A Longitudinal Study of Vulnerable Client-side Resources and Web Developers' Updating Behaviors

Kyungchan Lim
University of Tennessee
Knoxville, USA
klim7@utk.edu

Yonghwi Kwon
University of Maryland
College Park, USA
yongkwon@umd.edu

Doowon Kim
University of Tennessee
Knoxville, USA
doowon@utk.edu

ABSTRACT

Modern Websites commonly rely on various client-side web resources, such as JavaScript libraries, to provide rich and interactive end-user web experiences. Unfortunately, anecdotal evidence shows that improperly managed client-side resources could open up attack surfaces. However, there is still a lack of a comprehensive understanding of the updating practices among web developers and the potential impact of inaccuracies in Common Vulnerabilities and Exposures (CVE) information on the security of the web ecosystem. In this paper, we conduct a longitudinal (four-year) measurement study of the security practices and implications on client-side resources (e.g., JavaScript libraries and Adobe Flash) across the Web. Specifically, we collect a large-scale dataset of 157.2M webpages of Alexa Top 1M websites for four years in the wild. We find an average of 41.2% of websites (in each year of the four years) carry at least one vulnerable client-side resource (e.g., JavaScript or Adobe Flash). Worse, we observe that vulnerable JavaScript library versions are persistently observed in the wild, even after months of the release of their security patches. On average, we observe 531.2 days with 25,337 websites of the window of vulnerability, which can be mitigated by simply applying the released security patches immediately. Furthermore, we manually investigate the fidelity of CVE (Common Vulnerabilities and Exposures) reports on client-side resources, leveraging PoC (Proof of Concept) code. We find that 13 CVE reports (out of 27) have incorrect vulnerable version information, which may mislead security-related tasks such as security updates.

CCS CONCEPTS

• **Security and privacy → Web application security**.

KEYWORDS

Web Security, JavaScript Library, Adobe Flash, CVE

ACM Reference Format:
Kyungchan Lim, Yonghwi Kwon, and Doowon Kim. 2023. A Longitudinal Study of Vulnerable Client-side Resources and Web Developers' Updating Behaviors. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3618257.3624804

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10…$15.00
https://doi.org/10.1145/3618257.3624804

1 INTRODUCTION

The Internet (especially the Web) has become an essential part of our lives as the number of Internet users worldwide in 2021 was 4.9 billion, about two-thirds of the global population [42, 103]. Websites leverage various techniques, such as client-side scripting (JavaScript) and Cascading Style Sheets (CSS), to provide rich content to users. In particular, since its release in 1997 [1], JavaScript has been a core technique for implementing dynamic website features. To improve the productivity of development, the community has created various JavaScript libraries such as `jQuery` and `Bootstrap`, which now have become an essential part of the websites [48].

Unfortunately, those libraries often cause security concerns. Vulnerabilities of the widely-used libraries have been revealed and publicly reported. For example, CVE-2018-9206[1] (with a high base score of 9.8[2]) [91] was found in a `jQuery` plugin, called `jQuery-File-Upload` where adversaries can upload malicious files (*e.g.*, backdoors) on servers [118]. Worse, the vulnerable `jQuery` plugin was integrated into other high-profile Web projects (with hundreds of millions of websites), such as `WordPress`, `Drupal`, and `Joomla`, resulting in a significant security impact on the Web ecosystem.

To address such security concerns, the best practice for website developers is to update vulnerable client-side resources such as JavaScript libraries. Specifically, developers may need to check the CVE (Common Vulnerabilities and Exposures) database to identify and patch vulnerable resources, as CVE is considered the most reliable vulnerability information for the public (including web developers). A CVE report includes what and where a vulnerability is found (*e.g.*, which library version is vulnerable) and how it has been addressed (*e.g.*, which library version has patched the vulnerability), including the timeline of the actions. Unfortunately, it is known that CVE reports in practice are often inaccurate [68]. However, it is less clear how the potential security impact and implication of the quality of the CVE information for the security of client-side web resources on the entire web ecosystem. This motivates us to raise a research question, how does the accuracy of the CVE information impact the security of the web ecosystem?

Prior work [62, 82–84, 90, 92, 98, 100, 105, 109, 117] has studied security issues of the client-side web resources (*e.g.*, JavaScript). Notably, many technical reports [14, 18, 38] cursorily revealed that

[1] CVE (Common Vulnerabilities and Exposures) is a standardized system that assigns unique identifiers to known security vulnerabilities, facilitating their tracking and management.
[2] CVSS (Common Vulnerability Scoring System) score, is a numerical value that quantifies the severity and potential impact of a specific vulnerability.

not every website consistently maintains the latest secure client-side resources, particularly JavaScript libraries [82]. However, the studies provide a limited perspective as they relied on a *single snapshot* of dataset collection on a certain date, not from observation for a longer period. Without a longitudinal analysis, the scope and depth of the studies' findings are also limited. For example, a single snapshot cannot answer the question of how promptly (*e.g.*, how many days) the vulnerable client-side resources have been updated – *i.e.*, a window of vulnerability.

In this paper, we design a systematic, longitudinal measurement study on the Alexa Top 1M websites for four years (Mar. 2018 – Feb. 2022) to understand the security practices and implications of client-side resources in the Web ecosystem. Specifically, among the client-side resources, we mainly focus on analyzing the two resources (JavaScript libraries and Adobe Flash applet) because the two resources contain more critical security vulnerabilities than other resources (*e.g.*, CSS, XML, Favicon, etc.) and can have a greater impact on the security of the Web ecosystem than other resources do. With an emphasis on two client-side resources (JavaScript libraries and Adobe Flash applets), we particularly measure (1) how the two vulnerable resources impact the Web ecosystem, (2) the updating practices of insecure websites, and (3) the accuracy of the CVE information.

Our study reveals that numerous websites in the wild *still* use outdated, discontinued, vulnerable client-side resources (see Sections 6.3, 6.5, and 8), significantly impacting the security of the Web ecosystem.[3] For instance, an outdated and vulnerable `jQuery` version (v1.12.4) has been dominant for four years. In other words, even popular vulnerable libraries are not updated in a timely manner (see Section 7); specifically, on average, it takes 531.2 days (17.4 months) to update the vulnerable version. Furthermore, we find that many CVE reports contain *inaccurate* information (*e.g.*, the vulnerable version information), potentially misinforming the public and making it difficult to maintain websites secure (see Section 6.4).

Our contributions are summarized as follows:

- We conduct a longitudinal study on the security implications of client-side resources in the wild for four years (Mar. 2018 – Feb. 2022) using our collected 157.2M landing webpages of Alexa Top 1M websites.
- We study insecure websites due to the vulnerable client-side resources in the wild. *First*, an average of 41.2% of websites have carried at least one vulnerability for four years. *Second*, the vulnerable client-side resources in such websites are updated with significant delays to the latest versions; an average of 531.2 days are taken to update the vulnerable versions.
- We reveal that many CVE reports contain inaccurate version information, which may cause significant delays in updates. On average, it takes 701.2 days to update vulnerable versions of JavaScript libraries with the *understated* inaccurate CVE information, compared to 510 days estimated to take to update the versions if the correct CVE information is given.
- We discuss recommendations to enhance the security of client-side resources in the Web ecosystem and share our source code and our four-year data collection of Alexa top 1M domains at "https://moa-lab.net/measurement-client-side-resources/", in order to facilitate future research in the community.

[3]Note that the severity and security impact of the vulnerabilities of outdated resources can vary. Some vulnerabilities can be exploited only under specific conditions. Further details will be discussed in Section 9.

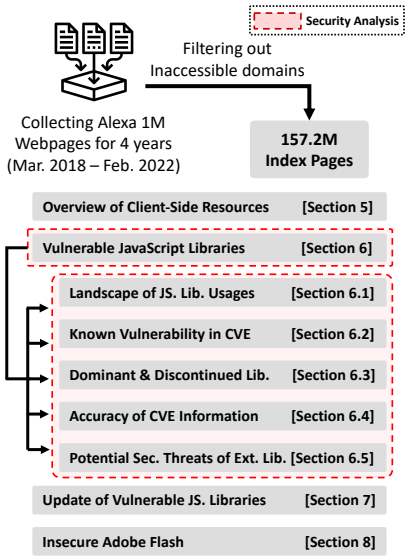


Figure 1: Overview of Our Security Analysis Study on Client-side Resources.

2 BACKGROUND

This section provides a brief overview of two client-side resources of our focus (JavaScript library and Adobe Flash) that could have vulnerabilities and lead to security issues.

2.1 JavaScript Library

Web developers often use JavaScript libraries which are essentially a library of pre-written code, providing common functionalities. For example, `jQuery` [76] is one of the most popular JavaScript libraries that help simplify HTML DOM tree manipulation and traversal, CSS animation, etc.

Versioning. JavaScript library projects typically use Semantic Versioning [108] where a version consists of `MAJOR`.`MINOR`.`PATCH` (*e.g.*, 2.1.12). The patch and minor versions increase when bugs are fixed and new features are added, respectively; these do not change their public APIs. The major version is for significant changes to libraries (*e.g.*, public API interface changes causing incompatibility). The version information typically is observable in the library's URL (as a part of a file name or in a URL path).

Delivery of Externally-Hosted Library. The content delivery network (CDN) is a common technique used to efficiently deliver externally-hosted JavaScript libraries to clients. Essentially, it is a geographically distributed network service that delivers content from servers close to each end-user, speeding up web content delivery by reducing the physical distance between a server and an end-user. Popular JavaScript library projects (*e.g.*, `jQuery`) often have their own CDNs, while other open source projects are hosted on the free and public CDNs (*e.g.*, cdnjs [65] and JSDelivr [80]).

Vulnerability Reporting and Patching. Vulnerabilities in JavaScript libraries are typically reported to the CVE system; each vulnerability is assigned a unique CVE identifier and publicly released. For example, CVE-2020-11022 [30] describes that `jQuery` versions v1.2 – v3.5.0 (excluding v3.5.0), had a vulnerability that could execute untrusted code due to a buggy regular expression in its DOM manipulation method. A new version of `jQuery` was released with the fix of the vulnerability. Web developers are recommended to update `jQuery` to the new version.

Security Best Practice for Web Developers. Web developers are strongly advised to *regularly update* the JavaScript libraries used in their websites. Outdated JavaScript libraries may contain security loopholes that can be exploited by adversaries. Another critical security threat can arise when externally hosted JavaScript libraries are compromised to deliver malicious payloads. To prevent such security threats, web developers are recommended to use *Subresource Integrity* (SRI) [89]. Specifically, web developers specify the hash value of JavaScript libraries in the `integrity` attribute of the `<script>` tag. `<link>` tag also supports the `integrity` attribute for the CSS files. Then, the web browser checks the hash values in the `integrity` attribute with the downloaded client-side resources' hash values, to ensure the resources are not modified (*i.e.*, compromised).

2.2 Adobe Flash Applet

Since its introduction in 1996, Adobe Flash has become popular in delivering dynamic multimedia web content across all browsers and platforms [70]. Under the hood, Adobe Flash Player, a web browser plug-in, runs Flash applet files (`.swf`). Unfortunately, it has become significantly attractive to adversaries: 1,118 CVEs have been publicly reported as of May 26, 2023[4] [2–6, 8, 12, 16, 19, 20]. Anecdotal evidence has shown that the Adobe Flash vulnerabilities helped adversaries successfully penetrate the RSA network in 2011 [71, 81].

Adobe publicly announced that Flash was no longer officially supported after Dec. 31, 2020 [56]. Adobe asked developers to replace it with HTML5 and end-users to *uninstall* it for safety reasons. Accordingly, all major web browsers officially removed the Flash Player components in Jan. 2021 [66, 73, 88]. Windows also removed Adobe Flash products through Windows Update in Jan. 2021 [49]. This essentially indicates that the Web is recommended not to use Flash.

3 MOTIVATION & RESEARCH QUESTION

Modern websites intensively rely on client-side web resources such as JavaScript libraries to provide dynamic, rich web experience and content to clients. Unfortunately, adversaries have been able to conduct attacks against clients by exploiting vulnerabilities of the client-side web resources (*e.g.*, a vulnerability, CVE-2018-9206 [91], in `jQuery` enabled adversaries to install backdoors).

While there have been prior studies that analyzed the various security issues of client-side web resources [62, 82–84, 90, 92, 98, 100, 105, 109, 117], the analysis was based on *a single snapshot* of data collection (*e.g.*, on a certain day), which limited its scope and depth in understanding the entire client-side resource ecosystem. Particularly, there is a lack of understanding of how web developers reactively address vulnerabilities in client-side resources; for example, how long it has taken for web developers to update the vulnerable versions.

Moreover, we empirically observe that some websites continue to employ Adobe Flash applications and a certain platform application actively promotes the use of Adobe Flash, despite the discontinuation of official support. However, the reasons behind the persistent usage of Adobe Flash and the contributing factors in these contexts remain relatively unexplored.

To this end, to better understand the entire ecosystem and web developers' security behaviors, we design a systematic, *longitudinal* study of the security impact on the client-side resources with an emphasis on the web developers' updating behaviors, the accuracy of CVE information, and the persistent usage of Adobe Flash.

Our Research Questions. We aim to understand the longitudinal security issues and implications of the client-side web resource ecosystem from various security perspectives. In particular, we aim to answer the following research questions.

- **RQ1)** How prevalent are insecure websites in the wild due to vulnerable JavaScript libraries?
- **RQ2)** How quickly do the insecure websites (*i.e.*, web developers) react to their vulnerable client-side resources and update them upon the release of the vulnerabilities and patches?
- **RQ3)** How does the accuracy of the CVE report impact the security of websites and developers' updating behaviors of vulnerable client-side resources as they rely on CVE information?
- **RQ4)** How does Adobe Flash exhibit vulnerabilities and what factors contribute to these vulnerabilities?

Research Focus. Figure 1 provides an overview and focus of our measurement study. Note that we focus on the *two client-side resources* (JavaScript libraries and Adobe Flash applets) because JavaScript libraries are the most used client-side resources on the Web. Also, the two resources contain more critical security vulnerabilities than other resources (*e.g.*, CSS, XML, Favicon, etc.) and can have a greater impact on the security of the Web ecosystem than other resources do. Note that for JavaScript libraries, we only focus on the libraries for the client-side environment, *not* those for the server-side environment such as `Node.js`.

4 DATASET COLLECTION

Our web crawler is written in the `Go` programming language. The crawler 1) periodically accesses Alexa 1M domains, 2) downloads the landing page (*e.g.*, `index.html`) from each domain, and 3) identifies client-side resources and their versions from the downloaded webpages. We also collect the vulnerability information of affected resources' versions from multiple third-party websites for cross-reference, such as the National Vulnerability Database (NVD) [45], CVE MITRE Corporation [40], cvedetails.com [41], and SNYK Vul. DB [50].

4.1 Landing Page Collection

We collect the webpages of Alexa Top 1M domains[5] on a *weekly* basis for *four years* (Mar. 2018 to Feb. 2022; 207 weeks); specifically,

[4] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=adobe+flash

[5] We utilize the single snapshot of the Alexa Top 1M domains of Mar. 2018.

our Web crawler (implemented in `Go` using the `net/http` library) visits each Alexa 1M domain over HTTPS and collects the landing page of each domain every week. During the four-year data collection, we had very few times experienced network issues (*e.g.*, unavailable network connection). We prune out 6 snapshots (out of 207 snapshots) and remove inaccessible domains from the collected dataset.

Filtering Inaccessible Domains. As expected, for four years (our collection period), we observe a number of inaccessible domains due to expired domains or unstable web servers. Particularly, we observe some of the lower-ranked domains are unstable in providing their services to clients. Moreover, we occasionally observe empty HTML pages and anti-crawling-bots blocking techniques (*e.g.*, '4xx' error code).

As such inaccessible domains may introduce bias into our analysis of the collected dataset, we conservatively remove them from our collected dataset. Specifically, we filter out the domains responding with error pages (*e.g.*, with '4xx' error status code) or empty pages (less than 400 bytes) for the *four consecutive weeks* in the *last month* of our data collection period. The reason why we consider 400 bytes as a threshold for the empty/error pages is that we manually check all HTML pages with less than 400 bytes and observe that all of them are either error or empty pages. Note that we manually check all of such pages in our dataset and confirm that they do not contain content related to the website's original purpose (*i.e.*, service). Instead, they present error messages (with 200 status codes) from anti-crawling-bots blocking techniques, saying "Not allowed to access." To this end, we eventually collect 157,242,243 (157.2 M) HTML files for the 201 weeks of the four years. On average, we consistently collect the index pages from the 782,300 domains (*i.e.*, websites) every week, as shown in Figure 2(a). The proportion of domains that we successfully collected is similar to what has been reported in prior work [99, 104, 105].

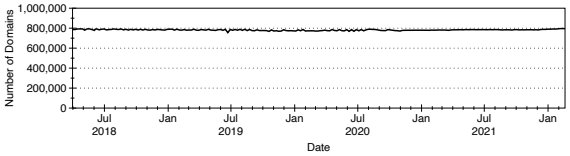
4.2 Identifying Resources and Versions

To better understand the current Web ecosystem, we first need to know what client-side resources (*e.g.*, JavaScript libraries, Adobe Flash applets, etc.) are used in each website from our collected HTML files and identify their versions from the resources. The identifications of client-side resources and their versions can help answer our research questions (RQ1, RQ2, and RQ4). We utilize a website profiling tool, called `Wappalyzer` [51] that is widely used in prior work [61, 67, 69, 74, 85, 96, 106] and can identify client-side resources and their versions on webpages. Specifically, given a static HTML file, `Wappalyzer` uses regular expressions to identify the client-side web resources such as JavaScript libraries, Adobe Flash, CSS, and their versions in the given HTML file.

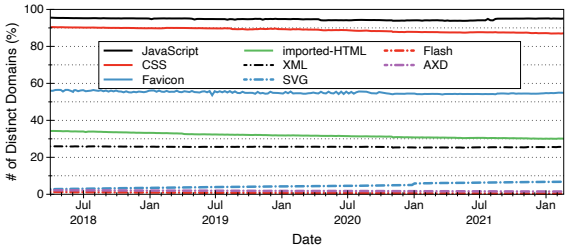
4.3 Collecting Vulnerability Information

After identifying client-side resources and their versions, we attempt to identify vulnerabilities in each version of the client-side resources and collect the vulnerability information to answer RQ1, RQ2, RQ3, and RQ4. Since there is no centralized database for vulnerabilities, we manually search each client-side resource with its versions and collect information on vulnerabilities[6] from National Vulnerability Database (NVD) [45], CVE MITRE Corporation [40], cvedetails.com [41], and SNYK Vul. DB [50].

[6]We focus on only vulnerabilities in conjunction with our collection period.



(a) Our Collected Websites for Four Years.



(b) Top 8 Client-side Resource Usages (%).

Figure 2: Our Collected Websites & Top 8 Client-side Resource Usages. (a) On average, 782,300 websites are collected every week for four years (201 weeks). (b) On average, 94.7% of websites (740,823 out of 782,300) include JavaScript.

5 OVERVIEW OF RESOURCES

From our collected dataset, we first measure various types of client-side resources that have been used in the web ecosystem. Figure 2(b) shows the top 8 client-side resources. JavaScript is the most used resource in the wild; on average, 94.7% of websites have at least more than one embedded client-side JavaScript code in their HTML code or URLs of external client-side JavaScript files. CSS (88.4%) is the next most frequently used resource, followed by Favicon (55.0%), and imported-HTML[7] (31.8%). XML occupies 25.6%, and all the remainder (i.e., SVG, Adobe Flash, and AXD) account for less than 2.4%.

JavaScript Library. Of 94.7% (740,823 out of 782,300) websites that use JavaScript, 97.04% (718,895 out of 740,823) websites use JavaScript libraries, meaning that the libraries are incredibly prevalent in practice. We further investigate popular JavaScript libraries in the dataset. In total, we find 79 distinct libraries in our dataset, and Table 1 shows the top 15 of them. Specifically, 64.0% of the websites (500,364 out of 782,300) use `jQuery`, followed by `Bootstrap` (21.5%), and `jQuery-Migrate` (20.8%). We further discuss the longitudinal landscape of JavaScript libraries in Section 6.1.

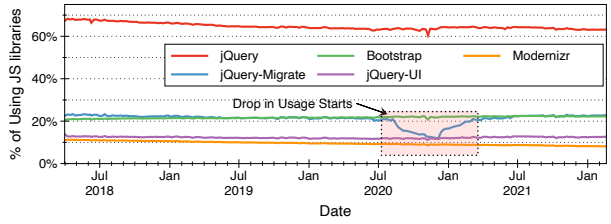
From our observations, we raise follow-up research questions regarding JavaScript libraries; **RQ1-1)** How many vulnerable JavaScript libraries are used in the wild?; **RQ2-1)** How the vulnerable JavaScript libraries are patched in practice?; **RQ3-1)** How does the CVE report impact the practices for securing websites? In Section 6, we aim to answer those research questions using our dataset.

[7]We infer this from the file extension (`.php`) in HTML tags (e.g., `<script>` or `<link>`). Note that those PHP scripts are used to dynamically generate client-side resources such as JavaScript and CSS.

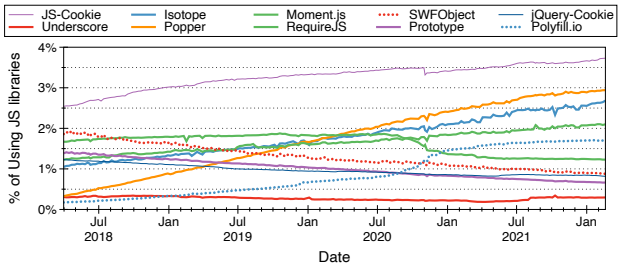
Library	Avg. Usage (%)	Inclusion Type			Version				# Vul.[6]
		Avg. Int.[1]	Avg. Ext.[1]	Avg. CDN[1,2]	Found[3]	Total[4]	Avg. Dominant	Latest[5]	
jQuery [76]	500,364 (64.0%)	**59.2% (9.2%↓)**	40.8% (3.9%↓)	**96.1%** (6.9%↓)	81	81	v1.12.4 (14.7%)	v3.6.0	**8**
Bootstrap [63]	168,088 (21.5%)	**71.6% (8.2%↓)**	28.4% (11.8%↑)	70.7% (10.3%↑)	62	64	v3.3.7 (24.3%)	v5.1.3	**7**
jQuery-Migrate [78]	163,386 (20.8%)	**88.4% (18.1%↓)**	11.62% (44.4%↑)	42.6% (60.3%↑)	16	16	v1.4.1 (50.4%)	v3.3.2	**1**
jQuery-UI [79]	95,058 (12.2%)	49.7% (8.9%↓)	**50.3%** (18.9%↓)	**91.9%** (21.6%↓)	47	47	v1.12.1 (14.2%)	v1.13.1	**6**
Modernizr [86]	74,129 (9.5%)	**78.1% (20.1%↓)**	21.9% (18.9%↓)	68.2% (27.9%↓)	26	43	v2.6.2 (14.8%)	v3.11.8	0
JS-Cookie [44]	25,601 (3.3%)	**80.5% (14.2%↑)**	19.5% (53.3%↑)	86.5% (59.2%↑)	23	23	v2.1.4 (86.4%)	v3.0.1	0
Underscore [107]	19,614 (2.5%)	**83.2% (17.3%↑)**	16.8% (47.3%↑)	49.7% (49.2%↑)	15	75	v1.8.3 (9.5%)	v1.13.2	**1**
Isotope [75]	13,868 (1.8%)	**90.8% (6.8%↓)**	9.2% (35.1%↑)	24.6% (21.7%↑)	28	42	v3.0.4 (15.6%)	v3.0.6	0
Popper [94]	13,539 (1.7%)	46.9% (79.7%↑)	**53.1%** (80.8%↑)	**92.0%** (80.2%↑)	46	133	v1.14.3 (24.3%)	v2.11.2	0
Moment.js [87]	12,827 (1.6%)	**70.4% (14.2%↑)**	29.6% (26.8%↑)	71.6% (27.3%↑)	63	89	v2.18.1 (8.2%)	v2.29.1	**2**
RequireJS [97]	12,541 (1.6%)	**64.8% (1.9%↑)**	35.2% (82.6%↓)	28.1% (21.0%↑)	36	37	v2.3.6 (32.3%)	v2.3.6	0
SWFObject [17][7]	10,096 (1.3%)	**74.2% (55.5%↓)**	25.8% (48.7%↓)	63.3% (69.7%↓)	3	3	v2.2 (46.9%)	v2.2	0
Prototype [95]	7,782 (1.0%)	**81.2% (54.8%↓)**	18.8% (37.5%↓)	57.9% (47.8%↓)	11	11	v1.7.1 (43.4%)	v1.7.3	**2**
jQuery-Cookie [77][7]	7,582 (1.0%)	**63.3% (14.3%↑)**	36.7% (9.0%↑)	86.5% (10.6%↑)	7	7	v1.4.1 (64.4%)	v1.4.1	0
Polyfill.io [93]	6,755 (0.9%)	14.5% (69.6%↑)	**85.5%** (40.7%↑)	37.8% (56.8%↑)	3	3	v3 (65.5%)	v3	0

1: The number in parentheses indicates how much the usage has increased from our first observation date; ↑ increase, ↓ decrease.
2: Out of Externally-Hosted JavaScript Libraries. 3: Number of versions found in our dataset. 4: Total number of JavaScript library versions.
5: Latest version in our collected dataset. 6: Number of vulnerabilities reported during our observation period. 7: No longer maintained.

Table 1: Top 15 JavaScript Usage, Inclusion Type, Version, and Vulnerabilities.



(a) JavaScript Library Usage (Top 5) for 4 Years



(b) JavaScript Library Usage (Top 6 to 15) for 4 Years

Figure 3: Percent of JavaScript Library Usage.

Adobe Flash. As shown in Figure 2(b), on average, 0.7% (5,678 out of 782,300) websites use Adobe Flash. We ask a follow-up question regarding Adobe Flash; specifically, **RQ4-1):** What factors contribute to the usage of Flash even after the suspension of their official support?

6 VULNERABLE JAVASCRIPT LIBRARIES

We measure the JavaScript libraries to understand vulnerable websites due to vulnerable JavaScript libraries.

6.1 Landscape of JavaScript Library Usage

We first understand the current statistics of the JavaScript libraries' usage before we further investigate the security issues. We find an average number of 20.9 JavaScript libraries are included in a single website, and it is one of the dominant client-side resources in the wild. We first study the current landscape of JavaScript libraries such as usage trends, inclusion types, active/discontinued projects, and insecure JavaScript library versions with a focus on the Top 15 JavaScript libraries. These Top 15 libraries account for 96.5%.

jQuery & Plugins. `jQuery` is one of the most popular JavaScript libraries. It accounts for 64.0% (500,364 out of 782,300) of websites as shown in Table 1. While the usage of `jQuery` has been steadily decreasing over time from 67.2% (528,841) in Mar. 2018 to 63.1% (502,185) in Feb. 2022, as shown in Figure 3, it is still the most dominant library.

Observe that the third most popular library is a `jQuery-Migrate` [78] (used by 20.8% of websites on average), which is a `jQuery` plugin. The plugin aims to seamlessly migrate deprecated APIs of `jQuery` older than 1.9 so that the developers can use newer and secure `jQuery` versions without having compatibility issues. The plugin's popularity suggests that the JavaScript library update issues are critical in practice.

jQuery-Migrate Usage Drop in Aug 2020. Figure 3(a) shows (marked in the red box) a noticeable trend: the usage of `jQuery-Migrate` sharply dropping by approximately 10% for four months (Aug. 2020 – Dec. 2020), and then getting back (*i.e.*, went up by the 10%) in Dec. 2020.

Our investigation on this dropping trend reveals that WordPress plays an important role. Specifically, WordPress versions earlier than v5.5 have been using `jQuery-Migrate` to handle compatibility issues. Then, WordPress v5.5 disables `jQuery-Migrate` by default, hoping web developers update the outdated `jQuery` code themselves [113]. This results in the dropping trend. Unfortunately, disabling the library resulted in numerous compatibility issues on websites using plugins and themes dependent on the old version of `jQuery`. As a temporary solution, WordPress introduced a new plugin '`jQuery Migrate Helper`' which essentially implements the same functionality of `jQuery-Migrate` [114]. In the next version (WordPress v5.6), WordPress officially *re-includes* `jQuery-Migrate` as a default library on Dec. 8th, 2020, leading to `jQuery-Migrate`'s

usage increasing from Dec. 15th, 2020. As of May 2023, the latest WordPress 6 still includes `jQuery-Migrate`. One vulnerability of the library related to 'CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)' is reported from snyk.io [9], the GitHub issue [11] and a blog post [101], but no CVE ID is assigned to the vulnerability.

Popular JavaScript Libraries and Trends. Among the top 15 JavaScript libraries (in Figure 3), all libraries' popularities have been decreasing or steady except for four libraries, `JS-Cookie`, `Polyfill`, `Underscore`, and `Popper`. It may suggest that the security of a few popular JavaScript libraries can have a more significant impact than other libraries. `Polyfill` [93] includes polyfill code (*e.g.*, implementing the text shadow effect) for web browsers that do not support it. `JS-Cookie` [44] is a helper library for cookies. `Underscore` [107] provides various functional programming helpers (*e.g.*, map and reduce). `Popper` is a UI helper for tooltips and popovers [94].

Inclusion Type: Internal vs. External. Recall that there are two inclusion types: *internal* and *external*. Table 1 lists the percentage of the inclusion types for the JavaScript library. During our four-year observation period, the internal inclusion type (on average, 67.7%, 681,401) is more frequently used than the external type (on average, 32.3%, 325,520). However, external inclusion is also substantially used in popular JavaScript libraries: `jQuery-UI` (50.3%), `Popper` (53.1%), and `Polyfill` (85.5%). Note that, compared to internal inclusion, external inclusion has an additional attack surface because the systems delivering the libraries can be compromised, which will be discussed more in Section 6.5.

Table 1 shows 84.8% of the external inclusions are delivered by CDNs. In particular, over 90% of the external inclusions of `jQuery`, `jQuery-UI`, and `Popper` are delivered by CDNs. The CDN usage has also been maintained steadily during our observation period. Table 5 presents the top 3 CDNs for libraries: ajax.googleapi.com, code.jquery.com, and cdnjs.cloudflare.com. Note that in the `jQuery-Migrate` case, `wp.com` is the most used one because the plugin is provided by WordPress [113], which is discussed more in Section 7.

6.2 Known Vulnerability using CVE Report

We utilize CVE (Common Vulnerabilities and Exposures) reports to identify vulnerable JavaScript libraries. CVE is a de-facto standard vulnerability reporting platform. Among the top 15 JavaScript libraries in Table 1, we find 27 CVE reports on seven libraries. Most vulnerabilities (20 out of 27) are related to XSS attacks, while others include one prototype pollution attack, one arbitrary code injection attack, two resource exhaustion attacks, one regular expression denial of service (ReDOS), and one missing authorization attack.

Vulnerable Websites. We find an average of 41.2% of websites have libraries including at least one reported vulnerability (*i.e.*, CVE) in our dataset during our four-year observation period. This shows that even a few vulnerabilities impact many real-world websites, suggesting the significant security impact of vulnerable JavaScript libraries. As shown in Figure 12, for four years, an average of 0.79 vulnerabilities are carried per website (median: 0.75, max: 15.6). Updating the vulnerable versions is further discussed in Section 7.

jQuery & Plugins. `jQuery` has eight reported CVEs. In addition, its two plugins, `jQuery-Migrate` and `jQuery-UI`, have one and six vulnerabilities, respectively. On average, the vulnerable `jQuery` versions account for 37.7% (281,144 out of 782,300) of websites. CVE-2020-11023 [32] is the most impactful vulnerability that affects 56.2% of websites, including `jQuery` versions between v1.0.3 and v3.5.0, followed by CVE-2020-11022 [31] (56.1%) and CVE-2019-11358 [28] (54.6%). Even worse, a vulnerability (CVE-2012-6708 [23]) still accounts for 12.5% even though the vulnerability was reported more than a decade ago (in Jun. 2012).

Still Unpatched Library. All libraries we have analyzed in the paper, have been fixed, and the corresponding patches have been officially released, except for `Prototype` (CVE-2020-27511 [35], Regular Expression Denial of Service, or ReDOS). The vulnerability is critical as it affects all versions of the `Prototype` library. Unfortunately, it seems the vulnerability does not get proper attention from the developers. While we even find a discussion thread and a pull request for a bug fix [72] initiated in 2021, but it is not yet accepted and merged into the main repository.[8] Such an unpatched vulnerability published in a CVE report can be exploited by adversaries. Hence, the CVEs and their corresponding patches indicate the security of JavaScript libraries in practice.

> **Takeaway:** We observe a large number of websites (41.2%) in the wild contain vulnerable JavaScript libraries, that can be exploited by adversaries. The vulnerabilities are from high-profile libraries such as `jQuery`.

6.3 Dominant Vulnerable Versions & Discontinued Library

Dominant Insecure Versions. Table 1 lists the most dominant version of each JavaScript library during our four-year observation period. Specifically, `jQuery` version 1.12.4, released in May 2016, is still dominant in the wild (accounting for 28.5% of websites). Unfortunately, this version has four reported vulnerabilities: three XSS (CVE-2020-11023 [32], CVE-2020-11022 [31], and CVE-2015-9251 [24]) and one Prototype Pollution (CVE-2019-5428 [28]). While the newer versions, `jQuery` 3.0 (patched CVE-2015-9251 [24]) and 3.5 (patched all four vulnerabilities) are released in Jun 2016 and Apr. 2020, the old version (v1.12.4) is still dominant in the wild.

We further investigate various web developer communities and identify that the compatibility issues caused by the upgrade are a major concern [7, 27, 34, 47]. The compatibility issues are one of the reasons that lead to the dominant usage of such outdated JavaScript libraries. For example, to resolve compatibility issues, `jQuery-Migrate` is developed and publicly released. This specific version (v1.4.1), released in May 2016, accounts for 50.0% of the total usage and helps resolve the compatibility issues resulting from `jQuery` v3.0 major changes since v3.0 is not fully compatible with `jQuery` before v1.12.3 or v2.2.3. Even though `jQuery` libraries in websites are updated to v3.0, most websites still need `jQuery-Migrate` v1.4.1 to use the legacy functions of `jQuery` before v1.12.3 or v2.2.3. This indicates that updating versions does not mean that legacy code is no longer used because such a library helps web developers use the legacy code of outdated libraries.

[8] We find that the project is not particularly active, and responses are slow. The last commit on the official repository of this library was in Apr. 2017.

Library	CVE Vulnerable Ver.		True Vulnerable Ver.		Patched Ver.	Disclosed[1]	Patched[2]	Attack Type	CVE ID
	Ver. from CVE	# of Website	Ver.	# of Website					
jQuery	< 1.9.0	60,956 (12.2%)	< 3.6.0	**291,579 (58.3%)** ↓	1.9.0	05/19/2020	01/15/2013	XSS	CVE-2020-7656
	1.0.3 ~ 3.5.0	281,144 (56.2%)	1.4.0 ~ 3.5.0	276,956 (55.4%) ↑	3.5.0	04/10/2020	04/10/2020	XSS	CVE-2020-11023
	1.2.0 ~ 3.5.0	280,968 (56.1%)	1.12.0 ~ 3.5.0	131,284 (26.2%) ↑	3.5.0	04/29/2020	04/10/2020	XSS	CVE-2020-11022
	< 3.4.0	273,092 (54.6%)	–	–	3.4.0	03/26/2019	04/10/2019	Prototype Pol.†	CVE-2019-11358
	1.12.0 ~ 3.0.0	88,757 (17.7%)	–	–	3.0.0	06/26/2015	06/09/2016	XSS	CVE-2015-9251
	1.4.2 ~ 1.6.2	10,414 (2.1%)	1.5.0 ~ 2.2.4	**214,078 (42.9%)** ↓	1.6.2	09/01/2014	06/30/2011	XSS	CVE-2014-6071
	< 1.9.1	62,431 (12.5%)	< 1.9.0	60,956 (12.2%) ↑	1.9.1	06/19/2012	02/04/2013	XSS	CVE-2012-6708
	< 1.6.3	16,857 (3.4%)	–	–	1.6.3	06/05/2011	09/01/2011	XSS	CVE-2011-4969
Bootstrap	< 3.4.1, < 4.3.1	46,545 (27.7%)	–	–	3.4.1, 4.3.1	02/11/2019	02/13/2019	XSS	CVE-2019-8331
	< 3.4.0	38,827 (23.1%)	3.2.0 ~ 3.4.0	36,019 (21.4%) ↑	3.4.0	08/13/2018	12/13/2018	XSS	CVE-2018-20676
	< 3.4.0	38,827 (23.1%)	3.2.0 ~ 3.4.0	36,019 (21.4%) ↑	3.4.0	01/09/2019	12/13/2018	XSS	CVE-2018-20677
	< 4.1.2	46,529 (27.7%)	2.3.0 ~ 4.1.2	46,287 (27.5%) ↑	4.1.2	05/29/2018	07/12/2018	XSS	CVE-2018-14042
	< 4.1.2	46,529 (27.7%)	–	–	4.1.2	05/29/2018	07/12/2018	XSS	CVE-2018-14041
	< 4.1.2	46,529 (27.7%)	2.3.0 ~ 4.1.2	46,287 (27.5%) ↑	4.1.2	05/29/2018	07/12/2018	XSS	CVE-2018-14040
	< 3.4.0	38,827 (23.1%)	2.1.0 ~ 3.4.0	38,748 (23.1%) ↑	3.4.0	06/27/2016	12/13/2018	XSS	CVE-2016-10735
jQuery-Migrate	< 1.2.1	956 (0.6%)	1.0.0 ~ 3.0.0	**95,165 (62.8%)** ↓	1.2.1	04/18/2013	09/16/2007		N/A*
jQuery-UI	< 1.10.0	13,948 (14.7%)	–	–	1.10.0	09/02/2010	01/17/2013	XSS	CVE-2010-5312
	< 1.10.0	13,948 (14.7%)	–	–	1.10.0	11/26/2012	01/17/2013	XSS	CVE-2012-6662
	< 1.12.0	42,856 (45.1%)	1.10.0~1.13.0	**43,312 (45.6%)** ↓	1.12.0	07/21/2016	07/08/2016	XSS	CVE-2016-7103
	< 1.13.0	57,261 (60.2%)	–	–	1.13.0	10/27/2021	10/07/2021	XSS	CVE-2021-41182
	< 1.13.0	57,261 (60.2%)	–	–	1.13.0	10/27/2021	10/07/2021	XSS	CVE-2021-41183
	< 1.13.0	57,261 (60.2%)	–	–	1.13.0	10/27/2021	10/07/2021	XSS	CVE-2021-41184
Underscore	1.3.2 ~ 1.12.1	1,930 (9.84%)	–	–	1.12.1	03/02/2021	03/19/2021	Arb. Code Inj.‡	CVE-2021-23358
Moment.js	< 2.19.3	4,322 (33.7%)	–	–	2.19.3	09/05/2017	11/29/2017	Res. Exhaust.#	CVE-2017-18214
	< 2.11.2	2,115 (16.5%)	2.8.1~2.15.2	**2,174 (17.0%)** ↓	2.11.2	01/26/2016	2/7/2016	Res. Exhaust.#	CVE-2016-4055
Prototype	≤ 1.7.3	7,782 (100%)	All versions	7,782 (100%)	N/A**	06/21/2021	N/A**	ReDOS	CVE-2020-27511
	< 1.6.0.1	4 (0.1%)	–	–	N/A***	02/03/2020	N/A**	Missing Auth.◇	CVE-2020-7993

*: No CVE ID is assigned (vulnerability was found from snyk.io and GitHub issue.) **: No patched versions are available. ***: Affected version is no longer available.
1: Disclosed Date. 2: Patched Date. †: Prototype Pollution. ‡: Arbitrary Code Injection. #: Resource Exhaustion. ◇: Missing Authorization.
↓: More versions are vulnerable than CVE's version (*understated* version). ↑: Fewer versions are vulnerable than CVE's version (*overstated* version).

Table 2: Vulnerabilities of Top 15 JavaScript Library. Among the top 15 libraries in Table 1, seven libraries are publicly reported to contain 28 vulnerabilities. The affected versions of the 12 vulnerabilities are incorrect.

Discontinued Library Projects. As shown in Table 1, we observe two discontinued JavaScript library projects: `jQuery-Cookie` [77] and `SWFObject` [17]. `jQuery-Cookie` is a `jQuery` plugin to help easily manage cookies. Although this library project is no longer maintained since 2015 (officially migrated and renamed to a new project, `JS-Cookie` [44][9]), we observe many websites (7,582 websites) still use this discontinued library. We also measure how many websites migrated to `JS-Cookie` as recommended. We find, in total, only 7,800 websites (39% of the total 19,992 websites using `jQuery-Cookie`) are migrated to `JS-Cookie`. Note that 61% of websites still use the legacy library even after 7 years, meaning that updating JavaScript libraries is extremely slow in practice. Moreover, discontinued library projects typically do not address or patch the bugs if new bugs are identified. Developers are either recommended to use a newly migrated project [77] or use it with their own risk [17].

Another discontinued project is `SWFObject`, a JavaScript library to play Adobe Flash content on a webpage. As briefly discussed in Section 2.2, Adobe Flash is officially no longer supported by Adobe, and major web browsers remove the Flash components. While this library project has been no longer maintained since 2013, it is still in use: an average usage of 1.3% (10,096 out of 782,300, ranked 12th) websites in our collected dataset. We find that WordPress plugins for `SWFObject` play a significant role in this Flash ecosystem; an average of 22.3% websites use WordPress plugins out of the total number of websites using the `SWFObject` library. We also observe that 12.7% of the `SWFObject` is delivered by the Google CDN (`ajax.googleapis.com`). We informed Google that the project is discontinued and potentially insecure and may require further actions (*e.g.*, warning users or finding an alternative). We discuss the possible suggestions regarding the discontinued projects in Section 9.

Takeaway: The dominant versions across all libraries are outdated and contain multiple vulnerabilities. We observe that backward compatibility is a major reason for preventing the update. Worse, 2.1% websites use *discontinued* libraries, exposing them to potential vulnerabilities.

6.4 Accuracy of CVE Vulnerability Info.

CVE is critical information for understanding vulnerable programs and websites, potentially impacting various security practices such as updating. We aim to understand how reliable the *version information* in CVE reports is. As web developers may rely on CVE reports to measure how vulnerable the JavaScript libraries they

[9] As of May 2023, `JS-Cookie` is an active project; the recent commit was on Apr 24, 2023 [39].

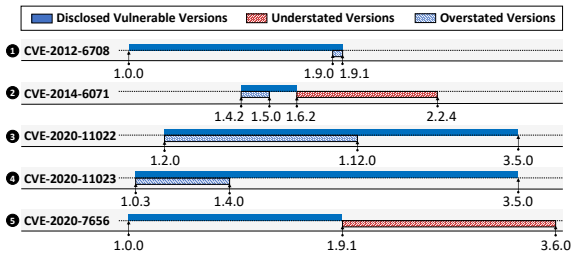


Figure 4: Comparison of `jQuery` Disclosed Vulnerable Versions (Upper line) and Understated/Overstated Versions (Lower line). The Understated Versions (red stripes) represent the vulnerable versions we newly revealed. The Overstated Versions (blue stripes) indicate the versions we revealed are not vulnerable.

use are. Incorrectly stated versions of CVE may impact subsequent decisions.

Setting Version Validation Experiment. We manually investigate each CVE to validate the described vulnerability and affected versions using proof of concept (PoC) code in our controlled experiment environment. We set up the controlled experiment environment with each vulnerable library version and its required dependency. Particularly, for the `jQuery` vulnerabilities, in total, we set up 85 different environments for each different version from v1.0.0 to the latest version (v3.7.0, as of May 2023).

We find and utilize the existing seven PoC codes out of 27 CVEs (CVE-2020-7656 [29], CVE-2014-6071 [13], CVE-2018-20677 [25], CVE-2018-14040 [26], CVE-2016-10735 [22], CVE-2016-7103 [15], and `jQuery-Migrate` vulnerability [10]) For those PoCs that we initially failed to reproduce, we manually analyze the vulnerability and reimplement new PoC code. Particularly, for CVE-2020-7656 of `jQuery`, the existing PoC [29] described in the CVE report does not properly reproduce the vulnerability. Therefore, we reimplement the PoC by removing the `jQuery` selector from the existing PoC code (Listing 1 and Listing 2 in Appendix).

Incorrect CVE information. The version information can be incorrect in two ways: (1) CVE may *understate* the version, meaning more versions can be affected but not known to the public. (2) CVE may *overstate* the version, meaning that fewer versions are affected by the vulnerability than what is known to be public.

- *Understated versions* may cause delays in vulnerable websites' updates, as web developers do not realize that they use vulnerable libraries. For example, while CVE-2020-7656 mentions that it only affects 40 versions (lower than 1.9.1), from our experiments we find out that 79 versions (higher versions than 1.9.1, such as 1.10.1) are also affected by the vulnerability. This would make developers using version 1.10.1 believe that their websites are not vulnerable (hence less motivated to update).
- *Overstated versions* may cause ill-advised updates, leading to higher maintenance and development costs.

True Vulnerable Versions (TVV) Identified. True Vulnerable Versions (TVV) means actually-affected versions identified from our validation experiments. From our experiments, described in Table 2, we find that 13 out of the 27 CVEs contain incorrect versions. Specifically, five of them (⬇) have *understated versions*, meaning that the vulnerabilities affect more versions than reported. The remaining eight of them (⬆) have *overstated* versions, where the vulnerabilities affect fewer versions than reported. Particularly, Figure 4 illustrates the impact of incorrect versions in `jQuery`. For each CVE, there are two lines where the upper line represents vulnerable versions disclosed by the CVE and the lower line shows understated and overstated versions revealed by our Version Validation Experiment. In this `jQuery` case, 5 out of 8 CVEs have incorrect versions. Particularly, CVE-2020-7656 specifies that the only affected version is lower than 1.9.1, while we reveal that it affects *all versions until 3.6.0*. Web developers who use a certain version (> 1.9.1) can be misled that their version is not affected by the vulnerability and do not need to update the library. We present 5 more cases (`jQuery-Migrate`, `jQuery-UI`, `Bootstrap`, `Moment`, and `Prototype`) in Appendix (Figure 13).

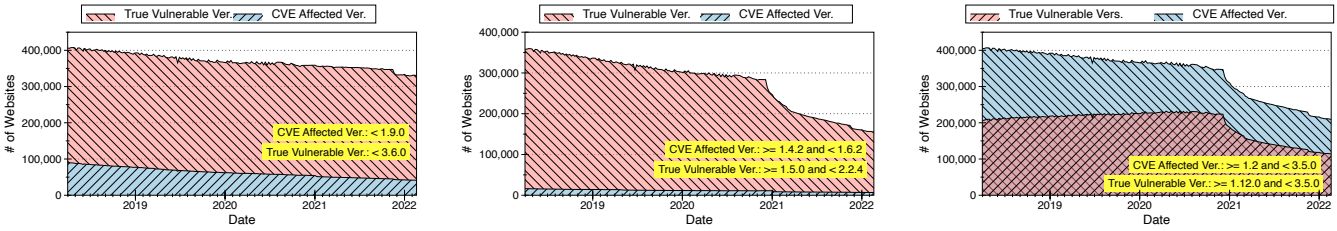
> **Takeaway**: We discover that 13 CVE reports (out of 27) incorrectly state vulnerable versions of the libraries, providing misleading information. Specifically, 5 CVEs understate the impact of the vulnerabilities, potentially downplaying the vulnerabilities. 8 are understated versions.

# of Websites Affected by Incorrect Versions. We measure the number of websites affected by these incorrect versions in CVEs. Figure 5 shows the three cases of `jQuery` vulnerabilities; the red background refers to 'vulnerable websites not mentioned in CVEs (*i.e.*, websites using vulnerable libraries but were not spotted),' and the blue background does 'websites using vulnerable versions mentioned in CVEs.' Specifically, Figure 5(a) and Figure 5(b) show that a large number of websites (296,818 in CVE-2020-7656 and 265,362 in CVE-2014-6071) may not be known whether they are susceptible to the vulnerabilities. Meanwhile, Figure 5(c) shows that CVE versions unnecessarily include versions that are not vulnerable (*i.e.*, *overstated versions*), potentially misleading and causing unnecessary updates (and compatibility issues).

> **Takeaway:** (337,773 websites are affected by incorrect version information in CVEs. 316,809 websites are undisclosed in the wild due to incorrect CVEs.

Refining Vulnerable Websites. Our previous conclusion that 41.2% are using at least one vulnerable JavaScript library in Section 6.2 was based on *the assumption that the CVEs' versions are correct, which we revealed that it is not true.* Hence, we now measure the number of vulnerable websites *again* with the *true vulnerable versions* (TVV) we discover.

We find an average of ***43.2%*** (+2%) websites (337,773 out of 782,300) are vulnerable as carrying at least one vulnerability; the incorrect CVE information results in increasing the number of vulnerable websites by 2%. Interestingly, in 2018, the average difference between them is only 0.1%, but in 2022, the gap increases by 2.9%. We find that it also affects high-profile (*i.e.*, popular) websites: `microsoft.com` (ranked 46th) and `onlinesbi.com` (ranked 111th) use `jQuery` v3.5.1, which we reveal that it is vulnerable while CVE does not mention (*understate*). Also, `docusign.com` (ranked 1,693rd) uses `jQuery` v2.2.3, which is another *understated* case that we reveal it is vulnerable.



(a) CVE-2020-7656 (b) CVE-2014-6071 (c) CVE-2020-11022

Figure 5: The Total Number of Websites with True Vulnerable Versions of `jQuery` Vulnerability. (a) and (b) show that more vulnerable versions are revealed, while (c) shows that it reveals a fewer number of versions are vulnerable.

Furthermore, we also refine the CDF of the average number of vulnerabilities per website as shown in Figure 12. Compared to the results from the CVE's version, with the true vulnerable versions (TVV) we reveal, the average number of vulnerabilities per website is almost 1 (mean: 0.97 and median: 0.96), which is an alarming result.

Takeaway: With the corrected version information in CVEs, we observe that (337,773 websites (43.2%) are vulnerable (which is approximately 2% more than the analysis based on existing CVEs).

6.5 Potential Security Threats of Untrustful External Libraries

Potential Security Threat. Section 2.1 mentions that a website can include JavaScript libraries either internally (*i.e.*, locally hosted[10]) or externally (*e.g.*, remotely hosted[11]). Potential security threats could arise when a web page loads compromised external JavaScript libraries. Compromised JavaScript libraries can obtain full privileges on the websites unless isolated in a frame; for example, the loaded JavaScript can modify the DOM, load other external content, redirect the visitors to phishing websites, or deliver malware to the visitors. Particularly, if the external JavaScript libraries are loaded from repositories of collaborative version control, such as GitHub, GitLab, and Bitbucket, the libraries cannot be trusted because the repositories (and the libraries) can be compromised or malicious. In other words, we cannot fully establish trust in the maintainers and contributors of the open-source library projects, compared to the libraries hosted on the official CDNs. For example, a malicious contributor of a JavaScript library project could insert malicious payloads into the library, or an adversary could make a malicious pull request to the library project if these are public repositories [116].

Libraries from Untrustful Sources. We measure how many websites could be vulnerable due to the JavaScript libraries hosted on collaborative version control platforms, such as `GitHub`, `GitLab`, and `Bitbucket`. We find that an average of 1,670 websites load more than one external JavaScript library externally hosted on the 57 GitHub repositories. Particularly, the most popular GitHub repository is `wp-r.github.io`[12] that accounts for 11.3% of the websites. It is an individual repository that hosts '`adsplacer`' and '`jquery.iframetracker`'. The first library is a WordPress plugin to help place advertisements, and the latter one is a `jQuery` plugin that can track users' clicks on iframes. The tracker is added on Adguard's Spyware Filter [54] to block trackers. Table 6 breaks down the GitHub repositories that are used by the top 10K websites.

Mitigation against Untrustful Sources (SRI). Subresource Integrity (SRI) [89] can be a viable security defense mechanism against the threat of untrustful JavaScript libraries. It ensures the unmodified JavaScript files that contain expected data are delivered, loaded, and executed. Specifically, web developers can specify an expected base64-encoded cryptographic hash (such as SHA256, SHA386, and SHA512) of a JavaScript file in the `integrity` attribute of the <`script`> tag. At runtime, if a specified hash value does not match the hash value of the downloaded JavaScript file, it is neither loaded nor executed.

We measure how the SRI is used on websites in the wild. Surprisingly, as shown in Figure 10, 99.7% of websites have at least one externally-hosted JavaScript library without the `integrity` attribute, which cannot be trusted when external hosts are compromised. Furthermore, we closely take a look at the libraries hosted on the repositories of collaborative version control services such as GitHub. Of 1,670 websites using the external libraries on GitHub, only an average of 10.1 websites (0.6%) use the `integrity` parameter as the mitigation.

We further check how many official websites of JavaScript libraries explicitly mention the SRI (or the `integrity` attribute) in its code snippet; for example, they may provide a code snippet that includes an `integrity` attribute and hash value, which helps web developers simply copy and paste the secure code snippet. We find that out of the top 15 JavaScript libraries, *only one* library (`Bootstrap`) provides a code snippet that includes an `integrity` attribute and its hash value. As software developers (including web developers) are typically known to have the copy and paste behaviors [53], we argue that we might be losing chances to secure external JavaScript libraries by not having the `integrity` attributes in the example code snippets on the official websites of JavaScript libraries.

Mitigation against Untrustful Sources (Crossorigin). We also measure how properly developers use `cross origin` in the wild.

[10] e.g., <script src="./bar.js"></script>
[11] e.g., <script src="https://foo.com/bar.js"></script>
[12] https://github.com/wp-r/wp-r.github.io

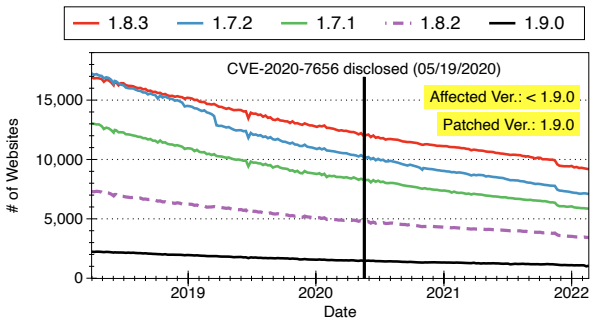


Figure 6: Usage Trends of Affected Versions of `jQuery` CVE-2020-7656 Vulnerability (versions are ordered by popularity).

`crossorigin` is an attribute to provide support for Cross-Origin Resource Sharing (CORS) and to control how to handle resources coming from cross-origin domains [43]. The best practice for cross-origin requests is to use the `anonymous` value because it prevents sending user credentials (e.g., via cookies) to cross-origin requests (i.e., requests from a JS library fetched from a different origin). If '`use-credentials`' is specified in the `crossorigin` attribute, user credentials could be sent even for cross-origin requests [46, 112], leading to "cross-origin data leakage" [110].

We find 97.1% of websites use '`anonymous`' and only 1.9% use '`use-credentials`,' out of the total websites using `crossorigin` with the `integrity` attribute. This indicates that while most web developers properly follow the security best practice for this attribute, a few developers misconfigure the value for `crossorigin`. The small number of such websites may potentially leak credentials for cross-origin requests.

Takeaway: The mitigation (*e.g.*, `integrity`) to potential security threats of the externally-hosted libraries are barely used in the wild, which indicates such websites could remain vulnerable to potential security threats.

7 UPDATE OF VULNERABLE JAVASCRIPT LIBRARIES

When a newly disclosed vulnerability affects JavaScript libraries used in a website and there is a patched version for the vulnerability, it is desirable to update the vulnerable libraries promptly. This section attempts to understand how well vulnerable JavaScript libraries are updated in practice over time. Particularly, when a vulnerability is publicly disclosed and assigned a CVE ID, and a patch for the vulnerability is released, we measure how long (*e.g.*, how many days) it takes to update their affected vulnerable versions after the patched version's release. Note that in this section, we rely on the versions stated in CVEs even if they might be inaccurate (*i.e.*, not the True Vulnerable Versions or TVVs). As a result, we aim to measure how the developers and administrators respond to the CVEs for security updates.

Updating jQuery. We focus on `jQuery` as it is dominant in the wild – other libraries' updating patterns are discussed in the Appendix (Figure 15). Figure 6 shows a few versions' usage trends with the CVE disclosed dates; CVE-2020-7656 specifies that all versions lower than (< v1.9.0) are affected, and its patched version is v1.9.0 as described in Table 2. We observe that the usage of the patched version (v1.9.0) has not increased, but rather slightly *decreased*. From this observation, we can learn that vulnerable JavaScript libraries are rarely updated, or worse, some websites newly *start to use the vulnerable version even after the CVEs are publicly disclosed.*

Furthermore, Figure 7(a) shows the updating trends of the most dominant version (v1.12.4), which is affected by two vulnerabilities (CVE-2020-11022 and CVE-2020-11023). From the 1.12.4 version, it is recommended to update to v3.5.0 or higher (≥ v3.5.0). Hence, ideally, the usage of v1.12.4 should decrease, and the usage of the versions higher than v3.5.0 (including v3.5.0, v3.5.1, v3.6.0, and v3.6.1) should increase if web developers properly follow the best practice of updating vulnerable versions.

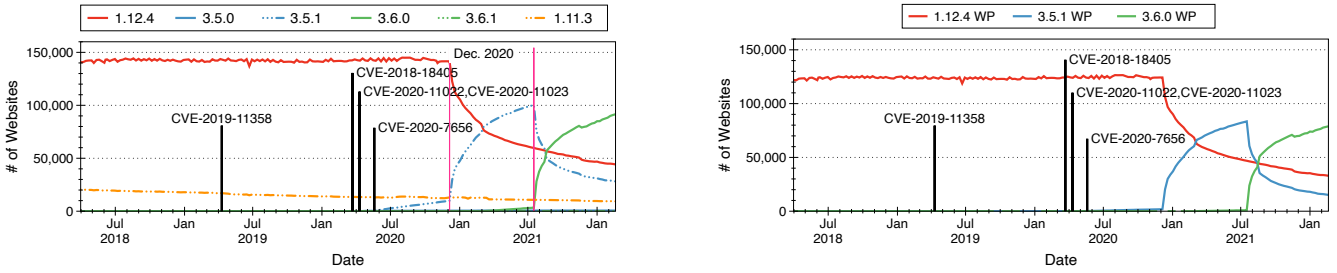
Unfortunately, we observe different usage patterns as shown in Figure 7(a). v3.5.0 is barely (nearly 0%) used in the wild, which means that no update operation is performed to v3.5.0; similarly, v3.6.1 has the same pattern. Observe that the v3.5.1 usage starts increasing one month after the CVEs are publicly disclosed (around July 2020). However, we do not believe that this is a potential update from v1.12.4 because v1.12.4 does not decrease when the usages of v3.5.0, v3.5.1, v3.6.0, and v3.6.1 start increasing. Moreover, we observe a different update behavior from Dec. 2020. Specifically, in Dec. 2020, the v3.5.1 usage sharply increases while v1.12.4 accordingly decreases. We randomly choose 100 websites in our dataset and manually analyze how the versions are updated. The result shows that they are truly updated (i.e., replaced the old version with the new version). In addition, from Aug. 2021, we observe a sharp increase in version updates from v3.5.1 to v3.6.0.

Takeaway: We observe that it averagely takes 531.2 days (17.4 months) to update the vulnerable version. The updating trend is also not linear, implying that there should be a particular contributor to this radical update trend.

Main Contributor to Updating. We further investigate to understand the major actors for the updates. Specifically, from our dataset, we notice a sharp increase in updating from Dec. 2020. We aim to answer the question: *Who or what events drove this radical update behavior?*

Interestingly, we find that WordPress has been the main contributor. Figure 7(b) shows the `jQuery` usage of WordPress over time. We can see the same patterns between the `jQuery` version usages (Figure 7(a)) and the WordPress' `jQuery` version usages (Figure 7(b)) where they have almost identical usage pattern changes for v3.5.1, v3.6.0, and v1.12.4. We further analyze how WordPress becomes the main contributor, and find the WordPress *Auto-Updating* feature [115] helps automatically update the old and vulnerable `jQuery` libraries at almost the same time, which leads the Web ecosystem to be more secure. From this observation, we can learn that automated mechanisms (*e.g.*, auto-updating) are effective in securing websites by updating JavaScript libraries.

Takeaway: WordPress's Auto-Updating feature contributes to the major update of `jQuery`. The Web security community may



(a) `jQuery` v1.12.4 Usage & Patched Version (> v3.5.0) Usage

(b) WordPress Versions Associated with jQuery Update

Figure 7: Trends of Vulnerable `jQuery` v1.12.4 Usage and Patched Version (> v3.5.0) Usage & WordPress Versions Associated with the `jQuery` Update.

suggest a new auto-updating feature for the client-side resources to secure the Web ecosystem.

Update Delays with True Vulnerable Versions. We measure the updating delays with the True Vulnerable Versions we revealed in Section 6.4.[13] We find that the *understated* CVE reports lead to more significant delays; on average, it takes 701.2 days (23 months, 1.9 years), compared to 510 days calculated with the versions from CVEs.

Takeaway: Due to the understated versions in CVEs, we find that the true delay in the update of vulnerable JS libraries is substantially more severe, +191.2 days, than our initial estimate.

8 INSECURE ADOBE FLASH

Flash Usage & Case Study. As mentioned in Section 2.2, the support for Adobe Flash is officially terminated (Jan. 2021), meaning that Adobe will not fix security vulnerabilities anymore. As a result, all major web browsers claim that Adobe Flash components are *removed* from the browsers as of Jan 2021, and Flash can no longer be played on the browsers [21, 33, 36, 37]. Also, web developers are strongly recommended to replace Adobe Flash with HTML5 [58, 59]; Adobe even released a conversion tool for developers from Flash to HTML5 [57].

We empirically observe some websites using Adobe Flash applications and a new platform application promoting the use of Adobe Flash applications, even though Flash was officially no longer supported. This observation motivates us to raise the research question; *If so, how many websites still use Flash*? and *what websites and functionalities still depend on the Flash in the wild*? To answer these research questions, we measure the usage of Adobe Flash in the wild. Note that compared to Section 6.3 where we aimed to measure the discontinued project, `SWFObject` (*e.g.*, how many websites use the discontinued project), in this section, we better understand how many Flash contents (*i.e.*, `.swf` files) are embedded and played in websites.[14]

As shown in Figure 8, the Flash usage has steadily decreased during our observation period (from 9,880 websites in Feb. 2018 to 4,218 and 3,195 websites in Dec. 2020 and Feb. 2022, respectively). However, an average of 3,553 websites still use Adobe Flash after its end of life.

Case Study of Top 10K Websites. We take a closer look at the top 10K websites that still use Adobe Flash after the end of life (Jan. 1st, 2021). Among the top 10K websites, we find thirteen websites still use Adobe Flash as of May 2023. Six out of thirteen domains use Flash for visible, dynamic content on the webpages (*e.g.*, a banner, a dynamic image application, and a media player). The remaining seven cases include `.swf` files or links in the HTML code of the webpages, but the Flash files are neither visually displayed nor played on the web browser. In invisible cases, as `.swf` objects are positioned outside of the page or behind certain images, end-users cannot recognize Adobe Flash is being played on their web browsers.

We further analyze who manages and operates the websites and which country the websites are from. It turns out that four out of thirteen domains are managed and operated by Chinese companies; other countries are Hungary, Iran, Japan, Portugal, Spain, Russia, and Taiwan. This result motivates us to raise a follow-up research question, "*why do Chinese websites still use Adobe Flash more than other countries do even after the official support is no longer available, and all major browsers are no longer supporting Flash*?"

Flash-Support Browsers & Ecosystem. To answer the follow-up question, we manually examine the top 10 desktop Web browsers in the world, focusing on how they support Flash. We test them both on macOS 12.4 and Windows 10. Our results are summarized in Table 3. We find that all Web browsers no longer support Flash *except for* `360 Browser` [52]. This browser is based on Google Chrome and developed by a Chinese internet security company, Qihoo 360. The browser has two versions: `Secure` and `Extreme`. Particularly, `360 Extreme` v12.2.1662.0 for macOS[15] (based on Chrome v78.0.3904.108 was released in Nov. 2019) still supports Flash as of May. 26, 2023. Moreover, when end-users access the Flash-embedded webpages, they are recommended to visit `www.flash.cn` to play the Flash contents. This website provides its own, customized version of the old Adobe Flash player components for end-users who still want to use Adobe Flash contents [119]. We can learn from these two

[13] 5 CVEs *understate* affected versions as shown in Table 2, which means that there are more undisclosed vulnerable versions in the wild.

[14] Note that .swf file can be embedded using the `embed` HTML tag, without the `SWFObject` JavaScript library.

[15] This version is available at https://browser.360.cn/ee/mac/index.html

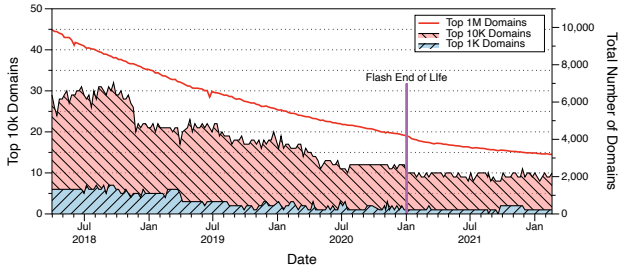


Figure 8: Adobe Flash Usages. The left y-axis indicates the number of domains for the top 1K and 10K domains, and the right y-axis is the number of domains for the top 1M domains.

cases (`360 Extreme` browser and `flash.cn`), this unique Flash ecosystem could play an important role in maintaining a number of Chinese websites using insecure Flash. This eventually leads users to remain exposed to the security threats of Flash.

Takeaway: An average of 3,553 websites still use Adobe Flash even after the end of the life of Flash. We find that a certain web browser and platform (`www.flash.cn`) still facilitate the use of Adobe Flash for end-users.

Insecure AllowScriptAccess Parameter. In the HTML code, the `AllowScriptAccess` parameter is to control if a `.swf` file (specifically, ActionScript) is allowed to call/access JavaScript and HTML DOM in the HTML page where the `.swf` is loaded. The parameter has three options; *always*, *sameDomain*, and *never* [55]. The *sameDomain* option allows a `.swf` file to call/access JavaScript and HTML DOM only when the `.swf` and HTML page are from the same domain. The *never* option never permits the JavaScript calls and the HTML DOM accesses. If no value is specified to this parameter, the *sameDomain* option is by default applied. However, the *always* option permits a `.swf` file to always call/access JavaScript and HTML DOM even though the domain of the `.swf` file is different from the one of the HTML page. A potential security threat could arise when external and untrustful `.swf` files are embedded and loaded. Then, they can maliciously access, call, and manipulate JavaScript and HTML DOM. For example, in a web forum where anyone can include an external `.swf`, an adversary can upload or link a malicious `.swf` to the target forum. Therefore, web developers are strongly recommended not to use the *always* option for the parameter by Web Hypertext Application Technology Working Group (WHATWG) [111].

We measure if web developers properly follow the best practice (not to use `AllowScriptAccess` parameter) in the wild. We find that an average of 24.7% websites (out of the total number of websites using Flash) use insecure parameters. Specifically, the insecure usage has increased by approximately 9% (from 21% to 30%). This indicates that 24.7% of websites are vulnerable to malicious `.swf` files loaded from cross-origin sources.

9 DISCUSSION

Suggestions. Based on our observation, we discuss potential suggestions that can help improve the Web ecosystem.

- *Discontinued JavaScript library Projects.* As observed in Section 6.3, Google CDNs and WordPress plugins are believed to be one of the main factors to facilitate the use of the outdated library for developers. We suggest that the official WordPress plugins website and Google CDN website, where web developers download plugins or utilize CDN URLs, may need to provide them with a warning icon or indicator saying that they are potentially vulnerable. We believe these interventions would help reduce the usage of discontinued libraries.
- *Inaccurate CVEs Information.* One of the main reasons why CVEs have incorrect version information would be that the web security community barely has the experiment environment that we have set up for this study. The community is in needs to have such experiment settings to accurately know the affected versions by validating each vulnerability using PoC code.
- *Our Large-Scale Dataset.* We publicly share our large-scale dataset (157.2M pages) collected for four years (Mar. 2018 – Feb. 2022) at "https://moa-lab.net/measurement-client-side-resources/". The security community can utilize our datasets to better understand the Web ecosystem from another perspective. We believe future measurement studies using our dataset can help secure the Web ecosystem.

Limitations. We discuss the limitations of our work.

- *Various Security Impacts.* The severity and security impact of vulnerabilities introduced in outdated or vulnerable JavaScript libraries can vary. Some vulnerabilities could be exploitable only under specific conditions. For example, CVE-2020-11022 [30] and CVE-2020-11023 [32], both vulnerabilities are only triggered under a certain condition where attackers can pass HTML to one of jQuery's DOM manipulation methods. Thus, using outdated or vulnerable JavaScript libraries in the websites does not necessarily mean that attackers would be able to exploit the vulnerabilities. However, we would like to highlight that the use of outdated or vulnerable JavaScript libraries potentially may lead to security vulnerabilities in websites under certain conditions unless the vulnerabilities are updated.
- *Validity Concern.* Our analysis may contain false positives due to the inherent errors of Wappalyzer (*e.g.*, we may not catch cases where administrators manually patched vulnerabilities instead of using the officially updated version's files). However, we empirically observe that it may happen rarely in practice. Specifically, we conduct an extra experiment with a newly collected dataset of Alexa 100K domains' client-side resources on June 7th, 2023. We compare all the downloaded JavaScript library files in the dataset with the official unmodified JavaScript library files (using file hashes) to understand whether manual modifications (or patches) are prevalent. We find that 1,521 JavaScript libraries do not match the original hash values of the official JavaScript libraries. We randomly select 100 samples and manually investigate the mismatched libraries. We find that all cases are caused by extra newlines or spaces and modified comments. We do not observe any cases that are manually patched in the dataset.
- *Representativeness of the entire Web ecosystem.* We utilize Alexa top 1M domains to understand the entire Web ecosystem as we assume that all websites in Alexa top 1M are representative of the entire Web ecosystem. This may provide a limited viewpoint

and could potentially skew the results. However, it's worth noting that previous studies [64, 67, 82, 105] attempting to better understand the Web ecosystem have predominantly examined and measured the ecosystem using Alexa top 1M domains.

Ethics. Our work identified several security issues. Accordingly, we responsibly disclosed the issue with Google CDN that supports the discontinued libraries.

Future Work. As mentioned in the limitation (Section 9), we mainly focus on the landing pages (*e.g.*, index.html). The future work is needed to further explore other pages within websites, such as interactive features, which could potentially introduce additional vulnerabilities. Furthermore, as part of future work, it would be valuable to examine cases in which websites have updated to patched versions but subsequently experienced regressions, potentially due to compatibility concerns, and to analyze the resultant implications for the overall security of the web ecosystem. Lastly, it would be also highly beneficial to evaluate the exploitability of the websites that have potentially vulnerable library versions; in other words, we need to measure how many websites (or JavaScript libraries) are actually vulnerable and exploitable. This is because the known vulnerabilities can be exploitable only under specific conditions as mentioned in Section 9.

10 RELATED WORK

Client-side Resource Measurement. There have been a number of measurement studies that attempted to better understand the Web ecosystem, especially the security practices of client-side resources [60, 67, 82, 84, 90, 92, 98, 105, 117]. Since JavaScript libraries have held a dominant position as client-side resources, prior measurement studies have focused on JavaScript libraries. Specifically, Demir et al. [67] performed a longitudinal study of the updating behaviors (*e.g.*, JavaScript library updates) and found that (even vulnerable) JavaScript libraries were barely updated. While this study offers an overview of the general trends in JavaScript library updates, our research delves deeper into the updating patterns of individual JavaScript libraries (and its versions) and Adobe Flash vulnerabilities. In particular, we identify the factor that contributes to the sharp increase in updates, which is the auto-updating feature provided by WordPress. Based on this finding, we offer a valuable recommendation that this auto-updating feature would be provided for JavaScript library users. Moreover, we further study the potential security threats of the discontinued JavaScript libraries and untrustful externally-hosted libraries. Finally, studying Adobe Flash (*e.g.*, a 3rd-party software dedicated to running Flash) and the accuracy of CVE information (*e.g.*, how inaccurate CVE information impacts the web ecosystem) are our unique contributions.

Furthermore, Nikiforakis et al. [90] and Lauinger et al. [82] worked on JavaScript library inclusions and identified that some JavaScript libraries used in the wild were vulnerable or could be compromised. Particularly, while Lauinger et al. [82] is more focused on vulnerable JavaScript library usage statistics with a single snapshot dataset of 2016 (seven years ago), our study uses a four-year longitudinal dataset (2018 – 2022) of Alexa's top 1M domains. Our security insight and implication of web developers' (or administrators') updating behaviors require the longitudinal dataset and analysis; such updating behaviors cannot be measured and observed with a single snapshot of domains. Moreover, we measure the inaccuracy of CVE reports by testing the Proof of Concept exploit of each report, if available. We find that 337,773 websites are affected by inaccurate CVE reports (*e.g.*, failed to realize and update vulnerable JavaScript libraries).

A few measurement studies, particularly on web trackers [84] and general client-side resources [105], were conducted using a longitudinal dataset collected from the Internet Archive's Wayback Machine (archive.org). Specifically, Lerner et al. [84] mainly focused on the trackers, not the security issues of JavaScript libraries. Compared to this study, our work presents a measurement study on the top 15 most used JavaScript libraries and how developers have updated the libraries over 4 years of period.

CVE Information. Ocariza et al. [92] empirically measured and classified the root causes of the vulnerabilities of JavaScript libraries using 317 bug reports from 12 bug repositories. Compared to this study, we measure the accuracy of CVE information and the impact of incorrect CVE information in the wild. Moreover, a recent study conducted by Dong et al. [68] examined the quality and consistency of CVE. While their work was solely focused on assessing the quality and consistency of CVE, our research takes a more holistic approach by looking deeper into our dataset. Since we collected a large-scale dataset of Alexa 1M domains for *four years* (157.2M webpages) that can provide a longitudinal, comprehensive view of the entire Web ecosystem and a general trend of Web security practices. Specifically, for each client-side resource (such as each JavaScript library), we longitudinally provide the trend of each library usage, updating behaviors of web developers for each library, how incorrect CVE descriptions of each library affected the Web ecosystem, and new security threats.

11 CONCLUSION

We conduct a longitudinal, large-scale study of the security practice and implications on client-side resources using our collected 157.2M webpages of Alexa Top 1M websites. From our findings, we answered each research question we developed: **RQ1)** 41.2% websites carry at least one vulnerability during our four-year observation period; **RQ2)** The dominant versions of vulnerable JavaScript libraries lag behind in updating to the latest versions: on average 531.2 days with 25,337 websites have delays; **RQ3)** Our CVE validation experiments reveal that 13 CVEs contain incorrect version information, resulting in updating delays or ill-advised updates; **RQ4)** We also identify security issues with outdated techniques, Flash. Our results highlight the importance of necessity on systematic understanding of security on client-side resources.

ACKNOWLEDGMENTS

We thank the anonymous referees and our shepherd, Michael Sirivianos, for their constructive feedback. The authors gratefully acknowledge the support of NSF (2210137, 2335798, 1908021, 1916499, and 2145616). This research was also supported by Science Alliance's StART program and gifts from Google exploreCSR and TensorFlow. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] 1997. ECMA-262, 1st edition, June 1997. https://www.ecma-international.org/wp-content/uploads/ECMA-262_1st_edition_june_1997.pdf. (Accessed on 05/26/2023).

[2] 2008. CVE-2008-4401 : ActionScript in Adobe Flash Player 9.0.124.0 and earlier does not require user interaction in conjunction with (1) the F. https://www.cvedetails.com/cve/CVE-2008-4401/. (Accessed on 05/26/2023).

[3] 2011. CVE-2011-0577 : Unspecified vulnerability in Adobe Flash Player before 10.2.152.26 allows remote attackers to execute arbitrary code. https://www.cvedetails.com/cve/CVE-2011-0577/. (Accessed on 05/26/2023).

[4] 2011. CVE-2011-0578 : Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service (memory co. https://www.cvedetails.com/cve/CVE-2011-0578/. (Accessed on 05/26/2023).

[5] 2011. CVE-2011-0607 : Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service. https://www.cvedetails.com/cve/CVE-2011-0607/. (Accessed on 05/26/2023).

[6] 2011. CVE-2011-0608 : Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service. https://www.cvedetails.com/cve/CVE-2011-0608/. (Accessed on 05/26/2023).

[7] 2011. jQuery 1.2 Released | Official jQuery Blog. https://blog.jquery.com/2007/09/10/jquery-1-2-released/#jQuery_1.1_Compatibility_Plugin. (Accessed on 05/26/2023).

[8] 2012. CVE-2012-5054 : Integer overflow in the copyRawDataTo method in the Matrix3D class in Adobe Flash Player before 11.4.402.265 allows remo. https://www.cvedetails.com/cve/CVE-2012-5054/. (Accessed on 05/26/2023).

[9] 2013. Cross-site Scripting (XSS) in jquery-migrate | Snyk. https://security.snyk.io/vuln/npm:jquery-migrate:20130419. (Accessed on 05/26/2023).

[10] 2013. JS Bin - Collaborative JavaScript Debugging. https://jsbin.com/UQEgAsO/3/edit?html,output. (Accessed on 05/26/2023).

[11] 2013. XSS · Issue #36 · jquery/jquery-migrate. https://github.com/jquery/jquery-migrate/issues/36. (Accessed on 05/26/2023).

[12] 2014. CVE-2014-0510 : Heap-based buffer overflow in Adobe Flash Player 12.0.0.77 allows remote attackers to execute arbitrary code and bypass. https://www.cvedetails.com/cve/CVE-2014-0510/. (Accessed on 05/26/2023).

[13] 2014. Full Disclosure: XSS Reflected JQuery 1.4.2 - Create object option in runtime client-side. https://seclists.org/fulldisclosure/2014/Sep/10. (Accessed on 05/26/2023).

[14] 2014. Scanning Alexa Top 100,000 for JavaScript libraries with known vulerabilities. https://erlend.oftedal.no/blog/static-142.html. (Accessed on 05/26/2023).

[15] 2015. XSS Vulnerability on closeText option of Dialog jQuery UI · Issue #281 · jquery/api.jqueryui.com. https://github.com/jquery/api.jqueryui.com/issues/281. (Accessed on 05/26/2023).

[16] 2016. CVE-2016-1019 : Adobe Flash Player 21.0.0.197 and earlier allows remote attackers to cause a denial of service (application crash) or po. https://www.cvedetails.com/cve/CVE-2016-1019/. (Accessed on 05/26/2023).

[17] 2016. swfobject/swfobject: An open source Javascript framework for detecting the Adobe Flash Player plugin and embedding Flash (swf) files. https://github.com/swfobject/swfobject. (Accessed on 05/26/2023).

[18] 2017. 77% of 433,000 Sites Use Vulnerable JavaScript Libraries. https://snyk.io/blog/77-percent-of-sites-still-vulnerable/. (Accessed on 05/26/2023).

[19] 2017. CVE-2017-3083 : Adobe Flash Player versions 25.0.0.171 and earlier have an exploitable use after free vulnerability in the Primetime SDK. https://www.cvedetails.com/cve/CVE-2017-3083/. (Accessed on 05/26/2023).

[20] 2017. CVE-2017-3084 : Adobe Flash Player versions 25.0.0.171 and earlier have an exploitable use after free vulnerability in the advertising m. https://www.cvedetails.com/cve/CVE-2017-3084/. (Accessed on 05/26/2023).

[21] 2017. Flash Player is no longer available - Google Chrome Help. https://support.google.com/chrome/answer/6258784?hl=en. (Accessed on 05/26/2023).

[22] 2017. JS Bin - Collaborative JavaScript Debugging. https://jsbin.com/qalekeroke/edit?html,output. (Accessed on 05/26/2023).

[23] 2018. CVE-2012-6708 : jQuery before 1.9.0 is vulnerable to Cross-site Scripting (XSS) attacks. The jQuery(strInput) function does not differen. https://www.cvedetails.com/cve/CVE-2012-6708/?q=CVE-2012-6708. (Accessed on 05/26/2023).

[24] 2018. CVE-2015-9251 : jQuery before 3.0.0 is vulnerable to Cross-site Scripting (XSS) attacks when a cross-domain Ajax request is performed wi. https://www.cvedetails.com/cve/CVE-2015-9251/. (Accessed on 05/26/2023).

[25] 2018. JS Bin - Collaborative JavaScript Debugging. https://jsbin.com/palokaxina/edit?html,output. (Accessed on 05/26/2023).

[26] 2018. JS Bin - Collaborative JavaScript Debugging. https://jsbin.com/xemininoniku/edit?html,output. (Accessed on 05/26/2023).

[27] 2019. Compatibility Issue with JQuery 3.4.x | WebDataRocks. https://www.webdatarocks.com/question/compatibility-issue-with-jquery-3-4-x-2/. (Accessed on 05/26/2023).

[28] 2019. CVE-2019-11358 : jQuery before 3.4.0, as used in Drupal, Backdrop CMS, and other products, mishandles jQuery.extend(true, {}, ...) becaus. https://www.cvedetails.com/cve/CVE-2019-11358/. (Accessed on 05/26/2023).

[29] 2020. Cross-site Scripting (XSS) in jquery | CVE-2020-7656 | Snyk. https://security.snyk.io/vuln/SNYK-JS-JQUERY-569619. (Accessed on 05/26/2023).

[30] 2020. CVE-2020-11022 : In jQuery versions greater than or equal to 1.2 and before 3.5.0, passing HTML from untrusted sources - even after sanit. https://www.cvedetails.com/cve/CVE-2020-11022/. (Accessed on 05/26/2023).

[31] 2020. CVE-2020-11022 : In jQuery versions greater than or equal to 1.2 and before 3.5.0, passing HTML from untrusted sources - even after sanit. https://www.cvedetails.com/cve/CVE-2020-11022/. (Accessed on 05/26/2023).

[32] 2020. CVE-2020-11023 : In jQuery versions greater than or equal to 1.0.3 and before 3.5.0, passing HTML containing <option> elements from. https://www.cvedetails.com/cve/CVE-2020-11023/. (Accessed on 05/26/2023).

[33] 2020. Safari 14 and flash player - Apple Community. https://discussions.apple.com/thread/251900220. (Accessed on 05/26/2023).

[34] 2021. Compatibility issues with latest jQuery 3.5.1. https://datatables.net/forums/discussion/67375/compatibility-issues-with-latest-jquery-3-5-1. (Accessed on 05/26/2023).

[35] 2021. CVE-2020-27511 : An issue was discovered in the stripTags and unescapeHTML components in Prototype 1.7.3 where an attacker can cause a Re. https://www.cvedetails.com/cve/CVE-2020-27511/?q=CVE-2020-27511. (Accessed on 05/26/2023).

[36] 2021. End of support for Adobe Flash | Firefox Help. https://support.mozilla.org/en-US/kb/end-support-adobe-flash. (Accessed on 05/26/2023).

[37] 2021. Update on Adobe Flash Player End of Support - Microsoft Edge Blog. https://blogs.windows.com/msedgedev/2020/09/04/update-adobe-flash-end-support/. (Accessed on 05/26/2023).

[38] 2021. Vulnerable Javascript Library. https://beaglesecurity.com/blog/vulnerability/vulnerable-javascript-library.html. (Accessed on 05/26/2023).

[39] 2022. Commits · js-cookie/js-cookie. https://github.com/js-cookie/js-cookie/commits/main. (Accessed on 05/26/2023).

[40] 2022. CVE - CVE. https://cve.mitre.org/index.html. (Accessed on 05/26/2023).

[41] 2022. CVE security vulnerability database. Security vulnerabilities, exploits, references and more. https://www.cvedetails.com/index.php. (Accessed on 05/26/2023).

[42] 2022. Digital 2022: Global Overview Report - DataReportal - Global Digital Insights. https://datareportal.com/reports/digital-2022-global-overview-report. (Accessed on 05/26/2023).

[43] 2022. HTML attribute: crossorigin - HTML: HyperText Markup Language | MDN. https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/crossorigin. (Accessed on 05/26/2023).

[44] 2022. js-cookie/js-cookie: A simple, lightweight JavaScript API for handling browser cookies. https://github.com/js-cookie/js-cookie. (Accessed on 05/26/2023).

[45] 2022. NVD - Vulnerabilities. https://nvd.nist.gov/vuln. (Accessed on 05/26/2023).

[46] 2022. Request.credentials - Web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/Request/credentials. (Accessed on 05/26/2023).

[47] 2023. Browser Support | jQuery. https://jquery.com/browser-support/. (Accessed on 05/26/2023).

[48] 2023. jQuery vs Bootstrap – What Is The Difference? – Remarkable Coder. https://remarkablecoder.com/jquery-vs-bootstrap. (Accessed on 05/26/2023).

[49] 2023. UPDATE: Adobe Flash Player end of support on December 31, 2020 - Microsoft Lifecycle | Microsoft Learn. https://learn.microsoft.com/en-us/lifecycle/announcements/update-adobe-flash-support. (Accessed on 05/26/2023).

[50] 2023. Vulnerability DB | Snyk. https://security.snyk.io/vuln. (Accessed on 05/26/2023).

[51] 2023. *wappalyzer/wappalyzer: Identify technology on websites.* (Accessed on 05/26/2023).

[52] 360. 2023. 360 Browser. https://browser.360.cn/ee/mac/index.html. (Accessed on 05/26/2023).

[53] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 289–305.

[54] Adguard. 2023. AdguardFilters/specific.txt at master · AdguardTeam/AdguardFilters. https://github.com/AdguardTeam/AdguardFilters/blob/master/SpywareFilter/sections/specific.txt. (Accessed on 05/26/2023).

[55] Adobe. 2017. Control access to scripts | Host web page. https://helpx.adobe.com/flash/kb/control-access-scripts-host-web.html. (Accessed on 05/26/2023).

[56] Adobe. 2021. Adobe Flash Player End of Life. https://www.adobe.com/products/flashplayer/end-of-life.html. (Accessed on 05/26/2023).

[57] Adobe. 2021. Create HTML5 Canvas documents in Animate. https://helpx.adobe.com/animate/using/creating-publishing-html5-canvas-document.html. (Accessed on 05/26/2023).

[58] Adobe. 2022. Best practices to convert/publish existing Flash-based projects to HTML5 in Captivate. https://helpx.adobe.com/captivate/kb/best-practices-convert-flash-html5-captivate.html. (Accessed on 05/26/2023).

[59] National Security Agency. 2019. CSA - CONTINUED USE OF ADOBE FLASH INVITES COMPROMISE.PDF. https://media.defense.gov/2019/Sep/25/2002186834/-1/-1/0/CSA%20-%20CONTINUED%20USE%20OF%20ADOBE%

20FLASH%20INVITES%20COMPROMISE.PDF. (Accessed on 05/26/2023).
[60] Pieter Agten, Steven Van Acker, Yoran Brondsema, Phu H. Phung, Lieven Desmet, and Frank Piessens. 2012. JSand: Complete Client-Side Sandboxing of Third-Party JavaScript without Browser Modifications. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC '12)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/2420950.2420952
[61] Danny E. Alvarez, Daniel B. Correa, and Fernando I. Arango. 2016. An analysis of XSS, CSRF and SQL injection in colombian software and web site development. In *2016 8th Euro American Conference on Telematics and Information Systems (EATIS)*. 1–5. https://doi.org/10.1109/EATIS.2016.7520140
[62] Adam Barth, Collin Jackson, and John C Mitchell. 2008. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*. 75–88.
[63] Bootstrap. 2023. Bootstrap · The most popular HTML, CSS, and JS library in the world. https://getbootstrap.com/. (Accessed on 05/26/2023).
[64] William J Buchanan, Scott Helme, and Alan Woodward. 2018. Analysis of the adoption of security headers in HTTP. *IET Information Security* 12, 2 (2018), 118–126.
[65] cdnjs. 2023. cdnjs - The #1 free and open source CDN built to make life easier for developers. https://cdnjs.com/. (Accessed on 05/26/2023).
[66] Chromium. 2021. Flash Roadmap. https://www.chromium.org/flash-roadmap/#TOC-Flash-Support-Removed-from-Chromium-Target:-Chrome-87---Dec-2020-. (Accessed on 05/26/2023).
[67] Nurullah Demir, Tobias Urban, Kevin Wittek, and Norbert Pohlmann. 2021. Our (in)Secure Web: Understanding Update Behavior of Websites and Its Impact on Security. In *Passive and Active Measurement*. Springer International Publishing, Cham, 76–92.
[68] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports.. In *USENIX Security Symposium*. 869–885.
[69] Carlos Duarte, Inês Matos, João Vicente, Ana Salvado, Carlos M. Duarte, and Luís Carriço. 2016. Development Technologies Impact in Web Accessibility. In *Proceedings of the 13th International Web for All Conference* (Montreal, Canada) *(W4A '16)*. Association for Computing Machinery, New York, NY, USA, Article 6, 4 pages. https://doi.org/10.1145/2899475.2899498
[70] J. Emigh. 2006. New Flash player rises in the Web-video market. *Computer* 39, 2 (2006), 14–16. https://doi.org/10.1109/MC.2006.66
[71] F-Secure. 2011. News from the Lab Archive : January 2004 to September 2015. https://archive.f-secure.com/weblog/archives/00002226.html. (Accessed on 05/26/2023).
[72] GitHub. 2021. Update regex for striptags method to prevent regex dos by jwestbrook · Pull Request #349 · prototypejs/prototype. https://github.com/prototypejs/prototype/pull/349. (Accessed on 05/26/2023).
[73] Google. 2017. Saying goodbye to Flash in Chrome. https://www.blog.google/products/chrome/saying-goodbye-flash-chrome/. (Accessed on 05/26/2023).
[74] Hao He, Lulu Chen, and Wenpu Guo. 2017/03. Research on Web Application Vulnerability Scanning System based on Fingerprint Feature. In *Proceedings of the 2017 International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2017)*. Atlantis Press, 150–155. https://doi.org/10.2991/mecae-17.2017.27
[75] Isotope. 2023. Isotope - Filter & sort magical layouts. https://isotope.metafizzy.co/. (Accessed on 05/26/2023).
[76] jQuery. 2023. jQuery. https://jquery.com/. (Accessed on 05/26/2023).
[77] jquery cookie. 2015. carhartl/jquery-cookie: No longer maintained, superseded by JS Cookie:. https://github.com/carhartl/jquery-cookie. (Accessed on 05/26/2023).
[78] jquery migrate. 2023. jquery/jquery-migrate: A development tool to help migrate away from APIs and features that have been or will be removed from jQuery core. https://github.com/jquery/jquery-migrate. (Accessed on 05/26/2023).
[79] jQuery UI. 2023. jQuery UI. https://jqueryui.com/. (Accessed on 05/26/2023).
[80] jsDelivr. 2023. jsDelivr - A free, fast, and reliable CDN for open source. https://www.jsdelivr.com/. (Accessed on 05/26/2023).
[81] Gregg Keizer. 2011. RSA hackers exploited Flash zero-day bug | Computerworld. https://www.computerworld.com/article/2507619/rsa-hackers-exploited-flash-zero-day-bug.html. (Accessed on 05/26/2023).
[82] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2018. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. *arXiv preprint arXiv:1811.00918* (2018).
[83] Sebastian Lekies, Ben Stock, and Martin Johns. 2013. 25 Million Flows Later: Large-Scale Detection of DOM-Based XSS. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 1193–1204. https://doi.org/10.1145/2508859.2516703
[84] Ada Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. 2016. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner
[85] Fabian Marquardt and Lennart Buhl. 2021. Déjà Vu? Client-Side Fingerprinting and Version Detection of Web Application Software. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. 81–89. https://doi.org/10.1109/LCN52139.2021.9524885
[86] Modernizr. 2023. Modernizr: the feature detection library for HTML5/CSS3. https://modernizr.com/. (Accessed on 05/26/2023).
[87] Moment. 2023. Moment.js | Home. https://momentjs.com/. (Accessed on 05/26/2023).
[88] Mozilla. 2021. End of support for Adobe Flash | Firefox Help. https://support.mozilla.org/en-US/kb/end-support-adobe-flash. (Accessed on 05/26/2023).
[89] Mozilla. 2022. Subresource Integrity - Web security | MDN. https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity. (Accessed on 05/26/2023).
[90] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 736–747.
[91] NIST. 2018. NVD - CVE-2018-9206. https://nvd.nist.gov/vuln/detail/CVE-2018-9206. (Accessed on 05/26/2023).
[92] Frolin Ocariza, Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. 2013. An Empirical Study of Client-Side JavaScript Bugs. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 55–64. https://doi.org/10.1109/ESEM.2013.18
[93] Polyfill. 2023. Polyfill.io. https://polyfill.io/v3/. (Accessed on 05/26/2023).
[94] Popper. 2023. Tooltip & Popover Positioning Engine. https://popper.js.org/. (Accessed on 05/26/2023).
[95] Prototype. 2015. Prototype JavaScript framework: a foundation for ambitious web applications. http://prototypejs.org/. (Accessed on 05/26/2023).
[96] Nur Aini Rakhmawati, Sayekti Harits, Deny Hermansyah, and Muhammad Ariful Furqon. 2018. A Survey of Web Technologies Used in Indonesia Local Governments. *SISFO Vol 7 No 3* 7 (2018).
[97] RequireJS. 2018. RequireJS. https://requirejs.org/. (Accessed on 05/26/2023).
[98] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. 2010. An Analysis of the Dynamic Behavior of JavaScript Programs. *SIGPLAN Not.* 45, 6 (jun 2010), 1–12. https://doi.org/10.1145/1809028.1806598
[99] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. 2020. Complex security policy? a longitudinal analysis of deployed content security policies. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*.
[100] Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. 2010. FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications.. In *NDSS*.
[101] IMQ Minded Security. 2013. IMQ Minded Security Blog: "jQuery Migrate"' is a Sink, too?! https://blog.mindedsecurity.com/2013/04/jquery-migrate-is-sink-too.html. (Accessed on 09/05/2023).
[102] Statcounter. 2023. Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share/desktop/worldwide. (Accessed on 05/26/2023).
[103] statista. 2023. Internet usage worldwide – statistics & facts. https://www.statista.com/topics/1145/internet-usage-worldwide/. (Accessed on 05/26/2023).
[104] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. 2021. Who's hosting the block party? studying third-party blockage of csp and sri. In *Network and Distributed Systems Security (NDSS) Symposium 2021*.
[105] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. 2017. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 971–987. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/stock
[106] Yuta Takata, Hiroshi Kumagai, and Masaki Kamizono. 2021. The Uncontrolled Web: Measuring Security Governance on the Web. *IEICE Transactions on Information and Systems* 104, 11 (2021), 1828–1838.
[107] Underscore. 2022. Underscore.js. https://underscorejs.org/. (Accessed on 05/26/2023).
[108] Semantic Versioning. 2023. Semantic Versioning 2.0.0. https://semver.org/. (Accessed on 05/26/2023).
[109] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2007. Cross site scripting prevention with dynamic data tainting and static analysis.. In *NDSS*, Vol. 2007. 12.
[110] W3. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/#cross-origin-data-leakage. (Accessed on 05/26/2023).
[111] W3.org. 2023. HTML Standard. https://html.spec.whatwg.org/multipage/iframe-embed-object.html#the-object-element. (Accessed on 05/26/2023).

[112] Whatwg. 2023. HTML Standard. https://html.spec.whatwg.org/multipage/urls-and-fetching.html#cors-settings-attributes. (Accessed on 05/26/2023).
[113] WordPress. 2022. Enable jQuery Migrate Helper – WordPress plugin. https://wordpress.org/plugins/enable-jquery-migrate-helper/. (Accessed on 05/26/2023).
[114] WordPress. 2022. Enable jQuery Migrate Helper – WordPress plugin. https://wordpress.org/plugins/enable-jquery-migrate-helper/#description. (Accessed on 05/26/2023).
[115] WordPress. 2023. Configuring Automatic Background Updates. https://wordpress.org/support/article/configuring-automatic-background-updates/. (Accessed on 05/26/2023).
[116] Qiushi Wu and Kangjie Lu. 2021. On the feasibility of stealthily introducing vulnerabilities in open-source software via hypocrite commits. In *Proc. Oakland*.
[117] Chuan Yue and Haining Wang. 2009. Characterizing Insecure Javascript Practices on the Web. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain) *(WWW '09)*. Association for Computing Machinery, New York, NY, USA, 961–970. https://doi.org/10.1145/1526709.1526838
[118] ZDNET. 2018. Zero-day in popular jQuery plugin actively exploited for at least three years | ZDNET. https://www.zdnet.com/article/zero-day-in-popular-jquery-plugin-actively-exploited-for-at-least-three-years/. (Accessed on 05/26/2023).
[119] ZDNET. 2021. Flash version distributed in China after EOL is installing adware | ZDNET. https://www.zdnet.com/article/flash-version-distributed-in-china-after-eol-is-installing-adware/. (Accessed on 05/26/2023).

APPENDIX

Revealed Vulnerabilities. As we have seen in Figure 5, we showed the number of websites affected by incorrect CVE information. Figure 14(a) shows that `jQuery-Migrate` has a CVE stating that the versions before 1.2.1 are vulnerable. However, we revealed that the vulnerability expands from 1.0.0 to 3.0.0 (an *understated* version case). The red region in the graph shows the domains using newly revealed vulnerable versions. For Figure 14(d), and Figure 14(e) same principle applies to Figure 14(a). For Figure 14(b) and Figure 14(c), the CVE states that more versions are vulnerable than the versions that are truly vulnerable (an *overstated* version case).

Top 5 Affected Versions. Figure 15 shows the top 5 affected versions for `Bootstrap`, `jQuery-UI`, and `Prototype`. Figure 15(a) has version 3.3.7 as the dominant version as shown in Table 1. However, it is affected by all of the CVEs we have discovered. From Figure 15(a), version 3.3.7 is decreasing. However, it is difficult to say it is caused by the disclosed vulnerability. Similarly, for Figure 15(b), version 1.7.1 is the most dominant version, and it decreases slightly. A similar result is shown but more pronounced because this clearly shows that the decrease is not affected by disclosed CVEs. For Figure 15(c), the dominant version is also in the top 5 affected versions. This figure also shows that there is no effect on updating behavior with disclosed CVEs.

Comparison of Overstated Versions and Understated Versions. As we have seen in Figure 4, a similar analysis is conducted. For `Moment`, `jQuery-Migrate`, `jQuery-UI`, and `Prototype`, all have understated versions indicated in red. As we can see with `jQuery-Migrate` and `Prototype`, more than half of the existing versions are understated versions. This indicates how inaccurate CVE information is. The `Bootstrap` only does not have understated versions, it only has revealed overstated versions. This is understated but has drawbacks explained in Section 6.4.

Top 10 Disclosed CVEs for WordPress. Table 4 shows the top 10 disclosed CVEs for `WordPress` (the most recent 5 CVEs and the most critical CVEs). `WordPress` is one of the interesting factors in our observation of JavaScript library updates. `WordPress` is one of the most popular content management systems. In our dataset, 26.9% of websites use `WordPress` as shown in Figure 9. `WordPress` has released a total number of 606 versions, excluding beta and RC (Release Candidate) versions. In our dataset, we found 521 versions (excluding the last two patches of each version from v3.7 to v5.9, and the 6.0 branch because they are released after our collection period). We further look into disclosed vulnerabilities for `WordPress`, and we found a total of 6,155 disclosed CVEs as of May. 2023. From 6,155 disclosed CVEs, we looked at the five most severe vulnerabilities (highest ranked in CVSS score) and five most recent CVEs (recent CVEs only have medium severity score). From what we have found, an average of 97.7% of websites is vulnerable according to the top 5 recently disclosed CVEs and 0.36% of websites are vulnerable according to the top 5 most severe CVEs. This indicates that `WordPress` tries to fix vulnerabilities as version updates and most websites are using relatively recent versions of `WordPress`. This implies in Section 7 that `WordPress` has an Auto-Update function which helps administrators/developers to keep up with the most recent version of the software.

CVE PoC code. We reimplemented this PoC code from CVE-2020-7656 to see which version of jquery was truly affected.

```
<html>
<head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
        jquery/1.8.3/jquery.js">
    </script>
</head>
<body>
  <h1>CVE-2020-7656</h1>
  <div id="CVE-2020-7656"></div>
  <script>
    $("#CVE-2020-7656").load('inject.html');
  </script>
  </div>
</body>
</html>
```

Listing 1: Modified PoC for CVE-2020-7656 [29]

```
    <div id="CVE-2020-7656">
       <script>alert('Arbitrary Code Execution');
    </script></div>
```

Listing 2: Modified PoC for CVE-2020-7656 (inject.html) [29]

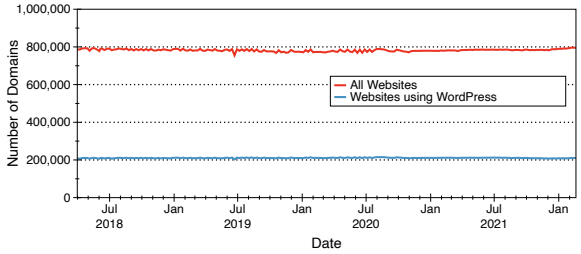


Figure 9: WordPress Usage. Of our collected websites (on average, 782,300), 26.9% websites are built with WordPress.

Browser	Market share*	Flash Support**
Chrome	66.45%	N
Edge	10.8%	N
Safari	9.59%	N
Firefox	7.16%	N
Opera	3.09%	N
IE	0.81%	N
360 Browser	0.66%	Y
Yandex Browser	0.39%	N
QQ Browser	0.20%	N
Edge Legacy	0.16%	N

*: Desktop Browser Market Share Worldwide (Apr. 2022 – Apr. 2023) [102]
**: Manually tested on May 26, 2023.

Table 3: Top 10 Web Browser Market share and Flash Support. 360 Browser still supports Adobe Flash even though it is officially no longer supported and major browsers such as Chrome completely removed the Flash components.

CVE ID	Disclosed Date	Ver	Patched Ver	Patched Date	#Websites
CVE-2022-21664	01/06/2022	4.1.34 ~ 5.8.3	5.8.3	01/06/2022	124,556
CVE-2022-21663	01/06/2022	3.7.37 ~ 5.8.3	5.8.3	01/06/2022	127,440
CVE-2022-21662	01/06/2022	3.7.37 ~ 5.8.3	5.8.3	01/06/2022	127,440
CVE-2022-21661	01/06/2022	3.7.37 ~ 5.8.3	5.8.3	01/06/2022	127,440
CVE-2021-44223	11/25/2021	< 5.8	5.8	07/20/2021	121,214
CVE-2012-2400	04/21/2012	< 3.3.2	3.3.2	04/20/2012	545
CVE-2012-2399	04/21/2012	< 3.5.2	3.5.2	06/21/2013*	913
CVE-2011-3125	08/10/2011	< 3.1.3	3.1.3	05/25/2011	380
CVE-2011-3122	08/10/2011	< 3.1.3	3.1.3	05/25/2011	380
CVE-2009-2853	08/18/2009	< 2.8.3	2.8.3	08/03/2009	30

*: the vulnerability was disclosed more than a year before the patched version was released

Table 4: Top 10 disclosed CVEs for WordPress The first 5 are the most recent CVEs, and the last 5 are the most severe CVEs.

Lib.	Hostname	%	Lib.	Hostname	%
jQuery	ajax.googleapi.com	26.0%	isotope	secureservercdn.net	3.3%
	code.jquery.com	10.0%		cdn.shopify.com	2.1%
	cdnjs.cloudflare.com	7.1%		cdn.jsdelivr.net	0.8%
jQuery-Migrate	c0.wp.com	22.1%	popper	cdnjs.cloudflare.com	77.3%
	cdnjs.cloudflare.com	4.5%		cdn.jsdelivr.net	9.0%
	secureservercdn.net	2.3%		unpkg.com	2.1%
Bootstrap	maxcdn.bootstrapcdn.com	33.6%	polyfill	polyfill.io	45.4%
	widget.trustpilot.com	10.0%		cdn.polyfill.io	30.8%
	stackpath.bootstrapcdn.com	9.7%		static.parastorage.com	4.1%
jQuery-UI	ajax.googleapis.com	49.6%	moment	cdnjs.cloudflare.com	51.8%
	code.jquery.com	30.7%		cdn.jsdelivr.net	6.1%
	cdnjs.cloudflare.com	4.2%		momentjs.com	1.7%
Modernizr	cdnjs.cloudflare.com	32.4%	swfobject	ajax.googleapis.com	49.1%
	cdn.shopify.com	21.8%		cdnjs.cloudflare.com	3.0%
	cdn.prestosports.com	1.0%		s0.wp.com	2.6%
JS-Cookie	cdn.jsdelivr.net	21.1%	jquery-cookie	cdnjs.cloudflare.com	62.6%
	c0.wp.com	12.3%		cdn.shopify.com	8.4%
	cdnjs.cloudflare.com	11.5%		c0.wp.com	0.9%
Underscore	c0.wp.com	20.5%	prototype	ajax.googleapis.com	27.7%
	cdnjs.cloudflare.com	13.3%		strato-editor.com	3.7%
	secureservercdn.net	1.5%		cdnjs.cloudflare.com	2.2%

Table 5: Top 3 CDNs for JavaScript Library.

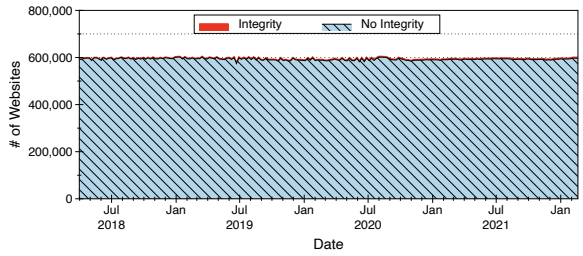


Figure 10: Subresource Integrity (SRI). 99.7% websites have at least one externally-hosted JS library without integrity.

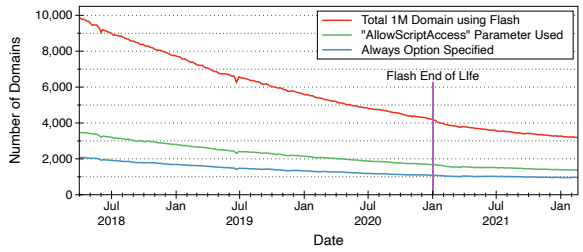


Figure 11: `AllowScriptAccess` Parameter and Insecure `Always` Option.

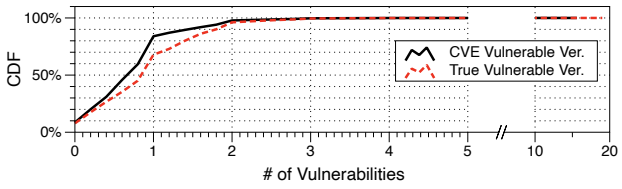
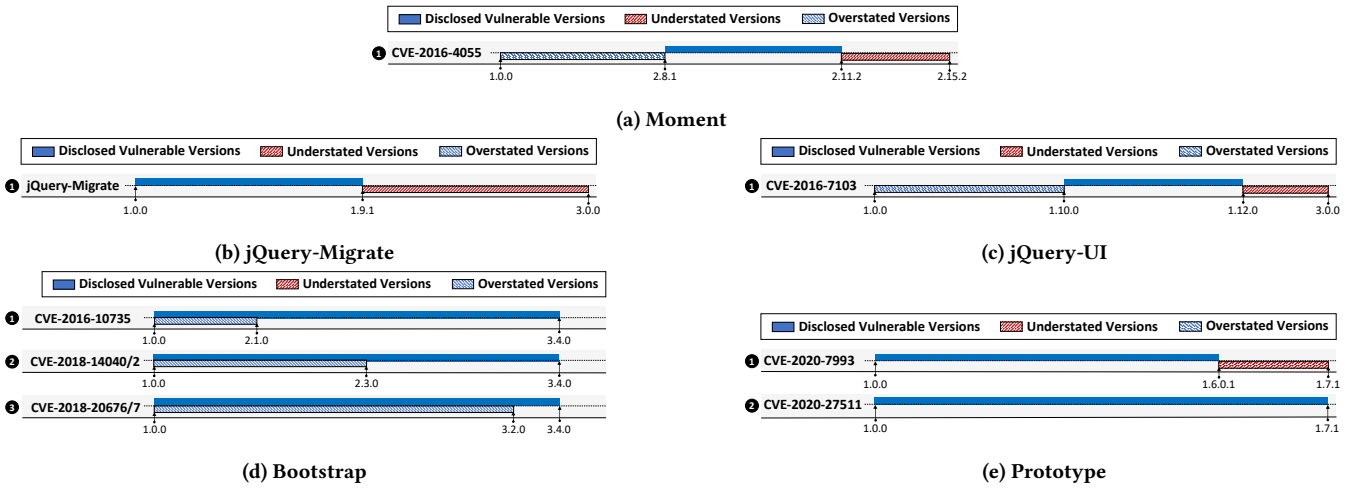


Figure 12: CDF of the Avg. Number of Vulnerabilities per Website. The average number of True Vulnerable Versions (mean: 0.97, median: 0.96) is higher than the one of CVE vulnerable versions (mean: 0.79, median: 0.75). This indicates that a significant number of True Vulnerable Versions are not disclosed due to incorrect CVE descriptions.



(a) Moment

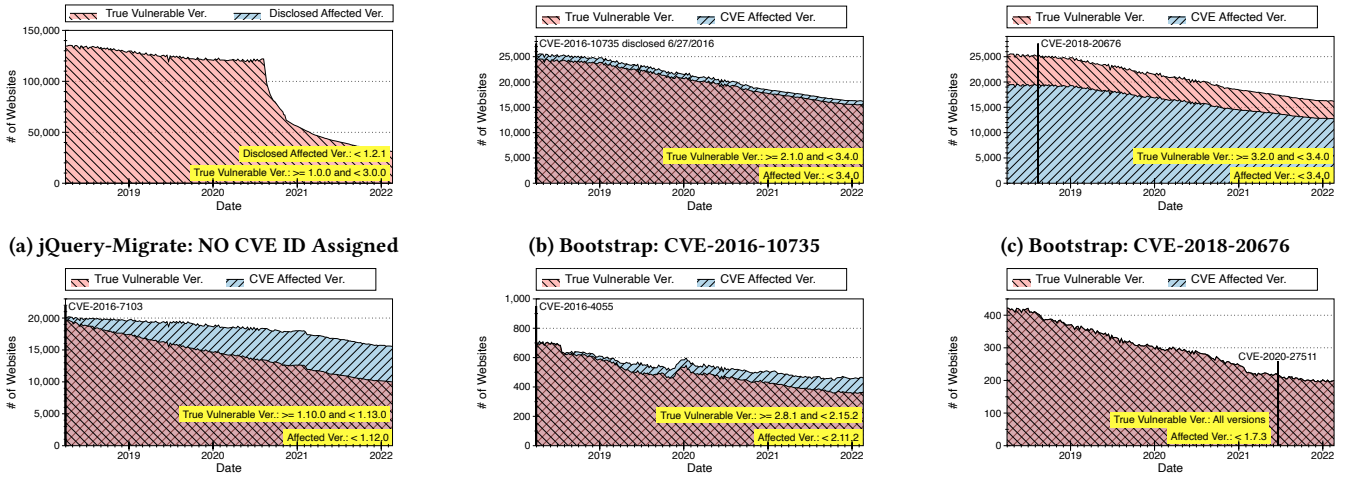
(b) jQuery-Migrate

(c) jQuery-UI

(d) Bootstrap

(e) Prototype

Figure 13: Comparison of Disclosed CVE Vulnerable Version and Understated/Overstated Versions (Moment, jQuery-Migrate, jQuery-UI, Bootstrap, and Prototype).



(a) jQuery-Migrate: NO CVE ID Assigned

(b) Bootstrap: CVE-2016-10735

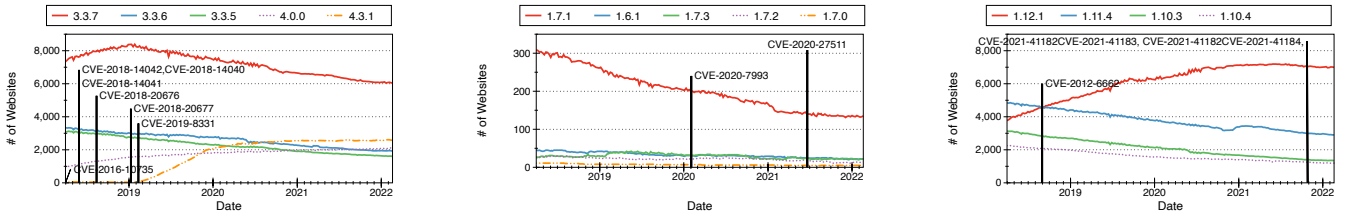
(c) Bootstrap: CVE-2018-20676

(d) jQuery-UI: CVE-2016-7103

(e) Moment: CVE-2016-4055

(f) Prototype: CVE-2020-27511

Figure 14: CVV and TVV of jQuery-Migrate, jQuery-UI, Bootstrap, Moment, and Prototype.



(a) Bootstrap

(b) Prototype

(c) jQuery-UI

Figure 15: Top 5 Affected Versions of Bootstrap, Prototype, and jQuery-UI.

Top 10K Domains using GitHub URL			Stats for GitHub	
Website	**Ranking**	**GitHub URL**	**Most Used Repository**	**#Websites**
kinogo.cc	594	kodir2.github.io/.../actualize.js	partnercoll.github.io/actualize.js	4
uptobox.com	744	blueimp.github.io/.../jquery.ui.widget.js	malsup.github.com/jquery.form.js	2
cnnindonesia.com	985	malsup.github.com/jquery.form.js	afarkas.github.io/.../lazysizes.min.js	2
wittyfeed.com	1639	hammerjs.github.io/.../hammer.min.js	blueimp.github.io/.../jquery.ui.widget.js	2
baskino.me	1,680	partnercoll.github.io/actualize.js	gitcdn.github.io/.../bootstrap-toggle.min.js	2
the-star.co.ke	2,029	radioafricagroup.github.io/.../cookiestrip.min.js radioafricagroup.github.io/.../jquery.popup.js	kodir2.github.io/actualize.js owlcarousel2.github.io/.../owl.carousel.js	2 2
colourpop.com	2,551	klevron.github.io/.../OrbitControls.js	weblion777.github.io/hdvb.js	2
canalrcn.com	3,274	afarkas.github.io/.../lazysizes.min.js	hammerjs.github.io/.../hammer.min.js	1
dostor.org	3,329	owlcarousel2.github.io/.../owl.carousel.js	malihu.github.io/.../jquery.mCustomScrollbar.concat.min.js	1
morningstar.com	3,976	jonathantneal.github.io/.../svg4everybody.min.js	kenwheeler.github.io/.../slick.js	1
raw.githubusercontent.com	4,087	assets-cdn.github.com/.../compat-432e5...a3c.js	actlz.github.io/actualize.js	1
bintjbeil.org	4,518	malihu.github.io/.../jquery.mCustomScrollbar.concat.min.js	assets-cdn.github.com/.../compat-432e5...a3c.js	1
vkmag.com	5,772	owlcarousel2.github.io/.../owl.carousel.js	blueimp.github.io/.../jquery.blueimp-gallery.min.js	1
atresplayer.com	6,455	malsup.github.com/jquery.form.js	blueimp.github.io/.../canvas-to-blob.min.js	1
kinoserv.net	6,714	weblion777.github.io/hdvb.js partnercoll.github.io/actualize.js	blueimp.github.io/.../load-image.all.min.js blueimp.github.io/.../tmpl.min.js	1 1
hdkinoteatr.com	6,820	weblion777.github.io/hdvb.js partnercoll.github.io/actualize.js	blueimp.github.io/.../jquery.fileupload-audio.js blueimp.github.io/.../jquery.fileupload-image.js	1 1
ohmynews.com	6,997	kenwheeler.github.io/.../slick.js	blueimp.github.io/.../jquery.fileupload-process.js	1
orangebookvalue.com	7,029	gitcdn.github.io/.../bootstrap-toggle.min.js	blueimp.github.io/.../jquery.fileupload-ui.js	1
bddatabase.net	7,971	hayageek.github.io/.../jquery.uploadfile.min.js	blueimp.github.io/.../jquery.fileupload-validate.js	1
noticiasrcn.com	8,008	afarkas.github.io/.../lazysizes.min.js	blueimp.github.io/.../jquery.fileupload-video.js	1
kinoplen.ru	8,018	"partnercoll.github.io/replace.js partnercoll.github.io/actualize.js	blueimp.github.io/.../jquery.fileupload.js blueimp.github.io/.../jquery.iframe-transport.js	1 1
english-films.com	8,242	actlz.github.io/actualize.js	hayageek.github.io/.../jquery.uploadfile.min.js	1
uptostream.com	8,796	blueimp.github.io/.../jquery.ui.widget.js blueimp.github.io/.../tmpl.min.js blueimp.github.io/.../load-image.all.min.js blueimp.github.io/.../canvas-to-blob.min.js blueimp.github.io/.../jquery.blueimp-gallery.min.js blueimp.github.io/.../jquery.iframe-transport.js blueimp.github.io/.../jquery.fileupload.js blueimp.github.io/.../jquery.fileupload-process.js blueimp.github.io/.../jquery.fileupload-image.js blueimp.github.io/.../jquery.fileupload-audio.js blueimp.github.io/.../jquery.fileupload-video.js blueimp.github.io/.../jquery.fileupload-validate.js blueimp.github.io/.../jquery.fileupload-ui.js	jonathantneal.github.io/.../svg4everybody.min.js klevron.github.io/.../OrbitControls.js partnercoll.github.io/replace.js radioafricagroup.github.io/.../jquery.popup.js radioafricagroup.github.io/.../cookiestrip.min.js	1 1 1 1 1
mega-mult.ru	9,165	kodir2.github.io/actualize.js		
rstudio.com	9,610	gitcdn.github.io/.../bootstrap-toggle.min.js		

Table 6: Top 10k Websites using JS Library directly from GitHub Repository. The left half of the table indicates domains representing libraries. The right half of the table indicates the number of domains using the same repositories