SoK: Systematization, Detection, and Hunting of Windows Malware Persistence Techniques

Jorik van Nielen
University of Twente
Enschede, The Netherlands

Andrea Oliveri
EURECOM
Biot, France

Jerre Starink
University of Twente
Enschede, The Netherlands

Andreas Peter
University of Oldenburg
Oldenburg, Germany

Marieke Huisman
University of Twente
Enschede, The Netherlands

Simone Aonzo
EURECOM
Biot, France

Davide Balzarotti
EURECOM
Biot, France

Andrea Continella
University of Twente
Enschede, The Netherlands

Abstract

In order to maintain its presence on an infected system, malware employs a variety of *persistence* techniques. Although persistence is a well-known tactic of modern malware, our community lacks a comprehensive understanding of the types and prevalence of techniques adopted by Windows malware.

In this paper we present the largest documented set of persistence techniques (72 in total), highlighting their common characteristics and proposing a novel classification. Leveraging our systematization, we then design and implement a unified framework to characterize the adoption of each technique. Our framework (1) detects behavioral patterns that indicate the adoption of known techniques and (2) hunts for undocumented techniques by inspecting suspicious changes to the file system and Registry database.

By studying the adoption of persistence techniques across 48,873 detonated Windows malware samples, our study reveals several fundamental insights. For instance, we show that only 55.2% of samples are persistent—contrary to the general expectation that the vast majority of active malware requires persistence. While most samples rely on well-documented techniques, a small portion adopts more exotic approaches. Besides, while persistence detection is generally considered a solved problem, we reveal that industry-standard persistence detection tools produce a significant number of false positives and false negatives. Finally, our investigation results in the discovery of a new persistence technique and two previously undocumented evasion strategies that are actively employed by real-world malware.

CCS Concepts

- **Security and privacy** → **Malware and its mitigation**.

ASIA CCS '26, Bangalore, India

ACM ISBN 979-8-4007-2356-8/2026/06
https://doi.org/10.1145/3779208.3805971

Keywords

Malware, Persistence, Detection, Windows

ACM Reference Format:
Jorik van Nielen, Andrea Oliveri, Jerre Starink, Andreas Peter, Marieke Huisman, Simone Aonzo, Davide Balzarotti, and Andrea Continella. 2026. SoK: Systematization, Detection, and Hunting of Windows Malware Persistence Techniques. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 01–05, 2026, Bangalore, India.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3779208.3805971

1 Introduction

Malware is still a prevalent and growing threat, especially for Windows systems, with more than 80% of all discovered malware samples specifically targeting the Windows Operating System (OS) [6]. The AV-TEST Institute records over 250K new malware samples every day [22], highlighting the need for accurate and scalable malware analysis and detection systems. However, despite the huge effort of both academic and industry researchers, a 2023 survey still found that a stunning 84.7% of organizations with 500 or more employees were victim of a cyberattack [7]. When malicious samples get past defenses and infect machines, further analysis is required to determine the impact, improve existing defenses, and clean infected systems. This analysis requires up-to-date knowledge of the techniques and procedures adopted by malware. Such techniques can be categorized based on their objective, for instance, reconnaissance, privilege escalation, and persistence.

Persistence [4, 73, 89] allows malware to continue executing despite disruptive events such as reboots or logouts. In other words, persistence is used to gain a foothold on the infected machine. There are many persistence techniques in Windows that malware authors can rely on, and as the OS and third-party software evolve over time, new techniques are occasionally discovered [89]. For example, traditional, well-known techniques to achieve persistence include the use of `Run` Registry keys [9], adding a file to the `Startup` folder [9, 16], or creating a new Windows service [10, 13, 15]. Essentially, all techniques make changes to the disk content of the machine [90], either directly by adding or modifying existing files or indirectly by writing to internal databases (e.g., Registry) that are controlled by the kernel or system services. These changes persist over a reboot, resulting in the execution of code installed by malware.

The systematization of malware techniques is an active area of research, with recent works focusing on evasive techniques [69, 74, 82], code injection techniques [88], and low entropy packing schemes [77]. While various works preliminarily study malware persistence [64, 66, 70, 76, 80, 81, 83, 89, 91], they are either limited in *breadth*, i.e., only limited number of techniques and samples are studied, or in *depth*, i.e., no in-the-wild adoption is measured. Besides, no prior work attempted to unify prior efforts and hunt for unknown or undocumented techniques. Thus, our community lacks a comprehensive systematization of persistence techniques and an in-depth investigation of their adoption.

To fill this gap, we conduct a comprehensive systematization of Windows persistence techniques, collecting them from academic literature, security research blogs, online listings, and community-published detection rules. We then categorize techniques based on where they reside: specific paths in the file system, entries in the Windows Registry, or a combination of the two. We also analyze the traits of each technique based on its requirements and effects.

Building on the insights gained from our systematization and on a comparison of existing detection tools, we design and implement a two-stage framework to study the persistence phenomenon. In the first stage, we analyze behavioral execution patterns and create detection rules for each identified persistence technique in our systematization. We effectively identify known persistence techniques by observing executed API/system calls and by tracking calls to the Windows Management Interface (WMI) subsystem. In the second stage, we hunt for previously unknown persistence methods by examining suspicious artifacts in the infected target.

We use our framework to evaluate the adoption and prevalence of persistence techniques by malware in the wild. We replicate a balanced 2023 dataset of 67,000 samples across 670 malware families (100 per family) [67]. In addition, we collect a second dataset of 18,424 recent malware samples that were detected within three months prior to our experiments.

Our experiments identify the adoption of 42 distinct persistence techniques. Not surprisingly, the `Run` Registry key is the most used, adopted by 58.0% of the malware families in the balanced dataset. We also discover that while 61.3% of the persistent malware samples employs only one technique, the others employ two or more, with 3.0% even adopting a combination of five or more. Overall, 55.2% of the analyzed malware samples are persistent—contrary to the general expectation that the vast majority of active malware requires persistence.

Crucially, our analysis of state-of-the-art tools reveals that 72.5% of Capa [18] detections are false positives—alongside presenting a 42.1% false negative rate (FNR). Furthermore, Autoruns [78] incurs a 23.5% FNR, and Persistence Sniper [72] a 34.9% FNR.

Finally, our investigation discovers one previously undocumented persistence technique and two evasion strategies (based on WMI) used to hide calls to persistence techniques. In summary, the contributions of our work are as follows:

- **A comprehensive systematization of Windows persistence techniques.** We systematize 72 persistence techniques, compare their traits, and group them into classes. We also study existing persistence detection tools, comparing their performance on our 72 techniques.
- **A unified framework to study and characterize persistence techniques and related evasion strategies.** We design a framework to dynamically study the adoption of persistence techniques in Windows malware samples using detection rules. Our framework integrates a post-analysis phase that automatically surfaces suspicious persistence artifacts at scale by mining sandbox reports, allowing for a deeper understanding of persistence behaviors. Our results show that it reduces the amount of data analysts must review by about 99% to uncover previously undocumented Windows persistence techniques.
- **A measurement of persistence techniques adopted by real-world malware and insights for future research.** We measure the adoption of persistence techniques by dynamically executing 85,424 real-world malware samples, revealing important insights and discovering one previously undocumented technique and two detection bypass techniques.
- **Open source contributions.** We contribute to the research community by submitting new detection rules to the official Capa [18] repository–now merged into version 8.0–broadening its coverage and improving its accuracy, and by integrating our WMI plugin into the CAPE Sandbox [19] to enhance its analysis capabilities[1].

2 Persistence Systematization

We define a persistence technique as a modification of the operating system (OS) that results in the execution of the attacker's code even after all malicious processes have been terminated, without direct execution of the malicious code by the user and without any further action by the attacker. As such, a persistence technique allows the malware to start at boot, at login, or to be triggered by a system service, specific system or time conditions, or actions performed by the user when they interact with the OS (e.g., opening a different application, clicking on a system default menu).

Besides, for our systematization, we introduce three criteria for persistence techniques to ensure relevance as well as a clear scope: I) The technique targets the Windows operating system; II) It is reported to be functional on Windows 10 or 11 (the two versions currently supported by Microsoft); III) The technique does not require installed third-party software other than browsers and first-party Microsoft products (such as MS Office and Windows Terminal).

[1] Our artifacts are available at https://github.com/utwente-scs/Sok-Windows-Malware-Persistence.

It is important to note that a persistence technique may use distinct pathways to achieve functional equivalents. For example, both the `Run` and `RunOnce` Registry keys ensure execution upon user logon. In this work, we refer to these cases as different *variants* of the same technique.

Additionally, malware authors can employ strategies to make a persistence technique more stealthy. For instance, a malware sample can indirectly call an API through a proxy process, or leverage alternative subsystems rarely monitored by sandboxes. In this work, we define these optional evasion techniques as *detection bypass strategies.*

2.1 Known Persistence Techniques

We conduct a large-scale field research to build the most comprehensive list of persistence techniques used by malware authors. The search involves a comprehensive literature review [75], including academic works, online listings of persistence techniques, blog posts published by security researchers, and persistence detection rules employed by other software. A list of the 44 resources we used can be found in Table 6 in the Appendix. While these resources often discuss one or more persistence techniques, no comprehensive overview exists to date that covers all techniques known to the community.

For each technique we encountered in our research, we developed a proof-of-concept (PoC) implementation, and we tested it on live Windows 10 and Windows 11 systems to verify if it meets our criteria. These PoCs are also used in Section 4 to evaluate our detection system.

In total, we identified 72 distinct persistence techniques (Table 1). Besides, we created an interactive webpage [60] offering complete descriptions and additional details.

2.2 Traits

To better understand the similarities and differences among persistence techniques, we first define a list of traits that describe them. Our traits are derived from existing catalogs [14, 43] and further refined based on the results of our analysis of the different techniques. Traits are defined as **bool**ean (true of false) or **categorical** (a finite set of values), and are grouped into two classes: *requirements* and *effects.*

Requirements. The requirements are the conditions the process attempting to achieve persistence must satisfy. If one of the requirements is not met, the technique would fail.

- **File System Write [bool].** The technique writes to the file system. For example, it creates a file in a specific folder.
- **Registry Write [bool].** The technique writes to the Windows Registry. For example, a malicious command has to be added to a specific key.
- **Elevated Privileges [bool].** The technique requires elevated privileges—e.g., administrator rights are needed to write to certain Registry keys. While some variants work without these privileges, elevated access offers advantages, such as modifying the `Run` Registry keys for all users.
- **Additional Software [bool].** The technique requires additional software that is not pre-installed with Windows. For example, a technique may only work when Microsoft Office is installed. As previously mentioned, we only consider techniques that require Microsoft applications or browsers.
- **Interprocess Communication [bool].** The technique requires another process to execute a task. For example, Background Intelligent Transfer Service (BITS) jobs are stored in a database on the file system, but only the BITS service has write access to the job database. Thus, a sample would need to interact with BITS in order to use this technique.

Effects. Effects are specific traits associated with the resulting persistent code and its processes that provide an indicator of the "quality" of the persistence.

- **Elevated Privileges [bool].** The technique results in elevated privileges. For example, a technique executes a command at boot with System privileges.
- **Destructiveness [bool].** The technique damages an OS functionality in a way that can be noticed by the user. For instance, the screensaver may not start anymore, or the system might hang. However, some destructive techniques can be turned into non-destructive ones. An example is the DLL hijack technique in which a malicious DLL is loaded instead of the intended one, potentially crashing or reducing the functionality of other programs. However, the original functionality can be retained by using DLL proxying, while still achieving persistence [63].
- **Time of Execution [categorical].** The moment that the attacker code is triggered by the persistence technique. For example, a technique might run an executable when the user opens the File Explorer. We identified four classes for this trait: (1) *Execution at boot*, comprising the entire boot process, including all processes initiated after user login, that do not require any user interaction; (2) *Execution after a user action*, including techniques that are activated directly by a user action—for example, a DLL hijack triggers malicious code execution when a target program is executed; (3) *Execution at a system event*, covering events that are not the direct results of user actions, such as a program crash; (4) *Periodic execution*, ensuring that the persistent code is executed at a time interval.
- **Execution Type [categorical].** The code the technique executes can be grouped into five main categories: (1) The technique launches an executable file; (2) The technique launches an executable file controlling its arguments—this allows the execution of benign programs, such as PowerShell, which can be misused for malicious purposes while also eliminating the need for malicious code to be stored on disk; (3) The technique loads a dynamically linked library; (4) The technique directly runs a shellcode, for example, by injecting it into a separate benign process; (5) The technique provides code expressed in a scripting language run by a separate interpreter—e.g., JS in a browser.

Table 1: Persistence techniques covered in our study. Exec=moment of execution (B=boot, U=user action, E=system event, P=periodically). FS=file system write. Reg=Registry write. Priv=elevated privileges. IPC=interprocess communication. SW=additional software. Destr=destructiveness. Type=payload type (E=exe, EwA=Exe with arguments, D=DLL, S=script for third party program, SC=shell code). A=AutoRuns [78], PS=PersistenceSniper [72], C=Capa [18] (before the introduction of our rule set). ◐=rule present, but misses variants.

				Requirements					Effects			Present in		
Class		Exec	Technique	FS	Reg	Priv	IPC	Sw	Priv	Destr	Type	A	PS	C
Registry		B	Active Setup	○	●	●	○	○	○	●	EwA	●	○	●
			BootVerificationProgram	○	●	●	○	○	●	○	E	●	●	○
			Group policy	○	●	●	○	○	●	○	EwA	◐	○	○
			Run	○	●	○	○	○	●	○	D/EwA	●	●	●
			Startup folder	○	●	○	○	○	○	○	E	○	○	●
			UserInitMprLogonScript	○	●	○	○	○	○	○	EwA	●	●	○
			Windows Load	○	●	○	○	○	○	○	E	●	●	●
		E	AeDebug	○	●	●	○	○	○	○	EwA	○	●	○
			Alternateshell	○	●	●	○	○	○	○	E	●	○	○
			Screensaver	○	●	○	○	○	○	○	E	●	●	●
			Windows Error Reporting	○	●	●	○	○	○	○	E	○	●	○
			dotNET DbgManagedDebugger	○	●	●	○	○	○	○	EwA	○	●	○
		P	Task scheduler	○	●	●	○	○	●	○	EwA	●	●	◐
			TelemetryController	○	●	●	○	○	●	○	EwA	○	●	○
			Windows service	○	●	●	○	○	●	○	D/EwA	●	●	●
		U	AMSI	○	●	●	○	○	○	○	D/E	○	●	○
			APPX	○	●	○	○	○	○	●	E	○	○	○
			App paths	○	●	○	○	○	○	○	E	○	●	○
			AutoplayHandlers	○	●	●	○	○	○	○	D/EwA	○	○	○
			COM hijack	○	●	○	○	○	○	○	D/E	●	○	○
			Command processor	○	●	○	○	○	○	○	EwA	●	●	○
			Default file association	○	●	●	○	○	○	○	EwA	◐	○	○
			Explorer tools	○	●	●	○	○	○	○	EwA	○	●	○
			Image File Execution Options	○	●	●	○	○	○	○	E	●	●	○
			Installation Repair	○	●	○	○	○	●	○	EwA	○	○	○
			Internet explorer	○	●	○	○	●	○	○	D	●	○	○
			PATH	○	●	●	○	○	○	○	E	○	○	○
			Protocol handler	○	●	●	○	○	○	○	D/E	●	○	○
			RDP startup program	○	●	●	○	○	○	○	E	●	●	○
			SilentProcessExit	○	●	●	○	○	○	○	EwA	○	●	○
			TS InitialProgram	○	●	○	○	○	○	○	E	◐	●	○
			Uninstall	○	●	○	○	○	●	○	EwA	○	○	○
			Universal App URI	○	●	○	○	○	○	○	EwA	○	○	○
			shellserviceobjectdelayload	○	●	●	○	○	●	○	D/E	●	○	○
Registry and Filesystem	OS Files	B	Keyboard layout	●	●	●	○	○	●	○	D	○	○	○
			LSA	●	●	●	○	○	●	○	D	◐	●	○
			Natural Language	●	●	●	○	○	●	○	D	○	●	○
			Network Provider	●	●	●	○	○	●	○	D	●	○	○
			Print monitors	●	●	●	○	○	●	○	D	●	○	○
			Print processors	●	●	●	○	○	●	○	D	○	○	○
			TimeProviders	●	●	●	○	○	○	○	D	○	○	○
		E	Code signing	●	●	●	○	○	○	○	D	○	○	○
			Netsh	●	●	●	○	○	○	○	D	○	●	○
			Winlogon	●	●	●	○	○	●	○	D	●	●	◐
		U	AppCertDLLs	●	●	●	○	○	○	○	D	●	●	○
			AppInit__DLLs	●	●	●	○	○	○	○	D	●	●	●
			AutodialDLL	●	●	●	○	○	○	○	D	●	●	○
			COM typelib	●	●	○	○	○	○	○	EwA	○	○	○
			COR__PROFILER__PATH	●	●	○	○	○	○	○	D	○	○	○
			DOTNET__STARTUP__HOOKS	●	●	○	○	○	○	○	D	○	●	○
			Disk Cleanup Handler	●	●	●	○	○	○	○	D	○	○	○
			Filter handlers	●	●	●	○	○	●	○	D	○	○	○
			HtmlHelp Author	●	●	○	○	○	○	○	D	○	●	○
			Office add-ins	●	●	○	○	●	○	○	D/EwA	●	●	○
			Shellex	●	●	○	○	○	○	○	D	●	◐	○
			hhctrl COM hijack	●	●	●	○	○	○	○	D	○	●	○
Filesystem		B	Startup folder	●	○	○	○	○	○	○	E	●	●	●
	OS Files	E	ErrorHandler script	●	○	●	○	○	○	○	EwA	○	●	○
			Host software binary compromise	●	○	○	○	○	○	○	E	○	○	○
		U	Accessibility Tools Backdoor	●	○	●	○	○	●	○	E	○	●	○
			Application shimming	●	○	●	○	○	○	○	SC	○	○	○
			DLL hijack	●	○	○	○	○	○	○	D	○	○	○
			Get-Variable hijack	●	○	○	○	○	○	○	E	○	○	○
			PowerShell profile	●	○	○	○	○	○	○	EwA	○	●	○
			lnk shortcut	●	○	○	○	○	○	○	EwA	○	○	○
			url shortcut	●	○	○	○	○	○	○	EwA	○	○	○
	Apps	B	Windows Terminal Profile	●	○	○	○	●	○	●	EwA	○	●	○
		U	Browser extension	●	○	○	○	●	○	○	S	○	○	○
			Office AI hijack	●	○	●	○	●	○	○	E	○	●	○
			iphlpapi DLL hijack	●	○	○	○	●	○	○	D	○	○	○
	DB	P	BITS job	●	○	○	●	○	○	○	EwA	○	●	○
			WMI event subscription	●	○	●	○	○	●	○	EwA	●	●	○
Total				38	56	41	1	6	19	3		30	38	10

2.3 Classes

We classify persistence techniques based on where they reside within the OS. We define the following main classes:

- **Registry.** Techniques in this class modify the Windows Registry, a configuration database that the OS and many applications use to store information [68]. These modifications include creating new keys, altering existing values, or deleting entries through specific API calls provided by the OS. For instance, by adding a key to `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run` and setting its value to the path of the desired executable, an attacker can ensure its execution at boot.
- **File System.** Techniques in this class write files directly to the disk. We split this class into three subclasses:
  - **Operating System Files.** Techniques in this class place files in folders that are used by the OS, such as `System32`. A technique of this class is DLL hijacking: placing a malicious DLL at a specific file-system location results in loading that DLL instead of the original when a target application starts.
  - **Additional Application Files.** Techniques in this class place files in folders used by additional software packages. The exact location can differ based on the installation of the software, making these persistence techniques harder to detect. The Windows Terminal Profile persistence technique represents an example of a technique in this class. By changing a JSON file in the installation folder of the Windows Terminal, an executable can be set to be run every time the terminal is started.
  - **Databases.** Techniques in this class modify local system databases, including the WMI database, which the Windows operating system uses to store hardware, system information, and special entries like WMI subscriptions that describe actions that should be performed when certain system events occur [5].
- **Registry and File System.** Techniques in this class require the modification of both the Registry and the file system. An example is the registration of a new keyboard layout, which requires changes to the `Control\KeyboardLayouts` registry key and to a DLL located in the `System32` folder.

We group the techniques into classes in Table 1.

2.4 Related Work

The research community made a significant effort to study and analyze the adoption of malware behaviors and techniques. Willems et al. [91] measure the persistence of 6,148 samples of autonomous spreading malware collected in 2006. They report 95% of the samples to be persistent. Bayer et al. [64] analyze 330,088 potentially malicious samples collected between 2007 and 2008. They detect, among others, persistence and report the top 10 most used techniques. Botacin et al. [66] propose a malware analysis environment that reboots the sandbox after samples write to a persistence Registry key. The authors find that 13.6 % of 2,937 samples are persistent, and compare the malware behavior pre- and post-reboot. Oosthoek et al. [80] measure the adoption of six of the most used persistence techniques in 951 carefully selected samples that were first observed between 2007 and 2018. Gittins et al. [70] present a case study of five advanced malware samples and how they utilize persistence techniques. Rana et al. [83] discuss six widely used persistence techniques, how they work, and how various tools can be used to detect them. Table 2 summarizes a comparison of related works featuring a measurement of persistent behavior.

Table 2: Comparison of persistence measurement studies, including the number of techniques measured (NR = Not Reported), malware sample count, discovery years, and whether state-of-the-art tooling is compared.

	Techniques	Samples	Year	SOTA Comp.
Willems et al. [91]	NR	6,148	2006	✗
Beyer et al. [64]	10	330,088	2007-2008	✗
Botacin et al. [66]	NR	2,937	2015	✗
Oosthoek et al. [80]	6	951	2007-2018	✗
Our work	72	48,873	2021-2024	✓

Apart from measurements, researchers have focused on classification, detection, and prevention. Mankin [76] presents a persistence detection approach through disk instrumentation. Villalón-Huerta et al. [89] proposed an OS-independent taxonomy of known persistence techniques, resulting in valuable insights into the nature of persistence techniques. Yet, the authors did not address the specifics of Windows persistence techniques, leaving challenges to understanding the nature of different techniques for effective detection. Phillips et al. [81] propose a solution to preventing malware reaching persistence by blocking writes to persistent locations. A recent work by Liu et al. [73] proposes an approach to detecting persistence deployed specifically by advanced persistence threats. To reduce false positives for security operation centers, they build provenance graphs and raise alarms only if a used persistence technique calls back to an attacker-controlled server.

Besides academic works, other practical resources and tools feature persistence techniques. The ATT&CK knowledge base provides a categorization of real-world techniques [43]. While this is of great help for red and blue team mappings, it is not intended as a complete overview of all persistence techniques. In fact, while some techniques are described in detail, others are grouped within generic classes (e.g., *Modify Registry*). The Hexacorn blog provides a collection of 141 techniques for Windows, with in-depth explanations [12] covering various Windows versions (from Windows 98, up to Windows 11) and 3rd party software. Still, they discuss only 40 of the techniques listed in our study. The Persistence-info project [14] features 44 persistence techniques classified according to various criteria, such as permission required, security context, code type. AutoRuns [78] and PersistenceSniper [72] analyze

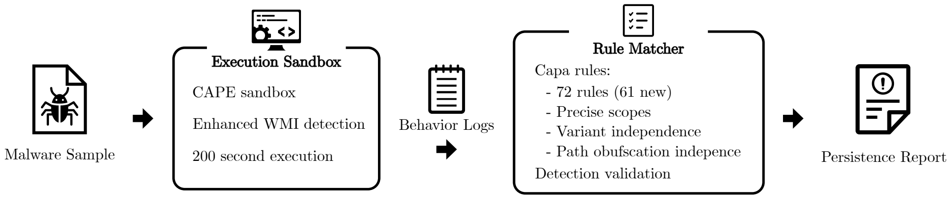

Figure 1: Persistence detection pipeline design. Malware samples are executed in a sandbox, tracing Windows and WMI API calls. Known persistence techniques are identified via Capa rules. The pipeline outputs a report detailing detected mechanisms.

the state of a given system to detect persistent software already present on the system. More relevant resources can be found in Table 6 in the Appendix.

Despite this large body of literature, our community still lacks a comprehensive understanding of the adoption of persistent behavior in modern Windows malware. No study combines *breadth*, i.e., a comprehensive list of persistence techniques, with *depth*, i.e., an in-the-wild, large-scale measurement. Besides, many prior studies are old, while our work reveals new, surprising insights (e.g., only 55.2% of modern malware is persistent), which confute long-held beliefs. Finally, Table 1 shows, in the three rightmost columns, the persistence techniques supported by state-of-the-art tools: Autoruns [78], PersistenceSniper [72], and Capa [18] (before being updated with our new rules). Of the 72 persistence techniques we catalogued, the most capable tool (i.e., PersistenceSniper) detects only 38, and even these detections are often incomplete because specific variants remain unrecognized. This further reinforces that persistence analysis is still an underserved facet of current malware analysis understanding and tooling.

3 Framework Design

Following our systematization, we design a persistence detection pipeline to characterize and measure the adoption of persistence techniques in real-world malware (Section 3.1). Then, we design an analysis system to hunt for undetected techniques (Section 3.2).

3.1 Persistence Detection

Our pipeline (Figure 1) is composed of two main components:

Execution Sandbox. To monitor Windows low-level API invoked by malware samples, our system leverages CAPE [19] as a dynamic execution sandbox. For our experiments we modified CAPE to further minimize the impact of evasion techniques deployed by the malware (details in Section 4.1) and to enhance its monitoring capability to cover subsystems which the sandbox was unable to intercept (e.g., WMI).

Rule Matcher. To detect existing techniques we wrote Capa rules for each technique in our systematization. Capa [18] is an open-source framework designed to identify capabilities in executable files by performing static analysis or by analyzing dynamic analysis reports from sandboxes, including CAPE.

Capa rules are written in YAML, and allow users to describe a wide range of features, such as the invocation of specific Windows APIs or the use of particular strings. Today they are the standard for the detection of specific behaviors in dynamic analysis traces and are also used by VirusTotal [56]. An example of a Capa detection rule is shown in Figure 2.

```
 1  rule:
 2   meta:
 3    name: Persist via Image File Execution Options Registry
            Key
 4     scopes:
 5       dynamic: call
 6   features:
 7     - and:
 8       - match: set Registry value
 9       - string: /Microsoft\\Windows NT\\CurrentVersion\\
              Image File Execution Options\\/i
10       - string: /Debugger/i
```

Figure 2: Example of a Capa rule from our rule set.

We improve upon the existing Capa persistence detection rule set in four ways:

(1) **New Rules.** We developed 72 rules, 61 of which did not have any equivalent in the Capa repository.
(2) **Precise Scopes.** We refined the scopes of existing rules to reduce false positives. In fact, existing Capa rules operated at the thread scope: for example, detecting the usage, the read or write of a Registry path and any Registry write operation, as long as both occur in the same thread. However, this could incorrectly flag cases where a read from that key and a write to a different one occur. To mitigate these false positives, we use a more precise scope: as shown in Figure 2 we set the dynamic scope to `call` (Line 5), which ensures that the Registry path (Line 9) and key (Line 10) are all referenced in the same call that sets the value of the `Microsoft\WindowsNT\CurrentVersion\ImageFileExecutionOptions` Registry key (Line 8).
(3) **Variants Independence.** The Windows Registry includes several key aliases, which are essentially links pointing to other existing keys [68]. Thus, persistence techniques can achieve the same effect by writing to several alias keys. Furthermore, some Registry keys exist in different roots of the Registry, permitting to configure software or a component systemwide by using the `HKLM` root, or on a per-user basis by using the `HKCU` root. A classic example is the already mentioned `\Software\Microsoft\RunOnce`, which exists in the `HKLM`, `HKCU`, and `HKU` roots.
Our rule set explicitly accounts for path variations, used by malware authors to fool existing tools. For example, the rule in Figure 2 does not specify the start of the Registry key (`HKLM\Software`) (Line 9) because malware can target the alias path `HKLM\Software\WOW6432Node`.

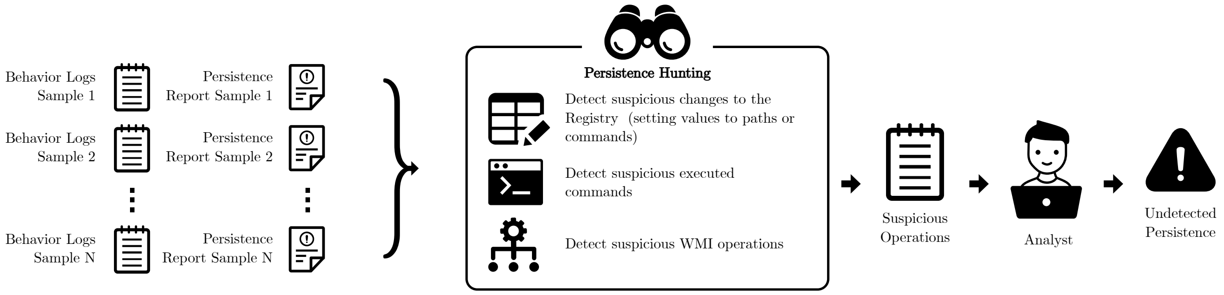
Behavior Logs Sample 1
Persistence Report Sample 1
Behavior Logs Sample 2
Persistence Report Sample 2
Behavior Logs Sample N
Persistence Report Sample N
Persistence Hunting
Detect suspicious changes to the Registry (setting values to paths or commands)
Detect suspicious executed commands
Detect suspicious WMI operations
Suspicious Operations
Analyst
Undetected Persistence

Figure 3: Pipeline of the persistence hunting phase: Starting from Windows API calls and persistence reports from the Rule Matcher, suspicious operations are filtered (hunting) and reported to the analyst to identify undetected persistence techniques.

(4) **Path Obfuscation Independence.** Malware can use various ways to obfuscate paths. For instance, by combining capital and non-capital letters or by adding a white space character to the end of a path. We deal with these issues by leveraging carefully crafted case-independent regular expression statements.

Some rules require additional context, which cannot be encoded with the Capa rule grammar. In order to accurately detect these techniques, our system implements additional **Detection Validation** checks:

- **Host Software Binary Compromise.** This technique involves malware samples overwriting existing executables on disk. Detecting whether a file is newly created or overwritten based solely on API calls is challenging. To address this, we generate a list of existing files in the sandbox's clean state and check for overwrites during execution.
- **DLL Hijack.** DLL hijacks result in a benign process loading a malicious DLL. Various variants of DLL hijacks require the overwriting of an existing DLL or the placement of a DLL with a specific name at a specific location [65]. To detect the overwriting of an existing DLL, we use the same approach as for Host Software Binary Compromise. To detect the introduction of new, hijackable DLLs, we cross-reference file names against `Hijacklibs` [20], an open-source collection of DLL names susceptible to DLL hijacks.
- **WMI Events.** Some techniques, like WMI Event Subscription, require the malware to interact with the WMI subsystem, either through PowerShell commands or direct access. However, since the CAPE sandbox cannot natively monitor WMI activity, we develop a custom plugin that leverages the in-kernel Event Tracing for Windows (ETW) system to intercept all WMI calls, allowing us to detect stealthy malware activity performed through WMI.

Open-Sourcing. All Capa rules developed during our research that did not require further validation have been reviewed and integrated into the official Capa repository as part of version 8.0, released by Mandiant. These rules primarily match atomic Windows API calls, focusing on registry and file system write operations. Additionally, our implementation of the WMI tracer has been merged into the CAPE sandbox.

3.2 Persistence Hunting

We rely on our Rule Matcher component to identify the use of known techniques. In addition, to validate and broaden our knowledge of persistent behavior, we hunt for undocumented persistence techniques by analyzing the Windows API calls performed by malicious samples as summarized in Figure 3.

Our hunting approach is intended to assist researchers in identifying undocumented persistence mechanisms at a large scale, *across entire malware datasets*—particularly those containing fresh samples that may introduce previously undocumented techniques. We leverage the collective volume of Registry modifications and file writes across all samples to identify anomalies and highlight potential undocumented techniques, enabling analysts to focus their manual checks on a smaller, high-value subset.

To do this, for each sample in the dataset, *we aggregate all* events where malware either writes executable code to files or inserts/overwrites paths of executables or DLLs in Registry keys not previously associated with persistence. We find such events by analyzing regular file and Registry API/system calls, parsing shell commands, and analyzing the WMI operation log (through our new CAPE plugin) for invocations of WMI native APIs and external tools like PowerShell or `wmic.exe`.

To minimize redundancy and highlight meaningful patterns, we organize the collected data hierarchically. More specifically, Registry keys are tokenized to replace variable elements with placeholders, reducing duplication across similar entries. Thus, the keys `HKCU\Software\Classes\clsid\{AB8902B4-09CA-4BB6-B78D-A8F59079A8D5}\InProcServer32` and `HKCU\Software\Classes\clsid\{083863F1-70DE-11D0-BD40-00A0C911CE86}\InProcServer32`, written by different malware samples, are generalized as `HKCU\Software\Classes\clsid\***UUID_PLACEHOLDER1***\InProcServer32`. This approach abstracts common patterns in Registry modifications, allowing researchers to identify persistence techniques that may be shared across multiple malware families. Each placeholder is mapped to the original tokens it replaced, along with the number of samples exhibiting the pattern. By clustering similar records, we reduce the analyst's workload, enabling prioritization of promising candidates, such as Registry paths used across multiple samples or rare paths present in only a single sample, which may indicate novel techniques.

In addition to data aggregation and pattern analysis, we automate the generation of verification scripts. These scripts, executed in a clean sandbox environment, deploy decoy executables or DLLs in identified file system locations likely used for persistence. When executed, the decoys record their own file paths in a log file. This allows us to identify which decoy is executed when an event is triggered at the system reboot or manually by the analyst, confirming the viability of the identified technique associated with the decoy path.

4 Empirical Study

In this section, we outline our experimental setup and the samples we used in our experiments. We also discuss the validation process for our detection rules, the novel techniques uncovered during our analysis, and the adoption and prevalence of various persistence techniques across our datasets.

4.1 Experimental Setup

For our experiments, we use a physical server equipped with a 64-core Intel Xeon Platinum processor at 2.40 GHz, 384 GB of RAM, and SSD storage. Our sandbox is built using CAPE [19], which provides numerous features to enhance malware analysis, such as customizing QEMU strings to conceal the virtual machine's presence and automatically instrumenting malicious code that detects debugging or analysis DLLs. We deploy 30 parallel Windows 10 22H2 VMs, each configured with 4 GB of RAM, two CPUs cores, 128 GB of disk space, and Internet access to enable large-scale malware analysis. To simulate a realistic environment, the file system on each VM is populated with common documents, spreadsheets, and other typical files that malware might recognize as valuable targets [79]. Following community guidelines [84, 85], we block potentially harmful traffic such as spam and isolate the sandbox within a separate subnet disconnected from production systems. Finally, based on findings by Kuechler et al. [71] showing that malware code execution plateaus after about 120 seconds, we conservatively run each sample for 200 seconds to capture comprehensive behavior.

4.2 Datasets

For our experiments we assembled two different datasets.

Dataset I is a replica of the family dataset used in 2023 by Dambra et al. [67]. It contains 67,000 Portable Executable files collected from the VirusTotal (VT) feed between August 2021 and March 2022. The samples are evenly distributed across 670 malware families, with 100 samples per family, with labels assigned by AVClass [86, 87]. We retrieved the list of hashes and family labels from their GitHub repository [11] and downloaded the samples from VT. This dataset allows us to analyze correlations in the usage of persistence techniques within malware families. Moreover, its balanced composition enables us to measure general trends in persistence adoption.

Dataset II consists of 18,424 recent malware samples collected from MalwareBazaar [37]. Following Zhu et al. [92], who show that false positives decrease significantly after *five* detections on VirusTotal (VT), we cross-reference each sample with its VT report to ensure detections exceed this threshold. According to VT, these samples were all first observed between August and November 2024. We use this dataset to study fresh malware, which is more likely to employ previously unknown persistence techniques.

Filtering. Since the purpose of our study is to conduct a measurement of the prevalence of persistence techniques, we need to be careful to exclude samples that failed to detonate in our sandbox. For this reason, we discarded samples that did not perform any Windows API calls as well as those that did not perform any potentially malicious actions according to CAPE—assuming that if a malicious action was performed, then the sample at least partially executed its payload. By using this heuristic we conservatively removed 26,478 samples from Dataset I (39.5%) and 10,073 (54.7%) from Dataset II. In total, our final datasets included 48,873 running samples.

We hypothesize two main reasons for inactive samples. First, their command and control (C2) infrastructure may be offline. Second, the samples might adopt evasion techniques that circumvent CAPE's countermeasures. Prior studies have shown that 40-80% of modern malware employs at least one evasion method [69, 74], and the proportion of filtered samples in our analysis is consistent with these findings.

Finally, given that the Dataset II is imbalanced with respect to family representation, a post-filtering verification was conducted to ascertain the presence of any families that were disproportionately represented in the dataset, as this could potentially introduce bias into the results. Fortunately, this is not the case. Using AVClass, we measured that the filtered Dataset II contains 523 families. The top three prevalent families are `Agenttesla` (5.9%), `Berbew` (5.0%), and `Cosmu` (4.3%) and for the 3.5% of the samples, AVClass was not able to determine a family label (the distribution of the 50 most prevalent is reported in Table 7 in the Appendix).

4.3 Framework Validation

False Negatives. We create proof-of-concept implementations for each of the 72 techniques in our systematization and use them to validate our detection pipeline. Whenever possible, these implementations rely solely on Windows API calls, capturing the typical malware behavior for each technique and ensuring portability across Windows versions.

Two techniques require additional DLLs due to their need to modify databases with custom file formats: the WMI subscription technique [5], which binds executable execution to Windows events such as user login, and the BITS jobs technique [8], which triggers execution upon the success or failure of background file transfers.

We manually verify that our framework successfully detects *all* implemented techniques, demonstrating its effectiveness in tracking persistence technique adoption.

False Positives. We run our persistence detection pipeline over the two datasets and randomly select 10 samples per detected persistence technique. If fewer than 10 samples are detected for a given technique, we include all of them. Out of the 334 samples we manually analyzed, we found 2 false positives

Table 3: Discovered persistence technique and bypasses

Name	Class	Dataset	#	First Seen
Installation Repair	Technique	I	8	2010-07-03
WMI Registry Modif.	Det. Bypass	I + II	118	2008-01-29
WMI Proc. Schedule	Det. Bypass	II	15	2024-08-02

(0.6%), caused by one of the monitored paths written *inside* a file. This is due to the fact that the Capa engine cannot distinguish between content written to a file and file locations, not permitting to fix this type of false positive by improving the associated Capa rule.

4.4 Persistence Hunting

We use our persistence hunting approach (Section 3.2) to analyze the CAPE reports of both datasets. We first extracted the set of Registry writes, executed commands, and WMI operations (157,166 for Dataset I and 53,992 for Dataset II). After data aggregation, the number of suspicious operations from Dataset I is reduced by 98.5% to 2,397 (0.04 operations per sample in average) while for Dataset II is reduced by 99.4% to 310 (0.01 operations per sample).

Through manual examination we identified one previously unknown Registry-based technique and two undocumented WMI detection bypass strategies (summarized in Table 3).

New Technique: Installation Repair. When new software is installed, Windows creates a Registry key at `(HKLM|HKCU)\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\<SOFTWARENAME>`, which stores essential information about the installation. This key typically includes metadata such as the software's name, version, and developer, displayed in the Control Panel's list of installed applications.

In some cases, the installation process adds an optional `modifypath` value to the Registry. This value enables the execution of a support script that allows users to modify or repair the software through the "Modify" option in the Software Installation Control Panel menu.

Our experiments show how malware in the wild exploits the `modifypath` value for persistence: by injecting a path to a malicious executable into this field, the sample triggers its payload when the user clicks the "Modify" action, effectively turning a legitimate and useful system feature into a vector for persistence.

Undocumented Detection Bypasses. Thanks to our CAPE extension to monitor WMI calls, our framework flagged a number of suspicious behaviors in which malware leveraged this technology as a mechanism for bypassing detection. We can group these evasive techniques into two cases:

I) Registry Modifications Through WMI. Our analysis detected samples that relied on the `SetStringValue` method of the WMI `StdRegProv` provider to create a Registry key (`HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`) associated to persistence. It is interesting to note that this strategy is employed by only one sample in Dataset 2, while all other remaining samples are from Dataset 1. This observation suggests that the use of this strategy has been gradually abandoned over time, although still functional.

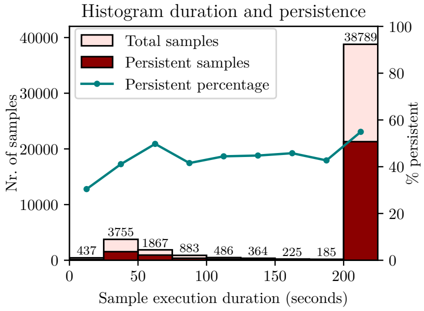

Figure 4: Execution duration vs. persistence count: Samples running under 200 seconds ended voluntarily, while those still active at 200 seconds were forcibly terminated by the sandbox.

II) Process Scheduling Through WMI. The Windows Management Instrumentation enables the automation of system management tasks, including the scheduling of process execution through the `PS_ScheduledTask` and `MSFT_ScheduledTask` WMI providers. Our experiments revealed that malware exploited five different methods from these providers to schedule tasks under various conditions, allowing them to gain persistence in ways that are equivalent, but more difficult to detect, than by directly invoking `tasksched.exe`. These methods include scheduling tasks for a specific user (`RegisterByUser`), triggering execution once (`NewTriggerByOnce`), or activating tasks based on user logon (`NewTriggerByLogon`) or system startup (`NewTriggerByStartup`), and using the `MSFT_ScheduledTask` class to schedule execution at a fixed date and time. Notably, we observed the use of these detection bypass strategy *exclusively* in recent samples from Dataset II. This suggests that malware authors have historically favored more traditional methods, such as using directly the WMI shell `wmic.exe` and only recently they have started to adopt this more stealthy approach.

4.5 Persistence Prevalence

We study the prevalence of the various persistence techniques across our two datasets of malware samples.

General Adoption. Overall, we measure a persistence rate of 55.2% for Dataset I and 41.3% for Dataset II.

Figure 4 shows the distribution of execution time of each sample in our full dataset, highlighting both the total number of samples and the persistent-only ones.

Observing the graph, it is clear that most samples (82.6%) are running until the sandbox time limit of 200 seconds is reached. For this long-execution group, we measure a persistence rate of 54.9%. For the remaining samples (17.4%) that terminate voluntarily before 200 seconds, we measure 43.1% to be persistent. This difference might be due to the fact that samples in the short-execution group often include first-stage components (e.g., droppers) and samples that only partially manifest their activity at the time of the analysis.

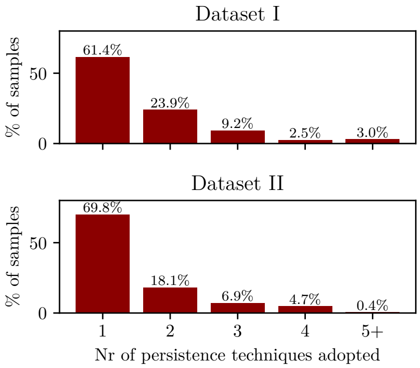

Figure 5: The distinct number of persistence techniques used per persistent malware sample.

Figure 5 shows the number of persistence techniques used per malware sample in Dataset I and Dataset II. For Dataset I, 61.3% of persistent samples adopt exactly one persistence technique, while for Dataset II this percentage is 69.8%. This means that tracking a single persistence location is sufficient to remove the infection only for more than half of the persistent malware. At the other end of the spectrum, a non-negligible 5.5% (Dataset I) and 5.1% (Dataset II) of malicious samples adopt four or more persistence techniques.

Comparison with state-of-the-art tools. Table 4 compares our detection results with Capa detection (prior to our contribution), Sysinternals Autoruns [78], and Persistence Sniper [72]. We take the results of our detection framework as the ground truth to calculate the metrics. Based on our validation described in Section 4.3, the very low false positive percentage of our detection framework has a minimal impact on the validity of our findings. To verify the false positives that we observe in the old Capa rules, we randomly select 50 samples that were detected by the old Capa rules but not by our detection system. We manually checked the API calls, confirming that all 50 cases are indeed false positives.

Autoruns and Persistence Sniper rules could not be applied directly to our trace of API calls, since they rely on system snapshots. Instead, we created Capa rules that correspond to their detection patterns. All three persistence detection tools successfully identify less than 80% of persistence techniques. Additionally, Capa incorrectly identifies 72,183 cases of persistence (72.5% of all its detections) due to imprecise scopes. False negatives mostly occurred due to incomplete collections of persistence techniques.

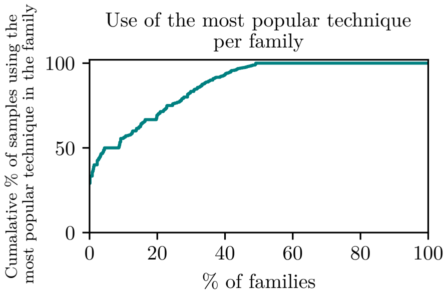

Figure 6: Cumulative distribution of the most popular persistence technique per family in Dataset I: In 50.9% of families, all samples use the same most popular technique.

Persistence Technique Popularity. The popularity of the various persistence techniques is summarized in Figure 7. The figure also shows whether the techniques are present in popular persistence-detection tools: Autoruns, PersistenceSniper, and Capa (before the introduction of our rule set). As expected, the simple `Run` Registry key is the solution most frequently adopted by malware authors. Other popular techniques are also widely used by benign software, e.g., Windows Services, lnk shortcuts, startup folder, and the `Winlogon` Registry key. Interestingly, 42 distinct persistence techniques were used in Dataset I and 25 in Dataset II. Moreover, some techniques actively adopted by malware are not detected by popular persistence detection tools, e.g., the `uninstall` Registry key.

Persistence Techniques Within Families. Figure 6 depicts the adoption rate of the most popular technique within our full dataset. Specifically, 50.9% of malware families consistently use the same persistence technique in each of their persistent samples. Some families, however, show more variability. For 19.7% of the families, only a maximum of 66.7% of the samples adopt the most commonly used persistence technique in that family, while other samples adopt different techniques. Thus, while samples within the same family tend to use the same technique, there are many samples where this is not the case.

Table 5 shows the class adoption of 10 malware families in Dataset II. Besides emphasizing the variety of techniques used by samples within the same family, it also shows that not all samples in the same family must necessarily be persistent.

Furthermore, we compare the persistence techniques used in malware families that are both present in Dataset I and Dataset II. Specifically, we only selected samples from Dataset I that were discovered in 2021, to reduce the variance in the gap in time. We find that some families used the same persistence

Table 4: Performance metrics of state-of-the-art tools, calculated with respect to persistence techniques detected by our pipeline. Obfuscation, WMI, and Improper scope do not apply to snapshot-based approaches (Autoruns and Persistence Sniper).

	True Positives	False Negatives				False Positives
Tool	**Match**	**Missing rule**	**Missing variant**	**Obfuscation**	**WMI**	**Improper scope**
Capa (without our improved rules)	57.9% (24,331)	41.2% (17,314)	0.0% (1)	0.8% (350)	0.0% (15)	72.5% (72,184)
Sysinternals Autoruns	75.1% (31,552)	23.5% (9,886)	1.4% (573)	-	-	-
Persistence Sniper	65.1% (27,343)	34.8% (14,618)	0.1% (50)	-	-	-

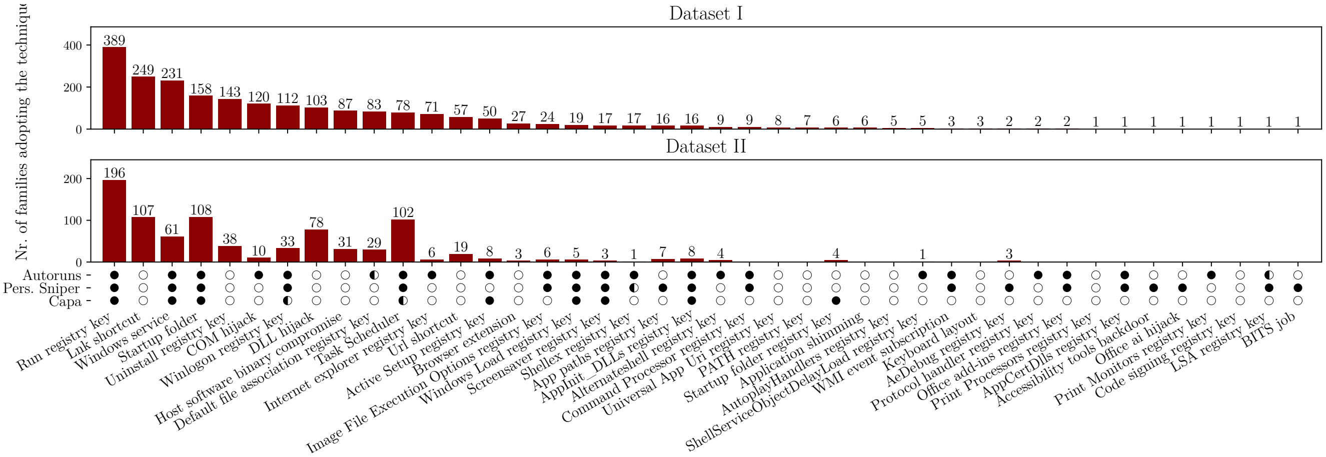

Figure 7: Persistence technique popularity in Dataset I & Dataset II. ●=present, ◐=present, but misses variants, ○=not present.

techniques in both 2021 and 2024 (e.g., `Berbew` and `Shipup`) while others shifted to different persistence techniques. For example, the `Strab` malware family used Windows services for persistence in 38 samples in 2021 but predominantly shifted to the startup folder and the `Run` Registry key by 2024. Table 9 in the Appendix contains more details.

Persistence Classes and Moment of Execution. Table 8 in the Appendix presents the detailed distribution of detected persistence classes. Registry-based techniques emerge as the most prevalent, accounting for 68.6% of all detections in Dataset I, followed by OS file-based techniques (24.0%). Within the Registry class, 53.9% of techniques are triggered at system boot—highlighting it as a reliable execution point for maintaining persistence. Interestingly, although 31% (22) of the studied techniques require both the Registry and file system, only 6.5% of samples actually adopt such combined methods.

Persistence in Malware Classes. In Figure 8, we show the persistence rates of various malware classes (determined using AVClass). Grayware has a relatively low persistence rate of 37.9%. This is expected, as users might launch the malware voluntarily, as they do not know it is malicious. Worms, backdoors, viruses, ransomware, and spyware all have higher persistence rates above 60%. This is not surprising, as malware in these classes generally cannot rely on the user launching them. Specifically, for the malware that does not fall into the classes of grayware, downloader, and no class, we find that 68.1% of the samples in Dataset I are persistent.

Figure 9 shows the moment of execution of the persistence techniques used per malware class. Overall, starting at boot seems the most popular time of execution. For grayware, 50.1% of the persistence techniques used require user interaction. This is in line with the expectations since grayware might not look malicious to the user.

Table 5: Measurement of persistence technique class adoption in the 10 most persistent malware families in Dataset II. Tot. P. = Total Persistent; Not P. = Not Persistent.

Family	Reg	Reg & FS	FS OS files	FS Apps	FS DB	Tot. P.	Not P.
Berbew	100.0% (418)	-	-	-	-	418	0
Agenttesla	85.2% (161)	-	18.5% (35)	-	-	189	304
Remcos	83.6% (138)	-	19.4% (32)	-	-	165	182
Loveletter	84.0% (137)	-	85.3% (139)	-	-	163	36
Ekstak	100.0% (155)	-	-	-	-	155	0
Taskun	97.2% (139)	-	6.3% (9)	-	-	143	66
Noon	91.0% (121)	-	10.5% (14)	-	-	133	134
Dcrat	98.5% (129)	63.4% (83)	35.9% (47)	-	-	131	164
Strab	23.7% (27)	-	78.9% (90)	-	-	114	185
Snakelogger	89.3% (92)	-	10.7% (11)	-	-	103	213

5 Discussion & Takeaways

Our research sheds light on Windows malware persistence, revealing that both our understanding of the phenomenon and the state-of-the-art detection rules are inadequate to deal with real-world malware. We summarize four main takeaways.

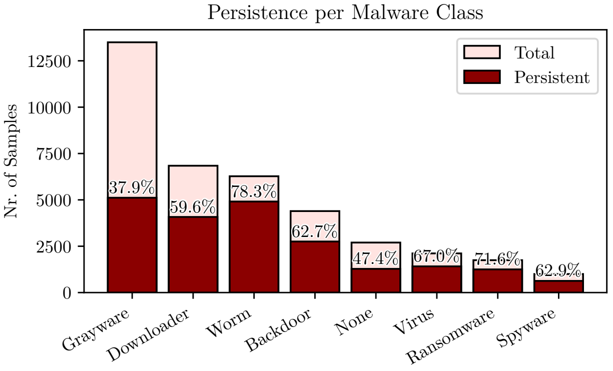

Figure 8: Persistence rate per malware class in Dataset 1: Classes are labeled using AVClass [86, 87], and only those with at least 1,000 samples are shown.

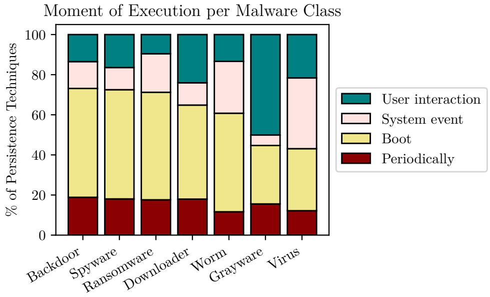

Figure 9: Execution timing distribution per malware class in Dataset I, based on AVClass labels. Only classes with more than 1,000 samples are included.

◇ **Overall Prevalence.** Persistence enables malware to survive on the system without the need to be reinfected, and is therefore often considered an essential aspect of most malware infections. However, our experiments show that only 55.2% of malware is persistent. This is important for incident responders, as the absence of running malware does not prove that no execution occurred—malware may exfiltrate data and delete itself without being resident. It is also an important finding for malware analysts, showing that the absence of persistence does not imply benign intent. Our experiments further show that persistence adoption varies across families and malware classes–e.g., grayware and downloaders often lack it. While expected, our findings validate this intuition.

◇ **Adoption Strategies.** Among persistent samples, over 30% employ two or more persistence techniques—a critical insight for incident responders, as removing a single vector may not fully eradicate the malware. While common techniques dominate due to their simplicity, we identified 42 distinct methods out of 72 documented, highlighting a long-tail distribution. The top 10 techniques cover many families, but full detection requires attention to rarer methods.

Another key finding is that malware families often switch persistence techniques over time. This means that knowing the family and its typical behavior doesn't guarantee knowledge of the specific technique used. Researchers must consider this, as analyzing a single sample per family would very likely lead to incomplete conclusions.

Lastly, our experiments show that only 43.2% of samples aim for execution at boot. Although persistence is often equated with boot-time execution, many samples prefer to trigger on system events, schedules, or user actions. Thus, collections focusing solely on boot-time techniques overlook a substantial portion of persistent malware.

◇ **Undocumented & Undetected Techniques.** One might expect the full list of program execution triggers to be well-known and clearly documented by Microsoft. However, our study reveals that information on persistence techniques is often sparse and scattered across various sources. While common methods like the `Run` Registry key are well-covered, others—such as `AutoplayHandlers`—lack accessible documentation. Alarmingly, current tools detect only a subset of the techniques we documented, despite real-world malware using lesser-known methods to evade detection.

By analyzing nearly 50K samples, we discovered one previously undocumented technique and new variants of known techniques. This suggests that larger datasets may reveal even more undocumented methods. Unfortunately, validating each alert manually is labor-intensive. Combined with frequent OS updates—which may introduce new persistence mechanisms—this creates an environment where malware authors have the upper hand. A scalable approach is essential. Our work offers a foundational step toward this goal.

◇ **Detection Bypasses.** Our systematization reveals that many persistence techniques have multiple implementation variants and that malware authors often employ *Detection Bypasses* to hide system modifications and evade detection. Although monitoring Windows API calls is effective for most techniques we documented, attackers can exploit less commonly monitored Windows subsystems to achieve the same effect more stealthily. In our experiments, we identified two previously unknown detection bypasses actively used in the wild.

Another common evasion strategy is the obfuscation of Registry and file system paths. While our detection rules are designed to handle such variants, future malware may develop new evasion methods.

◇ **Recommendations.** For *industry*, beyond regular scans, it is essential to log system events, as 44.8% of malware is non-persistent and may self-delete after execution. Persistence detection rules must be regularly updated, given the wide range of techniques, variants, and evasion methods. For *incident responders*, careful inspection of persistence vectors is crucial, since over 30% of malware employs multiple persistence techniques, and removing just one may be insufficient. For *researchers*, analyzing multiple samples per family is necessary, as behavior varies within the same malware family.

6 Conclusions

In this work, we unveil how Windows malware stays persistent. First, we systematize known persistence techniques into classes. Then, we build an automated analysis framework to identify both known and novel techniques. Analyzing 48,873 samples, we find that about half use one or more persistence techniques—mostly well-documented ones. Yet, certain families employ exotic methods, leading us to discover a new persistence technique and two detection bypasses. To support the community, we release our code and data open-source.

Acknowledgments

We thank our reviewers for their valuable comments and inputs. This work has been supported in part by the Dutch Ministry of Economic Affairs and Climate Policy (EZK) through the AVR project "FirmPatch", the French National Research Agency (ANR) under the France 2030 program under grant "ANR-22-PECY-0007" (DefMal), and the German Federal Ministry for Economic Affairs and Energy (BMWE) under agreement no. 03EI4101D (TRACEY).

References

[1] 2014. *FuzzySecurity Windows Userland Persistence Fundamentals.* https://fuzzysecurity.com/tutorials/19.html.

[2] 2018. *Eideon blog.* https://www.eideon.com/.

[3] 2019. *Defensive and offensive security in a nutshell.* https://giuliocomi.blogspot.com/.

[4] 2019. *Persistence Tactic | MITRE ATT&CK.* https://attack.mitre.org/tactics/TA0003/.

[5] MDSec 2019. *Persistence: "The Continued or Prolonged Existence of Something": Part 3 - WMI Event Subscription.* MDSec. https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-3-wmi-event-subscription/.

[6] 2020. AV-TEST Security Report 2019-2020. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2019-2020.pdf.

[7] 2023. 2023 Cyberthreat Defense Report. https://cyberedgegroup.com/resources/2023-cyberthreat-defense-report/.

[8] Mandiant 2023. *Back in a Bit: Attacker Use of the Windows Background Intelligent Transfer Service.* Mandiant. https://www.mandiant.com/resources/blog/attacker-use-of-windows-background-intelligent-transfer-service.

[9] 2023. *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Sub-technique T1547.001 - Enterprise | MITRE ATT&CK.* https://attack.mitre.org/techniques/T1547/001/.

[10] 2023. *Create or Modify System Process: Windows Service, Sub-technique T1543.003 - Enterprise | MITRE ATT&CK.* https://attack.mitre.org/techniques/T1543/003/.

[11] 2023. *Hash and family of each sample in the Family Dataset.* https://raw.githubusercontent.com/eurecom-s3/DecodingMLSecretsOfWindowsMalwareClassification/main/dataset/malware.

[12] 2023. *Hexacorn | Blog Beyond Good Ol' Run Key - All Parts.* https://www.hexacorn.com/blog/2017/01/28/beyond-good-ol-run-key-all-parts/.

[13] 2023. *Malware Development: Persistence - Part 4. Windows Services. Simple C++ Example.* https://cocomelonc.github.io/tutorial/2022/05/09/malware-pers-4.html.

[14] persistence-info.github.io 2023. *Persistence-Info.Github.Io.* persistence-info.github.io. https://persistence-info.github.io/.

[15] PSBits 2023. *Persistence with Windows Services.* PSBits. https://gtworek.github.io/PSBits/services.html.

[16] 2023. *Startup Folder.* https://persistence-info.github.io/Data/startupfolder.html.

[17] 2024. *Alpha Threat Blogs.* http://www.blog.alphathreat.in/.

[18] 2024. *Capa.* https://mandiant.github.io/capa/.

[19] 2024. *CAPE: Malware Configuration And Payload Extraction.* https://github.com/kevoreilly/CAPEv2.

[20] 2024. *HijackLibs.* https://hijacklibs.net/.

[21] 2024. *InQuest blog.* https://inquest.net/blog/.

[22] 2024. *Malware Statistics & Trend Report.* https://www.av-test.org/en/statistics/malware/.

[23] 2024. *Nasreddine Bencherchali blog.* https://nasbench.medium.com/.

[24] 2024. *Penetration testing lab blog.* https://pentestlab.blog/.

[25] 2024. *Wumb0in blog.* https://wumb0.in/.

[26] 2025. *Black Hat.* https://www.blackhat.com/.

[27] 2025. *BlackBerry ThreatVector Blog.* https://blogs.blackberry.com/en/home.

[28] 2025. *CICADA8 blog.* https://cicada-8.medium.com/.

[29] 2025. *Cocomelonc blog.* https://cocomelonc.github.io/.

[30] 2025. *Cyberark Threat Research Blog.* https://www.cyberark.com/resources/threat-research-blog.

[31] 2025. *Cyble blog.* https://cyble.com/blog/.

[32] 2025. *Google Cloud Blog.* https://cloud.google.com/blog.

[33] 2025. *HackTricks wiki.* https://github.com/HackTricks-wiki/hacktricks/.

[34] 2025. *hasherezade's 1001 nights.* https://hshrzd.wordpress.com/.

[35] 2025. *Helge Klein Blog.* https://helgeklein.com/blog-archive/.

[36] 2025. *IBM X-Force.* https://www.ibm.com/think/x-force.

[37] 2025. *MalwareBazaar.* https://bazaar.abuse.ch/.

[38] 2025. *Malwarebytes Labs.* https://www.malwarebytes.com/blog.

[39] 2025. *Matt Frisbie blog.* https://substack.com/@mattfrisbie.

[40] 2025. *MDSec Insights.* https://www.mdsec.co.uk/knowledge-centre/insights/.

[41] 2025. *Me, myself & IT.* https://skanthak.hier-im-netz.de/home.html.

[42] 2025. *Microsoft Learn.* https://learn.microsoft.com.

[43] 2025. *MITRE ATT&CK.* https://attack.mitre.org/.

[44] 2025. *Oddvar Moe's blog.* https://oddvar.moe/.

[45] 2025. *PSBits.* https://github.com/gtworek/PSBits.

[46] 2025. *The Red Canary Blog.* https://redcanary.com/blog/.

[47] 2025. *Red Team Notes.* https://www.ired.team/offensive-security/persistence.

[48] 2025. *Ristbs's blog.* https://ristbs.github.io/.

[49] 2025. *Securworks blog.* https://www.secureworks.com/blog.

[50] 2025. *Sigma.* https://github.com/SigmaHQ/sigma.

[51] 2025. *Specterops blog.* https://specterops.io/blog/.

[52] 2025. *Stmxcsr blog.* https://stmxcsr.com/.

[53] 2025. *ThreatDown Intelligence blog.* https://www.threatdown.com/blog/.

[54] 2025. *Trustedsec Security Blog.* https://trustedsec.com/blog.

[55] 2025. *Unit 42 Threat Research.* https://unit42.paloaltonetworks.com/category/threat-research/.

[56] 2025. *VirusTotal.* https://www.virustotal.com.

[57] 2025. *VirusTotal Blog.* https://blog.virustotal.com/.

[58] 2025. *Wietze Beukema blog.* https://www.wietzebeukema.nl/blog/.

[59] 2025. *Windows OSHub.* https://woshub.com/.

[60] 2025. *Windows persistence.* https://utwente-scs.github.io/Sok-Windows-Malware-Persistence/.

[61] 2025. *WithSecure Labs.* https://labs.withsecure.com/publications.

[62] 2025. *Zcaler blog.* https://www.zscaler.com/blogs?type=security-research.

[63] Kevin Almansa. 2017. *DLL Proxying.* InfoSec Blog. https://kevinalmansa.github.io/application%20security/DLL-Proxying/.

[64] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. 2009. A View on Current Malware Behaviors.. In *LEET*.

[65] Wietze Beukema. 2023. *Hijacking DLLs in Windows.* https://www.wietzebeukema.nl/blog/hijacking-dlls-in-windows.

[66] Marcus Felipe Botacin, Paulo Lício de Geus, and André Ricardo Abed Grégio. 2018. The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques* 14 (2018).

[67] Savino Dambra, Yufei Han, Simone Aonzo, Platon Kotzias, Antonino Vitale, Juan Caballero, Davide Balzarotti, and Leyla Bilge. 2023. Decoding the Secrets of Machine Learning in Malware Classification: A Deep Dive into Datasets, Feature Extraction, and Model Performance. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[68] Deland-Han. 2025. *Windows Registry for Advanced Users - Windows Server.* https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users.

[69] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and Longitudinal Study of Evasive Behaviors in Windows Malware. *IEEE Computers & Security* (February 2022), 102550.

[70] Zane Gittins and Michael Soltys. 2020. Malware Persistence Mechanisms. *Procedia Computer Science* (2020).

[71] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

[72] last byte. 2024. *PersistenceSniper.* https://github.com/last-byte/PersistenceSniper.

[73] Qi Liu, Muhammad Shoaib, Mati Ur Rehman, Kaibin Bao, Veit Hagenmeyer, and Wajih Ul Hassan. 2024. Accurate and scalable detection and investigation of cyber persistence threats. *arXiv preprint arXiv:2407.18832* (2024).

[74] Lorenzo Maffia, Dario Nisi, Platon Kotzias, Giovanni Lagorio, Simone Aonzo, and Davide Balzarotti. 2021. Longitudinal Study of the Prevalence of Malware Evasive Techniques. *arXiv preprint arXiv:2112.11289* (2021).

[75] Quenby Mahood, Dwayne Van Eerd, and Emma Irvin. 2014. Searching for grey literature for systematic reviews: challenges and benefits. *Research synthesis methods* 5, 3 (2014), 221–234.

[76] Jennifer Mankin. 2013. *Classification of malware persistence mechanisms using low-artifact disk instrumentation.* Ph. D. Dissertation. Northeastern University.

[77] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, and Davide Balzarotti. 2020. Prevalence and impact of low-entropy packing schemes in the malware ecosystem.

In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

[78] Mark Russinovich. 2024. *Autoruns for Windows - Windows Sysinternals*. https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns.

[79] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. 2017. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE.

[80] Kris Oosthoek and Christian Doerr. 2019. SoK: ATT&CK Techniques and Trends in Windows Malware. In *Security and Privacy in Communication Networks*, Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen (Eds.). Springer International Publishing.

[81] Nicholas Phillips and A Ali Gombe. 2022. Sterilized Persistence Vectors (SPVs): Defense Through Deception on Windows Systems. *Proc. CYBERWARE* (2022).

[82] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. 2017. Measuring and Defeating Anti-Instrumentation-Equipped Malware. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[83] Muhammad Usman Rana, Munam Ali Shah, and Osama Ellahi. 2021. Malware Persistence and Obfuscation: An Analysis on Concealed Strategies. In *Proceedings of the International Conference on Automation and Computing (ICAC)*.

[84] Dennis Reidsma, Jeroen van der Ham, and Andrea Continella. 2023. Operationalizing Cybersecurity Research Ethics Review: From Principles and Guidelines to Practice. In *Proceedings of the International Workshop on Ethics in Computer Security (EthiCS)*.

[85] Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. 2012. Prudent practices for designing malware experiments: Status quo and outlook. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[86] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVClass: A Tool for Massive Malware Labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*.

[87] Silvia Sebastián and Juan Caballero. 2020. AVClass2: Massive Malware Tag Extraction from AV Labels. In *Annual Computer Security Applications Conference*.

[88] Jerre Starink, Marieke Huisman, Andreas Peter, and Andrea Continella. 2023. Understanding and Measuring Inter-Process Code Injection in Windows Malware. In *Proceedings of the International Conference on Security and Privacy in Communication Networks (SecureComm)*.

[89] Antonio Villalón-Huerta, Hector Marco-Gisbert, and Ismael Ripoll-Ripoll. 2022. A Taxonomy for Threat Actors' Persistence Techniques. *IEEE Computers & Security* (2022).

[90] Sebastian Vogl, Jonas Pfoh, Thomas Kittel, and Claudia Eckert. 2014. Persistent Data-only Malware: Function Hooks without Code. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

[91] Carsten Willems, Thorsten Holz, and Felix Freiling. 2007. Toward automated dynamic malware analysis using cwsandbox. (2007).

[92] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online Anti-Malware engines. In *Proceedings of the USENIX Security Symposium*.

Appendix

Table 6: All the sources used to create a comprehensive overview of persistence techniques. The cumulative number of techniques exceeds 72, as multiple sources report on the same techniques.

Name	Nr. of relevant techniques.
Hexacorn [12]	40
Persistence-info [14]	39
Pentestlab [24]	34
Mitre ATT&CK† [43]	30
Cocomelonc [29]	24
Red Team Notes [47]	19
Microsoft Learn [42]	10*
Alpha Threat Blogs [17]	8
MDSec [40]	7
Sigma [50]	6*
Stmxcsr blog [52]	6
FuzzySecurity [1]	4
PSBits [45]	4*
Cyberark Threat Research Blog [30]	3*
Oddvar Moe's blog [44]	3
Trustedsec [54]	3
Malwarebytes Labs [38]	2*
Red Canary [46]	2*
WithSecure Labs [61]	2*
Black Hat [26]	1*
BlackBerry ThreatVector Blog [27]	1*
CICADA8 [28]	1*
Cyble [31]	1*
Eideon blog [2]	1
Giulio Comi blog [3]	1
Google cloud blog [32]	1*
HackTricks wiki [33]	1*
Hasherezade [34]	1
Helge Klein [35]	1
IBM X-Force [36]	1*
InQuest [21]	1*
Matt Frisbie [39]	1*
Me, myself & IT [41]	1*
Nasreddine Bencherchali [23]	1*
Ristbs [48]	1
Secureworks [49]	1*
Spectereops blog [51]	1*
ThreatDown Intelligence [53]	1*
Unit 42 Threat Research [55]	1*
VirusTotal blog [57]	1
Wietze Beukema blog[58]	1
Windows OSHub [59]	1*
Wumb0in blog [25]	1*
Zscaler blog [62]	1*

* Some sources do not contain a listing of techniques, nor a grouping of posts related to persistence. In these cases, we list the number of techniques we consulted the source for.
†While this collection is not meant as an overview of Windows persistence techniques, it still does include some information.

Table 7: The 50 most prevalent malware families in Dataset II (after filtering out inactive samples).

Family	%	Family	%	Family	%
Agenttesla	5.9	Cryptbot	1.1	Powedon	0.5
Berbew	5.0	Bladabindi	1.1	Crysan	0.5
Cosmu	4.3	Connectwise	1.1	Lazy	0.5
Remcos	4.2	Reline	1.1	Lummastealer	0.5
Snakelogger	3.8	Injuke	0.9	Tedy	0.5
Strab	3.6	Dofoil	0.9	Agentb	0.4
Dcrat	3.5	Autoitinject	0.9	Coins	0.4
None	3.5	Mokes	0.9	Scrop	0.4
Noon	3.2	Stealc	0.7	Passwordstealer	0.4
Amadey	2.5	Shipup	0.7	Lumma	0.4
Taskun	2.5	Cryptnot	0.7	Clipbanker	0.4
Loveletter	2.4	Stealerc	0.6	Pwsx	0.4
Ekstak	1.9	Leonem	0.6	Swotter	0.4
Runner	1.6	Redcap	0.6	Shiz	0.4
Asyncrat	1.5	Guloader	0.6	Znyonm	0.4
Formbook	1.5	Gamarue	0.6	**Other**	25.5
Credentialflusher	1.4	Quasar	0.6		
Makoob	1.2	Wacatac	0.6		

Table 8: Measurement result on the moment of execution of different persistence classes in Dataset I.

Class		Boot	System Event	Periodic	User Action	Total
Registry		53.9% (13,677)	3.6% (925)	21.9% (5556)	20.6% (5,214)	68.6% (25,372)
Registry & FS		0.4% (9)	86.0% (2,070)	-	13.7% (329)	6.5% (2,408)
FS	OS files	25.7% (2,280)	30.7% (2,721)	-	43.6% (3,871)	24.0% (8,872)
	Apps	-	-	-	100.0% (300)	0.8% (300)
	Database	-	-	100.0% (9)	-	0.0% (9)
Total		43.2% (15,966)	15.5% (5,716)	15.1% (5,565)	26.3% (9,714)	36,961

Table 9: Persistence techniques used per family, in 2021 (Dataset I) and 2024 (Dataset II), with more than 50 samples in both years. Top-3 persistence techniques are shown. Families are sorted alphabetically.

Family	2021		2024	
	Persistent	Techniques	Persistent	Techniques
Berbew	100.0% (85/85)	ShellServiceObjectDelayLoad Registry key - 85	100.0% (418/418)	ShellServiceObjectDelayLoad Registry key - 418
Cosmu	97.7% (86/88)	Host software binary compromise - 86	1.7% (6/357)	Run Registry key - 3 Host software binary compromise - 3
Dofoil	19.7% (14/71)	Windows service - 6 Run Registry key - 5 Lnk shortcut - 2	3.8% (3/78)	Run Registry key - 2 Lnk shortcut - 1 Task Scheduler - 1
Formbook	32.1% (17/53)	Run Registry key - 15 Lnk shortcut - 2	38.4% (48/125)	Task Scheduler - 39 Run Registry key - 7 Startup folder - 5
Injuke	28.0% (21/75)	Uninstall Registry key - 17 Windows service - 15 Browser extension - 14	21.5% (17/79)	Run Registry key - 8 Startup folder - 7 Lnk shortcut - 4
Mokes	28.9% (24/83)	DLL hijack - 18 Browser extension - 8 Windows service - 3	1.4% (1/74)	Windows service - 1
Noon	70.6% (36/51)	Task Scheduler - 24 Internet explorer Registry key - 13 Run Registry key - 11	49.8% (133/267)	Task Scheduler - 109 Run Registry key - 18 Startup folder - 14
Remcos	11.4% (10/88)	Run Registry key - 7 Url shortcut - 3 Startup folder - 3	47.6% (165/347)	Task Scheduler - 98 Run Registry key - 53 Startup folder - 29
Shipup	94.2% (81/86)	AppInit_DLLs Registry key - 81	94.7% (54/57)	AppInit_DLLs Registry key - 54
Strab	72.9% (43/59)	Windows service - 38 Run Registry key - 3 Startup folder - 1	38.1% (114/299)	Startup folder - 90 Run Registry key - 25 Lnk shortcut - 5