Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses

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

Alexandre Meylan
UNIL – HEC Lausanne
Switzerland
alexandre.meylan@unil.ch

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

Mathias Humbert
Swiss Data Science Center
ETH Zurich and EPFL
Switzerland
mathias.humbert@epfl.ch

Igor Bilogrevic
Google Inc.
Switzerland
ibilogrevic@google.com

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

ABSTRACT

Internet users can download software for their computers from app stores (e.g., Mac App Store and Windows Store) or from other sources, such as the developers' websites. Most Internet users in the US rely on the latter, according to our representative study, which makes them directly responsible for the content they download. To enable users to detect if the downloaded files have been corrupted, developers can publish a checksum together with the link to the program file; users can then manually verify that the checksum matches the one they obtain from the downloaded file. In this paper, we assess the prevalence of such behavior among the general Internet population in the US (N = 2,000), and we develop easy-to-use tools for users and developers to automate both the process of checksum verification and generation. Specifically, we propose an extension to the recent W3C specification for sub-resource integrity in order to provide integrity protection for download links. Also, we develop an extension for the popular Chrome browser that computes and verifies checksums of downloaded files automatically, and an extension for the WordPress CMS that developers can use to easily attach checksums to their remote content. Our in situ experiments with 40 participants demonstrate the usability and effectiveness issues of checksums verification, and shows user desirability for our extension.

CCS CONCEPTS

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

KEYWORDS

checksums; web downloads; security; usability

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243746

ACM Reference Format:
Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2018. Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3243734.3243746

1 INTRODUCTION

Nowadays, Internet is the main source for users to obtain programs for their computers. A popular and convenient way to download programs is to use official app stores such as Apple's Mac App Store and Microsoft's Windows Store. Such platforms, however, have several drawbacks for developers, including long review and validation times, technical restrictions (e.g., sandboxing), incompatibility with software licenses, and substantial commissions [45]. Therefore, it is quite common that developers make their programs available directly from their websites. This is the case of popular programs such as VLC media player, OpenOffice, and GIMP.

When developers make programs available from their websites, they can either host them directly on the same server as the website or rely on so-called third-party hosting platforms, such as mirrors and content delivery networks (CDNs) that provide substantial improvements for the users in terms of performance (e.g., bandwidth) and availability. However, this latter solution means relinquishing control over the program files and fully trusting the third-party platform: If the CDN is hacked, the programs can be corrupted to include all sorts of malware, thus infecting the computers of the users willing to install and run the original program. Recently, both the popular BitTorrent client Transmission [5, 29] and the Linux Mint distribution [3, 50] were corrupted by injecting, in the original programs, a ransomware in the former, and a backdoor in the latter. Such corruptions are particularly problematic for privacy and security software (e.g., PGP) used by at-risk populations such as journalists and political dissidents. This applies to all files that can harm a user's device; even PDF files can be infected. 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. It would indeed be problematic (not only from a security point

of view) if a picture linked from a website would be replaced by another one.

A popular way for developers to enable users to detect accidental or intentional modifications of their program files hosted on external platforms, such as mirrors and CDNs, is to provide so-called checksums on their websites. This practice is quite common in the open-source community but also for companies such as Google (e.g., checksums for Android Studio). Such checksums are usually derived from the output of cryptographic hash functions (e.g., SHA-1), in the form of sequences of alphanumeric digits called digests and displayed on the download webpages, or from digital (detached) signatures provided in separate files. Users can verify the integrity of the program files they download based on the provided checksums;[1] however, it usually requires the users to execute dedicated (command line)[2] programs and to manually compare long sequences, which has been proven, in contexts different from web downloads (e.g., PGP key fingerprint verification), to suffer from usability issues and to be error prone [26, 39]. 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.

In this paper, we consider the context of a website administrator (e.g., a software developer or vendor) who includes a link on one of their webpages for downloading a file such as a program. The program file is hosted on a third-party server (i.e., different from the server hosting the website, e.g., mirrors, CDNs) that is not controlled by the website administrator. In order to check the integrity of the file after download, this website administrator includes a checksum of the program file on their website (not hosted on the third party server) . The threat we consider is the case where the file pointed to by the link is corrupted, for instance by the third-party server operator or by hackers; furthermore, the checksum of the corrupted file can partially match that of the original file (e.g., the first ten digits match). And the users who download the file do not verify its checksum or do so in an inattentive way. To the best of our knowledge, no practical solution has been proposed for automatically verifying web downloads and the general topic of the integrity of programs downloaded on the Web has been mostly overlooked. We explore this research gap and address these challenges by performing a thorough analysis of the common practices regarding the use of checksums and their usability and effectiveness, and by proposing technical solutions to the issues we identify. Specifically, our contributions are as follows:

- We conduct the first comprehensive study on the use of checksums for verifying the integrity of web downloads. We rely on several instruments: a large-scale online survey with 2,000 participants to assess Internet users' knowledge of checksums and security behavior in general with respect to web downloads of program files; a website survey of 20 download pages of popular programs to assess how developers use checksums; and 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 eye-tracking technologies are used for studying usability and attention during the checksum verification process. We identify the main challenges with checksums from the point of view of usability, efficiency, security and adoption.
- 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) [49]; it standardizes the use of checksums for external resources such as Javascript files, in order to cover download links of program files. Our solution enables developers to rely on a standardized method that would significantly reduce the user burden of checksum verification.
- We develop two novel extensions for improving checksum verification. The first one is an automated checksum verification browser extension that alerts users when there is a potential mismatch between the checksum computed from the downloaded file and that (or those) available on the developer's website, even when the checksum is displayed in the page (and not through SRI), as is the case today. The second one is a checksum generation plugin for a popular web content-management system (CMS), which automatically implements our proposed extension to the W3C's SRI specification on the websites managed by such systems.

Our proposed solution would be deployed as follows: The use of checksums for web downloads and their inclusion in links would be standardized as an extension of SRI; alternatively, checksums would be extracted from the "Download" webpages in an ad-hoc fashion. The automatic extraction and verification of the checksums would be implemented in the web browser (or in an extension).

The results of our large-scale online survey demonstrate that, although 71% of the respondents declare to download program files from sources where they could be potentially corrupted, only 1.7% check the integrity of these downloads. Furthermore, when asked about what they would do if they saw a checksum on a website they downloaded a program from, only 5.2% of the participants stated they would use it to verify the file's integrity (out of 6 possible options), thus revealing the sheer number of computer users who know about this security technology. The results of our website survey show that a substantial fraction of download webpages include only checksums generated with weak hash functions (MD5 and SHA1). And, only a small fraction of the websites include instructions on how to verify file integrity with checksums or a description of their utility. Finally, our in-person experiments demonstrate 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 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 caused by a low number of fixations. The user feedback also shows a good desirability of verification mechanisms integrated in web browsers.

The rest of the paper 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

[1] Note that checksums only enable users to verify that the file they downloaded is indeed the one the website administrator intended to share.

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

verification in Section 3. We present the survey of websites that use checksums in Section 4. We follow up with the online user survey on Internet security behavior in Section 5. We then describe the proposed solutions in Section 6, and present the in situ user experiments with eye-tracking in Section 7. We discuss the main findings and limitations in Section 8. We conclude the paper in Section 9, by describing the outlook and perspectives for future work.

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. Hereafter we first present studies that focused on Internet users download behavior, then we focus on the works related to the effectiveness of security warnings and we then focus on the sub-area dedicated to file integrity verification.

2.1 Download Behavior

Internet users are increasingly exposed to security threats, as found by Furnell et al. [24]: They conducted a survey with 415 home users to assess their perceptions of security issues, and their attitudes towards the use of related safeguards. The survey revealed that, although the responders had a high degree of confidence, they lacked desirable knowledge of different safeguards that can increase their security. They also found that there were notable shortcomings among users who considered themselves "experts". A similar survey conducted more recently with 594 home computer users revealed that security-related behavior is 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]. Furthermore, a recent survey conducted in the UK on businesses and charity organizations [37] revealed that security knowledge of employees is the weakest link leading to many successful cyber-attacks and that often times enforcing security policies is ineffective. Download behavior is often also influenced by security recommendations, as studied by Redmiles et al. [34, 35]. The authors conducted semi-structured interviews and deployed a large-scale questionnaire to study security advice. In these studies, they found that users evaluate digital-security recommendations based on the trustworthiness of the source of the advice. When advised by knowledgeable peers, users might trust 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 a third-party app to users when they are about to perform an action on their system that could potentially have negative consequences. Such actions include downloading or opening a file containing a virus, visiting a website that contains malware or has phishing intents, 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 [7, 10, 16, 18, 27, 30, 36, 40, 42]. 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 they are, on the one hand, effective at reducing the rate at which users perform potentially harmful actions after they have been warned [7, 36, 40]. On the other hand, users tend to ignore such warnings due to their excessive frequency [11] and habituation effects [42]. In another work, Modic and Anderson [30] studied what makes a warning effective, and they have found that warnings should (i) contain a clear and non-technical description of a potential negative consequence and (ii) they should be given from a "position of authority" [30]. In addition to the content, the design matters as well, as shown in two separate instances: First, by Akhawe and Felt [7] who compared SSL warnings from two different web browsers and showed that users of one browser proceeded to potentially malicious websites twice as often as the users of the other web browser; second, by Bravo-Lillo et al. [11] 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 [20, 43, 51]. For instance, Fagan and Khan [20] show that most users who follow a security advice do so for security benefits, whereas those who do not follow it do so to avoid an inconvenience mostly related to the lack of time.

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 verifications. Hsiao et al. have compared the speed and accuracy of hash verification with various textual and visual representations [26]. Their between-subjects online 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 [39]. Their online 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 [44]. 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 [46]. 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 [6]. 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 [19]. 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 [1, 2]) 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.

Finally, digital certificates can be used to certify the authenticity and integrity of programs. Such a solution, however, has shortcomings including the fact that certificates are costly, that the problem of certificate validation remains, and that private keys (of developers and certification authorities) can be compromised [4, 48]. 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 focused on one aspect that was neglected by prior research: What is the behavior of the users when they are asked to verify file integrity? Instead of testing different design of the checksum, we focused on the process by which participants were comparing a checksum with the output of the hash functions and their overall understanding of it. This point is highly relevant for this area of research as finding that people have a hard time understanding the whole idea behind checksum, like we did, would make automating the verification process a more secure solution over designing simpler checksums. 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 people 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 Linux systems and they are used mainly by experienced users (e.g., users familiar with the terminal). Note that package managers are also subject to attacks [12].

3 SYSTEM AND THREAT MODEL

In this section, we describe the general system and adversarial model, as well as the technical background necessary to understand the work (i.e., checksums and digital signatures). Readers familiar with these concepts can skip the corresponding paragraphs.

3.1 System and Threat Model

We consider a website hosted on a given 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. It could be the operator of the external hosting platform or a hacker. In order to enable users to check the integrity of the files they download from the external server, when clicking on the link in the download page hosted on the original server, the download page contains a checksum of the program, generated as explained below.

3.2 Checksums and Digital Signatures

Checksum. 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., that the data has not been corrupted (e.g., when the data is transmitted or stored). 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 digits (i.e., 128 bits) to 128 digits (i.e., 512 bits). Cryptographic hash functions enjoy three core properties: pre-image resistance, second pre-image resistance, and collision resistance [33, 38]. 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 generate a huge number of different versions of a program (e.g., by varying a number of innocuous bytes such as strings in the program file) until it finds one with a hash that matches that of the original file. An adversary can perform a brute-force attack to forge a file with a hash that 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. That is, two checksums should be very different if applied to very similar (but not identical) data.

Today, the most popular cryptographic hash functions are: MD5, SHA-1, SHA-2 (with 256, 384 or 512 bits) and the upcoming SHA-3. MD5 was one of the first proposed cryptographic hash functions; it was broken in the late 1990's and its use is strongly discouraged.

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

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 [15].

For integrity verification, users must input the downloaded file to a dedicated program (e.g., `shasum`) and compare the computed checksum to the one specified on the download page.

Digital Signature. Digital signatures usually rely on asymmetric cryptography. The signer (e.g., the developer) has a pair of cryptographic keys: a public one, known to everyone, and a private one, kept secret. By using their private key, the signer can generate a fixed-size block of data, i.e., a (detached) digital signature, from a file. Based on the signer's public key, a user can verify the validity of the signature and thus the integrity (and authenticity) of the corresponding file. To do so, users must input the downloaded file, the signature and the public key of the signer to a dedicated program (e.g., `gpg`). A popular cryptographic standard for digital signature is OpenPGP[4] that relies on the RSA and DSA cryptographic schemes.

3.3 Subresource Integrity

Subresource integrity (SRI) was introduced by the W3C in 2016 [49]. 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`[6] 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 browser except Internet Explorer.

4 SURVEY OF WEBSITES

In order to assess the current practices of developers regarding the use of checksums and signatures[7] in the context of web downloads, we analyzed the download pages of various popular programs for computers (not for smartphones), such as VLC media player.

4.1 Methodology

To obtain a list of software programs to analyze, we wanted to focus on popular programs that can be freely downloaded on the Internet and that have an associated checksum on their download pages (on the official website). We started by looking at websites from the Alexa top 500 ranking, but we were unable to find any meaningful datasets, not to mention the fact that the number of visits on the website is not necessarily correlated with the number of downloads of the program. We then explored a second dataset from App Annie that provides a list of popular apps but only for mobile devices. A third possibility would have been to look at the official app stores, but this would make little sense, as those apps can already be safely downloaded from there. In addition, some popular apps are not on such app stores. Hence, we collected examples of free software programs from within our lab as well as from lists of popular free software gathered from the Web.[8] We converged on 20 programs that (i) can be freely downloaded from their respective websites, (ii) are provided with their associated checksums and (iii) are from different app categories, according to Wikipedia. We manually analyzed each of the download pages in a systematic way, by checking the following properties:

- If the checksum and the program are hosted on the same server, on different servers in the same domain, or on different servers in different domains. We compared the URLs of the checksum file (or of the webpage, if the checksum is embedded in the webpage) to that of the program. We performed a complementary comparison of the corresponding IP addresses.[9] From a security point of view, it is better to host the website and the program on different servers, as this reduces the risks that an attacker can tamper with both the checksum displayed on the page and program file. This is less problematic for signatures as they cannot be forged.
- Whether the checksum and the program are served through HTTP or HTTPS by default. From a security perspective, HTTPS is better as it protects the integrity of the data while in transit and authenticates the hosting website.
- Which cryptographic schemes are used to generate the checksum (i.e., MD5, SHA1, SHA2 with 256 or 512 bits, or PGP). As of today, both MD5 and SHA1 are considered insecure and their use is strongly discouraged [15].
- Whether the webpage contains instructions on how to verify the integrity of the downloaded program with the corresponding checksum.
- Whether our browser extension (described in Section 6) can extract the checksum from the original download page and verify the integrity of the downloaded program.

We collected, whenever available, the average number of downloads per day. We conducted the survey in April 2018. To limit the possible data collection errors, three experts from our lab independently verified each website according to the properties above and found no discrepancies. Table 1 shows the raw data of this survey. Note that the Tor client was included in our survey, even though it does not include a checksum (but includes a signature). This is because the integrity of such a program is particularly important, considering the contexts in which it is used (e.g., censorship circumvention).

4.2 Results and Analysis

It is important to note that the list of analyzed programs is not representative of all the programs available for download on the Web. Therefore, the statistics derived from this survey are useful mostly to obtain a sense of the current practices and the relevant criteria regarding the use of checksums for securing web downloads of programs. For future work, we intend to perform such an analysis

[4] https://www.openpgp.org

[5] https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

[6] The `link` element should not be confused with the `a` element which defines hyperlinks users can click on to download files or to navigate to other webpages.

[7] For the sake of clarity, in this section, we refer to both digests produced by cryptographic hash functions and signatures as checksums.

[8] See, e.g., https://www.techradar.com/news/the-best-open-source-software

[9] Note that this method is a heuristic; as such, it could induce both false positives and false negatives.

name	description	host	checksum	program	MD5	SHA1	SHA2	PGP	ext.	instr.	#dl/day
Android Studio	programming suite	diff. domain	https	https	✗	✗	✓	✗	✓	✗	-
Audacity	audio editor	diff. domain	http	https	✗	✗	✓	✗	✗	✗	6k
Blender	3D graphics editor	diff. domain	https	https	✓	✗	✗	✗	✗	✗	16k
FileZilla	FTP client	diff. domain	https	https	✗	✗	✗	✗	✗	✗	11k
GIMP	raster graphics editor	diff. domain*	https	https	✓	✗	✗	✗	✓	✗	-
GnuPG	cryptographic suite	same server	https	https	✗	✓	✗	✓	✗	✓	-
Handbrake	video transcoder	diff. domain*	https	https	✗	✓	✓	✓	✓	✓	-
Inkscape	vector graphics editor	same server	https	https	✓	✗	✗	✗	✗	✗	-
IntelliJ	programming suite	same server	https	https	✗	✗	✓	✗	✗	✗	-
KeePass	password manager	diff. domain	https	https	✗	✗	✗	✗	✗	✗	15k
Notepad++	text editor	same server	https	https	✓	✓	✗	✗	✗	✗	-
OpenOffice	office suite	diff. domain	https	https	✓	✗	✓	✓	✗	✓	85k
Plex	media server	same server	https	https	✗	✗	✓	✗	✓	✗	-
RealVNC	remote administration	same server	https	https	✗	✗	✓	✗	✓	✗	-
SpyBot	antivirus	same server	https	https	✓	✓	✓	✗	✗	✗	-
Tor	anonymity network	same server	https	https	✗	✗	✗	✓	✗	✓	90k
Transmission	BitTorrent client	diff. domain	https	https	✗	✗	✓	✗	✓	✗	-
Ubuntu	operating system	same server	http	http	✓	✓	✓	✓	✗	✓	-
VLC	media player	diff. domain	https	https	✗	✗	✓	✗	✓	✗	527k
VMware Fusion	virtual machine hypervisor	diff. server	https	https	✓	✓	✓	✗	✗	✓	-

Table 1: Raw results of the website survey on the use of checksum on the download pages of popular programs. "ext." stands for "extension", "instr." for "instructions", and "#dl/day" for "average number of downloads per day". In the host column, the "*" denotes that, although the program and the checksum files are hosted on the same server, they are also mirrored on a different server (typically under a different domain).

on a much larger scale, which would also enable us to assess the prevalence of checksums on the web.

From the results, we observe the following. First, 2 out of 20 (2/20) websites did not serve the checksum in a secure way (i.e., over HTTPS). This is problematic, as the webpage and the checksum it contains could be tampered with by an adversary through a man-in-the-middle attack. It should be noted, however, that, for Audacity, even though the website was by default served over HTTP, it was also possible to retrieve it over HTTPS. Second, 9/20 websites host the checksum and the program on the same server (according to our heuristic). This could be problematic, as an adversary who is able to break into the server could tamper with both the program file and its checksum, hence cover up the tampering.

We also observed that 7/20 websites include multiple checksums (e.g., Handbrake with SHA1 and SHA2), possibly in combination with PGP signatures (e.g., OpenOffice). A non-negligible fraction (5/20) of the websites analyzed (e.g., Blender and GIMP) include only checksums generated from weak hash functions, namely MD5 and SHA1. Furthermore, only 6/20 websites (e.g., OpenOffice and GnuPG) include instructions on how to verify checksums or a description of their utility. Finally, the fact that our extension does not work on 13/20 analyzed websites is due to the fact that it is currently instrumented to work in the cases were the checksum is available in the HTML code of the same website that contains the link to the program file. If the checksum is stored in a separate file, our extension would currently not detect it. As for Tor, the download page includes a detached signature (hosted on the same server, but this is not a problem for signatures) together with the instructions to check it.

In summary, due to frequent flaws in the way checksums are currently used (e.g., insecure communication, single server, weak hash function) and the lack of details on their utility and how-to guides, checksums do not achieve their full potential in securing web downloads.

5 LARGE-SCALE USER SURVEY

To date, little research has focused on the problem of checksum verification. In particular, we know little about the proportion of Internet users who know about checksums and those who regularly apply checksum verification to their downloads. Therefore, we pose the following research questions: (RQ1) *Which proportion of Internet users install programs that are manually downloaded directly from websites, therefore exposing themselves to potentially corrupted programs?* (RQ2) *What proportion of Internet users know about checksum verification methods? Among those who know, how many actually do it when they download programs from developer websites?*

5.1 Methodology

In order to study the current security behaviors of Internet users, and the associated risks, with respect to programs downloaded from the Internet and to answer the above research questions, we conducted a large-scale online user survey. The survey was approved by our institution's ethics committee. We contracted a vendor to deploy and conduct the survey; the vendor was in charge of selecting a representative sample, in terms of demographics, of Internet users based in the US [23]. The panelists were recruited via partnerships and invited via banners and messaging, and then go through quality controls. Panelists receive virtual points in exchange for

their participation in active research campaigns, which can be later redeemed for donations to charity or for vouchers to buy goods and services. The total cost of the survey was ~USD 5600 with an average incentive of ~USD 1.9/participant (excluding administrative costs).

We deployed the questionnaire twice in order to collect both qualitative and quantitative information from participants while avoiding potential biases in the answers. The first deployment (Q1) lasted one week and some of the questions were left as open ended. We collected responses from around 200 participants. Qualitative answers provided for Questions A5, A7, and A10 (see Appendix A) were categorized at the end of the week by two coders. The codebook was developed through inductive analysis and it was used to define multiple-choice questions for the second deployment of the questionnaire (Q2). We measured inter-coder agreement through Cohen's kappa at 0.89 and judged it sufficient. The few cases of conflict were resolved via discussion. Before deployment of Q1, to improve the effectiveness and readability of the questions, we conducted a cognitive pretest of the questions with 5 test participants. For instance one of the changes consisted in simplifying language. Instead of using technical terms such as "checksums", we mainly used: "sequences of numbers and characters". The questionnaire Q2 was deployed in April 2018. It should be noted that, as for most user surveys on security-related behavior, the information reported by the respondents might not exactly reflect their actual behaviors; this fact is well documented in the literature (see for instance Egelman et al. [17]).

5.2 Demographics and General Statistics

Two thousand valid responses were collected (N = 2,000) from questionnaire Q2. The proportion of female respondents was 51% and the age distribution was as follows: 18-29 (22%, or 440 respondents), 30-39 (17%, or 340), 40-49 (19%, or 380), 50-59 (18%, or 360), 60+ (24%, or 480). Respondents were well distributed across the four macro regions of the US: 22.1% (or 442) live in the Midwest, 20.1% (or 401) live in the Northeast, 34.2% (or 683) live in the South, and 23.7% (or 474) live in the West.

The large majority of the respondents (90.6%) use a laptop or desktop computer; the remaining 9.4% (or 189) use only a smartphone or tablet. Of those who use a computer, 75.9% use a computer running Windows, 12.4% use a computer running macOS, and 2.3% use a computer running Linux (A1). For the remaining of the statistics in this section, we will always refer to respondents who use a computer. We found that 89.7% of the respondents owned a smartphone, 8.5% owned a feature phone, and 1.8% did not own a mobile phone. We leave studying downloads of mobile apps to future work.

5.3 Results and Analysis

RQ1. In our survey (A4), 29.4% of the respondents declared to never run any program downloaded from the Internet or to do so from official app stores exclusively (6.1%), which could be considered a safe behavior. Out of the remaining 70.6% of the respondents, 58.6% declared downloading content from developers' websites, and 42.6% using P2P networks at least once a year. The majority of respondents using computers (62.2%) declared using official app stores (e.g., Mac App Store) for downloading programs. However, only 6.1% of respondents used this source exclusively. These results reveal that the large majority of Internet users are exposed to potential corruption of externally hosted programs, thus confirming the relevance of this research.

RQ2. For a final question, we asked (A9) our respondents whether they had ever noticed checksums on websites and (A10) what they would do with them if they ever found them when downloading programs. We found that 23.4% of respondents remembered seeing them on websites they used in the past. Concerning what they would do with them, most selected responses had nothing to do with what checksums are meant for (83% see Question A10). Interestingly, for 18.2% of the respondents, displaying the checksums on the webpage of the app would make them doubt of the website and search for something else. Consequently, using this security technology would be detrimental to the overall user experience. About 11.8% of the respondents would simply ignore checksums and continue installing the app. Regarding the purpose of checksums, only 5.2% of the respondents selected the correct answer (out of 6 possible options, plus the "not sure" and "other" options), i.e., to check the integrity of the downloaded programs. Therefore, we can estimate the proportion of users who know about checksums between 1.7% and 5.2%.

5.4 Summary

The results of the large-scale survey revealed that, based on Internet user behaviors, corruption of externally hosted programs could have a substantial negative effect. Integrity verification offers a defense against this. Unfortunately, only a tiny fraction of Internet users knows about this security technology and uses it on a regular basis. Developers, who rely on checksums, currently require users to perform the check *manually*. We learned, however, that this might be perceived negatively by less experienced users and might lead to users preferring a different developer.

5.5 Limitations

After the deployment of the survey, we realized that some of the terminology and the wording of the questions could have led to an ambiguous interpretation. For instance, in question A4, we did not provide any examples of P2P programs. Furthermore, the distinction between 'developer' and 'vendor' might not have been entirely clear to the respondents.

6 AUTOMATING CHECKSUM VERIFICATION

In the previous sections, we demonstrated through various surveys and experiments that checksums currently do not fully achieve their goals of securing web downloads. One of the main causes is that the task of computing and verifying checksums needs to be done manually by the users. In addition, checksums are not widespread on the web and Internet users are unaware of their utility and usage. In this section, we address these problems by proposing both recommendations and technical solutions that we designed, implemented and made available for testing (see below). Our solutions address the problems of generating, computing and verifying checksums.

6.1 Extending Subresource Integrity to Links

A direct solution for making checksum verifications automatic is to extend the subresource integrity (SRI) feature [49], recently introduced by the W3C and described in Section 3, to HTML `a` elements (i.e., hyperlinks) that point to files to be downloaded.

Our proposal is to include an `integrity` attribute to 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 hyperlink that specifies, in an `integrity` attribute, the checksum of the file it points to.[10]

```
<a href="https://github.com/.../Transmission-2.93.dmg"
    integrity="sha256-Yc2bdMxUJFj...">download</a>
```

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

6.2 Checksum Verification: Browser Extension

As web browsers do not currently handle SRI for hyperlinks, to automatically check the integrity of downloaded files, we developed a Chrome extension[11]. This extension should, of course, 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,[12] and PGP signatures (partially). It is implemented in JS and it relies on the md5.js library for computing MD5 digests,[13] the asmcrypto.js library for computing SHA digests,[14] and the openpgp.js library for checking PGP signatures. 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 digests of the downloaded files.

As SRI for hyperlinks is currently not supported by web developers, our extension is also capable of extracting 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 extract, by using regular expressions, hexadecimal strings that have the same format as checksums as well as the corresponding hash function names (e.g., SHA-1).
(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 hyperlink points to a file with a sensitive extension (e.g., dmg, exe, pkg) and/or mime type[15] (e.g., application/x-apple-diskimage, application/x-msdownload, application/x-debian-package),[16] 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.[17] The webpage is greyed out and a pop-up message is displayed to the user, as illustrated in Figure 1. Additionally, if the checksum originates from the text of the webpage, the matching text with the checksum is revealed (if originally hidden) and highlighted.

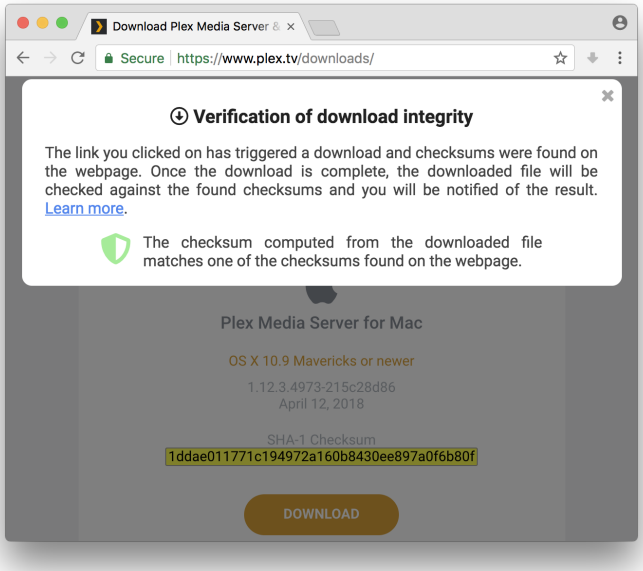


Figure 1: Screenshot of the extension on the Plex download page (https://www.plex.tv/downloads/). The checksum of the downloaded file is computed and successfully checked against that extracted from the webpage (highlighted).

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 2), 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 [7]. For the initial proof of concept, we experimented with the four messages shown in Figure 2. A careful consideration of alternatives that incorporate user feedback should be conducted

[10] For PGP, the integrity attributes include the string "pgp", the ID and fingerprint of the PGP key used for generating the signature, and the base-64 representation of the detached signature of the file.
[11] Ideally, such verifications should be performed by the web browser. One option would be to integrate directly in the Chromium open-source browser project.
[12] We chose to support the MD5 and SHA-1 functions despite their known weaknesses because they are still used, as mentioned in Section 4.
[13] https://github.com/blueimp/JavaScript-MD5
[14] https://github.com/asmcrypto/asmcrypto.js
[15] The mime type is determined by issuing a HEAD request to the target of the link.
[16] The complete list was built based on the extension-mime types mappings of the Apache and nginx web servers.
[17] 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, 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.

before the actual deployment in a product. We leave the careful design of the extension user interface for future work.

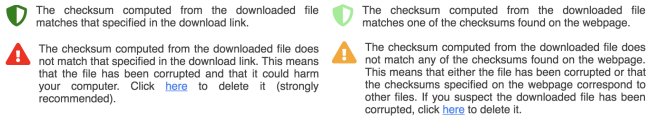


Figure 2: Messages displayed by the browser extension: left (`integrity` attribute) / right (text of the webpage), top (success) / bottom (failure).

The extension can be downloaded from at the following address: https://checksum-lab.github.io/chrome_extension.zip.[18] A test webpage can also be found at the following address: https://checksum-lab.github.io/. It contains test download links with and without (valid/invalid) 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 at the following address: https://checksum-lab.github.io/demo.mp4.

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 7.4 for more details) and best practices for the design of security warnings (see for instance [7, 8, 11, 16, 18, 21, 30, 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 linked on the download page. Furthermore, it does not include any settings UI for managing the features, such as disabling the extension on certain websites or making the verification process silent (e.g., do not show any message and simply delete the file in the background if the checksum does not match). Finally, the extension currently does not alert users when no checksums are available on the website, for any of the downloaded files. Finding the optimal way of informing users of such aspects is another design problem we intend to address by reviewing the best practices for security warning design, and by conducting additional user studies.

6.3 Checksum Generation: CMS Extension

In order for web browsers to automatically verify checksums as described above, checksums must be embedded in download links. Therefore, to alleviate the burden on the website administrators, we developed a tool to automatically generate and embed checksums in download links for websites that are managed through a Content Management Systems (CMS).

Design and Implementation. Because a substantial fraction of websites are powered by content management systems (CMS), and by WordPress in particular,[19] we implemented our tool in the form of a WordPress extension.

Our WordPress extension is implemented as a hook for the page/post update operation. Similarly to the Chrome extension, it should be considered as a proof of concept. It consists of ~100 lines of PHP code; it can be downloaded at the following address for testing purposes: https://checksum-lab.github.io/wordpress_plugin.html. The extension parses the HTML code of the page and extracts all the links (i.e., the `a` elements). For each link, the extension determines the mime type of the target by making a HTTP HEAD request on the server side. If the target is downloadable (i.e., not a webpage, or more generally, content that is not displayed in the browser), the extension fetches it (by making a HTTP GET request on the server side), computes the SHA-2 checksum and embeds it in the corresponding link, i.e., in an `integrity` attribute of the `a` element. For performance reasons, the computed checksums are cached in a database. Yet, they can be refreshed on demand, in order to adapt to potential changes of the linked file at different points in time. Moreover, the extension can be further enhanced to automatically re-compute the checksums for the target of the links (i.e., the `a` element) and to alert the website administrator in case of changes (caused by file update or corruption). It should be noted that downloading the file on the server side could be avoided by using the `content-md5` [22] or `instance digest` [31] feature of HTTP. Unfortunately, because the former is deprecated and the latter is not standardized, neither are supported by major HTTP servers. In the production version of the extension, we intend to implement the download and the checksum computation in an asynchronous fashion and to include a configuration widget in the HTML editor that is embedded into WordPress.

7 USER EXPERIMENT

In this section, we study the usability and the effectiveness of checksums in the context of web downloads. More specifically, we pose the following research questions: (RQ3) *Do users thoroughly verify checksums?*, (RQ4) *Can users be fooled by replacing some characters in the middle of the checksum?*, (RQ5) *Does automating the checksum verification improve general usability metrics?*

In order to answer the above questions, we conducted an in situ user experiment involving an eye-tracking screen. This methodology 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 [32]. The two metrics extracted through this method were 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 [25]. A longer dwell time indicates difficulty in extracting information, or it means that the object is more engaging in some way [28]. Our hypothesis was that participants who checked thoroughly the checksums would have to produce more fixations (and

[18] Instructions: Download the zip file and unzip it. In the extension tab of Chrome, activate developer mode and load the extension by clicking on the "Load unpacked" button and selecting the folder where the extension was unzipped.

[19] WordPress is the most widely used CMS; according to W3Tech (https://w3techs.com/technologies/overview/content_management/all; Last visited: March 2018), about 51% of all websites are powered by a CMS and 31% of all websites are powered by WordPress.

spend more time fixating) in the part of the user interface where these sequences were displayed.

The experiment was split in two phases. During the first phase, we asked participants to verify manually the checksums of four downloaded apps (this was addressing RQ3 and RQ4). In the second part of the experiment, we activated a browser extension that took care of verifying the integrity of the downloaded files based on their checksums (this was addressing RQ5). We did not counterbalance the presentation of these two parts for two reasons: (i) the second part of the experiment made explicit what the core of the experiment was and could have biased the results of the first part; (ii) the two parts of the study addressed different research questions. However, this design also had drawbacks that we report below in Section 7.4. The experiment was approved by our institution's ethics committee.

7.1 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 how and what for the elements circled in red on the following screenshots are used?[20] If yes, please describe it briefly in the text box below."), tech 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 OS 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. We number of participant was chosen so that it provides sufficient power to the statistical tests and keeps the total duration of the experimentations 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 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).

7.2 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[21]) 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 (either 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 3). 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 (AOI) 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.[22]

7.3 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 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. All materials were prepared in French and the sessions were conducted in French, which the main language spoken where the study was conducted.

First Phase. We asked the participants to download from the official website and execute/install four different programs (in this

[20] The screenshot depicted the official VLC download page with checksums circled.
[21] https://www.tobiipro.com/product-listing/tobii-pro-x2-60/

[22] We considered, as in [39], that a realistic adversary can forge, through brute-force, a corrupted program in such a way that the first and last 10 digits of its checksum match those of the original program's checksum.

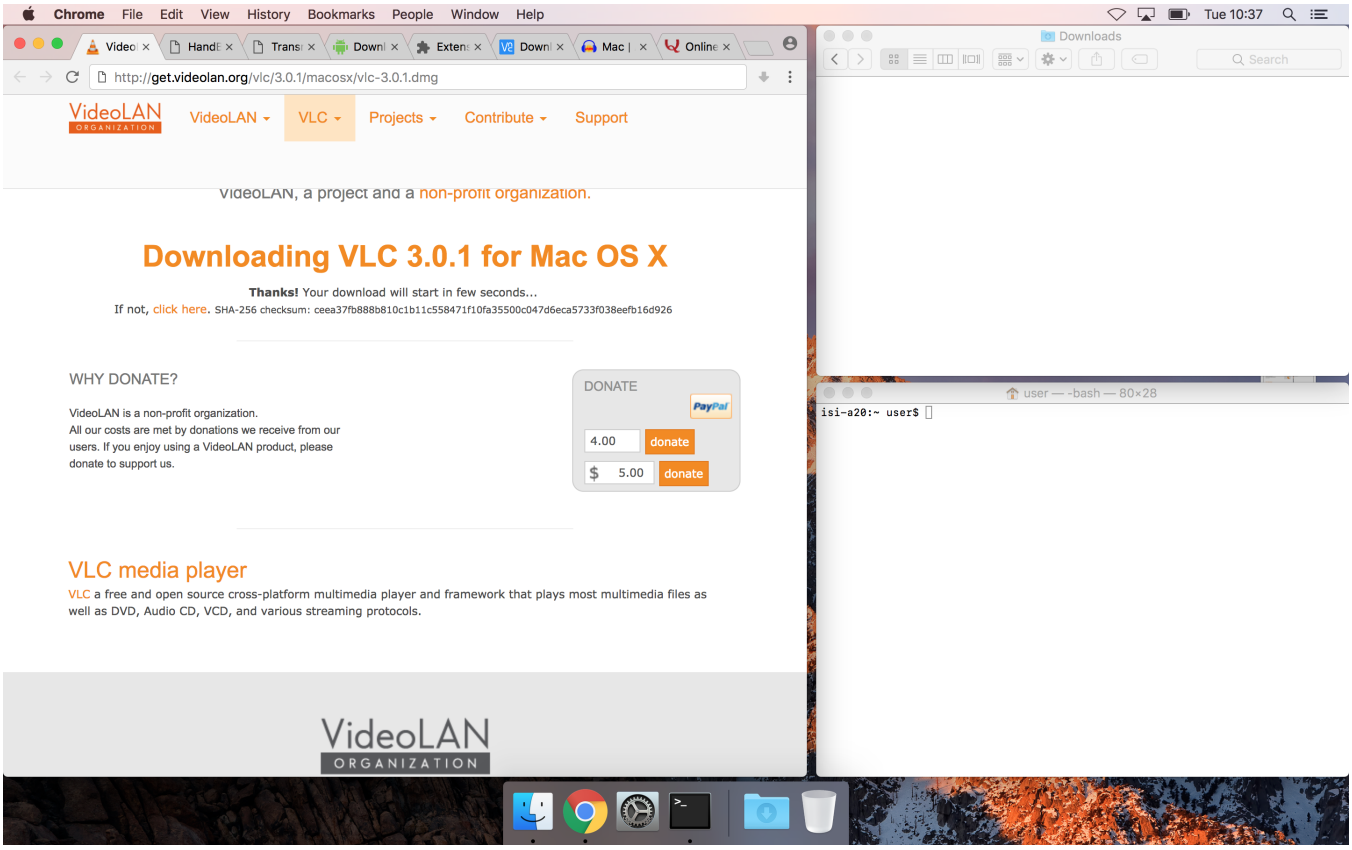


Figure 3: Screenshot of the window arrangement on the computer used for the experiment (macOS). The left half of the screen is occupied by the Chrome web browser in which multiple tabs have been pre-opened: the download pages of the first four programs, the extension tab to activate the extension, the download pages of the next two programs, and the questionnaire website (Qualtrics) 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.

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.[23] 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.

[23]The `clear` command is used to ensure that the checksum is always displayed at the same location on the screen, for eye-tracking purposes.

As explained in the previous subsection, the checksum of the third webpage, i.e., Transmission, was set to be incorrect.

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 (see Figure 1), whereas the second one was incorrect, hence resulting in the display of a warning message (see at the bottom right of Figure 2). The terminology used in the messages was inspired by the instructions found on the download pages of popular programs (e.g., Ubuntu). We recognize that it could be improved with a better design and with user feedback. We intend to revise the design of the extension in the future.

(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)

Figure 4: Sample subject gaze heat maps captured by the eye-tracking system on macOS.

For the last step, we asked the participants to fill a short questionnaire online to get some feedback about their perception of the manual verification of checksums and of the browser extension, satisfaction with the extension and net promoter score.

7.4 Results

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

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. These were the AOIs described prior in the text. Unfortunately, we had to remove eye-tracking recording for one participant due to corrupted data.

Figure 5: Areas of interest used for the checksums displayed in the terminal.

RQ3. 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 4. 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.

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 (see Figure 5) 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.[24]

[24]Concerning the total number of fixations, the Shapiro-Wilk normality tests were close to rejection: AOI 1 - ($W = .95$, $p = 0.085$), AOI 2 - ($W = .94$, $p = 0.027$), AOI 3 - ($W = .94$, $p = 0.037$), AOI 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.

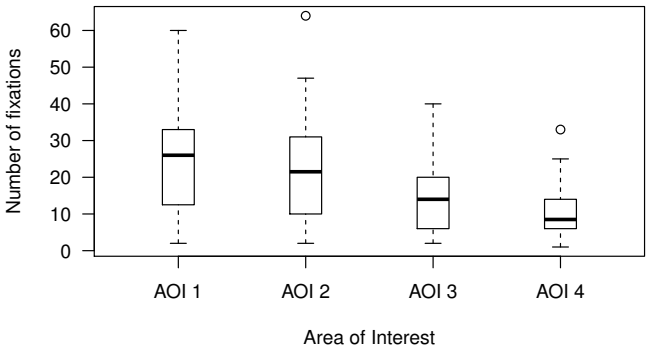


Figure 6: Boxplots of the distribution of user fixations across the four areas of interests covering the checksums.

AOI	1	2	3	4
1	-	445**	756***	773***
2	-	-	709***	688***
3	-	-	-	518***
4	-	-	-	-

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

Table 2: Wilcoxon signed rank tests of the number of fixations within the four areas of interest covering the checksums. Due to ex aequos in the data, the p-value is an approximation.

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. There was a significant difference in the scores: AOI 1 - M=25.15, SD= 13.11, AOI 2 - M=21.92, SD= 13.96, AOI 3 - M=13.92, SD= 9.55, and AOI 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 AOI. We include the detailed results of the tests and the boxplot of the distribution of fixations for each AOI, in Figure 6 and Table 2. 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.

RQ4. 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. We did not find a significant difference in the detection rate for participants who had prior checksum knowledge ($p = 1$, Fisher's exact test). 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 – specifically the lack of difference between the group of participants who were knowledgeable and those who were not – was observed for the previous results on RQ3.

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. A Wilcoxon rank sum test was conducted to compare the total number of fixations in the AOIs for the two groups. The values of the task with the modified 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 detect the mismatch. 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 (i.e., those who detected the mismatch vs. those who did not). We observed the same difference reported for Task 3; this reveals that those who were thorough were consistently so, during the entire experiment.

RQ5. We now report the results of our user experiment related to the browser extension carried out in the second phase. As explained in Section 7, 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 (with eye tracking), 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 to delete the (corrupted) downloaded file and the remaining 62.5% manually removed the file.

In the exit survey, the participants reported an average satisfaction score of 5.2±1.4 (on a scale from 1 to 7).[25] 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 part of the experiment.

Another observation from the user experiment was that 26/40 participants (65%) could not explain the objective 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.

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. 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.[26] Also, they mentioned that a warning originating directly from the browser in a standalone window would have been more effective.

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*

[25] For all the self-reported scores given in this section, we used a 7-level Likert scale.

[26] Showing fake (security) warnings within webpages to push users to download and install malicious programs is a common practice, e.g, fake antivirus or flash players.

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.) These results suggest that the extension was rated slightly desirable by most participants. 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 (see Section 8).

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. Also, we might have introduced a learning bias by choosing not to randomize the presentation of the first and the second part of the study might have .

8 DISCUSSION

Our large-scale user survey shows that the majority of computer users are vulnerable to attacks through corrupted download files. Checksum is one of the most prevalent solutions for countering such attacks. However, most users have a hard time grasping the idea behind integrity verification. Through the survey, we learned that users neither understand nor use them (95% do not know how to use them, 98% do not use them for verifying a file integrity). Through the user experiment, we also consolidated this finding: Even when users were explicitly asked to verify checksums, most of them have had a hard time understanding what they were doing and often failed at detecting strategic replacements (38% of the time). Furthermore, after having spent over 30 minutes manually verifying checksums, looking at our extension and answering questions related to checksums, 65% of the participants still had troubles explaining what was the purpose of checksums. Therefore, we argue that raising awareness around manual checksums verification would require substantial efforts.

In this paper, we argue that a viable solution to increase the security and usability of web downloads is to automate the checksum verification process. Through the lab study, we tested an initial design for an extension that would automate the integrity verification. In our initial design the user was left in charge of deciding whether the downloaded file (for which there was a mismatch) had to be removed from the system. Unfortunately, we learned that this design was not really effective at making people aware of potential threats. This finding resonates well with the suggestions of Tan et al. [44]. As highlighted by prior work, security warnings might be skipped for a variety of reasons: notifications overload, habituation [42], inattention blindness [47], etc.

We argue that designers should choose defaults to privilege users' security over preserving user's choice. In fact, a lack of action from the user does not necessarily mean that she is willing to accept the risk of proceeding but might simply be due to the aforementioned effects. Concretely, in the case of a mismatch, instead of warning the users and suggesting them to delete the corrupted files (opt-in approach), the extension should block access to the downloaded files a priori and ask for users' action to unblock it. This opt-out approach is similar to the one implemented by antivirus or spam filters which put files or e-mails in quarantine.

9 CONCLUSION & FUTURE WORK

In this work, we opened a line of research on the use of checksums for integrity verification of web downloads and made a number of contributions.

Given the user feedback and results of our in situ experiments, we plan to improve the user interface of our extension to make it clearer about the potential risks of installing corrupted download files. More specifically, we will deploy an instrumented beta version of our extension and run a longitudinal study to gather more feedback from the end users and at a larger scale. To do so, we will rely on the well-established experience sampling method that collects in situ data by requesting participants to provide short self-reports at multiple random occasions over time [13, 14]. This will also enable us to collect data about checksums in the wild.

Finally, we believe it is paramount to robustify and improve the software artifacts (browser and CMS extensions) we produced and make them available to a wide audience. In addition we intend to promote our proposals to (and collaborate with) the different stakeholders involved, that is the W3C, web browsers (e.g., Google, Mozilla) and CMS (e.g., WordPress, Drupal) development teams in order to have a concrete impact on the security of Internet users.

ACKNOWLEDGMENTS

The authors express their sincere gratitude to Italo Dacosta, Andreas Kramm, Nicolas Le Scouarnec, Adrienne Porter Felt, Lawrence You, Blase Ur (our shepherd) and the anonymous reviewers for their insightful feedback. The authors also warmly thank Holly Cogliati for her great editing job on the manuscript. The work was partially funded with a UNIL-HEC Lausanne Research Fund.

REFERENCES

[1] [n. d.]. Checksum On the Go - Chrome Webstore. https://chrome.google.com/webstore/detail/checksum-on-the-go/fholnooplijidhdagedffljaphholpea.
[2] [n. d.]. 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.
[3] [n. d.]. 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.
[4] 2016. Certificates and Digitally Signed Applications: A Double Edged Sword. https://eventtracker.com/tech-articles/certificates-and-digitally-signed-applications-a-double-edged-sword/.
[5] 2016. Transmission Hijacked Again to Spread Malware. https://blog.malwarebytes.com/threat-analysis/2016/09/transmission-hijacked-again-to-spread-malware/.
[6] Ruba Abu-Salma, M. Angela Sasse, Joseph Bonneau, Anastasia Danilova, Alena Naiakshina, and Matthew Smith. 2017. Obstacles to the Adoption of Secure Communication Tools. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, 137–153. https://doi.org/10.1109/SP.2017.65
[7] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness.. In *Proc. of the USENIX Security Symp. (USENIX Security)*, Vol. 13. USENIX.
[8] Bonnie Brinton Anderson, C. Brock Kirwan, Jeffrey L. Jenkins, David Eargle, Seth Howard, and Anthony Vance. 2015. 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, 2883–2892. https://doi.org/10.1145/2702123.2702322
[9] Catherine L. Anderson and Ritu Agarwal. 2010. Practicing Safe Computing: A Multimedia Empirical Examination of Home Computer User Security Behavioral

Intentions. *MIS Q.* 34, 3 (Sept. 2010), 613–643.
[10] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. 2015. 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, 931–948. https://doi.org/10.1109/SP.2015.62
[11] Cristian Bravo-Lillo, Saranga Komanduri, Lorrie Faith Cranor, Robert W. Reeder, Manya Sleeper, Julie Downs, and Stuart Schechter. 2013. 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, New York, NY, USA, 6:1–6:12. https://doi.org/10.1145/2501604.2501610
[12] Justin Cappos, Justin Samuel, Scott Baker, and John H. Hartman. 2008. A Look in the Mirror: Attacks on Package Managers. In *Proc. of the ACM Conf. on Computer and Communications Security (CCS)*. ACM, 565–574. https://doi.org/10.1145/1455770.1455841
[13] Juan Pablo Carrascal, Christopher Riederer, Vijay Erramilli, Mauro Cherubini, and Rodrigo de Oliveira. 2013. Your Browsing Behavior for a Big Mac: Economics of Personal Information Online. In *Proc. of the ACM Int'l Conf. on the World Wide Web (WWW)*. ACM, 189–200. https://doi.org/10.1145/2488388.2488406
[14] Mauro Cherubini and Nuria Oliver. 2009. A Refined Experience Sampling Method to Capture Mobile User Experience. *arXiv:0906.4125 [cs]* (June 2009). arXiv:cs/0906.4125
[15] Information Technology Laboratory Computer Security Division. [n. d.]. NIST Policy on Hash Functions - Hash Functions | CSRC. https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions.
[16] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. 2008. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 1065–1074. https://doi.org/10.1145/1357054.1357219
[17] Serge Egelman, Marian Harbach, and Eyal Peer. 2016. Behavior Ever Follows Intention?: A Validation of the Security Behavior Intentions Scale (SeBIS). In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 5257–5261. https://doi.org/10.1145/2858036.2858265
[18] Serge Egelman and Stuart Schechter. 2013. The Importance of Being Earnest [in Security Warnings]. In *Proc. of the Int'l Conf. on Financial Cryptography and Data Security (FC)*. Springer, 52–59. https://doi.org/10.1007/978-3-642-39884-1_5
[19] Elham Vaziripour, Justin Wu, Mark O'Neill, Ray Clinton, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. 2017. 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.
[20] Michael Fagan and Mohammad Maifi Hasan Khan. 2016. 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, 59–75.
[21] Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL Warnings: Comprehension and Adherence. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2893–2902. https://doi.org/10.1145/2702123.2702442
[22] R. Fielding and J. Reschke. 2014. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* RFC 7231. RFC Editor. http://www.rfc-editor.org/rfc/rfc7231.txt.
[23] Thom File and Camille Ryan. 2013. *Computer and Internet Use in the United States.* Technical Report ACS-28. U.S. Census Bureau.
[24] S. M. Furnell, P. Bryant, and A. D. Phippen. 2007. Assessing the Security Perceptions of Personal Internet Users. *Computers & Security* 26, 5 (Aug. 2007), 410–417. https://doi.org/10.1016/j.cose.2007.03.001
[25] Joseph H. Goldberg, Mark J. Stimson, Marion Lewenstein, Neil Scott, and Anna M. Wichansky. 2002. Eye Tracking in Web Search Tasks: Design Implications. In *Proc. of the Symp. on Eye Tracking Research & Applications (ETRA)*. ACM, 51. https://doi.org/10.1145/507072.507082
[26] Hsu-Chun Hsiao, Yue-Hsun Lin, Ahren Studer, Cassandra Studer, King-Hang Wang, Hiroaki Kikuchi, Adrian Perrig, Hung-Min Sun, and Bo-Yin Yang. 2009. A Study of User-Friendly Hash Comparison Schemes. In *Proc. of the Computer Security Applications Conf. (ACSAC)*. IEEE, 105–114. https://doi.org/10.1109/ACSAC.2009.20
[27] Jeffrey L. Jenkins, Bonnie Brinton Anderson, Anthony Vance, C. Brock Kirwan, and David Eargle. 2016. More Harm than Good? How Messages That Interrupt Can Make Us Vulnerable. *Information Systems Research* 27, 4 (2016), 880–896.
[28] Marcel Adam Just and Patricia A Carpenter. 1976. Eye Fixations and Cognitive Processes. *Cognitive Psychology* 8, 4 (Oct. 1976), 441–480. https://doi.org/10.1016/0010-0285(76)90015-3
[29] Swati Khandelwal. 2018. Flaw in Popular Transmission BitTorrent Client Lets Hackers Control Your PC Remotely. https://thehackernews.com/2018/01/bittorent-transmission-hacking.html.
[30] David Modic and Ross Anderson. 2014. Reading This May Harm Your Computer: The Psychology of Malware Warnings. *Computers in Human Behavior* 41 (2014), 71–79.
[31] J. Mogul and A. Van Hoff. 2002. *Instance Digests in HTTP.* RFC 3230. RFC Editor.
[32] Alex Poole and Linden J Ball. 2006. Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects. In *Encyclopedia of Human Computer Interaction*. 13.
[33] Bart Preneel. 1994. Cryptographic Hash Functions. *Transactions on Emerging Telecommunications Technologies* 5, 4 (1994), 431–448.
[34] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. 2016. 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, Vienna, Austria, 666–677. https://doi.org/10.1145/2976749.2978307
[35] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. 2017. 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, 931–936. https://doi.org/10.1145/3025453.3025673
[36] Robert W. Reeder, Adrienne Porter Felt, Sunny Consolvo, Nathan Malkin, Christopher Thompson, and Serge Egelman. 2018. An Experience Sampling Study of User Reactions to Browser Warnings in the Field. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, 512:1–512:13. https://doi.org/10.1145/3173574.3174086
[37] Vaidya Rishi. 2018. *Cyber Security Breaches Survey 2018.* Survey. United Kingdom.
[38] Phillip Rogaway and Thomas Shrimpton. 2004. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Proc. of Int'l Workshop on Fast Software Encryption (FSE) (Lecture Notes in Computer Science)*. Springer, 371–388. https://doi.org/10.1007/978-3-540-25937-4_24
[39] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. An Empirical Study of Textual Key-Fingerprint Representations. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX.
[40] Mario Silic and Andrea Back. 2017. Deterrent Effects of Warnings on User's Behavior in Preventing Malicious Software Use. Proceedings of the 50th Hawaii International Conference on System Sciences (2017).
[41] Mario Silic, Jordan Barlow, and Dustin Ormond. 2015. 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.
[42] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying Wolf: An Empirical Study of SSL Warning Effectiveness.. In *Proc. of the USENIX Security Symp. (USENIX Security)*. USENIX, 399–416.
[43] Leona Tam, Myron Glassman, and Mark Vandenwauver. 2010. The Psychology of Password Management: A Tradeoff between Security and Convenience. *Behaviour & Information Technology* 29, 3 (2010), 233–244.
[44] Joshua Tan, Lujo Bauer, Joseph Bonneau, Lorrie Faith Cranor, Jeremy Thomas, and Blase Ur. 2017. Can Unicorns Help Users Compare Crypto Key Fingerprints?. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI)*. ACM, 3787–3798. https://doi.org/10.1145/3025453.3025733
[45] Karen Turner. 2016-07-15T05:07-500. Developers Consider Apple's App Store Restrictive and Anticompetitive, Report Shows. *Washington Post* (2016-07-15T05:07-500).
[46] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. 2015. SoK: Secure Messaging. In *Proc. of the IEEE Symp. on Security and Privacy (S&P)*. IEEE, 232–249. https://doi.org/10.1109/SP.2015.22
[47] D. Alexander Varakin, Daniel T. Levin, and Roger Fidler. 2008. Unseen and Unaware: Implications of Recent Research on Failures of Visual Awareness for Human-Computer Interface Design. *Hum.-Comput. Interact.* 19, 4 (Dec. 2008), 389–422. https://doi.org/10.1207/s15327051hci1904_9
[48] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, and Jean-Pierre Hubaux. 2013. The Inconvenient Truth About Web Certificates. In *Proc. of the Workshop on Economics of Information Security and Privacy (WEIS)*. Springer, 79–117. https://doi.org/10.1007/978-1-4614-1981-5_5
[49] W3C. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/.
[50] Christina Warren. [n. d.]. Popular BitTorrent Client Transmission Gets Infected With Malware Again. https://gizmodo.com/mac-bittorrent-client-transmission-gets-infected-with-m-1785957214.
[51] Catherine S. Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack. 2009. User Perceptions of Security, Convenience and Usability for Ebanking Authentication Tokens. *Computers & Security* 28, 1-2 (2009), 47–62.

A TRANSCRIPT OF THE USER SURVEY

(A1) What type of laptop or desktop do you usually use, if any?
- ○ I use a computer running macOS
- ○ I use a computer running Windows
- ○ I use a computer running Linux
- ○ I do not use a computer

(A2) Do you have an antivirus installed on your computer?
- ○ Yes
- ○ Not sure
- ○ No

(A3) Software developers and vendors/resellers regularly provide updates and bug fixes. Approximately how often do you update the following technologies?

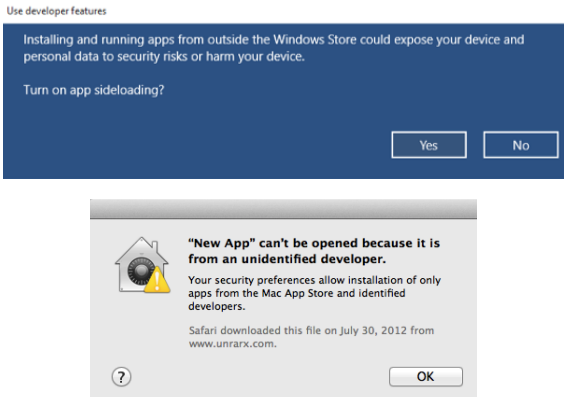
	Antivirus	Operating system (e.g., Window)	Applications (e.g., Word)
Never	○	○	○
Once a year	○	○	○
A few times a year	○	○	○
Once a month	○	○	○
Twice a month	○	○	○
Once a week	○	○	○
Multiple times a week	○	○	○
Every day	○	○	○
Happens automatically	○	○	○
Not sure	○	○	○

(A4) Approximately, how often do you perform the following actions on your computer? Download programs from:

	Peer-to-peer networks	Developer's websites	App stores	Vendor's websites
Never	○	○	○	○
Once a year	○	○	○	○
A few times a year	○	○	○	○
Once a month	○	○	○	○
Twice a month	○	○	○	○
Once a week	○	○	○	○
Multiple times a week	○	○	○	○
Every day	○	○	○	○
Not sure	○	○	○	○

(A5) What is the most typical thing you do after you have downloaded a program that did not come from the official app store on your computer?
- ○ Check it with my antivirus software
- ○ Open it
- ○ Check its legitimacy (please explain how)
- ○ Organize the file on my computer
- ○ Move it to an external storage
- ○ Ask a knowledgeable peer/family member for help
- ○ Not sure
- ○ Other (please explain)

(A6) Have you ever seen a warning message similar to that of the image below?[27]



- ○ Yes
- ○ Not sure
- ○ No

(A7) What would you do if you were presented with such a message when running a program that did not come from the official app store on your computer?
- ○ Stop opening the program and delete it from my system
- ○ Continue opening the program
- ○ Check the program with my antivirus software then continue
- ○ Double check whether I am on the right website and if so, continue running
- ○ Restart my computer then delete the program
- ○ Re-download the program
- ○ Ask a knowledgeable peer/family member for help
- ○ Not sure
- ○ Other (please explain)

(A8) How often did you experience the following situations in the last year? Your computer was infected by

	A virus	A spyware	An adware
Never	○	○	○
Once	○	○	○
A few times	○	○	○
Once a month	○	○	○
Twice a month	○	○	○
Once a week	○	○	○
Multiple times a week	○	○	○
Every day	○	○	○
Not sure	○	○	○

(A9) Have you ever noticed these sequences of numbers and characters, named "checksums" or "hashes" or "digests", on popular websites?[28] (see highlights on the examples below)

- ○ Yes
- ○ Not sure / I do not remember
- ○ No

(A10) What would you do with those sequences of numbers and characters, if you were to find them on the website of a program you would like to download?[29]
- ○ Use them to ensure the integrity of the program
- ○ Stop downloading the app and search for something else
- ○ Nothing, I will continue downloading the app
- ○ Ask a knowledgeable peer/family member for help
- ○ Google the app to find more information
- ○ Run an antivirus check
- ○ Not sure
- ○ Other (please explain)

(A11) Do you keep sensitive information on your computer?
- ☐ Personal
- ☐ Financial (e.g., credit card number, bank statements)
- ☐ Medical (e.g., exams results)
- ☐ Passwords
- ☐ Photos
- ☐ None of the above
- ☐ Other

[27] The warning message was customized depending on the answer provided by the respondent to question A1.

[28] The actual survey presented three distinct examples of popular websites to respondents. Respondents were asked whether they recalled seeing any similar "sequence of numbers and characters" in the past months from websites they visited.

[29] The same screenshots of question A9 were also displayed for question A10. Here respondents were presented a concrete scenario of downloading an app from a website with checksum and asked what they would do with the "sequence of letters and numbers".