Chromium

CVE-2016-1650

Context

The flaw occurs in an API used by extensions for doing a page capture

Problem

There is a race condition when a failure occurs in the PageCaptureSaveAsMHTMLFunction extension function (pageCapture.saveAsMHTML). There are a few different ways the use-after-free can occur, but suppose we have the following background extension script: chrome.pageCapture.saveAsMHTML({"tabId": 1337}, function(results) {}); When this function’s RunAsync() is called, a single AddRef() is called to add an additional reference to the function. This is “balanced” with a Release() in ReturnFailure(), and in OnMessageReceived(). The problem is that it’s possible for both Release()s to be called causing a race condition and a uaf crash.

Solution

Remove extra Release() in pageCapture extension function implementation.

Codes

"Application crash", "Race condition"

CVE-2016-1640

Context

The Web Store inline-installer implementation in the Extensions UI in Google Chrome before 49.0.2623.75 does not block installations upon deletion of an installation frame, which makes it easier for remote attackers to trick a user into believing that an installation request originated from the user's next navigation target via a crafted web site.

Problem

An installation can only trigger an extension install that is hosted in the same origin of the site who triggered the install. This vulnerability makes it possible to display an inline extension installation dialog on a different origin that initiated the install. Thus tricking the user into installing an Extension as if it was from that origin. As the dialog doesn't show the origin, it gives even more credibility to the attack.

Solution

Don't allow inline install if frame is deleted before user accepts. If the frame that called the chrome.webstore.install method to begin an inline install gets deleted before the user accepts from the dialog, we don't want the install to continue because a navigation could make it look like the install request was coming from some unrelated site. One downside of this approach is that the dialog stays around even after the frame is deleted, and hitting either accept or cancel buttons both just cancel the install. It would be better if the dialog is automatically cancelled, but doing that would involve a lot more refactoring. The approach in this CL was easier and is probably worth getting out, and we can improve on it in the future.

Codes

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

CVE-2016-1638

Context

Restrictions can be bypassed, which **allows remote attackers to bypass intended access restrictions** via a crafted platform app.

Problem

Chrome Version: 48.0.2564.103 (stable, and earlier) and 50.0.2633.0 (HEAD) Some web platform APIs are disabled in Chrome apps for security reasons (https://developer.chrome.com/apps/app_deprecated). This is implemented in platform_app.js [1]. These **restrictions can be bypassed**: (I) The restriction of document.open/write/writeln/close is implemented by **shadowing HTMLDocument.prototype.write**, but Document.prototype.write should be shadowed instead.

So, either of the following two ways allows the use of the restricted API: delete HTMLDocument.prototype.write; Document.prototype.write.call(document); (II) window.onbeforeunload is shadowed by Object.defineProperty with configurable:true. This allows the property descriptor to be removed via the delete operator: delete window.onbeforeunload; // Remove restriction window.onbeforeunload = function() { return 'This should not be visible!'; }; PoC for I: See issue 585268 (document.write/close was used for that exploit). PoC for II: 1. Download manifest.json and background.js 2. Load the app (either via chrome://extensions, or by uploading it to the Chrome Web Store and installing it). 3. The app uses the above trick, and then calls location.reload() to show a PoC. Expected result: "window.onbeforeunload is not available in packaged apps." error in console. Actual result : Upon unload, a dialog shows up.

Solution

It's easy enough to patch these particular issues - **disable the methods on Document instead of HTMLDocument**, don't do strict comparisons for onunload and related, remove the configurable, etc. The underlying problem though is that we're trying to mangle things in JS that really should **fundamentally be guaranteed at a lower level** (probably somewhere in blink). Unfortunately, I don't know the best way to begin going about that (or if it's even necessarily something we really want to do), and don't personally have time to implement it at the moment. If anyone knows a way that this is already done in blink (or content/, or v8), lemme know - in the meantime, I'll go ahead and make these fixes to at least harden our security a little bit.

Codes

"Not enforcing private code", "Bypass protection mechanism"

CVE-2016-1635

Context

extensions::LoadWatcher::CallbackAndDie is called when DidCreateDocumentElement is triggered. CallbackAndDie calls a user-defined JavaScript function and then deletes the LoadWatcher instance. However, **JavaScript code can easily replace the document** (e.g. via document.close/write) and DidCreateDocumentElement is triggered also whenever a document is created/replaced.

Problem

Malicious Chrome apps can cause CallbackAndDie to be called **re-entrantly** when the callback of chrome.apps.window.create replaces the document of the new app window, which results in a use-after-free. This results in a **crash** of the Web browser.

Solution

Avoid re-entrant calls by using **PostTask through observer notifications** that the method has been **called once and has not finished executing.**

Codes

"Reentrancy", "Application crash", "Enforcing atomicity of event dispatching"

CVE-2016-1622

Context

The Object.defineProperty() method defines a new property directly on an object, or modifies an existing property on an object, and returns the object. It was being used as an attack vector.

Problem

Malicious parties can change the content of an extension through its code being able to be accessed through Object.defineProperty method. This allows third parties to change the behavior of the extension through crafted Javascript code.

This results in a Same-Origin Policy bypass.

Solution

Don't allow built-in extensions code to be overridden.

Codes

"JS Objects Isolation", "Same-Origin Policy Bypass", "Alter Execution Logic"

CVE-2015-6779

Context

PDFium, as used in Google Chrome before 47.0.2526.73, does not properly restrict use of chrome: URLs, which allows remote attackers to bypass intended scheme restrictions via a crafted PDF document, as demonstrated by a document with a link to a chrome://settings URL.

Problem

A hyperlink inside a pdf file shown by the chrome pdf-viewer can link to a chrome:// url and can be opened as a new tab. As this is prohibited in html (it will open about:blank), it also shouldn't be possible in a pdf-file. PDF viewer allows navigation to file:// URLs, whereas it does not for webpages.

Solution

a mechanism for more granular link URL permissions (filtering on scheme/host). This fixes the bug that allowed PDFs to have working links to any "chrome://" URLs

Codes

"Bypass protection mechanism", "Trigger access to an arbitrary URL"

CVE-2015-6772

Context

Security: Universal XSS using plugin objects Google chomium allows that elements are attached/detached at runtime using javascript. Each attached document in iframes is checked against the Same-Origin Policy

Problem

This is a regression from issue 524120. Now that widget updates are deferred until after the frame is detached from the document (and beyond the lifetime of ScriptForbiddenScope, too), it is possible to attach another document to the frame before a new document is installed. The attached document can then be used to bypass the same-origin policy. "So the root cause of this issue is that running nested message loops that invoke script in Document::detach() generally results in broken invariants. The original patch tries to change the timing of running deferred widget updates to the message loop, to avoid re-entrancy. However, that hit a lot of crashes and didn't look like something that would be easy to merge to M47. Anotherr patch:https://codereview.chromium.org/1444183003 works but it turns out that it can leave a dangling Document/FrameView. The invariant being violated here is that Frame has no FrameView at the end of Document::detach().

Solution

"Note that the change looks large, but it's really just moving FrameNavigationDisabler from NavigationScheduler into LocalFrame. The core change in the patch is just adding one line to Document::detach to disable navigations: FrameNavigationDisabler navigationDisabler(*m_frame); Which is really low risk because prior to r350972, these sorts of navigations couldn't be triggered anyway."

Codes

"Reentrancy", "Same-Origin Policy Bypass", "Use after free"

CVE-2015-1302

Context

Security: Cross-site read access to PDF files. Pdf Data leakage across cross-origin.

Problem

The out-of-process viewer exposes an API when its MIME-type is set to application/x-google-chrome-pdf, presumably to support the print preview.

Among the supported API is a method to select all text and get the contents of the selection. This allows any web page to read the contents of a PDF file from any source.

I have attached a proof of concept.

1. Open the page.
2. Input the URL of a PDF file (I've used the Bitcoin paper as an example).
3. Click on the "Show content" button.
4. The contens of the PDF will be displayed in the PDF.

This can be fully automated, websites could scan for popular URLs and automatically read the contents of a PDF. The only defence for users is to disable plugin loading by default. There are three settings, "Run all plugin content", "Detect and run important plugin content" and "Let me choose when to run plugin content". Only the last option protects users from this exploit.

Solution

insert an iframe (containing a page from ChromeVox's origin) that directly communicates with the PDF component extension, via a MessagePort. The sender and receiver have to mutually authenticate each other, this can be done by communicating a random value over another channel (e.g. extension message passing API). - Run ChromeVox in the component extension, and directly communicate between the component extension and ChromeVox.

Codes

"Same-Origin Policy Bypass", "Data leakage"

CVE-2015-1298

Context

The chrome.runtime is exposed for app developers to access some information about the underlying runtime environment, as stated in the documentation: "Use the chrome.runtime API to retrieve the background page, return details about the manifest, and listen for and respond to events in the app or extension lifecycle. You can also use this API to convert the relative path of URLs to fully-qualified URLs." Source: https://developer.chrome.com/apps/runtime#method-setUninstallURL

Problem

chrome.runtime.setUninstallURL only checks whether the given parameter is a syntactically valid URL, but it does not enforce the blacklist of disallowed URLs. This allows extensions and apps (without requiring any install permissions) to open any URL, including special chrome://-URLs. In the worst case, this bug could be used to exploit a memory bug in the browser process (e.g. UAF in a browser thread upon shutdown).

Solution

To restrict chrome.runtime.setUninstallURL to http(s). Disallow URLs other than http(s) in chrome.runtime.setUninstallURL. And allow empty URLs to be set to clear the uninstallation URL. Added an optional callback, to know when setting the URL finished (or failed).

Codes

"User-assisted attack", "Not enforcing permissions", "Trigger access to an arbitrary URL"

CVE-2015-1297

Context

The WebRequest API implementation in extensions/browser/api/web_request/web_request_api.cc in Google Chrome before 45.0.2454.85 does not properly consider a request's source before accepting the request, which allows remote attackers to bypass intended access restrictions via a crafted (1) app or (2) extension.

Problem

webRequest API allows extensions to intercept and redirect requests from the browser. That includes requests from other extensions. However, it also allows to intercept XMLHttpRequest requests from Chrome Apps, which is quite possibly unintended. Chrome Apps are supposed to be as much independent from the browser as possible.

Solution

Hide requests in an extension from other extensions

Codes

"Dispatching events to unauthorized listeners", "Extensions being able to tamper with other extensions"

CVE-2015-1226

Context

Extensions API: The DebuggerFunction::InitAgentHost function in browser/extensions/api/debugger/debugger_api.cc in Google Chrome before 41.0.2272.76 does not properly restrict what URLs are available as debugger targets, which allows remote attackers to bypass intended access restrictions via a crafted extension.

Problem

Extensions can silently debug (run code) in ANY tab and escape the sandbox. The chrome.debugger extension API can attach to targets at any origin, including URLs such as file://, chrome://, chrome-extension:// and the Chrome Web store. Attaching to privileged targets is usually forbidden when the target is specified by tabId or extensionId. However, it is also possible to attach to a target by targetId, which is not subjected to any validation. This targetId can easily be obtained using the chrome.debugger.getTargets method. Because of these capabilities, anything that can be displayed in a tab is completely compromised when a user installs an extension that exploits this bug.

Solution

Validate debuggee.targetId before use in chrome.debugger and refactored the tests to make sure that the debugger is detached upon returning from RunAttachFunction. Previously, if the debugger unexpectedly succeeded in attaching, the method would return (because empty error != some error), causing the attached debugger to not be detached. In short, they added a permission check

Codes

"Sandbox escape", "Bypass protection mechanism", "Not enforcing permissions", "Allowing extensions to debug tabs"

CVE-2014-3172

Context

The Debugger extension API in browser/extensions/api/debugger/debugger_api.cc in Google Chrome before 37.0.2062.94 does not validate a tab's URL before an attach operation, which allows remote attackers to bypass intended access limitations via an extension that uses a restricted URL, as demonstrated by a chrome:// URL.

Problem

Any extension can debug any other extension, and be able to maliciously use the data of users. By using the "downloads" permission in conjunction with network_diag, network_logging, wpa_debug, and ff_debug it should be possible to snoop on a lot of private user data.

Solution

Have the Debugger extension api check that it has access to the tab Check PermissionsData::CanAccessTab() prior to attaching the debugger.

Codes

"Bypass protection mechanism", "Lack of security checks", "Not enforcing permissions", "Extensions being able to tamper with other extensions"

CVE-2014-3170

Context

Extensions/common/url_pattern.cc in Google Chrome before 37.0.2062.94 does not prevent use of a '\0' character in a host name, which allows remote attackers to spoof the extension permission dialog by relying on truncation after this character.

Problem

By inserting a NUL byte in a host permission, extension authors can hide all host permission requests, giving users a false sense of security when they install an extension.

Solution

Do not allow NUL characters in the hosts of host permissions.

Codes

"Privilege elevation", "Bypass protection mechanism", "Incorrect parsing of manifest file"

CVE-2014-1728

Context

out of bounds writes A Pwnium 4 entry achieved a sandbox escape by sending messages from a compromised swapped out renderer to a vulnerable extension. The problem seems to be in TabHelper, which is getting state confused by the swap out. creis' synopsis: TabHelper is not deleting state associated with a swapped out RVH. It should probably be listening to WebContentsObserver::RenderViewHostChanged in addition to RenderViewHostCreated.

Problem

The exploit is possible because the attacker's renderer process can send a message from a swapped out RenderFrameHost which makes it up into TabHelper. TabHelper isn't keeping track of who sent the message and assumes it's from WebContents::GetRenderViewHost(), which refers to the current RenderViewHost (legitimately for the extension process) and not the swapped out one (in the attacker's process). Multiple things are wrong here: 1) A message from a swapped out RFH is making it up to TabHelper. Swapped out hosts shouldn't be propagating their IPC messages up to observers. We actually have a check for that in RenderViewHostImpl::OnMessageReceived, where we filter out such IPC messages using CanHandleWhileSwappedOut. We're missing that check in RenderFrameHostImpl, which is how this snuck through. Nasko will fix that. 2) TabHelper doesn't know who sent it the message, since that information is not available in OnMessageReceived. In theory, it would be nice for it not to have to worry about that, since it's easy to miss an access control check. Fixing (1) means we don't have to worry about messages from swapped out RFHs, for example. Unfortunately, the problem still exists with current vs pending RFHs. More concretely, a WebContents can have both a current and a pending RFH at the same time (e.g., one in an attacker's process and one in an extension process). Until the pending RFH actually commits, IPC messages could come up to WebContentsObservers from either one, and the observers just assume that they're hearing from the current RFH, not the pending one. That could lead to the same kind of attack.

Solution

More concretely, a WebContents can have both a current and a pending RFH at the same time (e.g., one in an attacker's process and one in an extension process). Until the pending RFH actually commits, IPC messages could come up to WebContentsObservers from either one, and the observers just assume that they're hearing from the current RFH, not the pending one. That could lead to the same kind of attack. I'm not sure if it's possible to hide messages from the pending RFH until it commits, since it may need to initialize state (e.g., RenderFrameCreated) before the commit happens. Queuing the messages up inside WebContents until commit might be possible, but that feels really error prone if it delays acks or other message exchanges. Another option is to expose who sent the IPC message so that the observer can do an access control check. Exposing "who sent it" could be in the form of RenderFrameHost, routing ID + process ID, SiteInstance, or some kind of security principal. Longer term efforts like Mojo might help with that. In the shorter term, it's less clear how to fix that without exposing concepts like pending RFHs to observers.

Codes

"Sandbox escape", "Added origin check", "IPC has no info about origin of a message"

CVE-2013-2912

Context

Heap-use-after-free in ppapi::proxy::PluginResource::NotifyInstanceWasDeleted. A refactoring was suggested but reverted because it broke Docs print preview. A little more printf debugging reveals that the PluginResource destructor sends the destruct message to the host but never exits. It looks like a hang, but there shouldn't be multiple threads for in-process plugins.

Problem

A unique situation for in-process plugins. The repro.html file causes a load to start, then a reload, then moves the plugin element when the ready state changes. The instance is torn down while we're in one of the URLLoaderResource dtors, before it has removed itself from the tracker. The resource tracker tries to use the object which is half destructed. The repro.html forces it into a state it doesn't like, hitting a NOTREACHED which needs to be changed.

Solution

Change the PepperInProcessRouter to defer resource destruction messages. This changes the in process "proxy" so it posts tasks to send resource destruction messages instead of calling them directly. This prevents several kinds of re-entrancy into the plugin-side code. In this case, when a URLLoader is released, the plugin can finish before the host cancels the load and potentially deletes the instance.

Codes

"Reentrancy", "Application crash"

CVE-2013-2876

Context

Extensions UI- browser/extensions/api/tabs/tabs_api.cc in Google Chrome before 28.0.1500.71 does not properly enforce restrictions on the capture of screenshots by extensions, which allows remote attackers to obtain sensitive information about the content of a previous page via vectors involving an interstitial page.

Problem

Extensions are allowed to screenshot interstitial websites content without having permission to do so, hence being able to steal user's information.

Solution

Add check for permission screenshot even in interstitial websites or not allow it by default.

Codes

"Data leakage", "Bypass protection mechanism", "Lack of security checks"

CVE-2013-2868

Context

common/extensions/sync_helper.cc in Google Chrome before 28.0.1500.71 proceeds with sync operations for NPAPI extensions without checking for a certain plugin permission setting, which might allow remote attackers to trigger unwanted extension changes via unspecified vectors.

Problem

Chrome Sync is used to install an extension with NPAPI plugin and execute code. Though extensions with plugins cannot be installed through sync directly, they can be auto-updated, and the new version may contain plugins. chrome/common/extensions/sync_helper.cc checks PluginsInfo::HasPlugins to determine if an extension can be synced. HasPlugins returns false if there's an empty plugins section. chrome/common/extensions/api/plugins/plugins_handler.cc parses "plugins" from the manifest. It treats an empty "plugins" section differently from it not being there, though. It also adds the "plugins" permission if there are any plugins; it should be treated as a permissions increase (disabling the extension) if there are new plugins in the new version of the extension.

Solution

The fix adds a check for Ipluginl permission while syncing NPAPI plugins.

Codes

"Privilege elevation", "Bypass protection mechanism", "Plugin update"

CVE-2013-2841

Context

Use-after-free vulnerability allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors related to the handling of Pepper resources. generic WebKit DOM vs. PPAPI URL loading lifetime issue.

Problem

1. PPB_URLLoader_Impl(ppb_url_loader_impl.h) is a subclass of ppapi::Resource(ppapi/shared_impl/resource.h). 2. But for some strange reason destructor of ppapi::Resource is not executed when destructor of PPB_URLLoader_Impl is executed. 3. Think that is why this bug happens. Because destructor of ppapi::Resource should be executed to remove PPB_URLLoader_Impl instance from ResourceMap live_resources_ of ResourceTracker(ppapi/shared_impl/resource_tracker.cc). 4. Otherwise PPB_URLLoader_Impl isntance will remain in ResourceMap live_resources_ of ResourceTracker even after being deleted.

Solution

Remove Pepper URLLoader from resource tracker early. This protects against double delete if the instance is destroyed as a result of canceling a load.

Codes

"Application crash"

CVE-2013-0925

Context

Google Chrome before 26.0.1410.43 does not ensure that an extension has the tabs (aka APIPermission::kTab) permission before providing a URL to this extension, which has unspecified impact and remote attack vectors.

Problem

The chrome.tabs.onUpdated event is accessible to Chrome extensions even without the "tabs" permission, and leaks the URLs the user navigates to in the changeInfo.url argument to the event callback.

Solution

Do not pass URLs in onUpdated events to extensions unless they have the "tabs" permission.

Codes

"Steal data", "Dispatching events to unauthorized listeners"

CVE-2013-0924

Context

When you install an extension, there are warnings about host permissions. However, by design there aren't any for file: permissions - these are handled via a checkbox on the extension settings page. The same goes for the permissions API.

Problem

The API implementation doesn't respect the checkbox value on the extensions settings page, so silently allows extensions to obtain file level permissions.

Solution

The API implementation should check for the checkbox values by users in the extension settings page that regulate file: permissions.

Codes

"Installer", "Incorrect warning of permissions", "Silently allows extensions to obtain file level permissions"

CVE-2013-0910

Context

Google Chrome before 25.0.1364.152 does not properly manage the interaction between the browser process and renderer processes during authorization of the loading of a plug-in, which makes it easier for remote attackers to bypass intended access restrictions via vectors involving a blocked plug-in.

Problem

A compromised renderer can load banned plug. The renderer has the capability to block or unblock plug-in it, but having the decision of block/unblocking a plug-in in the renderer is weak: renderer interacts with content from the Web (untrusted) and may be compromised.

Therefore, a stronger decision is to have the checks of blocks/unblocked plug-ins in the browser-side.

- Allegedly, Java is installed on 66% of computers (largely independent of browser).
- A Java installation is frequently out of date; we block this situation.
- Even when uptodate, Java is a security nightmare -- it's currently the largest source of severe 0-day attacks in the browser ecosystem. Because of this, we block even an uptodate Java.

So, interestingly, now think about a compromised renderer. A compromised renderer gets to load any plug-in it pleases. This is largely because the decision to block a plug-in or not lives in the renderer -- and this, in turn, is necessitated by the click-to-play.

However, in the event that the browser determines that the status of a plug-in is "blocked" for whetever reason, we can refuse to load the plug-in at the browser side. This is only slightly complicated by the need to handle browser-mediated user authorizations (infobars, right-click menu and page action icon).

So we can become secure against compromised renderers. For example, a compromised renderer now cannot load the Java plug-in by default, unless the user has authorized a site to use Java and the attacker knows what that site is. A majority of users have Java installed yet never use Java. So we can protect those users.

Solution

Only permit plug-in loads in the browser if the plug-in isn't blocked or the user has authorized it with a browser-mediated interaction.For example, a compromised renderer now cannot load the Java plug-in by default, unless the user has authorized a site to use Java and the attacker knows what that site is. A majority of users have Java installed yet never use Java. So we can protect those users.

Codes

"Execution of user-blocked plug-in"

CVE-2013-0896

Context

Plug-in execution. BrowserPluginGuest blindly trusts the size of shared memory regions leading to overflow.

Problem

BrowserPluginGuest trusts the shared memory region sizes passed in messages from renderers. When the browser attaches to these regions it does not sanity check the region sizes and can be made to write beyond the end of the mapped region.

Solution

Browser Plugin: Simplified BrowserPlugin Damage Buffer

1. Less platform-specific code.

2. Use base::SharedMemory instead of TransportDIB.

3. Use scoped_ptr to simplify cleanup logic.

4. More validity checks

Codes

"Perform security check on unsanitized data", "Memory corruption"

CVE-2013-0831

Context

Extensions > Loading Resources (Directory Path Traversal)

Directory traversal vulnerability in Google Chrome before 24.0.1312.52 allows remote attackers to have an unspecified impact by leveraging access to an extension process.

Problem

I "found" this via code inspection while looking at an unrelated bug, checking for places where folks might have existing code I could borrow to prevent directory traversal escapes. Consider extension_resource.cc:66: for (std::vector::const_iterator i = components.begin(); i != components.end(); i++) { if (*i == FilePath::kParentDirectory) { depth--; } else { depth++; } if (depth < 0) { return FilePath(); } } This logic fails to account for "/." in path names, e.g. given something like ./../../.. we will be up two levels but will compute depth == 0.

Solution

Added a check to enforce that it does not escape the directory

Codes

"Installer", "Data leakage", "Improper validation of manifest files"

CVE-2012-5126

Context

Use-after-free vulnerability in Google Chrome before 23.0.1271.64 allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors related to the handling of plug-in placeholders.

Problem

plug-in is being removed from the DOM while we're initializing it

Solution

Set the new plug-in on the container before initializing it. Once the plug in is removed , destroy the old plug in

Codes

"Object deallocation", "Use after free"

CVE-2012-5125

Context

Extensions > Deallocator

Use-after-free vulnerability in Google Chrome before 23.0.1271.64 allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors related to the handling of extension tabs.

Problem

A Use-After-Free that crashes the browser after the extensions is closed. 0. Get a Chrome instrumented with ASan (the one at goto/chrome-asan is a bit stale, but fine. The bug is also reproducible with the ToT Chrome) 1. Install the "Screen Capture by Google" extension (ID: cpngackimfmofbokmjmljamhdncknpmg) 2. Open any webpage, click on the Screen Capture icon and select "Capture Whole Page" 3. In the Screen Capture window, click the "Close" button.

Solution

Do not access |this| after this point. Run() ended up closing the tab that owns us. Check that the object is still available before calling it.

Codes

"Application crash", "Object deallocation"

CVE-2012-2881

Context

It is possible to cause webkit to fire a readystatechange event when pdf object is removed by removing the pdf object on DOMContentLoaded event. Then it is possible to append the removed pdf object back to document which causes a use after free later. Steps ===== 1. Download and host test3.html on local web server. 2. Open test3.html on chrome. 3. Page will display an alert box. Press escape to dismiss alert box or click ok button of alert box. 4. Page will display an alert box again. Press escape to dismiss alert box or click ok button of alert box. 5. Page will display an alert box again for third time. Press escape to dismiss alert box or click ok button of alert box. Chrome will display sad tab due to heap use after free.

Problem

It is possible to cause webkit to fire a readystatechange event when pdf object is removed by removing the pdf object on DOMContentLoaded event. Then it is possible to append the removed pdf object back to document which causes a use after free later.
Steps
=====
1. Download and host test3.html on local web server.
2. Open test3.html on chrome.
3. Page will display an alert box.
   Press escape to dismiss alert box or click ok button of alert box.
4. Page will display an alert box again.
   Press escape to dismiss alert box or click ok button of alert box.
5. Page will display an alert box again for third time.
   Press escape to dismiss alert box or click ok button of alert box.
   Chrome will display sad tab due to heap use after free.

Analysis of this issue

======================

1. Web page has embed tag which embeds a pdf file.

2. This embed element is removed on DOMContentLoaded event of document.

3. This causes a readystatechange event to fire prematurely.

4. Then on readystatechange event removed embed element is attached to the document again. This is the cause of use after free.

Solution

ASSERT(!eventDispatchForbidden()) firest when removed plugin re-inserted as part of readyStateChange. Removing a plugin causes a detach which can cancel the last remaining load on a page, resulting in a readyStateChange event during a time when things are inconsisent. Defer the detach which triggers this chain of events until after the node is fully removed from the document's elementsById map.

Codes

"Application crash", "Use after free", "Early deallocation of plug-in objects"

CVE-2012-2880

Context

race condition with windowless plugin buffers

Problem

WebPluginProxy::CreatDIBAndCanvasFromHandle() calls scoped_ptr::reset(). This is deleting a SkCanvas subclass which apparently has a pending paint. Comments in CreateDIB...() suggest this could happen in a chain of multiple resizes. There's a Mac-only special case in Paint() using weak pointers to handle contexts changing during painting. If non-Mac code could change the canvas being used during delegate_->Paint(), we'd be restoring stage on the wrong canvas.

Solution

Fix race condition with windowless plugin buffers. The problem, which is already fixed for Mac, is that the buffers can be deleted during a paint because of a resize during an NPN_Evaluate call. So keep a local reference.

Codes

"Application crash", "Race condition"

CVE-2012-2878

Context

Heap-use-after-free in WebKit::WebElement::document

m_pluginWidget member of FrameLoaderClientImpl is keeping the widget alive past the destruction of the document. This keeps the instance registered so that a delayed GetWindowObject is still processed, rather than just erroring upon enter.

Problem

The plugininstance has a raw ptr to a webplugincontentsimpl which has a raw ptr to an HTMLPluginElement from the document which no longer exists. Now, the m_pluginWidget is just about to be cleared by FrameLoaderClientImpl::redirectDataToPlugin (called from PluginDocumentParser::appendBytes), but before that can happen, layout causes the plugin to be created, which processes a sync message, which can catch the delayed GetWindowObject in its nested message loop.

Solution

Check if pluginWidget object is alive

Codes

"Application crash", "Check object is not null", "Use after free"

CVE-2012-2877

Context

Extensions > Sandbox > Process Isolation > Shutdown Chrome extensions bug cause crash in all Chrome processes

Problem

Chrome extensions bug cause crash in all Chrome processes This crash its most likely a security-related issue, as it is most likely caused by jumping to an illegal address (SEGFAULT). You can see in the example (attached) chrome crash after: xhr.open("GET", "http://api.duckduckgo.com/?q=" + search_value + "&format=json", true); WinDbg crashes because the active_dialog of AppModalDialogQueue is non-NULL but garbage. There is a javascript alert open when the popup is dismissed, and the alert dialog is deleted just before the popup is deleted. The root cause is basically in the order the objects destructed: "What's happening is that ExtensionPopup is closing itself directly when it loses focus. This seems to close the alert dialog as well (maybe because it is in the view hierarchy?). The AppModalDialog is deleted as the widget's delegate. The ExtensionHost still isn't deleted in all this time, and won't be deleted until the ExtensionPopup is deleted. The dialog is cancelled from within ~ExtensionHost, and that's where the boom happens."

Solution

Change the order of the objects destruction, and checks that it is not trying to destroy an already destroyed object

Codes

"Application crash", "Check object is not null", "Process Isolation", "Changing the order of deallocation of objects"

CVE-2012-2816

Context

Windows does not properly isolate sandboxed processes, which might allow remote attackers to cause a denial of service (process interference) via unspecified vectors.

Problem

By default, sandboxed processes can open other sandboxed processes and manipulate them. Integrity levels and the restricted group prevent reaching into unsandboxed processes. However, it's possible to start a renderer with privileged IPCs, open the process, and manipulate it directly. You duplicate the renderer's own process handle with DUPLICATE_SAME_ACCESS. Then you can do whatever you want to the process (read/write memory, CreateRemoteThread, etc.).

Solution

This is a trick to keep the GPU out of low-integrity processes. It starts at low-integrity for UIPI to work, then drops below low-integrity after warm-up.

Codes

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

CVE-2011-3956

Context

Extensions > Privileges Enforcement > Sandbox > Origin In the implementation of extensions privileges

The extension implementation in Google Chrome before 17.0.963.46 does not properly handle sandboxed origins, which might allow remote attackers to bypass the Same Origin Policy via a crafted extension.

Problem

The iframe sandbox requires that the framed content run under the privileges of a unique origin and not the privileges of the document you downloaded from. But this doesn't seem to be true. Create an extension with a tabs permission, and 2 pages. The first say popup.html runs with full privileges and frames test.html in a sandbox. test.html has code to create a new tab, which should fail since test.html is running under a unique origin. But it doesn't. This is a same-origin bypass.

Solution

Consider the origin when computing extension permissions This patch teaches the extension system to use the document's origin when computing extension permissions. Ideally, we'd use only the document's origin, but because app extents don't cover entire origins, we need to also consider the document's URL.

Codes

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

CVE-2011-3107

Context

Chrome crashing trying to call hasMethod at http://trac.webkit.org/browser/trunk/Source/WebCore/bindings/v8/V8NPObject.cpp?rev=113111#L208

Google Chrome before 19.0.1084.52 does not properly implement JavaScript bindings for plug-ins, which allows remote attackers to cause a denial of service (application crash) or possibly have unspecified other impact via unknown vectors.

Problem

The NPObject **passed the alive check and has a valid (or null) hasProperty function pointer**. However, the **hasMethod function pointer is non-null and garbage.**

Solution

Added a check for NPN_IsAlive in branches/chromium/1132/Source/WebCore/bindings/v8/NPV8Object.cpp and branches/chromium/1132/Source/WebCore/bindings/v8/V8NPObject.cpp

Codes

"JS Objects Isolation", "Application crash", "Check object is not null"

CVE-2011-3080

Context

Race condition in the Inter-process Communication (IPC) implementation in Google Chrome before 18.0.1025.168 **allows attackers to bypass intended sandbox restrictions** via unspecified vectors.

Problem

Sandbox IPC length checking race The bug can be exploited to **allow for memory read and write inside the broker process** and as such **can be exploited to gain code execution**, inside the broker process, leading to a **complete compromise of broker process and the users machine.**

Solution

Fix race in CrossCallParamsEx::CreateFromBuffer

Codes

"IPC Service", "Sandbox escape", "Same-Origin Policy Bypass", "Data leakage"

CVE-2011-3055

Context

The browser native UI in Google Chrome before 17.0.963.83 **does not require user confirmation before an unpacked extension installation**, which allows **user-assisted remote attackers** to have an unspecified impact via a crafted extension.

Problem

"The invariant that must never be breached is: all extension installs, no matter how initiated, and whether they contain NPAPI or not, must be mediated by a browser dialog."

No permission prompt when loading unpacked extension with NPAPI plugin.

As part of the recent PinkiePie Pwnium exploit, it appears that the attacker was able to gain access to the extensions management page and get it to **load an unpacked extension with an NPAPI plugin** (see also bug 117715) **without generating a prompt**. Looking at the code in UnpackedInstaller::OnLoaded, it looks like **it should generate a prompt in all cases unless the extension is disabled**. It's unclear to me from reading the code if re-enabling the extension would still trigger the prompt, but my guess is that it doesn't. I'm not sure if this is what you were getting at, but I found what appear to be some holes. Say we have version 1 of the extension that has no plugin, and version 2 has a plugin (modifying the manifest in the same location).

This triggers a permission warning:

1. Load unpacked (version 1) from directory

2. Edit the manifest to be version 2 and include plugin section
3. Load unpacked (version 2) from directory

This doesn't trigger a permission warning:

1. Load unpacked (version 1) from directory
2. Disable
3. Edit the manifest to be version 2 and include plugin section
4. Load unpacked (version 2) from directory (it's still disabled)
5. Re-enable

Strangely, this also doesn't trigger a permission warning:

1. Load unpacked (version 1) from directory
2. Edit the manifest to be version 2 and include plugin section
3. 'Reload' the extension from chrome://extensions

Solution

To show the prompt message in such scenario (https://src.chromium.org/viewvc/chrome?revision=119135&view=revision) "Prevent unnecessary prompts when unpacked extensions use chrome.permissions.request. We now record what permissions have been granted to unpacked extensions to make developing against the permissions API simpler. With this change, chrome.permissions.request will **generate the same prompts for packed and unpacked extensions**. This also fixes an issue where we were not prompting for unpacked extensions with plugins at installation time."

Codes

"Not showing install warning dialog", "Silent install of plug-ins", "Consistent generation of install warning prompts"

CVE-2011-3049

Context

Extensions > Permissions > Malicious Extensions > Blacklist file [SIDE NOTE] There is also a side issue: "Also, I just noticed ttat the **blacklist is downloaded over HTTP**. That would **allow a man-in-the-middle to arbitrarily blacklist extensions**. So, I'll file a separate bug on that one." [...] "the blacklist manifest is fetched over https and includes a sha256 hash - when **we fetch the blacklist content over http we verify that the content's hash matches that**. See ExtensionUpdater::ProcessBlacklist. "

Problem

webRequest.onBeforeRequest can intercept calls to http://www.gstatic.com/chrome/extensions/blacklist/l_0_0_0_7.txt. Thus if an extension went rogue, Google **could add the extension to the blacklist but the extension could prevent Chrome from receiving the blacklist update**. To exploit the vulnerability, an extension has to put the following code in its background script: chrome.webRequest.onBeforeRequest.addListener(function(details) { var block = (details.url.indexOf("blacklist") !== -1); console.log(details.url, block); return { cancel: block }; }, {urls: ["http://*/*"]}, ["blocking"]); After doing so, you should see http://www.gstatic.com/chrome/extensions/blacklist/l_0_0_0_7.txt true appear in the background console logs (which means the blacklist URL was successfully blocked). Basically, **the listener above executes BEFORE any Web request is performed.** Once the URL is the **blacklist it blocks the request, which prevents the update.**

Solution

Hide downloads of extensions blacklist from web request API. This CL **prevents that extensions using the web request API** can prevent Chrome from updating its extensions blacklist. To do so, they added a method in the WebRequestAPI class to perform such checks:

```
139 // Returns true if the URL is sensitive and requests to this URL must not be 140 //
modified/canceled by extensions, e.g. because it is targeted to the webstore 141 // to check
```

```
for updates, extension blacklisting, etc. 142 bool IsSensitiveURL(const GURL& url) { 143
bool is_webstore_gallery_url = 144 StartsWithASCII(url.spec(),
extension_urls::kGalleryBrowsePrefix, true); 145 bool is_google_com_chrome_url = 146
EndsWith(url.host(), "google.com", true) && 147 StartsWithASCII(url.path(), "/chrome",
true); 148 std::string url_without_query = 149 url.spec().substr(0,
url.spec().find_first_of('?')); 150 return is_webstore_gallery_url ||
is_google_com_chrome_url || 151 extension_urls::IsWebstoreUpdateUrl(GURL(url_without_query))
|| 152 extension_urls::IsBlacklistUpdateUrl(url); 153 }
```

Codes

"Blacklists", "Dispatching events to unauthorized listeners", "Prevent blacklists from being updated"

CVE-2011-3047

Context

The GPU process in Google Chrome before 17.0.963.79 allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) by leveraging an error in the plug-in loading mechanism.

Problem

The plugin blocking logic wasn't being run for NaCl in prerendering.

Solution

Fixed by moving plugin loading in prerendering after the NaCl checks.

Codes

"Memory corruption", "Loading plug-ins"

CVE-2011-2853

Context

Use-after-free vulnerability in Google Chrome before 14.0.835.163 allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors related to plug-in handling.

Problem

At first glance it looks like no np plugin object is created or it gets deleted because of the empty swf file. Then the window is closed, and chrome tells the np plugin object to release itself (or releases the object, not sure yet).

Either way, there is no such object, only garbage, leading to random memory access in a method call. I'm not sure if it's flash specific or if any np plugin can have this problem. 1. WebKit can have torn down the window script object before getting around to tearing down the plugin, causing any attempt to release the window script object in NPP_Destroy to reference freed memory. 2. Chromium has a special-case in WebPluginDelegateProxy::PluginDestroyed(), which avoids trying to release the window script object during teardown, by marking it invalid; the comment states that that is done after NPP_Destroy so that NPP_Destroy can script the window, which is clearly nonsense if WebKit has already torn it down. The first crash of the three above arises when, while we are blocked waiting for NPP_Destroy to complete in the plugin process, the plugin sends us an IPC to release the window script object. The third crash of the three above could be referred to ajwong@, if it is easily reproducible (I haven't observed it myself). The underlying issue is that it's possible for references to a plugin element to cause it to (briefly) out-live its containing page, so that if it scripts the window object during deletion, it may trample freed memory in the renderer.

Solution

Cope gracefully with plugin being destroyed during NPObject Invoke or Evaluate. Cause the stub to ignore any further IPC messages, and to tear itself down the next time control returns to the message loop.The NPObject will be released only if |release_npobject| is true. This is used for the window script object stub in the renderer, which is freed with NPN_DeallocateObject to avoid leaks, and so we must not try to release it. void DeleteSoon(bool release_npobject);

Codes

"Application crash", "Object deallocation", "Data leakage"

CVE-2011-2789

Context

Use after free in Pepper plug-in instantiation

Problem

It's the stale instance hanging off the resource in ppapi

Solution

Maintain a map of all resources in the resource tracker and clear instance back pointers when needed,

Codes

"Installer", "Application crash", "Object deallocation"

CVE-2011-2785

Context

Extensions > Installer > Parsing manifest files

The extensions implementation in Google Chrome before 13.0.782.107 does not properly validate the URL for the home page, which allows remote attackers to have an unspecified impact via a crafted extension.

Problem

Extensions manifest can have a "javascript:" url in the "homepage_url" field. When a user opens the extensions page and clicks on the title of the extension, the homepage for that extension is opened. In this case, the JavaScript is executed in the context of the extensions page. This allows the extension to install another extension from the local file system. The PoC provided by kuzzcc tries to install a second extension that's packaged inside the first. The second extension can have any extension permissions it wants.

Solution

While parsing the manifest files, the solution enforces that extensions do not define homepages with schemes other than valid web extents.

Codes

"Installer", "Arbitrary code execution", "Bypass protection mechanism", "Code Injection"

CVE-2011-2783

Context

Extensions > UI > prompt user during install. Chrome does not prompt when you use the interface on chrome://extensions to load an unpacked extension that contains an NPAPI plugin.

Problem

Missing browser prompt when installing unpacked NPAPI extensions.
Chrome does not prompt when you use the interface on chrome://extensions to load an unpacked extension that contains an NPAPI plugin. We should add the browser prompt to this case as a defense in depth measure against things like http://crbug.com/83096.

It seems like the fear here is that someone will XSS chrome://extensions and be able to cause an arbitrary extension to be installed. This is only possible because (it appears) that javascript on chrome://extensions causes a file picker to be displayed then passes the result to C++, who trusts it. The solution is to have C++ show the file picker and never pass the path through JavaScript, not to have this weird dialog that only addresses one case

Solution

Show the install dialog for the initial load of an unpacked extension with plugins. For that, created a new callback method at *chrome/browser/extensions/extension_service.h*:

```
// Called by the backend when an unpacked extension has been loaded.
void OnLoadSingleExtension(const Extension* extension);
```

Then, added a new class (`SimpleExtensionLoadPrompt`) to implement this feature (in chrome/browser/extensions/extension_service.cc).

Codes

"Not showing install warning dialog", "Silent install of plug-ins"

CVE-2011-2358

Context

[DESIGN-LEVEL] Extensions > UI > prompt user.

Google Chrome before 13.0.782.107 does not ensure that extension installations are confirmed by a browser dialog, which makes it easier for remote attackers to modify the product's functionality via a Trojan horse extension.

Problem

Android had a bad security bug where you XSS on their gallery led to install on local machine. We can have the same issue on our store because we have no client UI in that case. We do mitigate this issue today by forcing a user gesture. And we also force the client UI in the case of NPAPI.

There is no client-UI-decision for the web store. The decision was made to have an integrated purchase flow, where you only do one confirmation for both money and security, not two separate dialogs.

This means that an XSS results in an install on local machine.

Solution

Add a webstore install method that lets us prompt the user before downloading. A while back we decided to minimize friction by showing extension/app permissions inline in the webstore, and let installs done via the private webstore API skip the regular extension installation confirmation that happens after downloading and unpacking the .crx file. We've reconsidered this and are now adding a new private install method that lets us go back to having the client display the confirmation dialog, but do it before downloading the .crx file. The webstore just needs to pass the manifest and icon, and then after downloading the .crx we make sure the unpacked extension's manifest matches what we had prompted with.

Codes

"Not showing install warning dialog", "Silent install of plug-ins"

CVE-2011-1819

Context

Malicious extensions modifying protected chrome: URLs

Problem

A packaged app/extension can access and modify all chrome: pages, read/write preferences, run chrome.send function, pass arguments directly to c++, without required permissions, without using NPAPI plugin, content script or chrome.tabs.executeScript

Solution

Added an if condition for checking that non-component extensions can only access chrome://favicon and no other // chrome:// scheme urls. if (url.SchemeIs(chrome::kChromeUIScheme) && url.host() != chrome::kChromeUIFaviconHost && location() != Extension::COMPONENT) return false;

Codes

"Privilege elevation", "Data leakage", "Code Injection", "Not enforcing permissions"

CVE-2011-1815

Context

How Chromium protect itself from malicious extension. It happens on the context of manifest files parsing.

Google Chrome before 12.0.742.91 allows remote attackers to inject script into a tab page via vectors related to extensions.

Problem

Bypass extensions permission: The "web_url" attribute on a manifest field should not allow javascript: or chrome: URLs. For example, the two manifest files shown below can be used to bypass extensions permissions: manifest1.json ======== { "name": "test", "description": "test", "version": "1", "app": { "launch": { "web_url": "javascript:alert('document.domain')" } } } manifest2.json ======= { "name": "test", "description": "test", "version": "1", "app": { "launch": { "web_url": "chrome://history/" } } } In short, extensions can inject the following code through the "web_url" parameter: - javascript:alert(document.domain) //chrome://newtab - chrome://appcache-internals/ xss Preconditions: 1. Need to install an extension. No popups will be shown, since the manifest have nothing except web_url. 2. Open a new tab and click on the app icon. executes in the context of chrome URLs.

Solution

Make sure that extensions can launch web urls with web safe schemes only. Developers added an if condition to enforce this.

Codes

"Installer", "Arbitrary code execution", "Code Injection"

CVE-2011-1813

Context

There is a **stale frame** in UserScriptSlave::InjectScripts.

Problem

The problem was that the UserScriptIdleScheduler was **relying on the FrameDetached notification** from its _original_ RenderView, to **know when it should delete itself since the frame_ object was gone**. But when the frame gets reparented, the FrameDetached only gets sent to observers of the _new_ RenderView.

Solution

The fix is to have ExtensionHelper, which is per-RenderView, **proxy all these calls to ExtensionHelper, which is per-renderer.** ExtensionHelper then keeps the map of WebFrame->UserScriptIdleSchedulers, and notifies them of these events.

Codes

"Application crash", "Object deallocation", "Stale pointer", "Notify deallocation events"

CVE-2011-1450

Context

Google Chrome before 11.0.696.57 does **not properly present file dialogs**, which **allows remote attackers to cause a denial of service** or possibly have unspecified other impact via unknown vectors that lead to "dangling pointers."

Problem

When an object is destroyed, its select **file dialog is not informed to clear its listener** which can call back that destroyed object causing a potential for attacks.

Solution

Before an object destruction, make sure that its **select dialogs are told that the object is gone** so that they **don't try to call it back.**

Codes

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

CVE-2011-1435

Context

Extensions leverage the chrome.tabs.captureVisibleTab to **capture images of any using a URL like "file://"**

Problem

The tabs permission for extensions **allows an extension to capture an image** of any text file, directory, or image from the user's computer via the captureVisibleTab method. **Extensions should not be allowed to open web unsafe urls in tabs.** Proof of Concept: 1. Download the attached extension. 2. Update the "file" variable in the popup.html file to point to a local text file on your computer (pick something fun, like your private key file). 3. Install the extension. 4. Open the extension's popup (sorry no icon). This all boils down to using the following: chrome.tabs.create({"url" : "file:///home//.ssh/id_rsa"}); chrome.tabs.captureVisibleTab(null, function(d) { // Do something EVIL with 'd'. });

Solution

Implement **new restrictions** on tab.captureVisibleTab() method. It **checks that the tab does have "host" permissions to access files in the user's machines**: captureVisibleTab() can access some of the same information as JavaScript running on the page. Ensure the extension has host permissions. if (!GetExtension()->CanExecuteScriptOnPage(tab_contents->GetURL(), NULL, &error_)) { return false; }

Codes

"Data leakage", "Bypass protection mechanism", "Not enforcing permissions", "Read user's files"

CVE-2011-1124

Context

Use-after-free vulnerability in Google Chrome before 9.0.597.107 allows remote attackers to cause a denial of service or possibly have unspecified other impact via vectors related to blocked plug-ins.

Problem

1. Install out-dated ice-tea java plugin.
2. Open attached crash.html.
   crash.html contains a applet.
   Chrome will show a info bar saying ice-tea java plugin is out-dated.
3. Move the mouse over java applet (Applet is not loaded at this moment, since plugin is outdated).
4. Wait about 3 seconds. crash.html will refresh itself.
   Once the page is refreshed chrome will display a sad tab.

Solution

Restore old title in WebViewPlugin only when loading the plugin.

Codes

"Application crash", "Use after free"

CVE-2011-1123

Context

It occurs when the extension has a NPAPI plugin and the extension has not specified whether the NPAPI binary is public (i.e., it omitted the "public" attribute in its manifest file).

Problem

Take an extension with a non-public: NPAPI plugin: https://chrome.google.com/extensions/detail/caehdcpeofiiigpdhbabniblemipncjj?hl=en "plugins": [{ "path": "plugins/npSwitchy.dll" }, { "path": "plugins/npSwitchy.so" }, { "path": "plugins/npSwitchy64.so" }, { "path": "plugins/iSwitchy.bundle" }] Then on any public web page, you can instantiate that plugin. var o = document.createElement("OBJECT"); o.type = "application/x-mhdhejazi-switchy-1.6"; document.body.appendChild(o); o.doSomething(); // replace with something dangerous here It's supposed to be the case that in order for this to work that the extension needed to add "public": "true" to their plugin declaration in the manifest: http://code.google.com/chrome/extensions/npapi.html

Solution

Private extension NPAPI plugins should not be loaded by public web pages. Fixed by adding a check that Web pages should not be calling the PluginList directly.

Codes

"Not enforcing private code", "Bypass protection mechanism"

CVE-2011-0779

Context

Loading of corrupted extension's configuration files.

Google Chrome before 9.0.597.84 does not properly handle a missing key in an extension, which allows remote attackers to cause a denial of service (application crash) via a crafted extension.

Problem

VULNERABILITY DETAILS There is a browser crash when loading a corrupted extension file REPRODUCTION CASE 1. Open the attached crx file 2. Click "continue" when prompted 3. The browser crashes The issue is that the crafted extension has an empty signature, that screws it up a memory allocation made in the code while loading the said extension. See Comment #1: We're bombing out on a zero-length allocation in SandboxedExtensionUnpacker::ValidateSignature due to an empty signature. It's a really easy fix; we just need to add the following to the header validation checks we're already doing: if (header.signature_size == 0) { ReportFailure("Key length is zero"); return false; }

Solution

Added a check on SandboxedExtensionUnpacker::ValidateSignature to check for an empty signature. In such case, an error is returned and the loading of the extension is stopped.

Codes

"Application crash", "Incorrect parsing of manifest file", "Accepting malformed manifest files"

CVE-2011-0470

Context

Google Chrome before 8.0.552.237 and Chrome OS before 8.0.552.344 do not properly handle extensions notification, which allows remote attackers to cause a denial of service (application crash) via unspecified vectors.

Problem

Steps ----- 1. Go to https://chrome.google.com/extensions/detail/leaabobiocgllbbfepphaknffhebpomn and install the extension - notification will be seen 2. Go to chrome://extensions and uninstall Notification demo extension 3. Exit browser Root cause: for security information, this is a race condition in window closing that can only happen at shutdown. Specifically, at shutdown the browser stops all renderers and additionally forces all windows to close, then the notification code detects the renderer death and tries to close the window a second time, being unaware of the first.

Solution

Changed the logic for capturing "window close" events in order to prevent the race condition. Listen for APP_TERMINATING in notification ui; close windows earlier in the process before they get clobbered by browser_shutdown leading to a potential double-close.

Codes

"Uninstaller", "Application crash", "Race condition"

CVE-2010-4491

Context

When opening background.html of a malicious extension

Google Chrome before 8.0.552.215 **does not properly restrict privileged extensions**, which allows remote attackers to cause a denial of service (memory corruption) via a crafted extension.

Problem

Caused by an use after free. "Looks like installing an extension can cause a use after free in browser process (sandbox escape)."

Solution

Only call WebInspector_syncDispatch if it's actually a function. The **frame might have navigated away from the front-end page** (which is still weird).

```
if (!dispatchFunction->IsFunction())

return;
```

Codes

"Installer", "Sandbox escape", "Application crash"

CVE-2010-3417

Context

When extension asks for permission

Google Chrome before 6.0.472.59 **does not prompt the user before granting access to the extension history**, which allows attackers to **obtain potentially sensitive information** via unspecified vectors.

Problem

Installing an extension with only the "history" permission does not generate a "Can access your history" warning (unlike apps with the "tabs" permission). This seems incorrect -- the history API gives even **more direct access to user history than the tabs/windows APIs do.** For reference, check out this extension https://chrome.google.com/extensions/detail/cahejgbbfgmlmjgdjlibphdjeldhagkp (not mine) which has the history permission but does not generate an install warning.

Solution

Issue a warning in the above scenario

Codes

"Installer", "Incorrect warning of permissions", "Data leakage"

CVE-2010-3250

Context

When Webpages **attempt to load resources from extensions.**

Unspecified vulnerability in Google Chrome before 6.0.472.53 **allows remote attackers to enumerate the set of installed extensions** via unknown vectors.

Problem

Web pages **should NOT be able to load resources** if there are NO content scripts from that extension on the page. We allow web pages to load resources from extensions as a feature to content scripts. However, if we know that an extension does not have any content scripts on a page, then we should not allow resources to be loaded from that extension. Summary: Refactored extension privilege enumeration and implemented URLPattern comparisons. This will allow checks on per origin extension resource access. Added origin check when loading extension resources.

Solution

Refactored extension privilege enumeration and implemented URLPattern comparisons. This will allow checks on per origin extension resource access. Added origin check when loading extension resources. Details: 1. Modify Extension::GetEffectiveHostPermissions() to return an ExtensionExtent (chrome/common/extensions/extension_extent.h). The ExtensionExtent should contain: - All the URLPatterns from Extension::host_permissions_ - All the URLPatterns from all the matches from all the content scripts. The path component of these URLPatterns should be set to "/*". - The code that is currently there does the above two steps, but instead of returning URLPatterns, it condenses the information down into just the hosts. That part is only needed by the install UI, and should move to extension_install_ui.cc somewhere. 2. Take a look at ExtensionInfo in chrome_url_request_context.h. Add an effective_host_permissions field and populate it from Extension::GetEffectiveHostPermissions() similarly to how the others are done. 3. In extension_protocols.cc, there are some other checks similar to the one you want to do in CreateExtensionURLRequestJob. Use context.effective_host_permissions.ContainsURL() to decide whether to block a resource load. Note that this is only checking whether the extension being requested _could_ run code in the page. Checking whether the extension _did_ run code in the page is much more complicated and would probably involve upstream changes to keep track of whether any injections had been done.

Codes

"Data leakage", "Lack of security checks", "Not enforcing permissions", "Added origin check"

CVE-2010-2110

Context

Related to when the extensions modify a Web page's DOM tree.

Google Chrome before 5.0.375.55 does not properly execute JavaScript code in the extension context, which has unspecified impact and remote attack vectors.

Problem

Summary: Inappropriate context isolation of the "Tabs" world and the "extensions" world. Details: If an extension causes DOM event to be fired (e.g. by modifying DOM tree, which is rather common, or explicitly calling dispatchEvent), the event listener installed by the main page will be erroneously called in extension's context. While actual handler code will be executed in context associated with the JS function (hence in page's world), JS wrappers for DOM objects will be retrieved from extension's world. This allows a malicious page to mess with extension's Object.prototype by following prototype chain of any DOM object -- e.g. by installing property getters/setters there. Data sent to/from background page may sometimes be intercepted and extension's logic altered. Note that the extent appears to be limited to DOM object prototypes -- the page handler still runs in page's world, and whatever code it may trick extension to execute by modifying prototypes, will still be running in page's context.

Solution

Changed its Javascript engine (V8) to check the context of the event to know where to dispatch the event: 48 v8::Local WorldContextHandle::adjustedContext(V8Proxy* proxy) const 49 { 50 if (m_worldToUse == UseMainWorld || !m_context || m_context->get().IsEmpty()) 51 return proxy->mainWorldContext(); 52 53 return v8::Local::New(m_context->get()); 54 }

Codes

"JS Objects Isolation", "Improper Objects Isolation"

CVE-2010-2108

Context

When the user blocks certain plug-ins to be executed it allows remote attackers to bypass the whitelist-mode plugin blocker via unknown vectors.

Problem

Plugins are not always blocked by content settings.

In Chromium, users are allowed to select certain Websites that are authorized to load plugins into the browser. The problem in this CVE is that it allows remote attackers to bypass the whitelist-mode plugin blocker via unknown vectors.

Repro steps:

1- Modify content settings to block all plugins

2- Load test case

3- Click button

4- Notice that the plugin loads and can be played

The bug is likely related to the fact that we only send down the blocked content settings in response to a top-level navigation. In this example, there is no top-level navigation since the newly opened window is to about:blank.

Solution

The key here is to realize that the newly-created window doesn't have a host, and thus no host-based settings could apply to it. And the only way for it to get a host is to navigate, at which time we'll pass the right setting for the new host. Therefore, the only settings that can apply are the global defaults. That suggests the fix: when the renderer wants to create a new view, it sends an IPC to the browser; at that time, we should pass down the default settings so the renderer can apply them to the new view. We can use our existing IPC for this. This should be a small fix so we can merge it to the branch even after the feature freeze, but if anyone wants to do it now they're welcome.

Codes

"Execution of user-blocked plug-in", "Passing user-defined blocked plugins info to app host"

CVE-2010-1229

Context

Sandbox Infrastructure (IPC mechanism) in Google Chrome before 4.1.249.1036 does not properly use pointers, which has unspecified impact and attack vectors.

Problem

A compromised renderer can pass an arbitrary pointer to the plugin process; this pointer is then dereferenced and manipulated (written) in the plugin process. This could be used to corrupt the plugin process and execute arbitrary code outside the sandbox.

Steps to reproduce:

1. Create a plugin.htm page with the following (this just loads the Acrobat plugin so you can mess with it):

```
<body>
        <embed id="pdf" type="application/pdf" hidden="true" width="0" height="0"></embed>
```

```
</body>
<script>
        var obj = document.getElementById("pdf");
        var x = new Object();
        obj.messageHandler = x;
</script>
```

2. Set a breakpoint in the renderer process on ParamTraits::Write()

3. Attach to the renderer process and load plugin.htm.

4. After breaking, change the value of p.type to 7 (which is NPVARIANT_PARAM_OBJECT_POINTER; it should initially be 6, which is NPVARIANT_PARAM_OBJECT_ROUTING_ID).

5. Change the value of p.npobject_pointer to 0x41414141 (or any garbage value) to trigger a crash on a write violation in the plugin process. The plugin process will read the supplied value with ParamTraits::Read() and treat it as an NPObject pointer. The reference count (address+4 on x86) is incremented shortly after reading, which is why a crash occurs when an invalid address is provided.

Root Cause of the Problem:

The **renderer and plugin processes can send over raw NPObjects valid in the other side's address space**. Basically, the way this works is if an NPObject is marshaled over to the other side, an NPObjectStub is created in the caller address space and a NPObjectProxy is created on the other side. The NPObjectProxy is passed the raw NPObject pointer which is **used as a cookie.** If the original NPObject needs to be passed back we pass the underlying NPObject saved in the NPObjectProxy. The receiver does not validate whether this NPObject is **valid before invoking on it**. While this is mostly fine, in the case of a compromised renderer invalid addresses could be passed back to the plugin which would invoke on these addresses and crash.

Solution

Fix is to never **pass raw object pointers** across and just pass the corresponding **routing id** of the NPObjectStub. The **receiver validates this object** by invoking a new method GetNPObjectListenerForRoute on the PluginChannelBase. This method returns the corresponding NPObject listener for the routing id. We then retrieve the underlying NPObject from the listener and use it. The map of NPObjectListeners which is maintained by PluginChannelBase has been changed to hold NPObjectBase pointers instead. NPObjectStub and NPObjectProxy implement the new NPObjectBase interface which provides methods to return the underlying NPObject and the IPC::Channel::Listener pointer.

Codes

"IPC Service", "Passing raw pointers to plug-ins", "Sandbox escape", "Remote Code Execution"