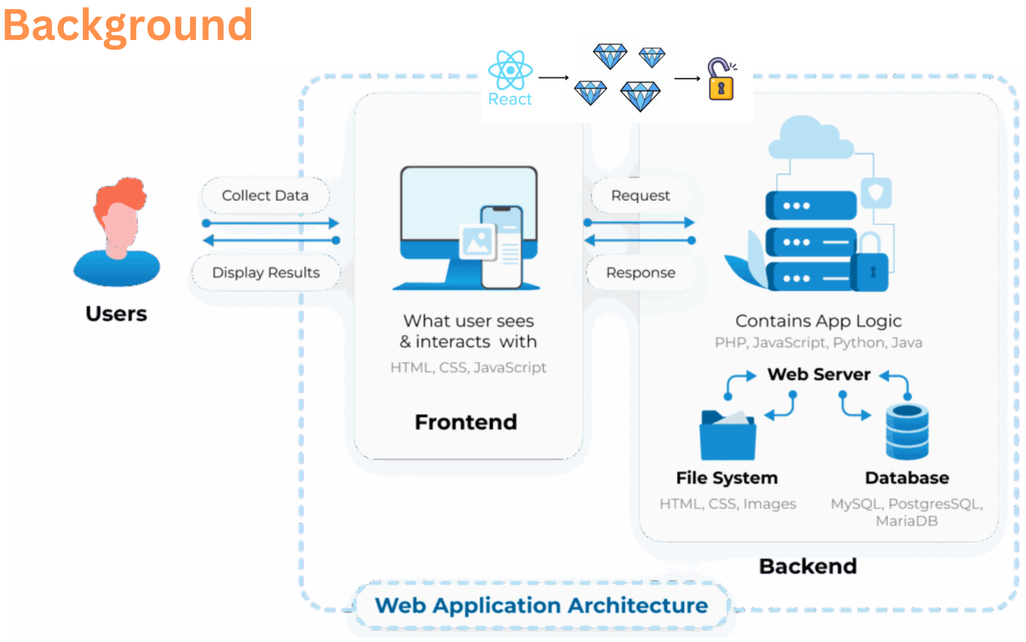
Web GEMs: Broken Access Control Vulnerabilities (BAC) in Large Web Front-Ends. An Empirical Study

Nicolò CAVALLI, Arnaud BLOUIN, Djamel KHELLADI, Olivier BARAIS

Background



- BAC issues are the most common on the web (OWASP A01, 2021)
- Component-based JS frameworks generate obfuscated and minified production code
- **Hypothesis**: devs may considered safe to disclose sensitive data to the front-end (ID, token) or let it directly enforce RBAC/ABAC policies
- Framework internal APIs are often **exposed in prod**, allowing an attacker to tamper with the component tree, extracting sensitive data and bypass weak server-side checks

Vulnerability condition

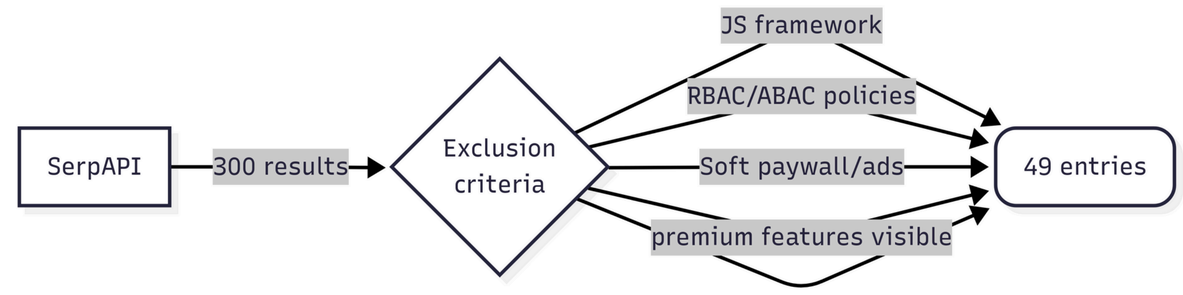
- **Overpowered front-end (OP)**. At least 1 feature or resource already available to the front-end for which the front-end applies all access control policies
- **Exposed resources (ER)**. At least 1 network request directed to a protected resource or feature that is directly accepted by the web server (IDOR, BOLA, MFLAC)
- **Overtrusted front-end (OT)**. At least 1 network request that is accepted exclusively on the user permission declared by the front-end (HTTP parameter tampering)

RQ1 JS frameworks comparative analysis

TABLE 1. BDT ACCESS TO JS FRAMEWORK INTERNAL APIS. A COMPARISON (N/A = NOT APPLICABLE, † = NEW FINDING))

		BDT Access			Security	
Framework	**Version**	**Built-in**	**Development**	**Production**	**Patch available**	**Revertible**
Angular	19.2.0	✓	✓	✗	N/A	✗
Ember	6.7.0	✓	✓	✓	✓†	✗
React	19.0.0	✓	✓	✓	✓ [23]–[25]	✓†
Solid	1.9.5	✗	✓	✗	N/A	✗
Svelte	5.39.10	✗	N/A	N/A	N/A	N/A
Vue	3.5.1	✓	✓	✗	N/A	✓

RQ2 Prevalence and impact



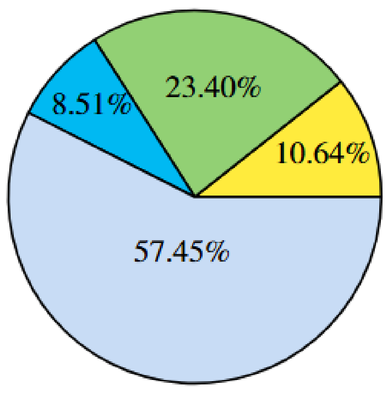


Fig 1: Distribution of vulnerable applications per category

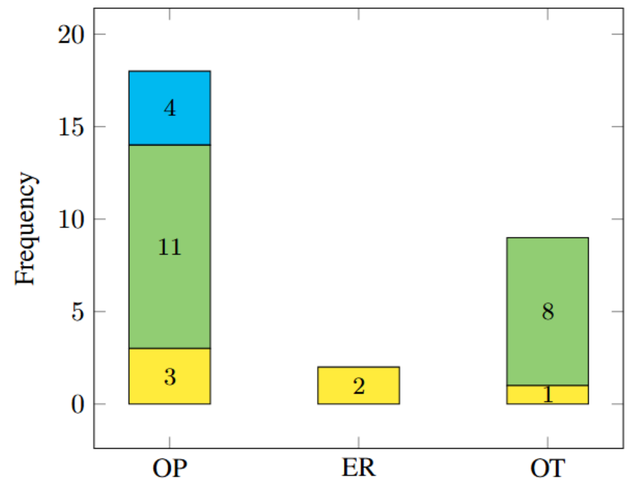


Fig.2 Distribution of vulnerability cases

RQ3 Countermeasures and best practices

Performing server-side checks on redirects, implementing incremental authorization techniques rather, keeping permission values under back-end control, securing at least critical functionality if major refactoring is not possible

Contributions

- **3** formal conditions for web GEM vulnerability
- **20** vulnerabilities on 49 entries, **3** official ack,
- React, Vue, Ember found vulnerable to JCH with no effective mitigation
- **3** case studies
- State-of-art web scanners (Burp, OWASP ZAP) **not** being **capable** of **detecting web GEM-based IDOR**
- Provided mitigation and best practices derived from non vulnerable apps

Future work

- Large scale analysis on Tranco's top 100k
- LLM/ML based detection of sensitive HTTP bodies