JAW: Studying Client-side CSRF with Hybrid Property Graphs and Declarative Traversals

Soheil Khodayari and Giancarlo Pellegrino
CISPA Helmholtz Center for Information Security

30th USENIX Security Symposium
August 11-13, 2021
Cross-Site Request Forgery (CSRF)

Robust defenses already known:
• Referrer/Origin Checks
  \[\text{getOrigin}(req_2) \neq \text{bank.com}\]
• Hard-to-guess tokens
  \[\text{getToken}(req_2) \neq \text{CSRFToken}\]
• SameSite Cookies
  • \text{SameSite}=\text{Lax} by default
  \[\text{isAuthenticated}(req_2) \neq \text{True}\]
Cross-Site Request Forgery (CSRF)

Robust defenses already known:
- Referrer/Origin Checks
- Hard-to-guess tokens
- SameSite Cookies
  - SameSite=Lax by default
- isAuthenticated(req2) != True

Are CSRF attacks solved?
Client-side CSRF

1. Attacker sends a link to Victim:
   
   `https://bank.com/#/transfer?amt=1000&to=attacker`

2. Victim clicks the link:
   
   The `window.location.hash` is extracted and checked for a length greater than 0.

3. If true, a request is made to the bank.com API:
   
   ```javascript
   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"});
     // [...]
   }
   ```

Target Server

Vulnerable Webpage
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) Attacker models and exploitations?
  - (RQ3) Degree of attacker control?
    - E.g., path, query, domain, body

**Challenge:** analysis of JavaScript programs to study client-side CSRF is not an easy task.

\(^1\) **Source:** facebook.com/notes/996734990846339/
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

https://soheilkhodayari.github.io/JAW
A scalable, graph-based framework for detection and exploratory analysis of client-side CSRF vulnerabilities

Components

- Data Collection
- Graph Construction
- Analysis Traversals

Approach Overview: JAW

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

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

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++
[Yamaguchi et al, S&P’14]

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

HPG for JavaScript
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 JS libraries

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

HPG for lib.js

Symbolic Model for lib.js

```
<script src="lib.js"/>
```
Evaluation: Forgeable Requests

- Evaluated JAW with all webapps from the Bitnami catalog
  - 106 webapps
  - 228M LoC
- Detected 12,701 forgeable requests affecting 87 webapps

Exploitations

- Manually looked for practical exploitations in 516 requests
- Created exploits for 203 requests of seven webapps
  - SuiteCRM, SugarCRM, Neos, Kibana, Modx, Odoo, and Shopware
  - Account takeover, deleting user assets, ...

<table>
<thead>
<tr>
<th>Input Source</th>
<th>Forgeable</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOM.COOKIES</td>
<td>67</td>
<td>5</td>
</tr>
<tr>
<td>DOM.READ</td>
<td>12,268</td>
<td>83</td>
</tr>
<tr>
<td>*-Storage</td>
<td>76</td>
<td>8</td>
</tr>
<tr>
<td>DOC.REFERRER</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>POST-MESSAGE</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WIN.NAME</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WIN.LOC</td>
<td>280</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total Forgeable</strong></td>
<td><strong>12,701</strong></td>
<td><strong>87</strong></td>
</tr>
<tr>
<td><strong>Total Requests</strong></td>
<td><strong>49,366</strong></td>
<td><strong>106</strong></td>
</tr>
</tbody>
</table>
Evaluation: Analysis 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 the paper for more

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

https://soheilkhodayari.github.io/JAW

Client-side CSRF

Vulnerable Webpage: bank.com

Target Server: bank.com

Attacker

Victim

https://bank.com/transfer#authenticate

```javascript
var url = decodeURIComponent(unsafeLocation.href);
if (url.length > 8) { 
  url = url.replace(/\?.*/, '');
  let req = new XMLHttpRequest();
  req.open('POST', url, true);
  req.send('nonce=' + Math.random() + '');//  // Send the nonce to the server
}
```

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 (IPC)
  - Event Registration, Dispatch and Dependency Graph (ERDDG)
  - Semantic Types and Symbolic Models

- **State Values (Dynamic)**
  - Event Traces
  - Environment Properties

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

Evaluation: Forgeable Requests

- Evaluated JAW with all webapps from the Bitnami catalog
  - 106 webapps
  - 228M LoC
- Detected 12,701 forgeable requests affecting 87 webapps

Exploitations

- Manually looked for practical exploitations in 516 requests
- Created exploits for 203 requests of seven webapps
  - SuiteCRM, SugarCRM, Neos, Kibana, Modx, Odoo, and Shopware
  - Account takeover, deleting user assets, ...

Thank You!

@Soheil__K