Thou Shalt Not Depend on Me:
Analyzing the Use of Outdated JavaScript Libraries on the Web
Date 2020/9/23
Overview

• Explores client-side JavaScript library usage and the resulting security implications
• Crawls data from 133k websites and tracks causal inclusion relationships using causality trees
• Finds out
  • prevalence of vulnerable library inclusions
  • libraries are used in unexpected ways
• Provides suggestions to improve the situation
Background Summary

• A JavaScript library is a plain-text script containing code with reasonably well-defined functionality.
• JavaScript libraries overwhelmingly use the Semantic Versioning convention of major.minor.patch.
• Two ways of including a library:
  • CDNs
  • Third Parties: advertising, trackers, social media or other widgets
• Vulnerabilities
  • Cross-Site Scripting (XSS)
  • $() function in jQuery
Question: Do you know any other JavaScript vulnerabilities or website vulnerabilities (not necessarily JavaScript)?

- Cross-site scripting (XSS) vulnerabilities
- Cross-site request forgery (CSRF)
- SQL injection
- Clickjacking / Framebusting
- Abusing Javascript language components/functionality
- Cache poisoning

- Vulnerable software: Sqlite, webassembly
Question: How does this situation affect users? Why should we care about JavaScript library security?

• Leak user private data
• Web-based fraud
• Introduce potential for malware
• Prevent users from using website functionalities
• Degrades on trust on the web

Site incentives: want latest versions, lower trust in the side may lead to fewer users/customers => less revenue

Site disincentives: want stability (backwards compatibility), vulnerable != exploitable, minimize development cost/maintenance
Methodology

• Cataloging JavaScript Libraries
  • collect meta data about popular JavaScript libraries
• Library Identification
  • determine if JavaScript code found in the wild is a known library.
• Data Collection
  • crawl websites
  • keep track of causal resource inclusion relationships
• Validation
Methodology: Cataloging JavaScript Libraries

• Selecting Libraries: Bower + Wappalyzer + public CDNs
  • Bower: JavaScript package manager
  • Wappalyzer: web technology survey

• Extracting Versioning Information: Github
  • No enough information on official websites!

• Obtaining Reference Files: download from official sites and CDNs

• Identifying Vulnerabilities: compile vulnerability information from databases and public platforms such as Github comments
Methodology: Library Identification

• Static Detection
  • compute file hashes

• Dynamic Detection
  • fingerprints of JavaScript run time environment (global variables)
  • library themselves (variable or function that return the version)
Question: Are the two detection methods enough for library identification? Are there any other possible method?

- Name in URL (e.g., does jQuery appear in the URL)
  - Validation: not so good 😞

- Content comparison (allowing for partial overlap)
Methodology: Data Collection

- Uses causality tree to identify
  - whether a library exists
  - how is it loaded (direct or indirect)

- Causality tree
  - snapshots of elements in the DOM tree at specific points in time

- Crawl websites
Methodology: Validation

• To prove that the detection methods work
• Dynamic detection
  • identify the name and version of 79.2% of the libraries
  • identify the name of 18.6% of the libraries
• Static Detection
  • low detection rate
  • because developers deploy customized versions of libraries
• Hypothetical “Name-in-URL” Detection
  • not robust enough
Question: How convincing is the validation? Are there other validation methods?

• False positives from dynamic detection due JS “pretending” to be a particular library or version

• Cross-comparison between methods (using one method’s results as ground truth)

• Validation focused on a small set of libraries
Analysis

• General overview of the data set
  • Alexa contains a larger share of more complex websites
  • Most scripts are directly included, and documents are more frequently dynamically generated
  • The most common script type are inline scripts
Analysis

• Highly-ranked websites tend to be less likely to include vulnerable libraries, but they are also less likely to include any detected library at all.

• Financial and governmental websites rank last with 52% and 50% vulnerable sites

• Malicious websites (e.g., spam) have the same proportion as the full data set

• Parked and adult sites are the least vulnerable
Question: What are the possible explanations of this phenomenon (highly-ranked sites, financial sites are less vulnerable, etc.)?

• Highly ranked:
  • These site use custom/closed-source libraries
  • More resources to better develop/secure their sites

• Parked + adult were least vulnerable, financial/gov’t most vulnerable among top sites, common sites the most vulnerable overall:
  • Financial/gov’t do have to comply w/ policies and care about security, so are somewhat less vulnerable
  • Gov’t sites might have lack of funding and bureaucracy
  • Malicious sites might be vulnerable on purpose as part of the attack strategy
  • Adult sites are purely online so highly dependent on developing that well, they also make money and want to their content, more security-oriented mindset, might have lots of resources
  • Parked websites are probably pretty sparse to start off with
Analysis

• Risk factors
  • direct inclusions are less likely to be vulnerable than indirect
  • inclusions by ad, widget or tracker code appear to be more vulnerable
• Unexpected, duplicate library inclusions
  • same version of library is loaded multiple times
  • different version are loaded into the same document
  • developers rarely use remediation like version patch-level updates to
Question: Why do developers use outdated libraries? The possible reasons that lead to this situation?

• Used to the current (outdated) version
• Upgrading is painful  
  • Finding updated version can be hard  
  • Compatibility issues with new library version  
  • Developers/web administrators just don’t know
• Existing tools used by developers may depend on outdated libraries
• Websites may not be actively developed and maintained
• Website development prioritizes features/functionality/bug fixes
• External dependencies (3\text{rd} party plugins) are not under developer control
Question: Can the situation be improved? How?

• Incentivizing website developers/administrators
  • Insurance

• Developer/administrator training/education/information
  • Notification about outdated libraries

• Centralization of libraries, documentation (package management ecosystem)

• Make JS more safe to use