Chrome Security / Site Isolation & Spectre

...

Charlie Reis
creis@chromium.org

My Background

- UW CSE PhD in 2009
  - Systems group w/ Hank Levy, Steve Gribble, Dan Grossman

- Chrome team since 2008
- Chrome Security since 2012
- Tech Lead for Site Isolation

How do you secure a browser?

Product / Platform Security

- **Defending a product is hard**, compared to finding attacks
  - Even harder: a platform designed to run untrusted code

- Just fix all the security bugs?

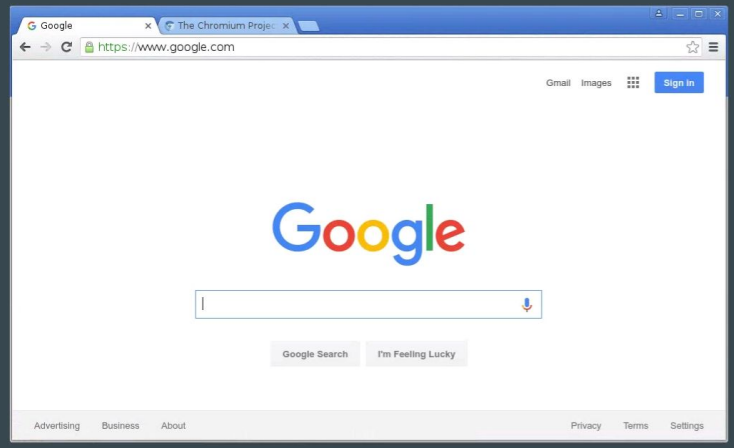
- Need to approach **from many directions**
- Go after **long-term improvements**

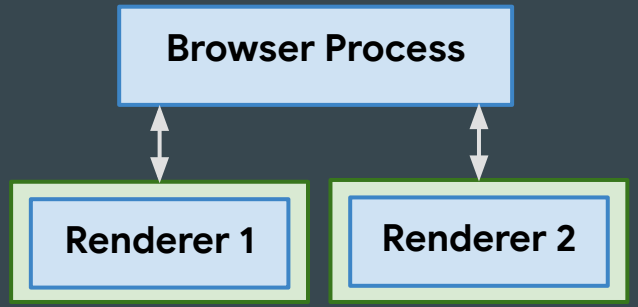
Security Architecture Team

Multi-Process Architecture

- Privileged browser process
- Sandboxed renderer processes
- >5 million lines of C++

- Handle untrusted input in sandbox
  - Harder for exploits to escape



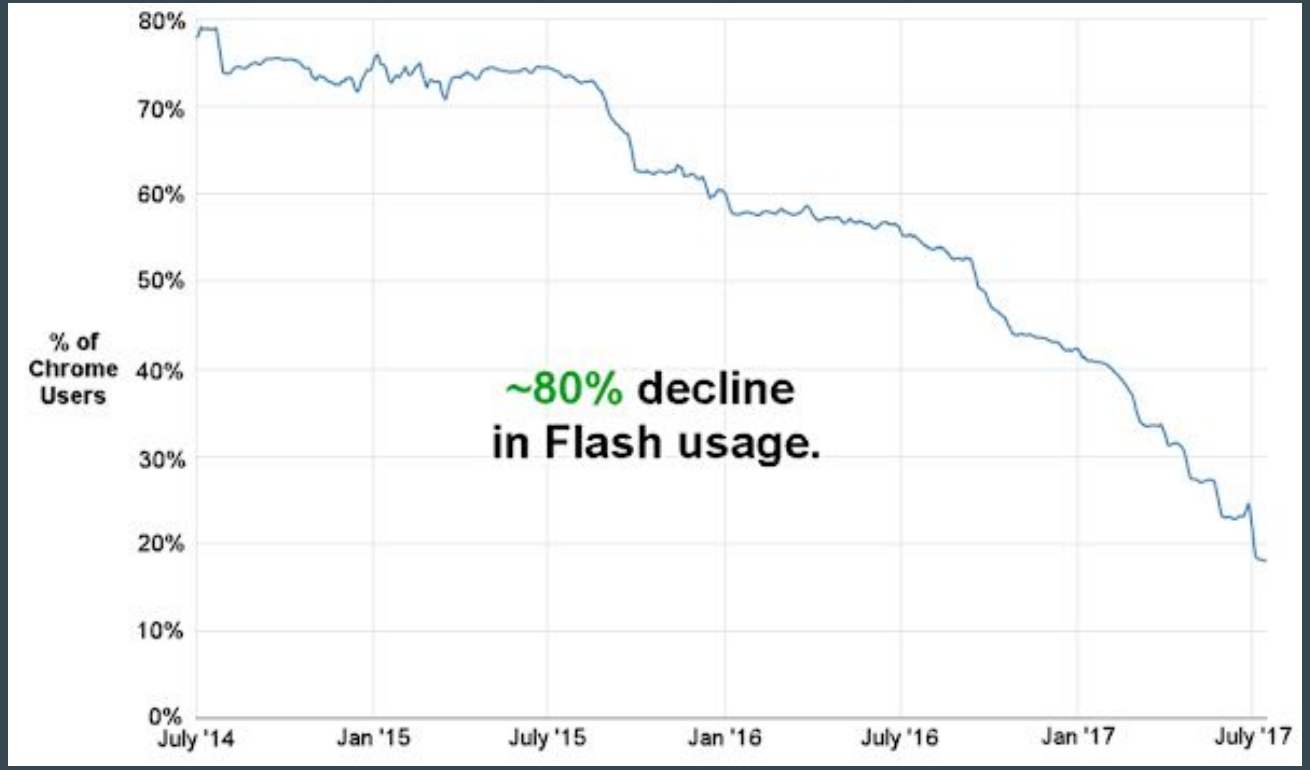


Improving Security Architecture

- Goals:
  - Limit damage that attacks can cause
  - Defense in depth

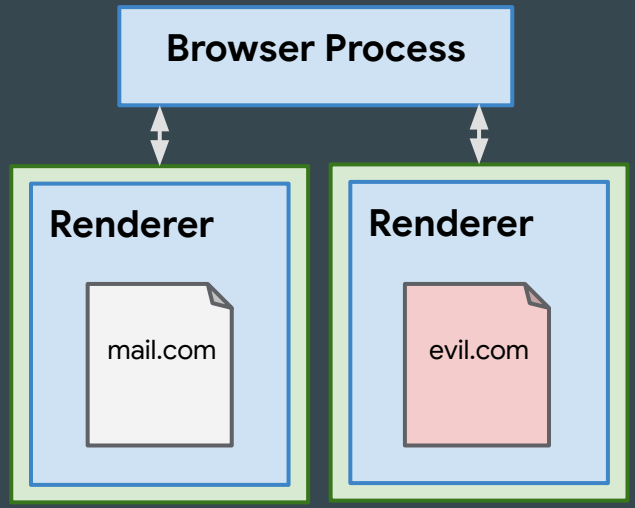
- Examples:
  - Making the sandbox stronger
  - Getting Flash, PDF into the sandbox
  - Deprecating NPAPI

Example: Flash deprecation (!!)



Site Isolation (my team)

- Sandbox should help enforce the Same Origin Policy
  - Don't let an exploited renderer steal your web accounts!
- Harder than it seems:
  - Cross-site iframes within a page
  - Move iframes out of process?
    (Browsers aren't built that way!)
- We'll come back to this...



Finding and Fixing Security Bugs

Vulnerability Reward Program

- Pay external researchers to find the bugs!
  - **$100,000** for full ChromeOS exploit (across guests and reboots)
  - **$22,500** for full chain sandbox escape on desktop

- We learn a lot from these exploits
  - How can attackers evade our security checks?
  - Which parts of the system need stronger defenses?

ClusterFuzz

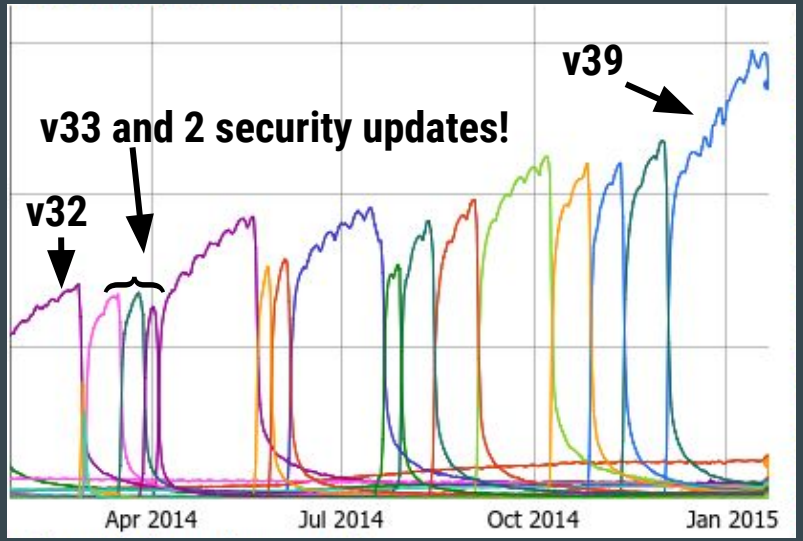
- **Fuzzing**: Send random input to make a program crash
  - Spot possible security bugs (e.g., use-after-free)

- Scale it up!
  - 25,000+ cores
  - 4,000+ security bugs fixed

- Invite others!
  - Run externally submitted fuzzers, pay for bugs they find
  - OSS-Fuzz for other projects

Get fixes to users quickly

- Auto-Update
  - Get users onto newest versions
  - Whole system on ChromeOS



Project Zero Team

- **Goal: No "Zero Day" Exploits**

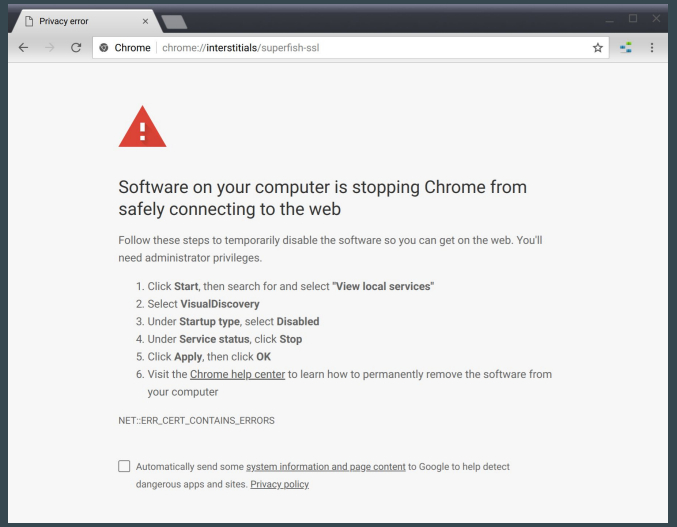
- Find the scariest bugs, in any software
- Responsibly disclose and ensure they get fixed (90 day deadline)

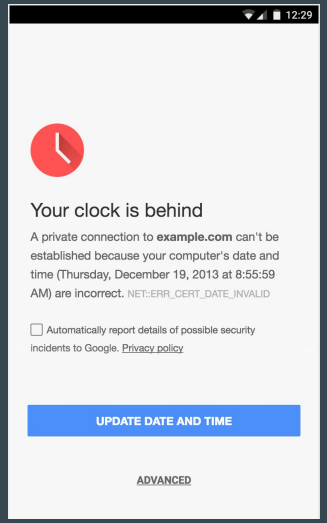
- Examples:
  - Spectre / Meltdown
  - Anti-Virus vulnerabilities

And many other sub-teams...

Security UX: Helping users make safe choices

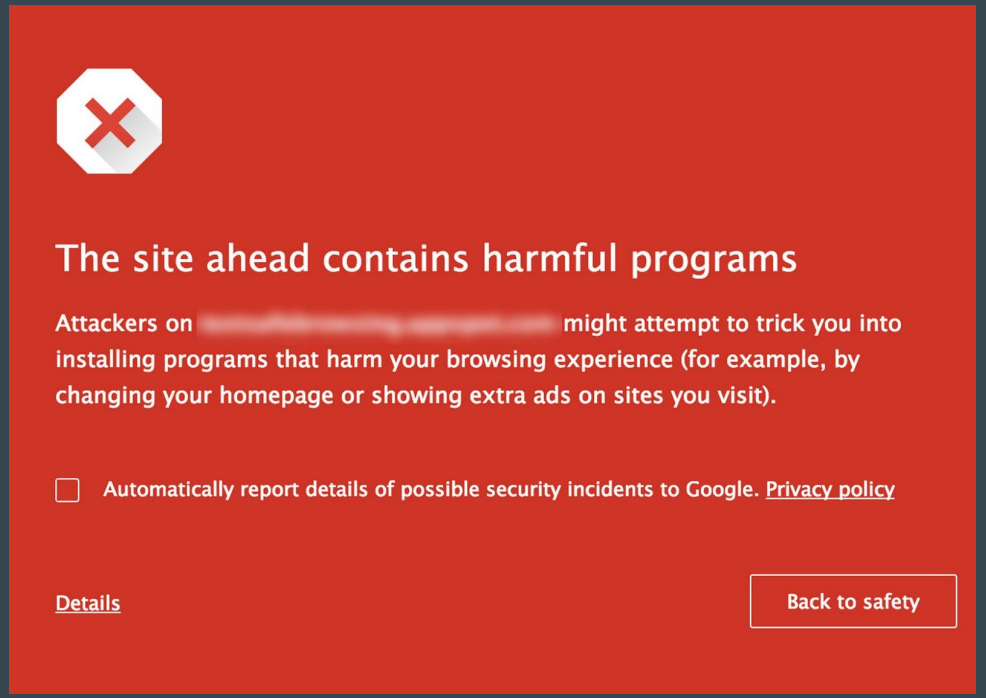
- Avoid "warning fatigue"
  - Give better info
  - Show fewer errors
- Research on HTTPS errors
  - Understand reasons
  - More actionable warnings





Safe Browsing

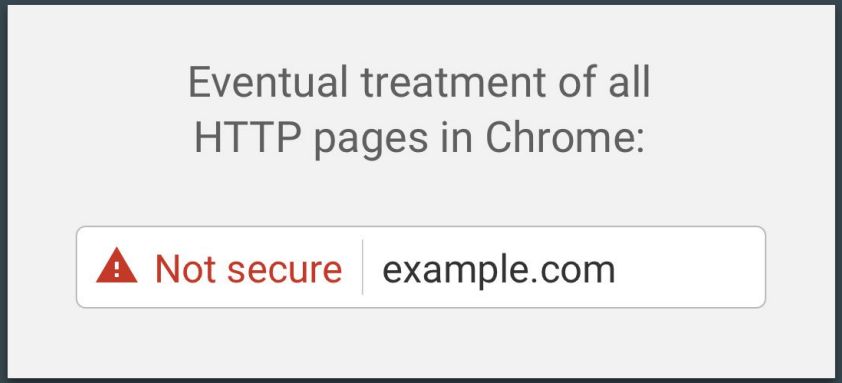
- Detect malware, phishing, etc
  - Crawl web, build up a list
- Warn the user on navigations and downloads



Secure Web Platform

- Improve web security standards: CSP, cookies, etc

- Moar TLS!
  - Certificate pinning, Certificate Transparency
  - HTTP should be derided like telnet
    - 90 of top 100 sites now default to HTTPS (up from 24 in 2016)



Security Reviews

- All of Chrome Security helps the rest of Chrome team

- Design level reviews
- Code level reviews

Summary

- Defense requires many different approaches
  - Design code for security
    - Minimize the damage bugs can cause
    - Defense in depth
  - Find and fix inevitable security bugs
  - Make it usable in practice

- Long-term efforts are worth pursuing

Site Isolation (and Spectre)

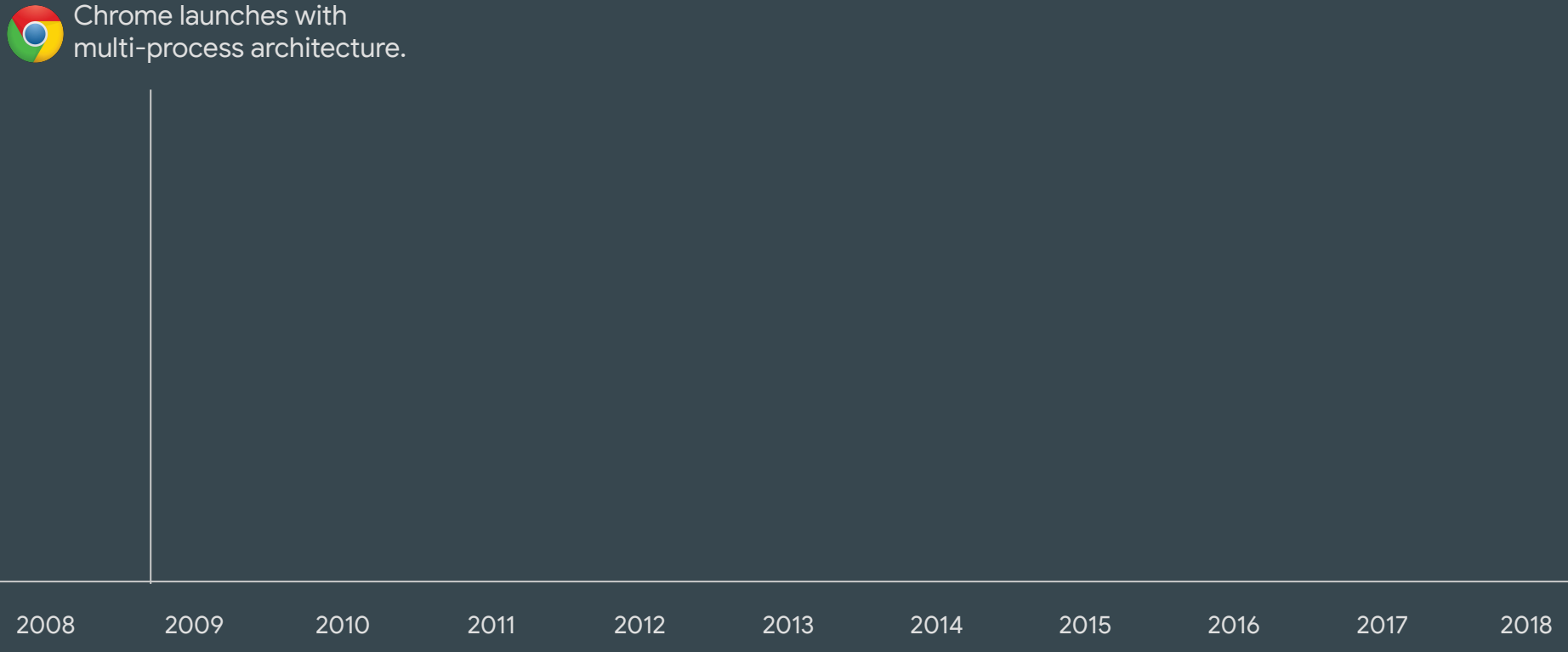
...

A brief history

Early Days at UW CSE

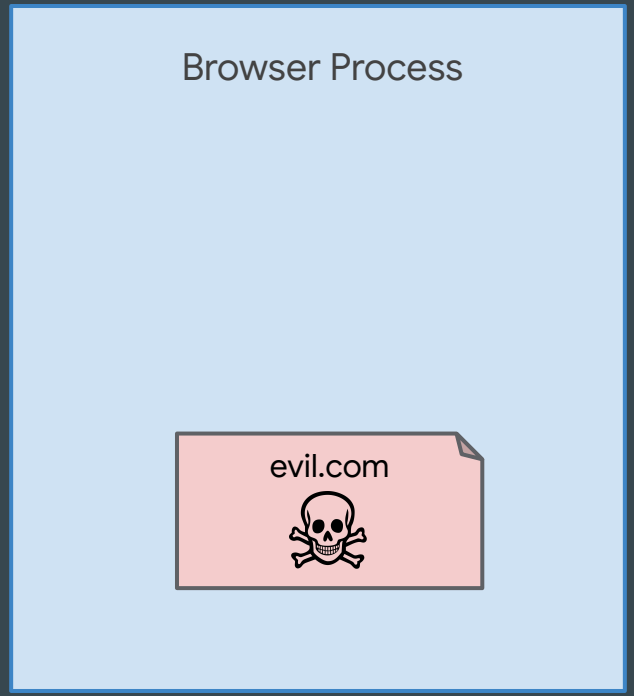
- 2007: Multi-process browser prototype

- 2008: Intern on Chrome team
  - Add cross-process navigations
  - SiteInstance abstraction



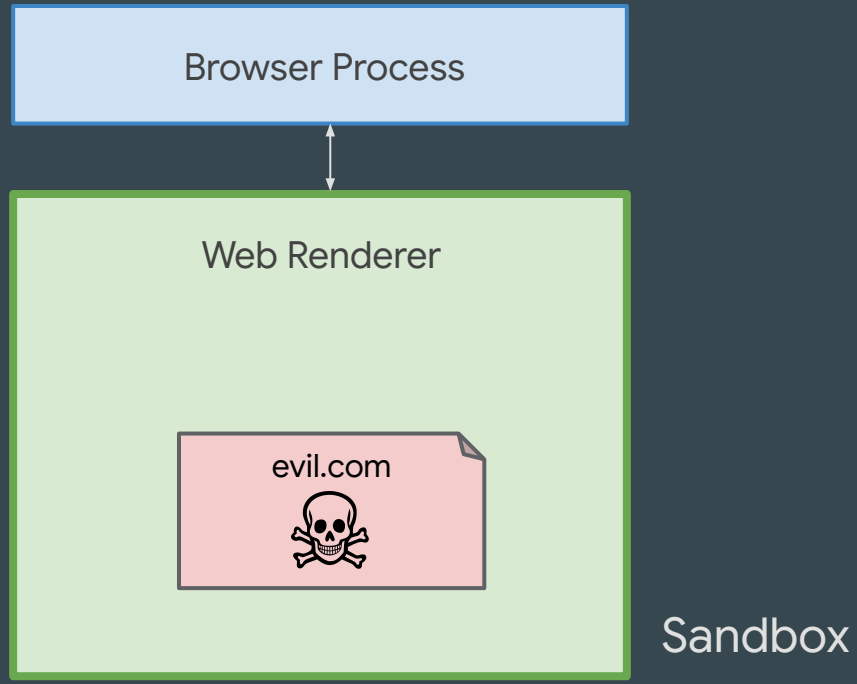
History of Site Isolation in Chrome

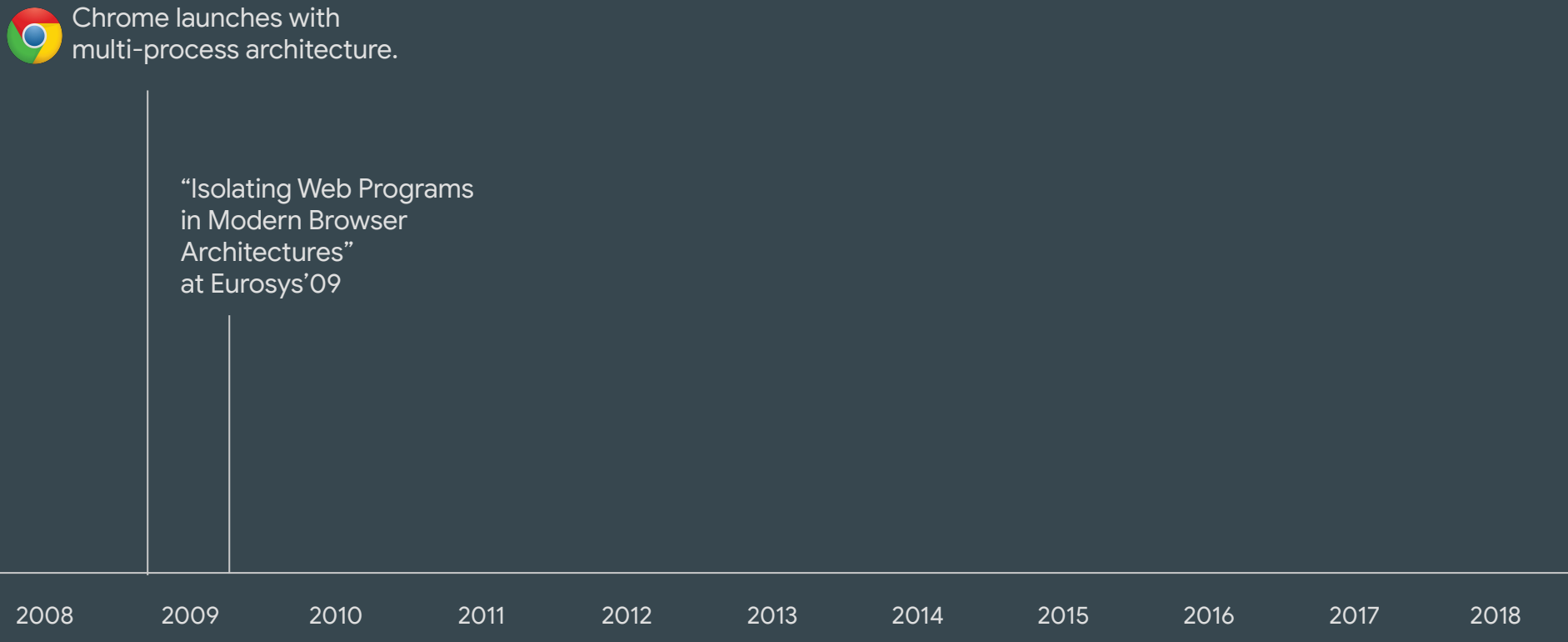
Older Browsers



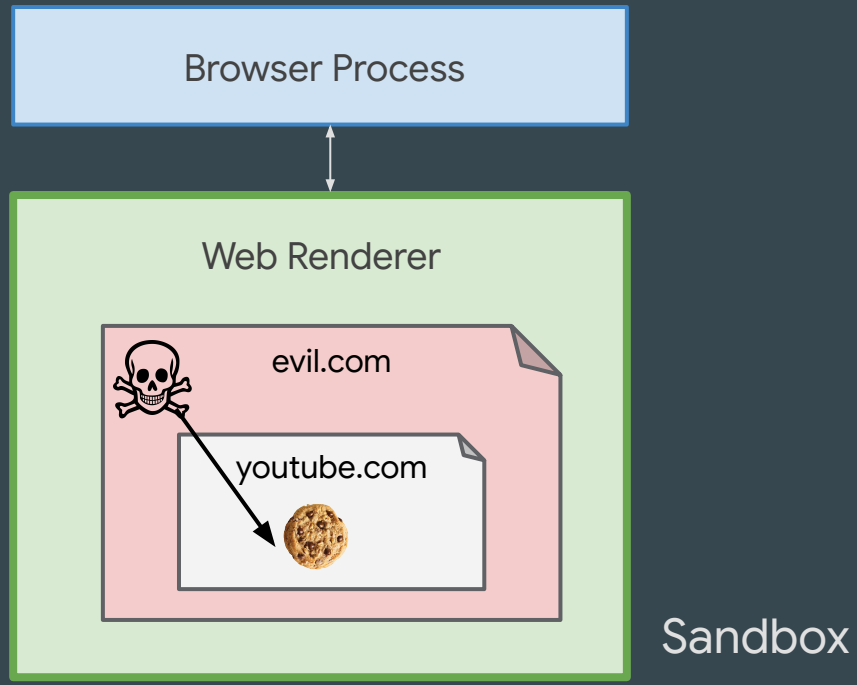
No Sandbox

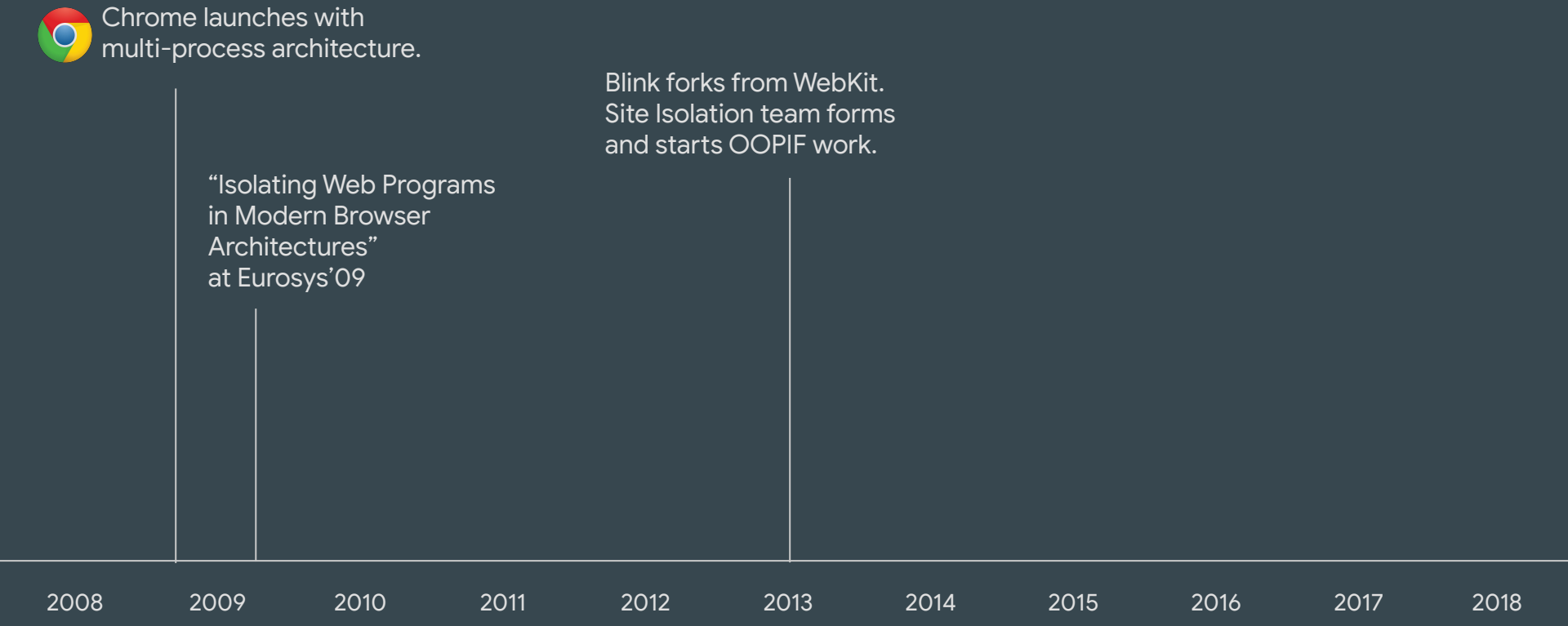
Multi-Process Browser



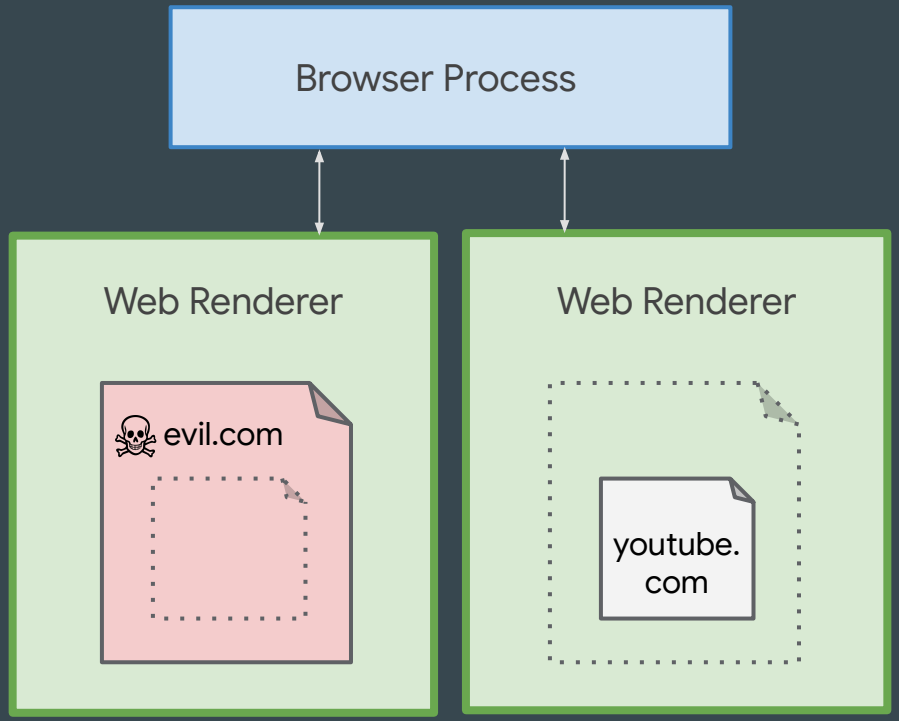


History of Site Isolation in Chrome

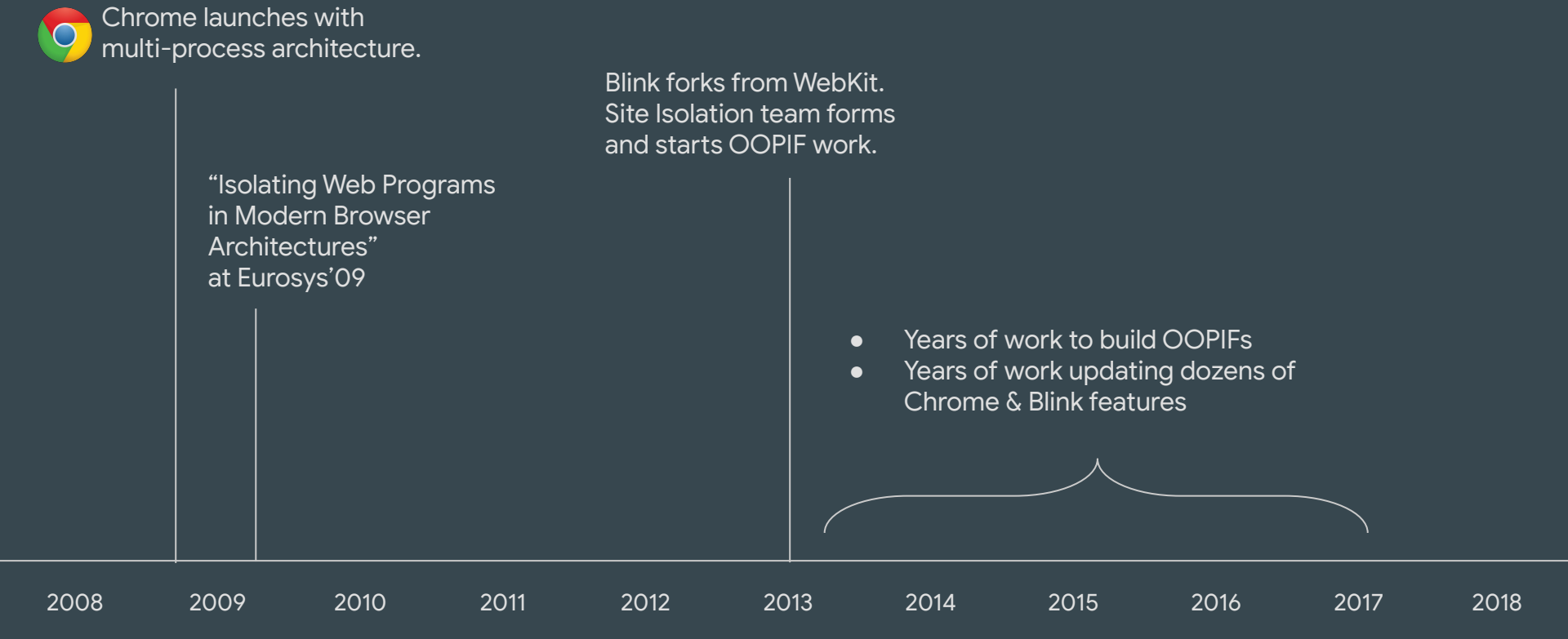

Browser Process
Web Renderer
evil.com
youtube.com
Sandbox



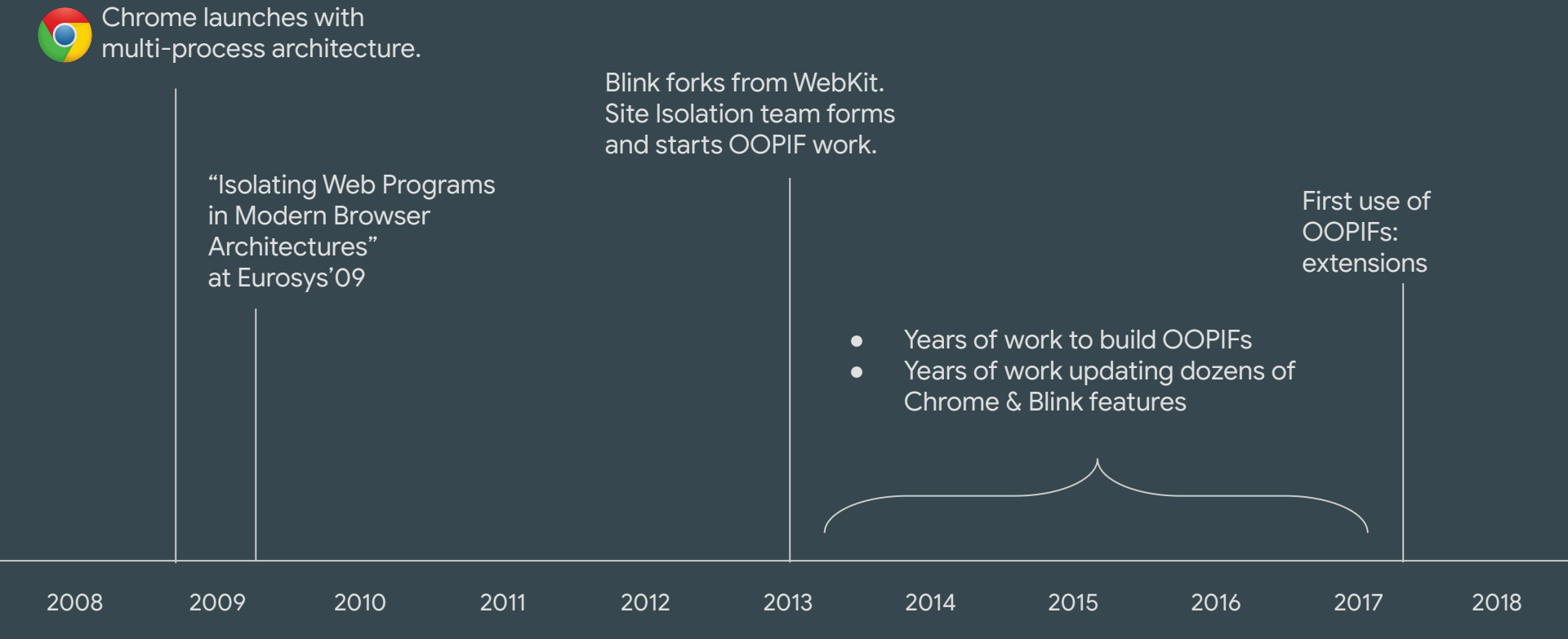
History of Site Isolation in Chrome



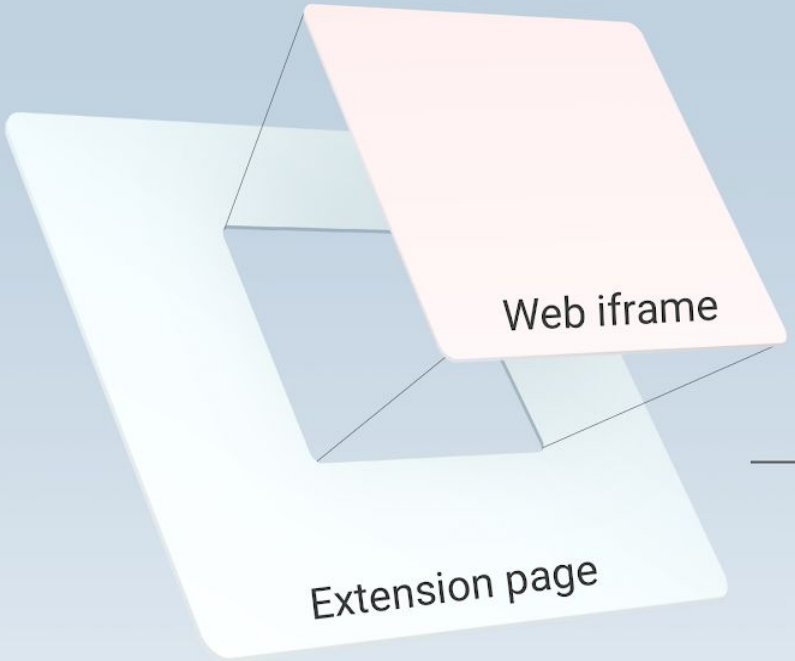
Out-of-process iframes
(OOPIFs)



History of Site Isolation in Chrome



History of Site Isolation in Chrome

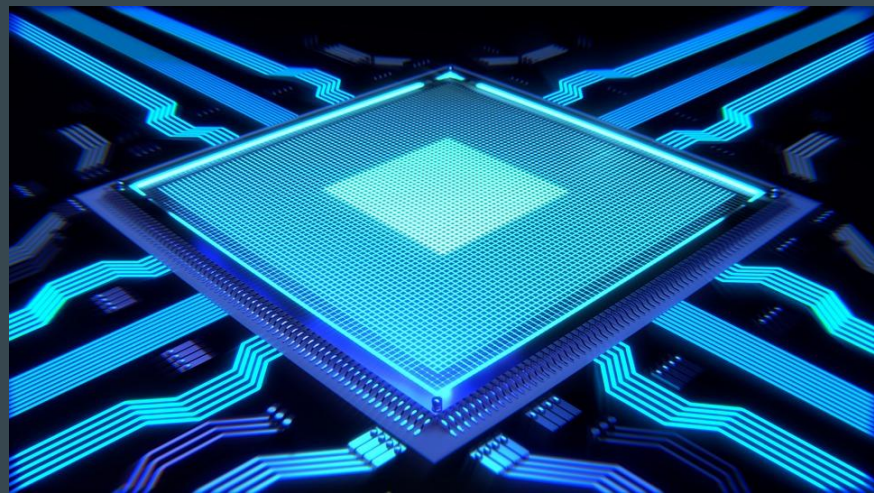


Web process

Extension process

Project Zero: Spectre and Meltdown

- June 2017: Jann Horn discovers flaw in almost all modern CPUs
  - Programs can read anything in their address space
- 6 months of mitigations across the industry, in secret



Speculative Execution + Side Channels

- Most CPUs keep running code while waiting for memory, etc.

```
if (x < array1.length) {
    y = array1[x];
}
```

Slow!

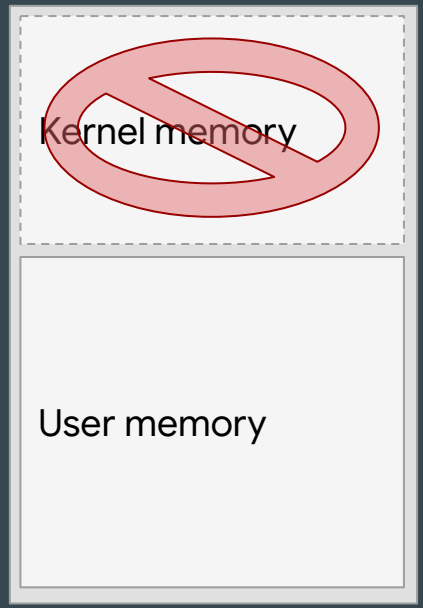
Run this in the meantime

- If prediction was wrong, just throw away results
- **But... those values stay in CPU cache and can be leaked**
  - Found way to leak them via cache timing attacks
  - Can basically read any memory address this way

Meltdown: Read kernel memory

- We used to put kernel memory in the same address space as user programs

- KPTI Mitigation
  - Don't put kernel memory in user space
  - Cost: Higher overhead for context switches



User process address space

Spectre: Read anything else in address space

- Matters for programs that try to confine untrusted code

- Example: **Web browsers**
  - Run untrusted JavaScript (compiled to native code)
  - Protect web site data using Same Origin Policy
  - Many sites end up in the same process

  - Risk that pages could read **cross-site pages, passwords, etc**

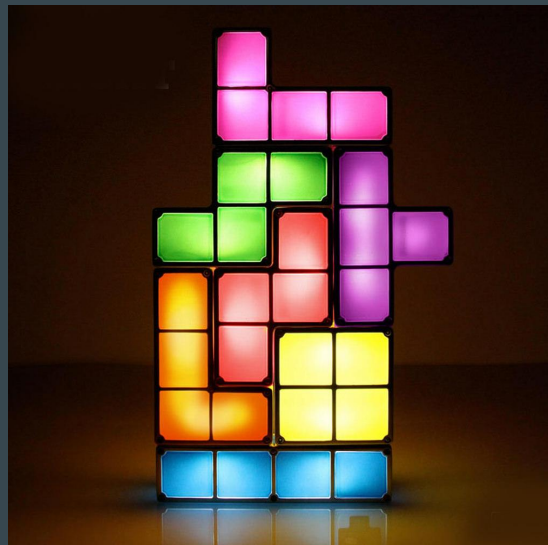
Browser Mitigations for Spectre

1. Make the timing attack harder

- Attack leaks data by measuring time to read from CPU cache
  - Generally depends on having precise timers
- All browsers added jitter to timers, disabled SharedArrayBuffer
- Not a great solution:
  - Not effective: can amplify signal with coarse timers
  - Web apps want precise timers

2. Make it harder to pull off speculation attacks

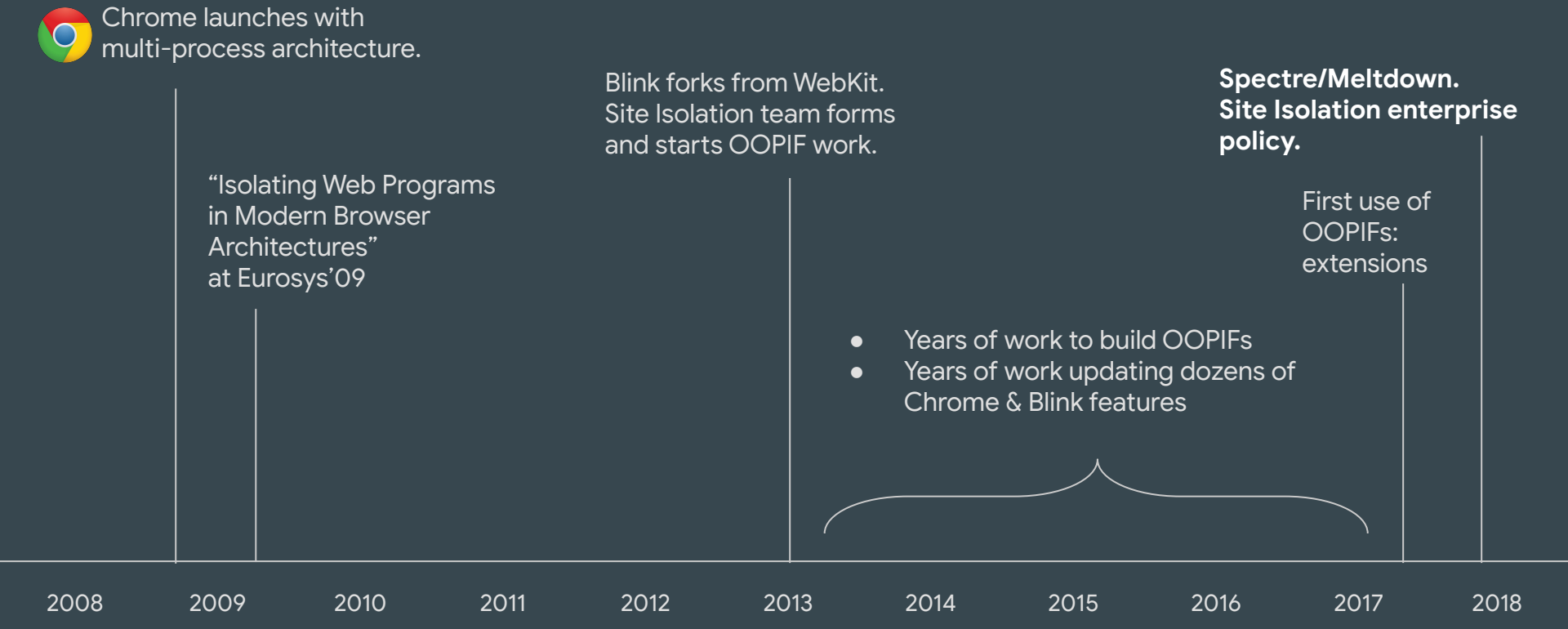
- Change JIT compiler: try not to emit "gadgets"
  - Avoid patterns of code that would let attacker access out-of-bounds data
- All browsers worked on these mitigations
- Still not a perfect answer
  - Adds overhead
  - Hard to address all variants



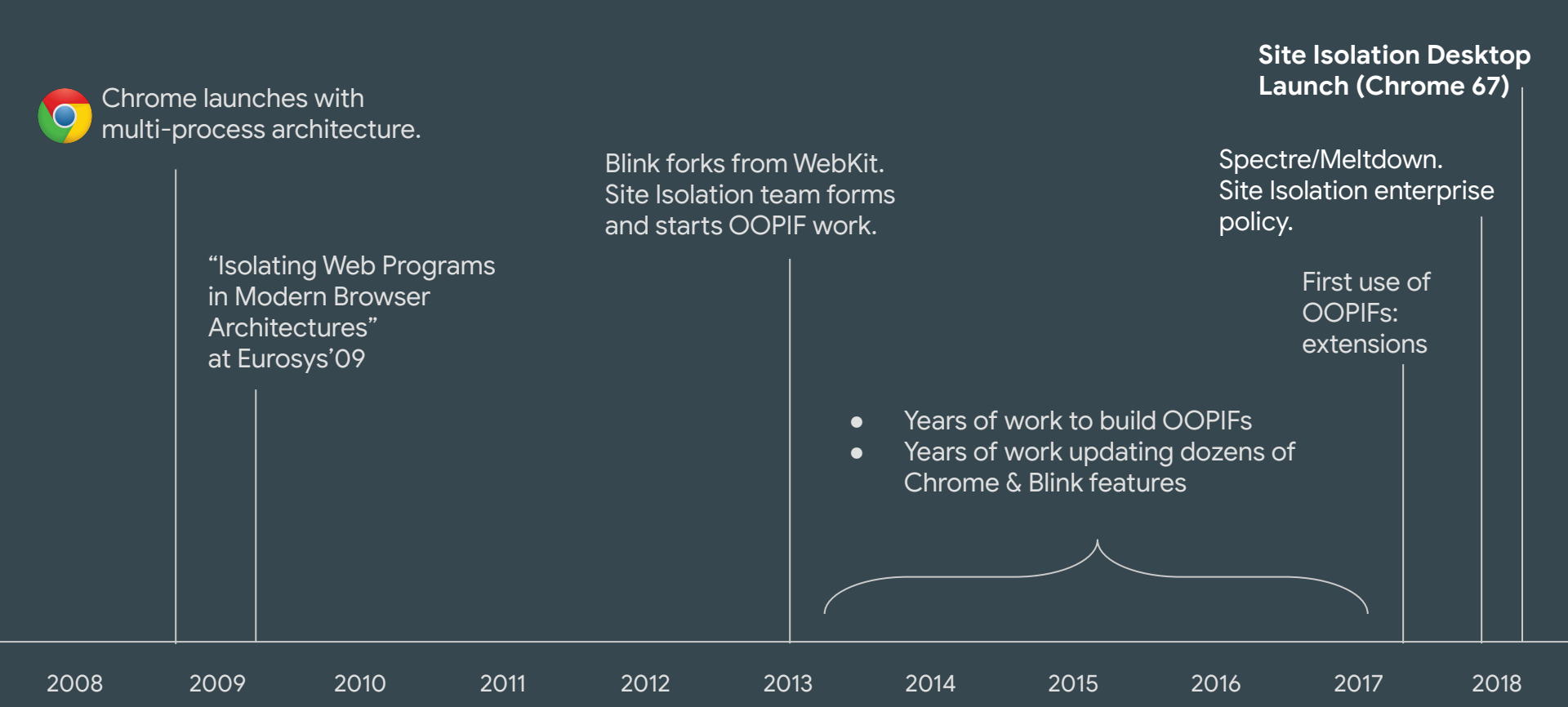
3. Avoid having data worth stealing in process

- Strategy: Assume attack will sometimes succeed
- Change browser so cross-site data isn't in the process

- **Site Isolation** was doing that anyway, for a different reason
  - Now it matters even for non-compromised processes!
  - Hurry up and get it ready...



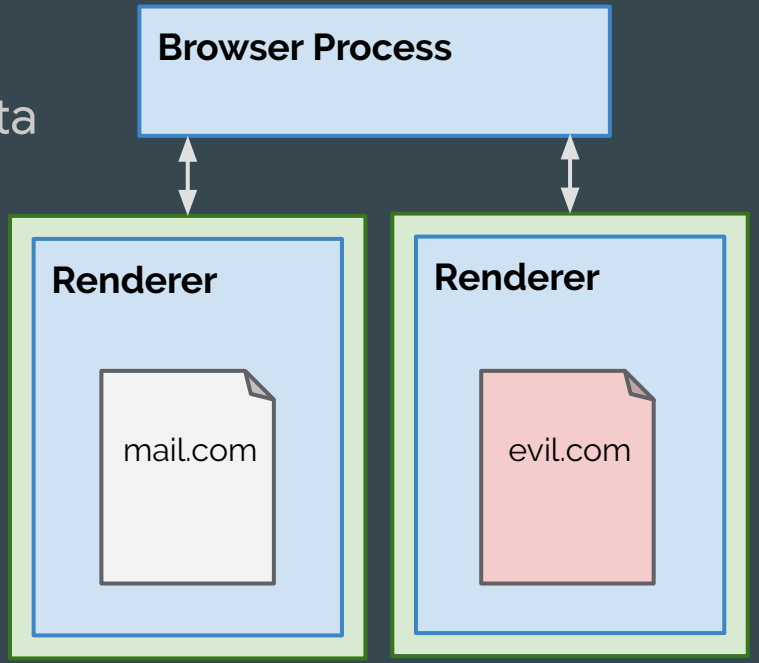
History of Site Isolation in Chrome



History of Site Isolation in Chrome

Dedicate processes to only one site each

- Don't let cross-site pages share a process
- Don't let processes request cross-site data



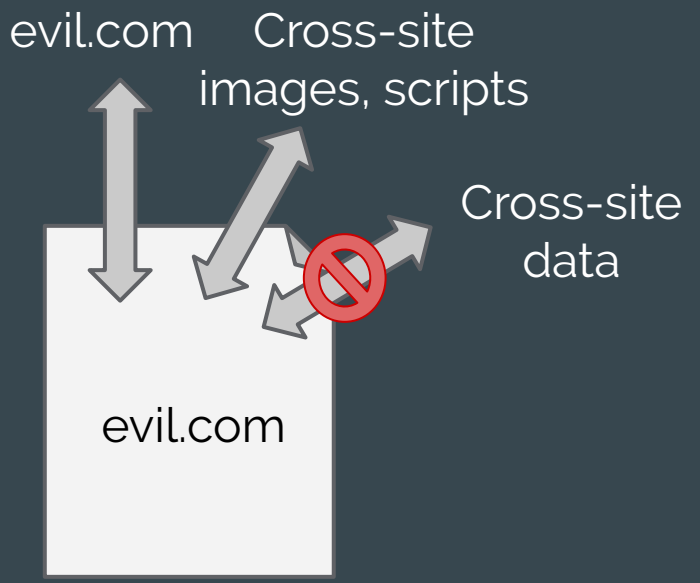
Cross-Origin Read Blocking (CORB)

- Attacker could still pull cross-site data into process:

```
<img src="https://bank.com/account.html">
```

- What cross-site requests to allow?
  - Must allow many subresources
  - Don't let HTML, XML, JSON through
  - Exceptions for CORS

- Now part of the spec



Chrome's Post-Spectre Threat Model

"We must assume that active web content
(JavaScript, WebAssembly, Native Client, Flash, PDFium, ...)
will be able to read any and all data
in the address space of the process that hosts it. "

https://chromium.googlesource.com/chromium/src/+/master/docs/security/side-channel-threat-model.md

Site Isolation: Next Steps

- Launch on Android
  - Planning to isolate a subset of sites, due to overhead concerns
- Work with other browsers on isolation techniques
  - Let some sites protect themselves (without OOPIFs)
  - Opt-in cross-browser headers
- Finish enforcements for compromised renderers
- Improve coverage (e.g., more types of data)

Conclusion

- Security landscape has changed
  - Hard to enforce boundaries within a process: *should assume you can't*
  - Still learning new variations

- Many mitigations needed across the industry
- When possible, align trust boundaries with process boundaries
  - **Site Isolation** lets the OS help with protecting web sites

- Long-term architecture work tends to be worth it

Q&A