

Fileless threats

Article • 04/25/2024

What exactly are fileless threats? The term "fileless" suggests that a threat doesn't come in a file, such as a backdoor that lives only in the memory of a machine. However, there's no one definition for fileless malware. The term is used broadly, and sometimes to describe malware families that do rely on files to operate.

Attacks involve [several stages](#) for functionalities like execution, persistence, or information theft. Some parts of the attack chain may be fileless, while others may involve the file system in some form.

For clarity, fileless threats are grouped into different categories.

Type III

Files required to achieve fileless persistence

EXPLOIT

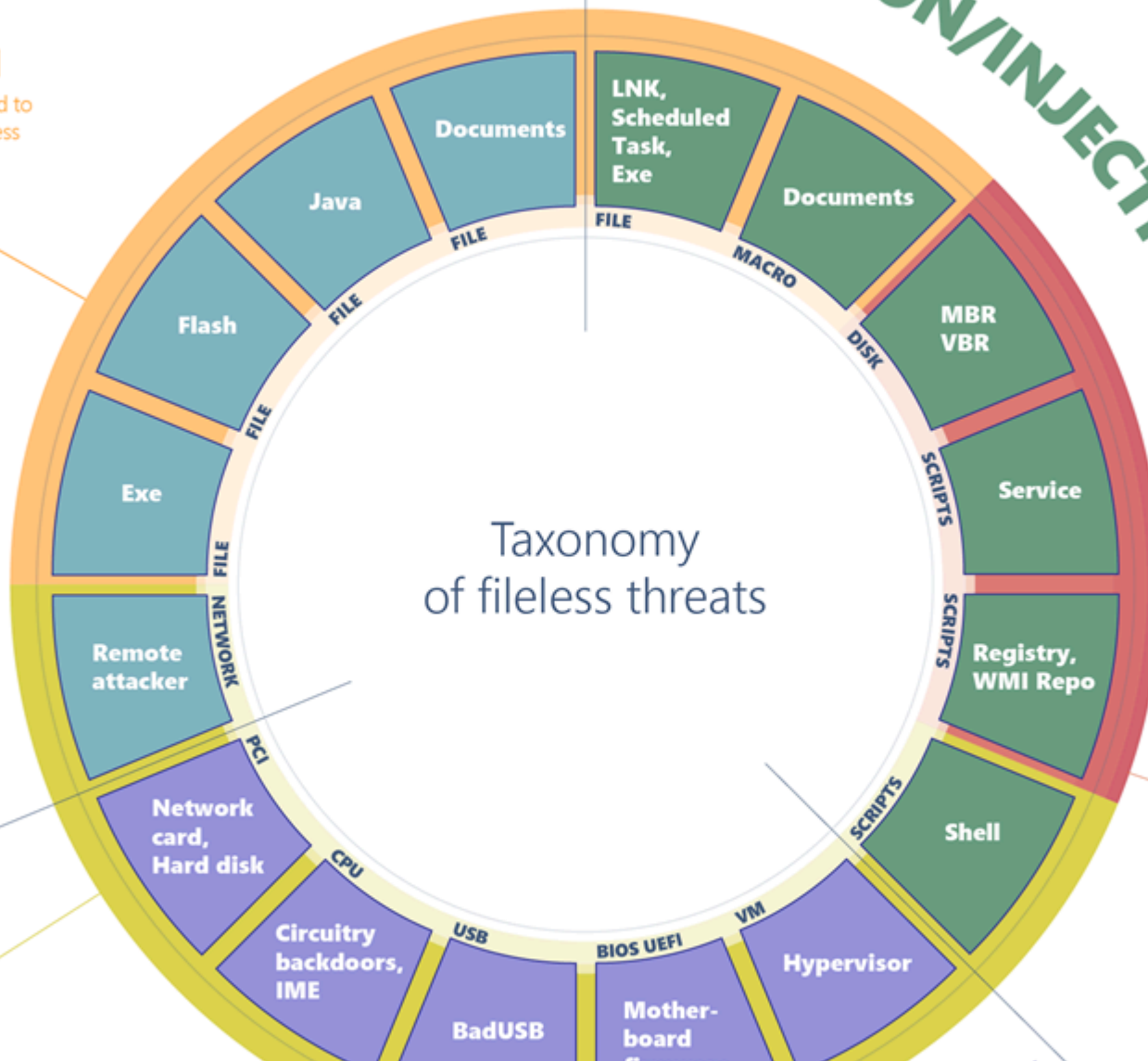
Type I

EXECUTION/INJECTION

Type II

No files written on disk, but some files used indirectly

Taxonomy of fileless threats



type I
No file activity
performed



Figure 1. Comprehensive diagram of fileless malware

Fileless threats can be classified by their entry point, which indicates how fileless malware can arrive on a machine. They can arrive via an exploit, through compromised hardware, or via regular execution of applications and scripts.

Next, list the form of entry point. For example, exploits can be based on files or network data, PCI peripherals are a type of hardware vector, and scripts and executables are subcategories of the execution vector.

Finally, classify the host of the infection. For example, a Flash application may contain a variety of threats such as an exploit, a simple executable, and malicious firmware from a hardware device.

Classifying helps you divide and categorize the various kinds of fileless threats. Some are more dangerous but also more difficult to implement, while others are more commonly used despite (or precisely because of) not being very advanced.

From this categorization, you can glean three main types of fileless threats based on how much fingerprint they may leave on infected machines.

Type I: No file activity performed

A fully fileless malware can be considered one that never requires writing a file on the disk. How would such malware infect a machine in the first place? One example is where a target machine receives malicious network packets that exploit the EternalBlue vulnerability.

The vulnerability allows the installation of the DoublePulsar backdoor, which ends up residing only in the kernel memory. In this case, there's no file or any data written on a file.

A compromised device may also have malicious code hiding in device firmware (such as a BIOS), a USB peripheral (like the BadUSB attack), or in the firmware of a network card. All these examples don't require a file on the disk to run, and can theoretically live only in memory. The malicious code would survive reboots, disk reformats, and OS reinstalls.

Infections of this type can be particularly difficult to detect because most antivirus products don't have the capability to inspect firmware. In cases where a product does have the ability to inspect and detect malicious firmware, there are still significant challenges associated with remediation of threats at this level. This type of fileless malware requires high levels of sophistication and often depends on particular hardware or software configuration. It's not an attack vector that can be exploited easily and reliably. While dangerous, threats of this type are uncommon and not practical for most attacks.

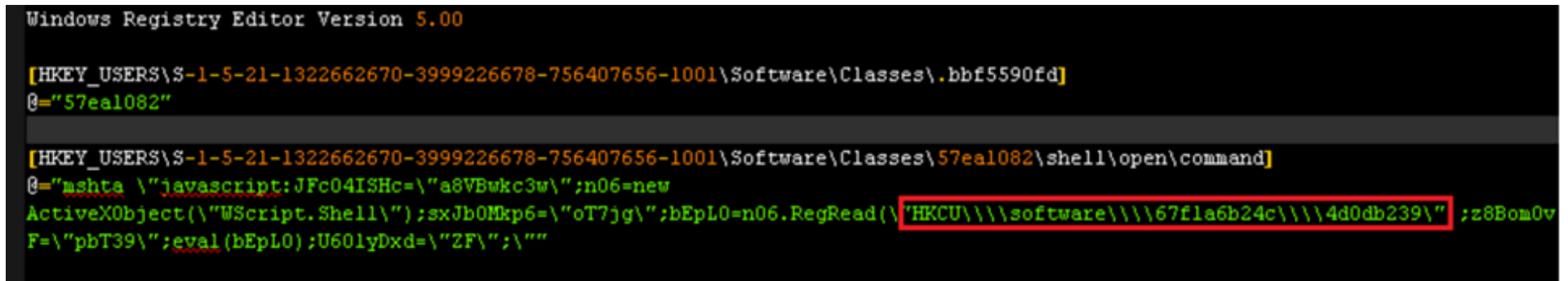
Type II: Indirect file activity

There are other ways that malware can achieve fileless presence on a machine without requiring significant engineering effort. Fileless malware of this type doesn't directly write files on the file system, but they can end up using files indirectly. For example, with the [Poshspy backdoor](#) attackers installed a malicious PowerShell command within the WMI repository and configured a WMI filter to run the command periodically.

It's possible to carry out such installation via command line without requiring a backdoor to already be on the file. The malware can be installed and theoretically run without ever touching the file system. However, the WMI repository is stored on a physical file in a central storage area managed by the CIM Object Manager, and usually contains legitimate data. Even though the infection chain does technically use a physical file, it's considered a fileless attack because the WMI repository is a multi-purpose data container that can't be detected and removed.

Type III: Files required to operate

Some malware can have a sort of fileless persistence, but not without using files to operate. An example for this scenario is Kovter, which creates a shell open verb handler in the registry for a random file extension. Opening a file with such extension will lead to the execution of a script through the legitimate tool mshta.exe.

A screenshot of the Windows Registry Editor. The title bar reads "Windows Registry Editor Version 5.00". The left pane shows the tree structure expanded to "HKEY_USERS\S-1-5-21-1322662670-3999226678-756407656-1001\Software\Classes\57eal082\shell\open\command". The right pane displays the registry value: "mshta \"javascript:JFc04ISHc=\"a8VBwkc3w\";n06=new ActiveXObject(\"WScript.Shell\");sxJb0Mxp6=\"oT7jg\";bEpL0=n06.RegRead(\"HKCU\\\\software\\\\67f1a6b24c\\\\4d0db239\";z8Bom0vF=\"pbT39\";eval(bEpL0);U60lyDxd=\"ZF\";\"\"". The path "HKCU\\\\software\\\\67f1a6b24c\\\\4d0db239\" is highlighted with a red rectangle.

```
Windows Registry Editor Version 5.00

[HKEY_USERS\S-1-5-21-1322662670-3999226678-756407656-1001\Software\Classes\57eal082]
@="57eal082"

[HKEY_USERS\S-1-5-21-1322662670-3999226678-756407656-1001\Software\Classes\57eal082\shell\open\command]
@="mshta \"javascript:JFc04ISHc=\"a8VBwkc3w\";n06=new
ActiveXObject(\"WScript.Shell\");sxJb0Mxp6=\"oT7jg\";bEpL0=n06.RegRead(\"HKCU\\\\software\\\\67f1a6b24c\\\\4d0db239\";z8Bom0v
F=\"pbT39\";eval(bEpL0);U60lyDxd=\"ZF\";\"\""
```

Figure 2. Kovter's registry key

When the open verb is invoked, the associated command from the registry is launched, which results in the execution of a small script. This script reads data from a further registry key and executes it, in turn leading to the loading of the final payload. However, to trigger the open verb in the first place, Kovter has to drop a file with the same extension targeted by the verb (in the example above, the extension is .bbf5590fd). It also has to set an autorun key configured to open such file when the machine starts.

Kovter is considered a fileless threat because the file system is of no practical use. The files with random extensions contain junk data that isn't usable in verifying the presence of the threat. The files that store the registry are containers that can't be detected and deleted if malicious content is present.

Categorizing fileless threats by infection host

Having described the broad categories, we can now dig into the details and provide a breakdown of the infection hosts. This comprehensive classification covers the panorama of what is usually referred to as fileless malware. It drives our efforts to research and develop new protection features that neutralize classes of attacks and ensure malware doesn't get the upper hand in the arms race.

Exploits

File-based (Type III: executable, Flash, Java, documents): An initial file may exploit the operating system, the browser, the Java engine, the Flash engine, etc. to execute a shellcode and deliver a payload in memory. While the payload is fileless, the initial entry vector is a file.

Network-based (Type I): A network communication that takes advantage of a vulnerability in the target machine can achieve code execution in the context of an application or the kernel. An example is WannaCry, which exploits a previously fixed vulnerability in the SMB protocol to deliver a backdoor within the kernel memory.

Hardware

Device-based (Type I: network card, hard disk): Devices like hard disks and network cards require chipsets and dedicated software to function. Software residing and running in the chipset of a device is called firmware. Although a complex task, the firmware can be infected by malware.

CPU-based (Type I): Modern CPUs are complex and may include subsystems running firmware for management purposes. Such firmware may be vulnerable to hijacking and allow the execution of malicious code that would operate from within the CPU. In December 2017, two researchers reported a vulnerability that can allow attackers to execute code inside the [Management Engine \(ME\)](#) present in any modern CPU from Intel. Meanwhile, the attacker group PLATINUM has been observed to have the capability to use Intel's [Active Management Technology \(AMT\)](#) to perform [invisible network communications](#), bypassing the installed operating system. ME and AMT are essentially autonomous micro-computers that live inside the CPU and that operate at a very low level. Because these technologies' purpose is to provide remote manageability, they have direct access to hardware, are independent of the operating system, and can run even if the computer is turned off.

Besides being vulnerable at the firmware level, CPUs could be manufactured with backdoors inserted directly in the hardware circuitry. This attack has been researched and proved possible in the past. It has been reported that certain models of x86 processors contain a

secondary embedded RISC-like CPU core that can [effectively provide a backdoor](#) through which regular applications can gain privileged execution.

USB-based (Type I): USB devices of all kinds can be reprogrammed with malicious firmware capable of interacting with the operating system in nefarious ways. For example, the [BadUSB technique](#) allows a reprogrammed USB stick to act as a keyboard that sends commands to machines via keystrokes, or as a network card that can redirect traffic at will.

BIOS-based (Type I): A BIOS is a firmware running inside a chipset. It executes when a machine is powered on, initializes the hardware, and then transfers control to the boot sector. The BIOS is an important component that operates at a low level and executes before the boot sector. It's possible to reprogram the BIOS firmware with malicious code, as has happened in the past with the [Mebromi rootkit](#).

Hypervisor-based (Type I): Modern CPUs provide hardware hypervisor support, allowing the operating system to create robust virtual machines. A virtual machine runs in a confined, simulated environment, and is in theory unaware of the emulation. A malware taking over a machine may implement a small hypervisor to hide itself outside of the realm of the running operating system. Malware of this kind has been theorized in the past, and eventually real hypervisor rootkits [have been observed](#), although few are known to date.

Execution and injection

File-based (Type III: executables, DLLs, LNK files, scheduled tasks): This is the standard execution vector. A simple executable can be launched as a first-stage malware to run an additional payload in memory, or injected into other legitimate running processes.

Macro-based (Type III: Office documents): The [VBA language](#) is a flexible and powerful tool designed to automate editing tasks and add dynamic functionality to documents. As such, it can be abused by attackers to carry out malicious operations like decoding, running, or injecting an executable payload, or even implementing an entire ransomware, like in [the case of qkG](#). Macros are executed within the context of an Office process (e.g., Winword.exe) and implemented in a scripting language. There's no binary executable that an antivirus can inspect. While Office apps require explicit consent from the user to execute macros from a document, attackers use social engineering techniques to trick users into allowing macros to execute.

Script-based (Type II: file, service, registry, WMI repo, shell): The JavaScript, VBScript, and PowerShell scripting languages are available by default on Windows platforms. Scripts have the same advantages as macros, they are textual files (not binary executables) and run within the context of the interpreter (like wscript.exe, powershell.exe), which is a clean and legitimate component. Scripts are versatile and can be run from a file (by double-clicking them) or executed directly on the command line of an interpreter. Running on the command line allows malware to encode malicious scripts as autostart services inside [autorun registry keys](#) as [WMI event subscriptions](#) from the WMI repo. Furthermore, an attacker who has gained access to an infected machine may input the script on the command prompt.

Disk-based (Type II: Boot Record): The Boot Record is the first sector of a disk or volume, and contains executable code required to start the boot process of the operating system. Threats like [Petya](#) are capable of infecting the Boot Record by overwriting it with malicious code. When the machine is booted, the malware immediately gains control. The Boot Record resides outside the file system, but it's accessible by the operating system. Modern antivirus products have the capability to scan and restore it.

Defeating fileless malware

At Microsoft, we actively monitor the security landscape to identify new threat trends and develop solutions to mitigate classes of threats. We instrument durable protections that are effective against a wide range of threats. Through AntiMalware Scan Interface (AMSI), behavior monitoring, memory scanning, and boot sector protection, [Microsoft Defender for Endpoint](#) can inspect fileless threats even with heavy obfuscation. Machine learning technologies in the cloud allow us to scale these protections against new and emerging threats.

To learn more, read: [Out of sight but not invisible: Defeating fileless malware with behavior monitoring, AMSI, and next-gen AV](#)

Additional resources and information

Learn how to [deploy threat protection capabilities across Microsoft 365 E5](#).
