose to ignore the warning and load the page anyway.	There is a risk that the blocked page will attempt to run malware, or the user will fall victim to social engineering attacks that result in identity theft.
Lack of sufficient sandboxing	As of 23rd October 2020, Mozilla is experimenting with a new feature for Firefox that's intended to improve security by isolating each site's content within its own sandbox. But it has not been released yet. So site isolation has yet to	Site isolation is crucial because without it, a malicious site can use a security vulnerability to access sensitive data from another site open at the same time.

	be effectively implemented.	
Password manager: Lockwise	By default, Firefox only encrypts passwords when stored on their servers for syncing between devices. If you want to also encrypt them locally so that other processes running in your user profile cannot read them, then you need to set a master password in the Firefox settings. This has to be done separately for each device using the Firefox account.	A master password can add a layer of security to Firefox on a PC, but most users would be unaware of this functionality. So the locally stored passwords can become easily available to any process running in the same user account on the local machine Moreover, the mobile version of Firefox doesn't have this functionality. So anyone who gains access to a Firefox account will also gain access to its Lockwise passwords even if you have a master password set up on one of your devices.

Architectural Overview

Firefox employs a layered architecture.

The top layer makes up the UI, which relies on the remainder of the framework to render the program. Through this layer, the client communicates with the Firefox framework.

The UI calls upon the subsequent layer. This layer contains both the Gecko and Necko subsystems, which are actualized as a Virtual Machine layered with funnel and channel components.

At last, the most reduced layer is the Display Backend. The Display Backend gives GUI formatting information to Gecko with the goal that pages can be correctly rendered on any stage as required.

The Gecko subsystem serves more than just a search engine and rendering engine. But its primary function is to render web content, such as HTML, CSS, XUL, JavaScript on the user's screen or print it. A couple of its prime components include:

Prionti Nasir
SWEN 331

1. SpiderMonkey: This is FireFox's JavaScript engine written in C/C++. It is an interpreter that works on a full scope of JavaScript values. Its parts are an interpreter, a compiler, two just-in-time compilers, a decompiler, garbage collection, and a standard library. SpiderMonkey gets JavaScript code to be interpreted before sending the interpreted code back to the Content Model for additional rendering. SpiderMonkey additionally contains a couple of open APIs with the goal that different applications can use SpiderMonkey for JavaScript support.

   If a developer introduces bugs in the implementation of this engine, it may result in exploitable vulnerabilities: extracting information from the garbage collector, manipulating the information being rendered etc.

   A simple Google search about SpiderMonkey vulnerabilities led me to multiple bugs and issues in the subsystems some of which even required developers to patch zero-day.

   There are many resources online about attacking just-in-time compilers - a crucial part of SpiderMonkey - as well:

   https://doar-e.github.io/blog/2018/11/19/introduction-to-spidermonkey-exploitation/

   https://www.nccgroup.com/globalassets/resources/us/presentations/documents/attacking_clientside_jit_compilers_paper.pdf

2. Necko: This is the main networking library of Mozilla Firefox. It is responsible for the transport of information from an area on the web to different segments of Firefox, which will render the information into a structure that can be shown by the UI layer. Necko can transfer information through the generally utilized locator known as URL (Uniform Resource Locator). Necko is equipped to deal with various kinds of information, including http, https and ftp.

   A look at Firefox's recent architectural goals shows that Necko is perilously coupled with the rest of the codebase even though it can ideally serve as a standalone component. Moreover, because of its "dependency on the profile (both for reading state and as a key for the disk cache) and flushing files to disk for prefs, it's not safe to use Necko from two distinct processes simultaneously for the same application." It is also not possible to make a network request without touching the disk. These limitations in Necko can be exploited by malicious code.

   Furthermore, Necko is planning to employ process isolation - a major concern harbored by the Firefox community. Firefox periodically has crashes due to anti-virus and other external actors modifying the network stack. In addition, new protocols are a potential surface area for crashes and attacks. Running network code in a separate process would help mitigate crashes and attacks. This is still in development.

More information about core architectural modules can be found here:
https://wiki.mozilla.org/Modules/Core

On top of this architecture of Firefox, plug-ins can be considered a separate layer. This layer can introduce many security issues.

Figure 1 shows the number of vulnerabilities reported in each browser and its average usage (CVE 2018; StatCounter 2018). In general, except for Firefox, the number of vulnerabilities is tied to the browser usage percentage because more vulnerabilities would be found and reported by more users. Chrome being the most popular browser has a predictably higher number of vulnerabilities.The most anomalous behaviour is shown by Firefox's numbers, what with it having a low percentage usage but a high number of vulnerabilities. This is possibly due to the flexibility that Firefox offers for using extensions and plug-ins which are mostly deprecated in other browsers and over time naturally (Sengupta and Park 6). The interfacing of Firefox and third-party plug-ins which can turn out to be visibly vulnerable opens the door for exploitation. Plug-ins enable users to have extra features that the browser inherently does not have. Most of the vulnerabilities in a browser are due to interactions between the browser and the plug-ins. Firefox provides a way where the user can enable the browser to check the status of all the installed plug-ins and keep them updated on its own. This reduces the hassle of checking for outdated plug-ins by a user. It has this feature enabled in the browser's default state where it does not allow out-of-date plug-ins to run and force the user to update it. Yet, the flexibility Firefox provides in terms of installation of relatively old and deprecated plug-ins continues to pose threats.

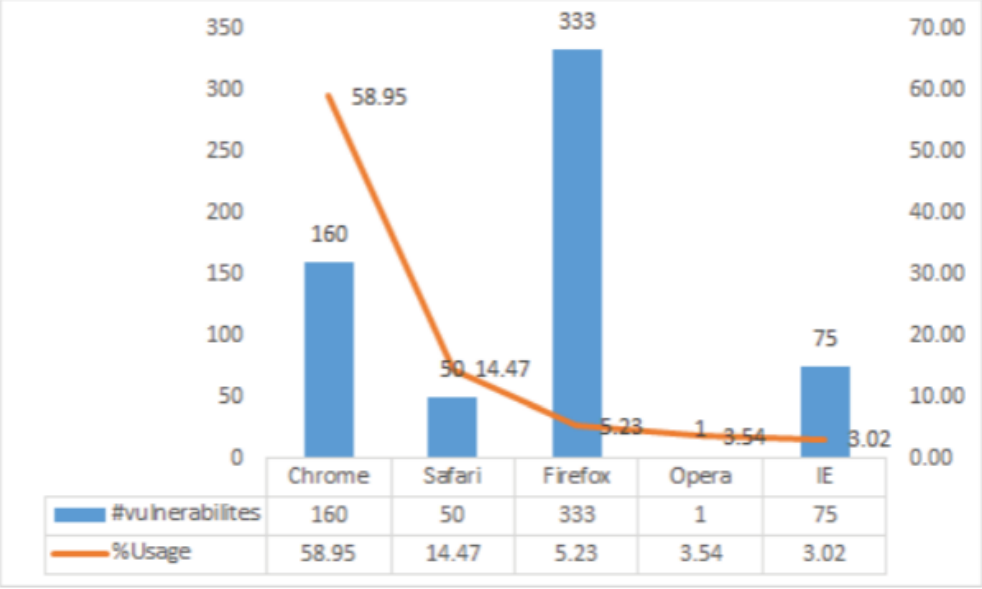

Figure 1: Number of vulnerabilities vs. usage for each browser

Prionti Nasir
SWEN 331

Example Attacks

Although effective security measures already exist in Firefox against many forms of potential attacks, there are still ample ways in which attacks can be made against users, against applications running within the browser, and against the browser itself.

For instance, man-in-the-middle attacks are a popular way of extracting information from web communications. One way in which Firefox helps prevent it is by not allowing websites to track your activities. It also displays warnings if a webpage is deemed to be in some way malicious by the Safe Browsing online reputational service. But the user can choose to ignore the warning and still load the page. In that case the page can run malware and end up being able to garner information from the user anyway. The page can also resemble a safe, familiar website and lead the user to believe they are where they intended to be, and gather data from them. Any such successful attacks would cause violation of confidentiality as well as integrity.

There are quite a few avenues for attackers to extract in-store or in-state data. All of the changes one makes in Firefox, like their homepage, what toolbars they use, extensions you have installed, saved passwords and your bookmarks, are all stored in the SQLite databases in a special folder, called a *profile.* This hosts multiple important data points, but most valuable for exploitation are:

Bookmarks, Downloads and Browsing History:

- *places.sqlite*
  This file contains all your Firefox bookmarks and lists of all the files you've downloaded and websites you've visited.
- *bookmarkbackups*
  This folder stores bookmark backup files, which can be used to restore your bookmarks.
- *favicons.sqlite*
  This file contains all of the favicons for your Firefox bookmarks.

Passwords:

- *key4.db*
- *logins.json*

Autocomplete history:

- formhistory.sqlite
  This file remembers what you have searched for in the Firefox search bar and what information you've entered into forms on websites. For more information, see Control whether Firefox automatically fills in forms.

Cookies:

- cookies.sqlite
  A cookie is a bit of information stored on your computer by a website you've visited. Usually this is something like your site preferences or login status. Cookies are all stored in this file.

This data storage can be hacked into using an open-source program called the SQLite Database Browser, and data can be browsed through and manipulated.

This data storage can be hacked into using an open-source program called the SQLite Database Browser, and data can be browsed through and manipulated. Firefox also keeps telemetry data, accessible by typing **about:telemetry** in the browser's URL address bar. The page shows deeply technical information about the user. From this telemetry data, the user's browser settings, installed add-ons, OS/hardware information, browser session details, and running processes can be extracted for browser fingerprinting. Browser fingerprinting is an investigative method that involves matching browser activity to identify individual users, and data gathered can be used to trace online activities to a specific person. Even using a VPN might not be enough to avoid browser fingerprinting.

Lack of sufficient sandboxing has been an oft-discussed problem in the Mozilla community. As of 23rd October 2020, Mozilla is experimenting with a new feature for Firefox intended to improve security by isolating each site's content within its own sandbox. It has not been released yet, and so site isolation has yet to be effectively implemented. Until the eventful day arrives, a malicious site can use a security vulnerability to access sensitive data from another site open at the same time.

Firefox uses Lockwise as a password manager for user accounts. By default, it only encrypts passwords when stored on their servers for syncing between devices. If you want to also encrypt them locally so that other processes running in your user profile cannot read them, then you need to set a master password in the Firefox settings. This has to be done separately for each device using the Firefox account. A master password can add a layer of security to Firefox on a PC, but most users would be unaware of this functionality. So the locally stored passwords can become easily available to any process running in the same user account on the local machine Moreover, the mobile version of Firefox doesn't have this functionality. So anyone who gains access to a Firefox account - be it through physical methods of spying or through more technical methods like man-in-the-middle interceptions - will also gain access to its Lockwise passwords even if you have a master password set up on one of your devices. This would of course amount to a breach of confidentiality and integrity.

There are also ways in which surrounding applications can lead to exploitation. We know that Firefox offers great flexibility to the user for installing extensions and plug-ins which can happen to be deprecated in other browsers. The interfacing of Firefox and third-party plug-ins which can turn out to be visibly vulnerable opens the door for exploitation. Plug-ins enable users to have extra features that the browser inherently does not have. Most of the vulnerabilities in a browser are due to interactions between the browser and the plug-ins. Plug-in developers are often not security experts and write buggy code that can be exploited by malicious web site operators. If such an attacker can exploit an extension vulnerability, the attacker can usurp the extension's broad privileges and install malware on the user's machine, launch unwanted processes, perform cross-extension exploitation and so on. Such attacks can be a threat to confidentiality, integrity and availability.

Attackers can also tap into the browser's codebase itself. For instance, SpiderMonkey, FireFox's JavaScript engine written in C/C++, consists of an interpreter, a compiler, two just-in-time compilers, a decompiler, garbage collection, and a standard library. SpiderMonkey gets JavaScript code to be interpreted before sending the interpreted code back to the Content Model for additional rendering. SpiderMonkey additionally contains a couple of open APIs with the goal that different applications can use SpiderMonkey for JavaScript support. If a developer introduces bugs in the implementation of this engine, it may result in exploitable vulnerabilities, causing attackers to be able to extract information from the garbage collector, manipulate the information being rendered etc. This can lead to violation of availability and confidentiality.

Vulnerability History

CVE-2020-6819: Use-after-free while running the nsDocShell destructor

On April 3, Mozilla published advisory 2020-11 for Mozilla Firefox and Mozilla Firefox Extended Support Release (ESR). The advisory includes fixes for two critical zero-day vulnerabilities, both of which were exploited in the wild as part of targeted attacks. One of the two is CVE-2020-6819.

CVE-2020-6819 is a use-after-free memory issue/vulnerability owing to a race condition emerging when the nsDocShell destructor is running. Successful exploitation could allow a remote attacker to execute arbitrary code. The highest threat from this vulnerability is to data confidentiality and integrity as well as system availability. Based on the GitHub commit history for the nsDocShell.cpp file, it seemed that the issue existed due to the mContentViewer not being released properly.

This issue was first reported 8 months ago, promptly handled within a day and closed as one can note on both Bugzilla and GitHub.

Upon code inspection, it was found that "nsCOMPtr" did not set "mRawPtr" to nullptr in its dtor, so if somehow something managed to access that memory after dtor ran, unexpected things would happen easily. It is interesting to see how the smallest of errors in memory allocation and deallocation can introduce such critical vulnerabilities. It is then important to realize that clean coding practices that are emphasized awfully frequently - things like allocating memory precisely, deallocating/freeing up memory after use and so on - but are often overlooked need to be adhered to in order to prevent data leaks and other vulnerabilities open to exploits.

CVE-2020-15650

Given an installed malicious file picker application, an attacker was able to overwrite local files and thus overwrite Firefox settings. However, it would not be able to access the previous profile. This issue only affected Firefox for Android. Other operating systems were unaffected. This vulnerability affected Firefox ESR < 68.11.

This vulnerability was reported first by Pedro Oliveira and marked to have a medium impact level. This vulnerability affected an unknown function of the component *File Picker Handler*. The manipulation with an

unknown input leads to a privilege escalation vulnerability. As a result, it is known to affect confidentiality, integrity, and availability. Moreover, this attack could be initiated remotely and no form of authentication would be required for a successful exploitation.

On July 14, 2020 Arturo Mejia opened an issue on GitHub that referenced the relevant bug. On July 16, he "improved file picker" as noted in this commit and implied that this fix would close the issue. On July 17, he added the issue to the 'in-progress' tab in A-C Android Components Sprint Planning. Some developers promptly went on to discuss the issue further on the pull request checklist here. This happened within the span of a day, and on July 17 itself the issue was moved to 'done' from 'in-progress.'

Here is a sneak peek into a part of the fix that was introduced in the FilePicker and kotlin/String files:

```
        if (intent?.clipData != null && request.isMultipleFilesSelection) {
            intent.clipData?.run {
                val uris = Array<Uri>(itemCount) { index -> getItemAt(index).uri }
+               // We want to verify that we are not exposing any private data
+               val sanitizedUris = uris.removeUrisUnderPrivateAppDir(container.context)
+               if (sanitizedUris.isEmpty()) {
+                   request.onDismiss()
+               } else {
+                   request.onMultipleFilesSelected(container.context, sanitizedUris)
+               }
            }
        } else {
            val uri = intent?.data ?: captureUri
            uri?.let {
+               // We want to verify that we are not exposing any private data
+               if (!it.isUnderPrivateAppDirectory(container.context)) {
+                   request.onSingleFileSelected(container.context, it)
+               } else {
+                   request.onDismiss()
+               }
+           } ?: request.onDismiss()
        }
```

```
  fun String.sanitizeURL(): String {
      return this.trim()
  }
+
+ /**
+  * Remove any unwanted character from string containing file name.
+  * For example for an input of "/../../../../../../directory/file.txt" you will get "file.txt"
+  */
+ fun String.sanitizeFileName(): String {
+     return this.substringAfterLast(File.separatorChar)
+ }
```

Prior to these changes, there was no sanitization for filenames. There were no checks for whether an Uri is under the application private directory (android) either. Post-fix, as we can see, one snippet allows for returning sanitized filenames to caller functions, and the other only handles a request and exposes data if Uris do not fall under the same. The latter patch was actually added preemptively, as noted in here, just in case they introduce support for file:// Uris in the picker later, and forget to add this sanitization at that time. Evidently, even the most minute of validations and sanitizations that can exist in between a request and our response to said request, should be implemented because they are crucial in ensuring that we protect our data from exploitation.

A list of known vulnerabilities in Firefox is outlined here:
https://www.mozilla.org/en-US/security/known-vulnerabilities/firefox/

Mozilla for iOS: https://github.com/mozilla-mobile/firefox-ios

Mozilla for Android: https://github.com/mozilla-mobile/fenix

Mozilla organization's code on GitHub with various tools and platforms that support Firefox:
https://github.com/mozilla

The development team overseeing the virtual software community through Firefox's IRC channels:
https://wiki.mozilla.org/Firefox/Team/whois

Prionti Nasir
SWEN 331

Works Cited

Sengupta, Siddhartha, and Joon Park. *An Analysis of Security Features on Web Browsers*. Reading: Academic Conferences International Limited, 2019. *ProQuest.* Web. 18 Nov. 2020.

Prionti Nasir
SWEN 331