Stopping XS-Leaks at scale

Deploying RIP and COOP at Google

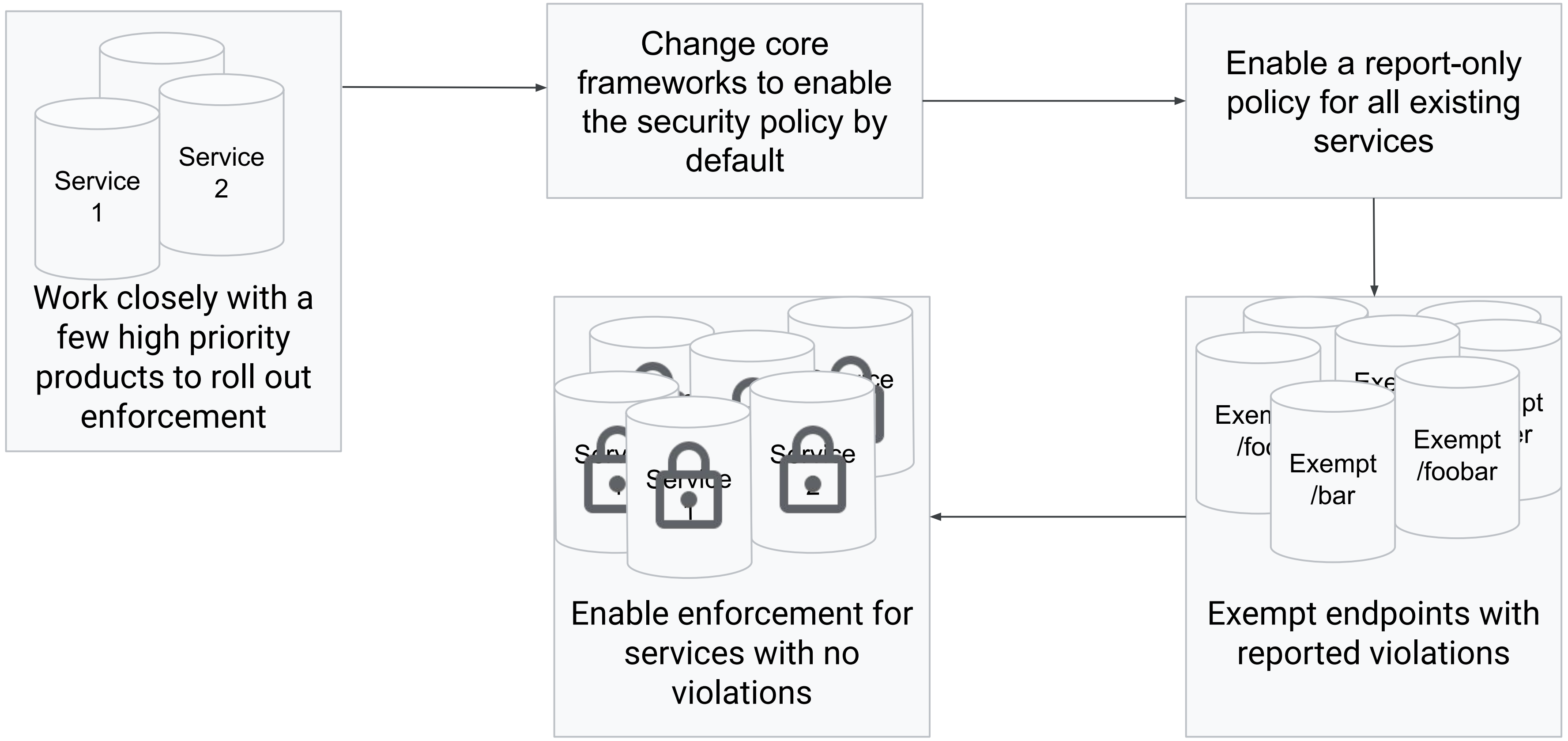
David Dworken

Background on rollouts at Google

- Challenges
    - Hundreds of web services all owned by different teams
    - Fast-paced environment with new services created daily
- Advantages
    - Monorepo, which enables:
        - One version policy: All services use the same version of web frameworks
        - Existing automation for splitting, testing, and sending changes to hundreds of teams[1]
    - All web traffic goes through a centralized load balancer where we can collect metrics

[1]: https://www.oreilly.com/library/view/software-engineering-at/9781492082781/ch22.html

Anatomy of a rollout at Google


Service 1
Service 2
Work closely with a few high priority products to roll out enforcement
Change core frameworks to enable the security policy by default
Enable a report-only policy for all existing services
Exempt /bar
Exempt /foobar
Exempt endpoints with reported violations
Service 1
Service 2
Enable enforcement for services with no violations

Resource Isolation Policy (RIP)

https://xsleaks.dev/docs/defenses/isolation-policies/resource-isolation/

- Summary: Reject cross-site requests based on Fetch Metadata headers
- Advantages
  - Purely server-side policy, so the report-only mode can detect all violations
- Challenges
  - The web is noisy!
  - Initially only supported by Chromium

Cross-Origin-Resource-Policy (CORP)

https://xsleaks.dev/docs/defenses/opt-in/corp/

- Summary: Tell browser to block cross-origin/cross-site loading of a resource
- At Google, we pair this with RIP:
  - If an endpoint enforces RIP, it automatically sets `CORP: same-site`
  - If an endpoint is exempted from RIP, it automatically sets `CORP: cross-origin`

Google

Cross-Origin-Opener-Policy (COOP)

https://xsleaks.dev/docs/defenses/opt-in/coop/

- Summary: Restrict cross-origin interactions with popups
- Advantages:
  - Has a report-only mode (Chrome only)
  - Much less noisy than RIP. Most of the time, a reported violation is a real problem, not just a weird client
- Challenges:
  - A lot…

COOP Reporting

- Originally didn’t have a report-only mode
- Added by Chrome in order to make COOP easier to roll out
- Two types of violation reports:
  - Navigation reports
  - Access reports
- Playground: https://coop-reporting-chrome-86.glitch.me/

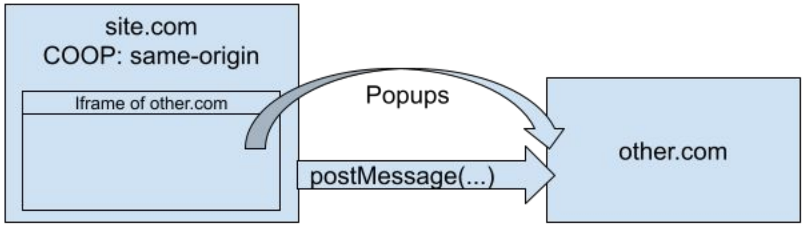
COOP Challenge: Reports are difficult to understand

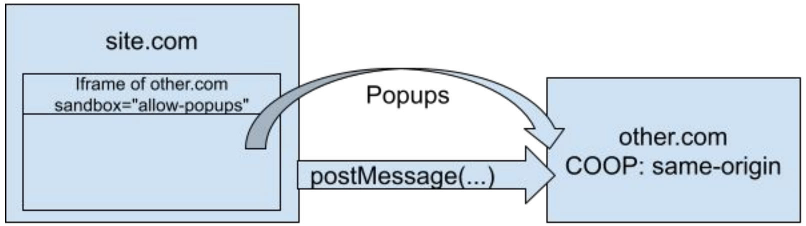
- Many different types of violation reports that are very easy to get confused
    - Access from opener v/s access from openee
- Not always clear what violation needs what policy to fix it
- “Solved” with extensive internal documentation

Access from opener
Access from openee
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.
imgflip.com

COOP Challenge: Reporting Gaps

- Certain violation scenarios don’t trigger COOP reports, but when COOP enforcement is enabled, things break
  - Unique compared to RIP/CSP/COEP

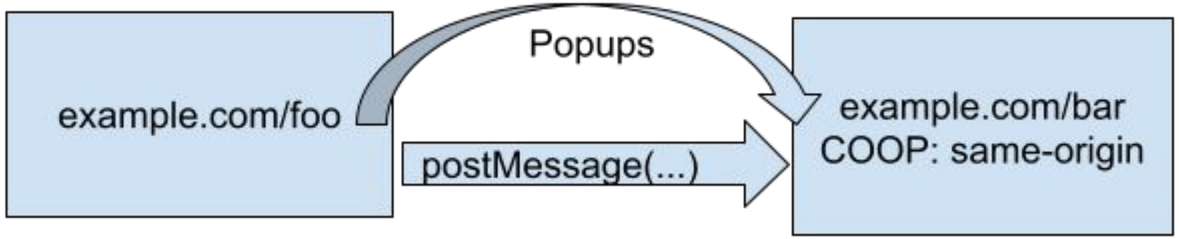




- When doing rollouts for hundreds of services, these scenarios are surprisingly common (~5% of services run into these gaps)
- All service owners are asked if they fall into any of these three cases before enabling enforcement

COOP Challenge: Rollouts

- COOP rollouts have to be done atomically for a given user
- Scenario:
    - Suppose that example.com is rolling out COOP. It has it enabled for a random sampling of 50% of requests.



    - example.com/foo and example.com/bar end up in different browsing context groups! So postMessage doesn't work

COOP Challenge: Atomic Rollouts

- Means that naively ramping up enforcement as a percent of requests doesn’t work!
- For authenticated services, we use experiments that bucket users based on their session
- For unauthenticated services, we atomically enable enforcement for all users at a specific time
    - Increased impact of any breakages 😨

Google

COOP Challenge: Client-side navigation

- Client-side navigation means that if /foo needs a COOP policy of same-origin-allow-popups, then every page that can client-side navigate to /foo also needs to set same-origin-allow-popups
    - Generally: If one endpoint needs to open a popup, the entire service needs to relax COOP
- We use client-side navigation heavily

COOP Challenge: COI Conflict

- Also conflicts with COI because sites that need cross-origin isolation (currently) can't set same-origin-allow-popups
  - Means that entire services (which may be large) have to decide between SharedArrayBuffer access and opening popups

imgflip.com

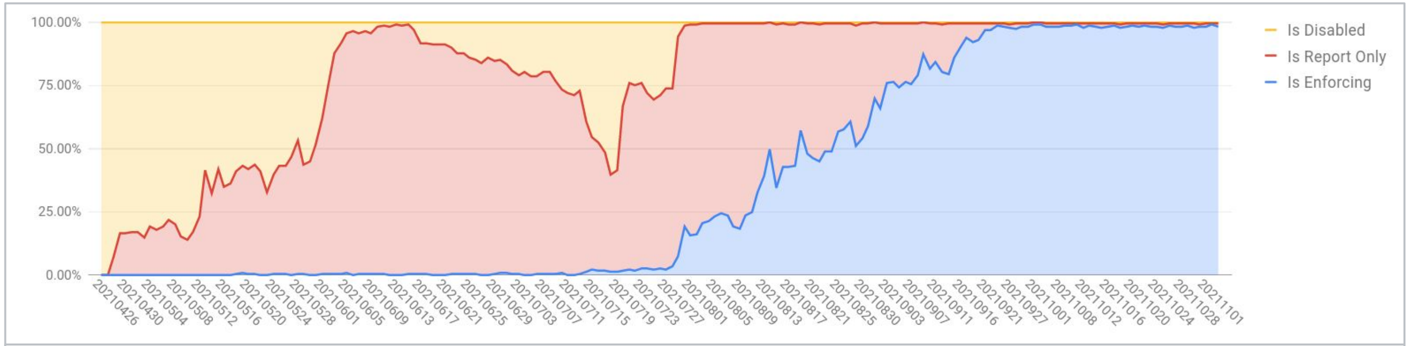
Google

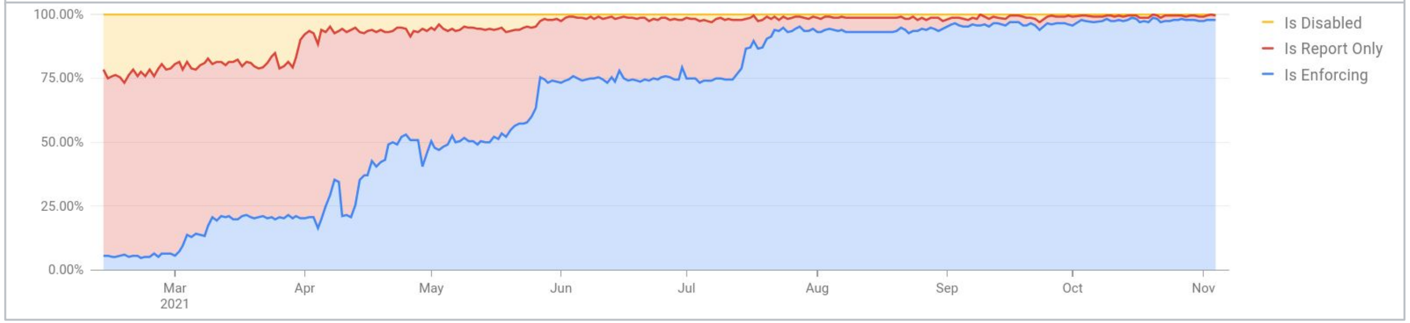
COOP Challenge: Very common violations

- Certain attribute accesses represent the vast majority of violations
  - These attributes often are not the ones most concerning from an XS-Leaks perspective
- Of the endpoints that need a COOP of `unsafe-none`:
  - 65% need `unsafe-none` because they need to expose the `closed` attribute
  - 20% are for endpoints that need to use `postMessage`
  - 4% are for endpoints that serve redirects
  - 2% are for endpoints that need to have `focus()` called on them
  - …
  - 0% are for endpoints that need to expose the `frames` attribute

Current Status

- ~150 billion requests a day enforcing COOP and RIP
- The two most popular web frameworks at Google enforce COOP and RIP by default



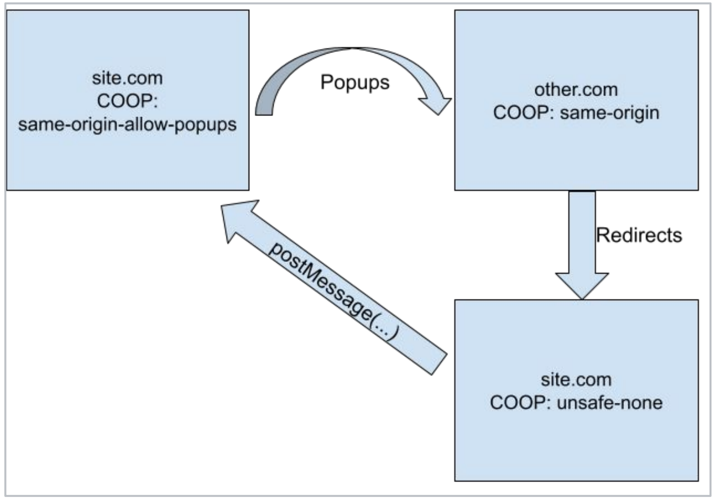


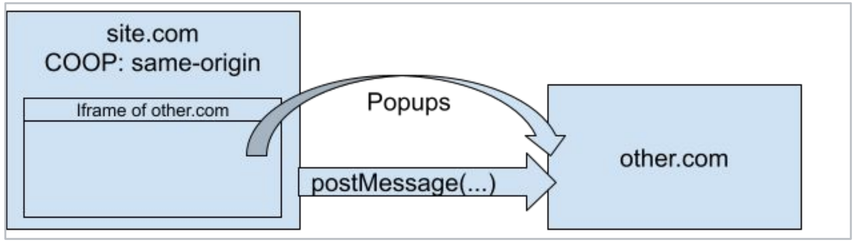
- Ongoing efforts to rollout COOP to other frameworks

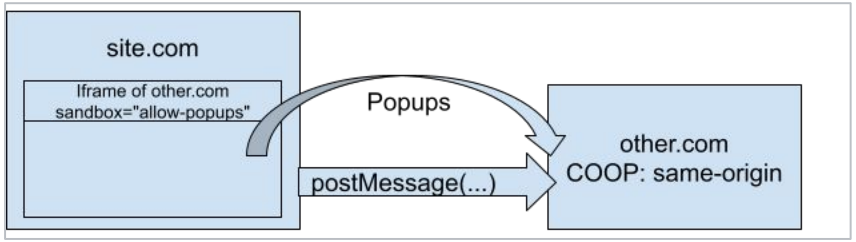
Can we make COOP rollouts easier?

- COOP Reporting Gaps
  - Same-origin policy prevents exposing that information, maybe we could expose a subset of it?
    - E.g. A report saying that an iframe opened a popup
- Could we have a \`same-origin-allow-popups-and-postMessage-and-closed\`
  - Or even better: \`same-origin-allow-popups postMessage closed …\`
  - Would make it possible to deploy COOP for pages that need to use postMessage
    - Currently our entire login flow can't deploy COOP since it needs to be opened in a popup and use postMessage

Appendix: COOP Reporting Gaps



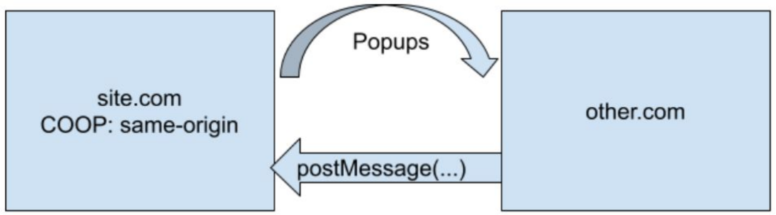




Appendix: Understanding COOP Reports

Access From Openee

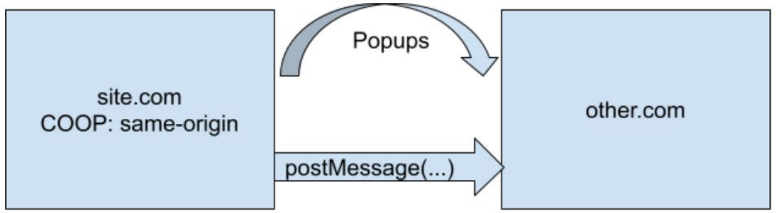
This violation type means that a COOP page opened a cross-origin page that tried to access a field on its opener, e.g. `window.opener.field`. One common way this can happen is if you open a popup that sends a message to your page.



This violation type can be fixed by setting `same-origin-allow-popups` on the COOP page that opened the window.

Access To Openee

This violation type means that a COOP page opened a cross-origin page and accessed a field of that window. For example, `window.open(other_site).field`. One common way this can happen is if you open a popup and send a message to it via `postMessage`.

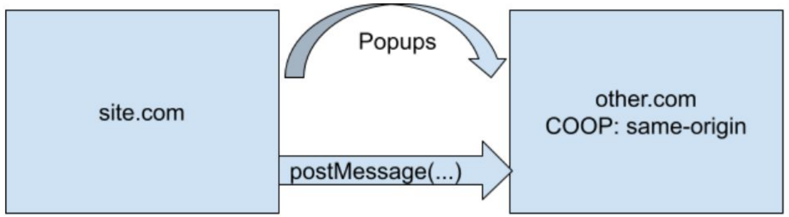


This violation type can be fixed by setting `same-origin-allow-popups` on the COOP page that opened the window.

Appendix: Understanding COOP Reports

Access From Opener

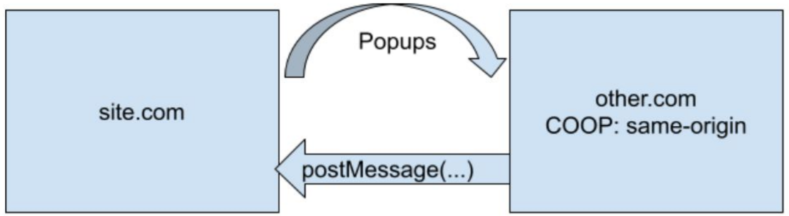
This violation type means that a page opened a cross-origin COOP page and tried to access a field on its opener, e.g. `window.open(other_site_with_coop).field`. One common way this can happen is if you open a popup and send a message to it via postMessage.



This violation type can be fixed by setting `unsafe-none` on the page that is being opened.

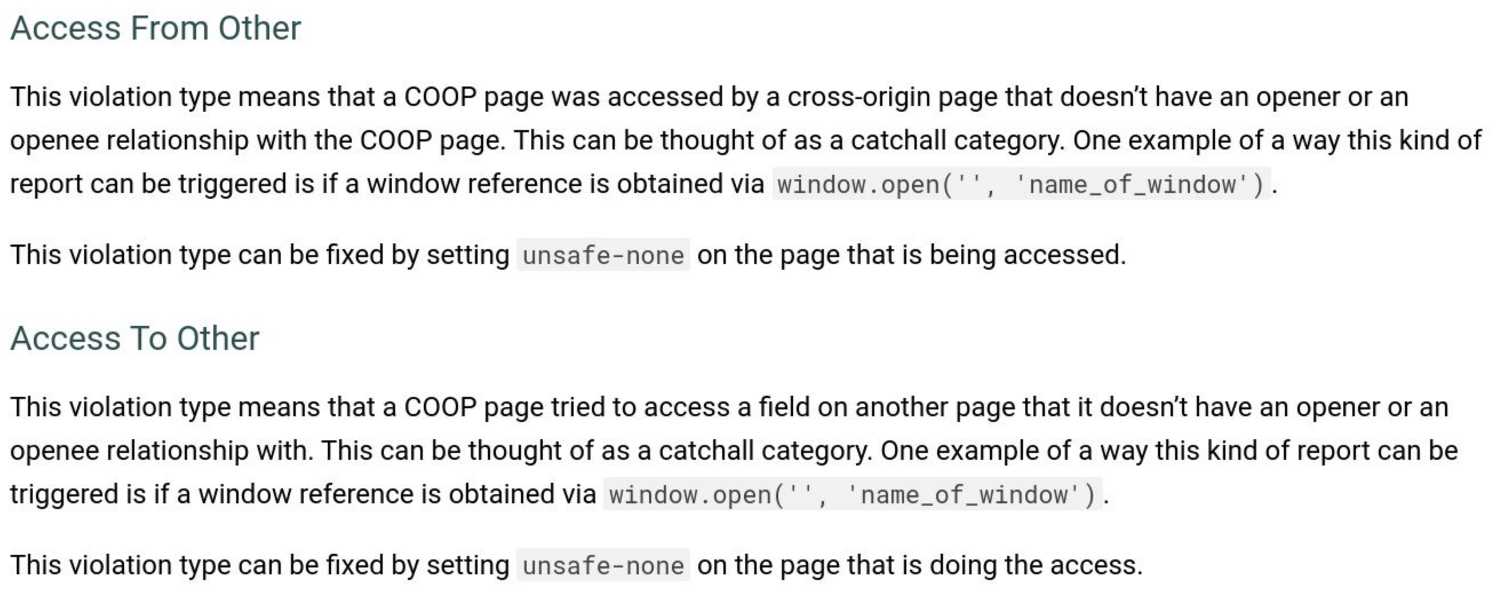
Access To Opener

This violation type means that a page opened a cross-origin COOP page that did `window.opener.field`. One common way this can happen is if you open a popup that sends a message to your page.



This violation type can be fixed by setting `unsafe-none` on the page that is being opened.

Appendix: Understanding COOP Reports



Access From Other

This violation type means that a COOP page was accessed by a cross-origin page that doesn't have an opener or an openee relationship with the COOP page. This can be thought of as a catchall category. One example of a way this kind of report can be triggered is if a window reference is obtained via `window.open('', 'name_of_window')`.

This violation type can be fixed by setting `unsafe-none` on the page that is being accessed.

Access To Other

This violation type means that a COOP page tried to access a field on another page that it doesn't have an opener or an openee relationship with. This can be thought of as a catchall category. One example of a way this kind of report can be triggered is if a window reference is obtained via `window.open('', 'name_of_window')`.

This violation type can be fixed by setting `unsafe-none` on the page that is doing the access.