An Empirical Study of the Usage of Checksums for Web Downloads

Gaël Bernard, Rémi Coudert, Bertil Chapuis, Kévin Huguenin

An Empirical Study of the Usage of Checksums for Web Downloads

Gaël Bernard
EPFL
Switzerland
gael.bernard@epfl.ch

Rémi Coudert
University of Lausanne
Switzerland
remi.coudert@unil.ch

Bertil Chapuis
University of Applied Sciences Western Switzerland
Switzerland
bertil.chapuis@heig-vd.ch

Kévin Huguenin
University of Lausanne
Switzerland
kevin.huguenin@unil.ch

ABSTRACT

Checksums, typically provided on webpages and generated from cryptographic hash functions (e.g., MD5, SHA256) or signature schemes (e.g., PGP), are commonly used on websites to enable users to verify that the files they download have not been tampered with when stored on possibly untrusted servers. In this paper, we elucidate the current practices regarding the usage of checksums for web downloads (hash functions used, visibility and validity of checksums, type of websites and files, *etc.*), as this has been mostly overlooked so far. Using a snowball-sampling strategy for the 200,000 most popular domains of the Web, we first crawled a dataset of 8.5M webpages, from which we built, through an active-learning approach, a unique dataset of 277 diverse webpages that contain checksums. Our analysis of these webpages reveals interesting findings about the usage of checksums. For instance, it shows that checksums are used mostly to verify program files, that weak hash functions are frequently used, and that a non-negligible proportion of the checksums provided on webpages do not match that of their associated files. Finally, we complement our analysis with a survey of the webmasters of the considered webpages ($N = 26$), thus shedding light on the reasons behind the checksum-related choices they make.

CCS CONCEPTS

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

KEYWORDS

web downloads; integrity; checksums; crawl; survey

ACM Reference Format:
Gaël Bernard, Rémi Coudert, Bertil Chapuis, and Kévin Huguenin. 2023. An Empirical Study of the Usage of Checksums for Web Downloads. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583326

1 INTRODUCTION

Data integrity, which means that data is not altered, is a paramount property in information security [9]. And checksums—fixed-size strings generated from cryptographic hash function (e.g., SHA256) or signature schemes (e.g., PGP) and typically represented in hexadecimal or base64—are commonly used to verify integrity. While integrity is often verified automatically by programs when they fetch data from the Internet, for web downloads, checksums are provided on webpages and users are expected to verify integrity manually, with the help of programs such as `md5sum` or `shasum`. This enables users to verify that the files they download have not been tampered with when stored on possibly untrusted servers such as those of content delivery networks (CDN).

This threat is real: Recently, hackers distributed a modified version of the video player `VLC` infected with a malware that gave them access to the victims' computers [7, 19]. Similarly, in 2016, the BitTorrent client `Transmission` was also infected [1, 24].

Although checksums are relatively common on the Web, they received only little attention from the research community and very little is known about the current usage of checksums for web downloads: To the best of our knowledge, the only study in the literature covers only twenty download webpages with checksums and provides only basic information about their usage [3]. A possible reason for that is the lack of appropriate datasets.

In this paper, we study the usage of checksums for web downloads based on a unique dataset of 277 diverse download webpages that contain checksums for verifying the integrity of the files offered for download. We built this dataset through an original active learning-based process involving multiple labelers over an optimized snowball crawl of 8.5M webpages from the 200,000 most popular domains on the Web (according to QuantCast). The quality of this process was high, with an inter-labeler agreement, measured through Fleiss' kappa, consistently greater than 80%. Through a combination of manual and automated annotations, we further extracted information about the webpages in our dataset and about their usage of checksums.

Our analysis of these webpages reveals interesting findings about the usage of checksums. For instance, it shows (1) that checksums are used mostly to verify program files for various platforms (Windows, macOS, Android), archives (typically source code), and disk images (typically of operating systems), (2) that hash functions considered as weak/broken today are still frequently used (43.0%

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583326

of the webpages use only weak hash functions), and (3) that a non-negligible proportion of the checksums provided on webpages do not match that of their associated files (6.1%). We complement our analysis with a survey that targets the webmasters of the pages in our dataset. Our results ($N = 26$) show that webmasters (1) provide checksums because they believe integrity verification is important, because their users asked them to, or to follow community standards, (2) provide multiple checksums to maximize the chances that users can verify one of them, and (3) would be likely to adopt sub-resource integrity [21] instead if it was extended to downloads.

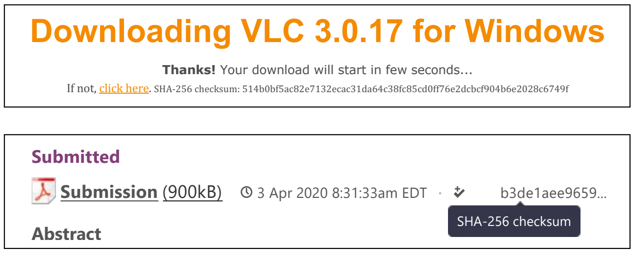
We make the following contributions: (1) We gather and annotate—to the best of our knowledge—the most extensive collection of checksums for web downloads. (2) We shed light on the current usage of checksums (and the underlying reasons for this usage) and on their validity. And (3) we make available to the community our dataset and the code for collecting and analyzing it.

The rest of the paper is organized as follows. In Section 2, we provide the necessary background on checksums and describe the considered adversarial model. In Section 3, we survey existing related work and position our work with respect to it. In Section 4, we describe our data collection process. We present the results of our analysis of the collected data in Section 5 and report on the results of our user survey (targeted at webmasters) in Section 6. Finally, we discuss the dissemination of our research data and code in Section 7 and conclude the paper in Section 8.

2 BACKGROUND

A checksum is a fixed-size piece of data usually obtained by applying a cryptographic hash function (e.g., SHA256) to some data (e.g., a file). The security properties of such hash functions guarantee that it is intractable for an adversary to forge a file with a pre-determined checksum or to forge another file with the same checksum as a given file (i.e., pre-image attacks). This is convenient when one wants to ensure that two files have the same content and, by extension, that a file has not been modified (i.e., its integrity is preserved). Some hash functions, such as MD5 and SHA1, were proved vulnerable to such attacks [22, 23].

Digital signatures (e.g., PGP+RSA/DSA) are similar to checksums, but they involve a private key in the generation process and the associated public key in the verification process. Public keys can usually be retrieved from so-called key servers (or public-key infrastructure, PKI). Digital certificates also offer similar features but we do not consider them in this paper. Signatures not only enable the integrity verification of files but also the authentication of the signer. Signatures can be embedded in the signed file (attached) or provided as a separate file (detached). Checksums and signatures can be used in combination, by providing a signed file containing a checksum. Although this authenticates the checksum and prevents an adversary to tamper with the checksum, it does not offer any additional protection if the adversary manages to mount a pre-image attack (i.e., tamper with the file without changing its digest). Checksums and signatures are often represented as hexadecimal or base64 strings, especially when provided "inline" on webpages. They are used to enable users to verify the integrity of the files they download or upload (see Figure 1). Of course, checksums make sense mostly in the case where the file and its checksum are stored in different locations (e.g., servers), otherwise an adversary who tampers with the file would also tamper with the checksum to make it match (a respondent of our survey provided an interesting case about this point; see Section 6).



Downloading VLC 3.0.17 for Windows

Thanks! Your download will start in few seconds...

If not, click here. SHA-256 checksum: 514b0bf5ac82e7132ecac31da64c38fc85cd0ff76e2dcbcf904b6e2028c6749f

Submitted

Submission (900kB) 3 Apr 2020 8:31:33am EDT · b3de1aee9659...

SHA-256 checksum

Abstract

Figure 1: Illustrations of the use of checksums for integrity verification on the Web.

The typical adversarial model underlying the use of checksums is that of an adversary who seeks to corrupt a file–typically a program–available for download, for instance to infect the devices of the users who download (and open) it. And the main goal of checksums is to enable users to detect such corrupted files. Ideally, checksum verifications should be done automatically without the need for user intervention; it is not (yet) the case for checksums for web downloads (which we study in this paper) but it is for web subresources with the subresource integrity (SRI [21]) mechanism.[1]

3 RELATED WORK

Our work relates to integrity verification techniques (see [9] for the different definitions of integrity, here we mean data integrity) , especially hash functions and signatures, and to the security of web (sub-)resources, including downloads.

From a usability perspective, Dechand et al. [5] show, through a user study ($N = 1047$), that the comparison of textual representations of fingerprints/checksums by humans is error-prone and vulnerable to partial pre-image attacks. Tan et al. [20] show that the comparison of graphical fingerprint representations is more intuitive and faster than with textual representations. Through an experiment ($N = 40$) that involved eye-tracking, Cherubini et al. [3] confirms the error-proneness of checksum verification, in the context of web downloads. They also study the usage of checksums on the download pages of twenty popular programs. Finally, through a user survey ($N = 2000$), they show that most internet users do not understand the purpose of checksums found on download pages. In a follow-up study, Meylan et al. [13] studied, through an in-the-wild experiment ($N = 134$), the exposure of internet users to checksums and their reactions.

Another body of works [2, 10, 16–18] studied the use of a standardized integrity verification mechanisms available in web browsers, namely the sub-resource integrity (SRI) security feature, [21] which consists in verifying automatically the integrity of a sub-resource by using a checksum specified in its attribute, within the HTML code of the webpage that includes it (e.g., `<script src="http://cdn.com/script.js"`

[1] We briefly describe SRI in Section 3. We believe that SRI should be extended to web downloads. We polled webmasters about SRI and report on the results in Section 6.

`integrity="sha256-26a4D..."/>`). The authors show that the adoption of SRI is low but increasing (between 3-4% of the websites in Common Crawl '19 [4] use it) and that some aspects of SRI are not well understood by web developers.

4 DATA COLLECTION

To study the current practices regarding checksums for web downloads, we built a unique dataset of (download) webpages with checksums. In a first attempt, we tried to identify such webpages from the Common Crawl dataset[2] by analyzing—with regular expressions for detecting specific keywords and checksums represented as hexadecimal strings—the content of the pages (around 3 billion) in the WARC (raw HTML, no subresources) and WET (plain text conversion) formats. However, we faced two main issues: (i) Common Crawl [4] is an incomplete (random) sample of the Web:[3] Typically, many of the pages with checksums identified by Cherubini et al. [3] were not present in it. (ii) A tremendous proportion of webpages contain checksum-like strings that are not related to the verification of downloads (i.e., false positives). Examples of such strings include technical discussions (e.g., documentation, tutorials, blogs), URLs, filenames, and commit versions (e.g., GitHub, GitLab) that contain hash values. For these reasons, we decided to take another approach, which we describe in this section.

4.1 Collection of Webpages

We collected webpages in late 2020 by using a snowball-sampling strategy from the top-200,000 most popular domains of the Web, according to Quantcast's ranking.[4] For each domain, and beginning with its homepage, we processed its webpages as follows. We parsed the *raw static* HTML code (without loading external resources and running embedded scripts)[5] of the page with the BeautifulSoup Python library and extracted all its internal URLs (i.e., pointing to the same domain) from the `href` attribute of HTML `<a>` elements. We inserted them in a priority queue (if not already present), where the priority was based on presence of keywords 'download', 'app', 'macos', 'windows', 'product', or 'software', because such URLs are more likely to point to download pages. The goal of using this heuristic is to maximize the chance of discovering (download) pages with checksums. Among pages with the same priority, the next page to process was drawn from the queue uniformly at random. We limited the sampling to 60 pages per domain and obtained a total of almost 8.5M pages, hereafter referred to as the CrawlQC8M dataset.

4.2 Identification of Pages with Checksums

On the one hand, and as discussed above, assessing automatically—in an unsupervised manner—if a webpage contains a checksum is a non-trivial task. On the other hand, assessing it manually is time consuming (it took 39.7 seconds per webpage on average for a human "labeler" in our experiments; this means 10+ person-years of work to evaluate CrawlQC8M fully) due to the cognitive effort involved. Typically, some checksums are challenging to spot due to information overload or because they appear only upon specific user actions (e.g., click, mouseover). In such a setting, active learning—where a learning algorithm (i.e., a classifier in our case) can interactively query a human agent to label new data instances—is a first class option, as "active learning is well-motivated in many modern machine-learning problems, where unlabeled data may be abundant but labels are difficult, time consuming, or expensive to obtain" [15].

We took several rounds of active learning to label (a part of) our CrawlQC8M dataset. Table 1 depicts the process in more detail.[6] First, we built a small training set: We started with the set of 20 webpages with checksums from previous work [3] (all were present in CrawlQC8M). Then, we expanded this by running an informal small-scale crowd-sourcing campaign in the lab, i.e., during a few months, the lab members reported to the authors the webpages with checksums they encountered; they also reported downloaded webpages without checksums to obtain related yet negative samples. This left us with an initial (training) dataset of 100 webpages with checksums and 139 without.

We built a binary classifier—based on a random forest—to assess if a page contains a checksum. We crafted a total of 14 features, including the presence of specific (stemmed)[7] keywords (e.g., integrity, verify, download), file extensions (e.g., exe, dmg, deb, zip), hash function / signature scheme names (e.g., md5, sha1, sha256, pgp), and strings that match regular expressions for checksums in a hexadecimal format. We built these features not only from the HTML code but also from sub-resources, including JS scripts. The complete list of features is provided in Appendix A. To select the model and associated parameters, we relied on a grid search with 10-fold cross-validation, applied to the initial dataset. We found that decision trees with 100 trees with a maximum depth of 8 were the best settings. We fixed these parameters for the remainder of the data collection.

In the three rounds of active learning, we iteratively expanded our dataset by: (1) applying the current version of the classifier to *all* the unlabeled pages from CrawlQC8M, (2) selecting the pages that are the most likely to include checksums, according to the classifier (classifiers often output a probability distribution over the possible labels, 'w/ checksum' and 'w/o checksum' in our case), and (3) manually labeling—with multiple human labelers (i.e., lab members) per page—the selected pages. The labeling of the webpages was done in a web-based tool.For this step, the webpages were fully rendered (not only the raw HTML code). At the end of each round, we resolved conflicts between labelers through discussion, and we updated (i.e., re-trained) the classifier with the newly expanded dataset.

In the first round, 400 pages were selected, based on the output of the classifier, and labeled by a single human labeler. Of these 400 webpages, 75 were labeled as "w/ checksums", and the other as "w/o checksums" (false positives). The classifier's precision (i.e.,~20%) is satisfactory for our use case.[8] At the end of the first round, the

[2] https://commoncrawl.org/, last visited: Feb. 2023.

[3] https://groups.google.com/g/common-crawl/c/xmSZX85cRjg/m/RYrdBn2EBAAJ, last visited: Feb. 2023.

[4] https://www.quantcast.com/top-sites/, last visited: Oct. 2022.

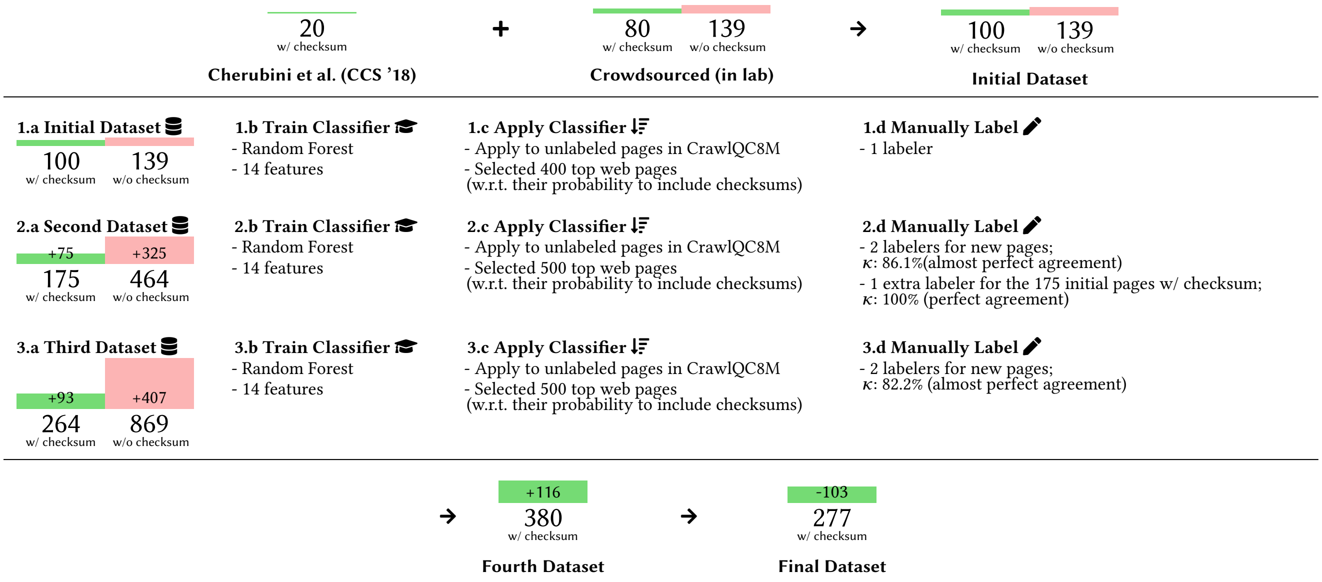
[5] Note that, for the automatic analysis of the webpages, we also included JS code and, for the manual analysis, we fully rendered the webpages by loading their sub-resources and running their scripts (see Section 4.2 and 4.3).

[6] Le et al. [12] took a similar approach, mixing automatic/manual analyses.

[7] See https://en.wikipedia.org/wiki/Stemming, last visited:: Feb. 2023.

[8] Note that we cannot analyze its recall, as we do not have access to the ground truth for the webpages that we do not manually annotate.

Table 1: Data collection process with active learning.



20 w/ checksum — Cherubini et al. (CCS '18) + 80 w/ checksum, 139 w/o checksum — Crowdsourced (in lab) → 100 w/ checksum, 139 w/o checksum — Initial Dataset

1.a Initial Dataset 100 w/ checksum 139 w/o checksum	**1.b Train Classifier** - Random Forest - 14 features	**1.c Apply Classifier** - Apply to unlabeled pages in CrawlQC8M - Selected 400 top web pages (w.r.t. their probability to include checksums)	**1.d Manually Label** - 1 labeler
2.a Second Dataset +75 / 175 w/ checksum +325 / 464 w/o checksum	**2.b Train Classifier** - Random Forest - 14 features	**2.c Apply Classifier** - Apply to unlabeled pages in CrawlQC8M - Selected 500 top web pages (w.r.t. their probability to include checksums)	**2.d Manually Label** - 2 labelers for new pages; κ: 86.1%(almost perfect agreement) - 1 extra labeler for the 175 initial pages w/ checksum; κ: 100% (perfect agreement)
3.a Third Dataset +93 / 264 w/ checksum +407 / 869 w/o checksum	**3.b Train Classifier** - Random Forest - 14 features	**3.c Apply Classifier** - Apply to unlabeled pages in CrawlQC8M - Selected 500 top web pages (w.r.t. their probability to include checksums)	**3.d Manually Label** - 2 labelers for new pages; κ: 82.2% (almost perfect agreement)

→ +116 380 w/ checksum — Fourth Dataset → -103 277 w/ checksum — Final Dataset

dataset contained 175 pages labeled as 'w/ checksums' and 464 pages labeled as 'w/o checksums'.

In the second round, 500 new pages were selected, again based on the classifier's output, and labeled by two different human labelers each. The labeling tasks were evenly distributed across six human labelers. The 175 pages labeled as 'w/ checksum' in the first round were labeled by an additional labeler in order to provide some quality control. To measure inter-labeler agreement (labelers are human: they make mistake, e.g., missing a checksum on a page) and the reliability of the labeling, we relied on Fleiss' kappa [6, 8, 25]. We obtained $\kappa = 86.1\%$ (i.e., "Almost perfect agreement", according to the interpretation table of Landis and Koch [11]) for the new pages and $\kappa = 100\%$ ("perfect agreement") for the initial 175 pages with checksums, i.e., the labels of all these pages were confirmed by the second labeler.

In the third round, we selected and labeled another 500 pages, with two labelers (among a pool of six) per page. We obtained $\kappa = 82.2\%$ ("Almost perfect agreement"). The proportion of pages labeled as 'w/ checksums' increased from round two to round three, showing an increase in the accuracy of the classifier. In Appendix C, we show the ranking of the features with respect to their importance in the classification process for the final round. The top three were: the fraction of words listed in keywords, the presence of names of techniques, and the presence of hash values.

Overall, our active learning-based data-collection process enabled us to reach a total of 380 webpages with checksums. In Appendix B, we provide details on the performance of the final classifier trained on this dataset.

4.3 Annotation of Pages with Checksums

Through a combination of manual and automatic annotation, we enriched our dataset with additional information for each of the webpages that contain checksums.

Manual Annotation. By using a dedicated web tool, we manually collected the following information:

(1) the URLs of the pages containing (resp.) the download link for the file, the checksum of the file, and the instructions for verifying checksums (if any, e.g., https://gnupg.org/download/integrity_check.html), as well as whether user action (e.g., click, mouseover) was needed to show the checksum (e.g., https://get.videolan.org/vlc/3.0.17.4/win32/vlc-3.0.17.4-win32.exe, "display checksum")
(2) the integrity verification technique(s) used (MD5, SHA1, SHA256, PGP, *etc.*)
(3) general information about the website: apache-like index page (e.g., https://unit.nginx.org/download/), file-hosting service (e.g., https://www.fosshub.com/Brave-Browser.html), forum or blog (e.g., https://newsgroup.xnview.com/viewtopic.php?f=82&t=40945)
(4) one checksum instance with the strongest technique (e.g., if an MD5 checksum and a SHA256 checksum were available, we selected the SHA256 checksum) containing the URL of the file to be downloaded and the value of the checksum or the URL of the checksum file. The file and the checksum file (if any) were subsequently downloaded
(5) the target platform for the file (e.g., Windows, macOS, Linux, Android) and information on whether the source code was available (for program files)
(6) the contact e-mail/form for the website. We collected e-mails and/or URLs of contact forms from the website and, when PGP was used, from the metadata of the public key used for the signature. The purpose of collecting this information was to survey the webmasters to understand the rationale and processes behind their use of checksums (see Section 6). For most webpages (196 or 70.8%), we identified at least one e-mail /

contact form. We identified more than one contact for a small fraction of pages (5.4%).

As for the identification of webpages with checksums, the annotation of the webpages was done by multiple lab members, but with only one annotator per page. Through annotation, we also removed duplicates (identical pages and pages from the same domains), 88 in total, and pages with paywall-protected downloads or broken links, 15 pages in total. We ended up with 277 webpages with checksums.

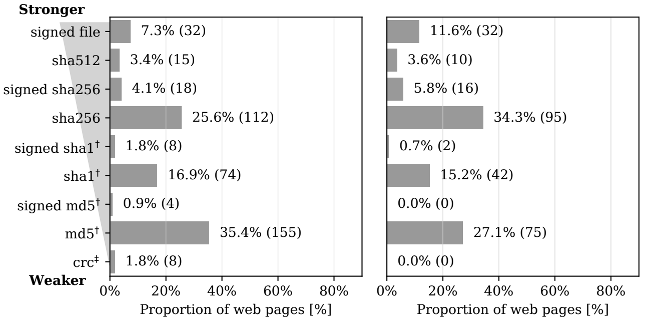
Determination of File Sizes and Types. We further extracted information about the downloaded files, essentially their sizes and their types. To determine the types of the downloaded files, we used the UNIX `file` command,[9] that relies on, among other information, the magic bytes of the file.[10] We further refined the file types by using the extensions of the files. We did so because some file types such as Java programs (.jar), Android mobile applications (.apk), and macOS applications (.dmg, .app)—which are all particularly relevant for our study—are, technically speaking, (compressed) archive files hence labeled as such by the `file` command. We labeled such files as 'executable'. Moreover, if an archive contained a single file, we extracted it and labeled it instead of using the 'archive' label. We regarded this approach as more reliable than those based solely on file extensions or on MIME types returned by web servers.

Validation of Checksums and Signatures. Finally, we determined whether the provided checksums and/or signatures matched the downloaded files. This automated process, depicted in Appendix D, differs for checksums, PGP signatures, and PGP-signed checksums. For checksums, we simply computed the digest of the file with the associated hash function and compared it to the provided checksum, regardless of whether the checksum was provided on the webpage (i.e., "inline") or in a separate file. For signatures, we simply verified their validity by using the associated public key obtained either from the website or from the Ubuntu key server (https://keyserver.ubuntu.com/), regardless of whether the signature was attached or detached. We further verified the expiration date of the public key. For signed checksum files, we first verified the signature then the checksum, as described above. The verification was considered successful if and only if both verifications succeeded.

If the verification was successful, the page was labeled as "succeed at first try". If the verification failed and the file was an archive, we made a second attempt: we extracted the files and applied the verification process described above to them. If this second validation was successful for at least one of the extracted files, the page was labeled as "succeed at second try". Otherwise, the page was labeled as "fail".

4.4 Limitations

Our data collection presents some limitations. First, it is biased by the features we crafted and the (initial) training dataset we used. In particular, we used (English) keywords related to specific file extensions and hash function and signature scheme names. Also, our initial (training) dataset was biased towards open-source software (CCS '18 [3]) and towards websites visited by researchers in computer science (i.e., the lab members), as the researchers are obviously not representative of the general internet population. Second, our crawl was limited to the most popular domains, according to QuantCast's ranking (note also that rankings are known to differ substantially), and it was not exhaustive. This also introduces a bias. Third, the first part of the analysis (i.e., *automatic* identification of pages with checksums) was done based only on the static code (HTML + JavaScript) of the analyzed webpage, thus we ignored the content that is dynamically loaded. Yet, this content was considered in the rest of the analysis (manual identification and annotation).



(a) Occurrences of techniques across all webpages.

(b) Strongest technique used per webpage.

Figure 2: Most frequently used techniques, ord. by strength (†: broken, ‡: not made to protect against adversaries).

5 RESULTS

We analyzed the data collected for the 277 webpages. Interestingly, 10.98% of the analyzed download webpages used SRI for at least one of their subresources (e.g., script, stylesheet)

Types of Websites. Among the websites in our dataset, 14.8% were file hosting websites (i.e., dedicated websites that host many different files produced by third parties, e.g., https://download.cnet.com, https://filehippo.com, https://www.fosshub.com). Moreover, 4.3% of the pages were default index pages (e.g., of the Apache HTTP server), where users should browse the list of files to find the appropriate checksum/signature, or forum/blog pages (3.6%).

Usage of Checksums. We first looked at the integrity-verification techniques used on the different webpages from our dataset, based on the manual annotations (see Section 4.3). The majority of webpages (52.7%) propose a single technique per file, 17.1% propose two (e.g., MD5 and SHA256 or SHA256 and PGP), and the rest (30.2%) propose three or more. A possible reason for using multiple techniques is to offer more options to the users. In particular, one could propose signatures for those users who have technical skills and care about authentication, and "simple" hash digests for the others.

The distribution of techniques used across all the webpage is depicted in Fig.2 (left). Only the nine most frequent techniques are shown; they are ordered by strength. We mark techniques considered as weak (i.e., MD5 and SHA1 [22, 23]) and those that do not offer—by design—protection against adversarial modifications (i.e., CRC). Present on 35.4% of the webpages, MD5 is the most frequently used technique, followed by SHA256 (25.6%) and SHA1 (16.9%). Some techniques, such as BLAKE2sp, MD4†, and SHA224, were

[9] https://en.wikipedia.org/wiki/File_(command).

[10] https://en.wikipedia.org/wiki/List_of_file_signatures.

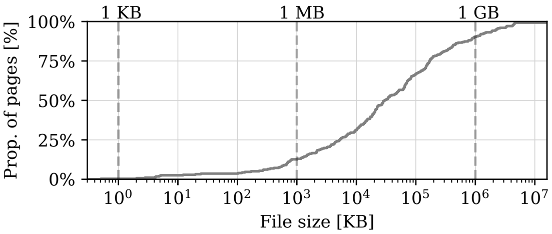


Figure 3: Distribution of file sizes (in KB); log-scale x-axis.

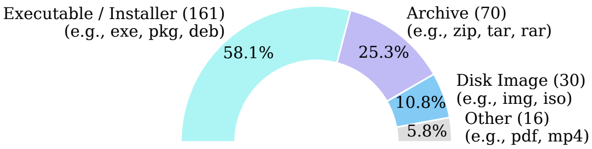


Figure 4: Distribution of files types.

also found but are not shown in the figure as they were observed less than 3 times. The fact that MD5 and SHA1 are weak limits the security provided by the checksums computed from these hash functions. Yet, as a substantial proportion of the webpages propose multiple techniques, such weak techniques might be accompanied by stronger ones, thus offering a secure alternative to the users. For this reason, we plotted the distribution of the *strongest* techniques used on the webpages and show it in Figure 2 (right). From this perspective, SHA256 is the most frequently used technique (34.3%, 40.1% if we include signed SHA256 checksums), followed by MD5 (27.1%), SHA1 (15.2%) and PGP signature (11.6%). CRC is never the strongest technique; i.e., it is always accompanied by other hashing functions. In total, a staggering 43.0% of the webpages use only weak techniques. Overall, the majority of the techniques are simple checksums (85.2%), followed by signed checksums (7.5%) and PGP (7.3%).

In terms of presentation, on 26 webpages, the inline checksums were made visible (in JavaScript) only upon a specific user action (e.g., click). In most cases (94.9%), checksums were available on the same webpage as the download. In 66.1% of the cases, the checksums were provided inline; in the other cases, they were provided as a separate file.

In terms of usability, only 16.2% of the webpages provided instructions on how to use checksums, either on the download page (51.1%) or on a dedicated webpage (48.9%). Given the limited knowledge of internet users regarding checksums [3], the lack of instructions impedes their wide use.

Sizes and Types of Files. We then looked at the sizes and types of the file for which a checksum was provided. We plotted the empirical CDF (in log-scale) of the file sizes and show it in Figure 3. It can be observed that the vast majority of the files weigh more than 1 MB. Files smaller than 1 MB were mostly (archived and possibly compressed) source codes.

We further looked at the types of the files, following the methodology described in Section 4.3, and show the distribution in Figure 4. It can be observed that more than half of the files (58.1%) are executable or program files. The next most frequent type of files protected with checksums are archives (25.3%), typically source code of programs. Note that 35.7% of the files were annotated as open-source by the annotators, based on the presence of the source code on the website, possibly in addition to the precompiled binary (i.e., the executable). The third most frequent file types is disk image, typically ISO images of operating systems (e.g., Ubuntu). Using integrity-verification techniques for these top-3 file types makes a lot of sense as it is rather straightforward to include virus/malware in them, compared to PDF documents for instance. In terms of target platforms, among the 161 pages where the download was an executable, 40.5% had a Windows version, 28.9% a macOS version, 23.2% a Linux version, and 7.4% an Android version.

Locality of Checksums and Files. Using the same heuristic as Chapuis et al. [2] on the URLs of the file and of the checksum file/page, we determined that 48% of the files were hosted on the "same server" as their checksum. While using checksum make less sense in such a set-up (one could assume that an adversary who could tamper with the file could tamper also with the checksum to make it match), it still makes sense (see Section 6–Purpose of Using Checksums).

Validity of Checksums and Signatures. In Figure 5, we show the results of the verification of the checksums and signatures, based on the methodology described in Section 4.3 and in Appendix D. We observed that most checksums were valid on the first try (92.4%) or after extracting the files (1.4%). For the rest, 17 pages in total, we could not match the checksum ($n = 13$) or signature ($n = 4$) to any file at all. After (manually) investigating the errors, we found out that some of these failure were due to a version mismatch between the file and the checksum. For instance, the checksum displayed for Spybot (https://www.safer-networking.org/files/) on the website was for version 2.8.67, while the downloaded (program) file was version 2.8.68. The risks of having such inconsistencies, and the complexity of the associated updating process, was mentioned as a major obstacle for the adoption of integrity-verification mechanisms (SRI in particular).

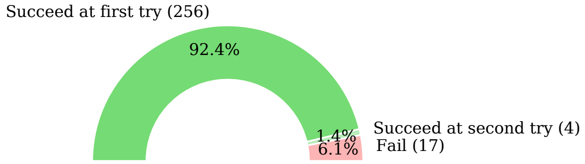


Figure 5: Distribution of outcome for checksum verification.

6 SURVEY OF WEBMASTERS

We created a questionnaire targeted at webmasters of websites that provide checksums for downloads. The questionnaire was personalized with the title of the website and a screenshot of the page containing the checksum(s). The questionnaire is available in the supplemental material. We first collected the consent of the respondents and asked them to confirm that they are involved in the management of the considered website and that they are aware that the website includes checksums for downloads. Then, we polled them—as webmasters (on this particular website) and as individuals—about their usage of checksums and about their interest in an alternative automatic technique for checksum verification for downloads (i.e., SRI). The questionnaire took about 10 minutes to complete and respondents were paid USD 30 for it, in the form of an Amazon voucher. The study was approved by our IRB.

We deployed the questionnaire (in the Fall of 2022) to the webmasters of the 277 pages with checksums we identified. For the 196 for which we found a contact e-mail or web form, we used it. For the rest, we used the generic "`webmaster@domain`" address. We obtained a total of $N = 26$ complete answers, including those from webmasters of very popular websites. Overall, the respondents took the survey very seriously, providing well-written, detailed, and insightful responses to the open-ended questions (a median of 25 words per response, for a total of 9 open-ended questions).

We analyzed the responses to the open-ended questions by using open and axial coding [14]. We iteratively read over the responses and coded them inductively. We subsequently classified the generated codes into main categories.

Use of Checksums. We asked the respondents how the decisions regarding checksums were made. We obtained a balance between individual (e.g., a single webmaster) and collective (e.g., "general agreement among the handful of people maintaining the website" [R15]) decisions. Often ($n = 7$), the respondents reported that the process was triggered by users' requests ("added in response to customer requests for it" [R13]). Interestingly, a respondent mentioned that the requests came "right after one of the SourceForge download servers was compromised" [R20]. Also, one respondent mentioned that "[they] don't think [they] would have done it on [their] own initiative" [R6]. Another factor that influenced the decision process was the community standards, especially for security-related software ("I've found that displaying a known-good checksum alongside a file download is a well-practiced convention." [R10]). Approximately one third of the respondents mentioned that they updated their choices (e.g., changing hash function) over time based on: the state of the art, community standards, and users' requests.

Purpose of Using Checksums. When asked about the purpose of using checksums, in general and for their websites, all respondents mentioned integrity or described the concept with their own words. A few respondents mentioned authenticity (for PGP). When describing the threats that checksums address, most respondents mentioned the risks of file corruption *in transit*. Although this is a valid threat, it is solved by the use of HTTPS. Corruption *at rest* (e.g., server hosting the website, mirror, CDN), which is addressed by checksums but not by HTTPS, was mentioned by several respondents ($n = 4$). A few respondents ($n = 3$) also stressed that checksums make sense only/mostly if the file and the checksum are hosted on different servers. Although this is a valid point, a comment from a respondent provides an interesting counterexample: "it happened that a virus (which I also already had on my servers) could change downloads so that they will act as distributor of virus (virus will usually only automatically infect binaries, they will not check webpages)" [R4]. Finally, a few respondents mentioned *only* accidental risks of corruption (e.g., disk failure, network error) but not adversarial ones (e.g., hackers).

Figure 6 depicts the importance of checksum verification, as reported by the respondents. It can be observed that 21 found it at least moderately important, hence showing that most webmasters of pages with checksums are convinced of their importance. In fact, a large majority of the respondents ($n = 17$) declared personally verifying checksums at least "sometimes" (when available) for the files they download from the Web. Interestingly, about half of the respondents ($n = 12$) declared that they came across a mismatch between the file and its checksum at least once.

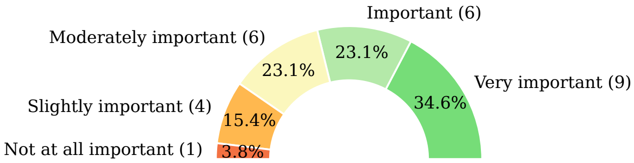


Figure 6: Reported importance of checksums for (verifying the integrity of) downloads.

Use of Weak Checksums. We asked the respondents whose websites include at least one checksum generated with a weak hash function (16 in total) whether they were aware of this weakness and why they decided to use them. All of them but one reported being aware of it. As for the reasons they decided to use weak hash functions despite this, the main reasons they mentioned were (1) historical reasons (e.g., started using SHA1 at a time it was not considered weak and stuck to it), (2) simplicity of use, compatibility, and familiarity to the users ("in Terminal we have '`md5 file`', but there is no '`sha256 file`' equivalent; you have to use '`openssl sha256 file`'..." [R18]),[11], and (3) the need to protect against only accidental corruption (and not adversarial ones). Interestingly, one respondent mentioned that they included MD5 and SHA1 because they were included in the output of the `gpg --print-mds` command, together with stronger ones (incl. SHA256); therefore, user had the choice to use stronger techniques. Finally, in some cases, weak hash functions were included on users' requests (for their convenience).

Use of Multiple Checksums. When asked about the use of multiple checksums, the respondents mentioned two main reasons: (1) to maximize the chances for the users to have the technical means (availability of software) and skills to verify at least one of the provided checksums, and for their convenience in doing so (familiarity with the software), (2) to provide a simpler means (digests) for regular users to check integrity and to provide a more complex one (PGP signatures) for advanced users to verify both integrity and authenticity. Only one respondent mentioned enhanced security (i.e., as it is harder to forge a file that matches *multiple* checksums) as motivation for including multiple checksums. As for the reasons for using a *single* checksum, respondents mentioned the facts that (1) it is enough, (2) it reduces confusion for users, and (3) it prevents overloading the page. They also reported that even when they decided to include a more recent checksum, they simply decided to replace the former checksum with the new one rather than keeping both.

[11] In some cases it was, again, on users' requests.

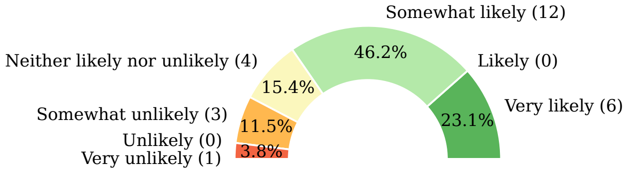


Figure 7: Reported likelihood of using SRI for verifying the integrity of web downloads.

Checksum Update Process. When asked about the checksum-update process, 12 respondents reported that it was manual and 13 automatic (typically a script that computes the checksum and pushes both the file and the page with the checksum to the server(s)); the others reported it was the mix of the two (e.g., manually running scripts). In the cases where the process was manual, it was often the responsibility of a single individual (i.e., the webmaster). In the cases where it was automatic, it was mostly done through home-made ad-hoc scripts; it was not part of an existing build-tool chain. This lack of integration could impede the use of checksums or cause mismatches. Other responses indicated that the process for updating the file and its checksum were often separated (with delays in some cases, e.g., checksums updated nightly) hence possibly creating (temporary) inconsistencies. Interestingly, a few respondents mentioned that they verify that the files and the checksums available on the pages match. One respondent preferred not to provide too much detail on the process, probably for security reasons. The majority of the respondents reported that the process took less than 5 minutes; it never took more than 30.

When asked to envision situations in which a checksum would not match the corresponding file (i.e., a mismatch), the main causes mentioned by the respondents were (1) corruption of the file on the server or during the transfer (accidental or adversarial), (2) different update delays for the server hosting the file (e.g., CDN) and for the server hosting the checksum–when different–hence creating inconsistencies, and (3) human errors (mostly when the process was manual, e.g., hashing the wrong file, copy-pasting only part of the checksum, updating only the file or the checksum).[12] Note that the first two causes are not related to the update process.

Extension of Sub-resource Integrity to Web Downloads. When asked about the likelihood of using SRI as an alternative to inline checksums for the downloads on their websites—if SRI were to be extended to downloads—, a large majority of respondents ($n = 18$) declared in the affirmative (see Figure 7). This is aligned with (and complementary to) the results of Chapuis et al. [2] for web *developers*. We personally believe this would be beneficial to users.

Finally, 2 respondent(s) reported they would be interested in a (paid) interview and 12 that they might be (they needed more information).

7 DISSEMINATION OF THE DATA & CODE

To enable researchers and practitioners to reproduce our work and to benefit from its result, we make our dataset and the code for collecting and analyzing it available on OSF.[13] More specifically, we provide (1) the pre-trained model—in the PMML format—of our classifier (for identifying webpages w/ checksums), together with a minimal example of how to use it in Python + `sklearn`, (2) our enriched dataset of webpages with checksums, together with the annotations (`csv` and `sqlite`), (3) the code of our crawler, and (4) the full transcript of our questionnaire.

8 CONCLUSION AND FUTURE WORK

In this paper, we have provided the first medium-scale study of the usage of checksums for web downloads, based on a unique dataset we built. Our results have shed light on the typical use of checksums (e.g., executable files and disk-image files) and have shown important issues with the current practices (e.g., use of weak hash functions, lack of instructions) that—together with the limitations of checksums highlighted in previous work [3] (e.g., limited understanding from users, error-proneness of the verification process)—call for improvements or alternative solutions. Our survey of webmasters has provided insights into the reasons behind the checksum-related choices they make. In particular, it has shown that checksums are often included on users' requests. It has also revealed the webmasters' strong interest in the extension of sub-resource integrity to web downloads.

For future work, by conducting individual interviews, we intend to gain more insight from webmasters about their usage of checksums for web downloads. We also intend to poll webmasters of websites that contain download pages w/o checksums. To increase the size and diversity of our dataset in the future, we set up a webpage (https://checksum.unil.ch/static/crowdsourcing/) for enabling people to submit URLs of download webpages with checksums. Finally, we intend to increase the accuracy of the classifier by including more features, including some extracted *after* having fully rendered the pages. (i.e., loading and running also their sub-resources), e.g., with Selenium.

ACKNOWLEDGMENTS

The authors are grateful to Holly Cogliati for her editing job, to Tanguy Berguerand for his help in the implementation of the preliminary analysis scripts, to Kavous Niksirat Salehzadeh for his feedback on the questionnaire, and to Dario Besson, Lev Velykoivanenko, and Noé Zufferey for their help in the annotation of the webpages. The work was partially funded with grant #19024 from the Hasler Foundation.

[12] One respondent mentioned that "there have been rare cases where [they] have modified the program (usually fixed a bug) after the signatures were generated [...] and then [they] forgot to regenerate the signature files" [R8].

[13] https://dx.doi.org/10.17605/OSF.IO/A9YKR

REFERENCES

[1] 2016. Transmission hijacked again to spread malware. https://blog.malwarebytes.com/threat-analysis/2016/09/transmission-hijacked-again-to-spread-malware/

[2] 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 *Proc. of the Web Conference*. IW3C2, 34–45. https://doi.org/10.1145/3366423.3380092

[3] 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

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

[5] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. An Empirical Study of Textual Key-Fingerprint Representations. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX.

[6] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76, 5 (1971), 378–382. https://doi.org/10.1037/h0031619

[7] Ghacks. 2022. Symantec says that hackers distributed a modified version of VLC and exploited it for malware attacks. https://www.ghacks.net/2022/04/11/symantec-says-that-hackers-distributed-a-modified-version-of-vlc-and-exploited-it-for-malware-attacks/

[8] Kevin A. Hallgren. 2012. Computing Inter-Rater Reliability for Observational Data: An Overview and Tutorial. *Tutorials in Quantitative Methods for Psychology* 8, 1 (Feb. 2012), 23–34. https://doi.org/10.20982/tqmp.08.1.p023

[9] Kelsey Harley and Rodney Cooper. 2021. Information Integrity: Are We There Yet? *ACM Comput. Surv.* 54, 2 (Feb. 2021), 1–35. https://doi.org/10.1145/3436817

[10] 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 event-place: Perth, Australia.

[11] J. R. Landis and G. G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics* 33, 1 (March 1977), 159–174.

[12] Hieu Le, Athina Markopoulou, and Zubair Shafiq. 2021. CV-Inspector: Towards Automating Detection of Adblock Circumvention. In *Proc. of the Network and Distributed System Security Symp. (NDSS)*. Internet Society, Virtual. https://doi.org/10.14722/ndss.2021.24055

[13] Alexandre Meylan, Mauro Cherubini, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2020. A Study on the Use of Checksums for Integrity Verification of Web Downloads. *ACM Trans. on Privacy and Security* 24, 1 (Sept. 2020), 1–36. https://doi.org/10.1145/3410154

[14] Johnny Saldana. 2021. *The Coding Manual for Qualitative Researchers* (4th ed ed.). SAGE Publishing, Thousand Oaks, California.

[15] Burr Settles. 2009. Active learning literature survey. (2009). Publisher: University of Wisconsin-Madison Department of Computer Sciences.

[16] 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

[17] 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. http://ijett.in/index.php/IJETT/article/view/345

[18] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. 2021. Who's Hosting the Block Party? Studying Third-Party Blockage of CSP and SRI. In *Proc. of the Network and Distributed System Security Symp. (NDSS)*. Internet Society, Virtual. https://doi.org/10.14722/ndss.2021.24028

[19] Symantec. 2022. Cicada: Chinese APT group widens targeting in recent espionage activity. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/cicada-apt10-china-ngo-government-attacks

[20] Joshua Tan, Lujo Bauer, Joseph Bonneau, Lorrie Faith Cranor, Jeremy Thomas, and Blase Ur. 2017. Can Unicorns Help Users Compare Crypto Key Fingerprints?. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 3787–3798. https://doi.org/10.1145/3025453.3025733

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

[22] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. 2005. Finding collisions in the full SHA-1. In *Annual international cryptology conference*. 17–36. tex.organization: Springer.

[23] Xiaoyun Wang and Hongbo Yu. 2005. How to break MD5 and other hash functions. In *Annual international conference on the theory and applications of cryptographic techniques*. 19–35. tex.organization: Springer.

[24] Christina Warren. [n.d.]. Popular BitTorrent Client Transmission Gets Infected With Malware Again. https://gizmodo.com/mac-bittorrent-client-transmission-gets-infected-with-m-1785957214

[25] Antonia Zapf, Stefanie Castell, Lars Morawietz, and André Karch. 2016. Measuring inter-rater reliability for nominal data – which coefficients and confidence intervals are appropriate? *BMC Medical Research Methodology* 16, 1 (Dec. 2016). https://doi.org/10.1186/s12874-016-0200-9

A LIST OF FEATURES

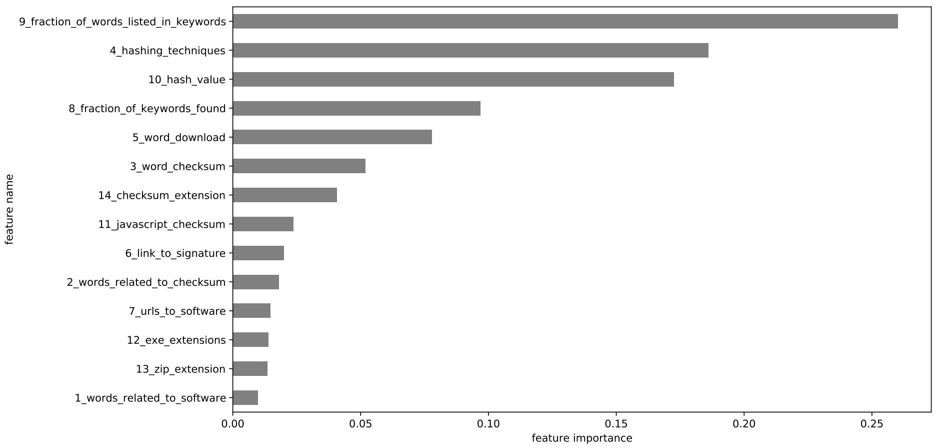
		Search Area							
Category	Name	Links[14]	Script[15]	Text[16]	Title[17]	URL[18]	URLs[19]	Type	Keywords[20]
Stemmed Keywords Lookup. True if at least one keyword is found.	1. Words related to software			✓				boolean	changelog, mirror, releas, tar, version
	2. Words related to checksum			✓				boolean	integr, verifi
	3. Word 'checksum'			✓				boolean	checksum
	4. Hash functions and signature schemes			✓				boolean	gpg, md5, pgp, sha, sha1, sha256, sha512
	5. Word 'download'				✓	✓		boolean	download
	6. Link to signatures	✓						boolean	gpg, pgp, signatur
	7. URLs to software						✓	boolean	donat, exe, ftp, mirror,win32, screenshot, sourceforg, sourceforg, tar, x86
Stemmed Keywords Summary. Summarize the presence of the keywords listed in the last column over the total set of words found in the web page.	8. Fraction of keywords found	✓	✓	✓	✓	✓	✓	float	changelog, checksum, donat, download, exe, ftp, gpg, integr, md5, mirror, mirror,win32, org, pgp, releas, screenshot, sha, sha1, sha256, sha512, signatur, sourceforg, tar, url, verifi, version, x86
	9. Fraction of words from the web page listed in keywords	✓	✓	✓	✓	✓	✓	float	changelog, checksum, donat, download, exe, ftp, gpg, integr, md5, mirror, mirror,win32, org, pgp, releas, screenshot, sha, sha1, sha256, sha512, signatur, sourceforg, tar, url, verifi, version, x86
Hash Value. Search for potential hash value.	10. Hash Value			✓				boolean	Regular expression for hexadecimal strings with valid checksum lengths
Javascript. True if both a hash value and a hash technique is found in the script.	11. Javascript Checksum		✓					boolean	md5, sha
Files' Extension. Search for specific extensions.	12. Exe extensions						✓	boolean	apk, app, deb, dmg, exe, msi, pkg, rpm
	13. Zip extensions						✓	boolean	7z, bz2, lzma, tar.gz, tgz, zip
	14. Checksum extensions						✓	boolean	checksum, gpg, hash, md, md5, pgp, sha, sig, signature, sum

B CLASSIFIER PERFORMANCE

At each stage of the active learning process, we selected an equal number of pages w/ checksums and w/o checksums and evaluated the performance of the resulting classifier. With leave-one-out cross-validation, we obtained the following performance for a random forest classifier (using the default parameters):

- Initial dataset: TP=0.48 , TN=0.47 , FP=0.03, FN=0.02, which yields accuracy=0.95, sensitivity=0.96, specificity=0.95, and f1=0.95
- Second dataset: TP=0.46 , TN=0.45 , FP=0.05, FN=0.04, which yields accuracy=0.91, sensitivity=0.91, specificity=0.9, and f1=0.91
- Third dataset: TP=0.45 , TN=0.43 , FP=0.07, FN=0.05, which yields accuracy=0.89, sensitivity=0.91, specificity=0.87, and f1=0.89
- Final dataset: TP=0.44, TN=0.46, FP=0.04, and FN=0.06, which yields accuracy=0.90, sensitivity=0.87, specificity=0.92, and f1=0.89.

C FEATURE IMPORTANCE



D CASE DIAGRAM FOR CHECKSUM VALIDATION

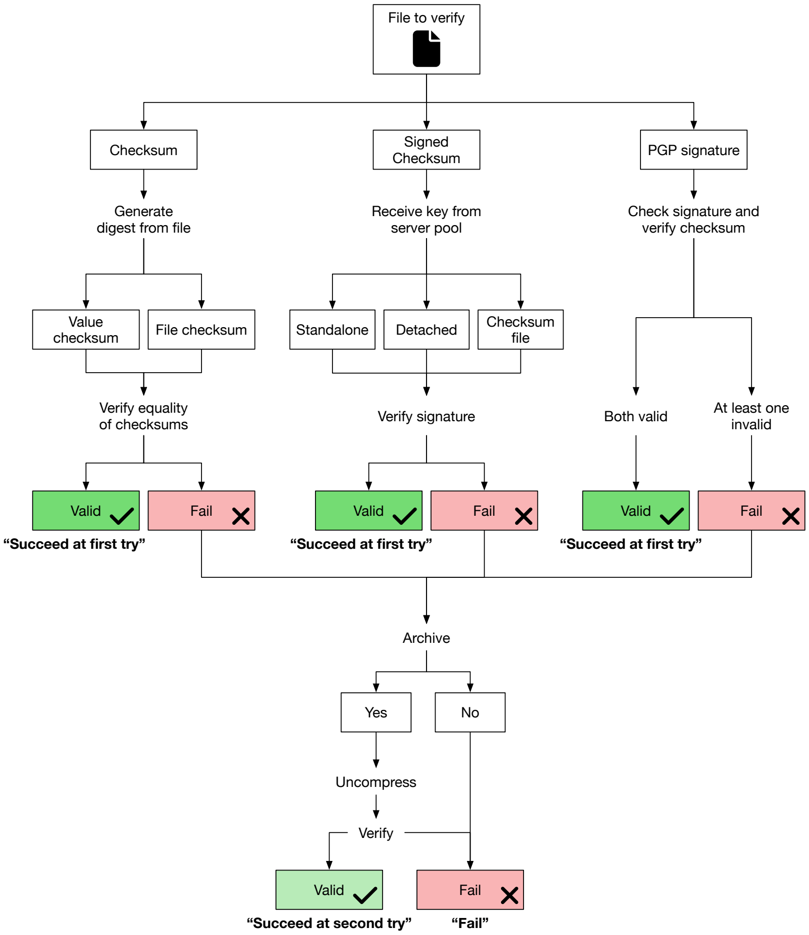


Figure 8: Case diagram of the automated validation process.