On the Security of Application Installers & Online Software Repositories

Marcus Botacin[1], Giovanni Bertão[2], Paulo de Geus[2], André Grégio[1], Christopher Kruegel[3], and Giovanni Vigna[3]

Federal University of Paraná (UFPR), Brazil – {mfbotacin, gregio}@inf.ufpr.br
University of Campinas (UNICAMP), Brazil – {bertao, paulo}@lasca.ic.unicamp.br
University of California Santa Barbara (UCSB), USA – {chris,vigna}@ucsb.edu

Abstract. The security of application installers is often overlooked, but the security risks associated to these pieces of code are not negligible. Online public repositories have been one of the most popular ways for end users to obtain software, but there is a lack of systematic security evaluation of popular public repositories. In this paper, we bridge this gap by analyzing five popular software repositories. We focus on their software updating dynamics, as well as the presence of traces of vulnerable and/or trojanized applications among the top-100 most downloaded Windows programs on each of the evaluated repositories. We analyzed 2,935 unique programs collected in a period of 144 consecutive days. Our results show that: (i) the repositories frequently exhibit rank changes due to applications fast climbing toward the first positions; (ii) the repositories often update their payloads, which may cause the distribution of distinct binaries for the same intended application (binaries for the same applications may also be different in each repository); (iii) the installers are composed by multiple components and often download payloads from the Internet to complete their installation steps, posing new risks for users (we demonstrate that some installers are vulnerable to content tampering through man-in-the-middle attacks); (iv) the ever-changing nature of repositories and installers makes them prone to abuse, as we observed that 30% of all applications were reported malicious by at least one AV.

Keywords: Installer · Downloader · Trojan

1 Introduction

Modern operating systems (OS) have been providing more resources to meet users requirements over time. However, the unique needs of an heterogeneous user population can only be fulfilled by third-party software. Whereas Linux-based systems model for obtaining new applications often depends from official distribution repositories [24], MS-Windows based systems do not present any centralized software repository, outsourcing to the users the responsibility for downloading additional programs.

In this scenario, online software repositories have become the *de-facto* standard repository for most users. On the one hand, these repositories are a very practical service, as they group multiple applications in a single place with ranking and searching features. On the other hand, these repositories hardly check binaries' security, neither regarding vulnerabilities nor maliciousness, and their providers often do not take full responsibility for the distributed software. Therefore, the users themselves are responsible for the implications of installing software downloaded from these repositories.

Actually, most users blindly trust the repositories, which makes them vulnerable to exploitable code constructions (e.g., buffer overflows and/or man-in-the-middle attacks) and/or Trojanization attacks, i.e., when malicious code is added to legitimate applications. Trojanization is a common practice among attackers to deceive users into installing their malicious payloads inconspicuously and, when deployed on popular repositories, it might have a large-scale impact if we consider the potential target population of trojanized downloads. Repository Trojanization examples include the cases of the Arch Linux repository [9], the Asus update system [38], and the Android platform [20].

This scenario becomes even worse if we consider that most software repositories are known for appending other components to their distributed applications (e.g., adware), in a process named "bundling" [17]. Software bundling might end up adding vulnerable components to previously safe applications. It might also add tracking capabilities to initially privacy-respecting applications. It also opens to attackers the opportunity of embedding malicious payloads in programs distributed through repositories. Recent cases include `Sourceforge` [34]—accused of distributing malware via bundled binaries [18]—and malicious samples distribution via application installers [28]. Despite all occurrences of trojanized software in popular online software repositories, the academic literature dedicated to investigate this phenomenon is limited, and the few existing work mostly target the Android OS [1, 4, 37], rather than MS-Windows, whose few existing work are still limited in coverage [13]. Therefore, to bridge this gap, we propose to investigate the five more popular online software repositories (according to Alexa [2]), aiming at shedding light on the occurrence of vulnerable constructions and Trojanized applications that actually may infect end users. To do so, we obtained the 100 most-downloaded Windows programs on each of the five chosen repositories for a period of consecutive 144 days (from Feb/2019 until May/2019). We submitted the resulting 2,935 distinct binaries to static and dynamic analysis systems. We also developed a tool to automatically install those programs during their run in the sandbox, which allowed us to observe interactions between the monitored program and the OS.

Our results show that (i) the repositories are very dynamic, presenting frequent rank changes, thus allowing applications to fast climb to the first rank positions; (ii) the repositories often update their payloads, with distinct binaries being distributed over time even for the same applications. We also observed differences in the installers for the same applications distributed by distinct repositories; (iii) the installers are very dynamic, presenting modular constructions and often downloading payloads from the Internet to complement their installation steps. Whereas enabling flexibility, relying on the Internet also poses new risks if security measures are not taken. In this sense, we demonstrate that some installers are vulnerable to content tampering via man-in-the-middle attacks; and finally (iv) all this dynamic characteristic of installers and repositories open space for abuse, with 30% of all applications being reported as malicious by at least one AV.

In summary, our contributions are as follows: (i) We characterize the way in which online software repositories update their application's rankings and binary sharing among distinct installers regarding their interaction with OS components to understand their implementation decisions, scope, and impact on users' devices; (ii) We present statistics about multiple aspects of the installers distributed by popular repositories aiming to support further research work and investigations; (iii) We investigate the interaction

between application installers and the OS and evaluate installer's implementation choices; and (iv) We pinpoint behaviors found in installers that are compatible with malicious actions deployed by malware samples, and discuss best practices that could be adopted for the next-generation of non-intrusive application installers.

This paper is organized as follows: In Section 2, we present the main characteristics of online software repositories; In Section 3, we present the methodology adopted to conduct the performed experiments; In Section 4, we present evaluation results regarding the files distributed in online software repositories; In Section 5, we discuss our findings, their implications, and open research questions; In Section 6, we present related work to better position our developments; we draw our conclusion in Section 7.

2 Online Software Repositories

Online software repositories are popular websites: Softpedia ranks first in the Alexa's Shareware website list [2], with million accesses and downloads everyday. Google Chrome ranks third in this repository and accounts for 6M downloads. Microsoft Skype, the 28^{th}, was downloaded 3M times. Other repositories present same magnitude data: Ubit ranks first in the CNET repository and was downloaded 24M times. Therefore, every action in these repositories has potential to affect million users. In this scenario, every small percentage matters in the long-tail.

Table 1: **Repository Summary.** Repositories are diverse in multiple aspects.

Repository	Uploaded By	Reviewed By	Sponsored Ranking	Servers	Security Checks
FileHorse	Users	Site	✓	Internal/External	✓
Cnet	Users	Site	✓	External*	✓
FileHippo	Site	Site	✓	Internal	✓
SourceForge	Users	✗	✗	Internal	✓
Softpedia	Users	Site	✗	Internal/External	✓

Table 1 summarizes the diverse operation of the software repositories. It shows who starts the procedure to include a software in the repository (e.g., according to user's requests or to the website managers), who reviews the inclusion request (e.g., website managers), if the rankings are sponsored or not (e.g., if applications can climb ranking positions if they pay for it), on which servers the payloads are stored (e.g., internal repository's servers or developer's servers), and if the repository checks the distributed binaries (e.g., by performing some type of AV scanning). For most repositories, the process for adding a new software is started by the user filling some form. This will be further reviewed by the website managers. All repositories advertise they assure the software quality, but no guideline is specified for any repository. FileHippo does not accept user requests and its managers decide by themselves which application will be included. In Sourceforge's case, a project can be directly imported from Github. Once a software is included, its download page mentions the software creator, but they do not report who requested the software to be included. Most repositories allow the software to become popular by themselves, according to the number of downloads. CNET is a noticeable exception, allowing developers to sponsor their applications and climb ranking positions. Therefore, the application ranked first in the CNET repository is not necessarily the most popular application among all.

Most payloads are stored on internal repository servers and some repositories also allow users to directly get files from external sources (as an alternative link option). In

most cases, the links point to the software creator's page. In CNET's case, they point to a CDN. Requests are performed along with tokens which allow identifying the request origin. In our tests, on the one hand, direct links always resulted in the download of the same updated binaries available in the software creator's page. On the other hand, internal links always served distinct files than the official release (mostly outdated versions). All repositories claim the provided files are security checked. Some of them are backed by popular solutions, such as Avast (FileHippo) and Bitdefender (SourceForge). Despite that, it is not clear to what extent analyses are performed.

3 Methodology

In this section, we describe our methodology for our experiments in collecting and analyzing programs distributed via online software repositories.

Repository Selection and Programs Collection We selected the five most popular online software repositories according to `Alexa` score [2]: `Softpedia` [32], `Source Forge` [34], `CNet` [10], `File Hippo` [11], and `File Horse` [12]. Our intention was to ensure a broad range of samples and, at the same time, to be able to process all collected data on a daily basis. We developed an automated crawler (using Python's `Scrapy` [29]) to collect programs distributed by the aforementioned repositories. Our crawler operates as follows: (i) it first traverses all application ranking pages enumerating the available software and pages; (ii) it selects the top 100 most downloaded apps in the ranking; (iii) it accesses each selected application page and retrieves the download links; (iv) it downloads the file to our storage. This process was repeated daily for the five selected repositories, for a consecutive period of 144 days (from Feb/2019 until May/2019). Metadata from downloaded files were stored on a `sqlite` database, allowing further queries, such as: (i) what binary hashes were associated to which repositories; (ii) the binary's ranking position on a given day; (iii) the amount of distinct hashes collected under the same program's name in a given repository, among other information presented in Section 4.

Automated Application Installation and Analysis Although some installers enable unattended software installs, most of them requires users to interact with GUIs to proceed with installing steps (Figure 1). Therefore, to scale analysis of thousand samples, we developed a "clicker", i.e., an installing automation script that simulates user interaction with application installers. More specifically, we developed an `Autoit` [5] script to click the `Next` and `Finish` buttons displayed within graphical windows, allowing installers to proceed without human interaction.

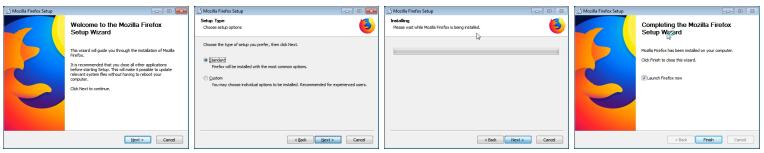

Fig. 1: **Automated Installation Example.** AutoIT scripts click on the next button until the installation is complete.

We leverage static and dynamic analyses procedures [31] to identify whether an installer was Trojanized with malicious payloads and/or was implemented following bad

development practices. To do so, we propose to match behaviors identified in installers to those knowingly exhibited by malware and suspicious software [16]. Our hypothesis is that benign software will exhibit none or few suspicious behaviors. We conducted static analysis procedures based on basic binary inspection—format and library identification, and samples submission to `VirusTotal` [35], to verify if those binaries would be detected by some AV installed on users' devices. The dynamic analysis consisted of running the samples in a virtualized sandbox machine with a malware monitoring system [7] to observe processes creation, filesystem operations, registry key changes, and network traffic. All valid Windows binaries were uploaded to that sandbox, in which each one was installed using our clicker.

Assumptions The experimental setup described in this section is supported by the following assumptions: (i) Our goal is not to provide an exhaustive analysis of all existing application installers, but a view on the most downloaded (and supposedly most installed) applications; (ii) Since not all websites will be reachable and not all binaries will be available every day, our goal is to provide a long-term view of the evaluated repositories dynamics, instead of a snapshot of a certain day; and (iii) We understand that some installers' operation might be unsuccessful due to the sandbox execution and the clicker stimulation. Thus, our goal is to provide an overview of common practices implemented by the applications installers, avoiding focusing on particular cases.

4 Repositories Evaluation Results

In this section, we present the results obtained from the evaluation of the programs distributed by the five selected online software repositories. Our experiment consisted of the following steps: (i) description of the collected dataset; (ii) evaluation of the content distribution dynamics within the repositories; (iii) drawing a landscape associating installers interaction with operating system internals; (iv) comparing the behavior exhibited by installers of the same software, but distributed by different repositories; (v) investigation for evidences of software trojanization.

4.1 Dataset Description

During the 144 days of collection, we successfully downloaded 46,018 files from the five online software repositories and built a dataset with 2,935 unique files, related to 1,633 distinct programs (Table 2). From those programs, 13 were software intended to remove other applications (uninstallers) and, due to that, they were evaluated separately from the remainder of the dataset samples (considered as "installers").

Table 2: **Dataset overview.** The number of unique files differs due to changes in distribution over time.

Repository	Programs (#)	Unique Files (#)
FileHorse	82	314
Cnet	118	295
FileHippo	433	906
SourceForge	99	631
Softpedia	901	897
Total	1,633	2,935

Table 3: **File sharing among repositories.** They usually do not share files for the same programs.

Repositories	Sharing Rate (%)
(Cnet, FileHorse)	48.04
(FileHippo, FileHorse)	17.65
(Cnet, FileHippo)	15.69
(FileHippo, Source Forge)	07.84
(Cnet, Softpedia)	04.90
(Cnet, Source Forge)	03.92
(FileHorse, Softpedia)	00.98
(FileHippo, Softpedia)	00.98

The number of unique files is greater than that of unique applications because the distributed files vary over time (among distinct repositories as well as within the same repository), and the total number of downloaded files does not correspond to the expected sum of each repository downloads. The reason is that 105 (3.6%) files were shared by two (95% of all shared files) or three (5% of all shared files) repositories. In Table 3, we show that most repositories do not share files among themselves even for the same programs, implying that they distribute distinct program versions or installers.

Programs distributed by the repositories are packaged in multiple formats (Table 4). Although Trojanization can be implemented via any packaging type, we focused on binaries with Windows PE file format [25], since they are the prevalent file format in our dataset, and are also self-contained installers, which makes Trojanization easier for attackers. Most PE files present in our dataset are 32-bits, still reflecting the long-term trend of developers that delay the adoption of new programming techniques to native support 64-bit applications, as reported in [36]. Interestingly, some installers are packed with UPX (2.6%) and/or Armadillo (0.6%) so as to compress their payloads. Only 19.3% of the PE installers were crypto-signed.

Table 4: **File types distribution.** Self-contained PE files are the prevalent type of program installers.

Type	Format Prevalence (%)	
Java		0.67
ISO		1.04
Compressed	7-zip 0.37	RAR 0.30
File	XZ 0.37	ZIP 20.47
Formats	bzip2 0.37	gzip 1.34
Windows	DOS 0.45	PE 65.63
Binaries	.Net 0.67	PE+ 0.45
Other		7.87

Table 5: **Binary file's size distribution.** Small binaries are associated to downloaders and large ones to droppers.

Interval (MB)	Frequency	Binaries(%)
[0.000, 0.400)	93	5.42
[0.400, 1.400)	128	7.46
[1.400, 5.000)	242	14.11
[5.000, 70.000)	619	36.08
[70.000, 150.400)	145	8.45
[150.400, 600.400)	105	6.12
[600.400, 888.000)	16	0.93

The variety of formats distributed by the software repositories affects the installers' file sizes, shown in Table 5. The differences in files sizes is important due to storage issues and because they may reveal implementation strategies behind the installer: smaller binaries usually only implement a client that downloads the actual payload from the Internet (Type I installer); larger binaries embed the payload themselves, dropping them at installation time (Type II installer). Although the first approach enables content creators to keep distributing up-to-date versions of their software, it makes security checking harder, as the distributed content changes very often. In terms of Trojanization, an attacker who controls a Type-I installer might implement a downloader [27], whereas an attacker who controls a Type-II installer might implement a dropper [16].

4.2 Repositories Dynamics

The chances that a malicious actor trojanizes a given repository and the impact that it can cause are strongly tied to the repository's operation dynamics, since more frequent repository updates make it harder to track newly added code. In addition, if it is easier for newly added software to climb the top ranking positions, their infection might become even more impacting. To delve into those dynamics, we evaluated the samples crawled daily from the repositories.

In Figure 2, we show the number of downloads from each repository along the experiment's period. Overall, all datasets grew almost linearly due to our daily queries to the top-100 ranking positions. Variations were caused due to unreachable servers on a given day, or broken links/Web pages.

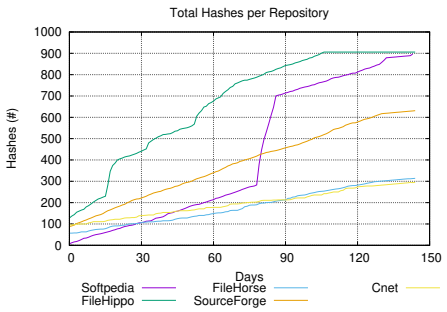

Fig. 2: **Accumulative downloads** for each software repository.

Fig. 3: **Daily Downloads.** FileHippo's servers were unreachable in the last week.

In Figure 3, it is possible to observe that the download of more than 80 unique files (from the top-100) was only accomplished within FileHippo and CNET. The daily number of collected programs was mostly constant, if we consider each repository, with few days presenting peaks or valleys in the crawling process. The observed variations were related to Website updates or unavailability.

Each repository distinguishes itself regarding the samples successfully downloaded, as in the addition of new samples. Figure 4 shows the number of new unique samples (based on the binaries' MD5 hash) added to the repositories daily. We notice that `FileHippo` has many more new additions each day than the other repositories (except for particular peaks in Softpedia, Sourceforge, and CNET). This is caused by the frequent update of the distributed payloads, which indicates that FileHippo is more volatile about the content of its distributed installers (therefore may be riskier for users).

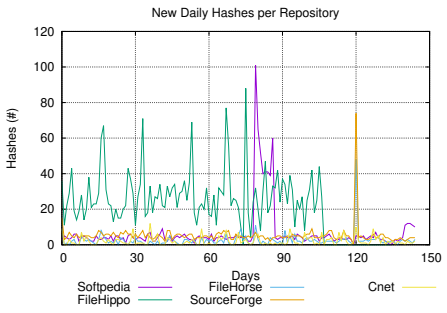

Fig. 4: **Download of new (unique) files.** FileHippo's repository exhibits periodical peaks of newly added hashes.

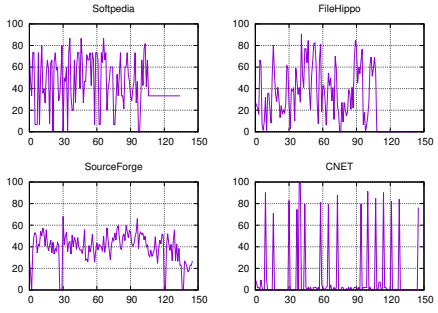

Fig. 5: **Ranking position changes** of the top-100 downloaded programs in each repository, but FileHorse. Observation days vs. applications (#) whose rank changed.

The observed strategy of payload replacement led us to hypothesize that the top-100 programs may also change their ranking positions frequently. To investigate this hypothesis, we measured the fraction of programs whose ranks changed each day. Figure 5 shows the change ratio per repository (we did not show FileHorse's results due to its incipient rank changes of less than 1% in most days), which confirms that almost all programs changed their position on some days. Similar to the aforementioned new hashes' case, we noticed that each repository has distinct ranking dynamics.

The ever-changing operation of software repositories is highlighted when we limit our analyses to the most downloaded programs. Initially, we believed that their ranking positions would hardly change, given their popularity. In practice, we observed that ranking changes affect even the most downloaded programs, occurring more frequently among the top-5 in all repositories. Understanding the phenomenon of frequent rank changes is important because it shows how quick a new (potentially malicious) software can reach the top of the ranking after its release. It also allows us to evaluate the extent of potential damages according to the number of affected users based on the popularity of programs. To explore this possibility, we measured how many programs change their ranking position at least once within a given repository, and how many positions on average they scale up the rank. Figure 6 shows that most programs change their position at least once (on average, only 12% finished the observed period in the same ranking position). We observed in all repositories' rankings that most programs scaled up few positions. We also observed that more programs increased their ranking instead of having it decreased. It happens due to the repository removing some programs from the top lists to add newer software, thus creating a gap in the former individual ranks while naturally allowing the latter to scale up some positions.

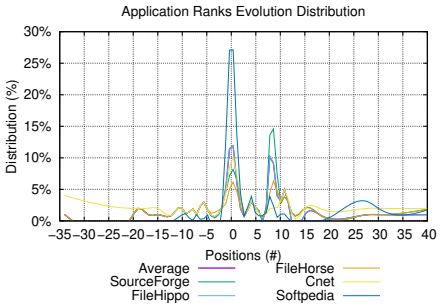

Fig. 6: **Distribution of Programs in Ranking Positions.** Most programs increase their ranking position (at least once).

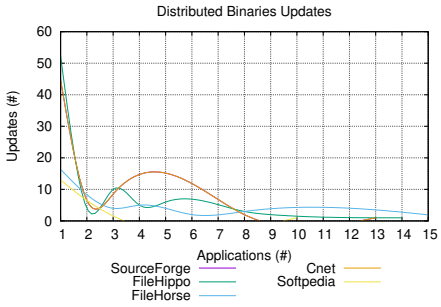

Fig. 7: **Distributed Binaries Updates.** Most programs were updated few times, whereas some others, every week.

Although most programs does not reach the top of rankings, some of them scaled from the last pages to the first positions. We also observed that this growth occurred in a short period of time (only 4 days for Google Chrome and a month for other programs). The popularity of these programs raise concerns about the potential harm that might be caused if one of them is Trojanized. Highly popular programs, such as Google Chrome, were not expected to be low in the rankings any time. However, in times of Google Chrome version releases, (72.0.3626 in the period [14]), the ranks have to be updated with a new entry for this program. The possibility of changes in the binaries distributed for

the same application over time also raises security concerns, since Trojanized versions of them could serve as a replacement to the legitimate ones. To evaluate this hypothesis, we measured how many repositories implement this practice and how frequent it is deployed by them. The rate of the software in each repository which had their binaries changed at least once in the observed period indicates that there is Trojanization opportunities for malicious actors: FileHorse (42.74%); FileHippo (30.36%), Sourceforge (29.58%); CNET (11.41%); Softpedia (9.43%). We consider these rates significant as they show that the repositories evolve not only by adding new software entries over time but also by modifying existing ones. The update of the distributed binaries is not homogeneous for all programs. Figure 7 shows the frequency in which each one of the applications have their distributed binaries updated during the observation period. We notice that while most programs are updated only few times—probably due to software updates—the remainder programs are updated very often. Some programs were updated more than 50 times (considering distinct repositories), an update rate greater than one time per week during the observation period. This constant updating routine opens a significant attack opportunity window, since at the time of the security analysis of previously distributed binaries is complete, the repository is already distributing a novel, not-yet-analyzed software version.

Among the programs whose binaries were updated more frequently, we highlight once again the importance of paying attention to the popular applications. For instance, Skype changed six times in FileHippo and seven times in FileHorse from February 13, 2019 to May 15, 2019. Those changes referred to updates either in the software version or in its distributed installer (discussed next in Section 4.3).

4.3 Installers' Dynamics

Repositories usually provide program installers, which perform numerous interactions with the underlying OS. For instance, they are responsible for copying contents to the correct directories, setting environment paths, adjusting Registry keys, loading drivers, installing additional services, and so on. The implementation strategies to accomplish those tasks is varied: installers may download payloads and related configuration files from remote servers, or directly extract them from embedded resources; their system configuration changes may affect a single user or the whole system; they may rely on system libraries or install their own ones; they may require privilege escalation or not. All of these actions affect system security, thus we present an overview of which of them were found in the evaluated installers, so as to draw a landscape of installers operations and the associated security risks. From the 1,633 collected programs, we limited our evaluation to the 993 unique binary samples packed as Windows executables (PE file format) that were successfully installed in our sandbox (the unsuccessful ones failed mostly due to corrupted files and/or missing environment variables).

Installers Modularity. We observed that installers present highly modular constructions. 52.62% of them created at least one child process during the installation process (98% of these created only a single process, but we identified one installer that created up to 15 child processes during its operation). Installers rely on child processes for multiple tasks: (i) 13.4% of the installers create new processes to relaunch the program installer with properly defined parameters, with the main installer executable being responsible only for displaying the Graphical User Interface (GUI), which allows users to specify what components will be installed; (ii) 1% of the installers create new processes to

launch external tools to extract compressed objects (e.g., unzip); (iii) another 1% of installers rely on child processes to launch downloaders; (iv) 1% use children to launch post-installation procedures, such as opening a browser to display installation messages; and (v) 1% make child processes execute `cmd` or `powershell` scripts for them. The remaining modules invoked by installers were system processes intended to perform generic tasks. A major motivation for installers launching child process is to execute payloads extracted from the main installation binary. This "dropping" strategy was identified in 25.3% of samples. Code 1.1 shows two installers writing their payloads in executable files on disk. Their goal is to distribute multiple components as a single file.

```
1 C:\installer.exe|Write|C:\Users\Win7\AppData\Local\Temp
      \{907A1104-E812-4b5c-959B-E4DAB37A96AB}\vsdrinst64.exe
2 C:\installer.exe|Write|C:\Users\Win7\AppData\Local\Temp
      \{907A1104-E812-4b5c-959B-E4DAB37A96AB}\Install.exe
```

Code 1.1: **Dropper Installer.** Some Installers drop embedded payloads to disk and launch them as new processes.

Installers might also retrieve payloads from the Internet—10.8% of the evaluated ones exhibited this behavior. On the one hand, downloading payloads from Internet allows installers to retrieve them according to the installation environment (e.g., distinct OS versions), and to install updated versions of all software components. On the other hand, it requires a machine connected to the Internet at the moment of the intended program's install, which makes the installer less self-contained. Code 1.2 illustrates an installer requesting to download a payload from the Internet. This request was encoded to not reveal much information about its content.

```
1 GET 200.143.247.9:80 (et1.zonealarm.com/V1?
2 TW9kdWxlPWluc3RhbGxlch98U2Vzc2lvbj0wYzNjNDA1OD)
```

Code 1.2: **Downloader Installer.** Some Installers perform (encoded) network requests to retrieve payloads from Internet.

The exhibited behaviors of modularity (many child processes), downloader, and dropper are also reflected in the installers' written files (Table 6). The prevalent file types are libraries, which allow code reuse. Executables are the second most popular ones, since they represent the programs being installed. Temporary files are the third most popular extensions, mostly due to the objects dropped during installation procedures: installers usually drop small pieces of data to files to reconstruct global, complex structures, and the temporary files are used to store binary blobs, raw text, and proprietary structures. We also identified that `VPX` files—closed source files used by `Avast` and `AVG` antiviruses to store malware definitions—are very popular within installers, being used to deploy signature updates. Finally, we observed that some installers write `SYS` files, which allow them to load kernel drivers and affect the system as a whole.

Table 6: Top-5 file extensions most written by installers.

Extension	DLL	EXE	TMP	VPX	SYS
Files (#)	6,949	1,309	1,302	811	790

Network Usage. Payload downloading enables updated software versions install (e.g., AVs with up-to-date signatures). However, download mechanisms proper deployment may be challenging, resulting in security issues. For instance, flawed cryptography (or the lack of support for encrypted connections) may expose users to payload tampering via Man-In-The-Middle (MITM) attacks [26]. We identified 39 applications that download binaries via HTTP-only connections, as shown in Code 1.3. The list of installers that retrieve payloads via HTTP includes popular programs, such as `Avast`, `BitDefender`, `AVG`, and `Kaspersky` AVs. The AV's choice for HTTP-only downloads has already been reported in the past [22], but it seems to keep its standard practice status over time.

```
1 GET iavs9x.u.avast.com/iavs9x/
    avast_free_antivirus_setup_online_x64.exe
2 GET download.bitdefender.com/windows/bp/all/avfree_64b.exe
3 GET iavs9x.avg.u.avcdn.net/avg/iavs9x/
    avg_antivirus_free_setup_x64.exe
4 GET dm.kaspersky-labs.com/en/KAV/19.0.0.1088/startup.exe
5 GET download.bullguard.com/BullGuard190AV_x64_190411.exe
```

Code 1.3: **Unencrypted Download by Installers.** The use of HTTP-only connections may make users vulnerable.

To test whether the installers were actually vulnerable to payload tampering, we performed a MITM against them. Despite the unencrypted payload downloads, all popular installers, including AVs, were not vulnerable to payload tampering, since they are able to realize payload changes through certificates and checksum verification. Other programs, such as the `BullGuard` backup solution, are vulnerable to this type of attack[1]: its installer downloaded our supplied payload and executed it without any checks. This opens a significant infection vector for the execution of any attacker-supplied code if the installer is executed in a hostile network.

Installation Tracking. Installers also rely on Internet support to track programs' installs. 4% of all installers sent clear tracking data back to their servers during the installation step (Code 1.4). Additional tracking data might be sent after the program runs for the first time (e.g., software that require users registration).

```
1 GET /v1/offer/campaignFilter/?bundleId=UT006&campaignId=5
    b6352b3ce72513ae0a6beef
2 GET sos.adaware.com|/v1/offer/campaignFilter/?bundleId=
    UT006&campaignId=5b6352b3ce72513ae0a6beef
3 GET flow.lavasoft.com|/v1/event-stat?ProductID=IS&Type=
    StubBundleStart
```

Code 1.4: **Installation Tracking.** Some installers sent back tracking information to notify providers about the installation.

Application installers collect tracking data for many reasons, such as identifying software popularity by keeping track of the number of installations, and displaying targeted ads campaigns. Unfortunately, most installers do not make this user data collection

[1] We contacted the vendor and disclosed all vulnerability's details so the company could fix it.

explicit. For instance, the privacy terms for Code 1.4's program installer state that: "*We collect some limited information that your device and browser routinely make available whenever you visit a website or interact with any online service.*", "*We collect this data to improve the overall quality of the online experience, including product monitoring, product improvement, and targeted advertising.*", and that "*We may also include offers from third parties as part of the installation process for our Software.*". Besides the claims that the program collects a wide range of data, it is not clear what kind of data is collected during website visits, software execution, and software installation. Moreover, the installation step deserved a single line in the whole privacy term, showing that the impact of software installation is often understated.

Installer's Proxies. To access the Internet, some installers end up performing intrusive system changes. We identified that 5% of all installers changed proxy settings of the whole system. Code 1.5 shows an installer that enabled a proxy by writing to a system's Registry key. While some installers define new proxies, others only remove previously defined proxy settings. Although it may happen with the solely purpose to ensure that the payloads are downloaded from a proper source, it affects all further network requests.

```
214|2019-2-12 C:\Users\Win7\AppData\Local\Temp\BullGuard
    Backup Setup.exe|SetValueKey|HKU\<userid>\Software\
    Microsoft\Windows\CurrentVersion\Internet Settings|
    ProxyEnable|1
```

Code 1.5: **Proxy Definition.** Some installers change system-wide proxy settings.

Installers Persistence. Installers may change Registry keys to allow binaries to be invoked upon a system reboot. We identified that 1% of them exhibit this behavior. One reason for installers implement persistence is to set the installed program as a background daemon. This task is often performed by security applications' installers, such as AVs (Code 1.6). Another reason for the persistence behavior is because it allows splitting the installation process in multiple steps. This is required when the installation of some components requires rebooting (e.g., to load kernel drivers). Whereas daemons are often set by writing to the `AutoRun` Registry keys, multi-step installers often implement their own counters, as exemplified in Code 1.7.

```
C:\Users\Win7\AppData\Local\Temp\7zS4DEAD364\Stub.exe|
    SetValueKey|HKU\<userid>\Software\Microsoft\Windows\
    CurrentVersion\RunOnce|PandaRunOnce|
```

Code 1.6: **Persistence.** Some installers set executable paths in the Registry to be executed after a system reboot.

```
C:\Users\Win7\AppData\Local\Temp\ajAE1E.exe|SetValueKey|
    HKLM\SOFTWARE\Wow6432Node\AVAST Software\Browser|
    installer_run_count|1
```

Code 1.7: **Multi-Step Installers.** They control how many times they will run.

Affected System Scope. Installers may modify several other Registry keys. In many cases, these modified keys affect the whole system instead of the single user running the

installer process. We identified that 56% of all installers affected only the single user who is installing the program (HKCU keys), whereas the remaining 44% also affected machine-wide Registry keys (HKLM).

Application Removal. Most installers do not implement proper cleanup routines after finishing the installation process. Only 33% of all installers dependent on temporary files deleted them before ending their process.

Allowing software to be properly removed is as important as to properly install the application. Unfortunately, not all installers provide adequate mechanisms to remove their installed objects: only 1% of them created an uninstaller object able to be invoked in a standalone fashion, as shown in Code 1.8.

```
1 C:\Users\Win7\AppData\Local\Temp\{907A1104-E812-4b5c-959B-
      E4DAB37A96AB}\Install.exe|Create|C:\Users\Win7\AppData\
      Local\Temp\{907A1104-E812-4b5c-959B-E4DAB37A96AB}\
      Uninst.exe
```

Code 1.8: **Uninstaller Definition.** Some Installers set uninstallers for the applications.

Identifying whether installers defined an uninstalling routine or not has proven to be a hard task: 1% of the tested programs define uninstalling routines based on specific parameters, as illustrated in Code 1.9.

```
1 C:\Program Files (x86)\GUM5D5C.tmp\fmanUpdate.exe|
      SetValueKey||HKU\<userid>\Software\fman\Update|
      UninstallCmdLine|"C:\Users\Win7\AppData\Local\fman\
      Update\fmanUpdate.exe" /uninstall
```

Code 1.9: **Parameter-Based Uninstallers.** They define command line parameters for software removal (difficult for users), instead of providing a self-contained uninstaller.

4.4 Comparison of Installers Versions

We identified that distinct binaries have been distributed for the same application over time and across repositories. Understanding the modifications that these binaries underwent might provide important insights to improve installers development and security.

Differences in installers within the same repository. We first evaluated how the binaries available for the same program and distributed by the same repositories change over time. We initially hypothesized that these binaries could be subject to significant modifications. However, we discovered that the modifications overall are more structural than behavioral, thus suggesting that the differences occur more due to installers evolution than due to other code insertion mechanisms.

In the cases when the installers were effectively modified to embed additional applications, their most prevalent payloads referred to toolbars and browsers add-ons. 1% of all binaries were versions of previous installers modified to include the `Google Toolbar`, which is often embedded as part of third party extensions within the main application (Code 1.10).

```
1 C:\installer\3rdPartyApp\GoogleToolBar\
      GoogleToolbarInstaller_zh-TW.exe
```

Code 1.10: **Google Toolbar** It is embedded as 3rd-party extensions of the main app.

In cases where the installers do not directly perform a toolbar installation, they managed to change the native Internet Explorer configurations to display customized settings, which includes adding new bookmarks and cookies (Code 1.11).

```
1 HKCU\Software\Microsoft\Internet Explorer\LinksBar\
      ItemCache\ToolBar|Add
```

Code 1.11: **IE Settings Modification.** New bookmarks, cookies, and configurations set in the browser.

Another 1% of all binaries were embedded with advertisement applications instead of toolbars. These applications, known as `adware` (advertisement software), often run in background and keep collecting users information to feed targeted ads campaigns. Code 1.12 shows an adware running from a temporary file dropped by the main installer.

```
1 C:\Users\Win7\AppData\Local\Temp\is-3ACQL.tmp\
      Advertising_english.exe
```

Code 1.12: **Adware.** The advertisement software is dropped from a file created by the main installer.

Differences in installers among the repositories. The tracking capabilities present in the installers are clearly revealed when we compare installers for the same applications downloaded from distinct repositories. While we were unable to identify any significant difference in the behaviors exhibited by the binaries, we easily noticed their tracking capabilities. Code 1.13 illustrates an excerpt of the installation trace for the same program, but using binaries downloaded from three distinct repositories. We notice that the `UserId` values considered in each installation is different for each binary. We executed many installation attempts and discovered that this number is not randomly generated, but seems to be tied to each binary. We considered this an indication that the installers are able to identify the origin of their installation.

```
1 C:\Setup.exe|SetValueKey|HKCU\Software\Microsoft\Client|
      UserId|{C2CFE0D4-A3A2-4458-A73F-F16F10E4C0D7}
2 C:\Setup.exe|SetValueKey|HKCU\Software\Microsoft\Client|
      UserId|{EA0CB74D-DB5D-40EE-A402-47A97F23904E}
3 C:\Setup.exe|SetValueKey|HKCU\Software\Microsoft\Client|
      UserId|{E81A6607-9EB3-49BA-B354-FA42817594BA}
```

Code 1.13: **Tracking IDs of installers of distinct repositories.** Each installer presents a distinct tracking ID according the repository from which they were downloaded.

4.5 Trojanization Evidences

The major problem associated with downloading software from third-party repositories is that the downloaded binary may be a Trojanized version of the original software. This type of attack has been becoming popular to the point of some installers explicitly warning users about this possibility, as shown in Figure 8.

To verify if Trojanization cases occur in practice, we performed AV scans on all downloaded binaries. We submitted all binaries to VirusTotal [35] and normalized the

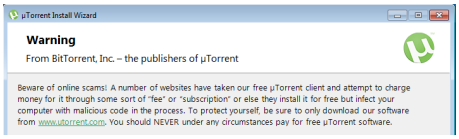

Fig. 8: **Security Warning.** Trojanization has become popular to the point of some installers warning users about this possibility.

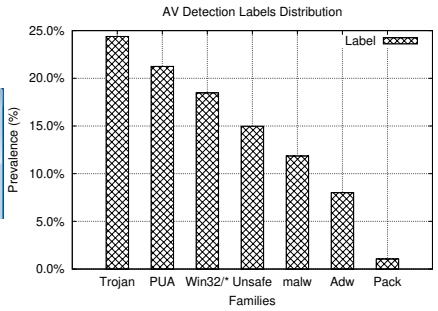

Fig. 9: **AV Labels Distribution.** Many samples were considered either as malicious or as trojanized.

retrieved labels using AVClass [30]. We discovered that 31% of all binaries were detected by at least one AV. We further investigated the nature of these detection occurrences by inspecting the assigned AV labels, whose distribution is shown in Figure 9.

The most prevalent detection label is "Trojan", which means that malicious code was inserted into application's native code. This finding shows that, as hypothesized, there is a real risk of application Trojanization in online software repositories. Among the Trojanized programs, we were able to identify 20 distinct families of the `Artemis` malware [33], thus showing that the attackers have been embedding real, harmful malware to the online repositories' distributed programs. Some AVs also detected the adware software embedded in part of the programs as malicious. This type of detection happens because the AV understands that the embodiment of advertising software to the original application implies on privacy leak risks to the user. A smaller part of the samples was detected as malicious due to their innate nature—12 installers were detected as downloaders and two as droppers, since the AVs were unable to distinguish their "legitimate" operation from the same behavior exhibited by malware classified as downloaders or droppers.

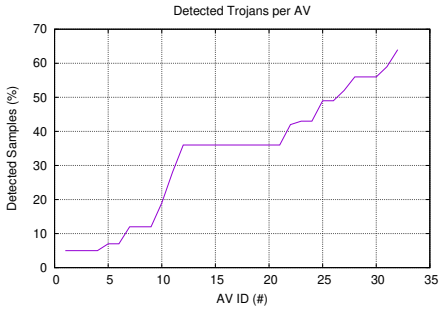

Fig. 10: **Trojanized Apps Detection per AV.** Distinct AVs present very distinct criteria and thus detection rates.

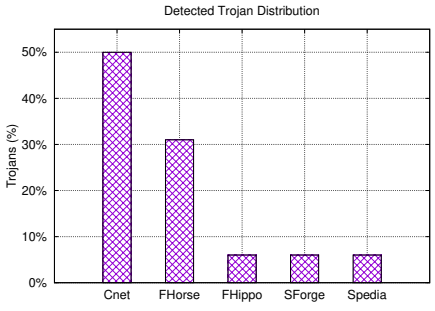

Fig. 11: **Trojanized Apps Detection per Repository.** Distinct repositories present very distinct rates.

The detection of Trojanized apps is not uniform among the AVs, as shown in Figure 10. Whereas some AVs detected only 3% of all samples reported as Trojanized by at least one AV, other AVs detected more than 60% of all reported samples. This shows that the AVs employ very distinct criteria for detecting Trojanization (e.g., adware inclusion is considered malicious for some but not for others). This highlights the need of checking multiple AVs in addition to the ones considered in the repository pages, as this AV might have a very lax detection criteria. The detection is also not uniform among the repositories, as shown in Figure 11. Whereas some repositories accounted for less than 10% of all detected malicious files in the period, CNET accounted for 50% of all samples. Despite that, we cannot claim that the CNET repository is more insecure than the others, as most detection occurrences are due to the repeated upload of the same flagged file. This shows that the evaluation of software repositories should also consider the frequency of upload of malicious files in addition to their occurrence.

5 Discussion

In this section, we revisit and discuss our findings to pinpoint existing gaps in the security of online software repositories and some possible and concrete improvement actions.

Paying Attention to Popular Applications. Although the software repositories may contain thousands of distinct applications, some of them gather more attention than others. Popular applications may be downloaded million times each month, thus presenting a huge potential of damage if they have been Trojanized. Our study showed that some programs are really popular, being present in the top download application rankings of multiple repositories simultaneously. In addition, in some cases, popular applications might quickly achieve the top ranking positions after a short period of time, which shows that the hypothesized popularity and usage broadness occurs in practice. In this scenario, it is essential for the repository administrators (and all security-related players) to pay attention to these programs to prevent trojanization cases, and counter them when they happen. In this sense, we consider that the recent decision of Google of extending its bug bounty program from its own applications to all other ones present in Google Play that have more than 100M installs [6] as a correct and very necessary move. Moreover, we consider that all other good security practices, such as fuzzy testing and audits, should be extended as well. Unfortunately, we are not aware of any kind of similar action regarding the samples provided by popular online Windows application repositories.

Reproducibility of Studies Leveraging Software Repositories. Many studies rely on software repositories as a source of binaries for their evaluation, either to measure bug prevalence in the software engineering context [15,40], or as a direct source of goodware for balancing malware analysis datasets and/or machine learning training. These studies may be strongly impacted by our findings, since we showed that software repositories are very dynamic. In this scenario, a study conducted with the top applications of one repository might result in completely different conclusions when applied to other repositories. The same effect may happen even within the same repository if the software is collected on different days, as ranks and binary versions change over time. Therefore, reproducibility should be a concern for all researchers whose works rely on software repositories. Researchers need to find ways to make samples and other information available and reproducible, as only stating that the most popular samples from a given

repositories were used in their study is not enough information to reproduce their experiments and obtained results in this ever-changing context.

Repositories as Source of Goodware. Binaries downloaded from software repositories are often used for malware classification and/or ground-truth [39]. Our findings also present strong implications to these cases. We showed that Trojanization might affect all repositories, thus even programs downloaded from "official" or popular repositories must be checked by antivirus solutions before being considered clean. Otherwise, the researcher could wrongly consider existing malicious behaviors embedded in the Trojanized application as ground-truth for benign applications. Even worse, one could mistakenly make a machine learning algorithm to learn a set of malicious behaviors as legitimate. Therefore, researchers should not blindly trust software repositories.

Other Repositories Issues. This work investigated the overall impact of using software repositories. Our results can be applied to both end-users downloading applications from these repositories as well as for researchers leveraging these applications as ground-truth for their experiments. However, software repositories present a myriad of applications that deserve special attention. Our goal in this work was not to exhaust the subject, but to give a first step towards a better understanding of characteristics of online repositories. We pinpoint that other repositories aspects might be addressed as future work. In particular, we understand that uninstallers might also be studied, in addition to the installers, since traces of previous applications can also significantly affect systems operations, either regarding continuous privacy leaks or performance degradation.

Limitations & Future Work. Software Repositories are very diverse and popular. Therefore, other repositories than the ones presented here should be studied to present a broader overview of security issues. This additional investigation might raise new hypothesis, such as if less popular repositories are more prone to be Trojanized than the ones here presented. The data collected in our experiments was not enough to cluster the tools used to trojanize the apps in classes. We expect that this task could be done via larger-scale experiments using multiple repositories.

6 Related Work

We here present related work to better position our contributions.

Trojanization is an effective and efficient approach to deliver malicious payloads, and its occurrence in practice presents large-scale implications. Code Trojanization has already been reported in practice in the repository of the `Arch Linux` distribution [9], in the `Asus` update platform [38], and even in the `Android` platform [20]. In the context of this research, we investigate occurrences similar to ones reported for `SourceForge`, accused of distributing malware among other applications [18]. We believe that Trojanization might become a prevalent problem in future years. Currently, Trojanization occurrence has been already reported even for hardware devices [8].

Software Repositories are very popular among many users as they allow gathering new software pieces in an easy way. Thus, they were studied by many researchers in the software engineering literature [15, 40]. These work, however, are more focused on source-code analysis rather than on the binaries distributed to end-users. This type of research was only made popular in recent years due the emergence of application stores for mobile devices, as observed in the rise of many studies targeting the `Android` platform [1, 4, 37]. These research work identified phenomena such as the same app

being distributed in different packages according the repository [3]. In this work, we extend this type of phenomenon observation to the scenario of online repositories for Windows binaries, whose few existing research work are still limited in coverage (e.g., evaluating less than thousand samples collected on a single day [13]).

Installers & Uninstallers are critical pieces of software for system operation as they perform extensive changes on the system's state. For instance, remaining registry entries after a software removal may cause systems to slowdown [19]. Unfortunately, there are currently a limited number of research work in the literature dedicated to investigate their impact, with most developments focusing on how to perform remote apps installation [41]. The closest work to ours are related to the investigation of the application installation logs on the `Android` platform [23] and the detection of piracy signs on application installers [21]. We extend these initiatives to investigate the occurrence of Trojanization on application installers.

7 Conclusions

In this paper, we investigated the occurrence of application Trojanization in the binaries distributed by popular Internet software repositories. We crawled the top-100 most downloaded Windows applications of five repositories for 144 days, which allowed us to characterize the dynamic of these repositories' operations. We also investigated the characteristics of the downloaded installers by running them in a sandbox solution instrumented with a clicker for automatic application installation, which allowed us to characterize installer's interactions with the operating systems. Our results show that: (i) the repositories are very dynamic, presenting frequent rank changes, thus allowing applications to fast climb to the first rank positions; (ii) the repositories often update their payloads, with distinct binaries being distributed for the same applications. There are also differences in the binaries for the same applications distributed by distinct repositories; (iii) the installers are very dynamic, presenting modular constructions and often downloading payloads from the Internet to complement their installation steps. Whereas enabling flexibility, this also poses new risks if security measures are not taken. We demonstrate that some installers are vulnerable to content tampering via man-in-the-middle attacks; and (iv) all this dynamic characteristic of installers and repositories open space for abuse, with 30% of all applications being reported as compromised by at least one AV solution. Our results shed light on some drawbacks of relying on software repositories, both by end-users installing these programs in their computers, as well for researchers leveraging these software repositories as ground-truth for their experiments. We also hope that our analysis could motivate other researchers to investigate other software repositories issues and help the community to understand their impact.

Reproducibility. all code developed to support this research work is available at `https://github.com/marcusbotacin/Application.Installers.Overview`

Acknowledgments. We thank the anonymous reviewers and the shepherd Pierre Laperdrix for their valuable comments. This project was funded by the Brazilian National Counsel of Technological and Scientific Development (CNPq, PhD Scholarship, process 164745/2017-3) and the Coordination for the Improvement of Higher Education Personnel (CAPES, Project FORTE, Forensics Sciences Program 24/2014, process 23038.007604/2014-69).

References

1. Al-Subaihin, A., Finkelstein, A., Harman, M., Jia, Y., Martin, W., Sarro, F., Zhang, Y.: App store mining and analysis. DeMobile, ACM (2015)
2. Alexa: Top sites. https://www.alexa.com/topsites (2019)
3. Ali, M., Joorabchi, M.E., Mesbah, A.: Same app, different app stores: A comparative study. MOBILESoft, IEEE (2017)
4. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Androzoo: Collecting millions of android apps for the research community. MSR, ACM (2016)
5. Auto-it: Auto-it. https://www.autoitscript.com (2019)
6. Bacchus, A., Porst, S., , Mutchler, P.: Expanding bug bounties on google play. https://security.googleblog.com/2019/08/expanding-bug-bounties-on-google-play.html (2019)
7. Botacin, M.F., de Geus, P.L., Grégio, A.R.A.: The other guys: automated analysis of marginalized malware. Journal of Computer Virology and Hacking Techniques (2017)
8. Bronchain, O., Dassy, L., Faust, S., Standaert, F.X.: Implementing trojan-resilient hardware from (mostly) untrusted components designed by colluding manufacturers. ASHES, ACM (2018)
9. Cimpanu, C.: Malware found in arch linux aur package repository. https://www.bleepingcomputer.com/news/security/malware-found-in-arch-linux-aur-package-repository/ (2018)
10. Cnet: Cnet. https://www.cnet.com/ (2019)
11. FileHippo: Filehippo. https://www.filehippo.com (2019)
12. FileHorse: Filehorse. https://www.filehorse.com (2019)
13. Geniola, A., Antikainen, M., Aura, T.: Automated analysis of freeware installers promoted by download portals. Computers & Security (2018)
14. Google: Release updates from the chrome team. https://chromereleases.googleblog.com/ (2019)
15. Gousios, G., Kalliamvakou, E., Spinellis, D.: Measuring developer contribution from software repository data. MSR, ACM (2008)
16. Grégio, A.R.A., Afonso, V.M., Filho, D.S.F., d. Geus, P.L., Jino, M.: Toward a taxonomy of malware behaviors. The Computer Journal (2015)
17. Han, J., Chung, T., Kim, S., Kwon, T.T., Kim, H.c., Choi, Y.: How prevalent is content bundling in bittorrent. SIGMETRICS, ACM (2011)
18. Hoffman, C.: Warning: Don't download software from sourceforge if you can help it [updated]. https://www.howtogeek.com/218764/warning-don%E2%80%99t-download-software-from-sourceforge-if-you-can-help-it/ (2018)
19. Kahvedžić, D., Kechadi, T.: On the persistence of deleted windows registry data structures. SAC, ACM (2009)
20. Khandelwal, S.: New malware replaced legit android apps with fake ones on 25m devices. https://thehackernews.com/2019/07/whatsapp-android-malware.html (2019)
21. Kim, D., Kim, Y., Moon, J., Cho, S.J., Woo, J., You, I.: Identifying windows installer package files for detection of pirated software. In: ICIMISUComp. (2014)
22. Koret, J., Bachaalany, E.: The Antivirus Hacker's Handbook. Wiley, 1st edn. (2015)
23. Lee, J., Lee, Y., Jin, M., Kim, J., Hong, J.: Analysis of application installation logs on android systems. SAC '19, ACM (2019)
24. McNab, N., Bryan, A.: An implementation of the linux software repository model for other operating systems. HotSWUp '09, ACM (2009)
25. Pietrek, M.: Peering inside the pe: A tour of the win32 portable executable file format. https://msdn.microsoft.com/en-us/library/ms809762.aspx (1994)

26. Potter, B., Fleck, B.: 802.11 Security. O'Reilly (2002)
27. Rossow, C., Dietrich, C., Bos, H.: Large-Scale Analysis of Malware Downloaders. Springer (2013)
28. Sans: Malware delivered via windows installer files. https://isc.sans.edu/forums/diary/Malware+Delivered+via+Windows+Installer+Files/23349/ (2018)
29. Scrapy: Scrapy. https://www.scrapy.org (2019)
30. Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: Avclass: A tool for massive malware labeling. In: RAID. Springer (2016)
31. Sikorski, M., Honig, A.: Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, San Francisco, CA, USA, 1st edn. (2012)
32. Softpedia: Softpedia. https://www.softpedia.com/ (2019)
33. Software, E.: Artemis trojan. https://www.enigmasoftware.com/artemistrojan-removal/ (2013)
34. SourceForge: Sourceforge. https://www.sourceforge.net (2019)
35. Total, V.: Virus total. https://www.virustotal.com (2019)
36. Vaughan-Nichols, S.J.: Why software development is lagging hardware improvements. https://www.cio.com/article/2431061/from-32-bit-to-64-bit--why-software-development-is-lagging-hardware-improvements.html (2009)
37. Wang, H., Li, H., Li, L., Guo, Y., Xu, G.: Why are android apps removed from google play?: A large-scale empirical study. MSR, ACM (2018)
38. Whitwam, R.: Asus live update pushed malware to 1 million pcs. https://www.extremetech.com/internet/288283-asus-update-servers-pushed-malware-to-hundreds-of-thousands-of-pcs (2019)
39. Willems, C., Freiling, F.C., Holz, T.: Using memory management to detect and extract illegitimate code for malware analysis. ACSAC, ACM (2012)
40. Williams, C.C., Hollingsworth, J.K.: Automatic mining of source code repositories to improve bug finding techniques. IEEE Transactions on Software Engineering (2005)
41. Zope, M.: Unattended installation and uninstallation of softwares remotely. ICWET, ACM (2010)