THE UNIVERSITY OF ILLINOIS AT CHICAGO

UIC

Awakening the Web's Sleeper Agents: Misusing Service Workers for Privacy Leakage

Soroush Karami, Panagiotis Ilia, Jason Polakis

University of Illinois at Chicago, USA

skaram5@uic.edu

NDSS SYMPOSIUM/2021

February 2021

abstract

- What are service workers?
- A measurement study on Service Workers
- A Security issue on service workers
- Novel attacks vectors

Service workers

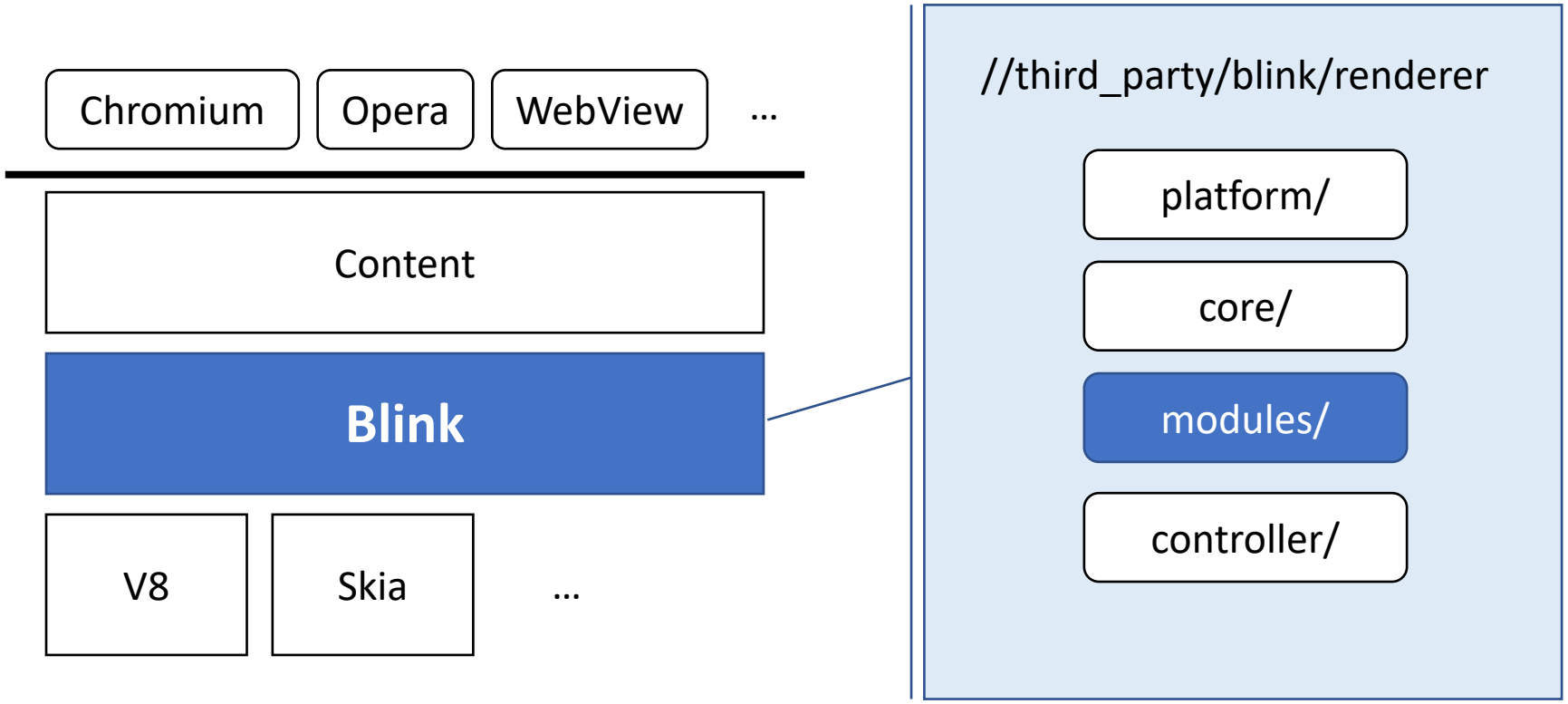
A new powerful technology

Service workers run independently of the web application

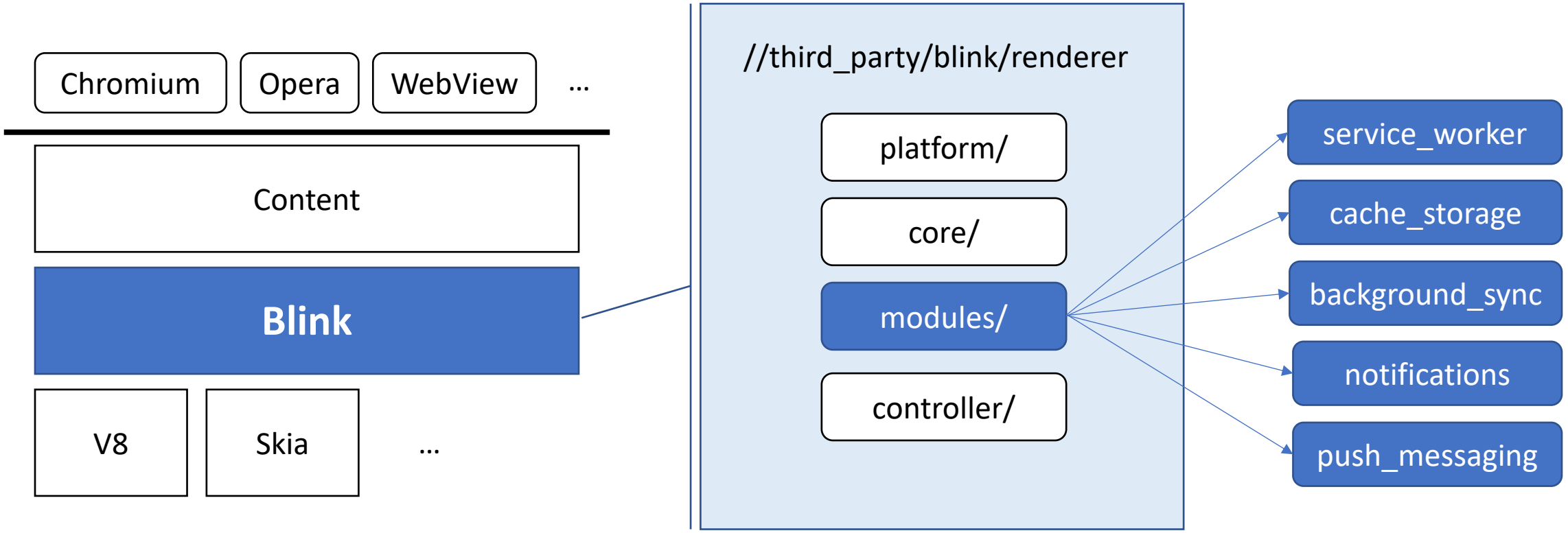
Fill the gap between native and web apps

- Push notifications functionality
- Syncing in the background
- Pre-caching for optimization
- Working offline

Chromium instrumentation

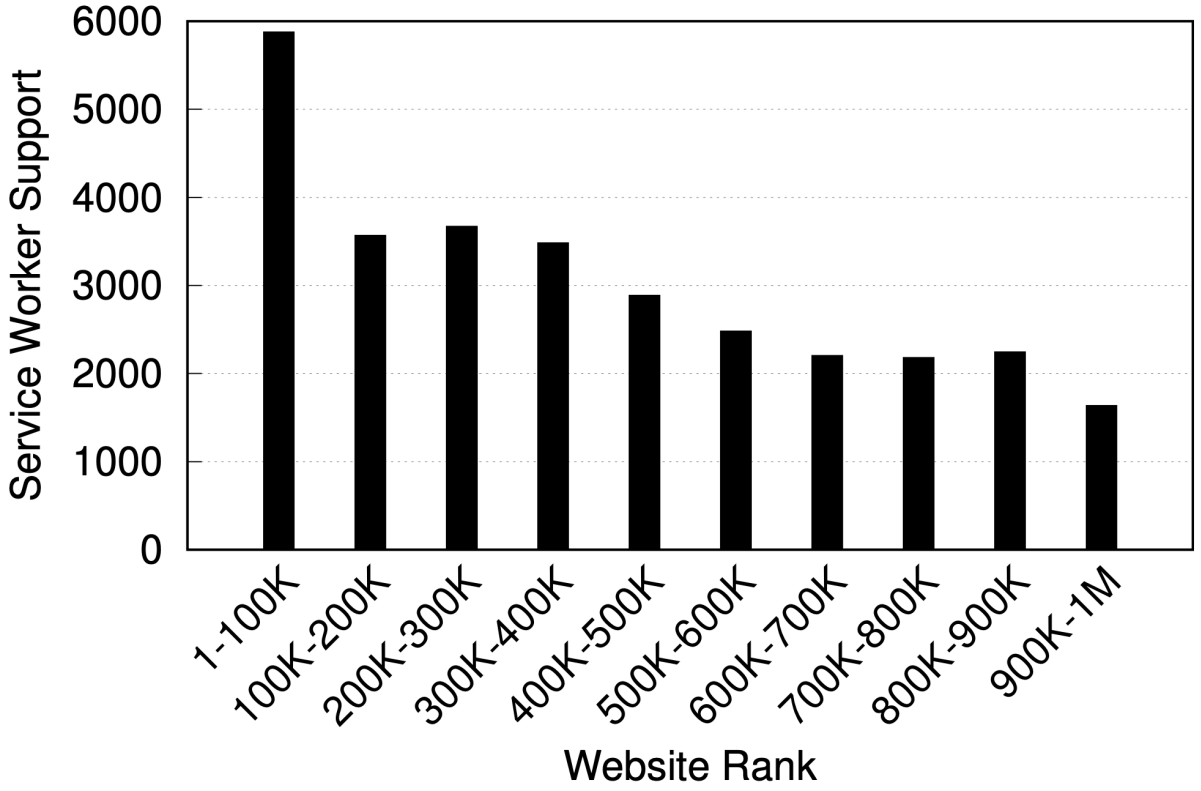


Chromium instrumentation



Service workers in the wild

Among top 1M Alexa websites we identify SWs on 30,229 sites



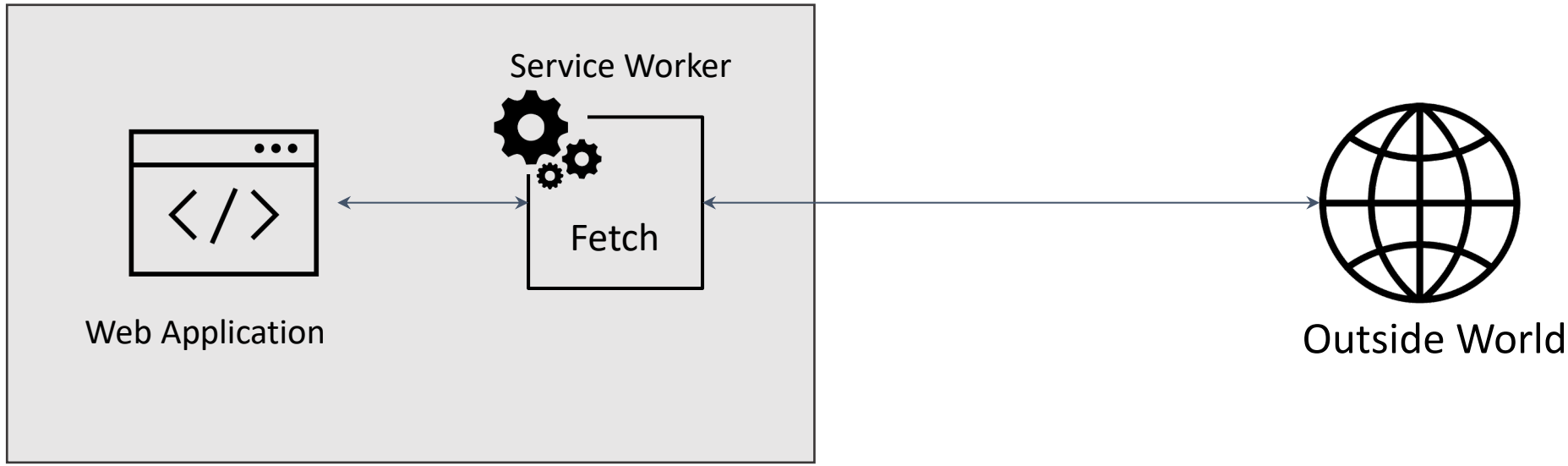
Provided functionality

Functionality	Domain
Caching	8,559
Fetch	8,895
Web push	23,227
Sync	90
SW to Client Message	8,339
Client to SW Message	10,593

Service workers - Provided functionality

Functionality	Domain
Caching	8,559
Fetch	8,895
Web push	23,227
Sync	90
SW to Client Message	8,339
Client to SW Message	10,593

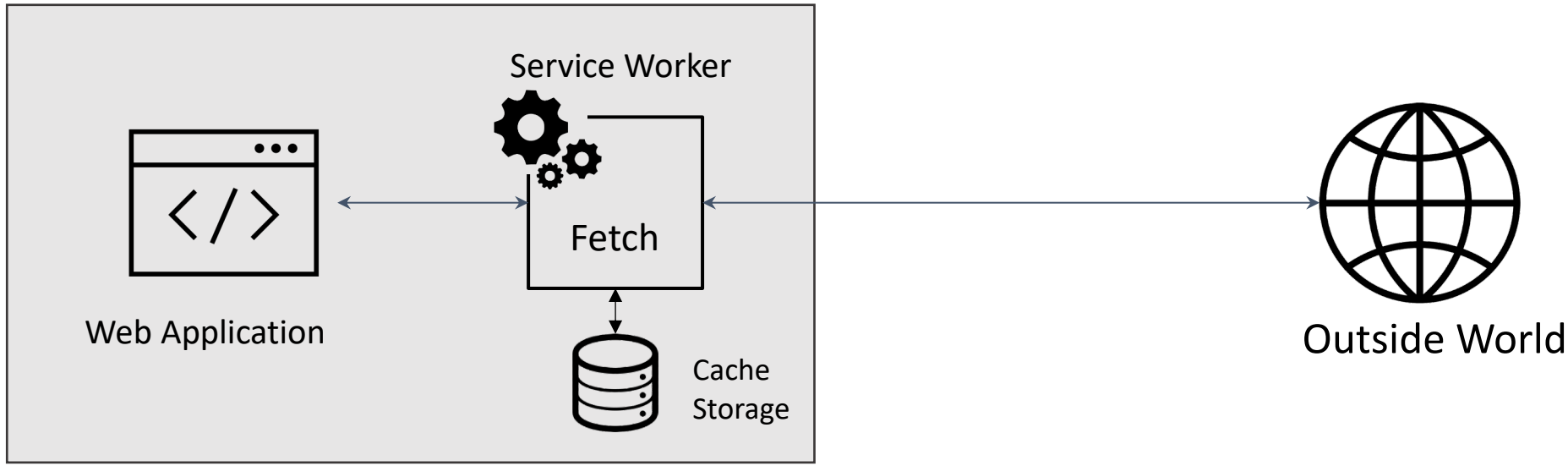
Feature: Fetch API



A **programmable** client side proxy

Intercepts requests from pages inside the SW's scope

Feature: Fetch API



Use cases:

- Controlling the caching behavior
- Providing offline pages

Browser Cache (HTTP cache) **VS.** Service Worker Cache Storage

1. Populating the cache
    - Browser Cache: during navigation of websites
    - SW Cache Storage: A programmable cache

```
this.addEventListener('install', function(event) {
    event.waitUntil(
        caches.open('v1').then(function(cache) {
            return cache.addAll([
                'index.html',
                'offline.html',
                'static/style.css',
                'static/app.js',
                'images/logo.jpg',
                'images/icon.png'
            ]);
        }));
});
```

Browser Cache (HTTP cache) **vs.** Service Worker Cache Storage

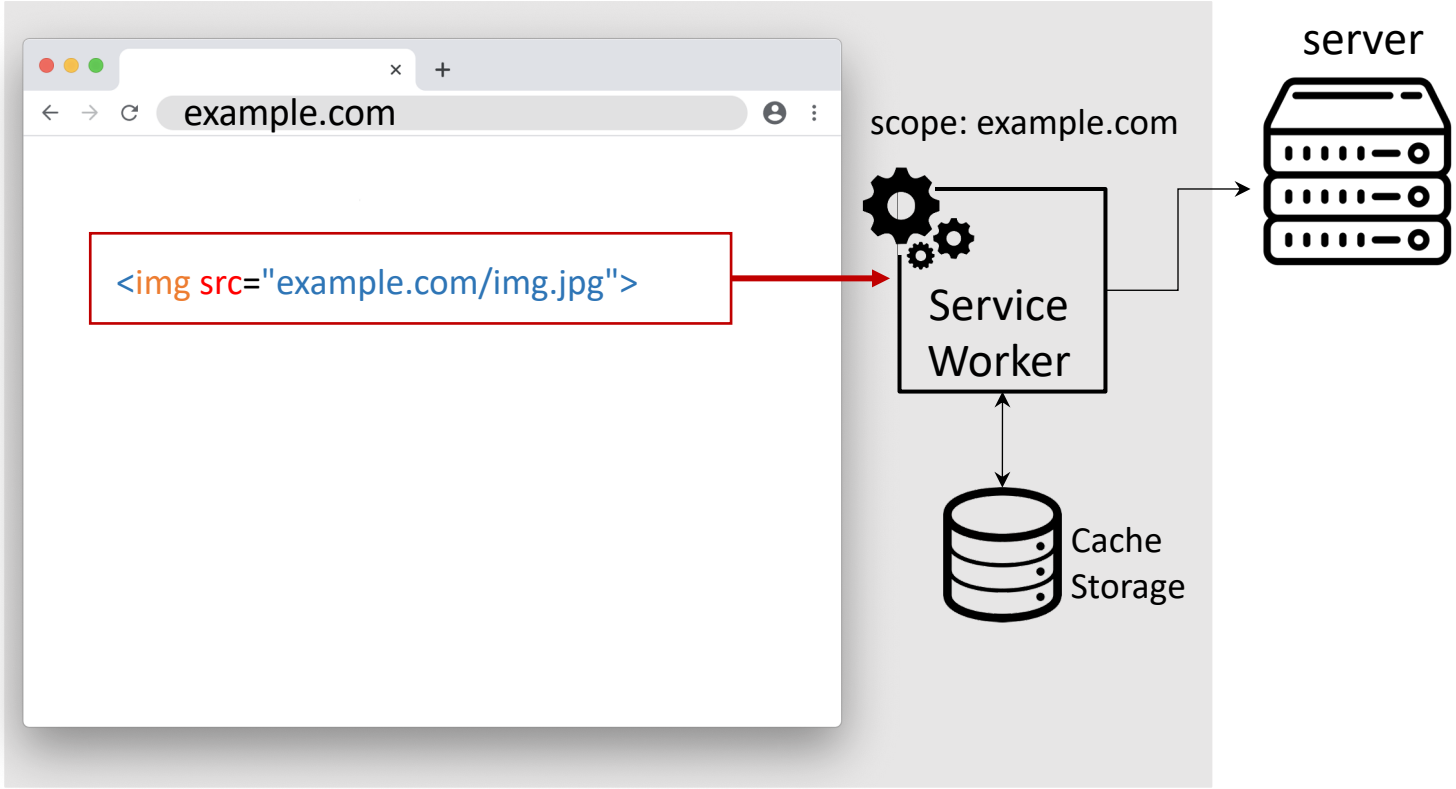
2. Managing the cached resources

- Browser Cache:
  - HTTP headers

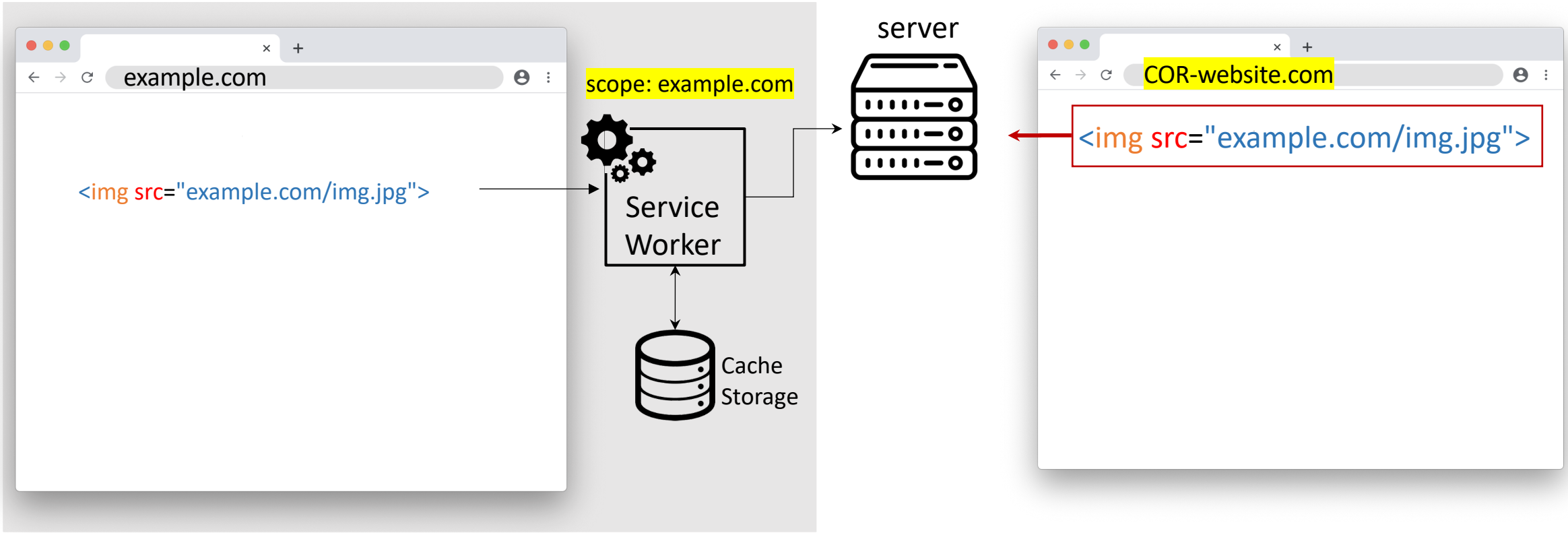
```
Last-Modified: Mon, 08 Sep 2020 19:23:51 GMT
ETag: "5485fac7-ae74"
Cache-Control: max-age=533280
Expires: Sun, 10 Oct 2020 23:02:37 GMT
```

  - Browser's built-in heuristics
- SW Cache Storage: A code-driven approach
  - Resources will persist until SW code explicitly removes them
  - No automatic, built-in expiration algorithms or freshness checks

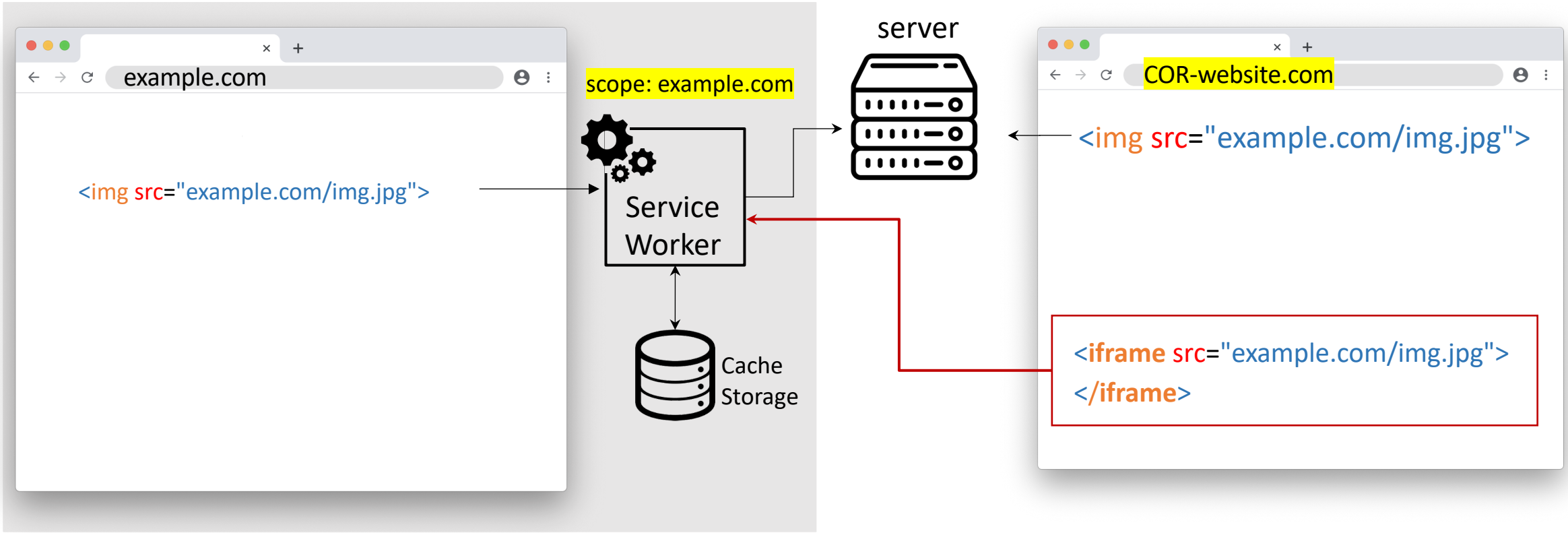
Navigating an in-scope page



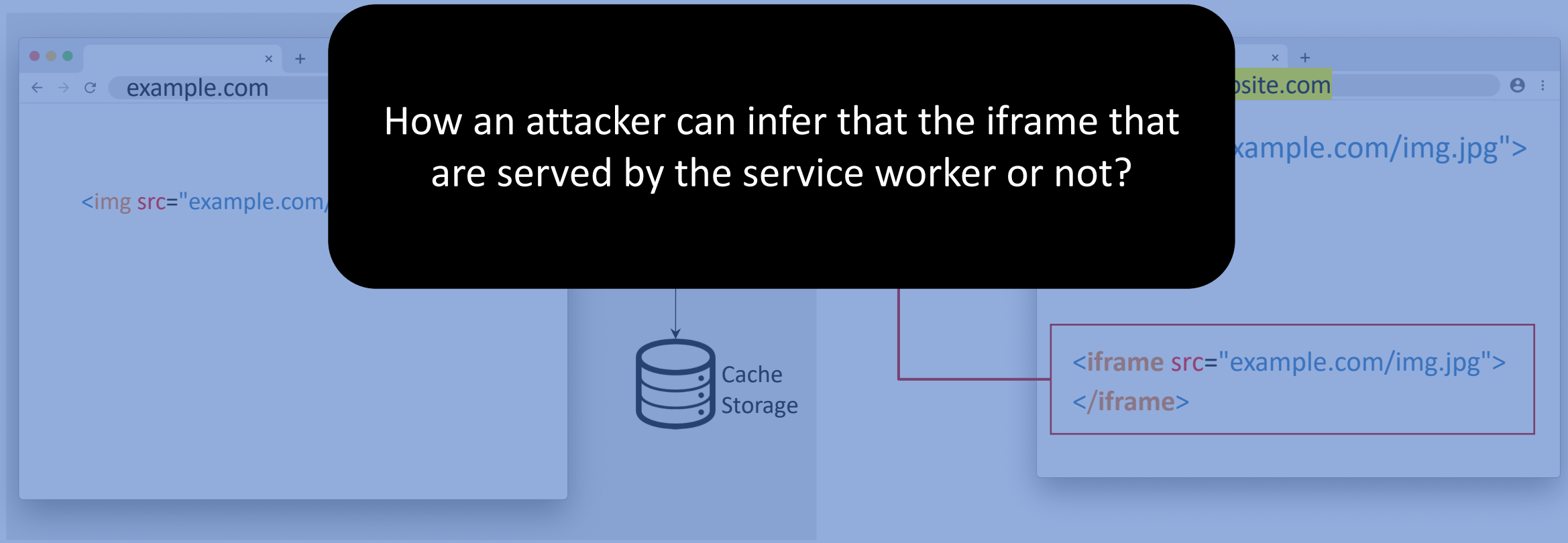
Navigating an out-of-scope page



Activation by an out-of-scope page

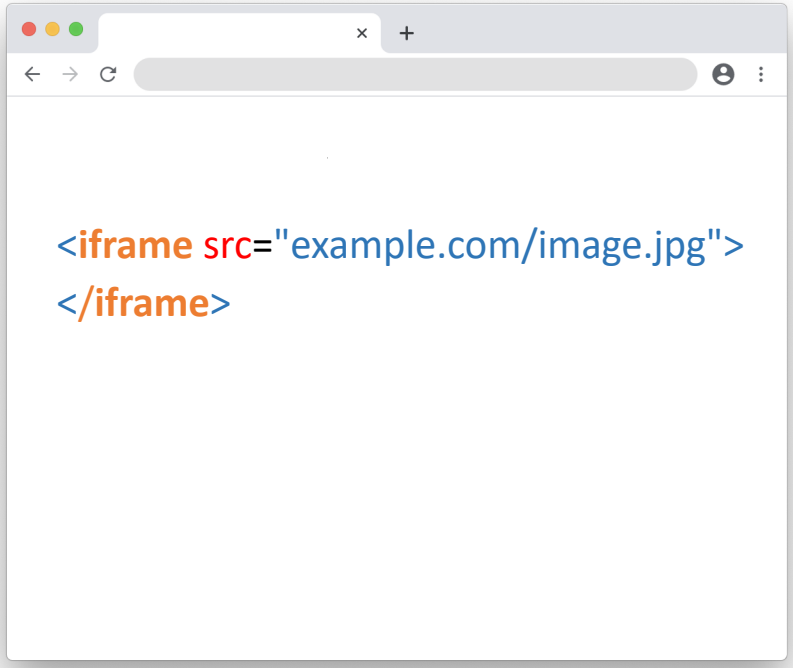


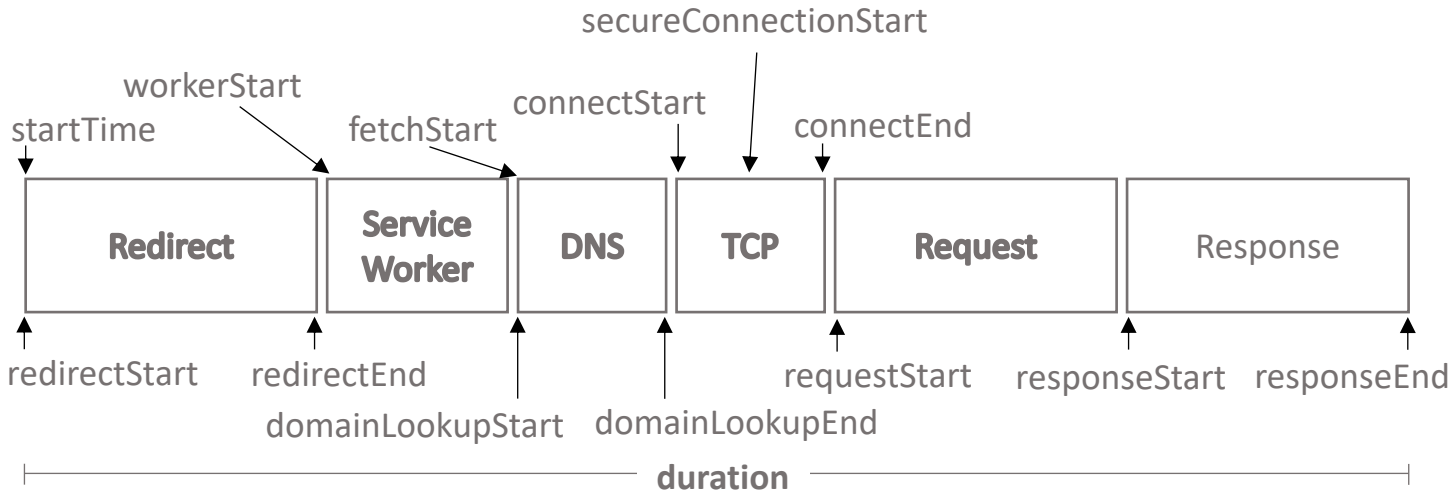
Activation by an out-of-scope page



Performance API

Provides detailed timing data regarding the loading of a website's resources

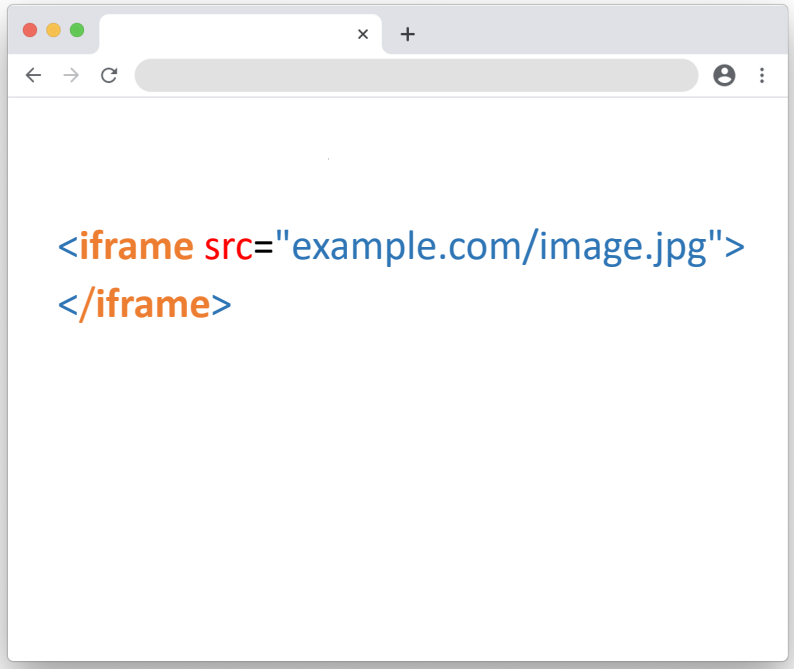




Performance API

Provides detailed timing data regarding the loading of a website's resources

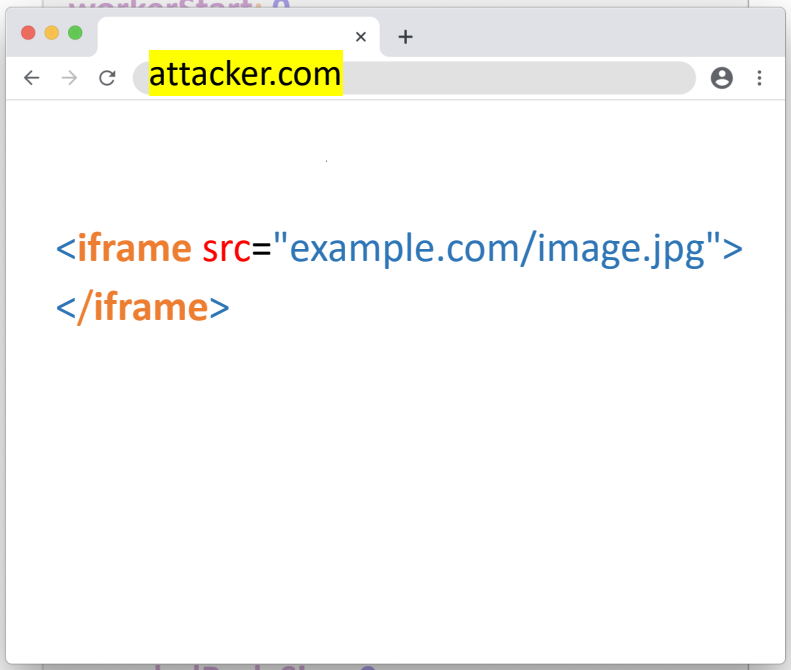
```
performance.getEntriesByName('https://example.com/image.jpg')
```



initiatorType: "iframe"
nextHopProtocol: ""
workerStart: 4849.369999952614
redirectStart: 0
redirectEnd: 0
fetchStart: 4849.459999939427
domainLookupStart: 4849.459999939427
domainLookupEnd: 4849.459999939427
connectStart: 4849.459999939427
connectEnd: 4849.459999939427
secureConnectionStart: 0
requestStart: 4849.384999950416
responseStart: 4853.985000052489
responseEnd: 4865.110000008717
transferSize: 0
encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://example.com/image.jpg"
entryType: "resource"
startTime: 4849.225000012666
duration: 15.884999996051192

Cross Origin without SW

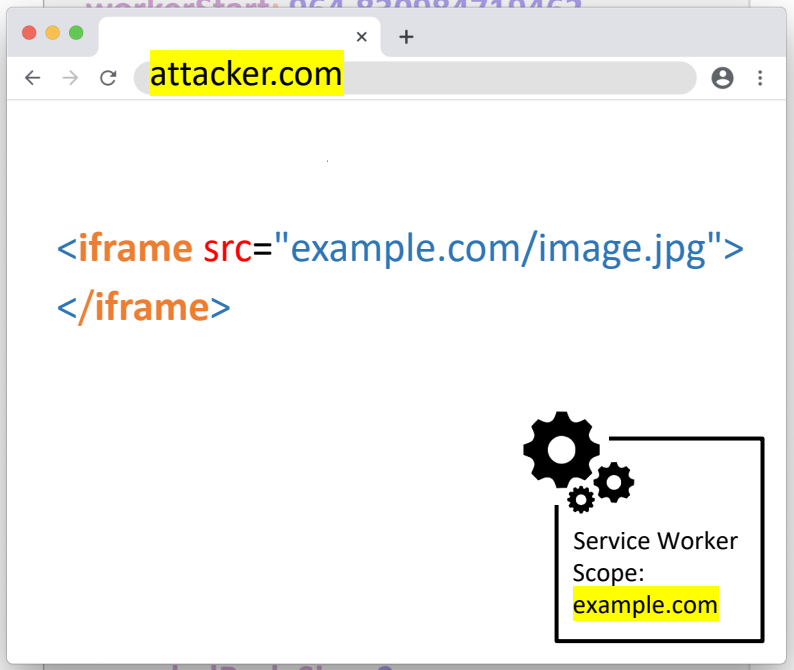
initiatorType: "iframe"
nextHopProtocol: "h2"



encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 39135.019810008719
duration: 269.8969009073553

Cross Origin with SW

initiatorType: "iframe"
nextHopProtocol: ""



encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 54815.060000051744
duration: 17.100000055506825

Cross Origin without SW

initiatorType: "iframe"
nextHopProtocol: "h2"
workerStart: 0
redirectStart: 0
redirectEnd: 0
fetchStart: 39142.29782008391
domainLookupStart: 0
domainLookupEnd: 0
connectStart: 0
connectEnd: 0
secureConnectionStart: 0
requestStart: 0
responseStart: 0
responseEnd: 39404.91671091608
transferSize: 0
encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 39135.019810008719
duration: 269.8969009073553

Cross Origin with SW

initiatorType: "iframe"
nextHopProtocol: ""
workerStart: 964.830984719462
redirectStart: 0
redirectEnd: 0
fetchStart: 54824.54000005964
domainLookupStart: 0
domainLookupEnd: 0
connectStart: 0
connectEnd: 0
secureConnectionStart: 0
requestStart: 0
responseStart: 0
responseEnd: 54832.16000010725
transferSize: 0
encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 54815.060000051744
duration: 17.100000055506825

Cross Origin without SW

initiatorType: "iframe"
nextHopProtocol: "h2"
workerStart: 0
redirectStart: 0
redirectEnd: 0
fetchStart: 39142.29782008391
domainLookupStart: 0
domainLookupEnd: 0
connectStart: 0
connectEnd: 0
secureConnectionStart: 0
requestStart: 0
encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 39135.019810008719
duration: 269.8969009073553

Cross Origin with SW

initiatorType: "iframe"
nextHopProtocol: ""
workerStart: 964.830984719462
redirectStart: 0
redirectEnd: 0
fetchStart: 54824.54000005964
domainLookupStart: 0
domainLookupEnd: 0
connectStart: 0
connectEnd: 0
secureConnectionStart: 0
requestStart: 0
encodedBodySize: 0
decodedBodySize: 0
serverTiming: []
name: "https://a.com/img.jpg"
entryType: "resource"
startTime: 54815.060000051744
duration: 17.100000055506825

The `workerStart` and `nextHopProtocol` attributes can be used for inferring if a resource was fetched through the SW.

Resource profiling

- Instrumented Chromium browser (version 79):
  - Log URLs of resources fetched through SW's FetchEvent
  - Log URLs of resources stored in the website's cache storage
- Use Selenium to launch our instrumented browser - visit a website
  - Installs a SW during the visit
- Log URLs of all resources (and filter out 3rd-party resources)
- Visit our own website that uses *iframes* to load these resources
  - Inspect value of `workerStart` and `nextHopProtocol` attributes

Privacy-invasive Attacks

- Registration inference
- Application-level inference
- Fine-grained history sniffing

1. Registration inference

- Websites insert additional resources into their cache **after login**
- Examples:
  - **Tinder** - a popular dating application/website
  - **Gab** - a social networking website that attracts “alt-right users, conspiracy theorists, and trolls, and high volumes of hate speech” [Zannettou et al., WWW ‘18]

Our attacks reveal not only that the user has visited a website at some point, but that they also have an **account** on that service

2. Application-level inference

- Example: web application of **WhatsApp** (https://web.whatsapp.com/)
  - Attacker can (partially) reconstruct the victim's **social graph**
  - Attacker can infer **group memberships**
  - SW stores in the cache photos of the victim's contacts and groups
    - `web.whatsapp.com/pp?t=s&u=<phonenumber>&i=<timestamp>`

It reveals that particular individuals are among the **victim's contacts,** or that the victim is a **member** in specific **groups**

3. Fine-grained history sniffing

- Some websites store additional resources when the user navigates different pages on that domain

- Example: https://spokeo.com
  - Aggregates information about people and allows to search
    - stores all user's search queries into the cache storage
    - allowing an attacker to infer whether the victim has searched for specific individuals

Provides fine-grained information about the **navigation** of the user within the visited website

Vulnerable Browsers

- Safari is not vulnerable to our attacks
  - it installs new SW for iframes

- Chrome has fixed the performance API issues
  - **workerStart** issue in version 80
  - **nextHopProtocol** in version 83

Browser	Version	Performance API		Timing
		`workerStart`	`nextHopProtocol`	
Firefox	72.0.2	●	●	●
Brave	1.3	○	●	●
Chrome	79	●	●	●
Edge	79	●	●	●
Opera	66	●	●	●
Safari	12.1.2	○	○	○

Attack Mitigation

- Root Cause: improper isolation of Service Workers in browsers
- Our solution: implementing access control logic inside Service Workers

```
self.addEventListener( 'fetch', function(event){
        referrer = (new URL(event.request.referrer)).host;
        if(referrer==self.location.hostname || referrer.match()!=null){
            /*Remaining SW functionality goes here*/
        }
});
```

Conclusion

- Conducted a large-scale measurement on Service Workers
  - At least 30,000 websites currently use Service Workers
  - At least 6% of the top 100K websites
- Service Worker isolation issue
- Privacy-invasive attacks
  - Registration inference
  - Application-level inference
  - Fine-grained history sniffing
- We disclosed our findings to affected vendors
  - Facebook fixed the issue
  - Chromium fixed the performance API issue and explores redesigning of its site isolation mechanism

UIC

Questions?

Feel free to contact me:
skaram5@uic.edu