

A Study on the Use of Checksums for Integrity Verification of Web Downloads

Alexandre Meylan, Mauro Cherubini, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, Kévin Huguenin

▶ **To cite this version:**

Alexandre Meylan, Mauro Cherubini, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, et al.. A Study on the Use of Checksums for Integrity Verification of Web Downloads. ACM Transactions on Privacy and Security, 2020, 24 (1), pp.4:1-4:36. 10.1145/3410154 . hal-02892926

HAL Id: hal-02892926
https://hal.science/hal-02892926

Submitted on 3 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Study on the Use of Checksums for Integrity Verification of Web Downloads

ALEXANDRE MEYLAN, Kudelski Security, Switzerland
MAURO CHERUBINI, University of Lausanne (UNIL), Switzerland
BERTIL CHAPUIS, University of Applied Sciences and Arts (HES-SO/HEIG-VD), Switzerland
MATHIAS HUMBERT, armasuisse S+T, Switzerland
IGOR BILOGREVIC, Google Inc., Switzerland
KÉVIN HUGUENIN, University of Lausanne (UNIL), Switzerland

App stores provide access to millions of different programs that users can download on their computers. Developers can also make their programs available for download on their websites and host the program files either directly on their website or on third-party platforms, such as mirrors. In the latter case, as users download the software without any vetting from the developers, they should take the necessary precautions to ensure that it is authentic. One way to accomplish this is to check that the published file's integrity verification code – the checksum – matches that (if provided) of the downloaded file. To date, however, there is little evidence to suggest that such process is effective. Even worse, very few usability studies about it exist.

In this paper, we provide the first comprehensive study that assesses the usability and effectiveness of the manual checksum verification process. First, by means of an in-situ experiment with 40 participants and eye-tracking technology, we show that the process is cumbersome and error-prone. Second, after a 4-month long in-the-wild experiment with 134 participants, we demonstrate how our proposed solution – a Chrome extension that verifies checksums automatically – significantly reduces human errors, improves coverage, and has only limited impact on usability. It also confirms that, sadly, only a tiny minority of websites that link to executable files in our sample provide checksums (0.01%), which is a strong call to action for web standards bodies, service providers and content creators to increase the use of file integrity verification on their properties.

Additional Key Words and Phrases: checksums, integrity, security, usability, web downloads

ACM Reference Format:
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. Priv. Sec.* 24, 1, Article 4 (August 2020), 35 pages. https://doi.org/10.1145/3410154

1 INTRODUCTION

App stores are a very popular means for Internet users to get access to millions of apps for their computers and mobile devices. The most popular ones – such as Apple's, Google's, and Microsoft's – offer a curated set of apps that are easy to access for users, and they simplify the distribution for developers. However, app stores usually impose certain conditions on the software they are willing to publish. Furthermore, developers may face additional challenges when publishing on them,

This article is a revised and extended version of a paper that appears in the Proceedings of the ACM Conference on Computer and Communications Security (CCS 2018), Cherubini *et al.* [1].
Authors' addresses: Alexandre Meylan, alexandre.meylan@kudelskisecurity.com, Kudelski Security, Cheseaux-sur-Lausanne, Switzerland; Mauro Cherubini, mauro.cherubini@unil.ch, University of Lausanne (UNIL), Faculty of Business and Economics (HEC), Lausanne, Switzerland; Bertil Chapuis, bertil.chapuis@heig-vd.ch, University of Applied Sciences and Arts (HES-SO/HEIG-VD), Institute for Information and Communication Technologies (IICT), Yverdon-les-Bains, Switzerland; Mathias Humbert, mathias.humbert@armasuisse.ch, armasuisse S+T, Cyber-defence Campus, Thun, Switzerland; Igor Bilogrevic, ibilogrevic@google.com, Google Inc. Zurich, Switzerland; Kévin Huguenin, kevin.huguenin@unil.ch, University of Lausanne (UNIL), Faculty of Business and Economics (HEC), Lausanne, Switzerland.

2020. 2471-2566/2020/8-ART4 $15.00
https://doi.org/10.1145/3410154

such as long review and validation times, technical restrictions (e.g., sandboxing), incompatibility with software licenses, and substantial commissions [2]. For developers, a common alternative for distributing their software is to include a download link on their own websites, like for the popular VLC program; in this case, the program file can be hosted either on the website itself or on a third-party web hosting platform (e.g., mirrors, content delivery networks).

Hosting software on a website has several advantages for the developers, but it could also negatively affect the users. In particular, security is an important concern when downloading files from the Internet. Users could be tricked into downloading corrupted software that contains malware, which could impair the performance of their machine or even steal personal data from it. This scenario is not limited to the case where the software is hosted on a malicious platform, but can also happen if a legitimate hosting platform is compromised. In any case, by choosing to host their software on the web, developers also accept the risk that it could be accidentally or intentionally modified in an unpredictable way. Recently, both the popular BitTorrent client Transmission [3] and the Linux Mint distribution [4] were corrupted; the former by a ransomware and the latter with a backdoor. Such corruptions are particularly problematic for privacy and security software – such as Tor – used by at-risk populations such as journalists and political dissidents. In general, it is crucial for website administrators to make sure that the content of the files downloaded by their visitors through external links matches the content of the files at the time the link was created.

To mitigate such threats, developers can publish alphanumeric strings whose purpose is to enable users to verify that the downloaded software has not been accidentally or intentionally modified from the moment it was published and linked by its developer. Such strings, called *checksums*, are commonly used in the open-source community but also by companies such as Google (for their software Android Studio and Golang). Checksums are usually derived from the output of cryptographic hash functions (such as SHA-256) in the form of sequences of alphanumeric digits called digests, and are either displayed on the download webpage or are provided in a separate file. Users can then verify the integrity of the file they download based on the provided checksums.[1] However, there is currently no standard or common way for users to verify such checksums, other than manually executing dedicated commands on the operating system's terminal.[2] Worse yet, users are required to visually compare such long sequences of characters, which has been proven to suffer from usability issues and to be prone to errors [5, 6], in contexts other than web downloads (e.g., PGP key fingerprint verification). Other solutions, such as code-signing, also suffer from some limitations and only partially address the aforementioned problem. These issues call for automated and reliable methods.

To the best of our knowledge, no standard or practical solutions have been proposed for automatically verifying web downloads. Moreover, the research community has mostly overlooked the important topic of the integrity of programs downloaded on the Web. Our research fills that gap and addresses these important challenges, by first performing a thorough analysis of prevalence of the threat, the usability and effectiveness of checksums for the visual verification of the integrity of web downloads, and then by proposing technical solutions to the issues we identify. Specifically, our contributions are as follows:

- We carry out the first comprehensive study on checksums verification for the integrity of web downloads. To do so, we conduct an in situ experiment with 40 participants that uses an eye-tracker to precisely evaluate how users verify checksums. It is the first time that

[1]Note that checksums only enable users to verify that the file they downloaded is indeed the one the website administrator intended to share. They do not provide any guarantee that the file is safe to execute.

[2]By default, the major operating systems include only command-line tools to compute checksums, such as `shasum` for macOS and Linux and `certutil` for Windows.

eye-tracking technologies are used for studying usability and attention during the checksum verification process.
- We develop an automated checksum verification browser extension that alerts users when there is a mismatch between the checksum computed from the downloaded file and that (or those) available on the developer's website, when the checksum is displayed in the page.
- We conduct a 4-month in-the-wild experiment with 134 participants in order to study their download and browsing behavior, their exposure and understanding of checksums, and their reactions to our browser extension.
- To address the usability and effectiveness issues of checksums, we propose an extension to the current World Wide Web Consortium (W3C) specification for subresource integrity (SRI) [7]; it standardizes the use of checksums for external resources such as JavaScript files, to cover download links of program files. Our solution enables developers to rely on a standardized method that significantly reduces the user burden of checksum verification.

Our in-situ experiments demonstrate that the verification process is taxing, with a median of around ten back-and-forth that the eyes of the participants have to do between the checksum shown on the Web page and the output of the program used to compute the checksum of the downloaded file. It also demonstrates that, despite being explicitly asked to verify the checksums, more than one third of our participants fail to detect the mismatch (i.e., partial pre-image attack) between the checksum displayed on a fake download webpage and the one computed from the (corrupted) downloaded file. Our in-depth eye-tracking analysis shows that users pay more attention to the first digits of the checksums, which reduces drastically the security provided by such checksums. It also suggests that failure to detect mismatch between checksums is associated with a low number of fixations. Finally, the user feedback collected during the test of the extension that automates the process shows a good desirability of verification mechanisms integrated in web browsers.

Our in-the-wild experiment shows that our participants regularly download files that could include malware (e.g., binary executable files but also PDF files), which would therefore benefit from integrity verification. Specifically, in our experiment, 7% of all the downloaded files were binary executables and 56% were PDFs. It also demonstrates that only very few download webpages in our deployment (0.02%) currently provide checksums for integrity verification. Furthermore, it shows that the vast majority of our participants (88,6%) do not even notice checksums, know or understand their purpose, or know how to use them. It also suggests that users of our browser extension feel more secure, as compared to those who do not use it.

Compared to the conference version of this article [1], we focus on the in-situ experiment and substantially improve and extend the analysis of the data from the eye-tracking device. In particular, we include and analyze results on how users navigate between the checksums specified on the webpage and those computed in the terminal after download, and on the attention users devote to the different parts of the user interface of the extension. Our results shed light on the visual process of checksum (or fingerprints in general) verification and provide actionable feedback for the design of an automated tool. In addition, we clarify the system and threat models and include a performance evaluation of the browser extension (in order to assess the delays incurred by checksum verifications) and we elaborate on the feedback provided by the participants of the experiment. Moreover, we describe and report in detail on our 4-month in-the-wild experiment, which was conducted after our original work [1]. Finally, we make available to the community the dataset we collected through the eye-tracking device. We also open-source the browser extension on GitHub and distribute it on the Chrome Web Store.

The rest of the article is organized as follows. We survey the related work in Section 2. We introduce the system and threat models as well as the background about checksums and file integrity

verification in Section 3. We describe the proposed solutions – i.e., the browser extension – in Section 4, and present the in situ user experiments with eye-tracking in Section 5. We present the in-the-wild experiment in Section 6. We discuss the main findings and limitations of our work in Section 7. We conclude the article in Section 8.

2 RELATED WORK

From a high-level perspective, our work can be framed within the broader category of online security behaviors as it touches upon the subject of security warnings through the lenses of file integrity verification.

2.1 Download Behavior

Internet users are increasingly exposed to online security threats [8], and their security-related behaviors are influenced by a combination of cognitive (i.e., understanding of the related threats), social, and psychological components (i.e., time pressure to complete the related task) [9]. Often the weakest link – leading to many successful cyber-attacks – is the insufficient knowledge of the employees, which led to many successful cyber-attacks in the UK [10]. The download behavior is often also influenced by security recommendations [11–13], meaning that users evaluate digital-security recommendations based on the trustworthiness of the source of the advice; users might trust knowledgeable peers more than the source over the content of the recommendation. Unfortunately, none of these studies focused specifically on Internet downloads, which is one of the goals of this study.

2.2 Effectiveness of Security Warnings

A security warning is a cautionary message usually delivered by the operating system or an app to users when they are about to perform an action on their device that could have negative consequences. Such actions include downloading or opening a file containing a virus, visiting a website that contains malware, or simply installing an app from an untrusted source. The users can either act on such warnings or ignore them. Over the past decade, the research community has extensively studied how users interact with such warnings, and whether the warnings are effective and understandable [14–22]. These studies are relevant to our work as we also designed an intervention through a browser extension.

The research on security warnings has shown that security warnings are, on the one hand, effective at reducing the rate at which users perform potentially harmful actions after they have been warned [16, 21, 22]. On the other hand, users tend to ignore such warnings due to their excessive frequency [23] and habituation effects [15]. In addition to the content, the design matters as well; a study by Akhawe and Felt [16] showed that users of one browser proceeded to potentially malicious websites twice as often as the users of the other browser, when presented with two different SSL warnings from two web browsers; a similar finding was made by Bravo-Lillo et al. [23], who showed that by changing the user interface (UI) elements in the warning to highlight the most important elements for the users, they can reduce by half the installation rate of potentially malicious apps.

When looking at what motivates users for act or ignore security warnings and advice, several studies point out that the most important factors are the perceived security/convenience trade-off and the perceived risk of pursuing potentially dangerous actions [24–26]. Yet, the risks associated with specific actions are often misunderstood by end users or even by developers and webmasters [27].

2.3 File Integrity Verification

Several works have studied, by means of online surveys, the security and usability of different fingerprint representations for authentication and integrity verification. Hsiao et al. have compared the speed and accuracy of hash verification with various textual and visual representations [5]. Their between-subjects study with 436 participants is the first to show that users struggle with comparing long fingerprints. More recently, Dechand et al. have studied the performance and usability of six textual fingerprint representations [6]. Their experiment with 1,047 participants demonstrates that the state-of-the-art hexadecimal representation is prone to partial pre-image attacks more than others, with more than 10% of attacks being missed by the users. Similarly, Tan et al. evaluate the usability and security of eight textual and visual fingerprint representations [28]. The results of their 661-participant experiments suggest that, when security is paramount, the best strategy is to remove the human from the loop and automate the verification process, which the authors did not test.

Research on secure messaging also provides us with relevant findings on the usability and security of fingerprints for authenticating the communicating entities. In their systematization of knowledge on secure messaging, Unger et al. emphasize the usability and adoption limitations of manual fingerprint verification [29]. Moreover, they mention short authentication strings, which rely on truncated cryptographic hashes, as a more usable alternative to fingerprints. In a 60-participant study on secure communication tools, Abu-Salma et al. show that fingerprints are not understood by participants, thus indirectly hindering the adoption of such tools [30]. Vaziripour et al. evaluate the usability of the authentication processes in three popular messaging applications (WhatsApp, Viber, Facebook Messenger) through a two-phase study involving 36 pairs of participants [31]. These participants notably report that fingerprint strings are too long, and some WhatsApp users appreciate being able to scan QR codes instead of having to compare long digit strings. Note that in these contexts, unlike for web downloads, automating fingerprint comparison is not possible because fingerprints usually come from a different channel. On the practical side, a number of programs (including browser extensions [32, 33]) to compute and verify checksums with graphical user interface are available. Yet, they only enable users to compute checksums, not to automatically verify them against those extracted from webpages.

In addition to checksums, digital certificates can be used to certify the authenticity and integrity of programs. However, some shortcomings of digital certificates include their cost, certificate validation issues, and private key (of developers and certification authorities) compromise [34, 35]. In fact, digital certificates (used for code-signing) do not provide the same guarantees that checksums do: Certificates guarantee that the downloaded files have been produced by certain developers, whereas checksums guarantee that the downloaded files are those the website administrators intended to point to. Therefore, checksums do not provide protection in the case where a malicious website administrator includes a link to a corrupted version of a program (e.g., Transmission). And certificates do not provide protection in the case where a hacker replaces a program file with a corrupted version of the program signed with the (valid) account of a malicious developer (or with a stolen account).

In our work, we focus on one aspect that was neglected by prior research: What is the behavior of the users when they (are asked to) verify file integrity by using checksums? Instead of testing different design of the checksum, we focus on the process by which participants compare an hexadecimal checksum with the output of the hash functions. In summary, we go beyond the sole investigation of manual fingerprint comparison, and we consider the overlooked context of web download integrity. We also employ eye-tracking techniques to gain a deeper understanding of how users perform fingerprint/checksum comparisons.

2.4 Automating Integrity Verification

In certain contexts, checksum verification is automated. It is the case with W3C's subresource integrity, described below in the background section. It is also the case of package managers such as brew (macOS) or aptitude (Linux), which enable users to download packages and programs from so-called repositories. They automatically compare the checksums of the downloaded packages to those specified in the package description: A typical brew "cask" package contains a link to an installer hosted on an external platform, a command line to run it and a checksum to verify its integrity (see that of VLC[3]). Such package managers, however, are mostly popular on UNIX systems and they are used mainly by experienced users (e.g., users familiar with the terminal).[4] Note that package managers are also subject to attacks [36].

3 SYSTEM AND THREAT MODEL

In this section, we describe the general system and threat model, as well as the necessary technical background.

3.1 System and Threat Model

We consider a website hosted on a web server. The website contains a download page that includes a link to a program hosted on an external web server (a hosting platform, typically on a mirror or a content delivery network) managed by a different entity. The original website is managed by the developers. We consider an adversary who is able to tamper with the program files hosted on the external server (e.g., the operator of the external hosting platform or a hacker) or to tamper with the insecure (e.g., no SSL/TLS) communication with the external server (e.g., the user's Internet service provider or a hacker), as illustrated in Figure 1.

Note that such a situation could also occur when the download webpage and the program are hosted on the same server but the adversary is only able to tamper with the program file (e.g., because it has access to only certain directories on the server).

In order to enable users to check the integrity of the files they download from the external server, the download page contains the checksum of the program file, which is generated as describe hereafter.

3.2 Checksums

A checksum is a fixed-size binary string derived from a block of data of arbitrary size (e.g., a file): it is used to verify the *integrity* of the data, i.e., whether the data has been tampered with after the checksum was created. In adversarial settings, the output of cryptographic hash functions, called hashes or digests, are used as checksums. Checksums are usually represented as hexadecimal strings (e.g., `2cae915ae0e...`), the sizes of which usually range from 32 to 128 digits (i.e., 128-512 bits). Cryptographic hash functions enjoy three core properties: pre-image resistance, second pre-image resistance, and collision resistance [37]. In the settings of web downloads hosted on external servers, the second property is key: It guarantees that it is computationally hard for an adversary with access to the original file (and its hash) to forge a different file (e.g., a malware) that has the *same* hash. Essentially, an adversary would have to rely on brute-force attacks, that is, to perform an exhaustive search of slightly modified versions of the file until it finds one with a hash that matches that of the original file. An adversary can also perform a brute-force attack to forge a file with

[3]https://github.com/caskroom/homebrew-cask/blob/master/Casks/vlc.rb

[4]As more and more graphical front-ends to UNIX package managers are available (e.g., Synaptic, Ubuntu Software Center), package managers do not require knowledge of the terminal anymore and become more accessible to inexperienced users, just like app stores.

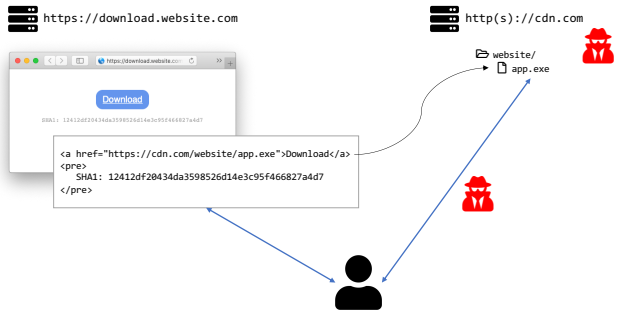


Fig. 1. System and threat model. A webpage, hosted on server `download.website.com` (assumed secure) and served over SSL/TLS, contains a download link (i.e., an HTML a element) to a file hosted on a different server (and domain) `cdn.com`. The file hosted on the external server could be corrupted and the communication between this external server and the user could be tampered with by the adversary. The download webpage contains a checksum so that users can verify the integrity of the downloaded file.

a hash that *only partially* matches that of the original file, namely partial pre-image attacks. In addition, hash functions usually ensure that even a minor change (even just one bit) in the input data results in a completely different output hash.

Popular cryptographic hash functions include MD5, SHA-1, and SHA-2. MD5 was one of the first proposed cryptographic hash functions; it was broken in the late 1990's and its use is strongly discouraged. SHA-1 was recommended by the National Institute of Standards and Technology (NIST) until 2015, when it was broken. SHA-2 is the most popular hash function today and it is currently the recommended (by NIST) algorithm for file integrity verification [38].

To verify the integrity of a file, users have to execute a dedicated program that takes the file as input and compare the output (i.e., checksum) with that specified on the download page.

3.3 Subresource Integrity

Subresource integrity (SRI) was introduced by the W3C in 2016 [7]. It specifies that, for external resources linked to a webpage through an HTML element, an `integrity` attribute containing a checksum can be added to the element.[5] This mechanism was introduced to detect corruption of externally hosted scripts. Therefore, in its current form, SRI covers only two elements: the `link` and `script`. These elements are used to include external style sheets (e.g., cascading style sheets–CSS) and scripts (e.g., JavaScript–JS) respectively. The verification of the integrity of the subresources, based on the provided checksum, is performed by the user agent, typically the web browser. SRI is currently supported by all the major browsers (except Internet Explorer). If the integrity verification of a subresource fails, it is not loaded.

It should be noted that integrity verification mechanisms have some limitations. In particular, the fact that the checksums must be updated together with the target files is a major issue, especially when the update process is manual. And failures to address this issue can create detrimental false alarm situations. These issues are discussed in more details (for SRI) in a recent study [39].

4 AUTOMATING CHECKSUM VERIFICATION

One of the main usability issue in the current form of checksum-based integrity verification is that the task of computing and verifying checksums needs to be done manually and visually by the

[5]https://www.w3.org/TR/SRI/. Last visited: Dec. 2019.

users. In addition, most Internet users are unaware of the utility and usage of checksums [1]. In this section, we address these problems by proposing both amendments to the existing standards as well as by technical solutions that we implemented.

4.1 Extending Subresource Integrity to Links

A direct solution for making checksum verifications automatic is to extend the subresource integrity (SRI) feature [7], introduced by the W3C and described in Section 3, to HTML `a` elements (i.e., links) that point to files to be downloaded.

Our proposal is to include an `integrity` attribute in the `a` elements, and optionally the `meta` and `iframe` elements, as web developers sometimes rely on them to trigger automatic downloads. Below, we give an example link that specifies in an `integrity` attribute the checksum of the file it points to.

```
<a href="https://github.com/.../file.dmg" integrity="sha256-Yc2bdMx...">download</a>
```

Upon a successful download of a file pointed to by a link that includes an `integrity` attribute, the integrity of the downloaded file should be checked by the user agent (i.e., the web browser or an extension) by comparing its (computed) checksum to the one specified in the `integrity` attribute.

A recent study by Chapuis *et al.* [39] shows that web developers have a strong interest in extending SRI to downloads (i.e., `a` elements) as well as pictures, videos, *etc.* We made a proposal in this direction and communicated it to W3C's WebAppSec Working Group. Our proposal includes other types of subresources, including images and videos. Note that such subresources have specificities that must be taken care of (e.g., progressive load of images).

4.2 Checksum Verification: Browser Extension

As browsers do not currently handle SRI for links, we developed a Chrome extension to automatically check the integrity of downloaded files.[6] This extension should be considered as a proof of concept and not as a final product.

Design and Implementation. Our extension supports three popular algorithms used to generate checksums: the MD5, SHA-1 and SHA-2 hash functions.[7] It is implemented in JS and it relies on the md5.js library for computing MD5 digests[8] and the asmcrypto.js library for computing SHA digests.[9] In total, the extension consists of ~400 lines of JavaScript code (excluding the libraries); it requires permission to access the browser's download manager in order to initiate and monitor downloads, as well as read-only access to the file system in order to compute the digest of the downloaded file.

Because SRI for links is currently not supported, the extension automatically extracts checksums directly from the text of HTML pages, thus requiring no changes to existing websites (such as VLC). It operates as follows:

(1) For each visited webpage, it navigates the HTML DOM tree and extracts, by using regular expressions, hexadecimal strings that have the same format as checksums and the corresponding hash function names (e.g., MD5).
(2) If checksums are detected (on the webpage or in the `integrity` attribute of the `a` element), it intercepts click events triggered by hyperlinks. If a link points to a file with a sensitive extension (e.g., dmg, exe) and/or multipurpose internet mail extension (MIME) type[10] (e.g.,

[6] Ideally, such a verification should be performed by the web browser.
[7] We chose to support the MD5 and SHA-1 functions despite their known weaknesses because they are still used [1].
[8] https://github.com/blueimp/JavaScript-MD5
[9] https://github.com/asmcrypto/asmcrypto.js
[10] The MIME type is determined by issuing a HEAD request to the target.

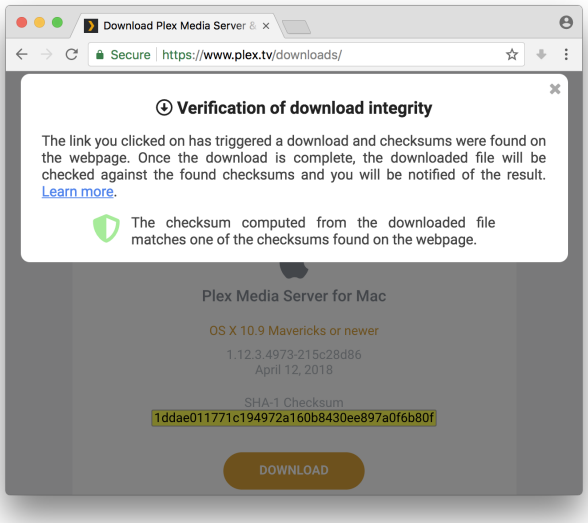


Fig. 2. Screenshot of the extension on the Plex download page. The checksum of the downloaded file is computed and successfully checked against that extracted from the webpage (highlighted). See Fig. 12 (p. 19) for the French version of the messages used in the experiment.

application/x-apple-diskimage, application/x-msdownload), the download is followed by the verification of the checksum, essentially a comparison between the checksum that is detected and the one computed from the downloaded file.

(3) If multiple checksums are extracted from the webpage, the verification is considered successful as long as the computed checksum matches *any* one of them.[11] The webpage is greyed out and a pop-up message is displayed to the user, as illustrated in Figure 2. Additionally, if the checksum originates from the text of the webpage, the matching text with the checksum is revealed (if originally hidden) and highlighted.

The extension displays a general message to the user and a status indicator (e.g., "downloading", "computing checksum") with an animation. Additionally, it can show four different messages according to the result of the verification (Figure 3), depending on the origin of the checksum (webpage text or integrity attribute) and on the outcome of the verification (success or failure). In the case of failure, users are offered the option to delete the possibly corrupted downloaded file (through a link). Clearly, there are multiple ways to communicate the result of the verification to the user, and the UI elements have a significant effect on the usability of our extension [16]. For the initial proof of concept, we experimented with the four messages shown in Figure 3. A careful consideration of alternatives that incorporate user feedback should be conducted before a public release of such an extension. We leave the careful design of the extension UI for future work.

An archive containing the source code of the extension used in the experiment can be downloaded at the following address: https://checksum-lab.github.io/chrome_extension.zip (SHA-256: `237ac0154e5d951d22f54c97300d3de81a88333c29ec66334c061edb44f2d368`).[12] A test webpage

[11]Note that this reduces only slightly the security of the verification procedure as download pages usually contains only a few checksums (8 at most in the websites we surveyed in [1], i.e., for Android Studio). As part of future work, we intend to match automatically checksums to download links by analyzing the DOM of the webpages.

[12]An updated version is available at: https://github.com/isplab-unil/download-checksum; alternatively, it can be installed from the Chrome Web Store: https://chrome.google.com/webstore/detail/automated-checksum-verifi/kabghagbpkdbojdeklmcbfamenmpilga

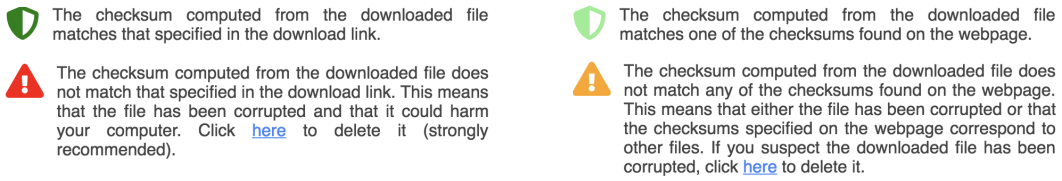


Fig. 3. Messages displayed by the browser extension: left (`integrity` attribute) / right (text of the webpage), top (success) / bottom (failure). See Fig. 12 (p. 19) for the French version of the messages used in the experiment.

can also be found at the following address: https://checksum-lab.github.io/. It contains test download links with and without (correct/incorrect) integrity attributes and links to the download pages (that include checksums) of popular software (e.g., Android Studio, Plex, VLC) on which the extension can be successfully tested. Alternatively, a demo video can be downloaded or watched at the following address: https://checksum-lab.github.io/demo.mp4.

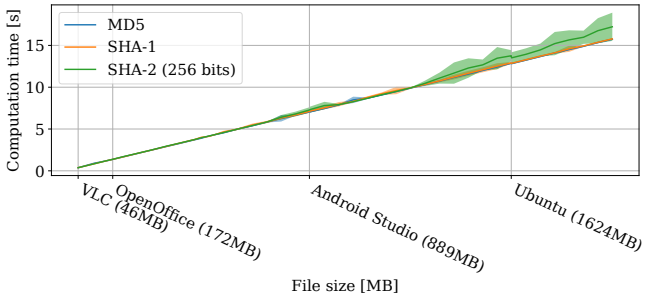


Fig. 4. Performance of the browser extension in terms of the checksum computation time for different hash functions and file sizes. The graph shows the mean and the standard deviation for 20 independent runs.

Performance Evaluation. In order to assess the delays induced by the verification of the checksums, we measured the computation times for different hash functions (namely MD5, SHA-1 and SHA-2 with 256 bits), based on the implementation of the libraries used in the Chrome extension, and for different file sizes ranging from 45MB (corresponding to VLC's app file) to 1.6GB (corresponding to Ubuntu's ISO image file). For each hash function and file size, we performed 20 independent runs and we measured the mean and the standard deviation of the computation time. The results were obtained in a standard setting (MacBook Pro 2014, SSD, 16GB of RAM, Core i7@2.2GHz, macOS 10.12.6, Chrome v.65 64-bit). They are shown in Figure 4. It can be observed that the computation time is reasonable. It takes less than one second to verify the checksum for small files (<50MB), and only about ten seconds for large files (~1GB). Note that we also compared the computation times for the extension against those for native programs (e.g., `shasum`) and find them to be comparable. Unsurprisingly, we find that the computation time grows linearly with the file size. The corresponding rates, obtained through a linear regression, are 121MB/s (MD5), 120MB/s (SHA-1), and 115MB/s (SHA-2 with 256bits). The verification throughput is much higher than those of most broadband connections; the verification time is therefore negligible compared to the download time for most users. To improve the performance, one can combine any of the following techniques: optimizing the library, optimizing the browser's JavaScript engine, using native libraries (e.g., `SubtleCrypto`), computing the checksum as the file is downloaded (i.e., pipelining).

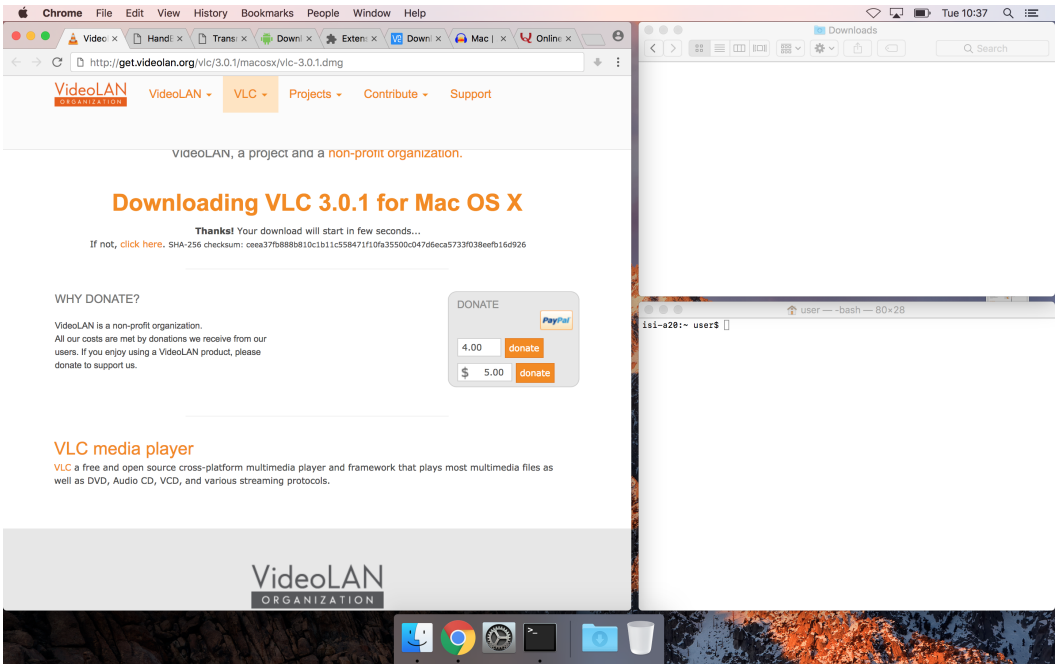


Fig. 5. Screenshot of the window arrangement on the computer used for the experiment (macOS). The left half of the screen is occupied by the Chrome browser in which multiple tabs have been opened: the download pages of the first four programs, the extension tab to activate the extension, the download pages of the next two programs, and the questionnaire website for the exit survey. The right half of the screen is occupied by the terminal application where the participants must type the command lines to compute the checksums of the downloaded programs (bottom) and the "Downloads" folder (top) were the programs downloaded from the browser are placed; the participants had to click on the icons of the downloaded programs (in that window) to execute them.

Shortcomings and Perspectives. There are several limitations and missing features that we intend to address in the future. First, the UI and the textual messages of the browser extension should be carefully designed by taking into account user feedback (see Section 5.4) and best practices for the design of security warnings (see for instance [14–18, 21, 23, 40–42]). Second, the extension does not handle the case of concurrent downloads *from the same tab* (e.g., multiple downloads from the same webpage). Third, the extension works only when the checksum and the direct link to the file are on the same page; for instance, the case where a download link redirects to a page with an automatic download based on a `meta` or `iframe` element is not supported. Similarly, it does not support the case where the checksums are in a separate file (e.g., .md5, .shasum, .sig) linked on the download page.

5 CONTROLLED USER EXPERIMENT

To better understand how Internet users handle file integrity verification, we conduct an in-situ experiment with 40 participants and an eye-tracking system. More specifically, we aim at answering the following research questions:

- (RQ1) *Do users thoroughly verify checksums and how do they proceed?*

- (RQ2) *Can users be fooled by replacing characters in the middle of the checksum (i.e., partial preimage attack)?*
- (RQ3) *Does automating the checksum verification improve general usability metrics?*

Eye-tracking has been used extensively in the last decade to study usability of new services, programs or mobile apps, as it enables the collection of accurate objective measurements of where the user looks on the screen without obtruding or disturbing their action [43]. The two metrics we extract from this experiment are the total number of *fixations* and the *total dwell time*. Fixations are indicative of the amount of processing being applied to objects at the point-of-regard [44]. A longer dwell time indicates difficulty in extracting information, or it means that the object is more engaging in some way [45]. Our hypotheses were that some participants would not thoroughly check the checksums (fixating only parts of them) and that participants who checked thoroughly the checksums would have to produce more fixations (and spend more time fixating) in the part of the user interface where these sequences were displayed.

The experiment was split in two phases, as detailed in Section 5.3. During the first phase, we asked participants to verify manually the checksums of four downloaded apps (this was addressing RQ1 and RQ2). In the second part of the experiment, we activated a browser extension that verifies the integrity of the downloaded files based on their checksums (this was addressing RQ3). We chose not to randomize the presentation order of these two parts as we considered that seeing the messages of the browser extension could have revealed the main topic of the experiment. With hindsight, we realize that this design also has drawbacks that we report below in Section 5.4.4. The experiment was approved by our institution's ethics committee.

5.1 Participants

We recruited the participants of our experiment from a student population through flyers displayed on two university campuses (i.e., UNIL and EPFL in Lausanne, Switzerland). To sign up for the experiment, potential subjects had to fill an online screening questionnaire first. In this questionnaire, they were asked about their basic demographic information (age and gender), major field of study, knowledge of checksums (i.e., "Do you know what the elements circled in red are used for and how?[13] If yes, please describe it briefly in the text box below."), technology savviness (i.e., "Check the technical terms related to computers that you understand well: ad-blocker, digest, firewall, VPN, etc.). Finally, we asked which was the operating system of their main computer.

We selected a total of 40 subjects (out of the 120 who completed the screener) and invited them to participate in the experiment. The number of participant was chosen so that it provides sufficient power to the statistical tests and keeps the total duration of the experimentation reasonable (we had only one eye-tracker). The sample was selected to maximize diversity. About half of the participants were macOS users (i.e., 21/40, that is 53%) and half Windows users (the actual breakdown in terms of operating systems (OS) among the participants who filled the screener was 56% macOS, 41% Windows, 3% Linux). The subject pool included 40% of female subjects and it was diverse in terms of major fields of studies, with more than 15 different majors represented. The average age of the subjects was 22.5±2.9. Out of the 40 subjects, 12 (30%) knew about checksums, 33 (83%) downloaded programs from developers websites and 20 (50%) from app stores, and 25 (63%) had an antivirus installed on their computers. The experiment took approximately 50 minutes per person to complete and the participants were compensated with CHF 20 (~USD 20). The whole experiment was conducted in French (i.e., the local language in Lausanne).

[13]The screenshot depicted VLC's download page with checksums circled.



(a) Thorough verification of a correct checksum

(b) Succeeded verification of an incorrect checksum (i.e., mismatch detected)

(c) Failed verification of an incorrect checksum (i.e., mismatch not detected)

Fig. 6. Sample subject gaze heat maps captured by the eye-tracking system on macOS.

5.2 Apparatus

The experiment took place in a UX-lab, a small room with a desktop computer. The computer was equipped with an eye-tracking system (maker Tobii, model X2-60[14]) which was sampling gaze at 60Hz. Two cameras and a few microphones were also placed in the room to record the experiment.

Depending on the OS the participant was most familiar with (macOS or Windows), we switched the computer that was used by the participants during the course of the experiment. Aside from the OS, the employed apps and the layout of the windows were the same on the two different OSes. Three windows were placed and arranged on the screen: the web browser (Chrome) that occupied the left half of the screen, the "Downloads" folder (Windows explorer/macOS finder) that occupied the top right quadrant, and the terminal that occupied the bottom right quadrant (see Figure 5). Participants were asked to not change the position of the three windows, and scrolling was disabled in the browser in order to reduce shifts in the areas of interest (AOIs) of the screen that were displaying the checksums.

All necessary pages were pre-loaded in the browser window in different tabs. We tampered with the checksum on the third webpage (i.e., Transmission) for the first part of the study and the second webpage (i.e., Audacity) for the second part of the study. All the other checksums were correct. Based on our running hypothesis that users check only the first and last digits of the checksum, we changed the 44 digits (out of 64) in the middle of the checksums; this means that only the first and last 10 digits remained unchanged. This corresponds to a 80-bit attack (i.e., 20 hexadecimal digits). We assumed, as in [6], that a realistic adversary can forge, through brute-force, a corrupted program in such a way that the first and last few digits of its checksum match those of the original program's checksum. In [6], the authors estimate the cost of such an attack to be between USD 610k and USD 16B. Note that recent advances[15] for computing hashes (e.g, GPU-based) and further optimizations (e.g., exploiting the visual similarity between digits) could be used to further decrease the cost of the attack, not to mention the decrease in computation costs. Note also that, as our results show, keeping the last digits unchanged is in fact not very important as most users focus their attention on the first digits (see Section 5.4); therefore, an inexpensive 40-bit attack could probably achieve the same results.

5.3 Procedure

First and foremost, we informed the participants that they would be recorded during the course of the experiment (and about our data management plan, including data anonymization and retention) and we asked them to sign, if they agreed, an informed consent agreement. We told the participants

[14]https://www.tobiipro.com/product-listing/tobii-pro-x2-60/

[15]The work in [6] was conducted more than four years ago.

that we were conducting a study on the way people download applications on their computers and that they had to download several applications on the lab computer. We asked the participants to behave as if they were using their own computer and we told them to not hesitate to call the experimenter in case of doubts or problems. We also explained that the experimenter had nothing to do with the design and implementation of the extension, therefore, the participants could freely express negative opinions without the risk of affecting the experimenter.

Next, we asked participants several preliminary questions, mainly to confirm some of the information they provided in the screener: the OS of their computer, whether they had an antivirus installed and whether they downloaded apps from the Internet from places other than official app stores. Then, we asked the participants to sit at the computer, and a 13-point calibration procedure for the eye-tracking system was completed. Finally, the participants were given a checklist containing the steps to follow during the session.

First Phase. We asked the participants to download from the official website and execute/install four different programs (in this specific order): VLC, Handbrake, Transmission, and Android Studio. Specifically, for each application, the participants were asked to

(1) Download the application. For the sake of simplicity, the download webpages were already opened in individual tabs of the web browser.
(2) Compute the checksum of the downloaded program and compare it to that specified on the webpage. The participants were provided with the exact command to type in the terminal, e.g., `clear ; shasum -a 256 Handbrake-1.1.0.dmg` for macOS.[16] All the checksums were SHA-2 with 256 bits.
(3) Run the program and report some information on the instruction leaflet: program version and copyright years found in the "About" box (macOS) or digital certificate issuer (Windows). The purpose of this last step was to avoid calling too much attention to the checksum verification as being the core of the experiment.

Second Phase. We asked the participants to activate the extension (by clicking on a button in the fifth tab of the browser), and to download and run/install two additional applications i.e., RealVNC and Audacity, in this order. We asked the participants to perform the same steps as in the first phase, except from the manual checksum verification that was automated by our browser extension. The first application's checksum was correct, resulting in the display of a confirmation message by the browser extension, whereas the second one was incorrect, hence resulting in the display of a warning message (see at the top and bottom right of Figure 3 resp.). The terminology used in the messages was inspired by the instructions found on the download pages of popular programs (e.g., Ubuntu).

Finally, we asked the participants to fill a short online questionnaire to get feedback about their perception of the manual verification of checksums and of the browser extension, satisfaction with the extension and net promoter score[17].

5.4 Results

We describe and analyze the results related to the manual checksum verification (first phase) and report on the usability and effectiveness of the browser extension (second phase).

[16]The `clear` command is used to ensure that the checksum is always displayed at the same location on the screen, for eye-tracking purposes.
[17]See https://en.wikipedia.org/wiki/Net_Promoter, last accessed: Dec. 2019.

In order to study the gaze behavior, in our analysis, we surrounded the parts of the UI that displayed the checksums, and we labelled each area of interest (AOI). Unfortunately, we had to remove eye-tracking recording for one participant due to corrupted data.

5.4.1 RQ1. From a qualitative analysis of the fixation heatmaps of the participants looking at the AOIs that contained the checksums, we could observe three distinct behaviors: (a) some participants produced extensive fixations throughout the sequence of characters (i.e., the checksum) covering most/all of the sequence; (b) other participants produced less fixations but still "sampled" the sequence at several points from beginning to end; (c) finally some other participants produced fewer fixations in the AOIs, typically pointing to the beginning and the end of the sequence. Examples of these three behaviors can be seen in Figure 6. While the first two behaviors typically led to identifying the incorrect checksum, the third was typically associated with *not* identifying the incorrect checksums. This was confirmed by our quantitative analysis presented below.

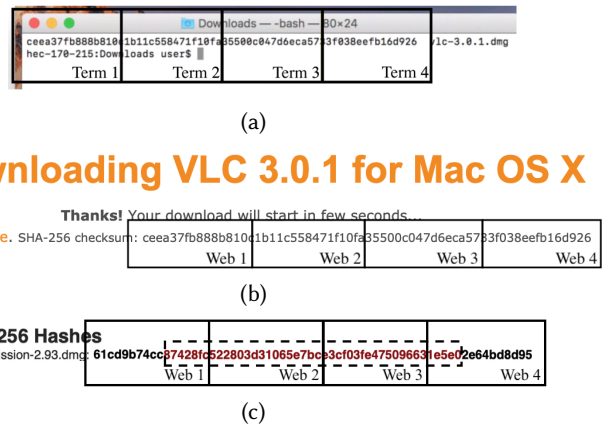


Fig. 7. Areas of interest (AOIs) used for the checksums (a) displayed in the terminal and (b)/(c) on the webpage. For Transmission (c), the mismatch spans from the end of sub-AOI 1 to the beginning of AOI 4 (dashed box).

To understand whether all the digits of the checksum were treated equally by the participants, we further subdivided the area where the checksum is displayed in four sub-AOIs, both in the terminal and in the webpage (see Figure 7), and measured differences of the total number of fixations falling in each of these areas. As the assumptions for parametric inferential statistics were violated, we used nonparametric statistics for the subsequent quantitative analysis.[18]

We conducted a Friedman test of differences among repeated measures to compare the total number of fixations that fell in each of the four sub-AOIs of the checksum displayed in the terminal. There was a significant difference in the scores: Term 1 - M=25.15, SD= 13.11, Term 2 - M=21.92, SD= 13.96, Term 3 - M=13.92, SD= 9.55, and Term 4 - M=10.58, SD= 6.99; $\chi^2(3) = 77.32$, $p < 0.001$. Six Wilcoxon signed rank tests with continuity correction were conducted to make post-hoc comparisons between AOIs. All the tests indicated that there was a significant difference between the number of fixations falling in each terminal AOI. We include the detailed results of the tests and

[18]Concerning the total number of fixations, the Shapiro-Wilk normality tests were close to rejection: Term 1 - ($W = .95$, $p = 0.085$), Term 2 - ($W = .94$, $p = 0.027$), Term 3 - ($W = .94$, $p = 0.037$), Term 4 - ($W = .92$, $p = 0.008$) and the assumption of homoscedasticity was violated when using the Modified Levene's Test ($F = 6.23$, $p < 0.001$). The conclusion was similar for the total dwell time.

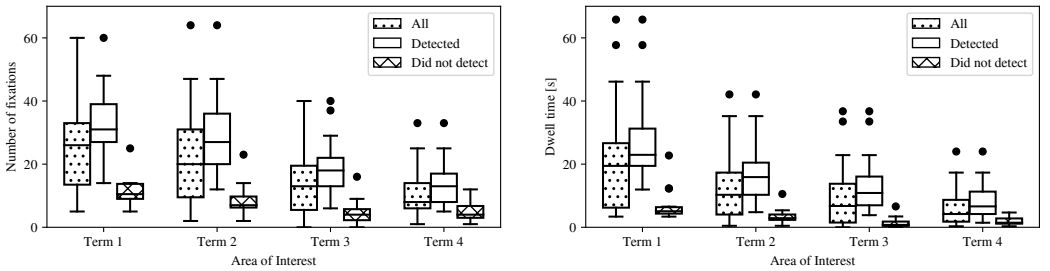
AOIs (Term.)	1	2	3	4
1	–	445^{**}	756^{***}	773^{***}
2	–	–	709^{***}	688^{***}
3	–	–	–	518^{***}
4	–	–	–	–

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$

Table 1. Wilcoxon signed rank tests of the number of fixations within the four AOIs covering the checksums in the terminal. Due to ex aequos in the data, the p-value is an approximation.

the boxplot of the distribution of fixations for each terminal AOI, in Figure 8 and Table 1. These results suggest that the attention given to the digits of the checksum is highest at the beginning and decreases as we progress in the sequence. This means that a partial pre-image attack should focus on keeping the first digits of the checksum unchanged.

5.4.2 RQ2. We observed that 15 (38%) of the participants did *not* detect the mismatch (for Transmission) between the checksum displayed on the download webpage and the checksum computed from the downloaded file (displayed in the terminal). This constitutes a substantial proportion of our subject pool. This number could be higher in real life as the subjects are likely to be more careful in a controlled environment compared to a situation where they are eager to run the program they just downloaded. Furthermore, we explicitly asked the subjects (in the instructions) to verify if the checksums on the webpage and in the terminal were identical. We did not find a significant difference in the detection rate for participants who had prior checksum knowledge ($p = 1$, Fisher's exact test). We hypothesize that participants with prior knowledge understand better the importance and functioning of checksums but, at the same time, they might be more sloppy in their verification as they know that an accidental modification would very likely change the first digits of the checksum. The same result was observed for the previous results on RQ1.



(a) Number of fixation in each sub-AOI of the terminal. (b) Dwell time in each sub-AOI of the terminal.

Fig. 8. Boxplot-representations of the distributions of the participants' (a) number of fixations and (b) total dwell time across the four sub-AOIs covering the checksums of the terminal in the four verification tasks. The distributions are displayed across all the participants (left), across participants who detected the mismatch and stopped (middle), and across participants who did not detect the mismatch and continued (right).

↪	Term 1	Term 2	Term 3	Term 4	Web 1	Web 2	Web 3	Web 4
Term 1	0	1.93	0.13	0.17	2.52	0.48	0.07	0.03
Term 2	1.92	0	1.47	0.19	0.59	1.6	0.35	0.01
Term 3	0.28	0.81	0	1.09	0.15	0.41	1.69	0.47
Term 4	0.43	0.17	0.59	0	0.05	0.08	0.35	1.32
Web 1	1.84	0.68	0.07	0.04	0	1.17	0.03	0.11
Web 2	0.33	1.75	0.35	0.03	0.47	0	1.21	0.15
Web 3	0	0.61	1.57	0.16	0.05	0.49	0	1.24
Web 4	0.09	0.13	0.76	1.47	0.07	0.05	0.48	0

(a) Correct / Detected

↪	Term 1	Term 2	Term 3	Term 4	Web 1	Web 2	Web 3	Web 4
Term 1	0	1.12	0.31	0.12	0.74	0.24	0.07	0.1
Term 2	1.29	0	0.4	0.14	0.12	0.07	0.02	0.02
Term 3	0.21	0.36	0	0.5	0.02	0.02	0.02	0.1
Term 4	0.14	0.19	0.17	0	0.05	0.05	0.19	0.21
Web 1	0.55	0.19	0.05	0.02	0	0.14	0.1	0.14
Web 2	0.07	0.1	0.02	0.02	0.17	0	0.17	0.07
Web 3	0.02	0.05	0.07	0.05	0.05	0.17	0	0.33
Web 4	0	0.12	0.19	0.33	0.05	0.02	0.17	0

(b) Correct / Did not detect

↪	Term 1	Term 2	Term 3	Term 4	Web 1	Web 2	Web 3	Web 4
Term 1	0	4.2	0.6	0.72	6.24	0.88	0.08	0.2
Term 2	5.92	0	1.32	0.52	0.88	0.4	0.12	0.04
Term 3	0.52	0.92	0	1.28	0.16	0.08	0.16	0.28
Term 4	0.96	0.52	0.6	0	0.2	0.04	0.24	1
Web 1	4.36	2	0.28	0.16	0	2.68	0.44	0.6
Web 2	0.64	0.92	0.16	0	2.04	0	0.4	0.36
Web 3	0.08	0.12	0.12	0.04	0.52	0.36	0	0.64
Web 4	0.28	0.36	0.32	0.88	0.6	0.12	0.56	0

(c) Incorrect / Detected

↪	Term 1	Term 2	Term 3	Term 4	Web 1	Web 2	Web 3	Web 4
Term 1	0	0.64	0.07	0.36	0.86	0	0.07	0
Term 2	0.86	0	0.36	0.14	0	0.07	0.07	0
Term 3	0.07	0.29	0	0.36	0.07	0.07	0	0
Term 4	0.07	0	0	0	0	0	0.29	0.64
Web 1	0.5	0.14	0.21	0	0	0.21	0.07	0.21
Web 2	0.07	0.07	0.07	0	0.21	0	0.07	0
Web 3	0	0.07	0	0	0.07	0.07	0	0.21
Web 4	0.07	0.14	0.14	0.21	0.14	0.14	0	0

(d) Incorrect / Did not detect

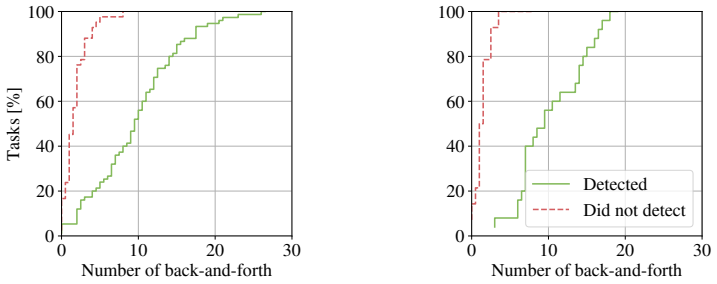
Fig. 9. Transition matrices representing the average number of transitions from one AOI to another. The row represents the origin of the transition and the column the destination. The names of the AOIs are those from Fig. 7. The top matrices correspond to the tasks with a correct checksum (i.e., VLC, Handbrake, Android Studio) and the bottom matrices correspond to those with an incorrect checksum (i.e., Transmission). The left matrices correspond to the participants who detected the mismatch for Transmission and the right matrices correspond to those who did not. The darker a cell, the higher the number of transitions.

To study more quantitatively if some behavioral differences existed between those who detected the mismatch and those who did not, we operated a post-hoc split of the participants. We focused our analysis on the terminal window. Figure 8 shows the distribution of the number of fixations and dwell time, for each of the four sub-AOIs in the terminal, across the participants who detected the mismatch and those who did not. A Wilcoxon rank sum test was conducted to compare the total number of fixations in the AOIs for the two groups of participants. The values of the task with the incorrect checksum were not considered in order to compare the usual behavior. There was a significant difference in the number of fixations for participants who detected the corrupted checksum (M=12.47 fixations, SD=5.01) and those who did not (M=3.88 fixations, SD=2.09); W=338.5, $p < 0.001$. Furthermore, the same test was conducted to compare total dwell time in the AOIs for the two groups. There was a significant difference in the amount of time spent in the checksum AOIs for participants who detected the corrupted checksum (M=15.63 seconds, SD=9.50) and those who did not (M=3.97 seconds, SD=2.60); W=333, $p < 0.001$.

These results suggest that participants who detected the corrupted checksum fixated the checksums significantly more frequently and spent significantly more time than those who did not. The observed ratios between the two behaviors were approximately 4:1. This analysis was also extended to tasks 1, 2 and 4 for the two groups of participants. We observed the same difference as for Task 3; this reveals that those who were thorough were consistently so, during the entire experiment.

To better understand how users conduct the checksum verification process, we extracted and analyzed the gaze movements between the sub-AOIs of both the checksum displayed in the terminal and the checksum displayed in the webpage; as the verification process consists in making sure that these two alphanumeric strings are identical, the participant had to look alternatively at the checksum in the terminal and at the checksum in the webpage. This is due to the fact that people

can only hold so much information in their working memory. To perform this analysis, we relied on the same sub-AOIs as before (see Figure 7) and computed the number of transitions, with respect to participants fixations. We define as a transition one or multiple fixations in one of the AOIs followed by one or multiple fixations in a different AOI. All fixations outside of the AOIs were ignored: If a participant fixates in Term 1, then somewhere else on the screen, and finally in Web 1, this counts as a transition from Term 1 to Web 1.



(a) Correct checksum (b) Incorrect checksum.

Fig. 10. Cumulative distribution functions of the number of back-and-forth transitions between the terminal and the webpage.

We look at the transitions between the different AOIs. Figure 9 depicts the matrices of transition between AOIs for the tasks with a correct/incorrect checksums (top/bottom) and for the participants who detected/did not detect (left/right) the mismatch for Transmission. A first general finding is that participants start by looking at a chunk of the checksum in one window (terminal or webpage) and then check by looking at the corresponding chunk in the other window (diagonal in the top-right and bottom-left quadrants of the transition matrices, e.g., "Term $i \leftrightarrow$ Web i" transitions). Note, however, that some participants look at multiple chunks of the checksum successively in the same window (sub-diagonal in the top-left and bottom-right quadrants, e.g., "Term $i \rightarrow$ Term $i+1$" transitions). It can also be observed that the participants who did not detect the mismatch stopped the verification process in the first parts of the checksums; this confirms our previous analysis. In the case where the checksum is incorrect (right sub-figures, i.e., Transmission), the behavior of the participants who did not detect the mismatch does not change substantially; there is no substantial difference between Figure 9b and Figure 9d. For the participants who did detect the mismatch, however, the difference is substantial: Indeed the participants' fixations gravitate around the first two AOIs (the mismatch started at the end of AOI 1) with many transitions between these two.

We further look at the distribution of the number of back-and-forth transitions (i.e., transitions from one window to another and back to the original window) between the AOIs in the terminal and those in the webpage across participants, as depicted in Figure 10 (cumulative distribution function). This metric reflects the cognitive load of the participants. It can be observed that the number of back-and-forth transitions is substantial, with a median number of around 10 and a maximum of 26 for the participants who thoroughly checked the checksums (i.e., those who detected the mismatch for Transmission); this number is substantially lower for those who did not. Therefore we can observe that identifying the mistake required more effort (and time). While participants who successfully identified the mismatch were thorough in checking the entire sequence of characters and numbers, those who did not identify the mismatch stopped right after the first few characters, perhaps thinking that if the beginning of the sequence matched so must the rest of it.

5.4.3 RQ3. We now report the results of our user experiment related to the browser extension carried out in the second phase. As explained in Section 5, in order to study user reaction to the messages displayed by the extension and to collect user feedback, in the second phase of our user experiment, we asked the subjects to activate the extension and to download two programs (RealVNC and Audacity) from the corresponding official websites. The checksum of the second download (Audacity) was incorrect.

During the experiment, 40% of the participants stopped when shown the warning message for the (corrupted) Audacity download. For those who did not, the reason they reported most frequently (in the exit survey) was that they tend to ignore popups shown on webpage systematically because they are too frequent and often irrelevant or even scams. Among the participants who did stop, 50% removed the download file: 37.5% of them clicked on the dedicated "delete" link embedded in the warning message and the remaining 62.5% manually removed the file.

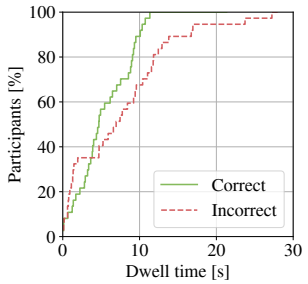
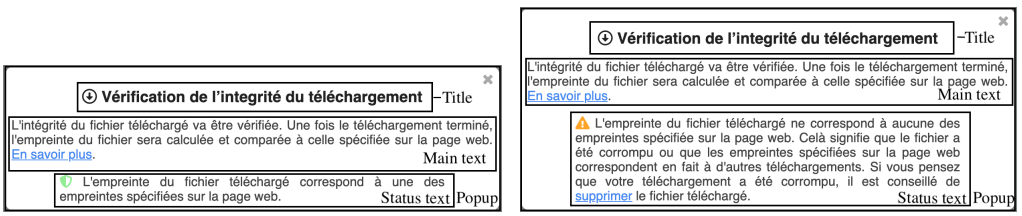


Fig. 11. Cumulative distribution function of the participants' total dwell time on the popup window of the browser extension.

In order to further analyze the participants' interaction with the popup window of the browser extension, we measured the participants' total dwell time on the popup; the cumulative distribution function across the participants is depicted in Figure 11. As expected, the median dwell time is higher for the incorrect checksum than for the correct ones. This could be explained by the fact that the participants tend to devote more time/attention to warnings (identified in many system-conventions with the orange warning icon). Surprisingly, in some cases the dwell time is lower for the warning (i.e., for the incorrect checksum); this could be explained by an habituation effect, as the incorrect checksum was always shown after the correct one in our experiment (as described in Section 5.3).



(a) Correct checksum (b) Incorrect checksum

Fig. 12. Screenshot of the extension popup when the verification succeeds (i.e., correct checksum; top) and fails (i.e., incorrect checksum; bottom). The experiment was conducted in French; an English version is available in Fig. 3.

We further defined sub-AOIs in the extension popup window (see Figure 12) and we measured the breakdown of the dwell time across them. The boxplot representations of the distributions of dwell time are depicted in Figure 13. It is interesting to notice that participants did spend more time on the status text, particularly when the error message was displayed. This indicates that the design was effective in capturing the participants' attention on the component that offered informative content to understand the status and behavior of the plugin.

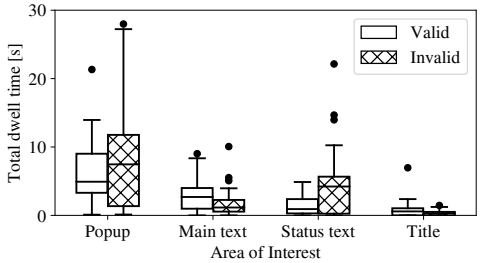


Fig. 13. Boxplot representation of the distribution of dwell time on the different AOIs on the extension popup.

In the exit survey, the participants reported an average satisfaction score of 5.2±1.4 (on a scale from 1 to 7).[19] Furthermore, the participants reported an average desirability score of 4.6±1.9 ("Should the extension be available for download, how likely would you be to use it?"), with 55% of the participants answering positively, and an average net promoter score of 4.5±1.9 ("How likely would you be to recommend it to a friend or relative?"), with 55% of the participants answering positively. In these questions, the comparison was implicit to the *status-quo* offered by the command-line interface that the participants tested in the first phase.

Another observation from the user experiment was that 26/40 participants (65%) could not explain the goal of integrity verification in the exit questionnaire (before the debriefing). This reveals the inability of non-technical users to grasp the concept behind checksum-based integrity verification. This was confirmed by the following remark made by one participant: "Sur mon ordinateur personnel, j'aurai quand meme téléchargé le fichier car l'antivirus de l'ordinateur ne m'a prévenu d'aucune menace et le site web à partir du quel j'ai téléchargé le fichier me semblait fiable (On my personal computer, I would have downloaded the file anyway as the antivirus on the computer did not notify me about any threat and the website from which I downloaded the file seemed trustworthy).". This remark also highlights a clear misunderstanding regarding the location of a website's subresources.

Finally, the participants gave us feedback on the messages displayed by the browser extension. The main comments were the following: The terminology used in the message was too technical or unclear (7 participants): "*Plutôt sobre je trouve bien mais pour un neophyte, il n'est pas très clair par rapport à son rôle. (It is rather sober I think but for a newbie it is not clear enough in relation to its role)*"; the popup did not sufficiently catch their attention (4 participants)–they suggested using larger icons and using colors for the text messages themselves or even to remove the icons–: "*Sans le petit logo vert, qui fait penser à celui d'un antivirus, c'est personnellement le genre de message auquel je fais très rarement attention. (The little green logo, which makes me think about an antivirus, should be removed as it is the kind of message that I would rarely pay attention to.)*"; the design of the skip button allowed participants to easily skip it (2 participants): "*Pour éviter que le message ne soit fermé tout de suite, il faudrait peut-être bloquer le reste de la navigation tant que le message n'est pas fermé.*

[19]For all the self-reported scores, we used a 7-level Likert scale.

Ou le laisser ouvert obligatoirement pendant quelques secondes. (To prevent the user from immediately dismissing the message the message, it would be necessary to block the user from pursuing navigation until the message is closed. Or to force the message to remain open for a few seconds).". Interestingly, during the informal feedback with the experimenter, several participants reported that they are, in general, annoyed by popups displayed within webpages and tend to ignore them.[20] Also, they mentioned that a warning originating directly from the browser in a standalone window would have been more effective. Finally, one participant reported that "[L'avertissement] est clair et bien expliqué, peut-être qu'un message plus 'effrayant' inciterait plus l'utilisateur à supprimer le fichier ([The warning] is clear and well explained, maybe a more 'frightening' message would push the user more to delete the file)".

During the experiment, we also received positive feedback on the extension. Several participants commented positively that the design of the message and the terms used were clear: "*Le message est assez clair et explique bien pourquoi le fichier devrait être supprimé (The message is rather clear and it explains well why the file has to be deleted)*", "*Ce message apparaît de manière assez claire dans la page, donc cela permet à l'utilisateur d'être au courant sur ce qu'il télécharge. (This message appears in a clear way on the page. This allows the user to be aware of what she is downloading.)*". Interestingly, one participant stated: "[L'avertissement] est également assez clair, j'y aurais fait attention hors du cadre de l'expérience. ([The warning] is rather clear, I would have paid attention to it outside of the context of the experiment)". This suggests that the browser extension would be useful in practice. The study helped identify several areas for improvement of the design, namely around the behavior of the extension and the messages displayed to encourage the users to delete the downloaded file in case of mismatch. We took some the aforementioned comments into account and refined the browser extension for the follow-up experiment described in Section 6.

5.4.4 Limitations. Like any lab study, the experiment suffered from low ecological validity. Also, the prescriptiveness of the sequence of tasks that we gave to participants reduced the ability to observe participants' spontaneous behavior when downloading files. Furthermore, we might have introduced a learning bias by choosing not to randomize the presentation of the first and the second part of the study, and the correct vs. incorrect checksums within each part. Finally, the participants of the lab study were all university students and many were technically literate as reported in Sec. 5.1. Hence, we might expect a smaller share of users to understand and use checksums in the general population than the share identified in the presented results.

5.4.5 Data availability. The eye-tracking data is available online.[21] The dataset is a 40GB Tobii Pro Studio (version 3.4.8) archive in the .nas format. The archive includes all the screen recordings; the sound and the webcam streams are not included for privacy reasons. The archive is shipped with a spreadsheet that contains the anonymized responses to the screener and exit questionnaires and the notes taken by the experimenter. The IDs in the spreadsheet corresponds to the participant number and recording number in the Tobii dataset.

6 USER EXPERIMENT IN THE WILD

To complement the insights of the in-situ experiment with data from a real-world deployment and therefore to increase the ecological validity of our study, we conducted a second user experiment. The goals of this experiment were:

[20] Showing fake (security) warnings in webpages to push users to download and install malicious programs is a common practice, e.g, fake antivirus.
[21] https://drive.switch.ch/index.php/s/hC4ayTNXqmPZptS

- To estimate the number and types (e.g., Microsoft Office or PDF document, image, binary executable) of files Internet users usually download on the Web.
- To quantify the number of websites that regular Internet users usually visit, which offer checksum-based integrity verification for the downloads.
- To collect data on how people would normally react to checksum verification in the wild.
- To observe users' responses to the browser extension that we presented in the lab study.

Therefore, we posed the following research questions:

- (RQ4) *How often do users download files from the Web and what types of files do they download?*
- (RQ5) *What is the prevalence and the current practices of checksums included in download webpages?*
- (RQ6) *What do Internet users most frequently do when they encounter a visible checksum?*
- (RQ7) *Would users feel more secure if they could use a system that automates parts of the verification process?*

To answer these questions, we conducted a medium-scale analysis of web users browsing and download habits and of the associated security-related behaviors.

6.1 Methodology

In order to capture data on how users behaved when facing a download with checksum information and how they reacted to the extension warnings, we followed a refined Experience Sampling Method (rESM) [46]. Experience Sampling involves asking participants to report on their experiences at specific points throughout the day. The method is regularly applied in studies of Human-Computer Interaction [47–49]. A typical drawback of the method is that it could be considered invasive by participants if they are sampled at random times. This is why, in recent years, researchers have proposed to refine the method by modeling the participants' context [46, 50]. The goal of these questionnaires was to: (a) collect data on whether people noticed checksums on webpages; (b) whether they understood how to use the checksums (i.e., how checksums work); (c) whether they were going to compute and verify the checksums or take other security precautions such as scanning the file for viruses; (d) record self-reported measure of security of their system. The questions of the rESM are available (in French) in Figure 14.

These questionnaires were presented to the participants of the study only if any one of the following criteria were met: (a) the participant triggered a download from a webpage that does not contain a checksum; (b) the automated verification of a checksum succeeded; (c) the automated verification of a checksum failed; (d) the participant encountered a checksum but did not download any file from the webpage. In the situation (a), (b) or (c), the rESM questionnaire was displayed immediately after the completion of either the download or the verification. A fixed delay of 2 seconds was added after the page was loaded (i.e., JavaScript `load` event), before presenting the questionnaire following trigger (d). This means that the checksum was visible on the page for a few seconds before the questionnaire was shown. In order not to overload the participants, we also established that the mini questionnaires should not be triggered more than once per day on the same participant, per type of event.

Given that the extension was designed to alter the natural online browsing behavior by making users more careful with regard to downloads, we included in the experimental design a control group. In the control condition, the extension would be collecting data but not intervening during downloads. Each participant was randomly assigned (with probability 1/2) to one of two groups. The control group did not have the checksum verification result in the user interface (see Figure 15). The experimental group had, based on the user feedback collected in the in-situ experiment, a revised messages and layout of the verification result popup (see Figure 15). Adapted questionnaire

messages were displayed when a verification was made. For both groups, the data collection of the browsing and download history was activated. Both groups received the rESM questionnaires when the specific browsing conditions were met.

Finally, as we foresaw a small occurrences of downloads for which participants might incur into a checksum, we designed an exit activity that participants had to complete before collecting their financial incentive for the experiment. We designed a webpage with links to two apps they had to install on their computers. The webpage contained checksum so that it triggered our browser extension. For one app, the checksum was correct, while for the other app, it was incorrect; the ordering of the apps (correct/incorrect) was randomized to avoid presentation biases. This was intentionally designed to trigger downloads with valid checksums and downloads which could have been potentially tampered, and analyze reactions.

6.1.1 Apparatus. In order to capture the browsing and download behavior of the users and the answers of the rESM questionnaires, we developed a system consisting of two parts: a browser extension – to be installed in the participants' browsers – and a web server that communicated with the extension.

Chrome Extension. For this experiment, we adapted the browser extension that we initially used in the lab experiment (see Sec. 4.2). We added the following new functionalities:

- It captured and stored all browsing and download activities of the user. This consisted of the visited/downloaded URL, the timestamp, and the unique ID assigned to each participant during the installation process. This data was stored on the local machine and regularly uploaded to our servers.

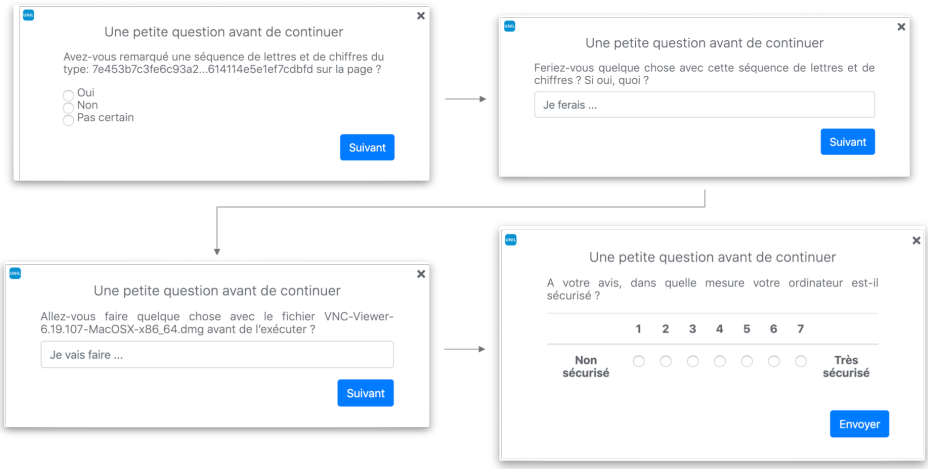
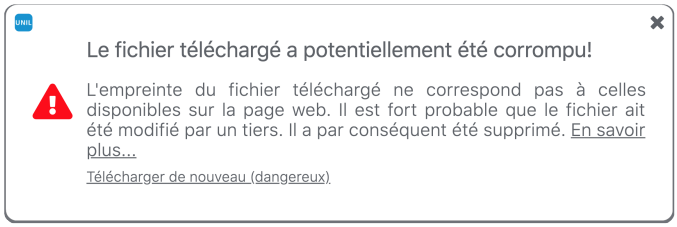


Fig. 14. Questions prompted by the browser extension when a download with a checksum on the webpage was detected. The first two questions were omitted in the case of a download without a checksum on the webpage. Top-left: *Did you notice this sequence of letters and numbers on the page [checksum string]?* [Yes/ No/ Not sure]. Top-right: *Would you do anything with this sequence of letters and numbers? If yes, what?* [free text] Bottom-left: *Are you going to do anything with the file you just downloaded [filename] before opening/executing it?* [free text] Bottom-right: *How secure is your computer in your opinion?* [Likert 7-levels from Not secure to Very secure].



(a) Correct checksum message. Title: "The integrity of the downloaded file has been successfully verified". Text: "The fingerprint of the downloaded file corresponds to the fingerprint highlighted in the webpage."



(b) Incorrect checksum. Title: "The downloaded file has possibly been corrupted!". Text: "The fingerprint of the downloaded file does not match those available on the webpage. It is very likely that the file has been modified by a third party. It has therefore been deleted. Learn more... Download again (dangerous)"

Fig. 15. Screenshot of the user interface, in French, of the extension (used in the in-the-wild experiment).

- It presented participants with the short aforementioned questionnaire. From a UI perspective, the questions were also displayed in popup windows with the question at the top and the answer options (or text field) right below.

In addition to these two functionalities, we updated the text of the popup messages according to the feedback we received during the lab study. Particularly, we changed the mechanism by which users were informed about non-matching checksums: while in the lab study we only displayed a warning message, for this experiment, the extension was deleting the potentially dangerous downloaded file and displaying a warning message. Basically, while for the lab experiment the participant could easily ignore the popup, in the field deployment we took a safer approach for which the user was actually required to read the warning and to explicitly click on a link if they wanted to bypass the verification.

The code of the revised browser extension was structured and implemented using the Google developers guidelines and the Chrome extension APIs. The extension had three main functions. Checksum verification are usually proposed for files that can be executed on the computer. However, according to our adversary model, any file hosted on a different server can benefit from an integrity verification (i.e., Microsoft Office or PDF documents [CVE-2017-0199], [51]). To collect relevant information, the extension monitored the Chrome download manager and sent back all information available in the `downloadItem` object (Object available through the Chrome Extension API).

Server. We set-up a Django web server with which the browser extension synchronizes. Additionally, the server contained a page with a step-by-step setup guide to install the extension and a dashboard for the researchers to monitor the progress of the study. Finally, the same server hosted the page we used for the exit task, where we asked participants to download two apps, one for

which the checksum was correct and the other for which the checksum was (purposely made) incorrect.

6.2 Participants

To take part in the 4-month long study, a total of 349 potential participants enrolled online for the experiment and were assessed for eligibility. These individuals volunteered to be part of a subject pool (consisting of approximately 8,000 subjects, most of whom were students) for behavioral experiments at the University of Lausanne (UNIL). A specialized unit at our institution, called Labex, managed the subject pool, took care of the randomization and enrollment processes, automated the transfers of financial incentives, and kept secure the contact information of the participants of the study. We collected demographic data through a short survey that also served to check eligibility for the study (i.e., a screener). The questionnaire verified the browser type used and only allowed Chrome users on desktop or laptop computer to continue. We also required the user to be at least once a week on their computer to join the experiment. If they corresponded to this profile, they were asked additional demographic questions and invited to participate. The main reason why potential participants were refused is that they only attempted to fill the screener from a mobile device and did not start the questionnaire again from Chrome on their computers. At the end of the screening process, a total of 152 people were selected to participate in the experiment. However, during the study, 18 participants dropped out (11.8% attrition rate), thus leaving us with a total of 134 that left the browser extension active for the 4 months of the study. Out of the 134 participants who completed the study, 57% (or 76) were female. The age distribution was as follows: 84% (113) were aged between 18 and 23, 15% (20) were between 24 and 30, and 1% (1) were over 30. We extracted the OS used from the user-agent string of the participant. About half of the participants were macOS users (45% or 60) and the other half Windows users (55%, or 74). A majority of users were from the Université de Lausanne (UNIL) (57%, or 77), the second group was from Ecole Polytechnique Fédérale de Lausanne (EPFL) (33%, or 44) and the last 10% (13) came from different schools in the French-speaking regions of Switzerland. A total of 134 participants remained active throughout the whole 4 months of the study. Concerning the two groups of the study, (44%, or 59) participants were assigned to the experimental group and (56%, or 75) to the control group. The results in the remainder of this section refer to the participants who remained active. However, concerning the last part of the study, only 117 participants completed the exit task. Therefore when we will describe this part of the experiment, the statistics refer to only the participants who completed this last activity.

6.2.1 Procedure. The screening process and the experiment were conducted in French. Registered subjects on the Labex panel received an e-mail invitation to fill out the online screener. The first page of the screener contained a description of the study, and a checkbox where participants could provide their informed consent. The consent form also described the goal of the study (i.e., an observation of the browsing and download behavior), the condition of participation, the data being collected (and the associated data management plan), the procedure to withdraw from the study, and information about the financial incentive. In addition to selecting the right participants, the screener questionnaire was used to setup participants for the study. At the end of the survey – and only for qualifying respondents – a request was made to our server to be assigned a participant ID. The server created a new ID in the users table and returned that to the survey platform that stored the ID together with the other responses of the screener. This process was used to separate the personal identifiable information of the participants (or PII) from the collected dataset. Also, this process would assign each new participant at random to the experimental or the control group.

Once the survey platform received the ID, the participant was automatically redirected to the extension installation instruction page.[22] The page contained a link to the extension on the Chrome web store and step-by-step guide on how to complete the setup. Participants also received the same information via e-mail. The page also provided information on how to pause the data collection of the extension, if the user wished to do so (e.g., for a browsing session that they might wanted to exclude from the data collection). Participants were also instructed to contact us via email if they wished to delete a browsing session that had been already captured. On startup, the extension verified two conditions. The first one was checking that the user ID existed in the database. This variable was saved using synchronized storage from the Chrome API. In the event that participant would start using Chrome on another computer and would log in their Google account, the extension would be installed on the new computer and the participant ID would be automatically added. This user ID also determines if users would see the download verification messages. If no ID was registered in the storage, the extension would use a JavaScript prompt to ask the user to enter their ID (communicated by e-mail and available on their installation page). We chose to use a JavaScript prompt because it is an intrusive way of communication and we did not want a participant to start a browsing session without being identified. The second check was to ensure that the extension would be able to access the downloaded files in order to verify checksums. We used a less intrusive way than the prompt to communicate this setup process to the user. If the URL file access was not allowed, the extension would open a page where it explains how to grant this permission to the extension.

Once the extension was correctly installed on the participant's computer, the only situation in which a user would interact with the extension would be either during a verified download (for the experimental group) or when an rESM questionnaire was triggered (for both groups) (see Sec. 6.1).

The participants were required to keep the extension installed on their main computer for a duration of 4 months. During this time, we monitored the server health. A secondary server was tasked to contact our server every minutes to ensure availability. Also, every days at 12pm, the main server sent an email to our team containing the last 24h graphs about CPU, memory and disk usage. The mail also included the IDs of participants that did not sent any data to the server for the last 5 days. When faced with such inactive participants, we contacted them by mail to ensure that they were still using the plugin on their personal computer. During the 4 months of the study, we contacted a total of 77 participants. Of these 59 reactivated their extension and continued the study while 18 participants dropped out of the study (11.8% attrition rate). The feedback we received from the inactive participants to explain their inactivity – when we contacted them – was very diverse. The most common reason was taking a few days off their computer, using only their smartphone/tablet instead for browsing the Web. The other reasons include not having Chrome set as default browser, the use of a secondary computer (e.g., a desktop for gaming), or even the purchase of a new computer.

At the end of the 4-month long experiment, we asked participants to complete a final task. They received instructions to visit one page located on our server. The page contained instructions to download and install two apps on their computers. Once installed, they had to enter their participant ID. This allowed us to collect feedback on the extension UI and see if they proceed to install the app with the incorrect checksum. Participants received CHF 20 (~USD 20) for their participation in the experiment. In addition, all participants took part in a raffle of 4 prices of CHF 100 at the end of the study.

[22]The original version of the webpage [French] is available at https://checksum.unil.ch/install/, last visited December 2019. Archival version at https://osf.io/za6j5/, last visited December 2019.

6.2.2 Measures. In order to answer our research questions, we relied on a combination of *objective observations* (logged by the browser extension) and *self-reported data* (collected through the rESM surveys). To address RQ4 and RQ5, for every webpage visited by a participant, the extension sent back a summary of the webpage to our server (i.e., a sanitized URL). On our server, we collected meta-data about the subject (IP address of the participant, user-agent string, time of visit and participant ID) and also, the fully qualified domain name (FQDN) of the page visited. It was the extension which took care of transforming the full URL into FQDN; for example, it transformed "`https://www.google.com/search?q=cat`" into "`www.google.com`". We did this transformation to avoid the collection of password or security token encoded in the URL. However, if a webpage visited would contain a checksum or a hashing algorithm name, the full URL and the related checksums information were saved on our server. We chose to keep the full URL so we could inspect the webpages that are false positive and refine our checksum selection criteria. The data was accessible only to the researchers of this study and will be deleted after the publication of the results, at the latest, one year after the end of the experiment. We mentioned these points when collecting consent from the participants at the beginning of the experiment (the participants had to sign a consent form).

In terms of self-reported data for RQ6, as the extension detected a checksums, the digest on the page was highlighted and the following two questions presented to the participant (see top questions of Figure 14):

(1) "*Avez-vous remarqué cette séquence de lettres et de chiffres [checksum] sur la page ?*" (Did you notice this sequence of letters and numbers on the page: [checksum string]? [Yes/No/Not sure]);
(2) "*A votre avis, à quoi sert cette séquence de lettres et de chiffres ?*" (What do you think this sequence of letters and numbers is used for? [free text]).

As explained in Sec. 6.1, in order not to overload the participants, we triggered these mini questionnaires at most once per day, per type of event, per participant.

In terms of objective data we stored concerning the downloaded files, the three main pieces of data that were relevant for RQ6 were (1) the MIME type (e.g., `application/pdf` for PDF documents) of the downloaded file, (2) the address that initiated the download (i.e. the page on which the user clicked to trigger the download) and (3) the address that the download was being made from after all redirects (i.e. the address from where the data was downloaded). The MIME type is a data format identifier that allows us to know if the downloaded file is potentially able to make modification on the computer thus of interest for an attacker. The association of the address that initiated the download and of the address that served the download allows us to determine whether the downloaded file is stored on an *external* server or not. Note that this is a simple heuristic: we compare the domain name of the server that served the webpage from where the download was triggered with the domain name of the server that served the downloaded file. If they are different, then we consider the download as happening on an external server. The extension was unable to monitor what would happen to the file once the download was complete. Therefore, we had to rely on the rESM survey to capture whether the participant was going to process the downloaded file before executing it (see bottom-left question of Figure 14).

Lastly for RQ7, we were interested in comparing the self-reported security of the computer between the control and the experimental group. For this reason, after each download (with or without checksum) we asked all participants to rate on a 7-levels scale from 'Not secure' to 'Very secure' the level of security of their computer (see bottom-right question of Figure 14).

6.2.3 Statistical Analysis. Nonparametric analysis was applied to the data considering the ordinal nature of some of the observed variables. Hence, differences between security valuation across

MIME type	num.	prop. [%]
application/pdf	9677	55.61
image/jpeg	1988	11.43
application/octet-stream	1151	6.61
application/vnd.openxmlformats-officedocument.wordprocessingml...	883	5.07
application/zip	648	3.72
audio/mpeg	284	1.63
application/vnd.openxmlformats-officedocument.presentationml...	221	1.27
image/png	210	1.21
text/plain	194	1.11
application/vnd.openxmlformats-officedocument.spreadsheetml...	182	1.05
application/msword	174	1.00
text/html	136	0.78
binary/octet-stream	106	0.61
video/mp4	100	0.57
image/gif	99	0.57
application/x-msdownload	93	0.53
application/x-bittorrent	92	0.53
application/binary	88	0.51
application/vnd.ms-powerpoint	88	0.51
application/force-download	57	0.33

Table 2. Top 20 MIME types of the 17,400 files participants downloaded during the experiment.

experimental conditions were tested suing the Kruskal-Wallis test (see RQ7 below). The level of significance was taken as $p < .05$.

6.3 Results and Analysis

(RQ4) How often do users download files from the Web and what types of files do they download? During the 4 months of observation, the participants of the study visited a total of 657,608 webpages.[23] On average each participant visited around 50.6 webpages every day. During the study, participants downloaded a total of 17,400 files from the Web. On average each participant downloaded about 130 files, that is around 33 downloads each month. Table 2 presents the breakdown of the different MIME types of the files downloaded by the participants of the study during the experiment. We notice that 7.7% of these files are executable (i.e., octet-stream, binary), and 3.7% are compressed archives, hence files that could potentially carry malicious code. Additionally, the large majority 55.6% of downloads are PDFs and office documents that could also be injected with corrupted macros or other harmful code. During the study, a total of 17 executable downloads contained a checksum on the page that was verified by the browser extension. For 6 of the 17 downloads the participants that originated the download was in the experiment group, hence the browser extension presented popup messages about the verification to the users. A large proportion of the downloaded files came from e-mail attachments accessed via a webmail. The modest number of executable files downloaded can be explained by the fact that users download the bulk of the programs they need when they set-up their computers and only a few ones, sporadically, after that.

(RQ5) What is the prevalence and the current practices of checksums included in download webpages? Of the 17,400 downloads recorded in the final dataset, 4,853 files (or 27.9%) were hosted on a server that had a distinct domain name from the server that served the webpage of the site. This shows that

[23]Note that the numbers of webpages/downloads reported in this section are total numbers, not numbers of *unique* webpages/downloads.

about a third of the downloaded files we recorded in this study could potentially be compromised following the threat model described in this paper. Similarly, we found that 923 downloads (or 5.3%) originated from a server that was not configured with HTTPS, hence allowing potential attackers to modify the files while being transferred to the machine requesting the download. Out of the 657,608 webpages visited by the participants during the study, only 153 pages (or 0.02%) contained a checksum string, and of the 17,400 downloads events in the final dataset, 37 originated from one of these webpages. We manually inspected these webpages and classified them into four categories: (i) 70 (or 45.7%) webpages linked executable files and references the name of the algorithm used to generate the checksum (i.e., SHA-256, MD5); (ii) 43 (or 28.1%) were false positives (i.e., webpages that contained alphanumeric strings that matched our regular expression but that were not checksums); (iii) 29 (or 19%) webpages linked torrent files (we will discuss this case below); and (iv) 11 (or 7.2%) webpages contained true checksums but the connected file was not an executable, hence the extension did not verify the integrity. For instance, `Zenodo` provides checksums for PDFs[24], and `Digicert` for security certificates.[25] Concerning the webpages with checksum that linked torrent files, in these cases the participants visited a webpage that contained checksum information about one or multiple files seeded through the peer-to-peer network.[26] In addition, these webpages also contained a `.torrent` file that could be downloaded from the webpage that contains metadata about files and folders to be distributed, and usually also a list of trackers, which are computers that help users of the system find each other. When participants of our experiment downloaded torrent files from these webpages, the extension was triggered, however it could not possibly verify the checksum as the file being downloaded from the browser (i.e., the .torrent file) was not the one the checksum information on the webpage referred to.

Of the 7.7% of downloads involving executable files (plus the 3.7% of downloads involving archives), 17 downloads were downloaded from a webpage containing both the checksum and the name of the algorithm used. These were all executable files (i.e. .exe, .dmg, or .pkg) or archives (i.e., .zip). Finally, it is worth reporting that all of the checksums reported on the webpages we identified in the study matched the linked resources. Table 3 reports the details of the resources with valid checksum that our participants downloaded during the study.

(RQ6) What do Internet users most frequently do when they encounter a visible checksum? Out of the 153 events in which participants opened a webpage that contained a checksum and were prompted with a rESM questionnaire, we collected 35 valid responses. These 153 events were created by only 45 distinct participants. The participant would typically look at multiple pages with checksum under the same FQDN during the same day and thus receive only one questionnaire. To the first question, namely whether they noticed the checksum on the webpage, 28 (or 80%) participants replied that they did not see the checksum, while 3 (or 8.6%) replied that they were not sure, and 4 (or 11.4%) that they had seen the checksum. In sum, 88.6% of respondents did not see or were not sure about whether the checksum was on the webpage. In the follow-up question, we asked the 35 respondents to explain, in their own words, what is the purpose of the sequence of letters and numbers (i.e., the checksum or digest). Unfortunately, only one respondent provided an almost-correct explanation of the purpose of checksums: *Elle sert a verifier que mon téléchargement est bien téléchargé car il s'agit de logiciel et le fichier doit être intact* (It is used to check that my download is correctly downloaded because it is a software and the file has to be intact) [Business School student]. The rest of the respondents provided answers that were incorrect: e.g., *Peut-être*

[24]See an example here https://zenodo.org/record/204969#.XfpP4db0k1J, last visited December 2019.
[25]See https://www.digicert.com/digicert-root-certificates.htm, last visited December 2019.
[26]As an example, see https://osf.io/zw7u3/, last visited: Dec. 2019.

Filename	Source address
gimp-2.10.10-setup.exe	https://www.gimp.org/downloads/
RStudio-1.2.1335.dmg	https://www.rstudio.com/products/rstudio/download/
R-3.5.3.pkg	https://cran.r-project.org/bin/macosx/
vlc-3.0.6.dmg	http://get.videolan.org/vlc/3.0.6/macosx/vlc-3.0.6.dmg
basic-miktex-2.9.7031-x64.exe	https://miktex.org/download
python-3.7.2-macosx10.9.pkg	https://www.python.org/downloads/
RStudio-1.1.463.exe	https://www.rstudio.com/products/rstudio/download/
python-3.7.2-amd64.exe	https://www.python.org/downloads/release/python-372/
VirtualBox-5.2.18-124319-OSX.dmg	https://nas-webdav.epfl.ch/vpsi1arch/images_vdi/IC_CO_IN-SC-Local/
VirtualBox-5.2.18-124319-Win.exe	https://nas-webdav.epfl.ch/vpsi1arch/images_vdi/IC_CO_IN-SC-Local/
basic-miktex-2.9.6942-x64.exe	https://miktex.org/download
python-3.7.2.exe	https://www.python.org/downloads/release/python-372/
winscp576setup.exe	https://winscp.net/download/winscp576setup.exe
Popcorn-Time-0.3.10-Setup.exe	http://mirror03.popcorntime.sh/repo/build/Popcorn-Time-0.3.10-Setup.exe
TeamSpeak3-Client-win64-3.2.3.exe	https://www.teamspeak.com/en/your-download/
Popcorn-Time-0.3.10-Mac.zip	http://mirror03.popcorntime.sh/repo/build/Popcorn-Time-0.3.10-Mac.zip
Panaustik64.exe	https://www.panaustik.com/telechargement/

Table 3. Executables downloaded by the participants during the study from web pages where a valid checksum was available.

un numéro de série ou d'identification pour le programme (Maybe is a serial number or identification number for the program) [Basic Sciences student].

After a download event, we also prompted participants of the study with an rESM questionnaire to understand whether they would do anything with the downloaded file before executing it. A total of 155 responses from 97 distinct participants provided answers to this question during the course of the study. The large majority of the responses (i.e., 124 or 80% of the responses) stated they would directly execute the file. The remaining declared to either scan the file with an antivirus software (i.e., 4 or 2.6% responses) or provided unclear answers (i.e., 26 or 16.8% responses). Only one respondent reported performing a checksum verification on the file: *Non, je fais confiance à l'éditeur en l'occurrence. Sinon, je fais un check MD5* (No, I trust the developer. Otherwise, I do a MD5 check) [Criminal Sciences student]. This shows a misconception regarding the trust assumption: Checksums are used in the case where a third-party host is compromised, not the software developer. It is interesting to notice, that scanning a corrupted file with an antivirus might not protect entirely from potential threats (e.g., malware with zero-day exploit).

At the end of the study we asked participants to complete a final task (see Sec. 6.2.1 above). A total of 117 participants completed this step (while 17 participants dropped out at the very end). Of the remaining participants, 48 (or 41%) were in the experimental condition (i.e., with extension warnings active) and 69 (or 59%) participants were in the control group. During the final tasks these participants were presented with the download of an app for which an incorrect checksum was provided on the webpage. While almost all participants in the control group installed the 'malicious' app (except 4 or 3.4% participants who did not understand the instructions), 12 (or 10.3%) participants of the experimental condition did not complete the install process even if they were instructed to do so. Most of the other people in the experimental condition who forced the download by using the 'Download again (dangerous)' button, did so because they trusted our institution: *Si j'en crois ce qui a été affiché, 'il a été corrompu'. Je présume qu'il s'agit cependant du déroulement*

habituel de l'expérience (If I believe what is displayed, the file is corrupted. I presume this is however the usual course of experience) [Criminal Sciences student]. This difference between the two groups has to be ascribed to the warnings of the browser extension, which made participants more wary of the potential threat.

(RQ7) Would users feel more secure if they could use a system that automates parts of the verification process? On the last screen of the rESM questionnaire, we asked participants to rate the perceived security of their computer using a Likert scale with 7 levels (this goes from 1=Extremely insecure, to 7=Extremely secure). In the exit task of the study, a total of 117 participants were asked to download and install two applications on their computer, one with a valid and one with an invalid checksum. These participants experienced installing an application that could have been potentially corrupted. To answer our RQ, we compared the security ratings provided by these participants to the rESM question. A Kruskal-Wallis test showed that participants in the experimental group reported higher security ratings (M=5.0 points, SD=1.4) for their computer than participants in the control group (M=4.3 points, SD=1.7); $H(1)=8.83$, $p<.05$.

6.4 Limitations

Our participants were all university students, hence their age was relatively homogeneous (around 20 years old). Typically, age is considered to be related to the level of technical expertise of the person. However, recent research has revealed that *cognitive ability* and *previous technology experience* are better predictors of the ability of people to solve information-retrieval tasks [52]. In the presented study, we did not control for these two factors. However, we might expect most university students to possess relatively high cognitive abilities and to have had prior exposure to online technology. Hence, we might expect a smaller share of users to understand and use checksums in the general population than the share identified in our results.

Additionally, our experimental design required participants to regularly use Google Chrome as their main browser or to be willing to use it primarily for the duration of the study. Although Chrome holds the largest market share,[27] there are lots of users that use alternatives such as Apple Safari, Microsoft Edge and IE. The interesting aspect to note is that while Safari, Edge and IE come preinstalled on computers running macOS and Windows, respectively, Chrome needs to be installed, hence its users might be more tech-savvy than users who use the preinstalled browser. Hence, we might expect that by including users of these other browsers in the sample we might observe a smaller share of users who understand and use checksums than those identified in this study.

Finally, in this study, we did not include browsing behavior on mobile devices. Reports show that an increasing number of users access the Internet primarily – or exclusively – from a mobile device.[28] Going forward, research should study the use of checksums on mobile devices, which might be specifically targeted by attackers.

7 GENERAL DISCUSSION

The number of Internet users potentially exposed to corrupted files is alarming. Our previous large-scale study [1] showed that, out of the 62.2% of all the respondents who download programs from the Internet, only 6.1% do so exclusively from official app stores, such as the Mac App Store or the Microsoft Store. Checksums, if used correctly, could therefore prevent the execution of potentially malicious code for more than half of the users who download files from the Internet.

[27] See https://en.wikipedia.org/wiki/Usage_share_of_web_browsers, last accessed June 2020.

[28] See https://www.pewresearch.org/internet/2019/06/13/mobile-technology-and-home-broadband-2019/, last accessed June 2020.

Sadly and as expected, our recent in-the-wild experiment confirmed that the vast majority of participants (88.6%) do not notice the presence of checksums, even when they are visible on the download page. To make things worse, most participants in our experiments did not know how to use this information even when we pointed them to the part of the page that contains the checksum. However, when the browser extension was active, we observed differences in the participants' reaction to corrupted downloads. Interestingly, only 25% of participants in this experiment stopped the installation process of the program that has triggered the warning, whereas 40% of them did so in the previous lab experiment. This is relatively surprising, as we might expect that users would be more cautious with their own computer. However, it could also be due to the fact that participants in the lab experiment were carefully instructed to follow some steps and were in a controlled environment, whereas those in the in-the-wild experiment had fewer instructions and were in their usual daily environment (where they dedicate less time to such tasks).

In this article, we have further uncovered some of the behavioral aspects that are associated with a successful detection of a mismatch in two checksums, as performed in the browser's UI as well as in a separate program, i.e., the command-line. Our statistical analysis showed that the number of transitions between the terminal and the web browser's window is significantly higher for participants who detected the mismatch, and that those participants also checked the entire sequence more often than the ones who did not notice the mismatch. However, to our surprise, the dwell time was smaller when the warning was shown, suggesting that the part of the UI that carries the warning message was effective.

In sum, these findings indicate that manually inspecting the integrity of downloads is a process that is cognitively intense, and requires a sophisticated mental model of the security concept behind checksums. We cannot reasonably expect that most Internet users will be able to manually perform these checks on downloaded files. Finally, even if we observed some positive effects of the extension on users' behavior, the results also show that we could improve the warning message in order to further reduce the fraction of users who execute potentially harmful files downloaded from the Internet.

In order to improve the security and usability of web downloads, we have shown that it was crucial to automate the checksum verification process, as alluded to by Tan et al. [28]. We propose an approach that consists of a mix of short- and long-term solutions. In the short term, our in-the-wild experiment has shown that our proof-of-concept Chrome extension did not detect corrupted files for any of the 17 downloads where it was triggered. Although very precise, we cannot exclude that it missed some websites where the checksums were available in some other form (e.g., an image or `iframe` element). Also, due to the limited sample size, we refrain from generalizing the success rate to the entire Web.

Due to the challenges in assessing its recall and false positives, such a short-term solution is likely insufficient to fully protect the 73% of the downloads (PDFs and executables) that could potentially be harmful, if corrupted. Instead, as a long-term solution, we propose to extend the coverage of the SRI specification [7] to include HTML `a` elements that point to files to be downloaded, and optionally the `meta` and `iframe` elements. Such a solution would, however, require more effort from the website owners (to serve the SRI integrity field), and perhaps from content creators as well (to generate checksums for their files). Finally, it is also crucial to increase awareness about the threat vectors antiviruses can and cannot handle. As we have observed, some participants felt safe because they scanned files with an antivirus software. Unfortunately, an antivirus does not protect from all possible threats, especially from malware with zero-day exploits.

We firmly believe that the entire web ecosystem (standards bodies, browser vendors, content publishers and end-users) would greatly benefit from a safer and more usable experience, if such an obvious and arguably underexploited attack vector was eradicated.

8 CONCLUSION & FUTURE WORK

In this work, we pursued a line of research on the use of checksums for integrity verification of web downloads and made a number of contributions. In particular, we showed that the current verification process is taxing and error prone, both in a controlled and in a real-world environment. Specifically, we demonstrated that an adversary can successfully mount an attack by replacing a program with a malware with a checksum that partially matches that of the program since many users check only the beginning of the checksums.

The logical outcome of this work is to automate the checksum verification, so as to increase the usability of this security feature and to make it available to non-technical users. Hence, as a second contribution of this work, we developed a browser extension that computes the checksums of the files downloaded from the Web and matches them against those found on webpages. The usability evaluation of the extension suggested that it simplified the verification process and was effective in dragging the user attention on the warnings describing the risks of downloading and executing possibly corrupted files. The four-month deployment of the extension showed that none of the downloaded files (with a checksum available) were corrupted (even though such download happened very rarely over the course of the study). This deployment further confirmed that warnings were not always sufficient to prevent a user from downloading a corrupted file, thus that a more disruptive change is needed to protect integrity of web downloads.

An interesting research avenue for future work is to investigate means for users to identify the origin of the checksums displayed on download webpages (i.e., developer-generated vs. host-generated) as well as means for handling updates of download files (i.e., the associated update of the checksums). One possible option is to rely on digital signatures[29] but such solutions might be vulnerable to version-rollback attacks (e.g., a program file could be maliciously replaced with an older version of it – with known exploitable vulnerabilities).

Finally, we are currently writing a W3C proposal to extend subresource integrity to other HTML elements including links. We intend to promote our proposal to (and collaborate with) the different stakeholders involved, that is the W3C and web browsers (e.g., Google, Mozilla) development teams in order to have a concrete impact on the security of Internet users.

ACKNOWLEDGEMENTS

The authors express their sincere gratitude to Italo Dacosta, Andreas Kramm, Nicolas Le Scouarnec, Adrienne Porter Felt, Nina Taft, Lawrence You, and Blase Ur for their feedback. The authors also warmly thank Holly Cogliati for her great editing job on the manuscript. This work was partially funded with grant #19024 from the Hasler Foundation and with a grant from HEC Lausanne. This work was carried out while Alexandre Meylan and Bertil Chapuis were with UNIL.

REFERENCES

[1] M. Cherubini, A. Meylan, B. Chapuis, M. Humbert, I. Bilogrevic, and K. Huguenin, "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, Oct. 2018, pp. 1256–1271.

[2] K. Turner, "Developers consider Apple's App Store restrictive and anticompetitive, report shows," *Washington Post*, 2016-07-15.

[3] S. Khandelwal, "Flaw in Popular Transmission BitTorrent Client Lets Hackers Control Your PC Remotely," https://thehackernews.com/2018/01/bittorent-transmission-hacking.html, 2018.

[4] "Linux Mint Website Hacked; ISO Downloads Replaced with a Backdoor - Security News - Trend Micro USA," https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/linux-mint-website-hacked-iso-downloads-replaced-with-a-backdoor.

[29] https://github.com/w3c/webappsec-subresource-integrity/blob/master/signature-based-restrictions-explainer.markdown, for instance; last accessed: July 2020.

[5] H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun, and B.-Y. Yang, "A Study of User-Friendly Hash Comparison Schemes," in *Proc. of the Computer Security Applications Conf. (ACSAC)*. IEEE, Dec. 2009, pp. 105–114.

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

[7] W3C, "Subresource Integrity," https://www.w3.org/TR/SRI/, 2016.

[8] S. M. Furnell, P. Bryant, and A. D. Phippen, "Assessing the security perceptions of personal Internet users," *Computers & Security*, vol. 26, no. 5, pp. 410–417, Aug. 2007.

[9] C. L. Anderson and R. Agarwal, "Practicing Safe Computing: A Multimedia Empirical Examination of Home Computer User Security Behavioral Intentions," *MIS Q.*, vol. 34, no. 3, pp. 613–643, Sep. 2010.

[10] V. Rishi, "Cyber Security Breaches Survey 2018," United Kingdom, Survey, Apr. 2018.

[11] E. M. Redmiles, S. Kross, and M. L. Mazurek, "Where is the Digital Divide?: A Survey of Security, Privacy, and Socioeconomics," in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2017, pp. 931–936.

[12] ——, "How I Learned to Be Secure: A Census-Representative Survey of Security Advice Sources and Behavior," in *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, 2016, pp. 666–677.

[13] Elisa M. Redmiles, Amelia R. Malone, and Michelle L. Mazurek, "I Think They're Trying to Tell Me Something: Advice Sources and Selection for Digital Security," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*, May 2016, pp. 272–288.

[14] S. Egelman, L. F. Cranor, and J. Hong, "You've been warned: An empirical study of the effectiveness of web browser phishing warnings," in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2008, pp. 1065–1074.

[15] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness." in *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, 2009, pp. 399–416.

[16] D. Akhawe and A. P. Felt, "Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness." in *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, 2013.

[17] S. Egelman and S. Schechter, "The importance of being earnest [in security warnings]," in *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*. Springer, 2013, pp. 52–59.

[18] D. Modic and R. Anderson, "Reading this may harm your computer: The psychology of malware warnings," *Computers in Human Behavior*, vol. 41, pp. 71–79, 2014.

[19] A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, and G. Vigna, "What the app is that? deception and countermeasures in the android user interface," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, 2015, pp. 931–948.

[20] J. L. Jenkins, B. B. Anderson, A. Vance, C. B. Kirwan, and D. Eargle, "More harm than good? How messages that interrupt can make us vulnerable," *Information Systems Research*, vol. 27, no. 4, pp. 880–896, 2016.

[21] M. Silic and A. Back, "Deterrent Effects of Warnings on User's Behavior in Preventing Malicious Software Use," in *Proc. of the Hawaii International Conference on System Sciences (HICSS)*, 2017.

[22] R. W. Reeder, A. P. Felt, S. Consolvo, N. Malkin, C. Thompson, and S. Egelman, "An Experience Sampling Study of User Reactions to Browser Warnings in the Field," in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2018, pp. 512:1–512:13.

[23] C. Bravo-Lillo, S. Komanduri, L. F. Cranor, R. W. Reeder, M. Sleeper, J. Downs, and S. Schechter, "Your Attention Please: Designing Security-decision UIs to Make Genuine Risks Harder to Ignore," in *Proc. of the Symp. on Usable Privacy and Security (SOUPS)*. ACM, 2013, pp. 6:1–6:12.

[24] C. S. Weir, G. Douglas, M. Carruthers, and M. Jack, "User perceptions of security, convenience and usability for ebanking authentication tokens," *Computers & Security*, vol. 28, no. 1-2, pp. 47–62, 2009.

[25] L. Tam, M. Glassman, and M. Vandenwauver, "The psychology of password management: A tradeoff between security and convenience," *Behaviour & Information Technology*, vol. 29, no. 3, pp. 233–244, 2010.

[26] M. Fagan and M. M. H. Khan, "Why do they do what they do?: A study of what motivates users to (not) follow computer security advice," in *Proc. of the Symp. on Usable Privacy and Security (SOUPS)*. ACM, 2016, pp. 59–75.

[27] K. Krombholz, K. Busse, K. Pfeffer, M. Smith, and E. von Zezschwitz, ""If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, May 2019, pp. 1138–1155.

[28] J. Tan, L. Bauer, J. Bonneau, L. F. Cranor, J. Thomas, and B. Ur, "Can Unicorns Help Users Compare Crypto Key Fingerprints?" in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2017, pp. 3787–3798.

[29] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, May 2015, pp. 232–249.

[30] R. Abu-Salma, M. A. Sasse, J. Bonneau, A. Danilova, A. Naiakshina, and M. Smith, "Obstacles to the Adoption of Secure Communication Tools," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, May 2017, pp. 137–153.
[31] Elham Vaziripour, Justin Wu, Mark O'Neill, Ray Clinton, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala, "Is that you, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications," in *Proc. of the Symp. on Usable Privacy and Security (SOUPS)*. ACM, 2017.
[32] "Checksum On the Go - Chrome Webstore," https://chrome.google.com/webstore/detail/checksum-on-the-go/fholnooplijidhdagedffljaphholpea.
[33] "Files MD5 SHA1 Calculate & Compare – Add-ons for Firefox," https://addons.mozilla.org/en-US/firefox/addon/calculate-md5-sha1-hash-che-1/?src=search.
[34] "Certificates and Digitally Signed Applications: A Double Edged Sword," https://eventtracker.com/tech-articles/certificates-and-digitally-signed-applications-a-double-edged-sword/, Feb. 2016.
[35] N. Vratonjic, J. Freudiger, V. Bindschaedler, and J.-P. Hubaux, "The Inconvenient Truth About Web Certificates," in *Proc. of the Workshop on Economics of Information Security and Privacy (WEIS)*. Springer, 2013, pp. 79–117.
[36] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "A Look in the Mirror: Attacks on Package Managers," in *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, 2008, pp. 565–574.
[37] B. Preneel, "Cryptographic hash functions," *Transactions on Emerging Telecommunications Technologies*, vol. 5, no. 4, pp. 431–448, 1994.
[38] Computer Security Division, Information Technology Laboratory, "NIST Policy on Hash Functions - Hash Functions | CSRC," https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions.
[39] B. Chapuis, O. Omolola, M. Cherubini, M. Humbert, and K. Huguenin, "An Empirical Study of the Use of Integrity Verification Mechanisms for Web Subresources," in *Proc. of The Web Conference*. ACM, Apr. 2020, pp. 34–45.
[40] B. B. Anderson, C. B. Kirwan, J. L. Jenkins, D. Eargle, S. Howard, and A. Vance, "How Polymorphic Warnings Reduce Habituation in the Brain: Insights from an fMRI Study," in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2015, pp. 2883–2892.
[41] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes, "Improving SSL Warnings: Comprehension and Adherence," in *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2015, pp. 2893–2902.
[42] M. Silic, J. Barlow, and D. Ormond, "Warning! A comprehensive model of the effects of digital information security warning messages," in *Proc. of the IFIP Workshop on Information Systems Security Research*. IFIP, 2015.
[43] A. Poole and L. J. Ball, "Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects," in *Encyclopedia of Human Computer Interaction*, 2006, p. 13.
[44] J. H. Goldberg, M. J. Stimson, M. Lewenstein, N. Scott, and A. M. Wichansky, "Eye tracking in web search tasks: Design implications," in *Proc. of the Symp. on Eye Tracking Research & Applications (ETRA)*. ACM, 2002, p. 51.
[45] M. A. Just and P. A. Carpenter, "Eye fixations and cognitive processes," *Cognitive Psychology*, vol. 8, no. 4, pp. 441–480, Oct. 1976.
[46] M. Cherubini and N. Oliver, "A Refined Experience Sampling Method to Capture Mobile User Experience," *arXiv:0906.4125 [cs]*, Jun. 2009.
[47] S. Consolvo and M. Walker, "Using the experience sampling method to evaluate ubicomp applications," *IEEE Pervasive Computing*, vol. 2, no. 2, p. 24–31, Jun 2003.
[48] G. Iachello, K. N. Truong, G. D. Abowd, G. R. Hayes, and M. Stevens, "Prototyping and sampling experience to evaluate ubiquitous computing privacy in the real world," in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2006, p. 1009–1018.
[49] C. Mancini, K. Thomas, Y. Rogers, B. A. Price, L. Jedrzejczyk, A. K. Bandara, A. N. Joinson, and B. Nuseibeh, "From spaces to places: Emerging contexts in mobile privacy," in *Proc. of the Int'l Conf. on Ubiquitous Computing (UbiComp)*. ACM, 2009, p. 1–10.
[50] S. S. Intille, J. Rondoni, C. Kukla, I. Ancona, and L. Bao, "A context-aware experience sampling tool," in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems (CHI) - Extended abstracts*. ACM, 2003, p. 972–973.
[51] F. Schmitt, J. Gassen, and E. Gerhards-Padilla, "PDF Scrutinizer: Detecting JavaScript-based attacks in PDF documents," in *Proc. of the Int'l Conf. on Privacy, Security and Trust (PST)*. IEEE, Jul. 2012, pp. 104–111.
[52] M. Crabb and V. L. Hanson, "Age, technology usage, and cognitive characteristics in relation to perceived disorientation and reported website ease of use," in *Proc. of the Int'l ACM SIGACCESS Conf. on Computers & Accessibility (ASSETS)*. ACM, Oct. 2014, pp. 193–200.