Thunderbird

CVE-2016-1966

Context

When Firefox handles **NPAPI plug-ins that create multiple objects** of type NPObject that **needs to be wrapped with an Object Wrapper**.

Problem

We believe there to be an incorrect assumption **regarding the purpose of a certain variable assignment** which is assumed to be obsolete. The 'entry' variable is a pointer to an entry inside the data storage of the global 'sNPObjWrappers' (which keeps track of the **object wrappers** used in the application). This may cause the **NPAPI subsystem to crash.** The high-level PoC to trigger the vulnerability and cause a crash is as follows:

1. write a NPAPI plug-in which has a function that creates and returns a new NPObject every time it is called; 2. call that function in a loop from Javascript. The **browser will likely crash** when a HashTable Object resizes its underlying data storage."

Solution

Fix an erroneous **nsNPObjWrapper** assertion.

Codes

"Application crash", "Object wrappers"

CVE-2013-1713

Context

In the plugin extensions (when **checking the principal** when validating URI loads of extensions)

Problem

The InstallTrigger component can **use the wrong principal when validating URI loads**. It was happening because this component was **grabing the origin information from the outer window**. This is a potential concern in other javascript components that use the document of the window they're accessible from to perform checks against URLs before performing sensitive actions, and could also potentially be used to **bypass the same origin policy** and other all around nastiness.

Solution

Fix is to **get the principal information from the right context**.

Codes

"Cross-site scripting (XSS)", "Incorrect origin check"

CVE-2013-0747

Context

Can **confuse PluginHandler Event** by listening for mutation events.

Problem

JavaScript error: chrome://browser/content/browser.js, line 10437: iconStatus is null

```
>          let installStatus = doc.getAnonymousElementByAttribute(plugin, "class", "installStatus");
>          installStatus.setAttribute("status", "ready");
```

```
>          let iconStatus = doc.getAnonymousElementByAttribute(plugin, "class", "icon");
>          iconStatus.setAttribute("status", "ready");
```

The page **gets an event whose originalTarget is an anonymous DIV**. It is not expected that the page be able to get a reference to the anonymous content. • Content pages shouldn't be able to access native anon content. There used to be an exception if that happened. A dedicated attacker could turn it into something pretty serious by rearranging the anonymous DOM and **clickjacking plugin install prompts.**

Solution

The fix was to add a `<binding native="true">` attribute which would **force the pluginProblem XBL subtree to be considered native-anonymous instead of just anonymous**, which would prevent access from content script.

Codes

"Same-Origin Policy Bypass", "Incorrect origin check"

CVE-2012-4194

Context

Location can be spoofed using |`valueOf`|

Problem

When Adobe Flash Player **checks the page location to apply the SOP (Same-Origin Policy)**, it reads the return value of javascript:top.location+"__flashplugin_unique__". When an object is joined with a string, its lvalueOfl method is called before ltoStringl, and **content can redefine the former**. This appears to have regressed in Firefox v16.0.1.

In short, the **property can be altered to gain access to attributes** that are not supposed to be accessed.

Solution

Prevent **shadow of built-in location.valueOf**.

Codes

"JS Objects Isolation", "Cross-site scripting (XSS)"

CVE-2012-3994

Context

Using **Object.defineProperty to interfere with other add-ons** (or the application).

Problem

The ***Object.defineProperty* can shadow *ltopl***. Plugins may try to access it through *ltop.locationl* -- for instance, Adobe Flash Player opens javascript:top.location+"__flashplugin_unique__" to determine the page origin. And it is possible t**o shadow ltopl using Object.defineProperty**. Incidentally, Google Chrome seems to disallow redefining ltopl.

Solution

Reload **Iframe and re-create docshell**

Codes

"JS Objects Isolation"

CVE-2012-3975

Context

The created document is a data document, so it itself shouldn't load anything. **HTML parser may speculatively load something**. (It shouldn't enable speculative loads for data documents)

Problem

This is a bad bug in the patch for bug 102699. Before that patch, the only codepath that could lead to parsing looked like this, in order: 1) Create a document with the DOMParser's mOriginalPrincipal. 2) Call EnableXULXBL() on the document if needed 3) Call StartDocumentLoad() 4) Set the document's base URI 5) Reset the document's principal to mPrincipal. 6) Feed data into the parser. That sequence of steps was pretty clearly documented (at least in terms of the whole principal dance) and _very_ critical. When that bug was fixed, the XML codepath stayed as above, but HTML codepath was written more like this: 1) Create a document with the DOMParser's mOriginalPrincipal. 2) Feed data into the parser. 3) Call EnableXULXBL() on the document if needed 4) Set the document's base URI 5) Reset the document's principal to mPrincipal. But the whole point of **resetting to mPrincipal is that it MUST happen before any data goes in. Otherwise you're parsing with the system principal**. Also, this is never calling StartDocumentLoad, so afaict it's not setting up whatever state that would normally set up (e.g. the document URI) the same way as the XML path. And it's calling EnableXULXBL() too late, of course. Not like this matters much for text/html. This bug means that **using DOMParser on text/html is pretty unsafe from chrome: It allows whatever string you're parsing to poke any URI it wants**, including ones that web content normally can't access. (On a Unix system **it allows at minimum a DoS attack by reading from file:///dev/tty**.)

Solution

make sure chrome **DOMParser doesn't load external resources**

Codes

"Data leakage", "Trigger access to an arbitrary URL"

CVE-2012-3960

Context

During **deallocation**

Problem

Use-after-free vulnerability in mozSpellChecker::SetCurrentDictionary. mozSpellChecker::SetCurrentDictionary gets called, and then mozHunspell::SetDictionary gets called (which is inlined), which in turn calls into the notification service: . The editor then catches that notification and calls nsEditor::SyncRealTimeSpell, which **can potentially lead into mInlineSpellChecker to get set to null** , which in turn releases its mSpellChecker member , which is a mozSpellChecker which we see on the 1st frame of the freeing call stack. Then, all of this stuff returns, and when we get back to the mozSpellChecker::SetCurrentDictionary frame, *this is dead, so any attempt to call it (such as calling Release on it) will dereference freed memory. Now, I _think_ that **you can't put arbitrary stuff on the stack between the time that the mozSpellChecker object dies and the time that mozSpellChecker::SetCurrentDictionary returns,** but if I'm wrong, and you could do that, then this gives you a very nice **remote exploit**, because the offset of Release in the vtable is pretty well known...

Solution

Part 1: **Make sure that mozSpellChecker's refcount doesn't go down prematurely**; Part 2: **Make sure that nsEditorSpellCheck's refcount doesn't go down prematurely**; Part 3: **Make sure that nsEditorSpellCheck's refcount doesn't go down prematurely**;

Codes

"Arbitrary code execution", "Use after free"

CVE-2012-1956

Context

It occurs when **extensions manipulate the Object.defineProperty as a method to shadow the location object** (aka window.location)

Problem

It is possible to **shadow the location object using Object.defineProperty**. This could be used **to confuse the current location to plugins, allowing for possible cross-site scripting (XSS) attacks.** it means that an attacker can **confuse Flash (or other plugins) into thinking that we're on one domain when, in reality, we're on another one** leading to XSS attacks.

Solution

Create a function that does **security checks specifically for the object** (js::CheckDefineProperty(JSContext *cx, HandleObject obj, HandleId id, HandleValue value, PropertyOp getter, StrictPropertyOp setter, unsigned attrs)).

Codes

"JS Objects Isolation", "Cross-site scripting (XSS)", "Lack of security checks"

CVE-2012-0446

Context

It occurs when frame **scripts that call untrusted objects**.

Problem

Frame scripts **bypass XPConnect security checks when calling untrusted objects**. This allows for **cross-site scripting (XSS) attacks** through web pages and Firefox extensions. Frame scripts run on the special JS context for which we call SetSecurityManagerForJSContext with flags=0, thus **if a frame script calls into an untrusted function, XPConnect does not do proper security checks**.

Solution

The fix enables the **Script Security Manager (SSM) to force security checks on all frame scripts**.

Codes

"Arbitrary code execution", "Cross-site scripting (XSS)", "HTML Injection", "Inject Javascript", "Lack of security checks"

CVE-2011-3001

Context

It occurs as part of a **user-assisted attack**. If you could convince a user to hold down the Enter key--as part of a game or test, perhaps--a **malicious page could pop up a download dialog where the held key would then activate the default Open action**.

Problem

For some file types this would be merely annoying (the equivalent of a pop-up) but other file types have powerful scripting capabilities. And this would provide an avenue for an attacker to exploit a vulnerability in applications not normally exposed to potentially hostile internet content. There are 2 layers of protection against an **unauthorized installation of extensions**: 1) The principal of the opener is checked **against whitelisted domains that are allowed to download the plugin without asking**. If the domain is not trusted, the user **is asked to allow to download the plugin**. 2) When the plugin is downloaded, **the user is asked to confirm the installation**. The first protection can be circumvented by **creating a hidden "Embed" element containing an arbitrary XPI as its "pluginspage" parameter**. The attacker can focus this element while the user holds Enter, causing a number of "Plugin Finder Service" windows to appear. The first window focuses the "Cancel" button and will just close, but all the subsequent ones will set focus on the "Manual Install" button directing to the malicious XPI. As soon as the user releases the key, the browser will start launching multiple windows with the provided URL. The windows will have a ChromeWindow object as their opener, so the user will not be asked to allow to download a plugin. The second protection can be **bypassed due to a logic error in amWebInstallListener.js**. When no window-watcher is registered in Services, this will throw:

Solution

It ensures that window watcher is defined, such that it can **show the install dialog.**

Codes

"Installer", "Silent install of plug-ins", "Arbitrary code execution", "User-assisted attack", "Bypass protection mechanism"

CVE-2010-1585

Context

During Plug-ins execution (protection mechanism **against unsafe javascript**).

Problem

The two ns(X)HTMLParanoidFragmentSink classes are used by nsIScriptableUnescapeHTML **to sanitize (X)HTML by stripping attributes and tags not on a built-in whitelist**. It allows javascript: URLs and other inline JavaScript when the embedding document is a chrome document. While there are no unsafe uses of this class in any released products, extension code could have potentially used it in an unsafe manner. **The sinks attempt to sanitize URLs by calling CheckLoadURI[...]DISALLOW_INHERIT_PRINCIPAL), but unfortunately when the target document is a chrome document (as is common with add-ons) this check allows any URI**. In particular malicious href="javascript:evil()" or <iframe src="data:evil"> can slip through and create sg-critical bugs.

In short, it does not properly **sanitize HTML in a chrome document**, which makes it easier for remote attackers to **execute arbitrary JavaScript with chrome privileges via a javascript.**

Solution

DISALLOW_INHERIT_PRINCIPAL always returned "ok" for system principals. Therefore, they used a **null principal when performing the validation**.

Codes

"Privilege elevation", "Perform security check on unsanitized data"

CVE-2010-0179

Context

When **dispatching events** from the XMLHttpRequestSpy module (a Firebug add-on)

Problem

When accessing this.xhrRequest.onreadystatechange, content functions (QueryInterface, getInterfaces, etc.) can be called. In other words, when add-ons try to get a reference to onreadystatechange, we call getInterfaces on the existing handler (through the nsXPCWrappedJS). Since the application does not properly handle interaction between the **XMLHttpRequestSpy object and chrome privileged objects**, it allows remote attackers to **execute arbitrary JavaScript** via a crafted HTTP response.

In short: Add-ons can get more information from calling functions they're not supposed to because **the application doesn't check for the principal**.

Solution

The fix is **to check the correct principal (origin)**. "If no scripted code is running "above" (or called from) fp, then instead of looking at cx->globalObject, lprincipall is returned."

Codes

"Code Injection", "Added origin check", "Incorrect origin check"

CVE-2008-2806

Context

Mozilla.org distributions on the Mac that can use Java have for some time bundled the Java Embedding Plugin, which allows non-WebKit browsers to use current Java versions on OS X.

Problem

The Firefox Mac OS X Java Plugin (MRJ Plugin) is vulnerable to the 'document.domain' bypass. Document.domain gets/sets the domain portion of the origin of the current document, as used by the same origin policy. By using the document.domain exception to the same origin policy, LiveConnect, which is a feature of web browsers which allows Java applets to communicate with the JavaScript engine in the browser, and JavaScript on the web page to interact with applets, can be used to create arbitrary socket connections.The code appears to use nsIPrincipal::GetOrigin(), an interface to a principal, which represents a security context, which takes document.domain into account.

In short, the issue was with how the plugin for Java applets makes calls to javascript and obtains the origin of the domains, bypassing the same origin policy.

Solution

The fix was to drop any use of GetOrigin() for Java and use the principal's URI

Codes

"Same-Origin Policy Bypass", "Arbitrary code execution", "Incorrect origin check"

CVE-2008-2803

Context

When loading scripts from extensions (function: mozIJSSubScriptLoader.LoadScript)

Problem

It's unsafe to use mozIJSSubScriptLoader.loadSubScript() with non-chrome urls or chrome urls whose scheme/host part contain uppercase characters. Scripts that are loaded in this way do not use implicit XPCNativeWrappers when accessing content, which is used whenever privileged code is used to access unprivileged code. It is used to create a security wrapper that guarantees that the "native" methods/properties of an object will be called (and not the methods overriden by the webpage). As an example, Google Toolbar uses mozIJSSubScriptLoader.loadSubScript() with file: url, and allows an attacker to run arbitrary code with chrome privileges.

Solution

They fixed their logic for obtaining a native object wrapper. The decision of whether the subscript gets XPCNativeWrappers or not will depend on the caller, not the file:// URI.

Codes

"Object wrappers", "Not enforcing permissions"

CVE-2007-3844

Context

When handling Privileges of plug-ins

Problem

window.open("about:blank"); or content.location = "about:blank"; or about:blank when loaded by chrome in these ways has chrome privileges. This behavior could cause security issues in certain extensions that are thinking that about:blank does not have chrome privileges. Imagine an extension that does: 1. Collect urls from content. 2. Load about:blank.

(window.open("about:blank") or content.location = "about:blank") 3. Generate links with the urls and insert those into the about:blank document. When an user clicks a javascript: link in the generated page, the script run with chrome privileges. I'm not sure whether this should be fixed or not. If not, we need to advertise the potential problem. (There is an affected extension on AMO.)

Solution

The three hunks of this patch do the following: 1) **Never allow chrome-privileged data:, javascript:, or about:blank loads in content docshells**. Switch them to **inheriting principals instead** (which is a no-op for about:blank). This behavior is now consistent across all "normal" ways of loading, whereas before window.location allowed chrome javascript: while the nsIWebNavigation APIs, tabbrowser, and setting "src" on s did not. It's still possible to do such loads via manual invocation of nsILinkHandler, but that's not a scriptable API, and doesn't take a principal pointer anyway (it takes a node). This fixes the window.location aspect of this bug. 2) **Don't propagate a system principal as the opener principal to new content windows**. This fixes the window.open() aspect of this bug. Note that we need both, because CreateAboutBlankContentViewer doesn't actually do a load. 3) **Remove now-redundant code in nsFrameLoader**. The only risk here, imo is that this does change the behavior of data: and javascript: URIs loaded from chrome in content windows via window.location. If we want I can try to avoid changing that, but the code would be more complex, and I don't think we want to allow it anyway. The former is certainly not safe.

Codes

"Privilege elevation"

CVE-2006-6499

Context

The problem is in the **JavaScript Engine** (function: jsdtoa)

Problem

When **a plugin decreases the float precision (a global configuration) it triggers a bug in the javascript engine**. In short, a loop in this engine is **controlled by the floating point arithmetic**. Since that is flawed (or rather becomes flawed after the precision is reduced), the loop becomes infinite, and it keeps adding characters to the string until it **crashes**. Detailed report: **Crash in jsdtoa after opening a new window**. There is some very bad logic in jsdtoa, for the case where the double has no fractional component. File: jsdtoa.c See line 2376 /* Do we have a "small" integer? */ In that case there is a loop that converts the double to a string, one digit at a time. In the loop, it looks at the most significant digit, adds that digit to the string, and then removes the digit from the double value. Then, it multiples the value by 10 so shift the digits over to the left. The problem is that **the looping logic is dependent on no floating point error being introduced**. But, floating point error is introduced when two doubles of different magnitudes are subtracted, which is done here: 2388: d -= L*ds; The check to exit the loop looks like this: 2410: if (!(d *= 10.)) break; It compares the double value to zero. Unfortunately, there is floating point error introduced, and the value of "d" never gets down to zero. Therefore, the loop becomes infinite, and we keep appending characters to the output string, ignoring the specified size of the buffer. (The buffer size test was done previously, based on the number of digits that it planned to place in the buffer.). My proposed fix is rather simple. Since we know up front how many digits we have to write out, we can use that number to specify the number of times that we loop, rather than depending on errorless floating point math. See the diff between the original and modified files attached. I changed the "for" line: From: for(i = 1; ; i++) { To: for(i = 1; i<=k+1; i++) { And, I removed the check on the bottom of the for loop. From: if (!(d *= 10.)) break; To: d *= 10.; Furthermore, the logic that is there is very bad. [Description about other problem cut, will file another bug where necessary.]

Solution

They fixed **the calculations in the loop**. [*SIDE NOTE*] *Even though the bug looks like a simple error logic in a loop, from what I saw in the discussions, if they had used the Chrome's approach of isolating javascript objects, this bug would never be exploited. Why? In Chrome, Javascript objects are different (each plugin has its own JS objects), this means that plugins cannot interfere with each other nor the hosting application. Thus, even if a plug-in had decreased the floating precision, the crash would not occur. So the problem seems more like not properly isolating JS objects between plugins and Thunderbird*

Codes

"Overwrite memory"

CVE-2005-0590

Context

The underlying scenario is during an **installation,** in which the application displays a **confirmation dialog** that shows a **spoofed URL**.

Problem

Between not checking for a **spoofed URL** with a username/password, and the unresizeable, unwrapped **dialog for XPInstall**, it's possible to make a fairly convincing **spoofed URL for an XPI with InstallTrigger, due to incorrect parsing of the URL**.

Solution

The solution was to **strip user:pass from URL display**. It added a function that does a preprocessing of the URL (nsXPITriggerItem::GetSafeURLString()).

Codes

"Installer", "Spoofed origin of an install request", "Tricking user into installing a malicous plug-in"

CVE-2004-0906

Context

It happened at the **XPInstall Engine**

Problem

After installation, **many installed files are GROUP and WORLD writable**, even though the installer was executed with the umask value 0022. In details: When **files are installed via XPInstall, the user's umask is ignored**. The XPInstall engine **should respect the user's umask setting**. The problem code is here: nsZipArchive::ExtractFile http://lxr.mozilla.org/mozilla/source/modules/libjar/nsZipArchive.cpp#641 nsJAR::Extract http://lxr.mozilla.org/mozilla/source/modules/libjar/nsJAR.cpp#251 both **open the file with 0644 perms and then chmod the file to the appropriate perms, but this sidesteps any umask**. Just to make things more complicated, the nsZipArchive is part of standalone libjar, where PR_Open is defined as fopen in zipstub.h, so passing the file's mode to PR_Open wouldn't work in that situation.

In short, the extraction of extensions files was made **with wrong file permissions, creating world-readable/writeable files**. This allows a **trojan to modify a benign application for malicious purposes**.

Solution

The fix was setting the **permissions on the extraction** function.

Codes

"Installer", "Extracting the plug-in to world-accessible location", "Replace benign plugins by a malicious one"

CVE-2004-0762

Context

During **initialization of extensions**

Problem

If a **malicious Web site can control or predict when and where a user will click,** it can get **them to install software**. Ways in which a Website can **predict user clicks**:

1. A game. 1a. Make the player "pick up" items by clicking them. Measure the speed with which the player moves the mouse and clicks, and once the speed stabilizes, pop up an install dialog just before the player clicks. 1b. Force the player to click at exactly

the right time: a reaction-time test, shoot the monkey, etc. 1c. Convince the player to double-click an object whose location I control. 1d. Tell the player that he has infinite ammo and can shoot by pressing or holding the 'i' button on the keyboard. Pop up an install dialog when the player runs out of ammo.

2. Pop-up hell. 2a. Make the 'x' for the pop-up ad appear just where a security dialog will appear. 2b. Make fake pop-ups out of images so you can measure the victim's reaction time, average mouse acceleration, etc. Get them on the fifth "pop-up".

Solution

They enumerated three possible alternatives, and decided to implement the following:

B) **Add a one-second delay between when the dialog gets focus and when the Install/OK button becomes enabled**. I say "gets focus" rather than "appears" because a site could hide an install dialog as a modal dialog of a background window for a minute and then bring the dialog to the front at a convenient time by closing the window in front.

The other alternatives were (but they're not implemented):

A) When a site tries to install software or calls enablePrivilege, display a status bar message, "This page would like to install software on your computer". Only display the dialog after the user clicks the status bar message. (Err, how do you click a status bar message with the keyboard?) C) If the total time to decide to install and download the xpi are less than five seconds, stall installation (with the "downloading..." dialog still up) until five seconds are up so the user has an extra chance to cancel. I'm worried that users might not know to cancel because they do not realize that they accidentally clicked "install" in a dangerous dialog.

Codes

"Installer", "Not showing install warning dialog", "Silent install of plug-ins", "User-assisted attack"