Firefox

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-2016-1949

Context

Mozilla Firefox before 44.0.2 does not properly restrict the interaction between Service Workers and plugins, which allows remote attackers to bypass the Same Origin Policy via a crafted web site that triggers spoofed responses to requests that use NPAPI, as demonstrated by a request for a crossdomain.xml file.

Problem

NPAPI-initiated network requests can be intercepted by service workers, hence breaking plugin origin expectations

Solution

Make plugin network requests bypass service worker interception

Codes

"Same-Origin Policy Bypass", "Steal data", "Intercept HTTP requests"

CVE-2016-1948

Context

Mozilla Firefox before 44.0 on Android does not ensure that HTTPS is used for a lightweight-theme installation, which allows man-in-the-middle attackers to replace a theme's images and colors by modifying the client-server data stream.

Problem

To install a lightweight theme, we listen for custom events from addons.mozilla.org, and we check the document URI to make sure these events are actually coming from addons.mozilla.org. However, we don't check the scheme to ensure it's https. This creates the opportunity for a malicious party to spoof the DNS entry for http://evil.addons.mozilla.org/ and from there still act with the same privileges as https://addons.mozilla.org/ and install themes.

Solution

Add the **HTTPS check for schemes**.

Codes

"Installer", "Man-in-the-middle attack", "Spoofed origin of an install request"

CVE-2015-7223

Context

The WebExtension APIs in Mozilla Firefox before 43.0 **allow remote attackers to gain privileges**, and **possibly obtain sensitive information or conduct cross-site scripting (XSS) attacks**, via a crafted web site.

Problem

Firefox **doesn't check that a document belongs to an extension before injecting APIs into it**. In the case of background pages, it **continues injecting APIs into new window globals even after the first load.** This means that if a background page navigates to a remote web page, that page has the **full privileges of the extension**. Remote URLs loaded into popups get the **same elevation of privileges too.**

Solution

Don't inject WebExtension APIs into documents without WebExtension principals

Codes

"Privilege elevation", "Steal data", "Cross-site scripting (XSS)", "Lack of security checks"

CVE-2015-7196

Context

Mozilla Firefox before 42.0 and Firefox ESR 38.x before 38.4, when a **Java plugin is enabled, allow remote attackers to cause a denial of service** (incorrect garbage collection and application crash) or **possibly execute arbitrary code** via a crafted Java applet that deallocates an in-use JavaScript wrapper.

Problem

A Java Plugin **destroys an object** in a thread other than the main thread, which causes its **buffer store entry not to be removed** in the main thread. This causes the GC to **crash** when it encounters the buffer store entry.

It possible result in **arbitrary code execution** via a crafted Java applet that deallocates an in-use JavaScript wrapper.

Solution

Add a MOZ_CRASH if the thread where the object is destroyed is not the main one.

Codes

"Application crash", "Arbitrary code execution"

CVE-2015-7187

Context

The Add-on SDK in Mozilla Firefox before 42.0 misinterprets a "script: false" panel setting, which makes it easier for remote attackers to conduct cross-site scripting (XSS) attacks via inline JavaScript code that is executed within a third-party extension.

Problem

When creating extensions, if it specifies that it will not use JS (allow: { script: false }), it can still write in-line JS code in HTML pages which will be executed. This means that attackers can perform XSS attacks through in-line JS in extensions. The problem was apparently in a default true to allow inline JS

Steps to reproduce:

1. Create a browser extension for Firefox.

2. Create a panel with script: false:

```
function createPanel () {
    var sd = require("sdk/self").data;
    myPanel = require("sdk/panel").Panel({
            width: 640,
            height: 522,
            allow: { script: false },
            contentScriptFile: [ sd.url("full.js"), sd.url("popupscript.js") ]
```

3. Pull an external html page into the panel and include: <script>alert('this shouldnt happen');</script>

Issue was found on firefox 40, but I believe it exists prior.

Actual results:

Alert box with "this shouldnt happen" appears.

Expected results:

Inline script should have been ignored due to this flag:

allow: { script: false }

Solution

Check for the allow script specification before running any sort of JS.

Codes

"Cross-site scripting (XSS)", "Bypass protection mechanism"

CVE-2015-4498

Context

The add-on installation feature in Mozilla Firefox before 40.0.3 and Firefox ESR 38.x before 38.2.1 allows remote attackers to bypass an intended user-confirmation requirement by constructing a crafted data: URL and triggering navigation to an arbitrary http: or https: URL at a certain early point in the installation process.

Problem

When the page is redirecting or navigating by ana href link, JavaScript or server redirect, Firefox throws the '*Firefox prevented this site (site.com) from asking you to install software on your computer.*' warning at the user, which needs to explicitly be accepted for the add-on to continue installing.

There is, however, a simple vulnerability that lets an **attacker bypass this dialog**, which allows a rather nasty attack on the user. Basically, there is one exception in which the dialog will not be shown, which is if the user pastes or copies the direct link/URL in the URL bar. The user could also just click on links, redirects etc that lead to this page, because a data uri redirected to the page which itself redirects with a 'page moved header' to the location of the add-on, will disrupt that 'chain' and **installation of the add-on will start without the dialog**, as if the user typed it in directly.

Solution

Check that the triggeringPrincipal subsumes the principal of the document loaded in the tab that started the install. The fix is to **block cross-origin add-on install requests**.

Codes

"Not showing install warning dialog", "Silent install of plug-ins", "Block cross-origin install requests"

CVE-2015-4495

Context

PlayPreview (PDF Viewer)

The PDF reader in Mozilla Firefox before 39.0.3, Firefox ESR 38.x before 38.1.1, and Firefox OS before 2.2 allows remote attackers to **bypass the Same Origin Policy, and read arbitrary files or gain privileges**, via vectors involving crafted JavaScript code and a native setter, as exploited in the wild in August 2015.

Problem

Security researcher Cody Crews reported on a way to **violate the same origin policy and inject script into a non-privileged part of the built-in PDF Viewer.**

This exploit **allows attackers to read and copy information on victim's computer**, once they view the web site crafted with this exploit.

Proof of Concept: Create a index.html and copy and paste the following html into it:

Test: Run the index.html (Make sure the main.js is in the same directory) and we should be able to see the directory listing.

Solution

WIP **disables native PDF plugins** (In reply to Boris Zbarsky [:bz] from comment #1)

"So I tried to hack on this last night but discovered that at least on my Mac > we're treating the Adobe PDF plug-in as always disabled, which is not very > helpful for debugging.

If you can tell me how to detect the "internal PDF viewer is enabled" state,

I can try to put up some patches that implement this proposal for testing... The PDF viewer detects if it is enabled via multiple configuration flags. See http://mxr.mozilla.org/mozilla-central/source/browser/extensions/pdfjs/content/PdfJs.jsm#275 . I think the best way to detect is PDF viewer is **enabled is to check stream converter**. I created a WIP patch (see attachment) -- I will check if the test page works on Windows with internval PDF viewer on/off soon. Remove PlayPreview usage from PDF viewer"

Codes

"JS Objects Isolation", "Same-Origin Policy Bypass", "Data leakage"

CVE-2015-2709

Context

Multiple unspecified vulnerabilities in the browser engine in Mozilla Firefox before 38.0 allow remote attackers to cause a **denial of service** (memory corruption and application crash) or **possibly execute arbitrary code** via unknown vectors.

Problem

A missing nullptr check in GetDocument and a missing check of the content pointer before initializing instanceOwner cause Firefox to crash.

Solution

Add null check

Codes

"Application crash", "Check object is not null"

CVE-2015-2706

Context

Race condition in the AsyncPaintWaitEvent::AsyncPaintWaitEvent function in Mozilla Firefox before 37.0.2 allows remote attackers to execute arbitrary code or cause a denial of service (use-after-free) via a crafted plugin that does not properly complete initialization.

Problem

Failed initialization of the plugins causes a race condition, because the destroyer of an object linked to the plugin is not called, which causes the potential for UAF.

Solution

Destroy the owner object - related to the plugin - if the plugin fails to initialize.

Codes

"Application crash", "Object deallocation", "Race condition", "Failed initialization of the plugins", "Use after free"

CVE-2015-0812

Context

Mozilla Firefox before 37.0 does not require an HTTPS session for lightweight theme add-on installations, which allows man-in-the-middle attackers to bypass an intended user-confirmation requirement by deploying a crafted web site and conducting a DNS spoofing attack against a mozilla.org subdomain.

Problem

Extended browser access is granted to some Mozilla-owned domains, such as addon management for addons.mozilla.org, wich is defined by the default_permissions file, as I understood it. But unlike the "UITour" permissions granted for www.mozilla.org, this API is not restricted to the https:// protocol. This should not be a security risk per se, since addons.mozilla.org provides HSTS headers and is included in the static HPKP pinning list anyways, which should render any attempt to tamper with plain HTTP traffic impossible. Nevertheless sub-subdomains do not seem to enforce SSL here. So a MITM can spoof the DNS entry for http://evil.addons.mozilla.org/ and from there still act with the same privileges as https://addons.mozilla.org/.

Solution

Add the HTTPS check for schemes. Side note: Comment #2: "AFAIK, the only reason not to enforce that are historical - changing it may break existing legitimate 3rd party install sites. And specialized cases like enterprise environments, I guess - but I don't have any data to back that up (and we can work around that anyway). We do perform a reasonably strict HTTPS check in the install phase, but we allow a bypass using a hash check. That's designed for a MitM of the actual install file, not the install request. If a MitM can control the install request site, then it would likely be pointing at a file on a site it controls anyway - so our checks in the install phase won't help. I think the benefits outweigh the costs of what we may break. Saying that, needinfo on Dave in case he has more historical context around this. In the mean time, I'll work up a patch."

Codes

"Installer", "Man-in-the-middle attack", "Spoofed origin of an install request"

CVE-2014-8643

Context

Mozilla Firefox before 35.0 on Windows allows remote attackers to **bypass** the Gecko Media Plugin (GMP) **sandbox protection** mechanism by **leveraging access to the GMP process,** as demonstrated by the OpenH264 plugin's process.

Problem

The sandboxed plugin-container.exe process on Windows holds a handle to the parent process. While the **access rights on this handle are somewhat limited** (0x101441) it still **allows us to duplicate handles in the parent process** (PROCESS_DUP_HANDLE). Using DuplicateHandle we can issue a call that duplicate the process handle from the parent process to our current process (-1 a pseudo handles to the parent and current (sandboxed process). The call **will succeed and a new handle to the parent process is created in the sandboxed child**. This handle has full access to the parent which then **allows for executing arbitrary code in the parent** through CreateRemoteThread.

Solution

Do not allow calls to OpenProcess function, which allows handles to be duplicated, after user content has been loaded.

Codes

"Sandbox escape", "Unprivileged process manipulates a high-privileged process", "Privilege elevation"

CVE-2014-1519

Context

PluginModuleParent **may delete its subprocess** before calling MessageChannel::Clear, resulting in badness

Multiple unspecified vulnerabilities in the browser engine in Mozilla Firefox before 29.0 and SeaMonkey before 2.26 allow remote attackers to cause a **denial of service** (memory corruption and application crash) or **possibly execute arbitrary code** via unknown vectors.

Problem

1) PluginModuleParent contains a PluginProcessParent, mSubprocess. When PluginModuleParent is created by static ::LoadModule, It passes parent->mSubprocess->GetChannel() to parent->Open(). [A] 2) PluginModuleParent::Open(), calls PluginModuleParent->mChannel->Open(), which is a MessageChannel. 3) MessageChannel::Open creates a ProcessLink, mLink, and passes it the channel from the subprocess in (1) [B] 4) NP_Initialize **returns an error from the plugin.** We set mShutdown = true without actually calling NP_Shutdown. [E] There are a few other pathways that do something similar [C] [D] (maybe [F], but the callers I checked are caused by channel shutdown) 5) We decide to destroy PluginModuleParent, and call in order: - ~PluginModuleParent - mSubprocess->Destroy() - ~MessageChannel (PluginModuleParent.mChannel destructor) - MessageChannel::Clear - delete mLink - ~ProcessLink - mTransport->set_listener(); mTransport is the channel obtained from mSubprocess in (1) 6) mSubprocess->Destroy() queues a task on the io thread to run Idelete thisl. This task then races with us getting to MessageChannel::Clear. **If it wins the race, MessageChannel.mLink now has a poisoned mTransport, and we crash.** This can be reproduced in a debugger by breaking at [G] and [E]. Fudge the plugin return code at [E]* and stop the main thread at [G] so the iothread wins the race. Backtrace attached, which appears identical to the bug 974933 crashes: https://crash-stats.mozilla.com/report/index/c68aee10-a11f-4191-ab54-298b12140227 * I'm not sure about this part -- we can't call NP_Shutdown or MessageChannel::Clear() will prevent the race. Any of the abnormal-failure paths that set lmShutdown = truel without calling ::Clear might be triggering this. [A] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginModuleParent.cpp#97 [B] http://dxr.mozilla.org/mozilla-central/source/ipc/glue/MessageChannel.cpp#296 [C] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginModuleParent.cpp#1196 [D] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginModuleParent.cpp#111 [E] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginModuleParent.cpp#1228 [F] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginModuleParent.cpp#731 [G] http://dxr.mozilla.org/mozilla-central/source/dom/plugins/ipc/PluginProcessParent.cpp#82

Solution

Close the channel when aborting before successful init So mShutdown tracks if our channel has died, and the destructor calls NP_Shutdown if it has not. This is circumvented, however, by error paths that set it to true during init to indicate that NP_Shutdown shouldn't be called. All of these paths except one in LoadModule also leave the channel open erroneously, so just calling Close() when we want to shutdown without calling the plugin should fix the issue. The one case in LoadModule sets mShutdown because the object dies before calling Open(). It doesn't look like there's a sane way to assert that MessageChannel was cleaned up -- double-calling Close() is a runtime abort, and MessageChannel::Connected() doesn't guarantee we're not somewhere between closed and connected (and is private anyway). Is there an assertion we could add to the destructor, or is it worth modifying MessageChannel to add one? Given that this patch restores mShutdown to properly track the "After Open() before Close()" state I'm fairly confident this can't happen in other ways in the existing code at least.

Codes

"Application crash", "Check object is not null", "Arbitrary code execution", "Memory corruption"

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-2013-0798

Context

Mozilla Firefox before 20.0 on Android uses world-writable and world-readable permissions for the app_tmp installation directory in the local filesystem, which allows attackers to modify add-ons before installation via an application that leverages the time window during which app_tmp is used.

Problem

The installation of the Firefox for Android (FFA) makes app_tmp directory world readable and writable(777). With this configuration other applications (might be malicious) can replace any addons installed through FFA. This leads installing mal-addons without any awareness from users.

Solution

move the about:memory dumps to somewhere on /sdcard (change the tmp directory) and set the right permissions. Note: It's complicated in Android to change permissions for existing directories, that's why the workaround (delete the old one and create a new one with the right permissions)

Codes

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

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-2012-3973

Context

The debugger in the developer-tools subsystem in Mozilla Firefox before 15.0, when remote debugging is disabled, does not properly restrict access to the remote-debugging service, which allows remote attackers to execute arbitrary code by leveraging the presence of the HTTPMonitor extension and connecting to that service through the HTTPMonitor port.

Problem

If remote debugging is disabled, but HTTPMonitor is enabled, a remote user can connect to and use the remote debug service.

Solution

having the server code take the remote-enabled flag into consideration before opening the socket.

Codes

"Data leakage", "Lack of security checks", "Bypassing restrictions on debugging remotely"

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-2011-3004

Context

The JSSubScriptLoader in Mozilla Firefox 4.x through 6 and SeaMonkey before 2.4 **does not properly handle XPCNativeWrappers** during calls to the loadSubScript method in an add-on, which makes it easier for remote attackers to **gain privileges** via a crafted web site that leverages certain **unwrapping behavior.**

Problem

When loading JS from add-ons, Firefox unwraps the wrapper objects and the code that is supposed to receive a wrapped window, it's now handed the underlying Window allowing that Window's code to **possibly catch things like expando sets and then inject its own code into the privileged call**

Solution

Use a Chrome sandbox object when loading unprivileged JS code

Codes

"Privilege elevation", "Object wrappers"

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-2011-2370

Context

Mozilla Firefox before 5.0 does not properly enforce the whitelist for the xpinstall functionality, which allows remote attackers to trigger an installation dialog for a (1) add-on or (2) theme via unspecified vectors.

Problem

In the install functionality, it's possible to redefine window.location. Thus, content code can control this.window.location.href. Moreover, this.window is not being wrapped.

Solution

Stops using the unwrapped window and also switches to using document.documentURIObject throughout the installation process.

Codes

"Installer", "Spoofed origin of an install request"

CVE-2011-0076

Context

Sandbox

Unspecified vulnerability in the Java Embedding Plugin (JEP) in Mozilla Firefox before 3.5.19 and 3.6.x before 3.6.17, and SeaMonkey before 2.0.14, on Mac OS X allows remote attackers to bypass intended access restrictions via unknown vectors.

Problem

Vulnerability in the Java Embedding Plugin (JEP) on Mac OS X allows remote attackers to bypass intended access restrictions via unknown vectors.

We have discovered a vulnerability in the JEP or LiveConnect java bridge in Firefox. We initially investigated this as a bug in the Java distribution, but it now seems to be a Mozilla-specific problem. Therefore I'm giving you the proof-of-concept so that you can investigate further. ===== javafs.html ======= ===== cut ======= 1. Put javafs.html (contents per the above) on a web server. 2. Visit that page in Firefox. 3. Note how the script was permitted to get a reference to the java.io.FileSystem class, even though it is declared package-local. I reproduced the issue with Firefox 3.6.13.

Solution

Not provided

Codes

"Not enforcing private code", "Data leakage", "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-2010-1198

Context

Use-after-free vulnerability allows remote attackers to execute arbitrary code via vectors involving multiple plugin instances. Deallocator A flaw was discovered in the way plugin instances interacted. An attacker could potentially exploit this and use one plugin to access freed memory from a second plugin to execute arbitrary code with the privileges of the user invoking the program.

Problem

two plugin instances could interact in a way in which one plugin gets a reference to an object owned by a second plugin and continues to hold that reference after the second plugin is unloaded and its object is destroyed. In these cases, the first plugin would contain a pointer to freed memory which, if accessed, could be used by an attacker to execute arbitrary code on a victim's computer.

Solution

instead of unwrapping an NPObject which is passed to a different instance (NPP), we should double-wrap it. This means that the other plugin would obtain a reference to a nsJSObjWrapper, instead of the other-plugin-implemented NPObject* which is destroyed when the plugin is destroyed.

Codes

"Improper Objects Isolation", "Application crash", "Plug-in interaction"

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-2010-0177

Context

frees the contents of the window.navigator.plugins array while a **reference to an array element is still active**, which allows remote attackers to **execute arbitrary code or cause a denial of service** (application crash) via unspecified vectors, related to a "dangling pointer vulnerability." window.navigator.plugins

Problem

error in the implementation of the window.navigator.plugins object. When a page reloads, the plugins array would r**eallocate all of its members without checking for existing references to each member**. This could result in the **deletion of objects for which valid pointers still exist**. An attacker could use this vulnerability to **crash a victim's browser and run arbitrary code on the victim's machine** Successful exploitation can lead to code execution under the context of the application. This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Mozilla Firefox. User interaction is required to exploit this vulnerability in that a user must be coerced to viewing a malicious document. The specific flaw exists within the way the application implements the window.navigator.plugins array. Due to the application freeing the contents of the array while a reference to one of the elements is still being used, an attacker can utilize the free reference to call arbitrary code. Successful exploitation can lead to code execution under the context of the application. The particular vulnerability occurs within the window.navigator.plugins array. This array is implemented within dom/src/base/nsPluginArray.cpp. Each element of this array contains a reference to the mime types installed by that particular plugin. Upon page reload, the plugin array will reallocate all of it's members without explictly checking the used reference count of each member. If an attacker grabs a reference out of the array, and causes the page to reload itself, the **attacker will then have a variable that references data that has been freed by the page refresh.**

Solution

detach the plugin if the mimeType is not null

Codes

"Arbitrary code execution", "Object deallocation"

CVE-2010-0170

Context

does not offer plugins the expected window.location protection mechanism, which might allow remote attackers to **bypass the Same Origin Policy and conduct cross-site scripting (XSS) attacks** via vectors that are **specific to each affected plugin Sandbox**

Problem

the window.location object was made a normal overridable JavaScript object in the Firefox 3.6 browser engine (Gecko 1.9.2) because new mechanisms were developed to **enforce the same-origin policy between windows and frames**. This object is unfortunately **also used by some plugins to determine the page origin used for access restrictions**. A malicious page could

override this object to **fool a plugin into granting access to data on another site or the local file system.** The behavior of older Firefox versions has been restored. we removed some code protecting the location object (on both the document and the window) because it isn't needed anymore for either web content or extensions (web pages are allowed to confuse themselves to their heart's content). In doing this, we forgot that plugins also use location.href to figure out what page they've been embedded in.

Solution

restore the code **protecting the location object** that had been removed

Codes

"JS Objects Isolation", "Same-Origin Policy Bypass", "Improper Objects Isolation"

CVE-2009-2665

Context

when certain add-ons are enabled, **does not properly handle a Link HTTP header,** which allows remote attackers to **execute arbitrary JavaScript with chrome privileges** via a crafted web page, **related to an incorrect security wrapper. sandboxing**

Problem

broken functionality on pages that had a Link: HTTP header when an add-on was installed which implemented a Content Policy in JavaScript, such as AdBlock Plus or NoScript. Mozilla security researcher moz_bug_r_a4 demonstrated that the broken functionality was due to the window's global object receiving an **incorrect security wrapper** and that this issue could be used to **execute arbitrary JavaScript with chrome privileges**.

Solution

If we already have a wrapper at this point, it might have the wrong parent and scope, so **reparent** it.

Codes

"Privilege elevation", "Arbitrary code execution", "Object wrappers"

CVE-2009-1837

Context

Race condition in the NPObjWrapper_NewResolve function in modules/plugin/base/src/nsJSNPRuntime.cpp in xul.dll might allow remote attackers to **execute arbitrary code** via a page transition during Java applet loading, related to a **use-after-free vulnerability for memory associated with a destroyed Java object**. **deallocation**

Problem

A vulnerability was reported in Mozilla Firefox. A remote user can cause **arbitrary code to be executed on the target user's system.** A remote user can create specially crafted HTML that, when loaded by the target user, will navigate away from a web page while a Java applet is **loading to cause the applet to be deleted and then later called**, potentially writing freed memory and executing arbitrary code on the target system. The code will run with the privileges of the target user. The vulnerability resides in NPObjWrapper_NewResolve and **occurs when accessing the properties of an NPObject**. race condition in NPObjWrapper_NewResolve when accessing the properties of a NPObject, a wrapped JSObject. Balle and Eiram demonstrated that this condition could be reached by navigating away from a web page during the loading of a Java applet. Under such conditions the Java object would be destroyed but later called into resulting in a free memory read. An attacker could potentially **write to the freed memory before it is reused and run arbitrary code** on the victim's computer.

Solution

Find out what plugin (NPP) is the owner of the object we're manipulating, and **make it own any JSObject wrappers created here.**

Codes

"Arbitrary code execution", "Object deallocation", "Race condition"

CVE-2009-1310

Context

Cross-site scripting (XSS) vulnerability in the MozSearch plugin implementation allows user-assisted remote attackers to inject arbitrary web script or HTML via a javascript: URI in the SearchForm element.

Problem

malicious MozSearch plugin could be created using a javascript: URI in the SearchForm value. This URI is used as the default landing page when an empty search is performed. If an attacker could get a user to install the malicious plugin and perform an empty search, the SearchForm javascript: URI would be executed within the context of the currently open page.

Solution

ignore search form urls filter the SearchForm value the same way we already filter templateURI and the IconURL.

Codes

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

CVE-2008-5013

Context

Mozilla Firefox Flash Player Dynamic Module Unloading Vulnerability Build identifier: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.12) Gecko/2008020121 Firefox/2.0.0.12 the POC page runs a lot slower. The HTML content is shown, then the dialogs, but then firefox crashes

Problem

Tipping Point has reported a bug in the Flash plugin for Firefox which they claim contains a buffer overflow. This could potentially allow an attacker to execute arbitrary code on victim's computer. Build identifier: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.12) Gecko/2008020121 Firefox/2.0.0.12 the POC page runs a lot slower. The HTML content is shown, then the dialogs, but then firefox crashes

Solution

This is a backport of the changes we took on the trunk for bug 410946. Given that our plugin initialization code is *really* different on the branch compared to trunk the changes to nsObjectFrame.cpp are not back-portable to the branch, but the changes to the plugin code alone seems to fix this particular problem. There's probably still fragility in the plugin frame code that is *not* addressed by this patch, and the only likely fix for that on the branch would be to port all the plugin loading changes from the trunk back to the branch, and that's *really* not trivial, and I would advice against investing in that given the *huge* number of regressions (and changes in functionality) we found from that change on the trunk, years after the change went in.

Codes

"Application crash", "Arbitrary code execution"

CVE-2008-2807

Context

Faulty .properties file results in uninitialized memory being used

Problem

I have hit a weird bug writing an extension for Firefox. The extension is localized in both french and english. One of the french *.properties file was **not UTF-8 but ISO8859-encoded.** The window opened by my extension retrieves 3 strings from that properties file. Surprisingly, the first retrieval (string 'textLayouts' in the attached file) did not fail, while the two others did fail. BUT what I got back from that first lGetStringFromName()l call was absolutely **not what's in the properties file but some text coming from a MS Word process also running on my machine** !!! My XUL window was showing me a bit of the text I was editing in Word... F**ixing the encoding of the file of course resolved the issue.**

Solution

Very safe fix **ensuring that the length** returned by UTF8InputStream::Read **doesn't exceed the number of characters** successfully converted.

Codes

"Assuming encoding of incoming data is UTF-8", "Data leakage", "Incorrect parsing of manifest file"

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-2006-2784

Context

The PLUGINSPAGE functionality in Mozilla Firefox before 1.5.0.4 allows remote user-assisted attackers to **execute privileged code by tricking a user into installing missing plugins** and selecting the "Manual Install" button, then using nested javascript: URLs. NOTE: the manual install button is used for downloading software from a remote web site, so this issue **would not cross privilege boundaries if the user progresses to the point of installing malicious software** from the attacker-controlled site.

Problem

The patch from the **previous advisory can be circumvented** if the following two changes are made: 1) The embed element is shown on a javascript page 2) The executed javascript accesses chrome using it's full priviledges to the opener object This **can be exploited using a small amount of user interaction which will likely occur given the right social engineering.**

Solution

Workaround Do not press the "Manual Install" button on the Firefox plugin finder. Instead use a search engine to find an appropriate plugin for the content. **moving some code in nsScriptSecurityManager.cpp.**

Codes

"Installer", "Unsanitized parameter", "User-assisted attack"

CVE-2005-0752

Context

Plug-in install

The Plugin Finder Service (PFS) in Firefox before 1.0.3 allows remote attackers to **execute arbitrary code** via a javascript: URL in the PLUGINSPAGE attribute of an EMBED tag.

Problem

When a webpage requires a plugin that is not installed the user can click to launch the Plugin Finder Service (PFS) to find an appropriate plugin. If the service does not have an appropriate plugin the EMBED tag is checked for a PLUGINSPAGE attribute, and if one is found the PFS dialog will contain a **"manual install"** button that will load the PLUGINSPAGE url.

Omar Khan reported that if the PLUGINSPAGE attribute contains a javascript: url then pressing the button could launch arbitrary code capable of **stealing local data** or **installing malicious code**.

Element `embed` pluginspage attribute allows javascript urls, and somehow somebody forgot to **sanitize it**. So can **execute arbitrary code**.

Reproducible: Always

Steps to Reproduce:

1. Load page with an embed's pluginspage attribute set to javascript url

2. Click install missing plugins

3. Click Manual install Actual

Results: **Arbitrary code executed**

Expected Results: Do not execute arbitrary code.

Solution

Do **security check when opening URL for manual plugin installation** A cleaner way, without security manager checks would be:

```
nsPluginInstallerWizard.prototype.loadURL = function (aUrl){
      if (window.opener.getBrowser().mCurrentBrowser)
            window.opener.getBrowser().mCurrentBrowser.contentWindow.open(aUrl);
}
```

use the current browser's contentWinow to open the url. This should be safe, thought perhaps if the current browser is pointing at an chrome:// url this could cause trouble. Probably needs more testing.

Codes

"Arbitrary code execution", "Steal data", "Perform security check on unsanitized data"

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-2005-0578

Context

Unsafe /tmp/plugtmp directory exploitable to **erase user's files**

Problem

A **predictable name is used** for the plugin temporary directory. A malicious local user could symlink this to the victim's home directory and wait for the victim to run Firefox. When Firefox shuts down the **victim's directory would be erased**. Mozilla creates the /tmp/plugtmp directory for storing plugin files, on exit, it empties this directory. A malicious local user could create a symlink at /tmp/plugtmp to a directory, when another user runs firefox the directory will be emptied. This is a **serious issue on multiuser systems.** Reproducible: Always Steps to Reproduce: 1. ln -s /home/victim /tmp/plugtmp 2. wait for victim to run and exit firefox 3. victim just lost the files in his homedir. Actual Results: Example: $ pwd /home/taviso $ mkdir test $ cd test $ for ((i=0;i<10;i++)); do touch ${RANDOM}.jpg; done $ ls 10659.jpg 16835.jpg 26339.jpg 4062.jpg 8234.jpg 15120.jpg 22838.jpg 29316.jpg 724.jpg 9053.jpg # now malicious user wants to remove these files (i'll use user nobody for this example) $ sudo -u nobody ln -s /home/taviso/test /tmp/plugtmp $ ls -l /tmp/plugtmp lrwxrwxrwx 1 nobody nobody 17 Feb 6 18:43 /tmp/plugtmp -> /home/taviso/test/ # now malicious user waits until I run firefox... $ firefox $ ls $ echo "arghhh, my files!" Expected Results: perhaps use a mkdtemp()-style directory instead of hardcoded /tmp/plugtmp?

Solution

Always **create unique plugintmp directories** for each browser instance, and **only delete what was created.**

Codes

"Extracting the plug-in to world-accessible location", "Erase user's files", "Symlink attack"

CVE-2005-0232

Context

Using Flash and the -moz-opacity filter you can get access to about:config and **make the user silently change values**

Problem

Plugins (such as flash) can be used to **load privileged content into a frame**. Once loaded, various spoofs can be applied to **get the user to interact with the privileged content**. Michael Krax's "Fireflashing" example demonstrates that an attacker can open about:config in a frame, hide it with an opacity setting, and if the attacker can **get the victim to click at a particular spot** (design some kind of simple game) you could **toggle boolean preferences, some of which would make further attacks easier.** The "firescrolling" example demonstrates arbitrary code execution (in this case downloading a file) by convincing the user to scroll twice. Details: Using Flash and the -moz-opacity filter it is possible to display the about:config site in a hidden frame. By making the user double-click at a specific screen position (e.g. using a DHTML game) you can toggle the status of boolean config parameters. As long as the number of about:config parameters is unchanged (unlikely a casual user will change them) you can move the parameter you want to change to the specified screen position by using CSS (change the .hideframe class). Reproducible: Always Steps to Reproduce: 1. Open http://www.mikx.de/fireflashing/ 2. Open example link (make sure Flash 7 is installed) 3. Double click on the red box Actual Results: You can silently toggle any boolean config parameter Expected Results: Security manager should prevent that a plugin can open about:config or file:/// links

Solution

Do **security checks when loading URIs from any plugin**. This will enforce that plug-ins cannot open the about:config Webpage (that has **access to privileges configuration**) r=dveditz, but don't we need to do the same thing down in nsPluginHostImpl::PostURL ?

Codes

"Privilege elevation", "User-assisted attack", "Bypass protection mechanism"

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"