A Large-Scale Analysis of the use of Integrity Verification Mechanisms for Web Resources

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

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

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

ABSTRACT

The Subresource Integrity (SRI) recommendation from the W3C ensures that a Webpage can safely integrate subresources hosted on a third-party server. As Edge Computing and Content Delivery Networks (CDN) get massively adopted, did web developers flocked to the SRI recommendation in order to protect their users? We conduct a large scale analysis of the CommonCrawl dataset to better understand the adoption of SRI on the Web from its inception in 2016 until 2019. We highlight the efforts made by different actors to drive the adoption of this recommendation. We survey web developers to assess their understanding of the security implications of SRI. Overall, this study presents the most accurate status of the SRI landscape and give hints on what could be done to accelerate the adoption of this important recommendation.

CCS CONCEPTS

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

KEYWORDS

web securitys; subresource integrity; checksums

ACM Reference Format:
Bertil Chapuis, Mauro Cherubini, and Kévin Huguenin. 2020. A Large-Scale Analysis of the use of Integrity Verification Mechanisms for Web Resources. In *Proceedings of (WWW '20)*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Web is a set of interlinked resources identified by their URL. A significant portion of these resources consists in HTML webpages that include navigable links and embed subresources, such as scripts, stylesheets, images or videos. For most of its history, the HTML standard did not enforce the integrity of these subresources embed in web pages, i.e., a change in the subresource will impact the webpage that includes it. However, the introduction of the recommendation for subresource integrity (SRI) addressed the most pressing aspects of this issue by recommending the use of an integrity verification mechanism for scripts and stylesheets.

Today, due to the advent of Content Delivery Networks (CDN) and Edge Computing, an increasing number of subresources are hosted by third-party providers. The advantage provided by such platforms are obvious: subresources are hosted at a fraction of the usual cost; subresources are delivered reliably at a low latency. However, their usage also come at a price: few things prevent a third party service to get hacked or to behave in a malicious way by injecting scripts or defacing stylesheets.

[**?**]

1.1 Research Goals

Given these risks and as discussions take place to extend the SRI standarlarge-scaled to subresources other than scripts and stylesheets, the following questions arise:

(1) To which extent did web developers and third-party provider adopt the SRI recommendation?
(2) To which extent do web developers understand the threats covered by the SRI recommendation?
(3) What are the forces that drive the adoption of the SRI recommendation since its inception?
(4) What could be done to accelerate and promote the adoption of the SRI recommendation?

1.2 Contributions and Roadmap

In this paper, we first disambiguate the threat model associated with the SRI recommendation and highlight its complementarity with HTTPS. We show that, contrary to a persistent popular belief, SRI is needed even if subresources are served over HTTPS.

(1) **Large-Scale Analysis** By using big data processing, we present a large-scale analysis of the use of the SRI recommendation on the Web.
(2) **Browser Support** From the analysis made of the SRI recommendation on the web, we test browser support for SRI according to the corner cases observed on the web.
(3) **Web Developer Behavior** We survey web developers to asses their understanding of the SRI recommendation and their user experience. We verify if their behavior correlates with the observations made in the large-scale analysis.
(4) **Adoption Incentives** We discuss the driving forces behind the adoption of SRI and give hints on what could foster its larger adoption in the future.

2 WHAT IS SRI?

2.1 Overview or the Recommendation

Subresource Integrity (SRI) let browsers verify that the subresources of a webpage are fetched without manipulation. Whereas this mechanism can be used with any subresource, the intent of the recommendation is to assert the integrity of resources served by

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

third party services, such as CDNs. This security feature allows the web developer to include a cryptographic checksum in the HTML that the downloaded subresource must match in order to be included. As of June 2019, SRI is fully supported by Chrome, Firefox, Opera and Safari, and partially supported by Edge.

```
<script src="example.js"
        integrity="sha256-TWupyv..."
        crossorigin="anonymous" />
```

Content Security Policy (CSP) let web developers mitigate certain kind of attacks, such as Cross Site Scripting (XSS), by specifying directives that delimitate the trust given to third party services. In the context of SRI, CSP directives can be used to mandate the web developper to include an integrity attribute for certain subresources. For instance, this can be achieved by specifying the following meta element in the head of the web page:

```
<meta http-equiv="Content-Security-Policy"
      content="require-sri-for␣script;">
```

Or by specifying the following HTTP header in the configuration of the web server:

```
Content-Security-Policy: require-sri-for script;
```

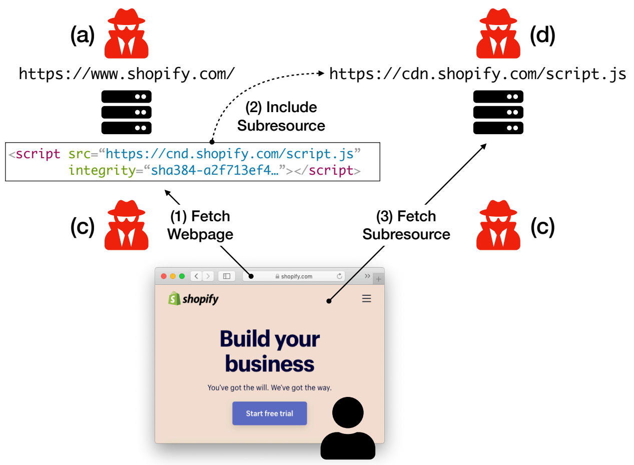
The latter mechanism has the advantage of enabling a separation of concerns between the web developer and the system administrator, who is usually more concerned by security issues. Managing SRI at the level of the configuration of the server enables the system administrator to force the adoption of the recommendation at a higher level. Unfortunately, due to its poor adoption, web browsers slowly abandon this directive.

Interestingly, given an integrity attribute, the SRI recommendation allows to specify multiple checksums (See SRI section 3.6), each of which is prefixed by the name of the hash function used to generate it followed by a dash (e.g., sha256-). This allows the browsers to pick the strongest hash algorithms, but also to specify several hash values for the same URL, which might be useful during a rolling upgrade.

2.2 Threat Model

The design goal of the SRI recommendation is: "to define a mechanism by which user agents may verify that a fetched resource has been delivered without unexpected manipulation".[1] The HTTPS specification share a common objective with SRI: "SSL, and its successor TLS were designed to provide channel-oriented security".[2] Despite the fact that the two standards seem to overlap, HTTPS provides channel integrity by authenticating the server, whereas SRI provides subresource integrity by authenticating the content.

Figure 1 illustrates a simplified threat model for HTTPS and SRI that helps understanding the threats covered by the specification and the recommendation. We consider an HTML webpage delivered by a server managed by a web developer. The webpage includes a script element that references a subresource (e.g. a JavaScript program or a CSS stylesheet) hosted on another server, typically managed by a third-party. In order to enable browsers to check the authenticity of the subresource, the webpage specifies an



```
<script src="https://cnd.shopify.com/script.js"
        integrity="sha384-a2f713ef4…"></script>
```

Figure 1: Simplified threat model for HTTPS and SRI

integrity attribute for the subresource. We consider four possible adversaries who could tamper with the webpage:

(a) The attacker gains access to the server managed by the web developer and tempers with the webpage. If this server gets compromised, neither HTTPS nor SRI help at protecting the users. Therefore, we assume that the server managed by the web developer is secure and trusted.
(b) The attacker tempers with the communication channel between the server hosted by the web developer and the users. A web page served over HTTP is vulnerable to such attack. Therefore, we assume that HTTPS is trustworthy in preventing their occurrences.
(c) The attacker tampers with the communication channel between the third-party server that hosts the subresource and the users. If the subresource is served over HTTP, SRI can prevent the execution of a compromised subresource by authenticating its content. However, HTTPS should be preferred as it provides authenticity at the level of the channel.
(d) The attacker gains access to the third-party server and tampers with the subresource. If the third-party server gets compromised, HTTPS is of no help, i.e., nothing prevent the attacker from modifying the subresource. SRI precisely addresses this issue: if the hash of the subresource does not match the integrity attribute, the browser prevents its inclusion and execution.

As highlighted here, the SRI recommendation covers a unique spot at the level of the third-party server that hosts subresources; a scenario which is far from being marginal in today's decentralized context. For instance, a CDN typically rely on a network of geographically distributed servers, each of which can get individually compromised, affecting a subset of the users. In fact, in the context of a CND, having the same domain for all resources does not mean that they are located on the same serve, hence the importance of content authenticity. Therefore, as web developers increasingly rely on CDNs to deliver their content, a mechanism that

[1] https://www.w3.org/TR/SRI/
[2] https://tools.ietf.org/html/rfc2818

enables them to guarantee the integrity of subresources is necessary to mitigate this kind of attacks and the widespread adoption of SRI becomes critical.

2.3 Developers' Behavior

When it comes to the adoption of SRI by web developers, the development practices they adopt are of great importance. In this Section, we describe the most popular web application architectures, we highlight the build tools associated with them, and give hints on how these tools can influence the behavior adopted by web developers.

2.3.1 Multi Page Application (MPA). Traditionally, each action performed by the user on an HTML webpage (clicking on a link, submitting a form) generates an HTTP request. A web server typically respond to requests with new HTML webpages, which are displayed in the browser. That is the multi page application (MPA) architecture. Web developers usually modify the server side templates to include JavaScript and CSS libraries (such jQuery or Bootstrap) in the head of the HTML.

2.3.2 Single Page Application (SPA). More recently, the single page application (SPA) paradigm started to gain in popularity with frameworks such as Angular, ReactJS, or VueJS. These applications load a single HTML webpage and, later on, dynamically update their content. To do so, these application bypass the traditional way of loading webpages by URLs. The content displayed in the application is transmitted with a micro-format, such as JSON, and its rendering occurs in the browser instead of on the server. The root page of an SPA is often managed with a build tool that includes the necessary dependencies in the head of the HTML. Web developers usually modify a configuration file, which is then used to inject development or production versions of the dependencies.

3 LARGE-SCALE ANALYSIS

In this section, we report on the analysis of the use of SRI on the Web. We describe our datasets and our methodology and we present our results.

3.1 Data sources

3.1.1 Common Crawl (CC). The Common Crawl (*CC*) dataset is a collection of snapshots of the Web.[3] It contains one snapshot per month since 2011-01. Each snapshot comes in three flavors: the WARC format which contains the raw data of the webpages (HTTP headers and HTML content but not the content of the subresources), the WAT format which contains the metadata of the webpages (HTTP headers and the hyperlinks contained in the webpage) and the WET format which contains the plaintext extracted from the webpages. The latest snapshot of Common Crawl (2019-08) weighs 50 TB and contains 3,000,000,000 URLs.[4] Each snapshot (in each of the format) is divided into multiple archives so as 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 using frameworks such as Hadoop and Spark. In this paper, we rely on the *CC* dataset to analyze the prevalence of SRI on the Web and its evolution over time.

3.1.2 Cisco Umbrella 1 Million (Cisco). The Cisco Umbrella (*Cisco*) dataset is a ranking of the top 1 million most popular domains names.[5] The algorithm used to generate this dataset uses DNS queries (100 Billion requests/days) and client IPs (65 million unique users) to measure the aggregated popularity of a domain name. As the Cisco Umbrella network is global (165 countries), it is representative of the overall traffic on the Web. Despite the fact that this dataset is not confined to HTTP traffic, we prefer it to the popular Alexa 1 Million dataset, which is not available in open access anymore.[6] We sometimes use the top 1 thousand (*Top1k*) and top 1 million (*Top1m*) domains extracted from the *Cisco* dataset to bound our metrics to the most visited part of the Internet.

3.2 Methodology

In order to study the use of SRI on the Web, we need to extract the subresources contained in webpages and therefore to parse their HTML content. Parsing the content all the webpages contained in a snapshot of *CC* is very time consuming, and hence expensive. As our analysis focuses on the use of SRI, we rely on a simple filter to detect webpages containing 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 contain subresources with an integrity attribute). We further detect the encoding of the webpages and parse them by using the beautifulsoup library (vXX) with the lxml parser. This enables us to extract the subresources of the webpages (and their attributes) and to filter out the webpages that do *not* contain any subresource with an integrity attribute (that have not been filtered out because they do contain the string "ìntegrity=" but somewhere else in the webpage). This constitutes the set of webpages on which we conduct our analysis, namely 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 header), and all its link and script elements/subresources.

In order to study not only the current use of SRI but also its evolution, in our analysis, we process a total of 7 *CC* snapshots from 2016-06 to 2019-08 (more specifically 2016-06, 2017-03, 2017-09, 2018-03, 2018-09, 2019-03, and 2019-08). Note that the SRI specification was issued by the W3C in late June 2016.

3.3 Results

3.3.1 HTTPS Adoption. As the HTTPS protocol provides channel integrity, we measured its adoption in the *CC* dataset. The *CC* dataset stores the complete URLs of the webpages and we parsed them to extract the protocol in use. In addition, we filtered the *CC* dataset with the *Top1k* and *Top1m* datasets understand how the most visited part of the Web are evolving in terms of protocol.

[3] http://commoncrawl.org/

[4] More statistics are available at: https://commoncrawl.github.io/cc-crawl-statistics, last visited: Sep. 2019.

[5] https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/

[6] https://www.alexa.com/topsites

Figure 2 displays the evolution of the percentage of webpages served over HTTPS for the *CC* dataset and its subsets (*Top1m*, *Top1k*). As of writing, almost half of the websites (47.90%) and most of the websites in *Top1k*and *Top1m*switched from HTTP to HTTPS and the adoption seems to be reaching a plateau. Interestingly, the adoption of HTTPS measured for the *Top1k* domains in 2016 is close from the adoption measured in Chrome at the same period by Felt et al **??**, which highlights the incredible progress made by popular websites in adopting HTTPS. Massive and fruitful efforts have been leveraged to accelerate the adoption of HTTPS. The release of let's encrypt (2016-04-12) and of the ACME protocol v1 (2016-04-12) and v2 (2018-03-13), which automates the deployment of public key infrastructure, played a major role in the adoption for HTTPS. The introduction of security warnings in Firefox (2017-03-07), Chrome (2018-07-24) and Safari (2018-07-09) for webpages served over HTTP have also been an effective incentive, in the sense that end-users complains about security warning affects web developers directly. Internet Explorer and Microsoft Edge did not yet introduced security warnings for HTTP webpages. However, the adoption of the Chromium platform by Microsoft Edge suggests that very soon all major browsers will visually encourage the adoption of HTTPS.

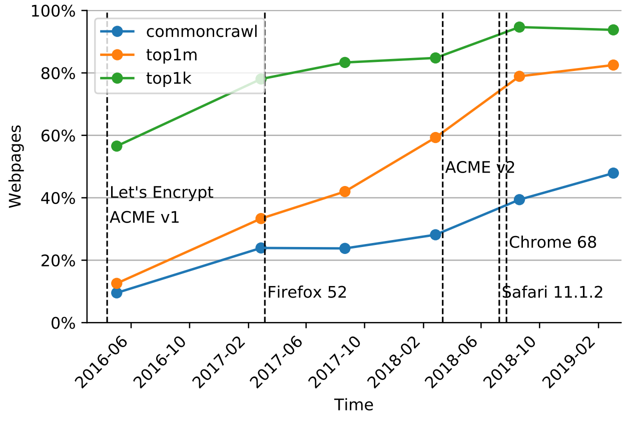


Figure 2: Evolution of the percentage of webpages served over HTTPS

3.3.2 SRI Adoption. The release of the SRI recommendation in June 2016 officially enabled its wide spread adoption by web developers. To measure the extent of this adoption, we study the use of the integrity attribute across all the versions of the *CC-SRI* dataset. To do so, we count the webpages that include at least one subresource that specifies the integrity attribute for *script* element, the *link* elements and the combination of the two (*any tag*). Additionally, we filter the *CC* dataset with the *Top1k* and *Top1m* datasets to measure the adoption of the integrity attribute in the most visited parts of the Web.

Figure 3 depicts the evolution of the percentage of webpages containing at least one element that specifies the integrity attribute over time. overall, the adoption of SRI is slowing down; this slow down particularly affects the *script* element and spare the *link* element. This might be explained by the increasing popularity of the SPA architecture to the detriment of the MPA architecture. Whereas the aging MPA architecture promoted the inclusion of libraries, such as jQuery, by copy pasting snippets in the head of the webpage, the SPA architecture often rely on build tools that inject scripts in the webpage but lack support for SRI. In this context, the use of the integrity does not affect the *link* element, which might be due to the growing popularity of the *Font Awesome* library, which still seems to be mostly included in webpages by copy pasting a snippet. Interestingly, a minor fraction of the webpages started including the integrity attribute before the release of the recommendation in June 2016. Most notably, the popular bootstrap framework started to provide code snippets containing the integrity attribute since October 2015, which highlights the living nature of the HTML standard. Finally, it can be observed that the adoption of the SRI recommendation by the *Top1k*and *Top1m*websites started late but accelerates very fast. This might be due to the fact that popular websites operated their own servers until recently and started to rely on third-party CNDs once their advantages became obvious. Today, as popular websites increasingly rely on CDNs and third party services, justified security concerns probably drive this adoption.

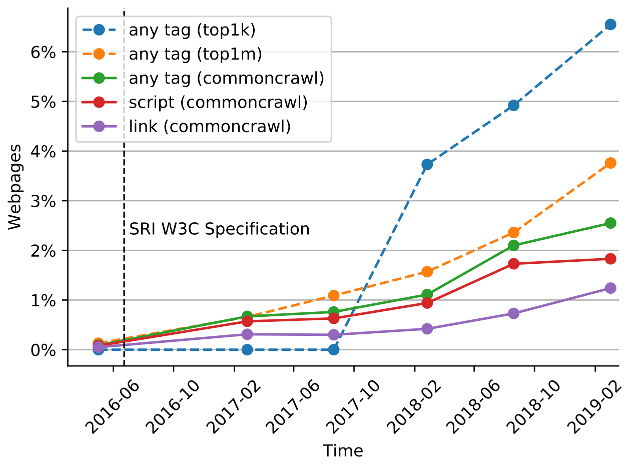


Figure 3: Evolution of the percentage of webpages containing one or more tag (script or link) that specifies the integrity attribute

3.3.3 Subresources per Webpage. A webpage can contains subresources that specify the integrity attribute along side subresources that do not specify it. We queried the *CC-SRI* dataset to obtain the mean number of subresources per webpages (with and without integrity attribute) and the associated standard deviation at different point in time.

Figure 4 displays the mean and standard deviations for subresources with and without the integrity attribute. The mean number of subresource with the integrity attribute per webpage is low and revolve around two. The large discrepancy between the mean number of subresources and the mean number of subresource without the integrity attribute suggests that web developers copy paste one or two code snippets that specify the integrity attribute in the head of their webpage. In addition, it seems to indicate that the automatically injection of SRI in webpages by built tools did not yet gained traction. In conclusion, the fact that the mean and standard deviation remained stable over time suggests that a shift in behavior (similar to what occurred for HTTPS) that would significantly improve the adoption of SRI did not occur yet.

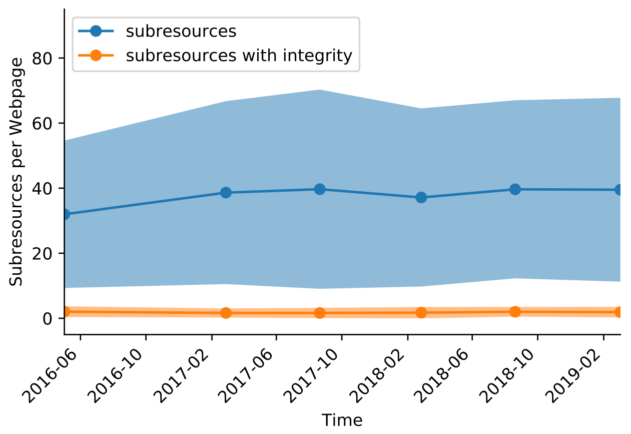


Figure 4: Mean and standard deviation of the number of subresources per webpage, among webpages that contain at least one SRI (2.55%**)**

3.3.4 *SRI per Webpage.*

The integrity attribute is supported by the *script* and *link* elements, which are used to load external subresources in webpages, such as script or stylesheets. We queried the 2019-08 snapshot of the *CC-SRI* dataset and counted the number of occurrence of the integrity attribute associated with these elements.

Figure 5 depicts the PDF/CDF for the integrity attribute in the 2019-08 snapshot of the *CC-SRI* dataset. It can be observed that, regardless of the element (*any tag*), 49.27%of webpages contain one integrity attribute and 33.79%of webpages contain two integrity attribute. The same observation holds when looking at the *script* element, i.e., web developers regularly include more than one element that specifies the integrity attribute. When it comes to *link* elements, the web developer's behaviour is slightly different, as 83.06% of them specify the integrity attribute only once. Overall, the fact that the number of elements that specify the integrity attribute is very low suggests a copy-paste behavior from Web developers, who simply integrate the snippets provided by popular javascript and css libraries, such as JQuery and Bootstrap.

3.3.5 *SRI Hash algorithms.*

SRI relies on the SHA2 hash algorithms for producing the digests included in integrity attributes; it allows for different digest sizes, essentially 256, 384 and 512 bits. The size of the digest used is specified at the beginning of the integrity attribute, followed by the base-64 encoded digest. SRI allows multiple (space-separated) digests in an integrity attribute; in this case, the subresource is loaded by the user agent if *at least* one of the digests matches that of the downloaded subresource or if the integrity attribute is malformed.

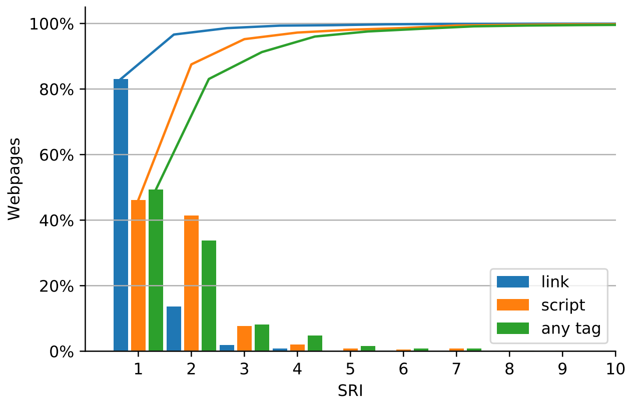


Figure 5: PDF/CDF of the number of SRI per webpages, among webpages that contain at least one SRI (2.55%**)**

More specifically, an SRI integrity attribute follows the CSP Level 2 rules below. For instance, "sha256-ZdAL...smUg= sha384-Cq5f...h0kN" is a valid integrity attribute with two digests of size 256 and 384 respectively:

```
base64-value  = 1*( ALPHA / DIGIT / "+" / "/" )*2( "=" )
hash-value    = base64-value
hash-algo     = "sha256" / "sha384" / "sha512"
hash-source   = hash-algo "-" hash-value
```

We study the sizes of the digests and the number of digests used in integrity attributes (for link and scripts elements) across the latest version of the *CC-SRI* dataset (2019-08). To do so, we analyze all the integrity attributes in the dataset; for each integrity attribute, we parse it according to the rules specified above. More specifically, we re-implemented the parsing and validation performed in Firefox v?. If the parsing or the validation fails, the attribute is considered malformed (and labeled as such). In our analysis, we distinguish malformed and empty attributes. Note that subresources with an empty integrity attribute are loaded by user agents. For the well-formed attribute, we extract the digest size(s) and label the attribute accordingly. When an integrity attribute contains multiple digests, e.g., SHA-256 and SHA-384, we label it with the different sizes (e.g., "SHA-256+SHA-384"): we use different labels for the different combinations of digest sizes.

Figure 6 depicts the distribution of the digest sizes across all the link and scripts elements with an integrity attribute in the dataset. It can be observed that most integrity attributes contain only one digest, and that the most popular lengths are SHA-384 and SHA-256 (in that order). Only a few percents of the integrity attributes contain more than one digest. We did not observe any integrity attribute with multiple digests of the same size (e.g., two SHA-256 digests), which is surprising. Indeed, using several digests enables webmasters to handle multiple version of a script by including the digests of all the supported versions (if the version of the script is not specified in the URL). In such a case, however, there is no particular reason to use different digest size. A possible explanation is that some webmasters misunderstood how the case of multiple digests is handled by user agents: They might have thought that having more than one digest increases the security of the integrity verification; it is actually the opposite as the subresource is loaded as soon as one of the digests matches that of the downloaded subresource. A very small (yet not negligible) proportion of the attributes are left empty. This is the case of the subresources on the Apple iTunes website. For the malformed integrity attributes, we manually investigate them; the causes include: missing hash algorithms (e.g., "Cq5f..."), unsupported hash algorithms (i.e., "md5"), mistyped hash algorithms (e.g., "ha256"), and injection through client-side template (e.g. "{{CHECKSUM}}"). Note that in this last example, the use case might be eventually be valid if the value of the attribute is (correct and) indeed inserted at the client side before the verification is made by the user agent.

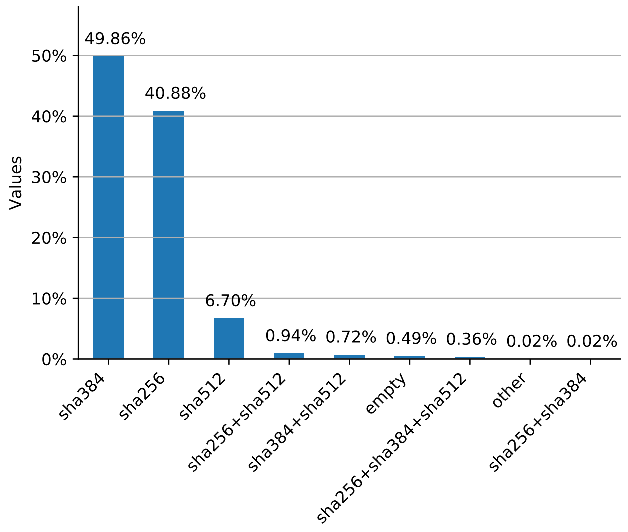


Figure 6: Distribution of the hash algorithms used to compute the checksums included in the value of the integrity attribute (the integrity attribute can specify more than one checksum).

3.3.6 *Distribution of SRI per host and target protocol.*

The protocol used by a webpage does not necessarily match the protocol of its subresources. For instance, a webpage served over can include an HTTPS subresource, whereas the reverse is not permitted by modern Web browsers for security reasons. In this context, the *host* protocol refers to the protocol used to serve the webpage that include the subresource. The *target* protocol refers to the unresolved protocol specified in the *src* or *href* attributes of the subresource included in the webpage. In practice, this *target* protocol can be specified explicitly (http://, https://), or implicitly, i.e., the protocol is inherited when the path is relative (.) or absolute (/) and when the URL use protocol inheritance (//).

Figure 7 depicts the distribution of subresources that specify the integrity attribute by *host* and *target* protocol. We observe that most of the *targets* fall into the absolute URL categories (http://, https:// and //). As the integrity attribute does not make a lot of sense in the context of relative URLs, we can assert that, in this regard, the SRI recommendation is well used and understood. Interestingly, the extensive use of protocol inheritance for *targets* (//), which allows to gracefully upgrade a website from HTTP to HTTPS, is syntactic sugar, as it is mostly used on webpages served over HTTPS.

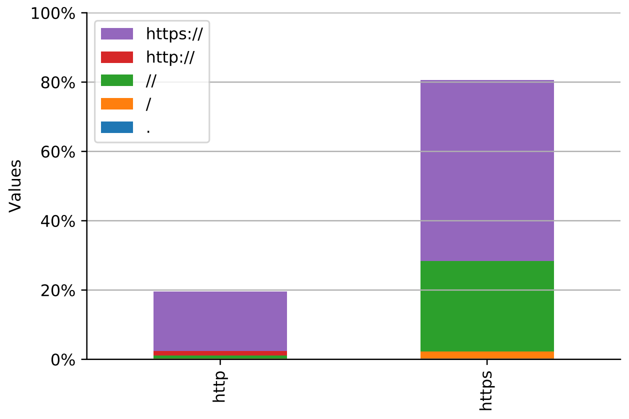


Figure 7: Distribution of SRI per host and target protocol

3.3.7 *Categorization of SRI per host protocol, target protocol and locality.*

The domain used by a webpage does not necessarily match the domain of its subresource. Given the fact that webpages served on the same domain are not necessarily served by the same server, the integrity attribute is of interest even if the domains are the same. However, when subresources are hosted on a CDN, it is common to observe a such a domain mismatch. Therefore, we queried the 2019-08 snapshot of the *CC-SRI* dataset to categorize the subresources that specify the integrity attribute per *host* protocol, *target* protocol and *locality* . Here, the *host* refers to the domain and protocol of the webpage, the *target* refers to the domain and resolved protocol of the subresource, and *locality* tells if the domain and protocol of the *host* is the same as the domain and protocol of the *target* (local or remote).

The tree depicted in Figure 8 breaks down our results according to the aforementioned criterion. The first level of the tree shows that the majority of websites using SRI (80.50%) already adopted HTTPS. Given the fact that 17.03% of the subresources having an HTTP *host* have an HTTPS *target*, the second level shows that overall the vast majority of subresources are already served over HTTPS (97.53%). The third level shows that 80.49% of the subresources specifying the integrity attribute are protected against all the attacks mentioned in the Threat model. However, 17.02% of the subresources are still exposed to tampering at the level of the communication channel (see (b) and (d) in Figure 1) The third level highlight that misuses of the HTTPS specification, such as downgrade of the protocol from HTTPS to HTTP, are very rare (0.01%). In case of downgrade, the user is not exposed to a security risk. However, his experience is degraded as the subresource is not included in the webpage by the browsers.

3.3.8 *Distribution of the top-10 domains among SRI targets.*

Some domains names capture most of the traffic generated by the inclusions of subresources with the integrity attribute in webpages. To identify these domains and better understand their reason for

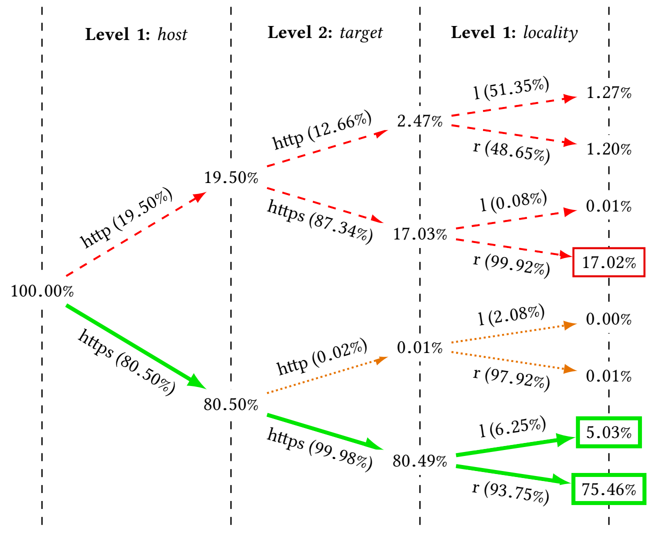


Figure 8: Categorization of SRI per host protocol (http or https), target protocol (http or https) and locality (local, remote)

being, we queried all the snapshots of the *CC-SRI* dataset and grouped subresources with the integrity attribute by their domain.

Figure 9 displays the evolution of the top 10 domains found in the *CC-SRI* dataset. The domains present in this top 10 cover 89.76% of the subresources that specify the integrity attribute. We identify three different use cases in this top 10:

- **Platform** providers, such as Shopify (1), use SRI to prevent subresource tampering on third-party servers. Shopify is a startup that provides a hosted proprietary e-commerce platform and guaranteeing the integrity of subresources on login and payment webpages is critical. As the platform is hosted and upgraded by Shopify, Web developers are not asked to specify the integrity attributes themselves. The fact that SRI is used by such platforms demonstrates that the coupling it introduce between webpages and their subresources is manageable at scale.
- **Library** maintainers usually provide releases hosted on third-party servers. In this context, Web developers using libraries are asked to copy paste a snippet that specifies the integrity attribute, hence guaranteeing the integrity of the subresource. For instance, Bootstrap (2 and 6) is a popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Up to version 3, the documentation of the framework suggested to include a snippet that points to a subresource hosted by MaxCDN (2). Since the release of the version 4, the snippet points to a resource hosted by StackPath (6). JQuery (4) is an open source, fast, small, and feature-rich JavaScript library. FontAwesome (3 and 10) is an open source collection vector icons and social logos for websites.
- **CDN** providers, such as cdnjs (5), unpkg (7), Google Hosted Libraries (8) and jsDelivr (9), address the need to deliver content globaly in a reliable way and at a low latency. Different reasons, such as gaining visibility among Web developers, can drive such providers at providing free access to third party libraries. As SRI introduces a strong coupling between the webpage and the subresource it integrates, it is tempting to relax the integrity constraints to provide more flexibility. For instance, as shown in Figure 10, jsDelivr provides different url schemes for subresources that allow to specify version ranges. Relaxing integrity is a questionable practice, which is only viable if the third party is assumed to be trustable and sustainable on the long run.

Overall, the presence of Shopify is striking because it does not reflect the integration of a code snippets by Web developers. In contrast, it shows the wide adoption of their platform by e-commerce owners, the concern they have when dealing with third-party providers, and the automated build pipeline they probably built to mitigate the effect of a strong coupling between webpages and subresources. The other results of this top 10 mostly relate to the inclusion of a code snippets by developers. Hence, cumulatively, the copy pating adopted by web developers when integrating javascript from library providers and CDNs constitute the driving force of the adoption of the SRI standard. In the future, the efforts made to stimulate web developers in adopting the SRI recommendation should incentivises them in adopting build tools that automatically inject the integrity attribute in webpages.

3.3.9 *Use of the require-sri-for directive.*

The *require-sri-for* directive leverage the ability to ensure that all subresources of a webpage specify the integrity attribute. As a result, it allows an elegant separation of concerns between system administrators and web developers. We queried the 2019-08 snapshot of the *CC-SRI* dataset to obtain the percentage of webpages that specify the *require-sri-for* directive in their HTTP headers. In practice, only a marginal number of webpages (0.01%) use this directive. Given this poor adoption, its upcoming deprecation by Web browsers is clearly justified.

4 BROWSER SUPPORT

4.1 Methodology

We created a testbed consisting in webpages to check the implementation of the SRI recommendation in major browsers. We visited these webpages with major browsers (Chrome, Mozilla Firefox, Safari, Edge, Explorer, Opera) to collect data.

4.2 Tests

Table 1 summarizes the tests we executed in different browsers. Hereafter, we provide implementation details that allow to reproduce these tests.

4.2.1 *Support require-sri-for.*

We tested if browsers enforce the require-sri-for directive that can be specified through HTTP headers. Overall, we noticed that this directive is not supported in web browsers. In Chrome and Firefox it has to be explicitly turned on with a flag when launching the browser. In addition, its removal from the W3C recommendation is scheduled.

4.2.2 *Support well formed sha2 checksums.*

We tested if browsers support the algorithms from the sha2 family (sha256, sha384 and

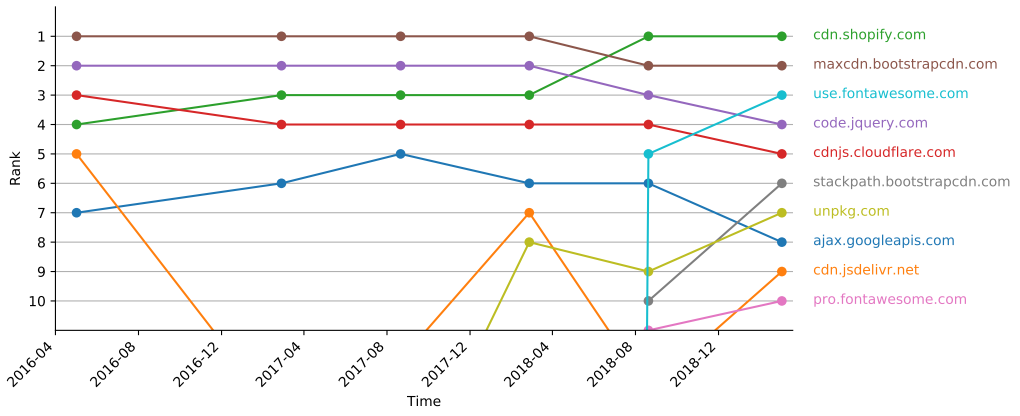
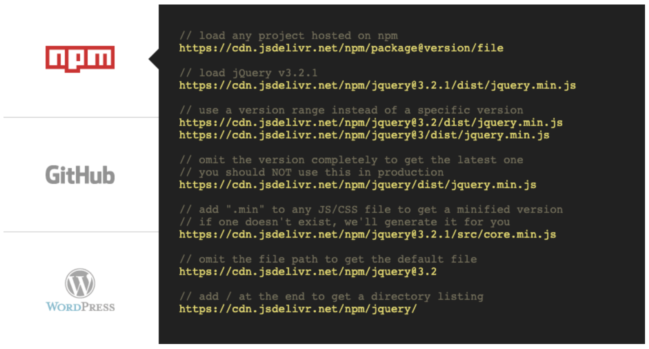


Figure 9: PDF/CDF of the top 20 domain names targeted by tags (script or link) that specifies the integrity attribute



```
// load any project hosted on npm
https://cdn.jsdelivr.net/npm/package@version/file

// load jQuery v3.2.1
https://cdn.jsdelivr.net/npm/jquery@3.2.1/dist/jquery.min.js

// use a version range instead of a specific version
https://cdn.jsdelivr.net/npm/jquery@3.2/dist/jquery.min.js
https://cdn.jsdelivr.net/npm/jquery@3/dist/jquery.min.js

// omit the version completely to get the latest one
// you should NOT use this in production
https://cdn.jsdelivr.net/npm/jquery/dist/jquery.min.js

// add ".min" to any JS/CSS file to get a minified version
// if one doesn't exist, we'll generate it for you
https://cdn.jsdelivr.net/npm/jquery@3.2.1/src/core.min.js

// omit the file path to get the default file
https://cdn.jsdelivr.net/npm/jquery@3.2

// add / at the end to get a directory listing
https://cdn.jsdelivr.net/npm/jquery/
```

Figure 10: jsDelivr relax integrity to gracefully upgrade subresources

sha512) and noticed that they behave according to the recommendation.

4.2.3 Support empty integrity attribute. We tested if browsers should include the subresources with an empty integrity attribute and noticed that they behave according to the recommendation.

4.2.4 Support missing base64 trailing equals. We tested if the removal of trailing equals (=) introduced by the encoding of the checksum in base64 affects the inclusion of the subresources. We found that all browsers support the removal of these extra characters.

4.2.5 Support conflicting checksums. We tested if browsers support conflicting checksums, i.e., what happen if two different sha256 checksums are specified for the same target. We found that all browsers support this feature and execute the script if one of the two checksums matches.

```
<script src="http://.../script.js"
        integrity="sha256-v1... sha256-v2..."
        crossorigin="anonymous"></script>
```

4.2.6 Block well formed wrong checksum. We tested if browsers prevent the execution of a script that specifies the wrong integrity attribute. We found that all browsers behave consistently.

```
<script src="http://.../script.js"
        integrity="sha256-wrong..."
        crossorigin="anonymous"></script>
```

4.2.7 Block malformed checksum. We tested if browsers prevent the execution of malformed integrity attributes, e.g., by using the string "malformed" as a value for the integrity attribute. Surprisingly, an error was raised by most browser, but the execution of the script was not prevented.

```
<script src="http://.../script.js"
        integrity="malformed"
        crossorigin="anonymous"></script>
```

4.2.8 Block md5 checksum. Knowing from our analysis that some websites specify md5 checksums, we specified one to check if such inclusion is prevented. Interestingly, we observed errors similar to the one raised for malformed checksums. In other words, the integrity attribute is considered as malformed, hence the script is executed.

```
<script src="http://.../script.js"
        integrity="md5-MGI0NTUx..."
        crossorigin="anonymous"></script>
```

4.2.9 Block missing hash prefix. Similarly, knowing that some web developers forget the dash between the algorithm and the digest, we tested this corner case and noticed that the scripts get executed.

```
<script src="http://.../script.js"
        integrity="sha256ivzZrYOz..."
        crossorigin="anonymous"></script>
```

4.3 Results

5 WEB DEVELOPER EXPERIENCE

We interview web developers and system administrators to assess their understanding of the SRI recommendation.

	Chrome	Firefox	Safari	Edge	Explorer	Opera	Android	Chrome	Firefox	Safari	Spec.
Support require-sri-for	✓*	✓*	-	-	-	-	-	-	-	-	✓*
Support well formed sha2 checksums	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓
Support empty integrity attribute	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓
Support missing base64 trailing equals	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓
Support conflicting checksums	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓
Block well formed wrong checksum	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓
Block malformed checksum	✗	✗	✗	-	-	✗	-	✗	✗	✗	-
Block md5 checksum	✗	✗	✗	-	-	✗	-	✗	✗	✗	-
Block missing hash prefix	✗	✗	✗	-	-	✗	-	✗	✗	✗	-

Table 1: Browser behaviors and recommended implementation regarding subresource handling. (A flag activates require-sri-for in Chrome and Firefox and the removal of the directive from the W3C recommendation is scheduled*).**

5.1 Screening Questionnaire

Research questions (Draft):

(1) Age/Gender/Profession/Education/...
(2) Are you a web developer? If yes, since...
(3) Are you a system administrator? If yes, since...
(4) ...

5.2 Interview protocol

Research questions (Draft):

(1) Have you already heard about SRI?
(2) Have you already used SRI?
(3) Have you already encountered the integrity attribute in a web page?
(4) If so, in which HTML element have you encountered this attribute?
(5) What is the purpose of the integrity attribute?
(6) Among these values for the integrity attribute, which one are well formed?
(7) Can SRI be used in place of HTTPS? Why?
(8) Can HTTPS be used in place of SRI? Why?
(9) Have you already integrated a third party library on a website?
(10) If so, how did you do?
(11) If so, what were your security concerns in doing so?
(12) Would you use SRI with other kind of subresources?
(13) ...

6 ADOPTION INCENTIVES

6.1 Methodology

To understand what could be done to improve the adoption of SRI, we compute the list of the most popular javascript and css subresources with or without the integrity attribute in a 1% random sample of the February 2019 *CC* dataset. For every subresource present in this sample, we take the one with an absolute url that ends with the **.js* or **.css* extensions.

6.2 Results

Table 2 shows some discrepancy between subresources with and without the integrity attribute. In other words, there is plenty of room left to improve the adoption of SRI. Here, we can categorize

(1) The results falling in the add-on category can hardly adopt SRI. For instance, as the social buttons and tracking codes from major providers are constantly evolving, the coupling introduced by the integrity attribute is unbearable. However, this question the responsibility of web developers when integrating such third-party libraries.
(2) The snippet provided by Facebook loads the script asynchronously which explains why their add-on is not included in the results.

7 RELATED WORK

SRI [? ? ? ? ?].
SRI limitations for unknown content [? ?].
Download checksums [?].
Third-party libraries [?].
Security awareness assessement [?].
Content security policy [? ? ? ?].
Alternatives to SRI [? ? ? ?].
Risks and Motivations [? ? ? ? ? ? ?].

- Checksums and integrity verification
- subresource integrity (SRI)

8 CONCLUSION

ACKNOWLEDGMENTS

The authors express their sincere gratitude to the anonymous reviewers for their insightful feedback. The authors also warmly thank Holly Cogliati for her great editing job on the manuscript and Romain Artru for his help in the implementation of the analysis scripts. The work was partially funded with grant #19024 from the Hasler Foundation and a UNIL-HEC Lausanne Research Fund.

Rank	Perc.	Name	Type	Snippet	SRI	Adoption
1	11.72%	Google Syndication	Add-on	✓		
2	6.39%	jQuery	Library	✓	✓	Mar.16
3	5.04%	Wordpress	Platform			
4	2.62%	Google APIs	Add-on	✓		
5	2.40%	Blogger	Platform			
6	2.21%	FontAwesome	Library	✓	✓	N/A
7	1.41%	TripAdvisor CDN	Platform			
8	1.38%	Twitter	Add-on	✓		
9	1.29%	SmugMug	Platform			
10	1.21%	Squarespace	Platform			
11	1.21%	Bootstrap	Library	✓	✓	Oct.15
12	1.19%	WIX	Platform			
13	1.13%	Google Ad Services	Add-on	✓		
14	0.94%	Google Tag Services	Add-on	✓		
15	0.94%	jQueryUI	Library	✓	✓	Mar.16
16	0.87%	Google Recaptcha	Add-on	✓		
17	0.86%	Google Analytics	Add-on	✓		
18	0.73%	BigCommerce	Platform			
19	0.73%	Shopify	Platform		✓	
20	0.66%	CookieConsent2	Library	✓		

Table 2: Most popular third-party resources referenced among script and link elements (N=158,284,348**)**