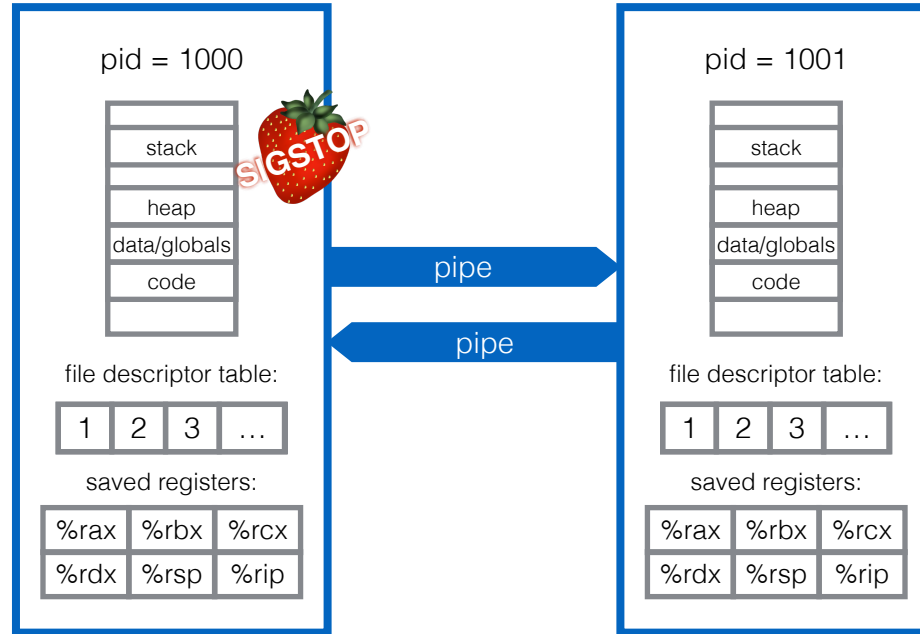


Browsers Case Study

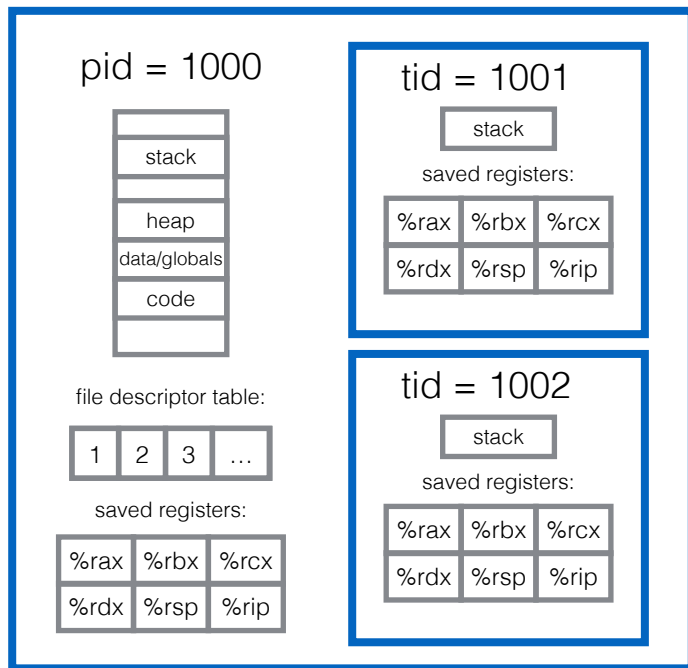
Ryan Eberhardt and Julio Ballista
May 6, 2021

Processes



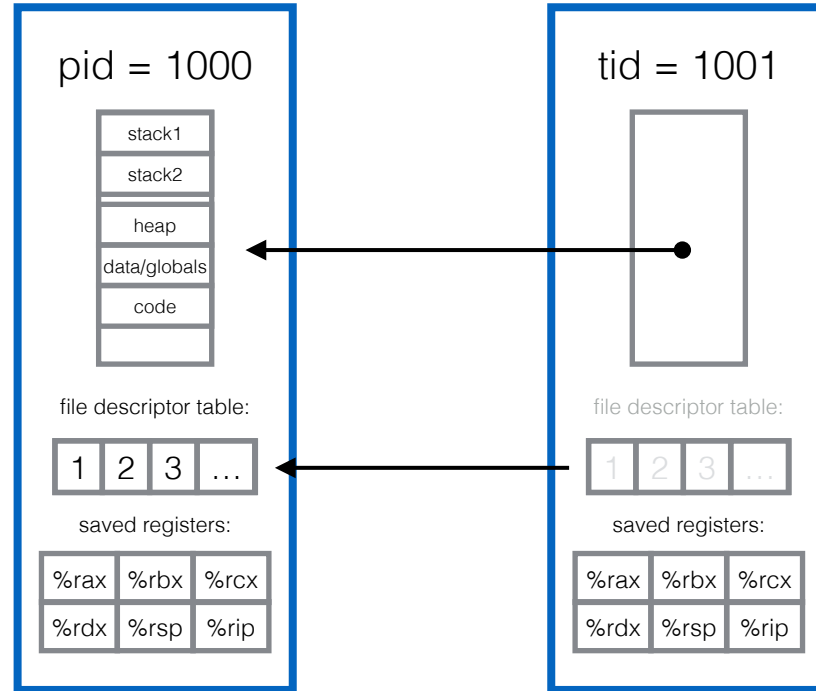
Processes can synchronize using signals and pipes

Threads



Threads are similar to processes; they have a separate stack and saved registers (and a handful of other separated things). But they share most resources across the process

Threads



Under the hood, a thread gets its own “process control block” and is scheduled independently, but it is linked to the process that spawned it

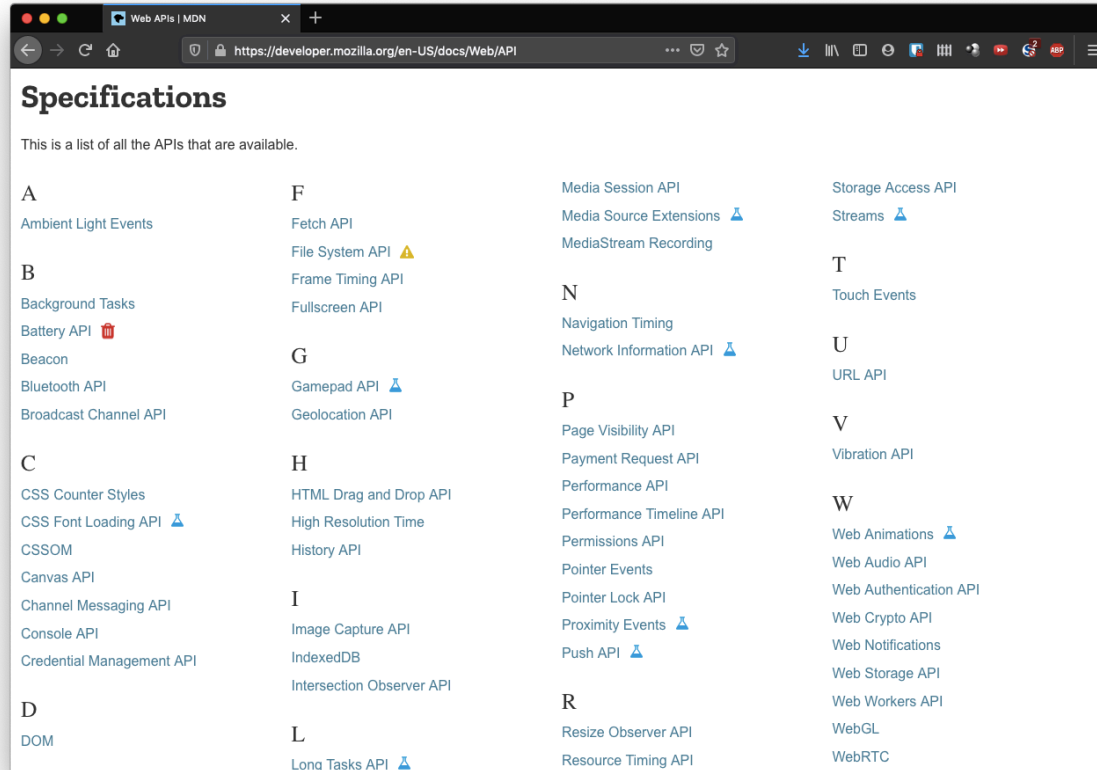
Considerations when designing a browser

- Speed
- Memory usage
- Battery/CPU usage
- Ease of development
- Security, stability

Considerations when designing a browser

- Speed
 - Typically faster to share memory and to use lightweight synchronization primitives
 - Processes incur additional context switching overhead
- Memory usage
 - Processes use more memory
- Battery/CPU usage
 - Processes incur additional context switching overhead
- Ease of development
 - Communication is *WAY* easier using threads
 - (That being said, bugs caused by multithreading are extremely hard to track down)
- Security, stability
 - Multiprocessing provides isolation. Multithreading does not.

Modern browsers are essentially operating systems



<https://developer.mozilla.org/en-US/docs/Web/API>

Modern browsers are essentially operating systems

- Storage APIs
- Concurrency APIs
- Hardware APIs (e.g. communicate with MIDI devices, even GPU)
- Run assembly
- Run Windows 95: <https://win95.ajf.me/>

Motivation for Chrome

It's nearly impossible to build a rendering engine that never crashes or hangs. It's also nearly impossible to build a rendering engine that is perfectly secure.

In some ways, the state of web browsers around 2006 was like that of the single-user, co-operatively multi-tasked operating systems of the past. As a misbehaving application in such an operating system could take down the entire system, so could a misbehaving web page in a web browser. All it took is one browser or plug-in bug to bring down the entire browser and all of the currently running tabs.

Modern operating systems are more robust because they put applications into separate processes that are walled off from one another. A crash in one application generally does not impair other applications or the integrity of the operating system, and each user's access to other users' data is restricted.

<https://www.chromium.org/developers/design-documents/multi-process-architecture>

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. Note that this only includes bugs that are reported to us or are found by our team.*
- Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).*
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).*

<https://www.chromium.org/Home/chromium-security/site-isolation>

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that **determined attackers will be able to find a way to compromise a renderer process**, for several reasons:

- Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. Note that this only includes bugs that are reported to us or are found by our team.
- Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).

<https://www.chromium.org/Home/chromium-security/site-isolation>

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- **Past experience suggests that potentially exploitable bugs will be present in future Chrome releases.** There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. Note that this only includes bugs that are reported to us or are found by our team.
- Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). **This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc.** Note that this only includes bugs that are reported to us or are found by our team.
- Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).

<https://www.chromium.org/Home/chromium-security/site-isolation>

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. **Note that this only includes bugs that are reported to us or are found by our team.***
- Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).*
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).*

Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- *Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. Note that this only includes bugs that are reported to us or are found by our team.*
- **Security bugs can often be made exploitable:** even 1-byte buffer overruns [can be turned into an exploit](#).
- Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).

<https://www.chromium.org/Home/chromium-security/site-isolation>

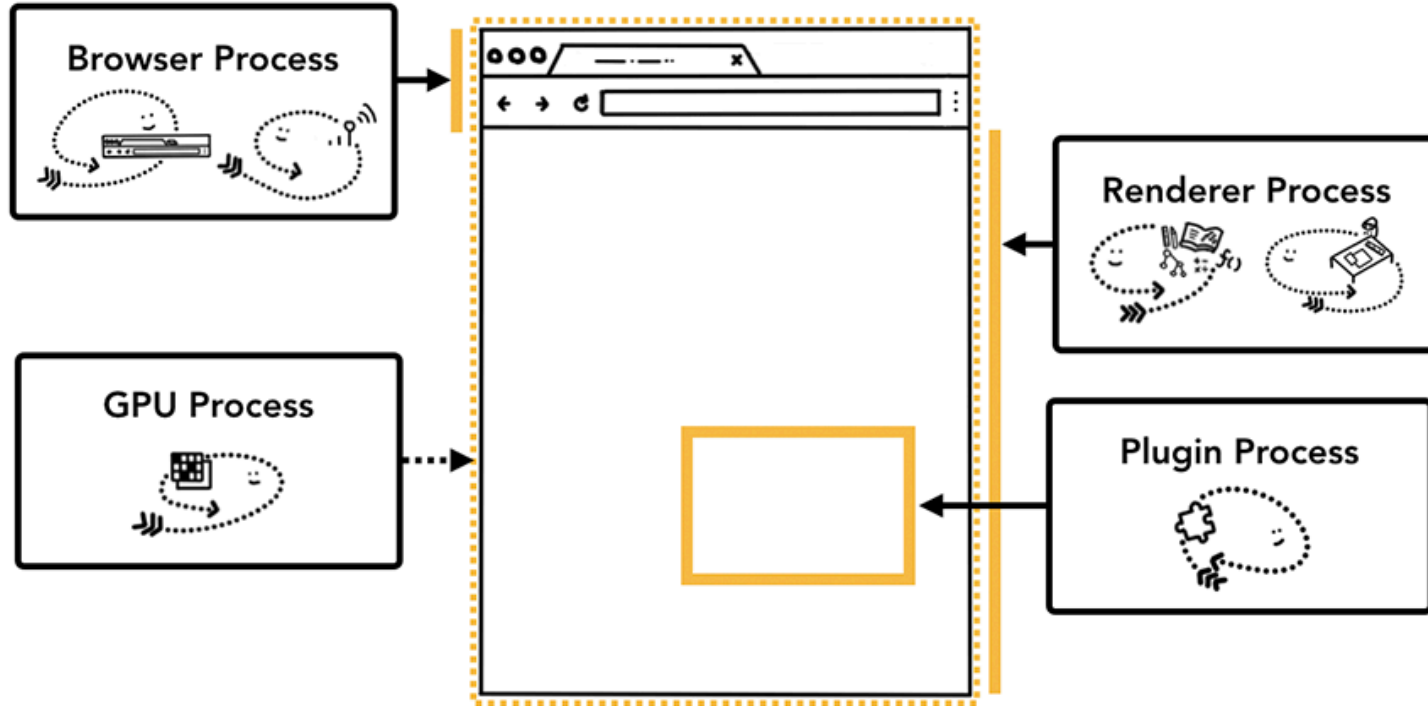
Motivation for Chrome

Compromised renderer processes (also known as "arbitrary code execution" attacks in the renderer process) need to be explicitly included in a browser's security threat model. We assume that determined attackers will be able to find a way to compromise a renderer process, for several reasons:

- *Past experience suggests that potentially exploitable bugs will be present in future Chrome releases. There were [10 potentially exploitable bugs in renderer components in M69](#), [5 in M70](#), [13 in M71](#), [13 in M72](#), [15 in M73](#). This volume of bugs holds steady despite years of investment into developer education, fuzzing, Vulnerability Reward Programs, etc. Note that this only includes bugs that are reported to us or are found by our team.*
- *Security bugs can often be made exploitable: even 1-byte buffer overruns [can be turned into an exploit](#).*
- ***Deployed mitigations (like [ASLR](#) or [DEP](#)) are [not always effective](#).***

<https://www.chromium.org/Home/chromium-security/site-isolation>

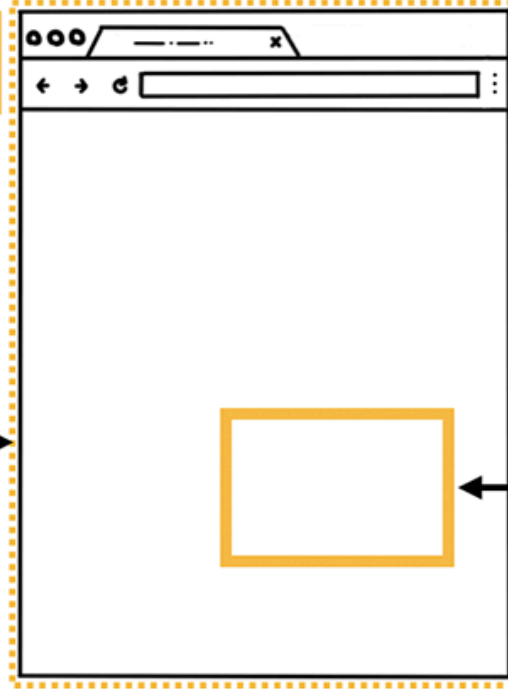
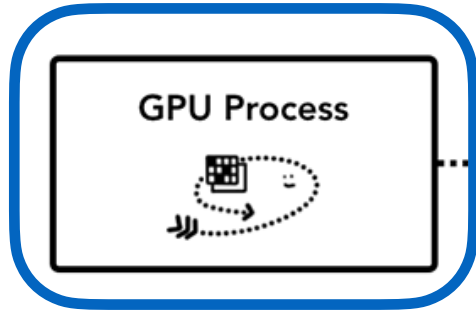
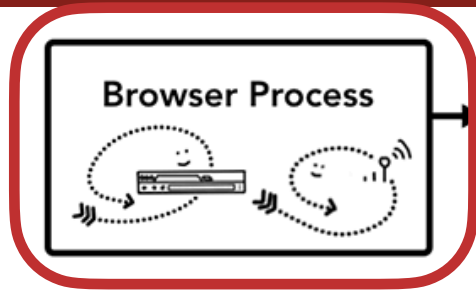
Chrome architecture



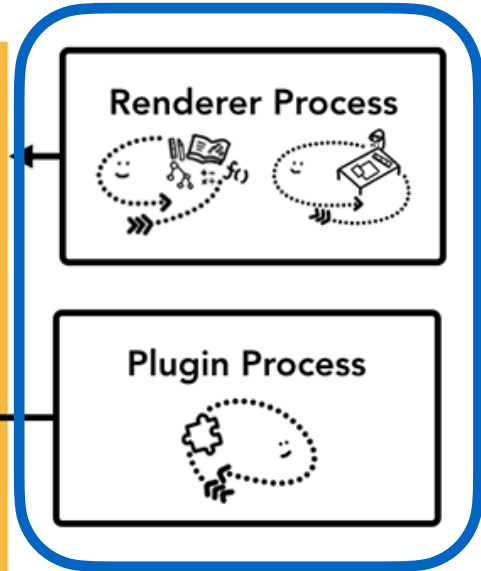
REALLY CUTE diagrams from <https://developers.google.com/web/updates/2018/09/inside-browser-part1>
(great read!)

Sandboxing: Defense against RCE

Privileged process:
Minimal
attack
surface

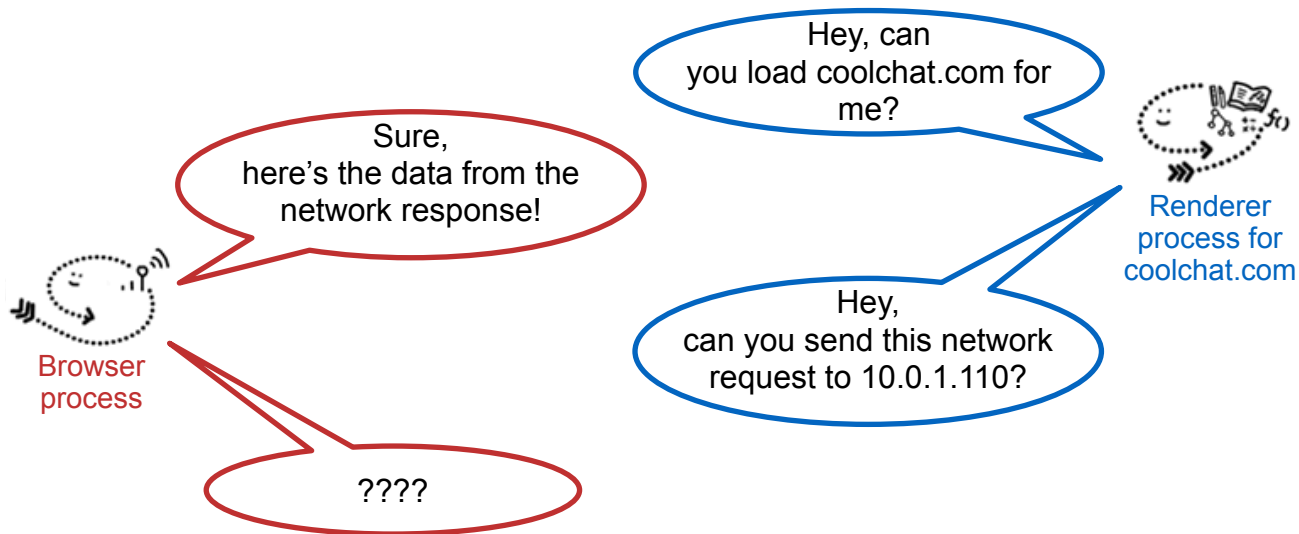


Unprivileged processes:
Majority of attack surface

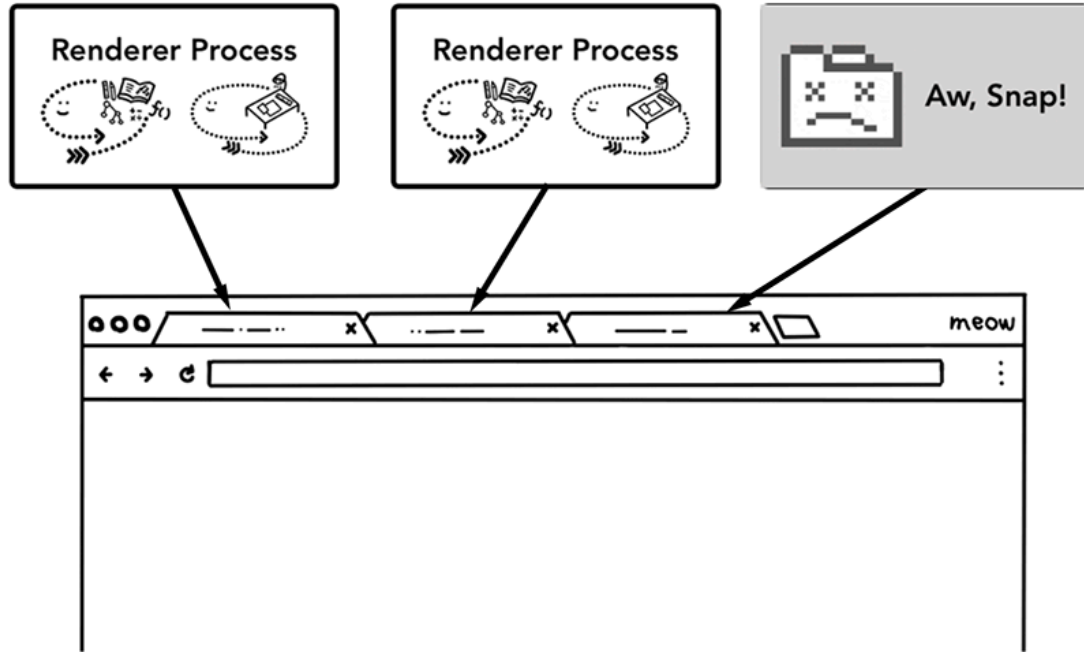


REALLY CUTE diagrams from <https://developers.google.com/web/updates/2018/09/inside-browser-part1>
(great read!)

Sandboxing: Defense against RCE

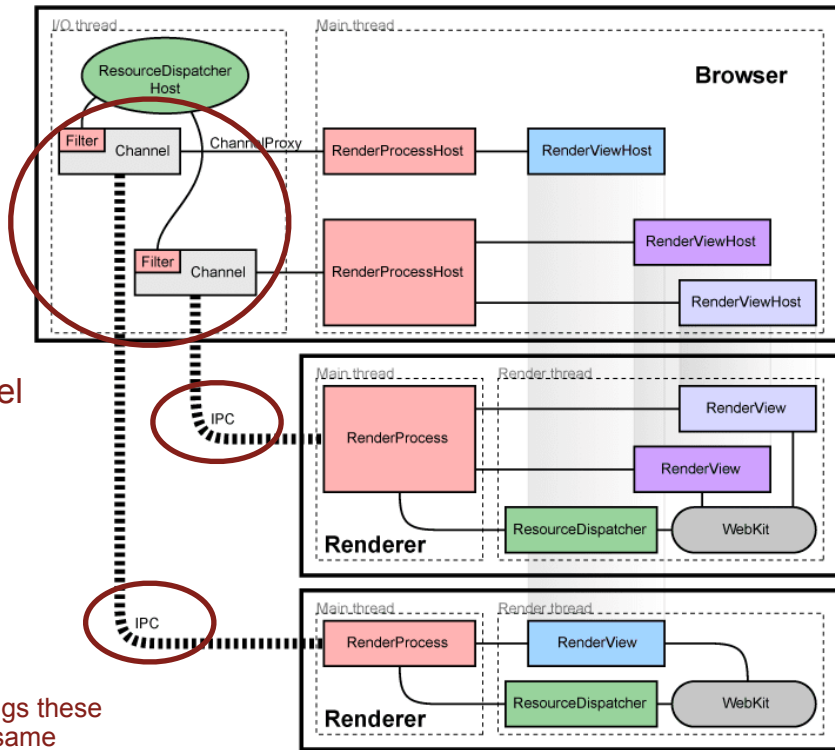


Isolation: Increased robustness



REALLY CUTE diagrams from <https://developers.google.com/web/updates/2018/09/inside-browser-part1>
(great read!)

Chrome architecture



IPC channels = pipes*

Message passing model

Events (e.g. click, keystroke, etc) are relayed through these pipes! No signals

* they use slightly fancier things these days, but the idea is still the same

Sandboxed processes: no access to network, filesystem, etc

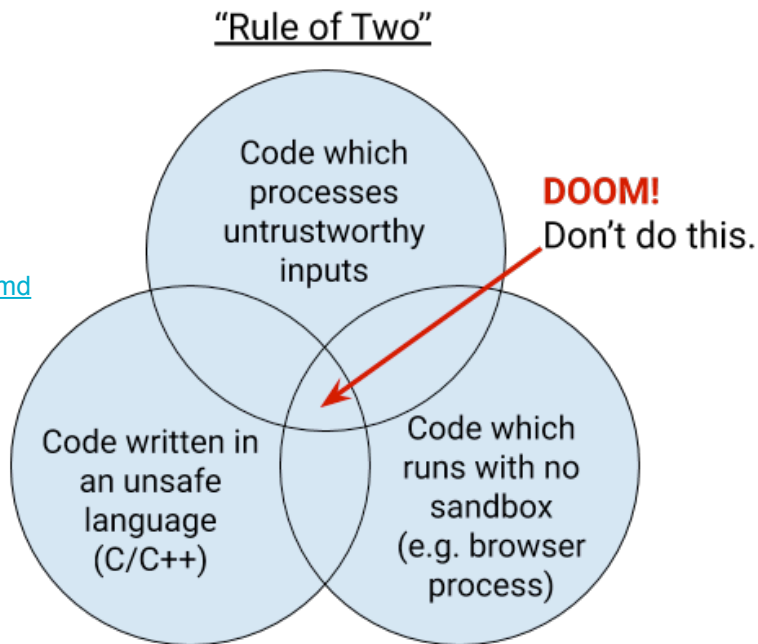
If there is embedded content, may use multiple threads to render that content and manage communication between frames

Chromium Rule of Two

The Rule Of 2 is: Pick no more than 2 of

- untrustworthy inputs;
- unsafe implementation language; and
- high privilege.

<https://chromium.googlesource.com/chromium/src/+master/docs/security/rule-of-2.md>



Not good enough

- What does all this work buy us?
 - Isolation between tabs
 - Isolation between (potentially malicious) websites and the host
- What does it *not* buy us?
 - Isolation between resources *within* a tab

Embedded content

The screenshot displays the Daily Mail website interface with several key elements highlighted by red boxes:

- Top Navigation:** Includes the Daily Mail logo, navigation links (Home, U.K., News, Sports, U.S. Showbiz, Australia, Femail, Health, Science, Money, Video, Travel, Shop, DailyMailTV), and a weather forecast for Wednesday, May 5th, 2021 (7PM 69°F, 10PM 60°F, 5-Day Forecast).
- Left Advertisement:** A Merrell advertisement for 'NOVA & ANTORA 2' sneakers, featuring the text 'GO PLACES SNEAKERS CAN'T' and 'AVAILABLE AT REI CO-OP'.
- Center Content:** A 'lendingtree Refinance Calculator' widget. It shows a current rate of 2.00% and an APR of 2.06% as of May 5, 2021. A graph indicates that 'Rates are at historic lows!'. Input fields include a loan amount of \$400,000, a 15-Year Fixed term, and an Excellent credit score. A 'Calculate Payment' button is present.
- Right Advertisement:** A Merrell advertisement for 'NOVA & ANTORA 2' sneakers, identical to the left one.
- Main Article:** A headline reads: 'Two California students, 19 and 20, are found guilty of murdering Italian cop and sentenced to life in prison for botched 2019 drug raid while on vacation'. Below the headline is a collage of images related to the story, including a man in a dark jacket, a man in a blue uniform, and a man in a white mask.
- Bottom Left Advertisement:** An advertisement for 'sweetgreen' titled 'Connecting People to Real Food'. It promotes 'this season's newest menu additions' and includes a map for a location in Palo Alto, along with 'STORE INFO' and 'DIRECTIONS' buttons.

Embedded content



Same-origin policy: `www.evil.com` can embed `bank.com`, but cannot interact with `bank.com` or see its data

Embedded content

- Site Isolation Project (2015-2019) aimed to put resources for different origins in different processes
- Extremely difficult undertaking. Cross-frame communication is common (JS postMessage API), and embedded frames need to share render buffers
 - Involved rearchitecting the most core parts of Chrome
- Became especially important in Jan 2018: Spectre and Meltdown
 - When the hardware fails to uphold its guarantees, JS can read arbitrary process memory (even kernel memory, and even if your software has no bugs)!
- Paper/video: <https://www.usenix.org/conference/usenixsecurity19/presentation/reis>

Still not good enough!

The screenshot shows a web browser window with the following elements:

- Browser Tab:** "Digging into the Third Zero-Day Chrome Flaw of 2021"
- Address Bar:** "https://www.tripwire.com/state-of-security/featured/digging-into-the-third-zero-day-chrome-flaw-of-2021"
- Page Header:** "The State of Security" with the tagline "NEWS. TRENDS. INSIGHTS." and the Tripwire logo.
- Navigation:** "FEATURED ARTICLES", "TOPICS", "PODCASTS", "VERT", "RESOURCES", and an "EXPLORE TRIPWIRE" button.
- Article Title:** "Digging Into the Third Zero-Day Chrome Flaw of 2021"
- Author Info:** "TRIPWIRE GUEST AUTHORS" with a profile icon, "APR 8, 2021", and a "FEATURED ARTICLES" badge.
- Image:** A colorful, abstract graphic with a circular shape in the center, rendered in a halftone or dot-matrix style.
- Right Sidebar:** A blue promotional box for a newsletter: "Join over 20,000 IT security pros who get our top stories delivered to their inbox every week!" with a "SUBSCRIBE" button and the Tripwire logo. Below it is a red promotional box for a free e-book: "New Free E-Book! MASTERING CONFIGURATION MANAGEMENT for the Modern Enterprise".

Still not good enough!

The screenshot shows a web browser window displaying a Threatpost article. The article title is "Google Chrome V8 Bug Allows Remote Code-Execution". Below the title is a large image of the Google Chrome logo on a screen. The article text states that Google rolled out the Chrome 90 stable channel release to address this and eight other security vulnerabilities. It mentions that the high-severity V8 issue is tracked as CVE-2021-21227 and was reported by Gengming Liu from Singular Security Lab. The article is dated April 28, 2021, at 1:48 pm. The author's name, "Iord Stănilă", is circled in red. On the right side of the page, there is a "Newsletter" section with a "Subscribe now" button.

Google Chrome V8 Bug Allows Remote Code-Execution

INFOSEC INSIDER

- Sneakers, Gaming, Nvidia Cards: Retailers Can Stop Shopping Bots**
May 4, 2021
- A Tale of Two Hacks: From SolarWinds to Microsoft Exchange**
April 30, 2021
- Smishing: Why Text-Based Phishing Should Be on Every CISO's Radar**
April 27, 2021
- 5 Fundamental But Effective IoT Device Security Controls**
April 23, 2021
- 4 Innovative Ways Cyberattackers Hunt for Security Bugs**
April 21, 2021

Newsletter
Subscribe to *Threatpost Today*
Join thousands of people who receive the latest breaking cybersecurity news every day.
Subscribe now

Author: Iord Stănilă
April 28, 2021 / 1:48 pm
minute read

The internet behemoth rolled out the Chrome 90 stable channel release to address this and eight other security vulnerabilities.

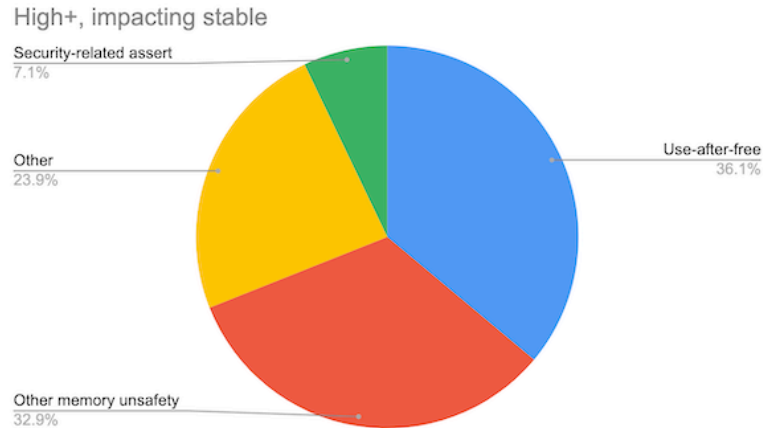
Google's Chrome browser has several security vulnerabilities that could pave the way to multiple types of attacks, including a V8 bug that could allow remote code execution (RCE) within a user's browser.

The high-severity V8 issue is tracked as CVE-2021-21227, and was reported by Gengming Liu from Singular Security Lab. Google describes the bug as "insufficient data validation in V8" but is keeping other details close to its vest.

[Write a comment](#)

April 28, 2021 — update your browsers!

Still not good enough!



- <https://www.chromium.org/Home/chromium-security/memory-safety>
- 70% of high-severity security bugs are caused by memory safety issues

The limits of sandboxing

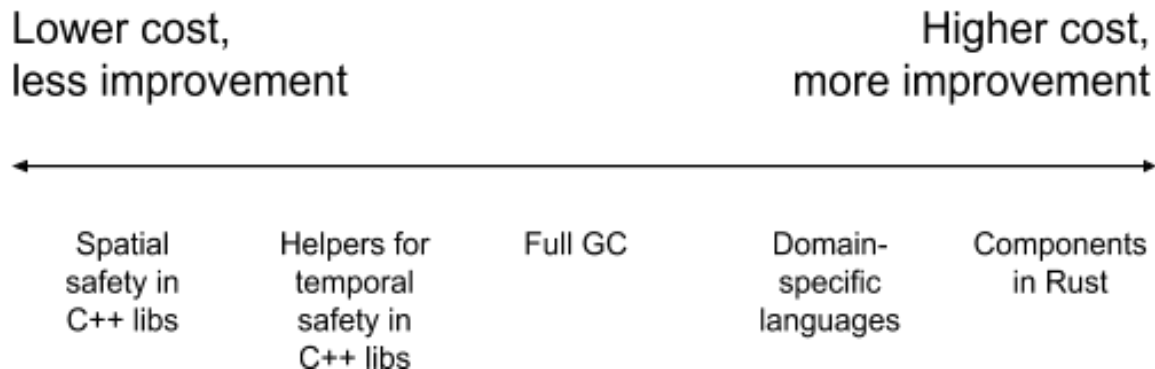
Chromium's [security architecture](#) has always been designed to assume that these bugs exist, and code is sandboxed to stop them taking over the host machine... **But we are reaching the limits of sandboxing and site isolation.**

A key limitation is that the process is the smallest unit of isolation, but processes are not cheap.

We still have processes sharing information about multiple sites. For example, **the network service is a large component written in C++ whose job is parsing very complex inputs from any maniac on the network. This is what we call “the doom zone”** in our [Rule Of 2](#) policy: the network service is a large, soft target and [vulnerabilities](#) there are of [Critical](#) severity.

Just as Site Isolation improved safety by tying renderers to specific sites, we can imagine doing the same with the network service: we could have many network service processes, each tied to a site or (preferably) an origin. That would be beautiful, and would hugely reduce the severity of network service compromise. **However, it would also explode the number of processes Chromium needs, with all the efficiency concerns that raises.**

What we're trying



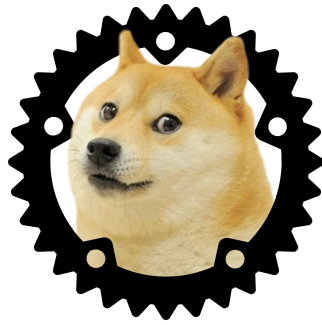
We expect this strategy will boil down to two major strands:

- *Significant changes to the C++ developer experience, with some performance impact. (For instance, **no raw pointers, bounds checks, and garbage collection.**)*
- *An option of a programming language designed for **compile-time safety checks with less runtime performance impact** — but obviously there is a cost to bridge between C++ and that new language.*

Anatomy of a sandbox escape

- <https://blog.chromium.org/2012/05/tale-of-two-pwnies-part-1.html> (2012 but it's more accessible than some other writeups)
 - First exploit chains together *six bugs* to escape the sandbox
 - Second one uses *ten(!)*
- <https://googleprojectzero.blogspot.com/2019/04/virtually-unlimited-memory-escaping.html> (2019)

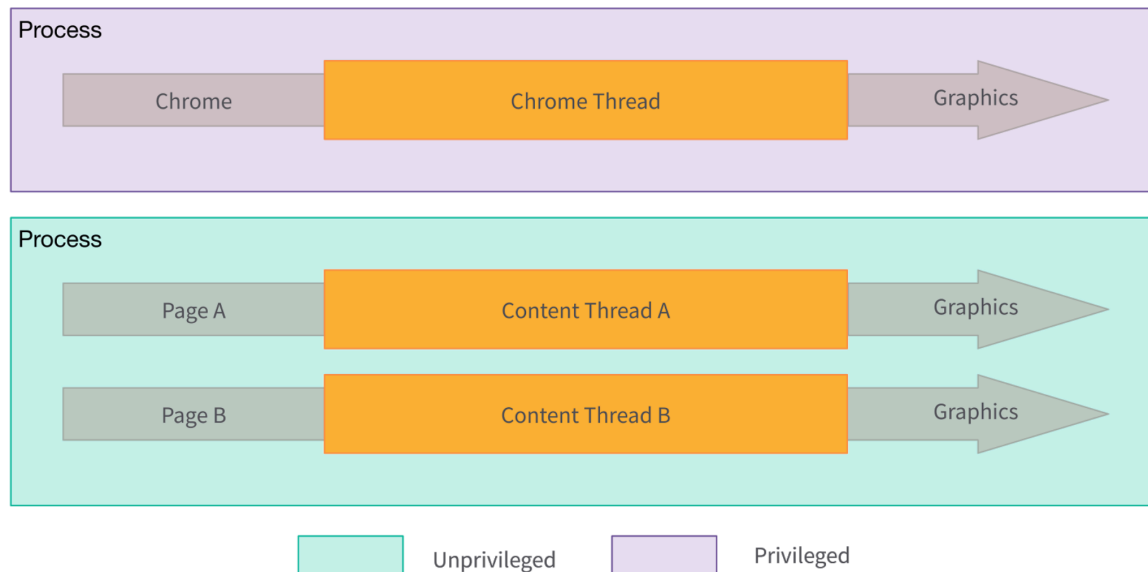
Alternative approach: Servo



Alternative approach

- Wha — this all sounds like a ton of work!
- What if we just implement the browser in a language that helps us avoid these mistakes in the first place?
- Servo is an experimental browser engine from Mozilla Research written in Rust
 - Components of Servo have been gradually adapted in Firefox (Gecko)
 - Note: security was not the primary motivation for Servo, but it's what we're focusing on here

Servo approach



- Have *some* sandboxing, but don't sweat it too much. Tabs often share processes
- Everything is written in Rust, so we don't have to worry about security issues, right?

Rust does not prevent all bugs

[Implications of Rewriting a Browser Component in Rust:](#)

Over the course of its lifetime, there have been 69 security bugs in Firefox's style component. ***If we'd had a time machine and could have written this component in Rust from the start, 51 (73.9%) of these bugs would not have been possible.*** While Rust makes it easier to write better code, it's not foolproof.

There are classes of bugs that Rust explicitly does not address—particularly correctness bugs. In fact, during the Quantum CSS rewrite, engineers accidentally reintroduced a critical security bug that had previously been patched in the C++ code, regressing the fix for bug 641731... As a trivial history-stealing bug, this is rated security-high.

Rust code can have memory safety issues too!

- Libraries often use “unsafe Rust” when we need to do things that the compiler can’t guarantee is safe, e.g.:
 - Data structures
 - Implementing new concurrency primitives
 - Running platform-specific assembly instructions
- Testing Firefox with ThreadSanitizer yielded two race conditions in Rust low-level library code: <https://hacks.mozilla.org/2021/04/eliminating-data-races-in-firefox-a-technical-report/>

Limitations of “Rewrite it in Rust!”

- It's impractical to rewrite an entire project in a new language
 - The majority of Firefox is still written in C++
- Rewriting projects introduces bugs (and sometimes reintroduces old, long-fixed bugs)
- Rust code still has security vulnerabilities
 - From correctness issues
 - And even memory safety issues from `unsafe` code

Conclusion

- There is no perfect solution
- We need all the tools we can get:
 - Memory-safe programming languages
 - Sandboxing
 - Fuzzing and dynamic analysis
 - Code review, audits, bug bounty programs
 - More!

More relevant reading

- How Chrome does fork():

<http://neugierig.org/software/chromium/notes/2011/08/zygote.html>

Fun related bug report: <https://bugs.chromium.org/p/chromium/issues/detail?id=35793>

What steps will reproduce the problem?

- 1. Develop a webapp, use chrome's devtools, minding your own business*
- 2. In the meantime, let chrome silently autoupdate in the background*

What is the expected result?

Devtools continue working

What happens instead?

Devtools break after refreshing the page after the autoupdate happened.