Dissecting Applications Uninstallers & Removers: Are they effective?

Marcus Botacin[1,2] and André Grégio[2]

[1]Texas A&M University, USA
[2]Federal University of Paraná (UFPR), Brazil
{mfbotacin, gregio}@inf.ufpr.br

Abstract. Developing a safe application is so important as to properly install it in a system, and not an application's tampered version. In a similar note, developers should properly care about applications' uninstall process to avoid leaving traces of sensitive data behind in the system or interfere with the remaining applications. Until now, the academic literature has paid little attention to uninstall procedures so far. Moreover, a whole ecosystem of application uninstallers has been created, making multiple uninstallers available in software repositories. A key point is to understand how these applications work so as to develop stronger systems. To this end, we present a landscape work evaluating the operation of the 11 most downloaded uninstaller applications from the three most popular Internet software repositories. We discovered that most of these applications are not very different from the native Windows uninstaller. Although evaluated uninstallers present a more organized User Interface, thus enhancing usability, they are only able to find the same installed application as the native Windows uninstaller, but not broken installations. Few uninstallers apply heuristics to find broken application installations. However, we show that those heuristics can be abused by attackers to remove third applications. Finally, we also show that none of the removers is resistant to malicious uninstallers that terminate the remover process.

Keywords: Uninstaller · Installer · Removal · Malicious Code

1 Introduction

Safety and security are key aspects of any modern application. Thus, to cope with the safety and security requirements of a modern application, software engineers are often looking for ways of writing better code [11, 19]. However, the challenge does not finish there. As important as developing a safe and secure application is to properly install this application in a system and not a tampered/vulnerable version of it [1]. Similarly, as important as properly installing an application is to uninstall it to not leave traces of sensitive data in the system [10] or interfere with the remaining applications.

Unfortunately, the academic literature has been giving little attention to uninstall procedures so far, and few to no works on the subject can be found in

the major research paper databases. It causes us to have a poor understanding of an ongoing phenomenon: the popularity of application uninstallers, which can be found at hundreds in any popular Internet software repository and with a large number of downloads (e.g., 80K for `IoBit` and 400K for `Revo` in the Softonic repository). Application uninstallers (or removers) are often recommended by users in forums and/or websites [9] to be used in the cases where the native Windows uninstaller fails, but its consequences are not well understood.

To bridge this gap, we delve into the internals of Windows uninstallers to present a landscape of the operation of application uninstallers in this platform. We selected the 11 most popular apps from 3 popular software repositories (CNET [3], Softonic [22], and Softpedia [23]) and completely analyzed their operations regarding their removal capabilities, interactions with the user and with the operating system.

We discovered that, on the one hand, most of these applications are not very different from the native Windows uninstaller in operation, often displaying the same installed apps with no additional capability of searching for broken installations. On the other hand, the User Interfaces presented by these applications are clearly more detailed than the Windows' native one, presenting much more information, which might explain user's preference for them.

A few installers present advanced uninstall capabilities, such as the ability to perform system checkpoints or the application of heuristics to find files remaining from broken installations. We discovered that these capabilities and heuristics implicitly assume that the targeted uninstaller will be well-behaved. We demonstrate that multiple attacks are possible if a malicious uninstaller is the target of them, such as removing third-party files and even processes termination. In this scenario, the usage of the uninstallers would cause more harm than good.

In summary, our contributions are as follows: **(1)** We contextualize the usage of application uninstallers and the challenges associated with their use; **(2)** We present a summary of legitimate and malicious uninstaller's operations on Windows; **(3)** We discuss the limits of their application to the removal of protected applications.

This paper is organized as follows: In Section 2, we introduce related work and discuss the gap of understanding on the operation of uninstallers; In Section 3, we revisit the operation of uninstallers on the Windows system; In Section 4, we introduce the methodology we adopted to conduct our experiments and the research questions we aim to answer; In Section 5, we present experiments results regarding the actual operation of popular uninstallers; In Section 6, we discuss the implications of our findings; In Section 7, we draw our conclusions.

2 Related Work: Why Studying Uninstallers?

Before we explain how we evaluated uninstallers, it is key to understand why evaluating them is important. When a user is not satisfied with a software piece and wants to remove it, the straightforward option is to use the native Windows uninstall solution. However, it is not rare to find cases where the native uninstaller

fails to remove an application. In these cases, it is common that users try to use standalone removal tools (Uninstallers and Removers), since it is also common to find websites recommending the use of this type of solution [9]. This ends up creating an entire ecosystem of application uninstallers, as can be found in any popular software repository (e.g., CNET [3]). This ecosystem must be studied and understood, as uninstalling software is as safety- and security-critical for a system as installing new ones.

Whereas the academic literature is full of good research on the software development topic (e.g., secure development and coding practices [11, 19]), little attention has been given so far to the problem of uninstalling applications: We could find almost no research work in the main research paper databases (e.g., ACM, IEEE, Springer, and so on). Meanwhile, the largest part of the information users can find about uninstallers is delivered via grey literature [24] (i.e., websites, blog posts, and so on). Unfortunately, publications in this type of literature often do not present formal evaluations of uninstallers or a strong methodology to evaluate them, such that we understand that there is currently an understanding gap about the operation of these solutions.

We believe the subject has been given little attention so far because the uninstallation problem is often seen in the literature as a management problem [18] rather than also a technological problem. A few works in the literature address the uninstallation problem from a more technical point-of-view and, if so, they are very limited in scope (e.g., a forensic analysis of uninstalled steganography apps [28]). We believe that uninstallers must be studied more broadly to present a landscape, as recently done for applications installers [1].

The related work on application installers already pinpointed some limitations of the associated uninstallers, such as the improper definition of the uninstaller executable [1]. In this work, we aim to go further and analyze the behavior of the applications designed to uninstall these failure-prone applications. For instance, we aim to evaluate whether uninstallers can clean registry keys after the application removal. Previous research work has demonstrated that potentially sensitive information remained resident in the registry after the uninstall of some specific software [10].

Therefore, this work aims to shed light on the greater scenario of uninstallers operation. It is important to notice that application uninstallers are available to most platforms (e.g., Android [7]), but we focused our efforts in this paper on the Windows platform due to the popularity of uninstallers in this ecosystem.

3 Background: How Windows supports uninstallers?

Before understanding how third-party uninstallers operate, it is key to understand how Windows applications are installed and removed natively. On Windows, installers should register the installed applications with the Windows registry by creating an entry in the proper registry branch [13]. Applications installed for a single user must add their information to the key at `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\AppPaths`. Applications installed for all

users in the machine should add their information to the key at `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\AppPaths`.

The Windows' "Installed Apps" menu gathers information from these keys to display the installed apps and their removal/edit options. The first uninstallation problem is that nothing prevents an app from not registering with Windows (e.g., executable files directly extracted from compressed files–zips). In these cases, the application will be not found in the Windows' "Installed Apps" menu.

When the application registers itself with Windows, it must set some required registry keys, such as the application name, the provider, and the uninstaller path. Therefore, when a user requires an application to be uninstalled by the native Windows uninstaller, the uninstaller checks the path stored in this registry key and launches the registered removal process.

The problem with this approach is that nothing prevents an application to register a fake or a broken uninstall path, such that the uninstaller will not be able to create a process from it. In this case, the application is never really uninstalled. Although this is considered a bad practice according to the security policies of large software ecosystem providers (e.g., Google [8], Microsoft [17]), this strategy is often used by many applications available for user's download.

Another problem is that Windows completely trusts the invoked uninstaller to remove the application files and keys. However, if the application's native installer does not do a great job removing its own application, files and registry keys will remain in the system. Overall, there are many reasons why an application might be not properly removed, for instance:

- Installations without registering associated keys [12].
- Installers setting the `APPNOREMOVE` key [14], that prevents the native uninstall to launch the uninstall process.
- Implementation bugs, such as applications setting registry keys greater than 60 chars [15], which is unsupported by Windows.

When we consider the possibilities above discussed, we notice that the process of uninstalling an application is not straightforward. Therefore, we consider that understanding how uninstallers handle those conditions is essential for developing better applications.

The role of the third-party uninstallers. Face to this challenging removal scenario, third-party uninstallers promise to succeed in the cases where conventional removal fails. They promise not only to remove the applications listed by the Windows, but also to discover the ones that did not register with the Windows, remove files from previous, broken installation, and even defeat protections that prevent a software to be uninstalled. All these cases are evaluated in this paper.

4 Methodology: What do we aim to discover? And how?

In this section, we present the questions we aimed to answer, the applications we considered in our analysis, and the approach for inspecting them.

Research Questions. We defined the following Research Questions (RQs) to help us to understand the uninstallers:

- **RQ1. What is the anatomy of the uninstallers?** This question aims to answer what are the modules and components of a typical uninstaller. It also aims to answer how uninstallers are structured.
  - **RQ1.1 Are there applications bundled in the uninstallers?** This derived question aims to answer whether additional components not essential to the uninstallers operations are added to them.
- **RQ2. How do uninstallers operate?** This question aims to answer how uninstallers interact with system components.
  - **RQ2.1 Do uninstallers include extra features?** This derived question aims to answer whether uninstallers provide non-traditional mechanisms to uninstall applications.
- **RQ3. What is the difference for the native uninstaller?** This question aims to answer whether there is any significant advantage on migrating to a standalone application.
- **RQ4. Do uninstallers handle drivers, services, and privileged components?** This question aims to answer what are the limits of uninstallers operations.
- **RQ5. Are there any performance gains in using an uninstaller?** This question aims to verify if claims of perceived performance gains made by some vendors and users are real.
- **RQ6. Do uninstallers leave files in the system?** This question aims to answer what is the potential of uninstallers for cleaning files.
  - **RQ6.1 Did uninstallers evolve?**. We repeated the experiments reported in the literature [10] to check whether the scenario changed over time.
- **RQ7. Are uninstallers able to remove protected applications?** This question aims to answer what are the capabilities of the uninstallers.
  - **RQ7.1 Are uninstallers resistant to tampering attempts?** This derived question aims to answer how resistant to malicious applications uninstallers are.
  - **RQ7.2 Are uninstallers able to remove malware?** This question aims to answer whether uninstallers can unlock resources from malicious processes, such as hypothesized by some users in forums [4, 5, 6].

Uninstallers Selection. To provide a landscape of the uninstallers, we followed the same strategy adopted in the reference study of application installers [1]: the search for applications in the most popular online software repositories. Our search in December/2021 revealed the existence of 275 uninstallers entries for the uninstaller keyword for CNET [3], 6 thousand for Softonic [22] (unfiltered), and 268 for Softpedia [23].

Unfortunately, we cannot handle these amounts via manual analysis, as required by the experiments we designed, such that we opted for selecting the applications ranked first in the repositories (the most downloaded ones),

Table 1: **Selected Uninstallers.** We selected the most popular applications that were successfully downloaded and not part of a security solution. Columns represent, respectively, tool name, tool version, if they are embedded in security solutions (✓) or not (✗), if they were successfully downloaded (✓) or not (✗), if they run on VM (✓) or not (✗), and their ranking in the repositories. Empty fields mean that the data is not available and/or the criteria does not apply.

Uninstaller	Version	Security	Downloaded	VM	CNET	Softonic	Softpedia
Iobit	11.0.1.14	✗	✓	✓	1	2	
Revo	2.1.5.0	✗	✓	✓	2	1	2
Your	7.5.2014.3	✗	✓	✓	3		3
Advanced		✗	✗		4	6	8
Easy		✗	✗		5		
Wise	2.3.6.140	✗	✓	✓	6		
Ashampoo	10.10.00.13	✗	✓	✓	7	9	15
Zsoft	2.5	✗	✓	✓	8		12
Anvi	1.0	✗	✓	✓	9		
Smarty	4.9.6	✗	✓	✓	10		11
Puran	3.0	✗	✓	✓	11		
Handy	1.2	✗	✓	✓	12	12	
Absolute	5.3.1.26	✗	✓	✓	13		9
Ccleaner		✓			14		
Bazooka Adware		✓			15		
Uninstall Manager		✗	✗		16		
Total Uninstall	6.16.0	✗	✓	✗	17	10	5

hypothesizing them to be more representative of the solutions most users install in their machines. We found a low agreement between the ranks of all repositories, such that we tried to maximize our coverage by considering the most popular apps in the higher rank positions in the majority of the repositories. We discovered that considering the CNET rank as a reference was the selection that maximize the rank position coverage.

From all possible uninstallers to be selected, we discarded those that were not successfully downloaded (e.g., server errors on the repository side and/or corrupted files), those that did not execute in Virtual Machines (VMs)–used for tests–,and also discarded those that were part of security solutions, since the analysis of security solutions is different from our goal of analyzing the uninstallers by themselves (specific analyses are reported in the academic literature [2]). Table 1 shows the 11 selected uninstallers and their respective rank positions for the multiple repositories. We also show the downloaded apps' versions for the sake of reproducibility.

Removed Apps Selection. The applications we targeted to remove using the uninstallers were the native applications installed with Windows, additional Microsoft products installed in typical user's machines (e.g., Office), and the top-10 most popular applications used by the users (e.g., browsers) according to the rankings of the same repositories that we downloaded the uninstallers. The

number of applications used in each experiment varied according to their goals: a random one, when designing a malicious uninstaller; all of them simultaneously, when evaluating the presence of sensitive information.

Analysis Methodology: To inspect the uninstaller applications, we manually installed each of them into a fresh Virtual Machine (VM) and inspected the installed files (static analysis) and their interaction with system components while we interacted with the application's UI (interactive dynamic analysis). All monitoring was performed using either Microsoft native tools, such as `regedit`, for registry inspection, or Microsoft complements, such as SysInternals [16], for advanced system state inspection. The VMs were restored to the original state after each uninstaller was tested.

Copyright Information: During our experiments, no decompilation was performed so as to not violate the creator's copyright. All analyses were performed by statically examining the installed software files and/or the behavior of the applications during their normal execution.

5 Evaluation: What we discovered?

In this section, we present our experiment's results and show how they contribute to understanding uninstaller's operations.

5.1 RQ1. The Anatomy

We started our investigation by analyzing the structure of the uninstallers, as summarized in Table 2. Our initial goal was to use the complexity of their constructions as a proxy for evaluating the complexity of their operations. Our initial hypothesis was that these installers would be complex pieces of software that integrate with multiple parts of the system to be able to perform the removal actions that the installers and the OS itself was not able to perform. Instead, we found that most applications were simpler than expected.

In fact, some uninstallers are even standalone applications, operating from a self-contained binary, which embeds all capabilities and presents all data hard-coded within it. Their simplicity associated with the great number of Windows libraries imported by them makes us to hypothesize then that they operate only like wrappers for invoking the proper Windows native functions designed to remove software.

Being a standalone application is not a problem, simplicity is desired, since the application is well-designed and follows the best practices. Interestingly, in the case of the `Anvi` installer, the standalone application does not register its own binary with the system, thus not appearing in the list of installed software. In some sense, the uninstaller application acts the same way that the software it is designed to remove acts.

Even the uninstallers that are not standalone are not complex, with a minority presenting even libraries. We hypothesized initially that libraries would be used

Table 2: **Uninstallers Anatomy.** Files and Components. Columns represent, respectively, tool name, if they are registered with the Windows (✓) or not (✗), the number of executable files it is composed of, if it is composed of shared libraries (✓) or not (✗), the number of kernel drivers composing the tool, if the applications stores data in databases (✓) or not (✗), and if it relies on configuration files (✓) or not (✗).

Unnstaller	Register	EXE	DLL	Drv	DB	Config
Iobit	✓	21	✓	3	✓	✗
Revo	✓	1	✗	✗	✗	✗
Your	✓	6	✗	✗	✗	✓
Wise	✓	3	✗	✗	✓	✓
Ashampoo	✓	6	✓	✗	✓	✗
ZSoft	✓	2	✗	✗	✗	✓
Anvi	✗	1	✗	✗	✗	✗
Smarty	✓	2	✓	✗	✗	✗
Puran	✓	2	✗	✗	✗	✓
Handy	✓	1	✗	✗	✗	✗
Absolute	✓	4	✓	1	✗	✗

to implement custom removal algorithms, but we mostly found libraries used for compatibility (e.g., zlib for compression support). Similar reasoning can be applied for kernel drivers, which are found only in two uninstallers. Most uninstallers operate in the same privilege level as the software they aim to remove.

Even when they are not standalone binaries, most uninstallers do not keep usage sessions in the current system. We only found 3 uninstallers storing information about the system in databases (all cases in `sqlite` databases). In most cases, they simply use information hardcoded in the binaries or gathered from configuration (config) files (often stored in plain) to increase their removal capabilities and intelligence.

In the case of the `Wise` uninstaller, the configuration file stores a list of ratings for popular applications and also an application exclusion list (e.g., Firefox, Chrome, Opera), whose files will not be touched by the uninstaller, limiting the aggressiveness of the heuristics, but also the removal capabilities. Lists are also found in the `Ashampoo` uninstaller. In this case, a whitelist, protected against modifications only by filesystem permissions, is used by the application. It means that the applications present in the whitelist will not be removed from a system by the removal software and its heuristics. The approach used by the `ZSoft` uninstaller is of blocklisting applications. It means that the removal software looks for the presence of specific, known "bad" applications to be removed. We identified that the "bad" reputation of a given software is given by the `StopBadware` list [25], present in the configuration files of this application.

RQ1.1. Bundling Most uninstaller applications are distributed in limited forms to incent a purchase, as shown in Table 3. As a limitation, a few of them

will display ads to the user. The worst case, however, is when the uninstallers are delivered with additional packages bundled in the original file. In this case, whereas users are looking for an application to remove software from their system, the final result is that more software is installed, which is not reasonable.

Table 3: **Application Bundling.** Some uninstallers distribute other applications during their installation. Columns shows, respectively, the tool name, the type of contented bundled with it, if it displays ads (✓) or not (✗) and of which type (if available), and the license type.

Uninstaller	**Bundled**	**Ads**	**Type**
Iobit	Itop suite Itop screen recorder Itop vpn	Opera Driverbooster	Freemium
Revo	✗	✗	Freemium
Your	✗	✗	Shareware
Wise	✗	✗	Freeware
Ashampoo	✗	✗	Shareware
ZSoft	✗	✗	Freeware
Anvi	✗	✗	Shareware
Smarty	✗	✗	Shareware
Puran	✗	✗	Freeware
Handy	✗	Random Ads	Freeware
Absolute	Games	✗	Freeware

5.2 RQ2. Operation

The key goal of our investigation is to discover how uninstallers really remove the applications. For such, we requested the evaluated uninstallers to remove multiple applications under different settings. A summary of the uninstaller's capabilities is shown in Table 4.

Our first finding is that in an overall manner the uninstallers display the same installed application as Windows, which indicates that they search the same registry locations. Only two uninstallers performed a full registry search. On the one hand, broader searches are useful to find software not installed in standard locations. On the other hand, this strategy ends up generating false positive reports.

We discovered that a universal uninstaller's strategy is to invoke the original uninstaller of the application to be removed before taking any other action to remove the software. This strategy is adopted by all evaluated applications. Whereas some applications seem limited to this functionality, acting only as another GUI for the removal process, some applications try to complement the removal procedure. In this sense, the uninstaller's philosophy seems more to try to clean residual entries than trying to remove applications by themselves.

Table 4: **Uninstallers Operation.** Uninstallers first invoke the native uninstaller. Some apply heuristics after that. Columns show, respectively, tool name, if the list of installed apps is retrieved from the Windows subsystem or directly from the registry, if it invokes the native uninstaller (✓) or not (✗), if the removal process is automated (✓) or not (✗), and if it has custom removal heuristics (✓) or not (✗).

Uninstaller	List	Native	Auto	Custom
Iobit	Windows	✓	✓	✗
Revo	Windows	✓	✓	✗
Your	Windows	✓	✓	✗
Wise	Windows	✓	✓	✗
Ashampoo	Windows	✓	✗	✓
ZSoft	Registry	✓	✗	✗
Anvi	Windows	✓	✗	✓
Smarty	Windows	✓	✓	✗
Puran	Windows	✓	✗	✓
Handy	Registry	✓	✗	✗
Absolute	Windows	✓	✗	✗

The strategies used to perform additional cleanings are varied. Some uninstallers perform custom scans, asking the user if they want to remove a given file. It does not seem to be a significant advantage in comparison to manual removal procedures. The only clear benefit in it is to automatically locate files, but no decision is taken by the application. Other uninstallers try to add intelligence to the process by employing heuristics to automatically identify which files must be removed.

To evaluate the identified third-party uninstaller's capabilities in practice, we created a crafted application installation with an integrated custom uninstaller that purposely did not remove registry keys and installation files. We applied the third-party uninstallers to check their actions over the remaining installation artifacts. We summarize the results in Table 5.

All uninstallers were able to run the native uninstaller (our custom one) and thus remove the application from the list of installed apps. However, it does not mean that applications were fully uninstalled by all of them. Not all uninstallers were able to wipe the registry entries associated with the application that were intentionally left by our uninstaller. Similarly, not all of them were able to follow the paths stored in the registry keys and delete the files intentionally left in the filesystem. The only solutions able to perform this type of uninstallation were the ones using heuristics. The heuristic used by the uninstallers is to follow the path added to the `InstallLocation` key and suggest the removal of whatever is pointed by it.

A major drawback of using heuristics is that they provide no guarantee that they will remove correct files, and this characteristic might even be abused. To demonstrate that, we configured an application installation whose `InstallLocation` path points to another application's folders, unrelated to our targeted application.

Table 5: **Removal Experiment.** Heuristics might be tricked to remove the wrong files. Columns represent, respectively, tool name, if the tool was able to remove apps from the installed apps list (✓) or not (✗), if they were able to remove associated registry keys (✓) or not (✗), if they were able to remove associated files (✓) or not (✗), and if they are prone to remove wrong files (✓) or not (✗).

Uninstaller	List	Registry	Files	Wrong
Iobit	✓	✓	✓	✓
Revo	✓	✓	✓	✓
Your	✓	✓	✗	✗
Wise	✓	✓	✓	✓
Ashampoo	✓	✓	✓	✓
ZSoft	✓	✓	✗	✗
Anvi	✓	✗	✗	✗
Smarty	✓	✓	✓	✓
Puran	✓	✗	✗	✗
Handy	✓	✓	✗	✗
Absolute	✓	✗	✗	✗

In all cases, the installers suggested removing the unrelated folders, even when we pointed them to native Windows folders, which might even break the system operation.

RQ2.1. Extra Features In addition to their original function of uninstalling applications, many uninstallers also offer other facilities to the users. Table 6 summarizes the extra features we found on the evaluated uninstallers.

Many features offered by the installers are focused on usability rather than on the removal process itself (e.g., identifying very large files, rarely used files, unattended update files left in the system, and so on). Some facilities are related to the removal but do not involve a specific process (e.g., cleaning the system), such that they were evaluated separately.

Two extra features are of our particular interest when evaluating uninstallers: (i) the ability to monitor new installations, and (ii) the ability of performing system checkpoints. These two functions are not natively provided by Windows and they would be useful to the users. We discovered, however, that these two functions are very limited in all uninstallers. We discovered, for instance, that the so-called monitors do not perform whole-system monitoring, as expected. Instead, they only search specific locations and registry keys, such that standalone installers are not identified (e.g., EXE files extracted from zip folders) and manual registry edits are also not reported.

The checkpoint mechanism works similarly on both evaluated uninstallers that offer this capability. The user takes a snapshot of the current system state (files and registry keys) before installing an application, installs it, and takes a

Table 6: **Extra Features.** Some uninstallers present additional monitoring and management resources. Columns show, respectively, tool name, if they filter installers by size (✓) or not (✗), by usage frequency (✓) or not (✗), if they handled installed updates (✓) or not (✗), if they have cleaning capabilities (✓) or not (✗), if they monitor new installations (✓) or not (✗), and if they create installation checkpoints (✓) or not (✗).

Uninstaller	Size	Freq.	Upd.	Clean.	Mon.	Checkpoint
Iobit	✓	✓	✗	✗	✓	✗
Revo	✗	✗	✗	✗	✗	✗
Your	✗	✗	✗	✓	✗	✗
Wise	✗	✗	✗	✗	✗	✗
Ashampoo	✗	✗	✓	✗	✓	✗
ZSoft	✗	✗	✗	✗	✗	✓
Anvi	✓	✗	✗	✗	✗	✗
Smarty	✗	✗	✗	✗	✗	✓
Puran	✗	✗	✗	✗	✗	✗
Handy	✗	✗	✓	✗	✗	✗
Absolute	✗	✓	✓	✗	✗	✗

new snapshot after it. The newly added files and registry keys are then reported. If the user asks for application removal, these files will be removed.

The snapshot mechanism is very fast, such that we hypothesized that the uninstallers do not look for file contents (not even a hash/digest). We evaluated this hypothesis by modifying an existing file and we discovered that this was also reported as a new file. We then hypothesized that the uninstaller was identifying it via the filesystem's modification time. We confirmed that by re-saving files, with no actual modification, during the snapshot, such that these files were reported as new. Whereas timing-efficient, this approach is problematic because it reports any modified file as belonging to the installed application and suggests its removal. When we modified a system file, this file was also suggested for removal, which might break the system operation.

5.3 RQ3. The Differences

One of the goals of this research work is to investigate the reasons why one would prefer a third-part uninstaller than the native solution. We did not discover many differences to support such migration, except the one here discussed.

Applications might mark themselves to not be removed by setting the `APPNOREMOVE` key in the registry. In this case, the native Windows uninstaller will not display the uninstall button for that application, even though the application will still be displayed in the list of installed apps. We evaluated the behavior of the other uninstallers in this case. Table 7 shows that all uninstallers except for `ZSoft` simply ignore this registry key and invoke the registered uninstaller anyway. Whereas skilled users might perform manual registry editing to remove the key and allow the native Windows uninstaller to remove the application, we

Table 7: **Applications with `NOREMOVE` option.** Most uninstallers simply ignore the option. The set of columns show, respectively, tool name and if the tool was able to remove applications with the `NOREMOVE` option set (✓) or not (✗).

Uninstaller	Removed	Uninstaller	Removed
Windows	✗	ZSoft	✗
IoBit	✓	Anvi	✓
Revo	✓	Smarty	✓
Your	✓	Puran	✓
Wise	✓	Handy	✓
Ashampoo	✓	Absolute	✓

consider that in this scenario the third-part uninstallers perform better than the native one because even users with less knowledge about Windows internals can remove applications in this setting.

5.4 RQ4. Privileges

As important as to identify that a given resource must be removed is to have the ability to remove it. In practice, this might be challenging due to permission issues (e.g., a file might be locked, a process might still be running, the access to a key might require admin privileges). Therefore, we aimed to evaluate how uninstallers handle these conditions. We discovered that, since the uninstallers rely on the invocation of the original uninstaller, they have almost the same capabilities as them. In this sense, the uninstallers do not elevate themselves to admin, but they wait for the native uninstallers to do so to remove the files. If they do, the files are removed. If they do not remove a file that requires special permissions, the standalone uninstallers will not be able to remove them as well. Similarly, if the native uninstaller unloads kernel drivers and stops services, these will be removed. However, if the standalone uninstallers are required to remove them while running, any attempt will fail due to the lack of proper permissions. Whereas we understand that this scenario is somehow expected and thus acceptable, most uninstallers do not make it clear to the users. In our searches, we found that only the `Revo` uninstaller stated in its manual that drivers must be removed in safe mode [26].

5.5 RQ5. Performance

It is common to find users in Web forums recommending the use of uninstallers and/or cleaners to speed up system performance. The hypothesis behind it is that having fewer files and/or registry keys in the system would make searches faster, as the system would have to traverse smaller structures, which is a reasonable hypothesis at a first glance [20]. This “popular knowledge” become widespread to the point of that some solutions even advertise performance gains. Therefore, it is important to investigate to which extent these supposed performance gains are significant.

Two of the uninstallers solutions that we evaluated made explicit performance claims: `You Uninstaller` and `Ashampoo`. In the first case, the advertised cleaning function is, in fact, limited to a few pre-defined locations, such as browser's history, cookies, and so on. Even though this might have a (limited) impact on navigation, it is hard to consider these actions as a performance improvement to the system. In the second case, the uninstaller presents a solution to clean the registry tree as a whole. We evaluated its impact by taking a snapshot of the registry tree before and after the cleaning. We discovered that the solution is very conservative when removing keys. It only removes registry entries with no associated keys (empty), but it does not remove orphaned keys (e.g., keys that point to invalid paths). On the one hand, the solution works both for the current user (HKCU) as well as for the other uses (HKLM). On the other hand, it is very conservative and does not touch keys that affect the system (e.g., HKCR).

In the end, after an average of 10 repetitions, the solution removed 500 keys from an average of 318 thousand keys with 564 thousand associated values present in our fresh Windows installation. We consider that this result (less than 0.2% effect) is not significant to support claims of performance gains. To confirm this hypothesis, we executed 10 repetitions of a Windows registry benchmarking tool [21] and measured the depth of the traversed registry branches and the actual time spent traversing them. We noticed no statistical difference between the system state before and after the system cleanup.

5.6 RQ6. Remaining Files

A good uninstaller application should be able to remove all traces of an installed application from the registry and filesystem. Of course, this is the main task of the native application uninstaller, but since in this work we assume the user is using some other application because the native uninstaller already failed, we would like to verify whether the third-part uninstallers are able to bridge this gap. Unfortunately, they are not. Since most uninstallers only invoke the native uninstaller to remove the application, they end up failing in the same aspects as the original uninstaller. The uninstallers that perform additional heuristic checks indeed remove more files, but no uninstaller completely removed all files of any application we tested. This essentially happened because applications often install their files in two distinct locations: `Program Files`, for the binaries; and `AppData`, for the configuration files. Since only the first location is affected by the requirements to register the application with the Windows registry, the heuristics are only able to correlate this location with the installed application, always leaving the second folder untouched.

RQ6.1. Evolution A major problem with files and registry keys left in the filesystem is that they can potentially reveal sensitive information about the users, as demonstrated by a previous work [10]. We repeated their experiment to verify if the situation improved with time and if the use of third-party uninstallers is a viable option for cleaning the system after an uninstall. To do so, we installed the

same application considered in the original work (in updated versions) in a fresh Windows installation. We populated these applications with data from an entire day of use. For instance, for the mail client, we registered an account in it and sent and received emails. We inspected the filesystem and the registry after we uninstalled the applications using the native uninstaller and the third-part ones. We notice that no clear sensitive information is left by the native uninstaller (e.g., no key storing email addresses), which we might credit to enhancement to the native uninstaller itself over time. On the other hand, configuration and temporary files were still spread all over the filesystem after the installation. The files stored in the `AppData` folder were cleaned by the third-part uninstallers, but there were remaining files in other folders after the application of all solutions. Therefore, we conclude that whereas uninstallers might help removing some orphan files, they are not the solution for definitively eliminating all files, especially if one is concerned with privacy leaks.

5.7 RQ7. Protection

If uninstallers are supposed to remove badly-behaved applications, they should be protected at least against the basic types of interference attempts, such as termination. We inspected the uninstallers in search of signs of self-protection mechanisms to evaluate their protection level. We did not find, however, for all uninstallers, OS-independent protection mechanisms, which indicates that they assume that the software they will uninstall is well-behaved (i.e., they will operate following the best standards, ordinary methods, and not abusing interfaces).

For two uninstallers, we identified components that could be used to increase self-protection (e.g., kernel drivers that could be used to prevent access to the uninstaller files). We discovered, however, that these components are only part of the uninstaller engine and not part of self-protection modules (e.g., kernel drivers are used as callback mechanisms). In the case of the `Absolute` uninstaller, the driver could be terminated by any user/process having admin privileges (in the last instance, it could be even the application requested to be uninstalled). In the case of the `IoBit`, there were 3 drivers running in our test environment (responsible for the process, registry, and filesystem callbacks, respectively). Whereas the first two were resistant to termination due to the lack of proper permissions, the filesystem driver was easily terminated by the admin (which we interpreted as a bug, since the other 2 drivers were protected).

RQ7.1. Anti-Tampering To demonstrate uninstallers vulnerabilities due to the lack of self-protection mechanisms, we developed some attacks[1] that could be leveraged by a malicious application to not be removed by an uninstalling application.

[1] Attack demos available at: `https://www.youtube.com/watch?v=Rkw6WbD-nMY`, `https://www.youtube.com/watch?v=mZPb7h4cy80`, and `https://www.youtube.com/watch?v=0AjFCZWUhfU`

Our first attack is based on the fact that the standalone uninstallers directly call from their main process the application registered in the registry as an uninstaller for the target application rather than calling them from a child/protected process. This allows the targeted uninstaller to identify the PID of their parent processes (the standalone uninstallers) and directly attempt to terminate this Process ID (PID). If no protection mechanism is employed, the attack will succeed and the targeted application remains installed in the system.

Table 8: **Uninstaller Termination.** Uninstallers can be terminated by the targeted uninstall application. The set of columns show, respectively, tool names, and if the installers were terminate (✓) or not (✗) by a malicious uninstaller.

Uninstaller	**Terminated**	**Uninstaller**	**Terminated**
Windows	Crashed	ZSoft	✓
IoBit	✓	Anvi	✓
Revo	✓	Smarty	✓
Your	✓	Puran	✓
Wise	✓	Handy	✓
Ashampoo	✓	Absolute	✓

We developed a Proof-of-Concept (PoC) uninstall application for this attack and registered it as the uninstall of an application to be removed by the standalone uninstallers. Table 8 shows this experiment's results. Whereas the Windows installer crashed, but did not terminate, all standalone uninstallers terminated before removing the targeted application.

The problem with the first attack is that terminating the application is noticeable for the user and might raise concerns. A more effective strategy would be to remove the application from the list of installed software without actually uninstalling it. We developed a second class of attacks with this goal by exploiting the facts that (i) installers have no self-protection mechanisms; and (ii) they rely on standard system interfaces for their operation.

Our second attack consisted of injecting a DLL into the uninstaller applications to hook the Windows APIs used by the uninstallers to remove our PoC application from the installed applications list. In other words, we developed a userland rootkit. Table 9 shows that we were able to inject the DLL and remove applications from the list of all standalone uninstallers when the processes were already launched with the injected DLL. It also shows that injection was possible in runtime into all but three uninstallers. The two failure cases are due to their process being protected against memory writes after the process setup phase, which is the only self-protection measure we found among all uninstallers we inspected.

We also developed a third attack that does not depend on code injection to demonstrate that the reliance on OS APIs is the weakest point of the uninstaller's security model. We developed a kernel driver that implements callbacks to prevent uninstallers from accessing registry keys associated with the targeted application, which could be performed by a malicious software that prevents uninstallations.

Table 9: **Uninstaller Tampering.** External code might affect uninstaller's operations. Columns show, respectively, the tool name, if the tool is affected by code injection at startup (✓) or not (✗), at runtime (✓) or not (✗), and by external kernel drivers (✓) or not (✗).

Uninstaller	**Userland**		**Kernel**
	Startup	**Runtime**	
IoBit	✓	✗	✓
Revo	✓	✗	✓
Your	✓	✓	✓
Wise	✓	✓	✓
Ashampoo	✓	✓	✓
ZSoft	✓	✓	✓
Anvi	✓	✓	✓
Smarty	✓	✗	✓
Puran	✓	✓	✓
Handy	✓	✓	✓
Absolute	✓	✓	✓

In other words, we developed a kernel rootkit. Table 9 shows that this strategy succeeded against all uninstallers, removing the targeted application from the list of installed software without actually removing the application from the system.

RQ7.2. Anti-Malware Considering the self-protection limitations that we presented above, we understand that uninstallers are not suitable as a replacement for security solutions for the task of malware detection, as suggested in some Web forums, since an armored malware could terminate them or interfere with their operation. Moreover, we also did not find evidence of an actual capability of removing malware traces in any of the solutions. In our tests, we infected a system with multiple samples collected from VirusShare [27] (10 randomly-chosen samples tested against each uninstaller) that created `AutoRun` keys for persisting in the registry. Even after we manually removed the malicious binaries from the system and left the keys orphan, the uninstallers were still not able to detect it and remove these entries from the registry. In other words, the uninstallers were not able to identify that the malware samples were the original parents of the leftover AutoRun keys. This happens mostly because the removal heuristics used by the uninstallers rely on data that is often set by benign applications (paths in registry keys) that were not set by the malware samples.

6 Discussion: What are the implications of our findings?

Based on our findings, we here present a brief discussion about some implications of our findings.

The need for better uninstallers using current technology. Whereas most of the investigated uninstallers did not present significantly greater removal capabilities than the native Windows one, all of them present a better user interface (UI)/user experience (UX), in the sense that applications are categorized, ranked, locations are displayed, and so on (see RQ 2.1). These information pieces are not available in the standard Windows tool. We believe that this might be one of the reasons why users adopt this kind of solution rather than using the native installer. Therefore, OS developers (e.g., Microsoft/Windows) should investigate refactoring and enhancement possibilities of their native solutions. We believe that incorporating the features from the third-part solutions to a native system is an immediately applicable action that does not depend on the development of new technologies.

The need for better uninstallers using next-gen technology. Whereas enhancing the UI/UX is an immediately applicable action, enhancing the removal capabilities depends on the development of new technologies. When we analyze the removal procedure, we observe that uninstallers fail to remove files remaining from broken installation because no system component keeps track of processes' interactions all the time. Therefore, for the development of an efficient uninstallation procedure, systems would have either to (i) monitor the system constantly to identify which files/registry keys/so on were accessed by each process; or (ii) tag the touched files so one could identify to which applications a file/registry key belongs. If we develop a tagging mechanism, one could remove files associated with a process by looking at the registry keys tags or, otherwise, remove registry keys associated with a process by looking for the tags assigned to a file belonging to that process. Efficiently tagging resources consists of a significant open research problem that must be addressed by the research community.

Study Limitations. Whereas this study presented a comprehensive analysis of the most popular uninstallers, thus covering a significant user base, it is important to highlight that this study is noth exhaustive. The manual approach required to inspect the uninstallers limited the number of uninstallers that our research team was able to analyze. We understand that our current findings are a first step to shed light on the uninstallers landscape. For the future, more research is warranted to cover a greater number of uninstallers, of distinct nature, and covering distinct platforms.

7 Conclusion

In this work, we investigated the operation of the 11 most downloaded application uninstallers from the 3 most popular Internet software repositories. We analyzed their operation against well-behaved and malformed uninstallers to characterize their weak and strong aspects. Based on our experiments, we concluded the following about their operation: **(1)** Most installers are similar to the native Windows uninstaller, finding the same installed applications and not locating broken installations; **(2)** Some installers provide interesting additional features, such as creating a system checkpoint, but this feature might corrupt files if not

applied immediately after the broken application installation; **(3)** The heuristics employed by a few installers to clean broken installation files might be abused by a malicious uninstaller to force the removal of third-party's files; **(4)** The installers are not resistant against a malicious uninstaller designed to terminate the uninstaller application.

We recommend the OS vendors to: **(1)** Redesign their uninstallation systems. The third-part uninstallers all present better application organization than the native uninstaller (e.g., categorizing them), such that this might work as an incentive for users adopting this type of solution rather than the native one.

We recommend for users: **(1)** Do not confuse removing an application from the installed apps list with actually removing the application files. Sensitive files might still be resident in the filesystem after an application removal; **(2)** Application uninstallers are not secure robust enough to remove malware, thus they should not be used as a replacement for Antiviruses and security solutions.

Reproducibility. All code developed for our experiments are available at `https://github.com/marcusbotacin/Uninstallers`.

Acknowledgments. The authors would like to thank the Brazilian Ministry of Education for supporting this work (Research Project "Plataforma MEC de Recursos Educacionais Digitais", Funding Agency: Fundo Nacional de Desenvolvimento da Educação - FNDE, TED n. 10.959).

References

1. Botacin, M., Bertão, G., de Geus, P., Grégio, A., Kruegel, C., Vigna, G.: On the security of application installers and online software repositories. In: Maurice, C., Bilge, L., Stringhini, G., Neves, N. (eds.) DIMVA. Springer (2020)
2. Botacin, M., Domingues, F.D., Ceschin, F., Machnicki, R., Zanata Alves, M.A., de Geus, P.L., Grégio, A.: Antiviruses under the microscope: A hands-on perspective. Computers & Security (2022)
3. Cnet: Uninstall - search. `https://download.cnet.com/s/uninstall/?platform=linux` (2021)
4. Forum, A.: When does one use revo uninstaller? `https://forum.avast.com/index.php?topic=127051.0` (2013)
5. Forum, I.: Cbs/cnet recommended i use an uninstaller to remove their malware. `https://forums.iobit.com/topic/12814-cbscnet-recommended-i-use-an-uninstaller-to-remove-their-malware/` (2014)
6. Forum, V.: My opinion of revo uninstaller pro. `https://forum.videohelp.com/threads/351573-My-opinion-of-Revo-Uninstaller-Pro` (2003)
7. Google: Uninstallers - google play. `https://play.google.com/store/search?q=uninstaller` (2021)
8. Google: Unwanted software policy. `https://www.google.com/about/unwanted-software-policy.html` (2021)
9. Hoffman, C.: Should you use a third-party uninstaller? `https://www.howtogeek.com/172050/htg-explains-should-you-use-a-third-party-uninstaller/` (2016)
10. Kim, Y., Lee, S., Hong, D.: Suspects' data hiding at remaining registry values of uninstalled programs. In: e-Forensics. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008)

11. Liou, J.C., Duclervil, S.R.: A Survey on the Effectiveness of the Secure Software Development Life Cycle Models. Springer (2020)
12. Microsoft: Adding and removing an application and leaving no trace in the registry. https://docs.microsoft.com/en-us/windows/win32/msi/adding-and-removing-an-application-and-leaving-no-trace-in-the-registry (2018)
13. Microsoft: Application registration. https://docs.microsoft.com/en-us/windows/win32/shell/app-registration#registering-verbs-and-other-file-association-information (2018)
14. Microsoft: Configuring add/remove programs with windows installer. https://docs.microsoft.com/en-us/windows/win32/msi/configuring-add-remove-programs-with-windows-installer (2018)
15. Microsoft: Program is not listed in add/remove programs after installation. https://support.microsoft.com/en-us/topic/program-is-not-listed-in-add-remove-programs-after-installation-0866db2a-f8d9-fb0f-16d2-850f5072e536 (2018)
16. Microsoft: Windows sysinternals. https://docs.microsoft.com/en-us/sysinternals/ (2021)
17. Microsoft: Software download products & services, freeware & shareware. https://about.ads.microsoft.com/en-us/policies/restricted-categories/software-freeware-shareware#uninstall-functionality (2022)
18. Primiero, G., Boender, J.: Managing software uninstall with negative trust. In: Trust Management XI. Springer (2017)
19. Ramirez, A., Aiello, A., Lincke, S.J.: A survey and comparison of secure software development standards. In: 2020 13th CMI Conference on Cybersecurity and Privacy (CMI) (2020)
20. Raymond: The performance cost of reading a registry key. https://devblogs.microsoft.com/oldnewthing/20060222-11/?p=32193 (2006)
21. RegBench: Regbench, windows registry benchmark utility. https://bitsum.com/regbench.php (2017)
22. Softonic: Uninstallers - search. https://www.softonic.com.br/s/uninstallers (2021)
23. Softpedia: Uninstallers - search. https://www.softpedia.com/dyn-search.php?search_term=uninstallers (2021)
24. Soldani, J.: Grey literature: A safe bridge between academy and industry? SIGSOFT Softw. Eng. Notes **44**(3), 11–12 (Nov 2019). https://doi.org/10.1145/3356773.3356776, https://doi.org/10.1145/3356773.3356776
25. StopBadware: Zango. https://www.stopbadware.org/tags/zango?__cf_chl_jschl_tk__=pmd_a220ec1f116838d84c8791496582d0446d9606f7-1632851755-0-gqNtZGzNAc2jcnBszQjO (2009)
26. Uninstaller, R.: How to force uninstall a program that won't uninstall. https://www.revouninstaller.com/blog/how-to-force-uninstall-a-program-that-wont-uninstall/ (2021)
27. VirusShare: Virusshare. https://virusshare.com/ (2021)
28. Zax, R., Adelstein, F.: Faust: Forensic artifacts of uninstalled steganography tools. Digital Investigation **6**(1), 25–38 (2009). https://doi.org/https://doi.org/10.1016/j.diin.2009.02.002, https://www.sciencedirect.com/science/article/pii/S1742287609000267