OWASP 2022 VIRTUAL APPSEC JUN 6-10

TESTABLE

PRESENTED BY: Soheil Khodayari

Everything You Wanted to Know About Client-side CSRF (But Were Afraid to Ask)

CISPA Helmholtz Center for Information Security

soheil.khodayari@cispa.de @Soheil__K

CISPA

SCAN ME

About Me

Soheil Khodayari

PhD Candidate @CISPA, Germany (2019 – Present)

Web Security, Program Analysis

Double MSc. in Computer Science (2017-2019)

- Polytechnic University of Madrid - Technical University of Kaiserslautern
- R&D Engineer @IMDEA, Madrid

Publications in NDSS, USENIX Security, IEEE S&P, RAID

SCAN ME

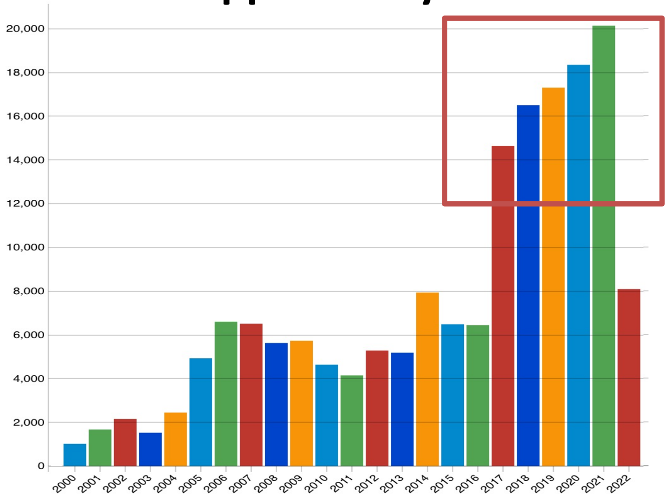
Web Applications Testability

- We know that webapp vulnerability detection is critical
  - Over 4.8 billion websites online, 1.8 billion users [1]
  - Contain a variety of security-sensitive data

Banking Shopping Education

- The complexity of webapps are rising

Webapp CVEs By Year [2]

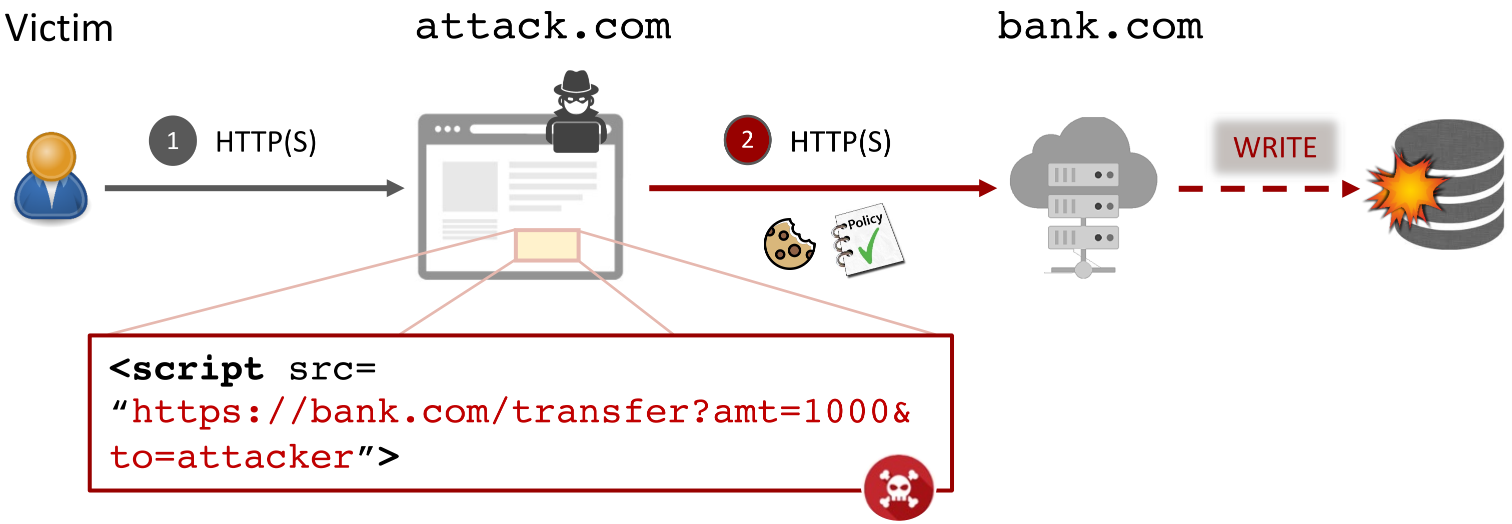


Problem:

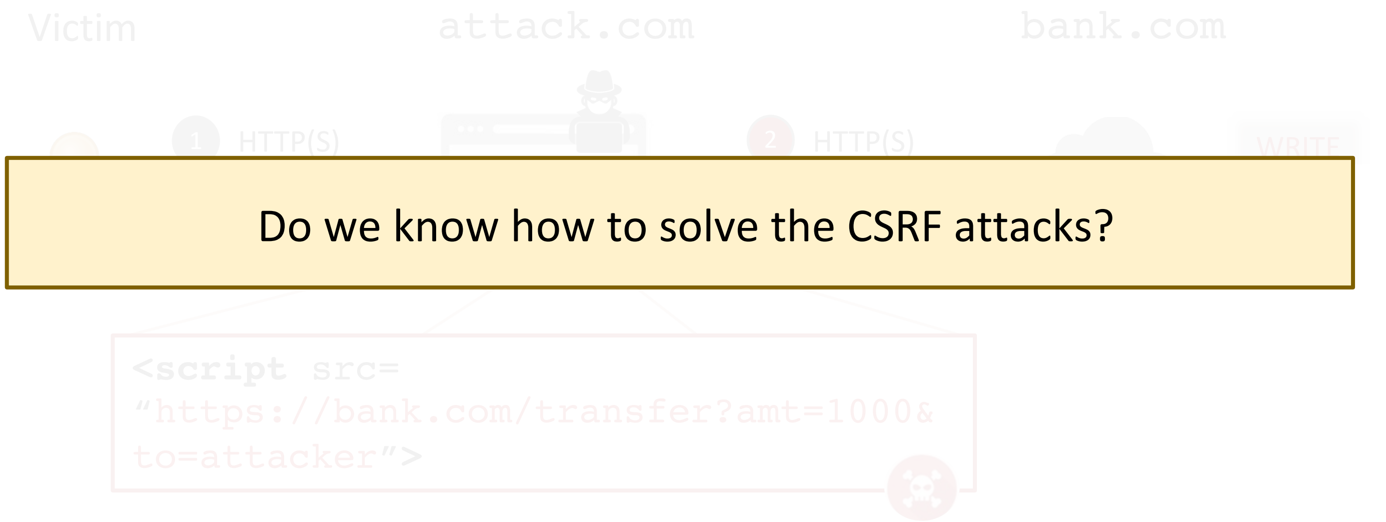
⚠ Existing vulnerability detection tools fall short of capturing this complexity

Sources:
1 internetlivestats.com
2 nvd.nist.gov

Cross-Site Request Forgery (CSRF)



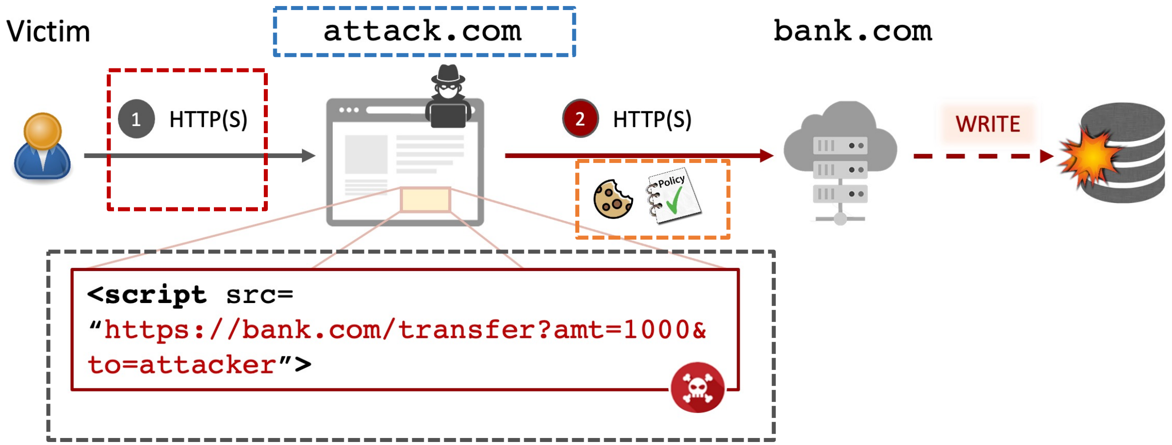
Cross-Site Request Forgery (CSRF)



Do we know how to solve the CSRF attacks?

Anti-CSRF Defenses

✓ Robust CSRF defenses are well-known



Origin Checks
- Referrer/Origin Check
- Custom Request Headers

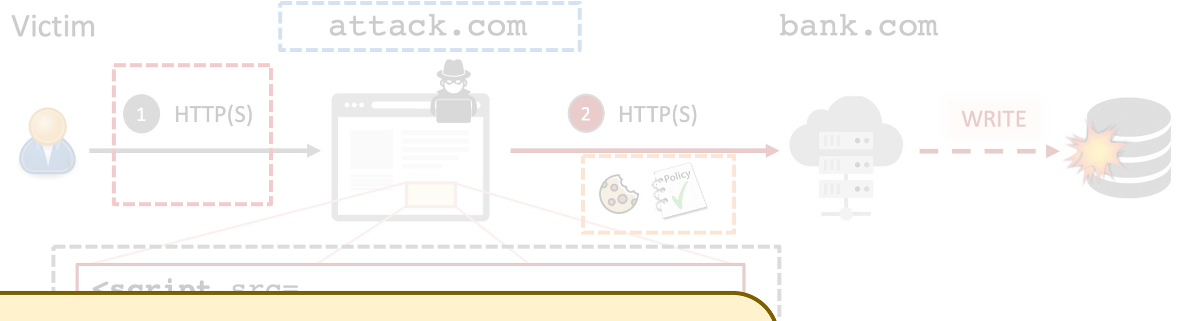
Req. Unguessability
- Plain Token
- HMAC Token
- Double/Triple Submit
- Cookie-less User Sessions

SOP for Cookies
- SameSite Cookies
- Freq. Log Outs (server)
- Browser Extensions
- Server-side Proxies

User Intention
- Re-authentication
- One-Time Token
- (re)CAPTCHA

Anti-CSRF Defenses

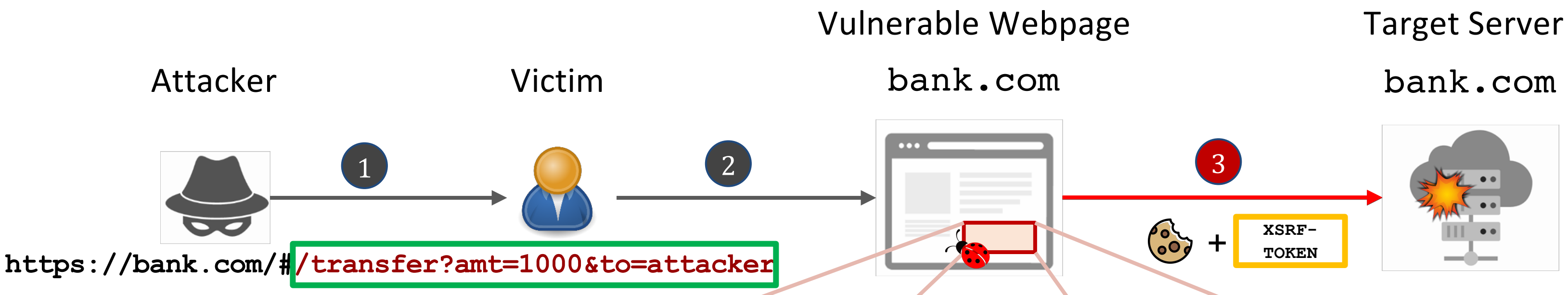
Robust CSRF defenses are well-known



With these defenses properly implemented, are CSRF attacks solved?

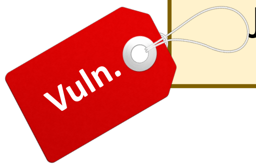
Origin			ntention
Referrer/Origin Check	Plain Token	SameSite Cookies	Re-authentication
Custom Request Headers	HMAC Token	Freq. Log Outs (server)	One-Time Token
	Double/Triple Submit	Browser Extensions	(re)CAPTCHA
	Cookie-less User Sessions	Server-side Proxies	

Client-side CSRF: Vulnerability




```
var uri = window.location.hash.substr(1);
if (uri.length > 0) {
    uri = "bank.com/api" + uri;
    let req = new asyncRequest("POST", uri);
    req.setBody({"csrf_token" : "XSRF-TOKEN"});
    // [...]
}
```

Root Cause:

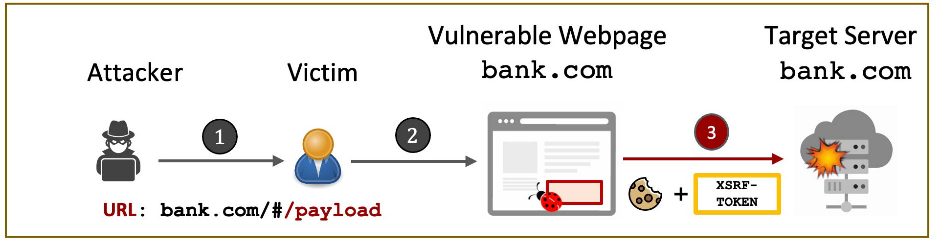


JavaScript progam uses attacker-controlled inputs to generate async HTTP requests

Client-side CSRF: Problem Statement

- Limited knowledge about client-side CSRF.
  - Facebook in 2018[1]

- **Objective:** studying client-side CSRF vulnerabilities
  - **(RQ1)** Prevalence of client-side CSRF in webapps?
  - **(RQ2)** Exploitations for different attacker models?
  - **(RQ3)** Degree of attacker control?
    - E.g., path, query, domain, body




```
POST /path/file.php?q=v\r\n
Host: example.com\r\n
\r\n
{body}
```

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

Static Analysis (to the Rescue)

- **Javascript**
  - Event-driven language
  - Prototype-based inheritance (no static class hierarchies)
  - Runtime types, coercions (no static type checking)

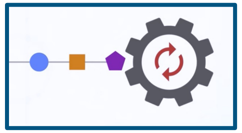
> Analysis of client-side JavaScript programs is not an easy task

- **Challenges**
  - Inherent dynamic language features
    - E.g., `eval()`, or `setTimeout()` functions [S.H. Jensen, ISSTA'12]
  - Pointer analysis (e.g., ThisExpressions like `this.property`) [S. Wei et. al., ECOOP'14], [B Stein et. al. PACMPL'19]
  - Inter-procedural calls [G. Antal, SCAM'18]
  - Minified scripts and obfuscated code

Static Analysis (Cont'd)

- Modeling JavaScript is not enough, code environment also matters
  - ECMAScript standard library
  - Browser APIs [S.H. Jensen, FSE'11]
  - HTML DOM tree
  - Client-side Events

- JavaScript streaming programming model [S. Guarnieri et. al., USENIX'10]



- Modern client-side libraries
  - Sweet on the outside, bitter on the inside [M. Madsen et. al., FSE'13]
  - E.g., JQuery, Dojo, YUI, etc

Client-side CSRF: Exemplifying Detection Challenges

- **(C1)** Event-based transfer of control
- **(C2)** Dynamic web execution environment
- **(C3)** Modelling shared third-party code

```
function sendRequest() {
  var uri = window.location.hash.substr(1)
  ~~fetch(uri ,{ method: "POST" })~~
  extLibraryHttpRequest(uri ,{ method: "POST" })      4
  // [...]
}
var invoice = document.querySelector('div')           3
invoice.addEventListener('LoadInvoice', sendRequest)
// [...]
function showInvoicePrice() {                         2
  // [...]
  let invoice = document.getElementById('invoice-id')
  invoice.dispatchEvent(new CustomEvent('LoadInvoice', {}))
}
// [...]
showInvoicePrice()                                    1
```

Other General Challenges for CSRF

- **Detection Challenges**
  - Support for modern scanning barriers, e.g., login
  - Scalability and performance

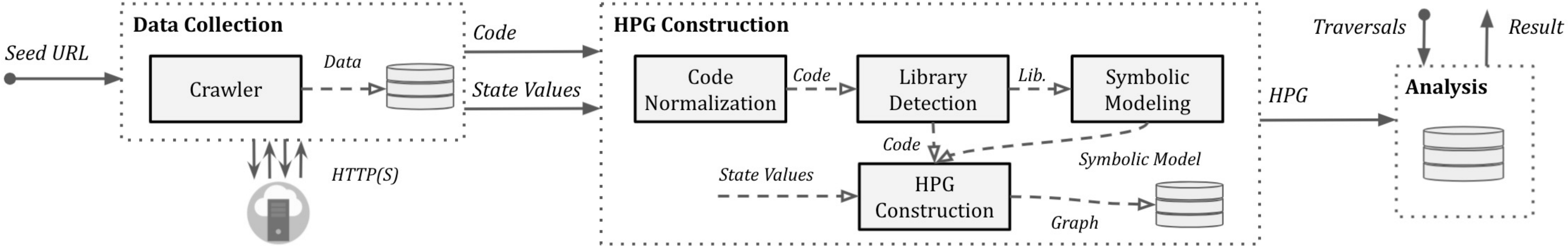
- **Operational Challenges**
  - Side-effect free testing
  - Security-relevant state changes

Approach Overview: JAW

- A scalable, graph-based framework for detection and exploratory analysis of client-side CSRF vulnerabilities
- Components
  - Data Collection
  - Graph Construction
  - Analysis Traversals

SCAN ME

https://soheilkhodayari.github.io/JAW

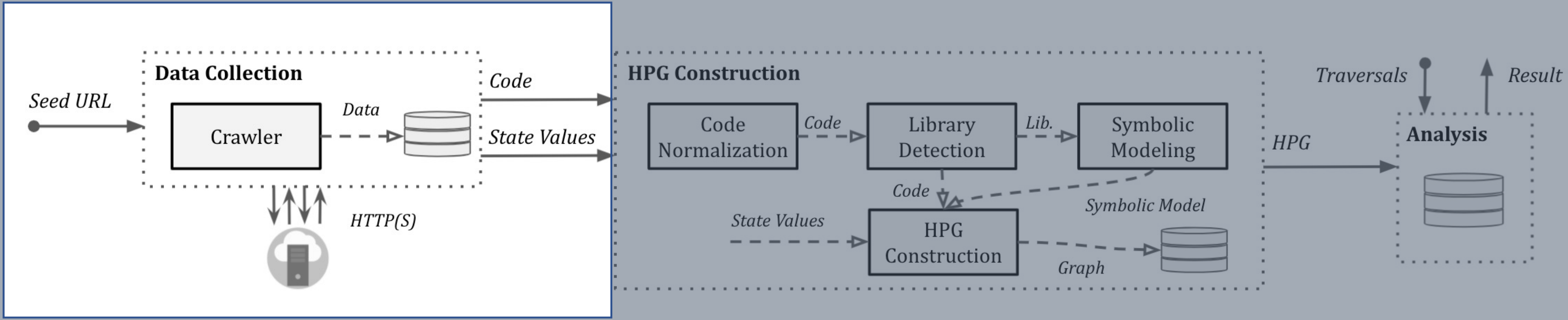


Approach Overview: JAW

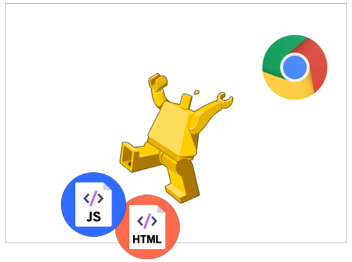
- A scalable, graph-based framework for detection and exploratory analysis of client-side CSRF vulnerabilities
- Components
  - Data Collection
  - Graph Construction
  - Analysis Traversals

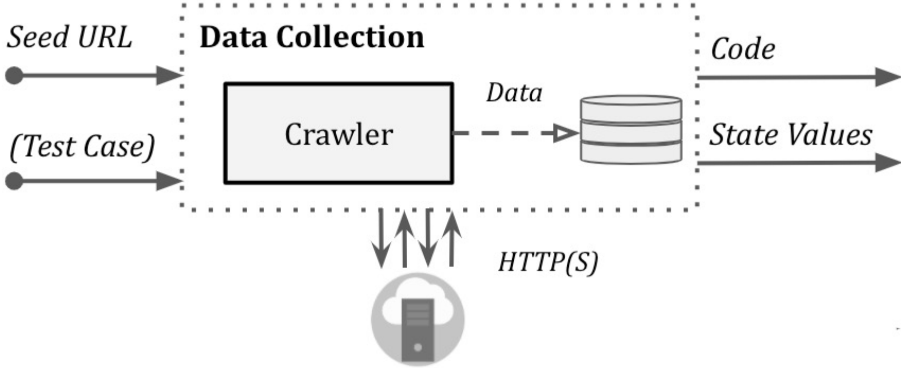
SCAN ME

https://soheilkhodayari.github.io/JAW



JAW: Data Collection





- Chrome-based crawler with Selenium
- Enhanced with chrome extensions

- **Outputs:**
  - JavaScript Code
  - HTTP Requests and Responses
  - Dynamically Fired Events
  - Concrete snapshots of the global `Window` object
    - `window.document` (DOM tree)
    - `window.localStorage`
    - `window.document.cookie`
    - …

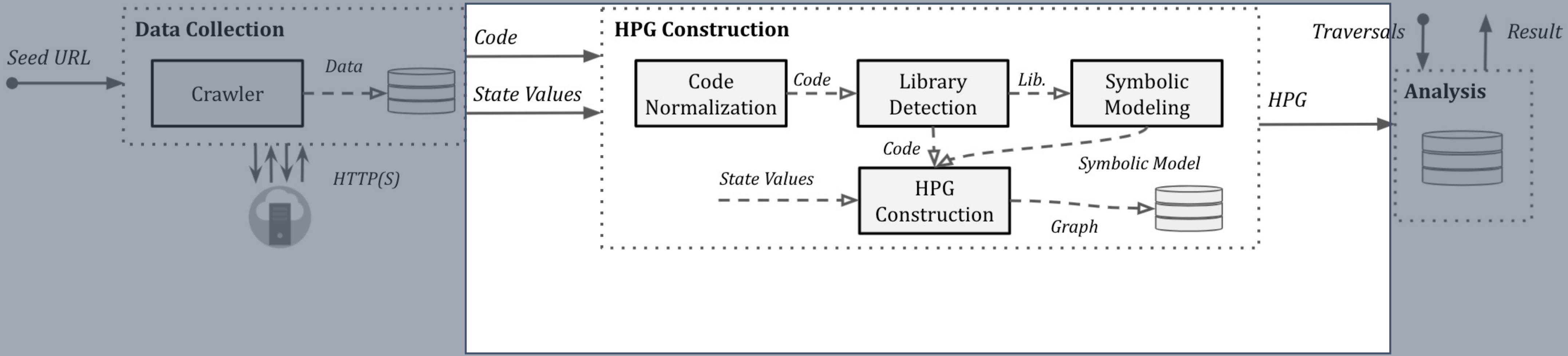
Dynamic Information

Approach Overview: JAW

- A scalable, graph-based framework for detection and exploratory analysis of client-side CSRF vulnerabilities
- Components
  - Data Collection ✓
  - Graph Construction
  - Analysis Traversals

SCAN ME

https://soheilkhodayari.github.io/JAW



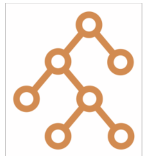
Hybrid Property Graphs (HPGs): Building Blocks

- **Code Representation (Static)**
  - Abstract Syntax Tree (AST)
  - Control Flow Graph (CFG)
  - Program Dependence Graph (PDG)
  - Inter-Procedural Call Graph (IPCG)
  - Event Registration, Dispatch and Dependency Graph (ERDDG)
  - Semantic Types and Symbolic Models
- **State Values (Dynamic)**
  - Event Traces
  - Environment Properties

CPG for C/C++
[Yamaguchi et al, S&P’14]

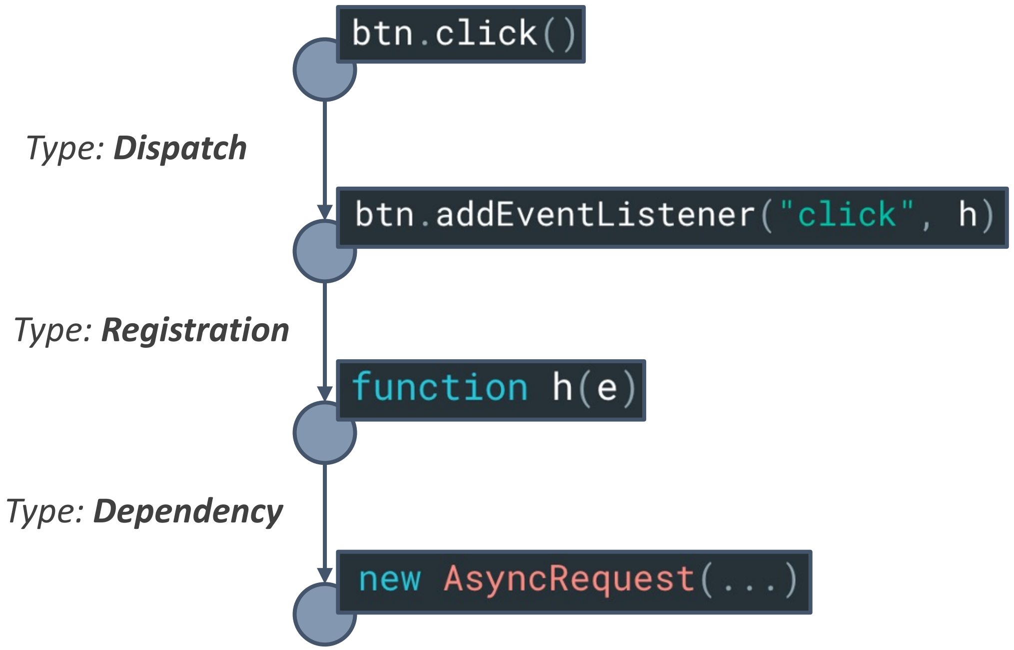
CPG for PHP
[Backes et al., EuroS&P’17]

HPG for JavaScript



HPGs: Event Registration, Dispatch and Dependency Graph (ERDDG)

- **Problem:**
  - Event dispatches can change the state of JavaScript programs
  - Need to be modeled
- **Solution:**
  - The ERDDG



Example Snippet:

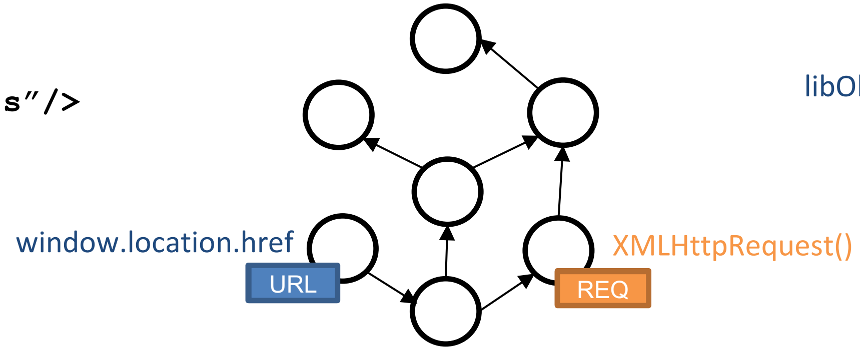
```
var btn = document.querySelector("button");
function h(e){
        new AsyncRequest(...);   ③
}
btn.addEventListener("click", h);   ②
// [...]
btn.click();   ①
```

HPGs: Symbolic Models and Semantic Types Propagation

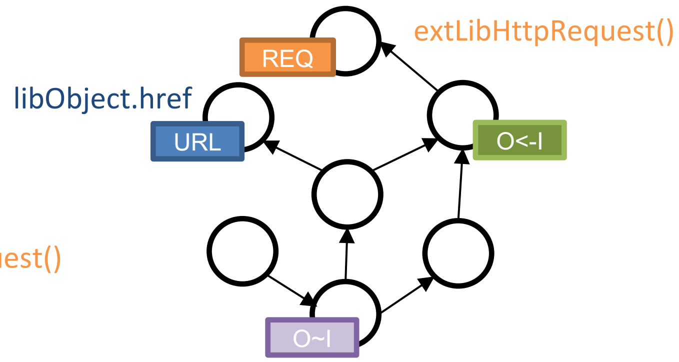
- External libraries: over 60% of the total LoC of each webpage.
- **Problem:**
  - Existing approaches: Inefficient, include library code in the analysis
- **Idea:** Shared models for JavaScript libraries

`<script src="lib.js"/>`

HPG for *lib.js*



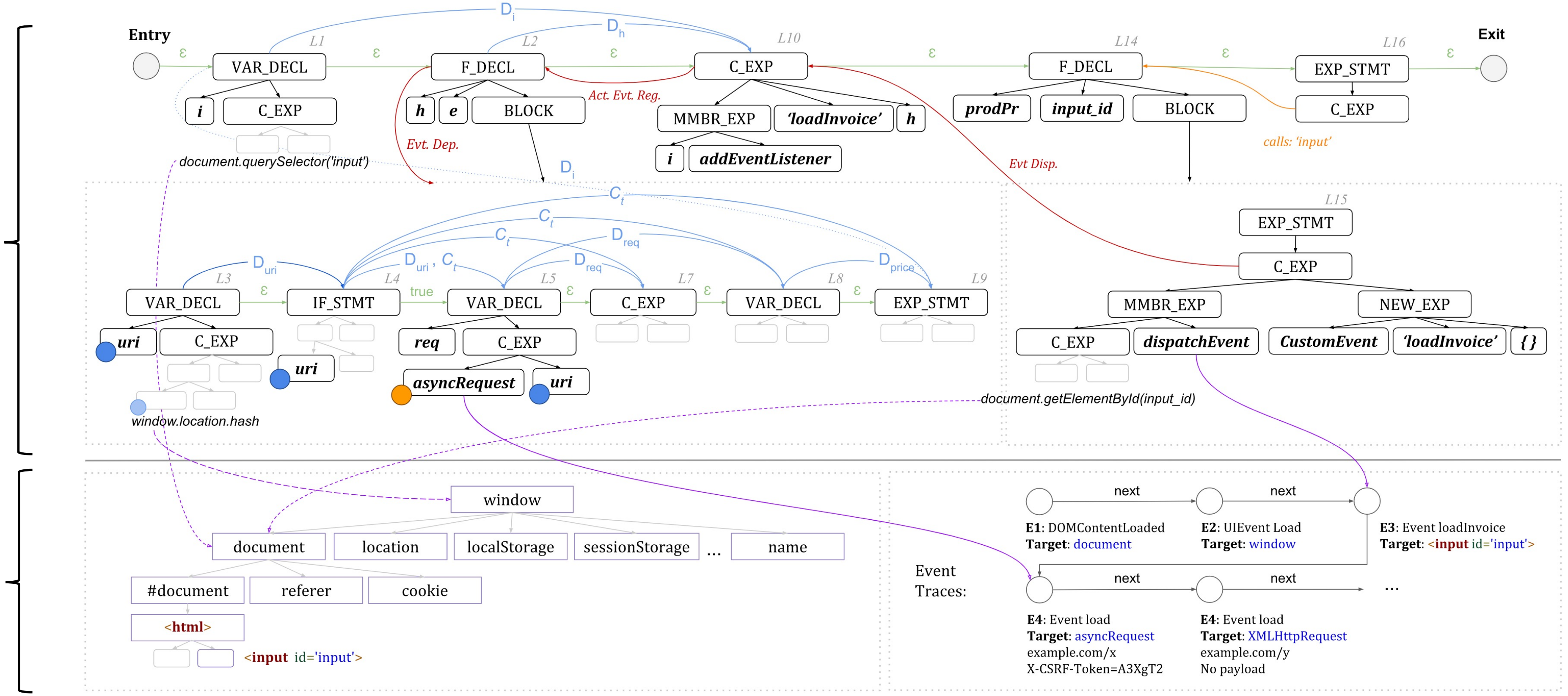
Symbolic Model for *lib.js*



Hybrid Property Graph: Example

Code Representation
(*Static Part*)

State Values
(*Dynamic Part*)

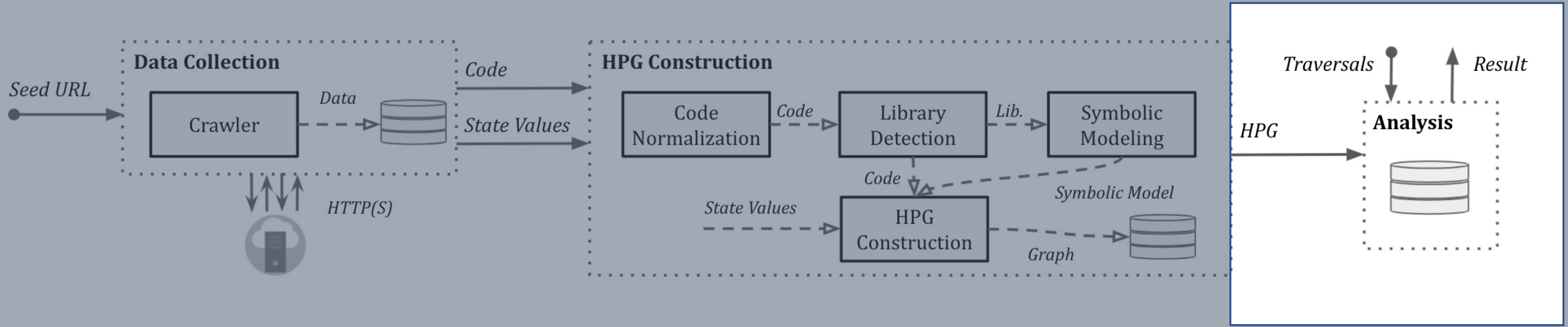


Approach Overview: JAW

- A scalable, graph-based framework for detection and exploratory analysis of client-side CSRF vulnerabilities
- Components
  - Data Collection ✔
  - Graph Construction ✔
  - Analysis Traversals

SCAN ME

https://soheilkhodayari.github.io/JAW



Vulnerability Analysis

- **Client-side CSRF**
  A. Data flow from an attacker-controlled input to a param of a request R.
    - lines of code having both **"URL"** and **"REQ"** semantic types.

  B. R is reachable at page load.

- Model both conditions using declarative graph traversals

neo4j
CQL
Cypher Query Language

- A query Q contains all nodes n of HPG for which a predicate P is true: Q = {n : P(n)}

Detection Query

```
QA ={n : isDeclOrStmt(n) ∧ ∃c1, c2, c1 != c2
        ∧ hasChild(n, c1) ∧ hasSemType(c1, "REQ"),
        ∧ hasChild(n, c2) ∧ hasSemType(c2, "URL")
}
```

Exploitation Landscape: Attacker Models

Attacker Goal

- Forge HTTP requests by manipulating various JavaScript input sources

JavaScript Input Sources

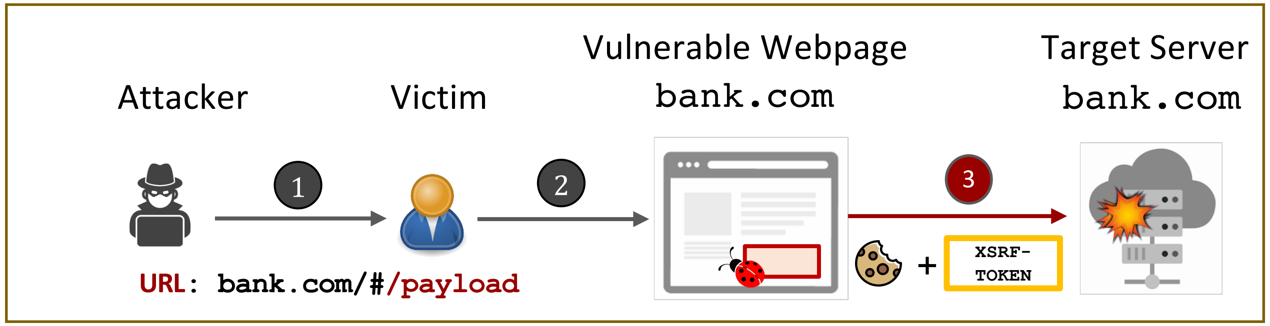
- Window URL
- Window Name
- postMessages
- Doc. Referrer

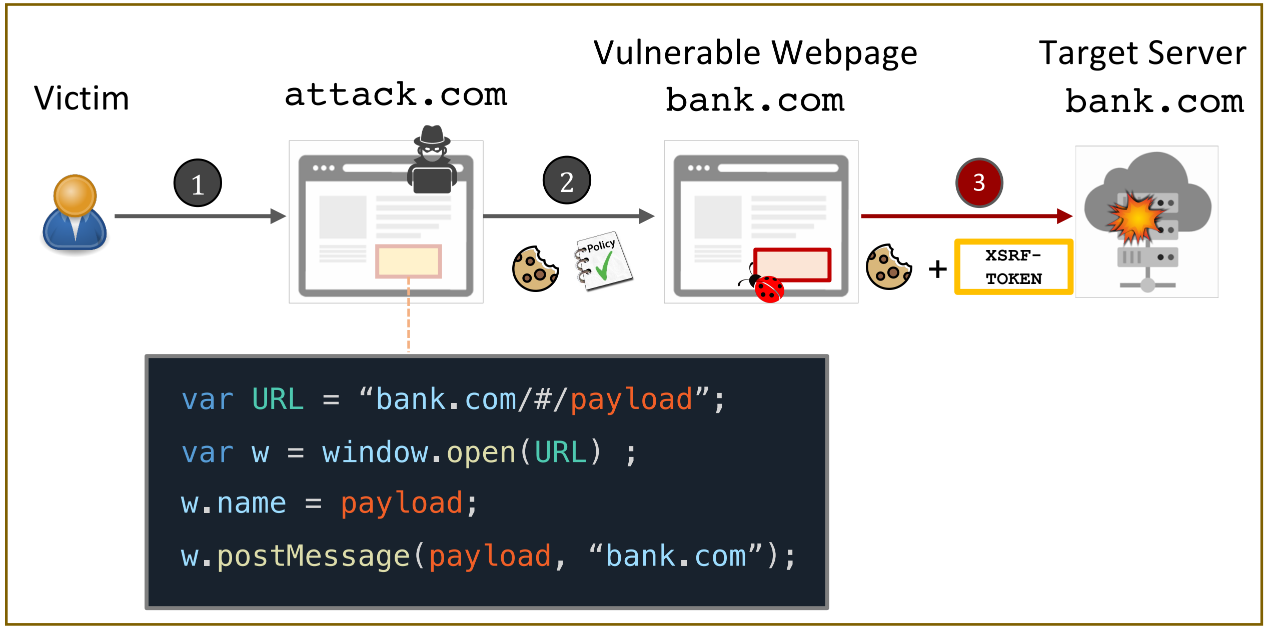
→ Web Attacker

- Web Storage
- HTML attributes

→ Web Attacker → Functionality / Injection Vuln.

- Cookies → Network Attacker

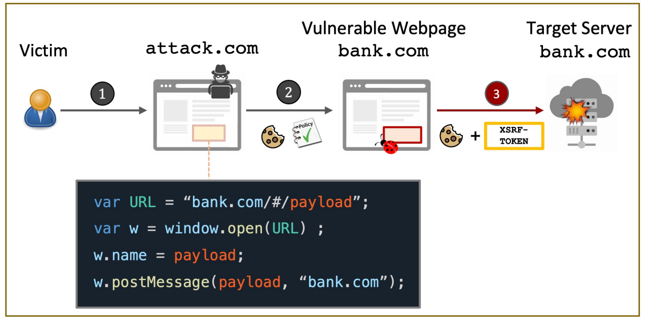




Evaluation: Forgeable Requests

- Evaluated JAW with all webapps from the Bitnami catalog
  - 106 webapps
  - 228M LoC

⚠ Detected 12,701 forgeable requests affecting 87 webapps



Input Source	# Forgeable	# Apps
DOM.COOKIES	67	5
DOM.READ	12,268	83
*-Storage	76	8
DOC.REFERRER	1	1
POST-MESSAGE	8	8
WIN.NAME	1	1
WIN.LOC	280	12
Total Forgeable	12,701	87
Total Requests	49,366	106

Exploitations

- Manually looked for practical exploitations in 516 requests

☠ Created exploits for 203 requests of seven webapps

  - E.g., SuiteCRM, Neos, Kibana, Modx
  - Account takeover, deleting user assets, …

Impact: SuiteCRM - Example 1/2

Vulnerability

- URL hash fragment
- Example:
  - `https://suitecrm.com#ajaxUILOC=`**URL**

Attack

- Forge authenticated requests to any sensitive endpoint
- Corrupt the database integrity
  - Delete accounts, contacts, cases, or tasks

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)
}
```

```
// 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);
}
```

Impact: Cotonti - Example 2/2

Simplied Snippet: Cotonti

Vulnerability

- Use URL hash fragment as the endpoint of an async HTTP requests
- Control also the request method

Attack

- Example:
  - `https://cotonti.com/admin.php?m=config#get;m=config&n=edit&o=plug&p=cleaner&a=reset&v=userprune&t=1m`
- Change administrative configuration
  - e.g., delete inactive user accounts older than one minute, delete logs, etc

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

1

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

2

3

Anatomy of Forgeable Requests

- Exploitation landscape can be influenced by:
  - Type of controllable fields
  - Operation to forge a field
- Identified 25 distinct templates. For example:
  - 185/ 516 requests: manipulate any part of domain + path + query
  - 20/ 516 requests: manipulate multiple parts of path + body
  - 166/ 516 requests: manipulate a single part of body
  - See our *paper* for more

```
POST /path/file.php?q=v\r\n
Host: example.com\r\n
\r\n
{body}
```

Outgoing HTTP Request						Total	
Dom.	Path	Query	Body	Part	Control	Reqs	Apps
		✓		One	-, A, -	16	11
			✓	One	-, A, -	5	5
			✓	One	W, -, -	(*)166	25
			✓	One	-, -, P	1	1
	✓			One	W, -, -	28	1
	✓			One	-, A, -	7	7
	✓			One	-, -, P	6	6
		✓		One	-, -, P	11	11
	✓		✓	Mult	-, A, -	4	1
	✓		✓	Mult	W, -, -	(*)20	1
	✓	✓		Mult	W, A, P	6	1
		✓	✓	Mult	W, -, -	2	1
		✓		Mult	-, A, -	7	7
			✓	Mult	-, -, P	2	2
	✓			Mult	-, A, -	3	3
		✓		Mult	-, -, P	1	1
			✓	Mult	-, A, -	5	5
	✓			Mult	-, -, P	6	6
			✓	Mult	W, -, -	28	8
	✓	✓		Any	W, -, -	1	1
✓	✓	✓		Any	W, -, -	(*)185	8
✓	✓	✓	✓	Any	W, -, -	1	1
		✓	✓	Any	W, -, -	(*)1	1
			✓	Any	W, -, -	2	2
	✓	✓	✓	Any	W, -, -	1	1

Legend: A=Appending; P=Prepending; W=Writing.

Evaluation: Contributions of New Models

Role of the Event Graph

- Event Dispatch Edges: 6,451,582
- Function Call Edges: 7,179,021

→ **+89.8%** in edges transferring the control.

Importance of Symbolic Modeling

- Total of ~ 228M LoC of which ~ 138M are libraries
- Distinct library code only ~ 412K (335 times smaller)

→ **-60.3%** LoC to process to build the HPG

Impact of Dynamic Snapshotting

- Captured ~ 10.7M more nodes & ~ 13.3M more edges (i.e., dynamic insertion of script tags)
- Identification of 840 more forgeable requests in 14 webapps

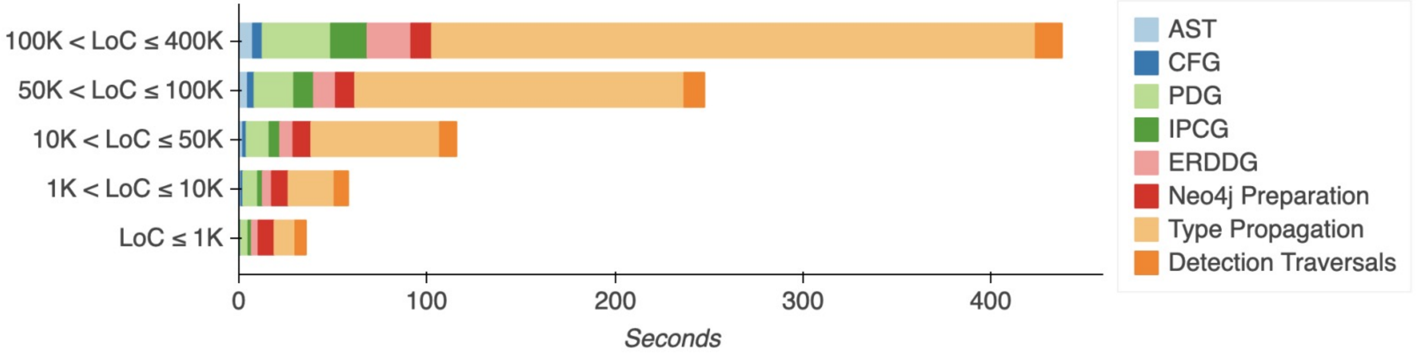
→ **+7%** forgeable requests
+19.1% vulnerable apps

JAW: Scalability and Performance

- Analysis time depends on lines of code and its complexity
  - i.e., control and data dependencies
  - Least time consuming: AST and Intra-procedural CFG generation
  - Most time consuming: Semantic type propagation (i.e., data flow analysis)

Runtime Configuration

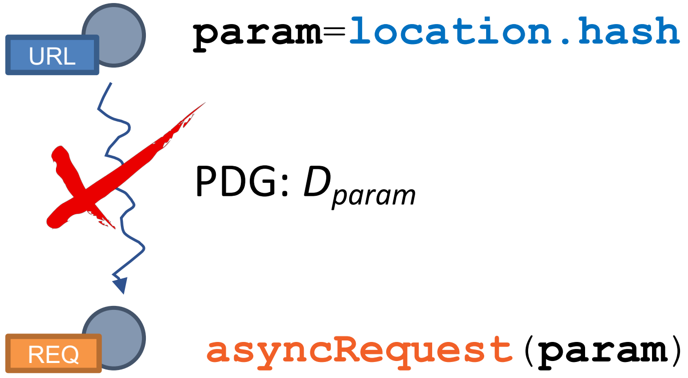
Ubuntu 18.04 on an
Intel(R) Xeon(R) CPU E5-2695 v4
with 2.10 GHz and 72 cores, 252 GB RAM



Client-side CSRF: Defenses

Problem:

⚠ Misplaced trust in unsafe input components (e.g., URL)



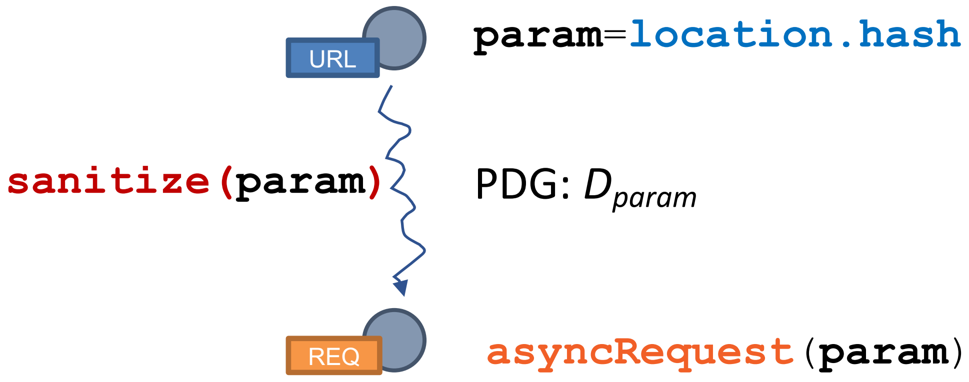
Independent Requests

- Do not use JavaScript input sources to generate HTTP requests
- Use a safelist instead
  - A pre-defined list of safe request data (e.g., endpoints)
  - Switch parameter from input to select an option from the list

Client-side CSRF: Defenses

Problem:

⚠ Misplaced trust in unsafe input components (e.g., URL)



Independent Requests

- Do not use JavaScript input sources to generate HTTP requests
- Use a safelist instead
  - A pre-defined list of safe request data (e.g., endpoints)
  - Switch parameter from input to select an option from the list

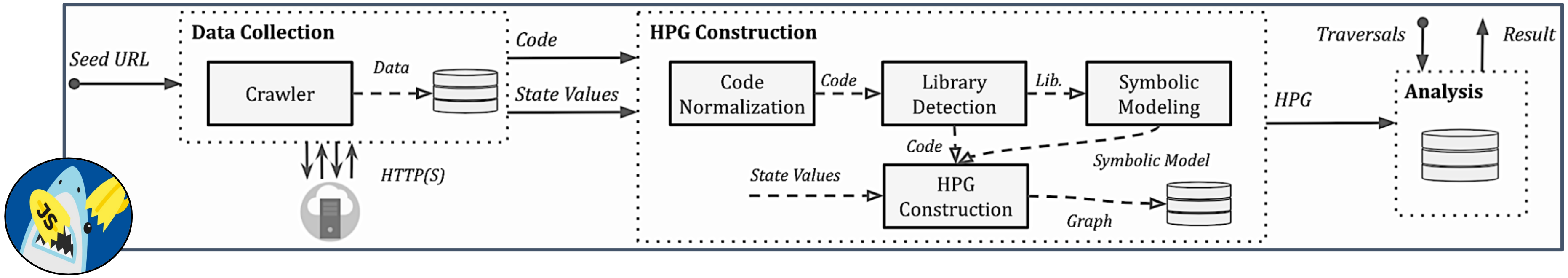
Input Validation

- Validate JavaScript input sources before using them in security-sensitive requests
- Pre-define route structures and process URL params
  - E.g., using modern client-side router libraries like Angular/Backbone/React

JAW: Security Analysis Beyond Client-side CSRF

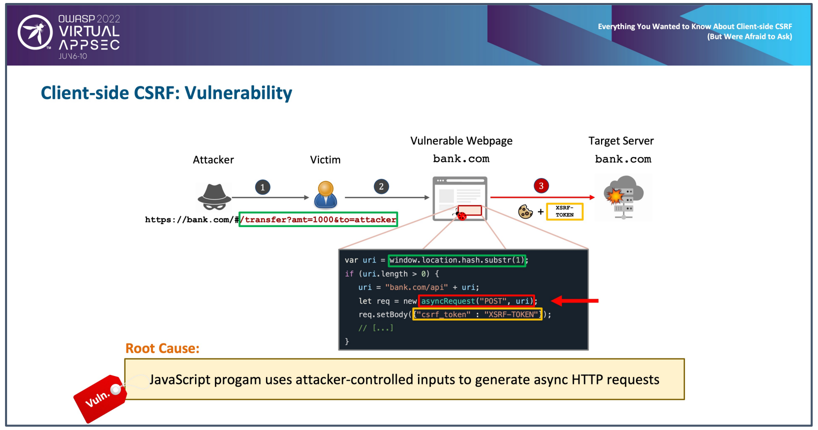
- Support for additional vulnerability classes
  - Possible to define your own semantic types
  - Detecting taint-style vulnerabilities, e.g., client-side XSS

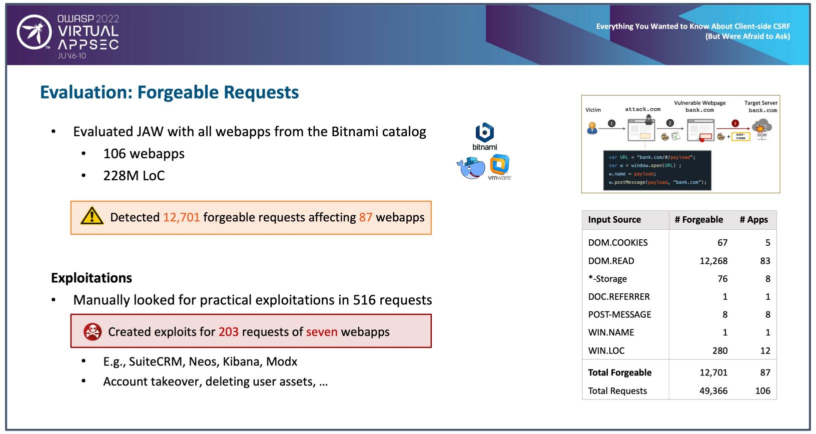
```
Q ={n : isDeclOrStmt(n) ∧ ∃c1, c2, c1 != c2
        ∧ hasChild(n, c1) ∧ hasSemType(c1, “sinkSemType”),
        ∧ hasChild(n, c2) ∧ hasSemType(c2, “sourceSemType”)
}
```

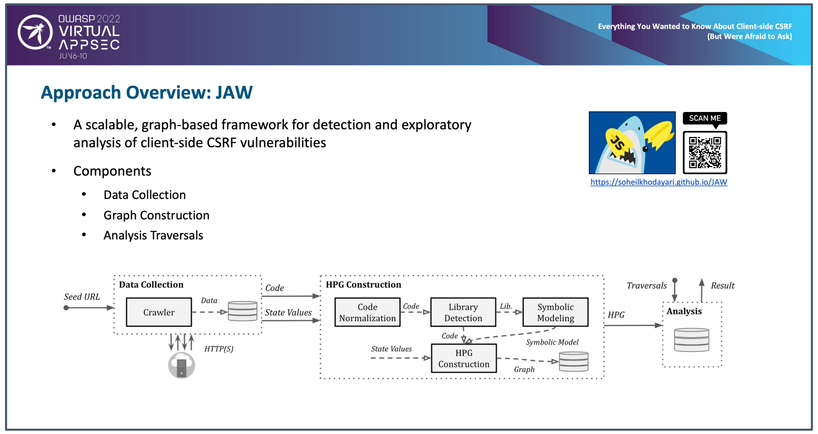


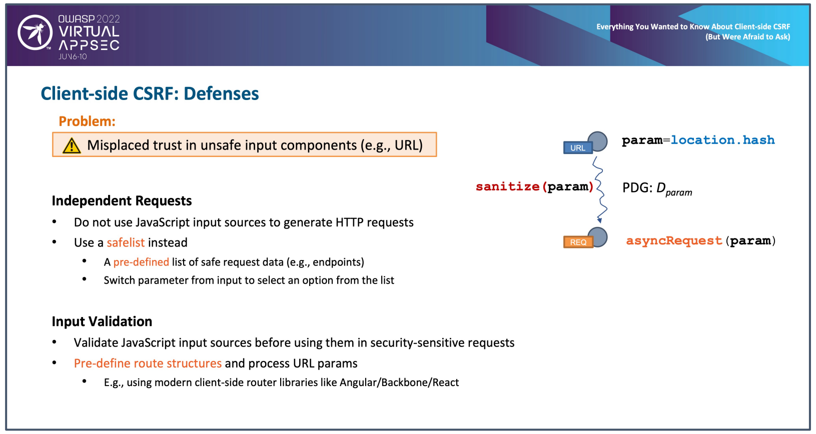
Conclusion

https://soheilkhodayari.github.io/JAW









SCAN ME

@Soheil__K

Thank You!