In the DOM We Trust: Exploring the Hidden Dangers of Reading from the DOM on the Web

Jan Drescher[‡], **Sepehr Mirzaei**[*], Soheil Khodayari[*], David Klein[‡], Thomas Barber[Φ], Martin Johns[‡], Giancarlo Pellegrino[*]

[*]CISPA - Helmholtz Center for Information Security

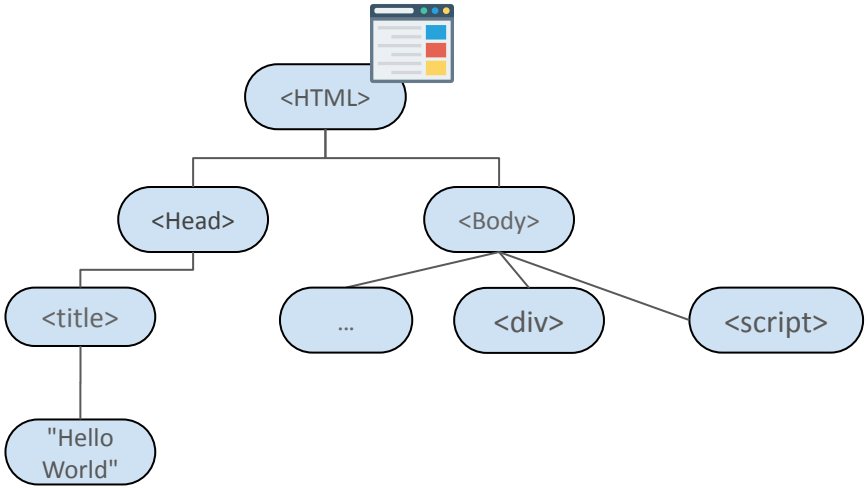
[‡]Technical University of Braunschweig

[Φ]SAP Security Research

CCS '25, October 13–17, 2025

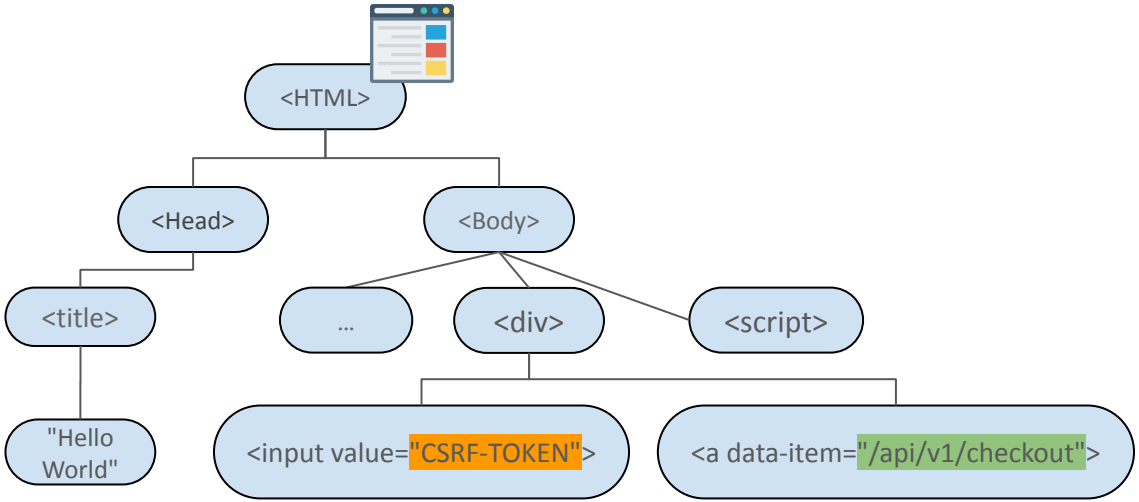
The Story of Webpages: From Dynamic Interfaces to Data Repos

- Web apps today are getting increasingly **dynamic**
  - Frequently update webpage content via JavaScript
  - Manipulate the **Document Object Model (DOM),** a tree-like page representation



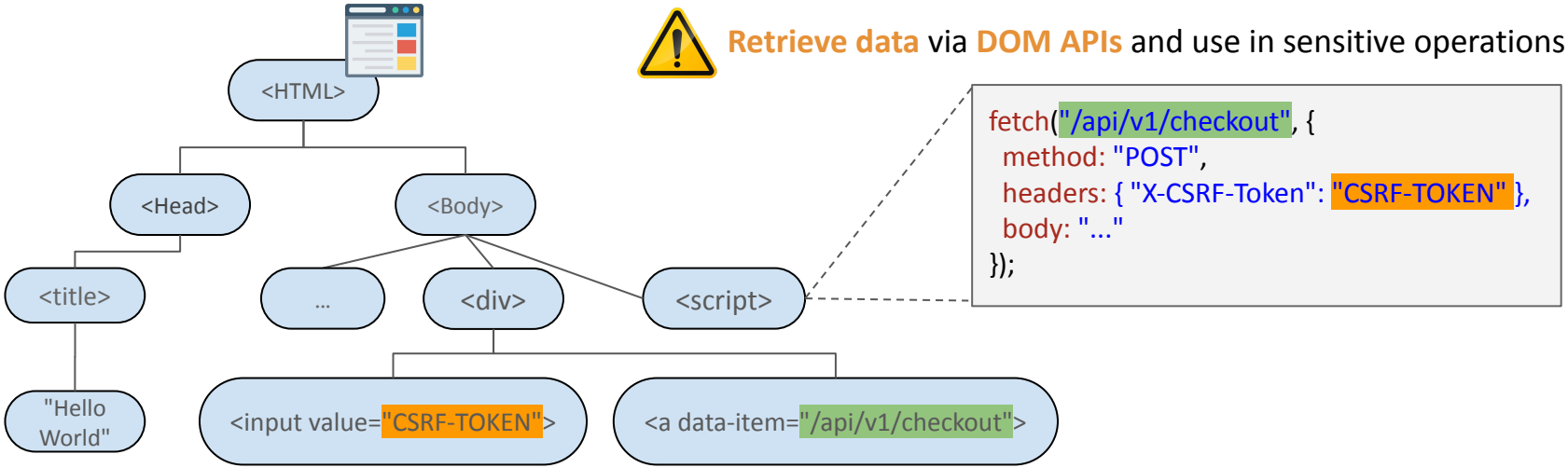
The Story of Webpages: From Dynamic Interfaces to Data Repos

- Web apps today are getting increasingly **dynamic**
  - Frequently update webpage content via JavaScript
  - Manipulate the **Document Object Model (DOM),** a tree-like page representation
- More recently, DOM used to also store **sensitive data** (e.g., one-time tokens)



The Story of Webpages: From Dynamic Interfaces to Data Repos

- Web apps today are getting increasingly **dynamic**
  - Frequently update webpage content via JavaScript
  - Manipulate the **Document Object Model (DOM),** a tree-like page representation
- More recently, DOM used to also store **sensitive data** (e.g., one-time tokens)



The Story of Webpages: From Dynamic Interfaces to Data Repos

- Web apps today are getting increasingly **dynamic**
  - Frequently update webpage content via JavaScript
  - Manipulate the **Document Object Model (DOM),** a tree-like page representation
- More recently, DOM used to also store **sensitive data** (e.g., one-time tokens)

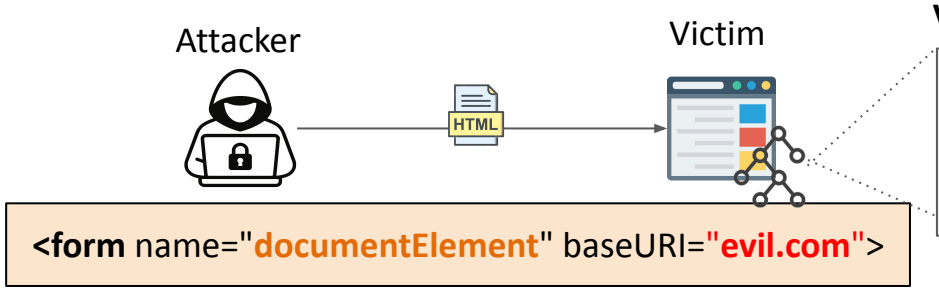
⚠ Are we **validating** all the **read data** properly

sensitive operations

```
, {
  method: "POST",
  headers: { "X-CSRF-Token": "CSRF-TOKEN" },
  body: "..."
});
```

Reading from DOM: Past Vulnerability Instances

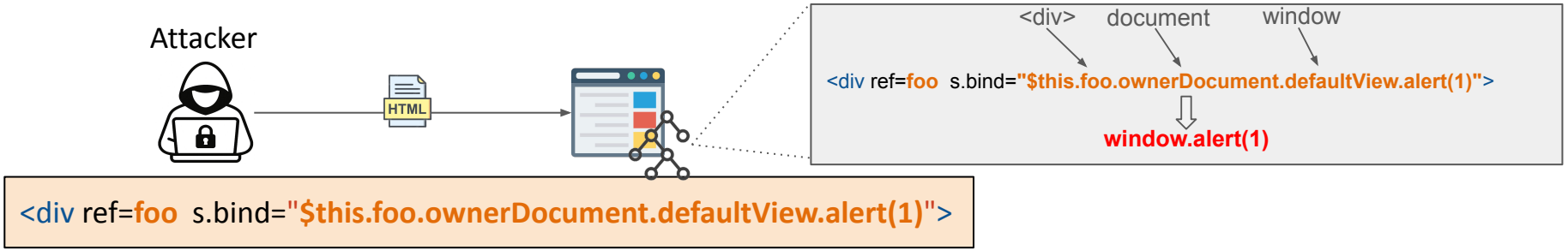
- **DOM Clobbering** [IEEE SP'23, USENIX'25]
  - Inject HTML with **colliding id/names**, achieve code execution



Reading from DOM: Past Vulnerability Instances

- **DOM Clobbering** [IEEE SP'23, USENIX'25]
  - Inject HTML with colliding id/names, achieve code execution

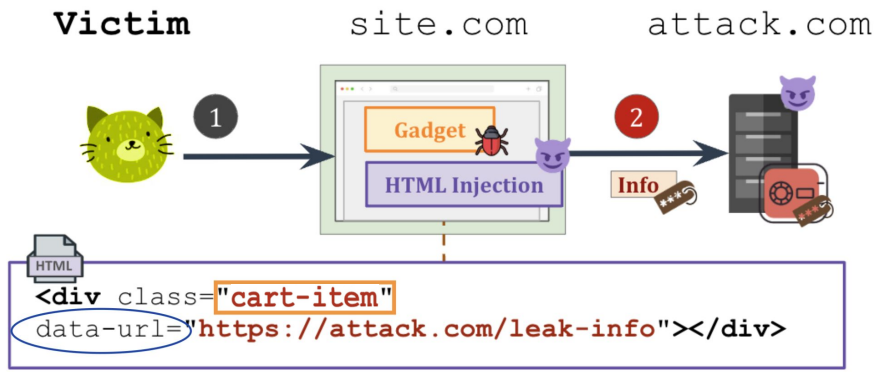
- **Script gadgets** [CCS '17]
  - Inject HTML matching **DOM selector API**, achieve code execution



Reading from DOM: Unexplored and Diverse Attacks Beyond XSS

- Exploit **legitimate** application JavaScript code (**DOM gadgets**) for malicious purposes
  - Data exfiltration
  - Cross-site Request Forgery (CSRF)
  - …

```
var cartItems =
    document.querySelectorAll('.cart-item');
for (const elem of cartItems) {
    var url = elem.getAttribute('data-url');
    // check item inventory and price
    fetch(url, {
        method: 'POST',
        headers: { 'XSRF-Token': "xyz" }),
        body: JSON.stringify({ ... }),
    }).then(resp => { /* [...] /* });
}
```

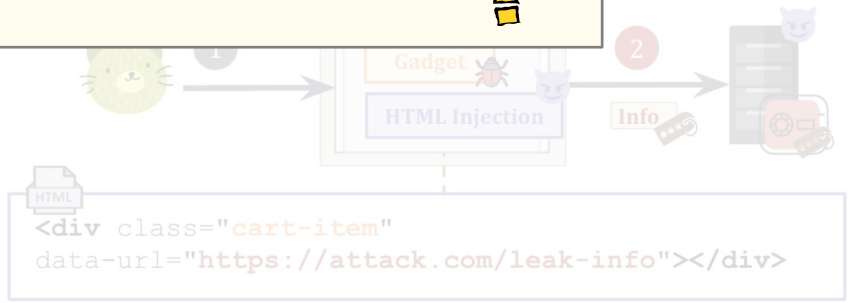


Reading from DOM: Unexplored and Diverse Attacks Beyond XSS

- Exploit **legitimate** application JavaScript code (**DOM gadgets**) for malicious purposes
  - Data exfiltration
  - Cross-site Request Forgery (CSRF)
  - …

What **additional attacks** can originate from unsafe DOM reads

```
1  <scr
2  var
3
4  for
5
6      // check item inventory and price
7      fetch(url, {
8          method: 'POST',
9          headers: { 'XSRF-Token': "xyz" }),
10         body: JSON.stringify({ ... }),
11     }).then(resp => { /* [...] /* });
12 }
13 </script>
14 <div class="cart-item"
15   data-url="/api/v1/checkInventory?id=item-12345">
16 </div>
```



Information Leakage!

Research Questions

- **RQ1: Gadget Systematization**
  - What types of DOM gadgets exists?
  - With what techniques can they be exploited?

Research Questions

- **RQ1: Gadget Systematization**
  - What types of DOM gadgets exists?
  - With what techniques can they be exploited?

- **RQ2: Gadget Detection and Prevalence**
  - How can we detect DOM gadgets? How prevalent are they?

Research Questions

- **RQ1: Gadget Systematization**
  - What types of DOM gadgets exists?
  - With what techniques can they be exploited?

- **RQ2: Gadget Detection and Prevalence**
  - How can we detect DOM gadgets? How prevalent are they?

- **RQ3: Exploitable Gadgets and Impact**
  - How many pages with DOM gadgets are truly exploitable, that is, have also a markup injection vulnerability to trigger them?

RQ1- Gadget Systematization

- Identified **7 gadget variants** by systematically reviewing HTML living standards & prior work
  - **Known:** code execution & markup injection **(script gadgets)**
  - **New:** Async. requests, WebSockets, Navigation, object loading, forms/links

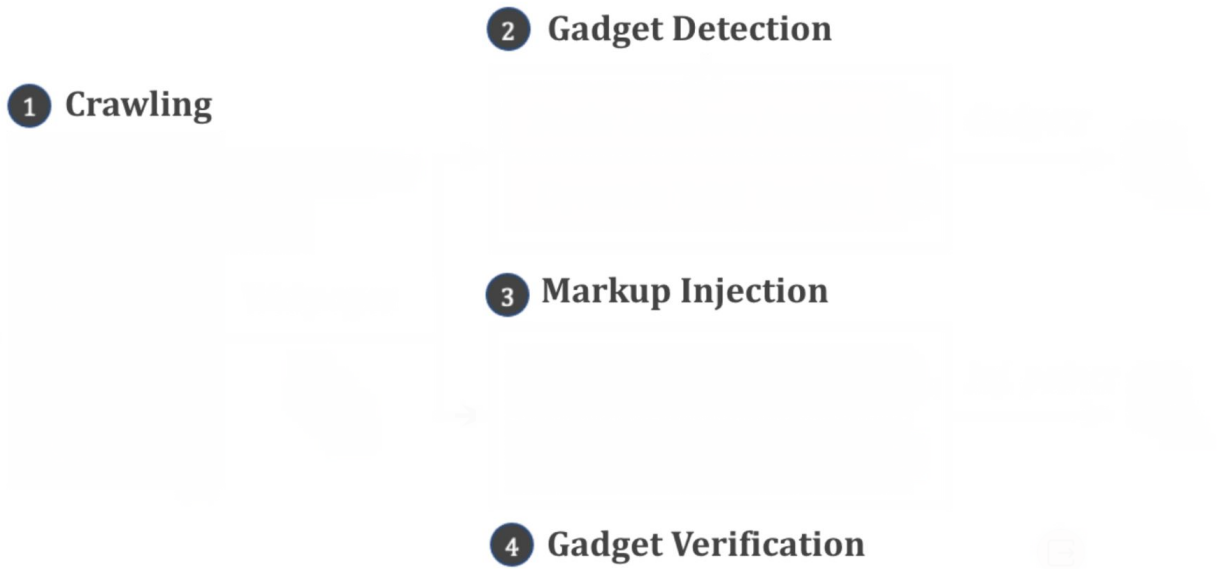
RQ1- Gadget Systematization

- Identified **7 gadget variants** by systematically reviewing HTML living standards & prior work
  - **Known:** code execution & markup injection **(script gadgets)**
  - **New:** Async. requests, WebSockets, Navigation, object loading, forms/links

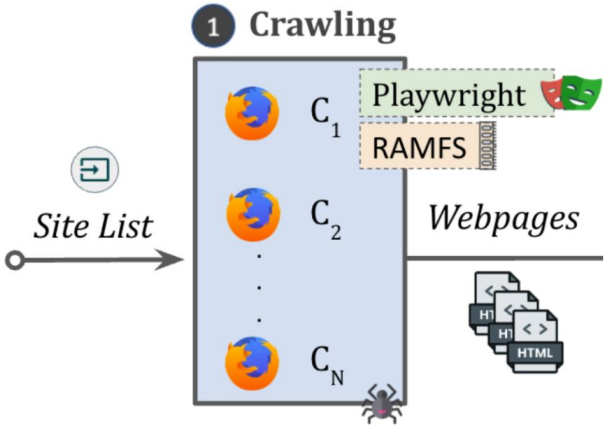
- Each variant tied to **specific sensitive instructions** → distinct threats

Gadget	XSS	Content Manip.	Phishing	Unauth Action	Info Leak	Session Hijack.	Open Redirect	Drive-by Downl.	Rogue Plugin
Code Execution	●	○	○	○	○	○	○	○	○
Markup Injection	●	●	●	○	○	○	○	○	○
⊕ Async. Request	○	○	○	●	●	○	○	○	○
⊕ WebSocket	○	○	○	●	●	●	○	○	○
⊕ Navigation	●	○	●	●	○	●	●	○	○
⊕ Object Loading	●	○	○	○	●	○	○	●	●
⊕ Form/Link Manip.	●	○	●	○	●	○	●	○	●

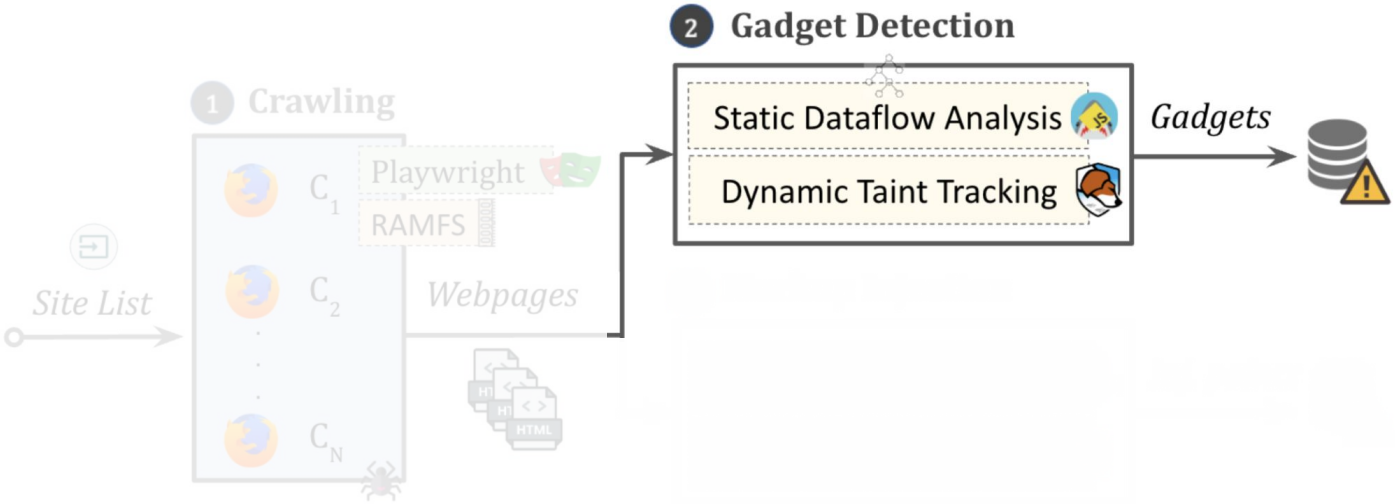
RQ2- Gadget Detection and Prevalence: Methodology



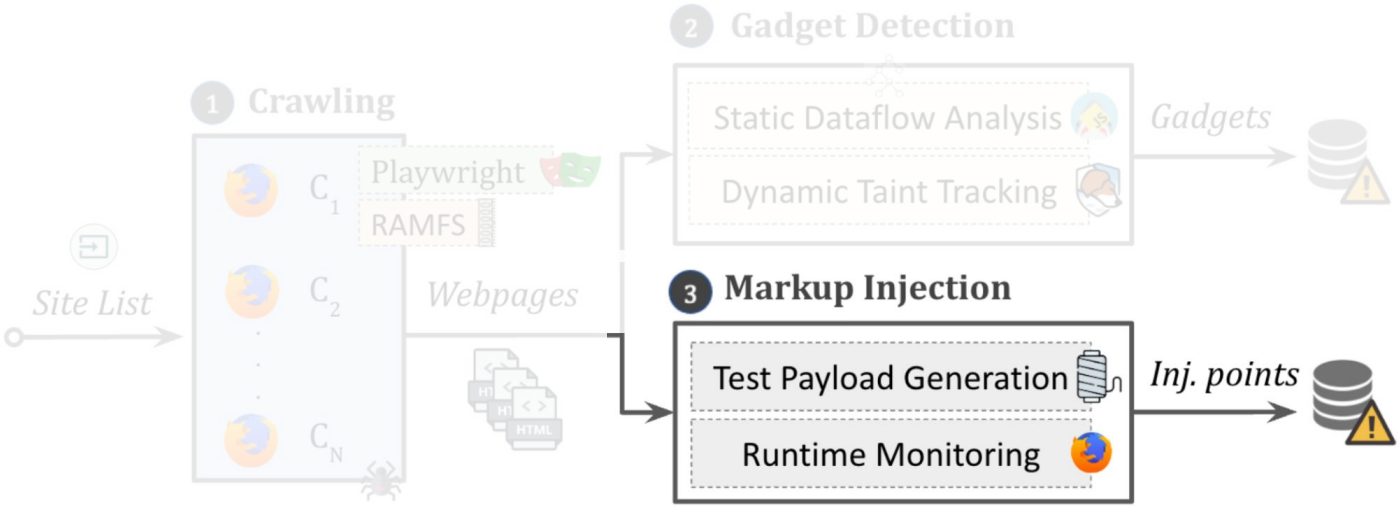
RQ2- Gadget Detection and Prevalence: Methodology



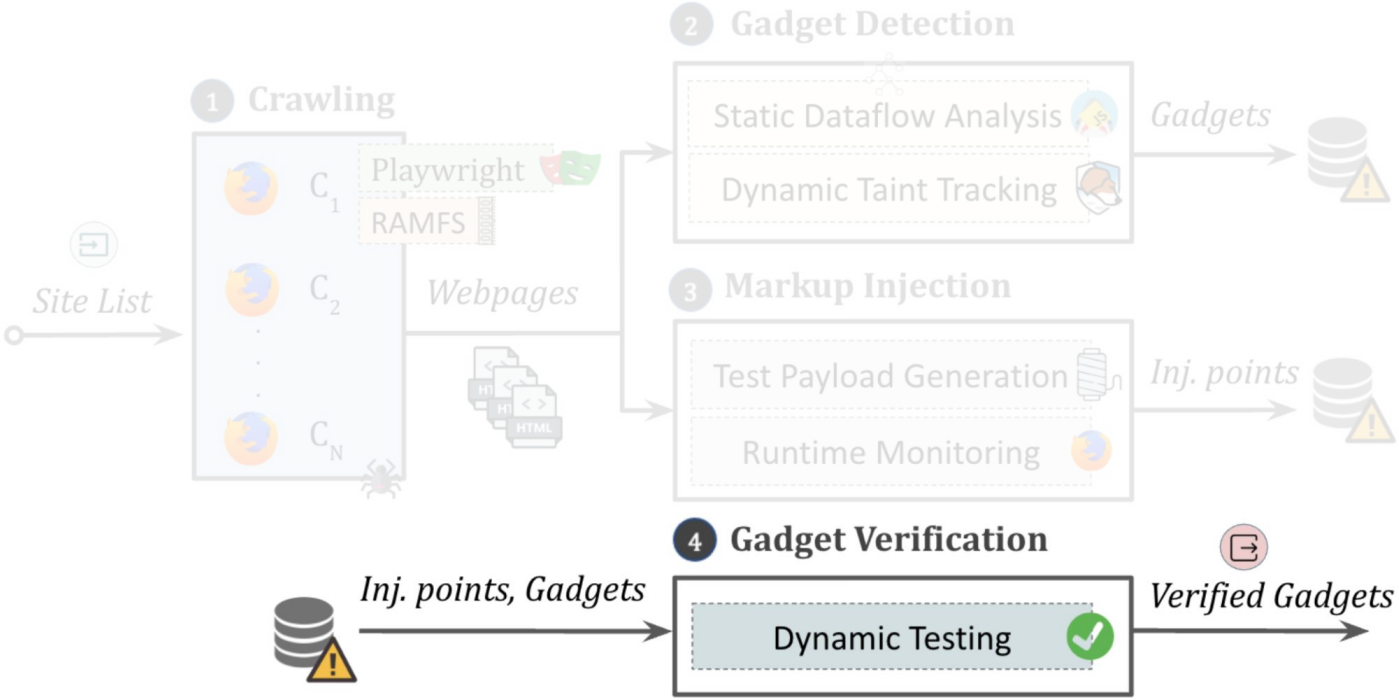
RQ2- Gadget Detection and Prevalence: Methodology



RQ2- Gadget Detection and Prevalence: Methodology



RQ2- Gadget Detection and Prevalence: Methodology



RQ2- Gadget Detection and Prevalence: Results

- Collected snapshots of webpages using Playwright and an instrumented Firefox browser

Tranco top **15K sites**, over **522K pages**, 19M scripts, and 10B LoC

June 2024

RQ2- Gadget Detection and Prevalence: Results

- Collected snapshots of webpages using Playwright and an instrumented Firefox browser

Tranco top **15K sites**, over **522K pages**, 19M scripts, and 10B LoC

June 2024

- Static-dynamic detection pipeline

2.6M DOM gadgets on **364K pages** across over **9K sites**

- Most common variants:
  - Asynchronous Request gadgets ⟶ **1.2M**
  - Object gadgets ⟶ **581K**
  - Script gadgets ⟶ **305K**

RQ2- Gadget Detection: Contribution of Static and Dynamic Analysis

- Static **complements** dynamic analysis
  - **Dynamic analysis:** **scalable** and low false positives, but misses condition-dependent flows (e.g., user actions)
  - **Static analysis:** explores unexecuted or hard-to-reach paths ⟶ provides **unique coverage**

RQ2- Gadget Detection: Contribution of Static and Dynamic Analysis

- Static **complements** dynamic analysis
  - **Dynamic analysis:** scalable and low false positives, but misses condition-dependent flows (e.g., user actions)
  - **Static analysis:** explores unexecuted or hard-to-reach paths → provides **unique coverage**
  - Static found at least one gadget **missed by** dynamic analysis:
    - Code Execution (5,081 pages)
    - Markup (11,065 pages)
    - Link (4,544 pages)

Takeaway: Dynamic gives **scale**, static adds **unique coverage**

RQ2- Gadgets In the Wild: Input Validation

- Looked for presence of input validation or sanitization checks on statically-discovered gadgets
  - **60%** lacked any sanitization/validation logic
  - **10%** had **no checks at all**

Takeaway: developers **trust DOM** content, leaving sensitive operations unprotected

RQ3- Exploitability: Markup Injections In the Wild

- Markup injection is a **crucial requirement** for exploiting DOM gadgets
- **Methodology:**
  - Used Foxhound to detect dataflows from web attacker sources like URL to markup sinks
  - Auto-generated breakout payloads to (a) attempt XSS or (b) inject script-less markup when XSS is blocked

RQ3- Exploitability: Markup Injections In the Wild

- Markup injection is a **crucial requirement** for exploiting DOM gadgets
- **Methodology:**
  - Used Foxhound to detect dataflows from web attacker sources like URL to markup sinks
  - Auto-generated breakout payloads to (a) attempt XSS or (b) inject script-less markup when XSS is blocked

- **Results:**
  - Discovered **204K** markup injection data flows across **1.8K** domains
  - Verified **4.7K markup injection flows**
  - **343** verified flows do **not** yield XSS — they are exploitable **only** via DOM gadgets

RQ3- Gadget Verification and Exploitability

- **Verification method:**
  - Auto-generate matching markup, inject into the page **before parsing**
  - Wrap sink prototypes to log arguments when our payload reaches a sink

RQ3- Gadget Verification and Exploitability

- **Verification method:**
  - Auto-generate matching markup, inject into the page **before parsing**
  - Wrap sink prototypes to log arguments when our payload reaches a sink

- **Verified gadgets:** **357K** DOM gadgets confirmed across **2,500** sites.

Over **77%** of **verified** gadgets are **new gadget** types (non-script gadgets)

RQ3- Gadget Verification and Exploitability

- **Verification method:**
  - Auto-generate matching markup, inject into the page **before parsing**
  - Wrap sink prototypes to log arguments when our payload reaches a sink

- **Verified gadgets:** **357K** DOM gadgets confirmed across **2,500** sites.

Over **77%** of **verified** gadgets are **new gadget** types (non-script gadgets)

- **End-to-end flows:** cross-referencing DOM gadgets & markup flows produced **304K** candidates

657 candidates on **37 sites** have both **verified** markup injection and **verified** gadget

Summary

- **First characterization** and **end-to-end detection** approach for DOM gadget vulnerabilities
- **Ubiquity: 70%** of the analyzed pages contain at least one DOM gadget
- **Novelty: 77%** of detected gadgets belong to **new DOM gadget variants**
- **Developer posture: 10%** of DOM read dataflows have **no validation** at all

https://github.com/jndre/In-the-DOM-We-Trust

Questions? sepehrmirzaei98@gmail.com
jan.drescher@tu-braunschweig.de