An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources

Bertil Chapuis
UNIL – HEC Lausanne
Switzerland
bertil.chapuis@unil.ch

Olamide Omolola
TU Graz
Austria
olamide.omolola@iaik.tugraz.at

Mauro Cherubini
UNIL – HEC Lausanne
Switzerland
mauro.cherubini@unil.ch

Mathias Humbert
armasuisse S+T
Switzerland
mathias.humbert@armasuisse.ch

Kévin Huguenin
UNIL – HEC Lausanne
Switzerland
kevin.huguenin@unil.ch

ABSTRACT

Web developers can (and do) include subresources such as scripts, stylesheets and images in their webpages. Such subresources might be stored on content delivery networks (CDNs). This practice creates security and privacy risks, should a subresource be corrupted. The subresource integrity (SRI) recommendation, released in mid-2016 by the W3C, enables developers to include digests in their webpages in order for web browsers to verify the integrity of subresources before loading them. In this paper, we conduct the first large-scale longitudinal study of the use of SRI on the Web by analyzing massive crawls (≈3B URLs) of the Web over the last 3.5 years. Our results show that the adoption of SRI is modest (≈3.40%), but grows at an increasing rate and is highly influenced by the practices of popular library developers (e.g., Bootstrap) and CDN operators (e.g., jsDelivr). We complement our analysis about SRI with a survey of web developers (N=227): It shows that a substantial proportion of developers know SRI and understand its basic functioning, but most of them ignore important aspects of the recommendation. The results of the survey also show that the integration of SRI by developers is mostly manual – hence not scalable and error prone. This calls for a better integration of SRI in build tools.

CCS CONCEPTS

• **Security and privacy** → *Web protocol security*; Hash functions and message authentication codes.

KEYWORDS

web security; subresource integrity; common crawl

ACM Reference Format:
Bertil Chapuis, Olamide Omolola, Mauro Cherubini, Mathias Humbert, and Kévin Huguenin. 2020. An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380092

1 INTRODUCTION

The Web is a set of interlinked resources identified by their URLs. A significant portion of these resources consists of HTML webpages that include navigable links and subresources, such as scripts, stylesheets, images or videos. A change in a subresource can affect the webpage that includes it.

With the advent of content delivery networks (CDNs), an increasing number of subresources are hosted by third-party providers (in this paper, we will refer to these as *external subresources*, also called cross-domain subresources). The advantages provided by such platforms include reduced costs and latency as well as increased reliability. However, their usage comes at a security price: A subresource can be altered (accidentally or not) upon transmission from these third-party providers or directly on them, as it was the case for the British Airways in 2018 [14]. The consequences can be dramatic, including the theft of user credentials (i.e., on login pages) and credit card data (i.e., on payment pages), malware injection, and website defacement (i.e., modification of the content). In general, when an external subresource is included in a webpage, there is no guarantee that its content will remain the same.

The subresource integrity (SRI) recommendation [46], released in mid-2016 by the W3C, addresses this issue by enabling web developers/webmasters to include digests in order for web browsers to verify the integrity of subresources before loading them. SRI is implemented in the vast majority of desktop and mobile browsers. Unfortunately, no in-depth analysis about the adoption and the understanding of SRI (by web developers) has been performed so far. Most existing works [25, 39] focus on modest-size datasets of webpages, focus on one snapshot only, and only look at the basic statistics, e.g., they do not study the main *factors* behind the adoption/usage of SRI. Our work fills this gap (i) by conducting the first large-scale longitudinal study on the adoption/usage of SRI on the Web, and (ii) by surveying web developers regarding their understanding and usage of SRI.

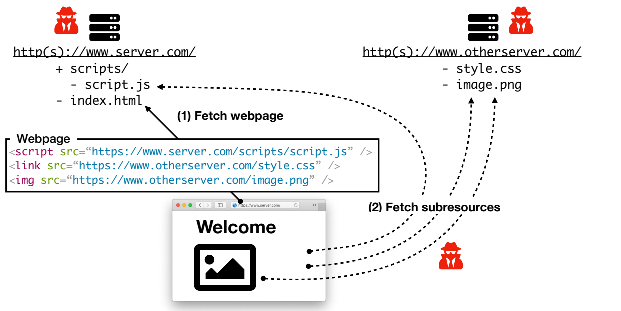
Contributions. By relying on a massive dataset of about 3B URLs, we first thoroughly analyze the use of the SRI recommendation on the Web over the last 3.5 years. We began our analysis in May 2016, right before the official release of the SRI recommendation and took a snapshot approximately every six months. Our analysis of the 3B webpages shows the following: first, we measure the extent of the most typical threat, i.e., the use of external subresources, and we find that more than 80% of the webpages include at least one external subresource; second, we study the adoption and usage of SRI over time and observe an increasing, but still modest usage, from less than 1% in October 2017 to 3.40% in September 2019. We observe that the use of SRI is linked to a number of popular libraries and CDN operators, including Bootstrap and jsDelivr, that provide snippets of code that use SRI for including their subresources.

Given the results of our large-scale analysis of SRI adoption and related security mechanisms, we evaluated the extent of knowledge of web developers about SRI. To do so, we conducted an online survey with 227 respondents asking if web developers are aware of the risks, are aware of SRI, to what extent they understand the implementation of SRI, and what their current practices are when using SRI. Particular care was put in the survey instrument that was designed and tested iteratively before deployment. Panelists were recruited among the Wordpress and NPM package repository contributors. Our results show that about two thirds of the respondents could identify the main threats in subresource usage, such as malicious code injection or cross-site scripting, but most of them ignore some important aspects of SRI implementation, e.g., the cases where the digests used in SRI are malformed or multiple. They also show that the integration of SRI by developers is mostly manual – hence not scalable and error prone. This calls for a better integration of SRI in build tools.

2 SYSTEM AND THREAT MODEL

We consider a webpage hosted on a given server (`www.server.com`). The webpage includes a number of subresources such as scripts (JavaScript, through a `script` element), stylesheets (CSS, through a `link`), and images (through an `img`). The inclusion is done *by reference*, i.e., the subresources are not copied in the webpage. The subresources can be internal (stored on the same server as the webpage, possibly in a different volume or folder, e.g., `scripts/`) or external/cross-domain (stored on a different server, e.g., `www.otherserver.com`). Typical external subresources include subresources found on another website or hosted on a mirror or content delivery network for reliability and performance reasons (e.g., the jQuery library). This last case opens the door to cross-domain script inclusion risks. When a user visits the webpage, the browser first fetches the webpage from the server and then it fetches its subresources. Finally, it renders the webpage to the user.

We consider the threat where the content of the subresource is altered, meaning that it did not correspond to the initial content of the subresource when it was included in the webpage (i.e., what the web developer intended to include). Such a situation can occur in multiple cases: (i) because the communication channel between the client and the server was compromised by an adversary (e.g., an Internet service provider), (ii) because the storage of the server was compromised by an adversary (e.g., a hacker who broke into the server or a malicious mirror operator), or (iii) simply because the subresource was changed by its maintainer. A subresource integrity threat can have important consequences. For instance, a corrupted script can – among other things – compromise the visitors' devices (e.g., by redirecting them to a malicious website), steal their private data (e.g., passwords and credit card information), track them, or shock them (and compromise the reputation of the author) by changing the content of the webpage (i.e., defacing). Figure 1 depicts the system and threat model for subresource integrity.



```
<script src="https://www.server.com/scripts/script.js" />
<link src="https://www.otherserver.com/style.css" />
<img src="https://www.otherserver.com/image.png" />
```

Figure 1: System and threat model for subresource integrity.

3 THE W3C SRI RECOMMENDATION

SRI [46] enables web developers to specify an integrity attribute for some types of subresources they include in their webpages, in such a way that the user agent can verify their integrity before loading them. This guarantees that the content of the subresources corresponds to what the developers intended to include, specifically that it has not changed. As of September 2019, SRI covers `script` (i.e., JavaScript) and `link` (i.e., CSS) elements and it is fully supported by Chrome, Firefox, Opera, Safari, and partially by Edge. A typical use of SRI is as follows (in the head section of the HTML page):

```
<script src="https://www.server.com/script.js"
        integrity="sha256-47D...␣sha512-8HB..." />
```

An integrity attribute contains one or multiple space-separated hash expressions. Each hash expression is composed of the name of a hashing algorithm (i.e., `sha256`, `sha384` or `sha512`) and a base64-encoded digest generated with the corresponding algorithm. The content of a subresource is said to match a hash expression if the digest of the subresource is equal to the digest specified in the expression. When rendering the HTML snippet above, the browser first fetches the subresource (i.e., `script.js`). The browser tries to match the content of the subresource to the different hash expressions specified in the integrity attribute and loads the subresource according to the following rules: (1) When the attribute contains a *single* hash expression, the subresource is loaded if it matches it; (2) when the attribute contains *multiple* expressions generated with the *same* hash algorithm, the subresource is loaded if it matches one of them; and (3) when the attribute contains *multiple* expressions generated with *different* hash algorithms, the subresource is loaded if it matches one of the hash expressions with the *strongest* digest (`sha512` > `sha384` > `sha256`). If the integrity attribute is empty or malformed, the subresource is loaded nevertheless. Note that since SRI "fixes" the content of the subresources, it is not appropriate for subresources that can change (e.g., latest version of a library).

Content Security Policy (CSP) directives (specified in HTML `meta` elements or HTTP headers, e.g., `Content-Security-Policy: require-sri-for script;`) can be used to force web developers to specify a *valid* integrity attribute for each subresource. In this case, subresources without an integrity attribute or with a malformed one are *not* loaded. Such a mechanism enables the separation of concerns between web developers and system administrators. More recently, these directives are being abandoned by Web browsers and their removal from the recommendation is scheduled [8].

4 LARGE-SCALE ANALYSIS

In this section, we report on the large-scale analysis of the use of SRI on the Web. We describe our data sources and methodology and then report on the results.

4.1 Data sources

We rely on two main data sources: large-scale crawls of the web and a popularity-based ranking of domains.

Web Crawl: Common Crawl. The Common Crawl (*CC*) dataset is a collection of snapshots of the Web [12]. It contains one snapshot per month, since 2011-01. Each snapshot is available in the WARC format, which contains the raw data of the webpages (HTTP headers and HTML content but not the content of the subresources). The latest snapshot of Common Crawl (2019-09) contains 2,954,836,069 URLs [13]. Note that the number of URLs for a given website does not necessarily reflect its number of webpages, rather it depends on its architecture (i.e., single page vs. multi-page applications [29]). Each snapshot (in each of the format) is divided into multiple archives so as to enable parallel and distributed processing of the dataset. The *CC* dataset is publicly available on `Amazon S3`, where it can be processed and analyzed using `Amazon EMR`.

Domain Name Ranking: Cisco Umbrella 1 Million (Top1m). The Cisco Umbrella 1 Million dataset (*Top1m*) ranks popular domain names based on statistics on DNS queries (≈100 billion requests/day) and client IPs (≈65 million unique users) across 165 countries [20].

4.2 Methodology

To study the use of SRI, we parse the HTML content of the webpages and identify the subresources included in webpages. Parsing the content of all the webpages contained in a snapshot of *CC* is time consuming. As our analysis focuses mainly on the use of SRI, we rely on a simple filter to detect webpages that include subresources with an integrity attribute. More specifically, we keep only the webpages that contain the string "integrity=". Note that this filtering is done on the *static* (i.e., returned by the server, without any client-side manipulation such as JavaScript execution) *raw* (i.e., in the binary format, that is before decoding) content of the webpages; this can lead to false negatives (i.e., filtering out webpages that do include subresources with an integrity attribute). We further detect the encoding of the webpages and parse them by using the Python `beautifulsoup` library (v4.7.1) with the `lxml` parser (v4.3.3). This enables us to extract the subresources of the webpages (and their attributes) and to filter out the webpages that do *not* include any subresource with an integrity attribute. Indeed, some webpages might not have been filtered out because they do contain the string "integrity=", but somewhere else in the webpage, e.g., in the text of the webpage on SRI on the W3C website. This constitutes the set of webpages on which we conduct our analysis, specifically the "webpages that contain at least one SRI" set (*CC-SRI*). For each webpage in the *CC-SRI* dataset, we extract its URL, content security policy (CSP) (from the HTTP header) and all its `link` and `script` elements/subresources. In order to study not only the current use of SRI but also its evolution over time, we process a total of 7 *CC* snapshots from 2016-05 (i.e., before the release of the SRI specification in late June 2016) to 2019-09 (more specifically 2016-05, 2017-02, 2017-08, 2018-02, 2018-08, 2019-03, 2019-09). In order to study the use of subresources on the web in general (not only for pages that do use SRI), we also extract a sample (*CC-all-1%*) that contains 1% of the latest snapshot (2019-09) of the *CC* dataset. For reproducibility purposes, the source code of our analysis scripts is available online at https://github.com/isplab-unil/cc-sri.

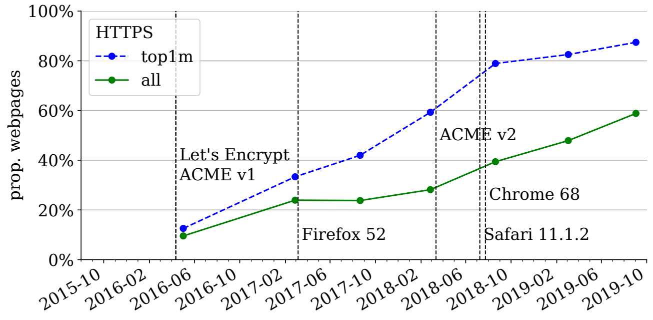


Figure 2: Evolution of the proportion of webpages served over HTTPS (in *CC*).

4.3 Results

We now present the results of our analysis of the use of SRI, based on the *CC* dataset.

4.3.1 HTTPS Adoption. The TLS protocol used in HTTPS provides, in addition to authentication, channel integrity. As such, it is related and sometimes complementary to SRI that provides channel and storage integrity (i.e., it also protects against adversaries who tamper with the content stored on the external server). Therefore, we start by measuring the adoption of TLS/HTTPS, in the *CC* dataset. We identify the protocol used (i.e., HTTP vs. HTTPS) based on the URL of the webpage.

Figure 2 depicts the evolution of the proportion of webpages served over HTTPS for the *CC* dataset and its subset (*Top1m*). It also features the important milestones in the development and deployment of HTTPS, including the release of `Let's encrypt` and of the `ACME` protocols v1 and v2 (which respectively automate the generation/distribution of certificates and the deployment of PKIs) as well as the introduction of security warnings for non-HTTPS webpages in major web browsers.

In the latest *CC* snapshot (2019-09), 58.82% of the webpages are served over HTTPS. The use of HTTPS is substantially higher among the most popular webpages (87.43% in *Top1m*).

4.3.2 Extent of the Threat. We measure the extent of the most typical threat to subresource integrity: the case where a webpage includes an external subresource. To do so, we compute the proportion of webpages that include at least one external subresource, in the *CC-all-1%* dataset. A subresource is called external if the host that serves it (in terms of its fully qualified domain name, e.g., www.otherserver.com) is *different* than the host that serves the webpage. Note that this is a heuristic: A same hostname can point to different servers (e.g., reverse proxy) and different hostnames

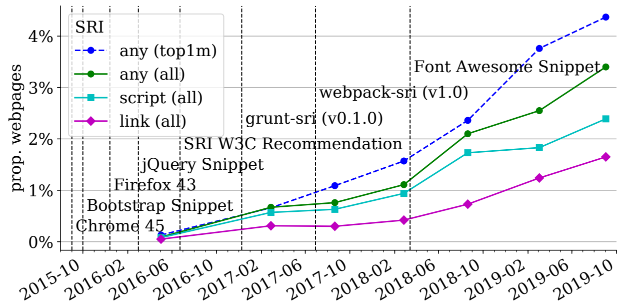


Figure 3: Evolution of the proportion of webpages containing subresource(s) w/ an integrity attribute (in *CC*).

Rank	Prop.	Name	Resources	Type	Snippet	SRI	Adoption
1	11.72%	Google Syndication	JS	Add-on	✓		
2	6.39%	jQuery	JS/CSS	Library	✓	✓	Mar.16
3	5.04%	Wordpress	JS/CSS	Platform			
4	2.62%	Google APIs	JS	Add-on	✓		
5	2.40%	Blogger	JS/CSS	Platform			
6	2.21%	FontAwesome	CSS	Library	✓	✓	Mar. 18
7	1.41%	TripAdvisor CDN	JS/CSS	Platform			
8	1.38%	Twitter	JS/CSS	Add-on	✓		
9	1.29%	SmugMug	JS/CSS	Platform			
10	1.21%	Squarespace	JS/CSS	Platform			
11	1.21%	Bootstrap	JS/CSS	Library	✓	✓	Oct.15
12	1.19%	WIX	JS/CSS	Platform			
13	1.13%	Google Ad Services	JS	Add-on	✓		
14	0.94%	Google Tag Services	JS	Add-on	✓		
15	0.94%	jQueryUI	JS/CSS	Library	✓	✓	Mar.16

Table 1: Most popular subresources (in *CC-all-1%*).

can point to the same server. We found that 82.76% of webpages include at least one link or script external subresource[1] and that these webpages include on average 8.24 ± 14.71 such subresources. All these webpages are potentially exposed to threats to subresource integrity and can benefit from SRI (some do, as explained below). For images (i.e., `img` elements), which are common subresources but not covered by SRI, the proportion of webpages including an external image is 54.37%.

4.3.3 SRI Adoption. We measure the adoption of SRI by counting the proportion of webpages that include *at least* one subresource with an integrity attribute (i.e., the size of the *CC-SRI* dataset). We further distinguish between the types of subresources: link and script, which are the only covered by SRI. Figure 3 depicts the evolution over time of the proportion of webpages containing at least one element (link, script or any) with an integrity attribute. It also features some important milestones in the development and adoption of SRI; in particular, we include the dates at which some popular libraries (e.g., Bootstrap) began to include integrity attributes in the code snippets provided on their webpages, typically in the "Quick Start" section. Note that CDNs hosting popular libraries (e.g., jsDelivr) also include such snippets.

It can be observed that some websites began to use SRI before the recommendation was officially released. This can be explained by the fact that the draft of the recommendation was available before its release and, more importantly, some libraries (e.g., Bootstrap) included integrity attributes in their code snippets as early as in late 2015 (see Table 1). Additionally, SRI was implemented in some web browsers before its release, as early as 2015-09 for Chrome (v45) and 2015-12 for Firefox (v43), because developers from both Google and Mozilla were involved in the edition of the SRI recommendation.

We find that the overall adoption of SRI is modest, with only 3.40% of all webpages in *CC*, but it grows at an increasing rate (the increase in 2018 is twice as large as in 2017). The adoption of SRI is highly influenced by the inclusion of the integrity attribute in code snippets provided by library developers on their websites. Another factor that could accelerate the adoption of SRI is the automatic inclusion of integrity attributes by build tools; we discuss this in Section 7. Although this cannot be directly concluded from Figure 3, it becomes clear when analyzing the targets of the subresources with an integrity attribute (as explained below). As hosting subresources on third-party servers comes with risks that major websites are probably reluctant to take, the adoption of the SRI recommendation by the *Top1m* websites is faster than for the rest of the Web. As mentioned above, the adoption of SRI is influenced by library developers and CDN operators (this is confirmed by the results of our survey of web developers, as a large fraction of developers report including integrity attributes by copy-pasting snippets; see Section 5). In order to better understand this point, we analyze (1) the main subresources (i.e., libraries, add-ons) used on the web and whether the corresponding websites promote SRI and (2) the main domains hosting subresources for which SRI is used.

Subresources. We compute the list of the most popular subresources in the *CC-all-1%* dataset and check whether they include code snippets and – if yes – whether these snippets use SRI (i.e., include an integrity attribute). For snippets that use SRI, we determine the date at which they began to do so by relying on the Wayback Machine, an online archive of the Web [21]. As the same subresources (e.g., JQuery) can be hosted on multiple domains with different URL formats, we devised a heuristic to identify them. The URL of the subresource is contained in the `target` attribute of the element. We grouped the subresources by domains and selected the top-100 domains (in terms of number of URLs); the top-100 covers 62.71% of the subresources present in *CC-all-1%*. For these domains, we manually identify patterns and build regular expressions to extract the names of the subresources from the URLs (e.g., "https://cdn.jsdelivr.net/npm/jquery@3.2.1/dist/jquery.min.js").

Table 1 lists the top-15 subresources (in terms of number of inclusions) found with our heuristic. It can be observed that a few subresources account for a substantial portion of the subresources in *CC-all-1%*, e.g., 11.72% for Google Syndication and 6.39% for jQuery. Note that some popular subresources, such as those of the Facebook ad network, do *not* appear in our results. This is because these subresources are loaded asynchronously using JavaScript. We observe that only a few subresources include snippets with SRI. In particular, add-ons do not use SRI; this is because the corresponding subresources are often transparently updated (i.e., without changing the URL: http://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js) by the providers, as noted by Lauinger et al. [26]. SRI is not well suited for such subresources.

Domains. We compute, in each snapshot of the *CC-SRI* dataset, the popularity of domains in terms of the number of subresources *with* an integrity attribute that are hosted on the considered domain (e.g., based on the URL in the `target` attribute). Figure 4

[1]We focus our analysis mainly on link and script elements as these are the only covered by the SRI recommendation at the moment. We discuss this in Sections 5 and 7.

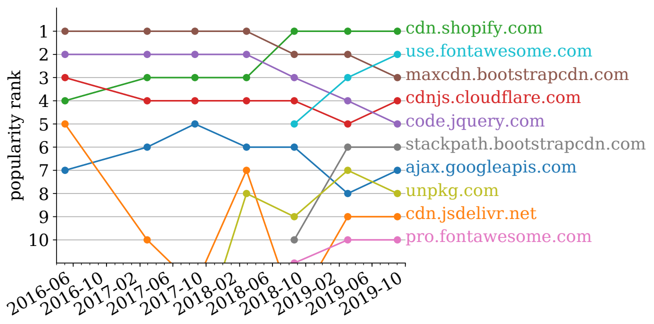


Figure 4: Evolution of the top-10 domains targeted by subresources with an integrity attribute (in *CC-SRI*).

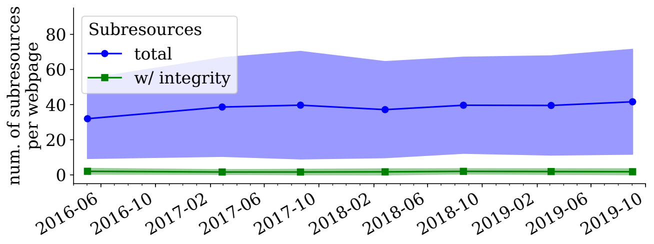


Figure 5: Mean and standard deviation of the number of subresources (all or with an integrity attribute) per webpage (in *CC-SRI*) that contains at least one subresource w/ an integrity attribute.

depicts the evolution over time of the top-10 domains found in the *CC-SRI* dataset. The top-10 domains cover 94.23% of all the subresources with an integrity attribute in *CC-SRI*. All the domains, except `Shopify`, provide snippets with SRI on their websites. Shopify does not because it is a platform, not a library included in other websites; but it uses SRI for its platform, which is very popular.

4.3.4 SRI Usage. We analyze the current practices of web developers when they use SRI.

Number of Subresources. Even though a webpage uses SRI for some of its subresources, it does not necessarily use SRI for all of them. We compute the number of resources with and without an integrity attribute in the snapshots of the *CC-SRI* dataset. Figure 5 depicts the number (mean and standard deviation) of subresources per webpage with the integrity attribute and the total number of subresources per webpage (w/ or w/o an integrity attribute). In the latest snapshot, webpages contain an average of 41.61 subresources and the number of subresources varies highly across webpages. The average number of subresources per webpage with the integrity attribute is much lower at 1.79.

Hash Algorithms. We compute the distribution of the hash algorithms (e.g., sha384) and of the number of hash expressions (i.e., digests) used in integrity attributes (for `link` and `scripts` elements, without distinctions), in the latest snapshot. To do so, we parse all the integrity attributes according to the format specified in the SRI recommendation. If the parsing fails, the attribute is considered malformed. In our analysis, we distinguish between malformed and empty attributes even though in practice, both are loaded by the browser (unless a CSP directive specifies otherwise). For the well-formed attributes, we extract the hash algorithm and label the attribute accordingly. When an integrity attribute contains multiple digests, e.g., sha256 and sha384, we label it with the different algorithms sorted by increasing strength (e.g., "sha256+384"): We use different labels for the different combinations of hash algorithms. Because a large proportion of web developers simply copy-paste snippets from websites (e.g., libraries and CDNs), the observations on the usage of SRI apply to a large extent to the developers of the included libraries and to the operators of the CDNs.

Figure 6 depicts the distribution of the hash algorithms across all the subresources with an integrity attribute. Most integrity attributes contain a single digest, and the most popular algorithms are sha384, sha256 and sha512 (in that order). Only 1.16% of the integrity attributes contain more than one digest with *different* hash algorithm. Note that, as browsers consider only the digests generated with the *strongest* hash algorithms and as the browsers that support SRI all support all the hash algorithms, such a practice does not make sense in practice. A possible explanation is that some web developers misunderstood how the case of multiple digests is handled by browsers (this is confirmed by the results of our survey; see Section 5): They might have (erroneously) thought that having more than one digest increases the security of the integrity verification. We observed even fewer (0.0004%; they are part of the "other" bar of the histogram) integrity attributes that contain more than one digest with the *same hash* algorithm (e.g., two sha256 digests). Such a practice enables developers to support multiple versions of a subresource by including the digest of each version (with the same hash algorithm); though convenient, this practice is very marginal.

For the malformed integrity attributes (1.41%), we manually investigate them; the causes include: missing hash algorithms, unsupported hash algorithms (i.e., "md5"), mistyped hash algorithms (e.g., "ha256"), and (possibly failed) injection through templates (e.g. "{{CHECKSUM}}"). This last example might be valid if the value of the attribute is correct and indeed inserted at the client before the verification is made by the browser. Yet, we manually tested these templates and none of them was properly rendered.

Finally, we looked at the distribution of hash algorithms for two popular libraries for which the snippets of code provided on their websites use different hash algorithms: sha-256 for JQuery and sha-384 for Bootstrap. In the *CC-SRI* dataset, 86.21% of the integrity attributes for the jQuery library (hosted on the jQuery CDN) use sha-256 and 98.14% of the integrity attributes for the Bootstrap library (hosted on the bootstrap CDN) use sha-384. This suggests that the snippets of code are often simply copy-pasted.

Protocols and Paths. Webpages that include subresources can be exposed to different adversaries and associated threats. More specifically there are four possible threats: Alteration of the webpage/subresource on the server/communication channel (see Section 2). Depending on the considered setting and adversary, SRI and TLS (i.e., HTTPS) can offer protection to the security of the webpage. We categorize the different settings using three criteria: whether the *webpage* is served using TLS (i.e., HTTP vs. HTTPS), whether the *subresource* is served using TLS, and whether the path to the subresource is local or external (i.e., "scripts/script.js" vs. "https://www.cdn.com/script.js"). Note that the protocol is sometimes omitted (e.g., "//server.com/script.js"), thus inherited: If TLS

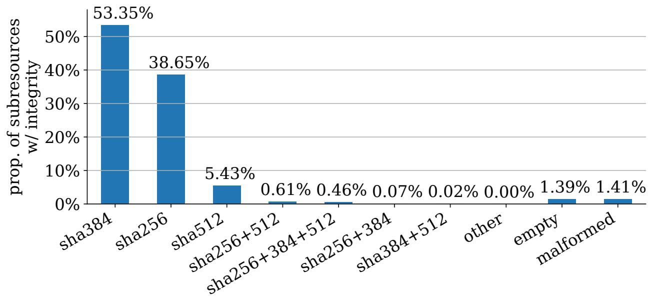


Figure 6: Distribution of the hash algorithms used in the integrity attributes (in latest snapshot of *CC-SRI*). An integrity attribute can contain more than one digest.

is used for serving the webpage, it is used for serving the subresource, otherwise, it is not used either for the subresource. We build the complete path of the subresources (by combining the URL of the webpage with that of the target of the subresource by using `urljoin` from Python's standard library) and analyze the breakdown between the different aforementioned settings, in the latest snapshot of *CC-SRI*. We also observed that protocol inheritance (i.e., //) is substantially used, especially in webpages served over TLS: 27.61% of the subresources served over TLS and included in webpages served over TLS are specified with protocol inheritance.

We observed that most of the subresources use absolute URLs (i.e., http://, https://, and //) for specifying the path in the target attribute, which are usually used for external subresources. Figure 7 depicts the breakdown in the form of a tree; the levels of the tree correspond to the following criteria: (1) webpage protocol, (2) subresource protocol, and (3) locality. The typical use case (i.e., HTTPS/HTTPS/external) is the most frequent. SRI is particularly meaningful when the webpage is served over TLS as the integrity attribute of its subresources is protected upon transmission. This represents 85.84% of the settings. In this setting, it makes even more sense when subresources are served without TLS, as SRI protects against corruption on the server *and* on the channel (i.e., 0.01% of the settings). Yet, such a practice (i.e., including a subresource served without TLS in a webpage served with TLS) is *not* allowed by browsers (i.e., *mixed content* error) – for valid security reasons – and the subresource will therefore *not* be loaded. Yet, this is marginal. When the webpage is served without TLS (14.16% of the settings), the integrity attributes of the subresources are not protected upon transmission and the security of the webpage is not guaranteed. Yet, assuming that the host and the channel for serving the webpage are not compromised, SRI provides protection against corruption on the server and/or channel serving the subresources.

Although SRI is meant primarily for securing the integrity of external subresources, its use for local subresources still makes sense (i.e., if the subresource is hosted on another server through a reverse proxy or if only some files on the server could be corrupted by the adversary) and should not be interpreted as erroneous or meaningless (Salvador et al. [37] discuss this point in detail). It could be the result of build tools that automatically compute and insert integrity attributes even for local subresources.

4.3.5 Use of the require-sri-for Directive. We compute, in the latest snapshot of *CC-SRI*, the proportion of webpages for which the

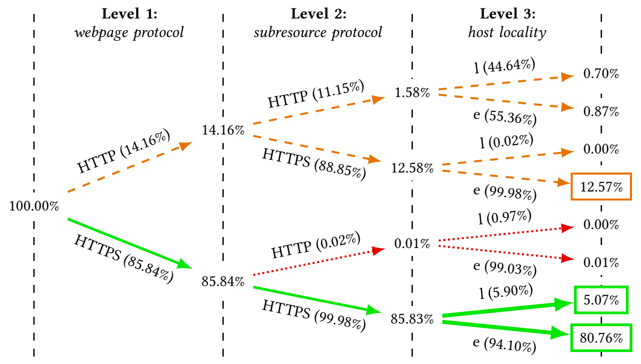


Figure 7: Categorization of subresources w/ an integrity attribute (in latest snapshot of *CC-SRI*) per webpage protocol (HTTP or HTTPS), subresource protocol (HTTP or HTTPS) and host locality (local, external). Frequent settings are framed. Secure settings are depicted with solid green lines; partially secure settings with dashed orange lines and problematic settings with red dotted lines.

`require-sri-for` CSP directive is specified in their HTTP headers: It is the case for only 0.02% of the webpages. We manually tested a small sample of webpages with this directive hosted on different domains: 8% of the subresources were blocked. It is marginal and its (scheduled) removal from the SRI recommendation will affect only a tiny fraction of the Web. Yet, we believe it should be maintained as it enables system administrators to enforce security policies.

5 WEB DEVELOPER EXPERIENCE

Given the results reported in the previous section, we decided to study the level of awareness and understanding of web developers regarding the SRI recommendation. Therefore, we posed the following research questions:

- **RQ1**. Are the web developers aware of the risks associated with (external) subresources?
- **RQ2**. Are web developers aware of SRI?
- **RQ3**. To what extent do web developers understand the implementation of SRI (general and specific behaviors)?
- **RQ4**. What are the current practices of developers when using SRI? (i.e., are they coherent with the recommendation?)

To answer these questions, we conducted an online survey of web developers. We adapted our methodology from similar surveys [1, 6]. The panelists were recruited via e-mail then went through quality controls. Panelists that completed the survey participated in a raffle for USD 100 Amazon vouchers. Next, we describe the method used to design the survey instrument, recruit participants, analyze the data as well as the deployment strategies we adopted to distribute the survey. The study was approved by our IRB.

5.1 Design of the Survey Instrument

The questionnaire contained 32 items, organized into four sections.[2] (i) The first section contained two screening questions to make sure

[2]The full questionnaire is available online: https://osf.io/yshrx/.

the respondent was comfortable reading and writing in English (the language of the questionnaire) and that the respondent was indeed an active web developer. (ii) The second section focused on the respondent's awareness of the threat model described in this paper (providing data for RQ1) and the SRI recommendation (cf. RQ2). This section had a skip logic: Respondents with no knowledge of SRI were brought to the fourth section. (iii) The third section contained 4 quiz questions designed to assess the respondent's understanding of the SRI recommendation (cf. RQ3) (note that all major browsers – i.e., Chrome/Firefox/Safari/Opera/Edge – strictly follow the recommendation for these quiz questions) and questions to understand how they used SRI in their work (cf. RQ4). (iv) The last section contained questions about the company the respondent works for and their demographic information.

On the last page of the questionnaire, we asked the respondents whether they wanted to receive a summary of the results of the research; 85.4% of respondents opted in to receive a follow-up, thus revealing the general interest in this topic. Furthermore, we clarified the goal of the research and provided a reference to the SRI recommendation, in case respondents were interested to learn more. It took about 10 minutes to complete the questionnaire.

To eliminate possible presentation effects, the answer options of multiple-choice questions were randomized. Before deploying the questionnaire, we conducted four pre-tests that involved individuals at our institution (including some native English speakers). One of the authors sat with the participants and, for each question, asked the participant to re-state, in their own words, what the question asked and how they would answer. Feedback provided at this stage was used to adjust wording and provide additional context.

5.2 Data Reliability and Coding Process

To ensure high data reliability, several quality-assurance (QA) processes were followed when administering the survey instrument: *speeders* and *straightliners* were removed before the analysis. One of the authors went through the open-ended responses and removed respondents that provided answers to open-ended questions that were nonsensical. For open-ended questions, we opted for collaborative coding on the qualitative responses [36]. For each question, a codebook was developed iteratively by a lead coder who analyzed an initial set of answers (i.e., ≈100). For the next step, a second coder independently coded the data again using the same codebook. Cohen's kappa (or κ) was used to measure inter-coder agreement to each open-ended question. The average κ value was 0.83 (std 0.18), which was judged sufficient to proceed further with the analysis. Next, the cases of disagreement between the two coders were resolved through discussion [28].

5.3 Deployment Strategies

To reach to the web development community, the questionnaire was disseminated to e-mail addresses obtained as follows:

(1) *Wordpress plugin/theme authors.* Wordpress is a web content management system (CMS) based on PHP, JavaScript, HTML and CSS. Any developer can create plugins/themes for Wordpress. The "readme" file of Wordpress plugins/themes often contains the e-mail of the authors. We considered a sample of $N{\approx}9500$ e-mails from Wordpress.
(2) *NPM package authors.* NPM is a repository for JavaScript packages. These packages were developed to be used in web applications or websites. Each package contains a *package.json* file that optionally provides the e-mail of the package developer. We considered a sample of $N{\approx}19000$ e-mails from NPM.

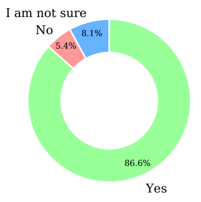
Selected web developers were sent an e-mail invitation to fill out the survey. The e-mail contained the following information: the academic research goal of the questionnaire (described as "understanding web development practices" in order to not prime the respondents towards security), the conditions for participation, the incentive, information about data management and anonymity of the responses, contact information of the researchers, and the link to the survey. As our e-mails were unsolicited, we gave them the opportunity to opt out, and we did not send any form of reminders. We did not collect any respondent personal identifiable information and did not link their responses to their e-mail. We sent a total of 28500 e-mails (in 2 batches) and received a total of 477 responses in Sep. 2019. After applying the QA processes described in Section 5.2 and removing incomplete answers, we were left with 227 valid responses. The results reported in the rest of the paper are based only on the valid responses.

5.4 Demographics and General Statistics

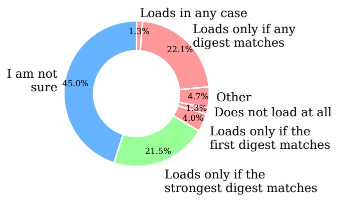
We received answers from professionals in various age ranges: 17.6% of respondents (or 40) were between 18 and 24 years old; the majority, or 42.3% (or 96) were between 25 and 34; 27.8% (or 63) between 35 and 44; and 11.9% (or 27) were older than 44 years. One respondent preferred not to disclose their age. Most respondents were employed full-time (i.e., 141 or 62.1%). 26% of respondents (or 59) were independent contractors, freelancers, or self-employed. The remaining respondents worked part-time (5.7% or 13), were unemployed (2.2% or 5), were retired (0.9% or 2) or had other work arrangements (3.1% or 7). Of the 213 respondents in the workforce, 46% worked for small companies, either startups or individually owned companies (or 98). A fourth worked for SMEs (24.4% or 52) and finally 29.6% (or 63) worked for large corporations. This shows that the sample was well balanced across different types of companies. About half of the sample declared to possess prior education on IT security (44.9% or 102). The remaining respondents did not have any prior education in IT security (48.5% or 110) and 6.6% (15) answered 'Other', which is worrisome given that we advertised among Wordpress and NPM plug-in developers, who produce popular software.

5.5 Results

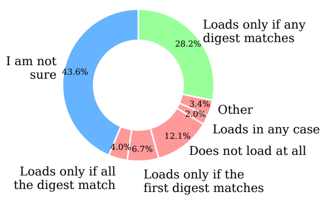
RQ1. Are the web developers aware of risks associated with (external) subresources? In the second section of the survey, we asked respondents to discuss potential threats that could affect a website if some subresources (e.g., scripts, stylesheets, images, videos) were hosted in a server separate from the server where the main website was hosted. In the rest of this section, we will refer to the described configuration as *the scenario.* The majority of respondents could identify the main threats of the scenario we provided, namely malicious code injection, cross-site scripting, etc. (56.8% or 129). A smaller group of participants (13.2% or 30) described secondary effects that could be produced if the main attacks could be completed:



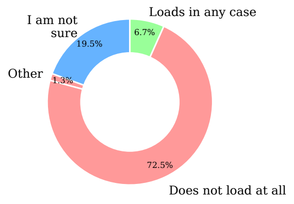
(a) Is it meaningful to use Subresource Integrity (SRI) if the target is served through HTTPS? (Q2)



(b) What happens when an integrity attribute has 2 or more valid hash values created with different algorithms? (Q6)



(c) What happens when an integrity attribute has 2 or more valid hash values generated with the same algorithm? (Q7)



(d) What happens if the digest in an integrity attribute is malformed (i.e., unsupported algorithm, non-base64)? (Q8)

Figure 8: Response statistics for the questions related to the functioning and implementation of SRI.

compromised users' privacy, key-logging, redirection to fake websites, DDoS attacks, etc. Finally, about a third of participants (30% or 68) provided generic answers, non-applicable responses, or simply had no idea about possible risks.

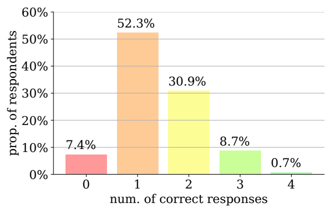
RQ2. Are web developers aware of SRI? In a follow-up question, we asked the respondents to list possible solutions to protect the website, in the considered scenario, against the threats they reported. The respondents could list multiple solutions. More than half of the respondents (50.7% or 115) provided answers that were not applicable, or described solutions that would introduce additional problems without providing a definitive solution to the threat (e.g., "*do not use CDNs*"). The other large portion of respondents (30.8%, or 70) provided the right answers, essentially naming SRI or using a descriptive explanation in case they were not familiar with the name of the recommendation (e.g., "*content validation with MD5 checksum*"). Finally, 18.5% (or 42) of the respondents described a technique or technology that would not ensure protection from the attacks (e.g., "*https*", "*two factor authentication*"). When we asked whether respondents had knowledge of the SRI recommendation (i.e., recognition over recall), we found that 41% (or 93) of the respondents had basic knowledge, and 24.6% (or 56) used SRI as part of their web development practices. About a third of the respondents (34.4% or 78) had no knowledge about SRI. Therefore, comparing the results to these two questions, we conclude that although two thirds of respondents declared to know or use SRI, only about 31% of the respondents could match the scenario with the solution provided with SRI. Hence, the difference (i.e., $\approx$ 30%) could be due to respondents who heard the acronym (or saw a snippet of code referring to SRI) but had no concrete idea of its purpose.

RQ3. To what extent do web developers understand the implementation of SRI? To those respondents who reported knowing or using SRI (149 or 65.6%), we asked them to describe in their own words how SRI could be used on a website and for what purpose. Most respondents described correctly the purpose and implementation of SRI (88.6% or 132). However, 11.4% (or 17) respondents could not. Next, we asked respondents 4 questions designed to assess the actual level of understanding of the recommendation. We manually investigated the "Other (please specify)" responses and edited them to the closest option whenever appropriate (otherwise we kept them as "Other" and considered them as incorrect). In total, we edited 3 responses, all for the second question and for the same reason (detailed below). The raw results are depicted in Figure 8.

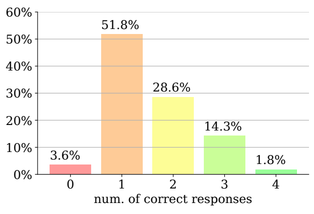
The first question looked at whether it was meaningful to use SRI in combination with HTTPS (for subresources). For this question, 86.5% (or 129) answered 'Yes' (i.e., the rest answered 'No' or 'Not Sure'). Then, we asked respondents what would happen if the integrity attribute would contain multiple (i.e., >1) valid hash values generated with *different* algorithms. Unfortunately, only 21.5% (or 32) identified the correct response, specifically that the browser would load the resource only if the digest created with the strongest hashing algorithm matches that of the resource. Three participants selected "Other" and specified that it is in fact the strongest algorithm *supported by the browser.* This is a valid point; but in practice, both responses are equivalent, as all browsers support all hashing algorithms. We edited them to the correct responses. The following question was similar to the previous, but this time the two hash values were created with the *same* algorithm. Slightly more respondents found the correct answer (28.2% or 42), namely that the browser would load the resource only if any digest in the list matches that of the resource. For the following question, we asked respondents what would happen if the digest in an integrity attribute would be malformed. To this question only 6.7% (or 10) of respondents found the correct answer. This shows that the recommendation is somewhat counterintuitive, assuming that the respondents did not know the recommendation on this particular point and tried to answer this question based on common sense. This choice was probably in line with the best-effort strategy implemented in browsers for rendering webpages. We believe that the default behavior should be *to not load* resources with malformed integrity attribute. It should be noted that for the last three questions, the proportion of respondents who were not sure is substantial; it is worrisome to observe that many web developers doubt about the behavior of browsers regarding some security features they use.

Figure 9 presents the number of correct responses identified by the respondents. Only 0.7% (or 1) of respondents correctly answered *all* the 4 questions, thus revealing the small proportion of web developers who have a deep understanding of SRI implementation. We observe that the respondents who reported using SRI (i.e., 24.6% or 56) have a slightly better understanding than the average.

RQ4. What are the current practices of developers when using SRI? We asked respondents to select from a list of options about how they typically include SRI in their development practices. We wrote the list of options based on a number of practices we identified while performing the literature review and speculated from the result of the CommonCrawl analysis. We provided respondents



(a) all respondents (N=149)



(b) respondents who use SRI (N=56)

Figure 9: Number of correct responses for each of the 4 questions testing the respondents' understanding of SRI.

the ability to select multiple practices from the list and to specify additional practices via a free text field. Most respondents reported to copy-paste snippets with examples of code using SRI from official documentations (45%, or 67). This strategy is not optimal because it might work for popular libraries but might not be available for all external subresources (e.g., specialized or custom libraries). Also, it is not automated (i.e., it requires human intervention) and it is thus not scalable and error prone (e.g., miss the last or first digits upon copy-paste as we observed in our *CC* analysis). The second most frequent option reported by the respondents was to configure the build tools to compute and to include the checksums automatically (38.9%, or 58). This is the most secure and scalable approach to implement the SRI recommendation. The third largest group of respondents reported to compute the checksums of the subresources themselves and include these manually in the code (19.5% or 29). This strategy is secure but, again, not automated hence not scalable and error prone. Every time the external subresources are updated, the developers must download the updated version of the files, compute the new digests and update the code on their website. A fourth group of respondents declared to copy-paste snippets from online communities (10.1%, or 15). This is the most dangerous strategy because developers basically trust other random contributors. A community developer could be malicious and publish code crafted to create harm. Finally, the remaining respondents were either not sure (9.4%, or 14) or had not used SRI yet (12.1%, or 18).

To the participants who had knowledge or used SRI, we asked whether they thought SRI should be extended to additional types of subresources (beyond stylesheets and scripts). Majority of developers (66.4% or 99) answered "Yes". In the follow-up questions, the respondents selected the following types from a pre-defined list: images (83.8% or 83), videos (79.8% or 79), sounds (73.7% or 73), and downloads (i.e., `<a>` elements pointing to a file not rendered in the browser)–as suggested by Cherubini et al. [10]–(67.7% or 67).

6 RELATED WORK

Our work relates to the literature on the use of web security mechanisms, including SRI and its alternatives, and on web developer security practices.

Analysis of the use of security mechanisms on the web. A large body of work focuses on the use of HTTPS on the Web and on the effect on users of browser warnings [2, 15–17, 22, 30, 34]. Felt et al. [18] study the adoption of HTTPS from a browser perspective. Lavrenovs and Melón [27] study the HTTP security headers of the 1M most popular websites. In particular, they analyze the prevalence of the most important response headers related to web security aspects, such as `Content-Security-Policy`. They notably show that HTTPS websites are more inclined to implement web security policies. Stark et al. [43] study the adoption of the certificate transparency (CT), which fixes several structural flaws in the TLS certificate system and measure the error rates users experience. They show that CT has been widely adopted with minimal amount of warning displayed to the users. Another body of work focuses on the issues related to the inclusion of third-party subresources and trackers in webpages [4, 7, 31, 32, 35, 41]. Arshad et al. [5] perform the first large-scale analysis of scriptless CSS injection. They show that around 9% of 10k most popular websites contain at least one vulnerable page, out of which more than one third can be exploited. Anis et al. [3] argue that many web applications contain vulnerabilities and promote various security mechanisms, including SRI. Van Acker et al. [45] assess the security of 50k+ login webpages and show that very few of them deploy security measures; e.g., only 98 use SRI.

Closer to our work, Shah and Patil [40] present existing attacks and describe how SRI improves the current situation. In a follow-up work, they perform a preliminary analysis of the use of SRI in the 1M most popular websites in 2017 [39]. Their analysis shows that only 7k websites (i.e., 0.7%) implement SRI, and that less than 1% of those enforce SRI on all external subresources. This is consistent with our results. Kumar et al. [25] also study security issues in the 1M most popular websites and find that less than 1% rely on SRI, which corroborates the findings of Shah and Patil [39] and ours. Lauinger et al. [26] conduct a large-scale study on client-side JavaScript library over 133k websites and show that 37% include at least one library with a known vulnerability. They mention SRI as one possible solution but stress the fact that SRI is misaligned with the objective of libraries to be transparently updated by third-party storage providers such as CDNs. Soni et al. [42] argue (prior to the publication of SRI) that SRI applies only to websites that remain mostly static and evaluate, on the 500 most popular websites, the proportion of those that rely on static or changing scripts. Based on their 3-month longitudinal study, they identify 33k scripts, of which about 2300 change over time, which implies that SRI could be applied to 93% of the scripts without affecting website usability. However, they also find that only 69 out of 500 websites have *all* their scripts that remain static. They develop a multi-layered solution for whitelisting scripts that can tolerate changes without sacrificing security. Recently, a scanner for monitoring and alerting on accidental/intentional modifications to external subresources has been developed and open-sourced by CISCO [11].

Unlike in prior studies, in this work, we consider a much *larger* set of webpages (≈3B URLs/snapshot), we study the *evolution* of the use of SRI over time (for more than three years), and we perform an *in-depth* analysis of how SRI is used. In addition, we study the main *factors* behind the development, adoption and usage of SRI.

Another, less related, body of works studies the limitations of integrity verification mechanisms (including SRI) and proposes solutions and alternatives. Salvador et al. [37] highlight the risks when the server that hosts the webpages is compromised. They address this issue by developing *wraudit*, a tool that transparently monitors the integrity of the published code based on a trusted and user-input baseline. Cap and Leiding [9] address this issue with openly accessible code reviews of static code files combined with blockchain technology. West [47] addresses the restriction of SRI to static content by enabling SRI to validate the integrity of subresources based a public key signatures instead of digests, thus enabling the inclusion of changing scripts published by the same (trusted) entity. Yet, this would open the door to downgrade attacks, i.e., replacing a recent version of a library with an old one with known and exploitable vulnerabilities. Kerschbaumer [23] proposes to enforce content security by default. In the background, he mentions SRI as one important specification for ensuring content security. Cherubini et al. [10] study the use of checksums for verifying the integrity of web downloads and, via a 40-participant in situ experiment, show that checksums suffer serious usability issues that negatively impact their effectiveness as a security mechanism. They develop a Chrome extension for automatically verifying checksums of downloaded files, whenever available.

Analysis of developer practices on (Web) security mechanisms. Although security practices of (web) developers have been extensively studied, to the best of our knowledge, there are no such studies related to SRI. Krombholz et al. [24] present a qualitative study of the mental models associated with HTTPS and highlight knowledge gaps that affect experts. By interviewing 18 end-users and 12 experienced administrators, they find that "non-experts" underestimate the protection offered by HTTPS and that even "experts" have very little knowledge of how HTTPS works. Acar et al. [1] systematically study how the use of various information sources affects the security of mobile applications. By surveying 295 developers with apps listed in the Google Play Store, they observe that most developers use search engines and Stack Overflow to find write security-related code. Their lab study with 54 Android developers shows that developers who use Stack Overflow are more likely to write functionally correct code, but less likely to come up with a secure solution. In order to better understand the context in which developers produce security-relevant code, Tahaei and Vaniea [44] survey 49 research papers at the intersection between usable security and software development. They provide an overview of existing works on developer-centered security and show that security is often being ignored because it is considered a secondary requirement. Balebako et al. [6] study the security-related decision of app developers with a two-phase approach. First, they conduct interviews with 13 developers to better understand what decisions they make and what resource they use to make them. Second, they perform an online survey with 228 developers. They find that many developers lack awareness about security measures.

7 DISCUSSION AND CONCLUSION

In this article, we have provided the first comprehensive study on the use of the SRI recommendation. Our study, based on a longitudinal analysis of the CommonCrawl datasets over the last 3.5 years sheds light on the current adoption and usage of SRI: The adoption rate is modest (currently at ≈3.40%) but growing, and it is influenced by library developers and CDN providers who make code snippets that include integrity attributes available to developers. As pointed out in prior work [42], the fact that SRI is suited only for subresources that do not change might impede its adoption. Our complementary survey of web developers has shown a good awareness and knowledge of SRI among developers but also some worrisome misunderstandings regarding its functioning in some situations. It has also revealed that the use of SRI by developers is mostly manual hence is not scalable and error prone, thus calling for a better integration of SRI by build tools.

Our study has some limitations, mainly due to our data sources. CommonCrawl might not be representative of the entire web. Also, it gives substantially more weight to multiple-page websites (counted as distinct URLs) compared to single page websites (counted as a single URL) [29]. Finally, the webpages consist of raw HTML code (i.e., not rendered), before the execution of scripts at the client side. Our questionnaire data is of modest size (i.e., 227 valid answers in total, 149 for the quiz) and the respondents might not be representative of web developers in general (i.e., "power developers", English-speaking, from specific ecosystems–i.e., NPM and WordPress). In addition, our results are based on self-reported behaviors, which might differ from actual (observed) behaviors.

The recent advances in the web development community are quite encouraging: More and more major libraries and CDN providers provide snippets with SRI (Bootstrap, jsdelivr). And major build tools, including Ruby on Rails [33], Webpack [38], and Grunt [19], now integrate SRI, mostly through plug-ins. The *native* integration of SRI in build tools, but also in CMS such as WordPress and Drupal, would substantially increase the adoption of SRI. Note that plug-ins are available for WordPress and the integration of SRI in Drupal (core) is being discussed. Finally, the extension of SRI to signature-based verification (i.e., with a public key as integrity attribute) [47] instead of digest-based verification, discussed in the WebAppSec group, would make SRI more suitable for subresources that are transparently updated, thus increasing its adoption.

We intend to improve the awareness and understanding of SRI and to promote its use through a dedicated website that would contain a brief description and illustration of SRI, including its functioning in specific scenarios that are not well understood by developers, as well as the statistics provided in this article, updated periodically based on the CommonCrawl datasets. We intend to investigate the adoption of SRI in different categories of websites (e.g., popular, banking, e-commerce). We also intend to push (or participate in) the revision of the recommandation to extend it to other resources such as images and videos, because this would thwart the risks of media-based web defacement (a majority of developers reported being interested in such an extension). Extending SRI to downloads would also favorably replace checksums displayed in webpages, as they suffer from serious usability issues. Finally, we also intend to extend our survey to a larger population of web developers, but also to further study developers' perception of SRI – through interviews – in order to gain a deeper understanding of their mental models on SRI, as recently done by Krombholz et al. [24] for HTTPS or by Acar et al. [1] for the security of mobile apps.

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers for their insightful feedback, to Holly Cogliati for her great editing job, and to Romain Artru for his help in the implementation of the preliminary analysis scripts. The work was partially funded with grant #19024 from the Hasler Foundation and with a grant from HEC Lausanne.

REFERENCES

[1] 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 *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, San Jose, CA, USA, 289–305. https://doi.org/10.1109/SP.2016.25

[2] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness.. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, Washington D.C., USA, 257–272.

[3] A. Anis, M. Zulkernine, S. Iqbal, C. Liem, and C. Chambers. 2018. Securing Web Applications with Secure Coding Practices and Integrity Verification. In *Proc. of the IEEE Int'l Conf. on Dependable, Autonomic and Secure Computing, Int'l Conf on Pervasive Intelligence and Computing, 4th Int'l Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. IEEE, Athens, Greece, 618–625. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00112

[4] Sajjad Arshad, Amin Kharraz, and William Robertson. 2016. Include Me Out: In-Browser Detection of Malicious Third-Party Content Inclusions. In *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*, Jens Grossklags and Bart Preneel (Eds.). Springer, Christ Church, Barbados, 441–459. https://doi.org/10.1007/978-3-662-54970-4_26

[5] Sajjad Arshad, Seyed Ali Mirheidari, Tobias Lauinger, Bruno Crispo, Engin Kirda, and William Robertson. 2018. Large-Scale Analysis of Style Injection by Relative Path Overwrite. In *Proc. of the Int'l Conf. on World Wide Web (WWW)*. ACM, Lyon, France, 237–246. https://doi.org/10.1145/3178876.3186090

[6] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. In *Proc. of the Workshop on Usable Security (USEC)*. Internet Society, San Diego, CA. https://doi.org/10.14722/usec.2014.23006

[7] Nataliia Bielova. 2013. Survey on JavaScript Security Policies and Their Enforcement Mechanisms in a Web Browser. *The Journal of Logic and Algebraic Programming* 82, 8 (2013), 243–262. https://doi.org/10.1016/j.jlap.2013.05.001

[8] Frederik Braun. 2019. Revert 'require-Sri-For' from the SRI Recommendation. https://github.com/w3c/webappsec-subresource-integrity/pull/82, last visited: Oct. 2019.

[9] Clemens H. Cap and Benjamin Leiding. 2018. Ensuring Resource Trust and Integrity in Web Browsers Using Blockchain Technology. In *Advanced Information Systems Engineering Workshops (Lecture Notes in Business Information Processing)*, Raimundas Matulevičius and Remco Dijkman (Eds.). Springer, 115–125.

[10] Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2018. Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Toronto, ON, Canada, 1256–1271. https://doi.org/10.1145/3243734.3243746

[11] Cisco CSIRT. 2019. SRI-Monitor: Subresource Integrity Monitor. https://github.com/ciscocsirt/SRI-Monitor, last visited: Oct. 2019.

[12] Common Crawl. 2019. Common Crawl. https://commoncrawl.org/.

[13] Common Crawl. 2019. Statistics of Common Crawl Monthly Archives. https://commoncrawl.github.io/cc-crawl-statistics/, last visited: Oct. 2019.

[14] Terence Eden. 2018. Major Sites Running Unauthenticated JavaScript on Their Payment Pages. https://shkspr.mobi/blog/2018/11/major-sites-running-unauthenticated-javascript-on-their-payment-pages/, last visited: Oct. 2019.

[15] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. 2008. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Florence, Italy, 1065–1074. https://doi.org/10.1145/1357054.1357219

[16] Serge Egelman and Stuart Schechter. 2013. The Importance of Being Earnest [in Security Warnings]. In *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*. Springer, Okinawa, Japan, 52–59. https://doi.org/10.1007/978-3-642-39884-1_5

[17] Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL Warnings: Comprehension and Adherence. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Seoul, Republic of Korea, 2893–2902. https://doi.org/10.1145/2702123.2702442

[18] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, Vancouver, BC, Canada, 16.

[19] Neftaly Hernandez. 2019. Grunt-Sri: SRI Plug-in for Grunt. https://github.com/neftaly/grunt-sri, last visited: Oct. 2019.

[20] Dan Hubbard. 2019. Cisco Umbrella 1 Million. https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/, last visited: Oct 2019.

[21] Internet Archive. 2019. Wayback Machine. https://web.archive.org/.

[22] Jeffrey L. Jenkins, Bonnie Brinton Anderson, Anthony Vance, C. Brock Kirwan, and David Eargle. 2016. More Harm than Good? How Messages That Interrupt Can Make Us Vulnerable. *Information Systems Research* 27, 4 (2016), 880–896. https://doi.org/10.1287/isre.2016.0644

[23] Christoph Kerschbaumer. 2016. Enforcing Content Security by Default within Web Browsers. In *Proc. of the IEEE Conf. on Cybersecurity Development (SecDev)*. IEEE, Boston, MA, USA, 101–106. https://doi.org/10.1109/SecDev.2016.033

[24] K. Krombholz, K. Busse, K. Pfeffer, M. Smith, and E. von Zezschwitz. 2019. "If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, San Francisco, CA, USA, 1138–1155. https://doi.org/10.1109/SP.2019.00060

[25] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J. Alex Halderman, and Michael Bailey. 2017. Security Challenges in an Increasingly Tangled Web. In *Proc. of the Int'l Conf. on World Wide Web (WWW)*. ACM, Perth, Australia, 677–684. https://doi.org/10.1145/3038912.3052686

[26] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *Proc. of the Network and Distributed System Security Symp. (NDSS)*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2017.23414

[27] A. Lavrenovs and F. J. R. Melón. 2018. HTTP Security Headers Analysis of Top One Million Websites. In *Proc. of the Int'l Conf. on Cyber Conflict (CyCon)*. IEEE, Tallinn, Estonia, 345–370. https://doi.org/10.23919/CYCON.2018.8405025

[28] Kathleen M. MacQueen, Eleanor McLellan, Kelly Kay, and Bobby Milstein. 1998. Codebook Development for Team-Based Qualitative Analysis. *CAM Journal* 10, 2 (1998), 31–36. https://doi.org/10.1177/1525822X980100020301

[29] Michael Mikowski and Josh Powell. 2013. *Single Page Web Applications: JavaScript End-to-End* (1st ed.). Manning Publications Co., Greenwich, CT, USA.

[30] David Modic and Ross Anderson. 2014. Reading This May Harm Your Computer: The Psychology of Malware Warnings. *Computers in Human Behavior* 41 (2014), 71–79. https://doi.org/10.1016/j.chb.2014.09.014

[31] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. 2019. ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices. In *Proc. of the ACM Asia Conf. on Computer and Communications Security (Asia CCS)*. ACM, 391–402. https://doi.org/10.1145/3321705.3329841

[32] 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 *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Raleigh, NC, USA, 736–747. https://doi.org/10.1145/2382196.2382274

[33] Joshua Peek. 2019. Sprockets-Rails: SRI Plug-in for Rails. https://github.com/rails/sprockets-rails, last visited: Oct. 2019.

[34] Robert W. Reeder, Adrienne Porter Felt, Sunny Consolvo, Nathan Malkin, Christopher Thompson, and Serge Egelman. 2018. An Experience Sampling Study of User Reactions to Browser Warnings in the Field. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, Montréal, Canada, 512:1–512:13. https://doi.org/10.1145/3173574.3174086

[35] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. 2012. Detecting and Defending Against Third-Party Tracking on the Web. In *Proc. of the Symp. on Networked Systems Design and Implementation (NSDI)*. USENIX, San Jose, CA, USA, 14.

[36] Johnny Saldaña. 2015. *The Coding Manual for Qualitative Researchers*. Sage, Arizona State University, USA.

[37] David Salvador, Jordi Cucurull, and Pau Julià. 2018. Wraudit: A Tool to Transparently Monitor Web Resources' Integrity. In *Proc. of the Int'l Conf. on Mining Intelligence and Knowledge Exploration (MIKE)*, Adrian Groza and Rajendra Prasath (Eds.). Springer, Cluj-Napoca, Romania, 239–247. https://doi.org/10.1007/978-3-030-05918-7_21

[38] Julian Scheid. 2019. Webpack-Subresource-Integrity: SRI Plug-in for Webpack. https://github.com/waysact/webpack-subresource-integrity, last visited: Oct. 2019.

[39] Ronak Shah and Kailas Patil. 2018. A Measurement Study of the Subresource Integrity Mechanism on Real-World Applications. *International Journal of Security and Networks* 13, 2 (2018), 129. https://doi.org/10.1504/IJSN.2018.092474

[40] Ronak N Shah and Kailas R Patil. 2017. Securing Third-Party Web Resources Using Subresource Integrity Automation. *International Journal on Emerging Trends in Technology* 4, 2 (2017), 5.

[41] Dolière Francis Somé, Nataliia Bielova, and Tamara Rezk. 2017. Control What You Include! Server-Side Protection against Third Party Web Tracking. In *Proc. of the Int'l Symp. Engineering Secure Software and Systems (ESSoS)*, Vol. 10379. Springer, 115–132. https://doi.org/10.1007/978-3-319-62105-0_8

[42] Pratik Soni, Enrico Budianto, and Prateek Saxena. 2015. The SICILIAN Defense: Signature-Based Whitelisting of Web JavaScript. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, Denver, CO, USA, 1542–1557. https://doi.org/10.1145/2810103.2813710

[43] E. Stark, R. Sleevi, R. Muminovic, D. O'Brien, E. Messeri, A. Felt, B. McMillion, and P. Tabriz. 2019. Does Certificate Transparency Break the Web? Measuring Adoption and Error Rate. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, Los Alamitos, CA, USA, 463–478. https://doi.org/10.1109/SP.2019.00027

[44] Mohammad Tahaei and Kami Vaniea. 2019. A Survey on Developer-Centred Security. In *Proc. of the IEEE European Symp. on Security and Privacy Workshops (EuroS&PW)*. IEEE, Stockholm, Sweden, 129–138. https://doi.org/10.1109/EuroSPW.2019.00021

[45] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. 2017. Measuring Login Webpage Security. In *Proc. of the ACM Symp. on Applied Computing (SAC)*. ACM, Marrakech, Morocco, 1753–1760. https://doi.org/10.1145/3019612.3019798

[46] W3C. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/.

[47] Mike West. 2017. Signed Content Should Include Resource Names. https://github.com/mikewest/signature-based-sri/issues/5, last visited: Oct. 2019.