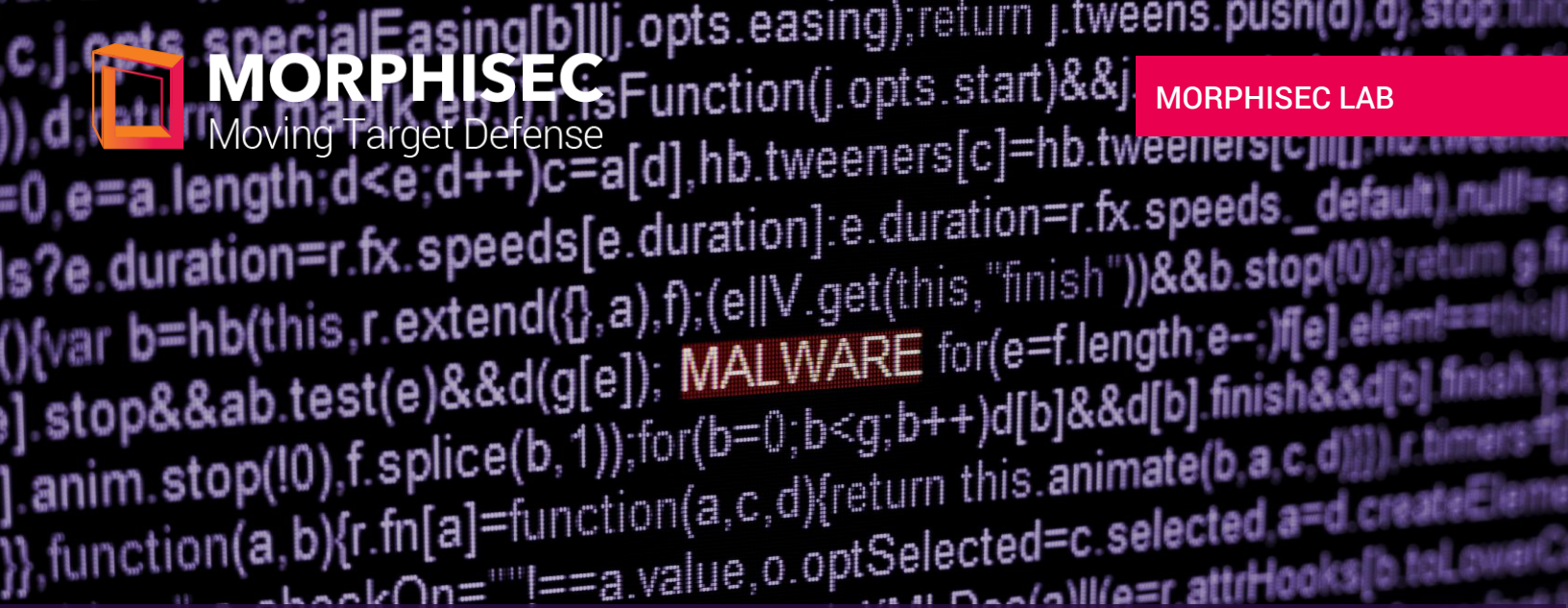

Fileless Malware: Attack Trend Exposed

A report co-authored by Michael Gorelik, CTO and VP R&D, and Roy Moshailov, Malware Researcher at Morphisec.

INTRODUCTION

Fileless malware is a type of a malicious code execution technique that operates completely within process memory; no files are dropped onto the disk. Without any artifacts on the hard drive to detect, these attacks easily evade current detection solutions.

Also known as in-memory or non-malware attacks, fileless malware has existed for years but posed a limited threat as it was very rare and was removed upon system reboot. This changed in 2014 with Poweliks, a click-fraud Trojan which was the first fileless malware to demonstrate persistency functionality. Today, fileless techniques are part of almost any cybercrime or nation-sponsored groups' arsenal and present one of the most dangerous threats to organizations in every industry. The new 2017 State of Endpoint Security Risk study by Ponemon found that the number of fileless attacks increased by 45% in 2017 and that 77% of successful breaches involved fileless techniques.

"Fileless Malware: Attack Trend Exposed" traces the evolution of this trending attack vector, as marked by exponential growth in both fully fileless attacks and commodity malware adopting fileless tactics. It looks at different fileless techniques and examines how various malware incorporates these techniques to avoid being detected.

Script-based malware is often considered to be in the same category as it does not drop any portable executable files (PE) on disk. It's not 100% fileless however since it does drop script-based interpreted files such as JavaScript, HTA, VBA, PowerShell, etc. The malware is executed using legitimate Windows processes, making it still very difficult to detect. In this report we consider both fully fileless and script-based malware types.

WHY IS FILELESS MALWARE SO DIFFICULT TO DETECT?

While security detection solutions apply a number of techniques to identify and detect malicious activity, static analysis is one of the main foundations. Static analysis can be performed with no execution of the code and is generally applied without interference to normal user operations (offline processing). When it works, this implies a shorter detection cycle and better performance scale. The basic requirement for any static analysis technique is that the binary is visible and accessible.

When there are no files on the disk or when the code persists solely through process memory, the code is not accessible and, without doing in-memory dynamic analysis, the code is also not visible. **No Code/Files => No Detection.**

Whitelisting can help – certainly the liberal application of whitelisting solutions helps to limit execution by interpreters. But it also limits the operational flexibility of an enterprise. Moreover, we see a clear pattern of attackers inventing new patterns to bypass whitelisting solutions on a weekly basis.

Since we are also talking about quasi-fileless malware, we should say a few words about interpreters. Although interpreted files such as JavaScript, VisualBasic, HTA and PowerShell can be scanned, this introduces even bigger issues:

- Where should we stop, are we to scan .txt files,.sct files, .xml files?
- Are we to build a parser /interpreter for each type of interpreted file?
- Are we to block any suspicious string, even if it is just a comment in a report?

These are the questions solutions face when trying to balance false positives, user interference and early detection. This is the reason some security vendors limit their static scanning only to a specific type of interpreted file. But even in this case, they still have difficulty scanning those files as a result of easily available obfuscation options (but this is a different conversation).

TYPES OF FILELESS TECHNIQUES

The main fileless techniques utilized today by the different malware families can be divided into three types:

- **Windows registry manipulation** – In this technique the fileless code is usually written and executed directly from the registry by a regular Windows process. This helps to achieve some of the following: persistence / bypass UAC / bypass whitelisting / injecting code into different processes.
- **Memory code injection** – This technique allows the malware to keep living and reside solely within process memory while the processes are running on the system. The malware will distribute and re-inject itself into numerous legitimate processes that are critical to normal Windows operational activity and therefore cannot be whitelisted or even scanned. Security vendors will need a proper justification to kill, block or quarantine such a process, making it a very attractive target for a hacker. Some well-known code injection techniques are remote thread injection, APC, atom bombing, process hollowing, local shellcode injection and reflective loading.
- **Script-based** – As described previously, this is not a 100% fileless technique but creates similar issues for detection solutions and is one of the preferred methods by attackers to maintain stealth.

MALWARE UTILIZING FILELESS TECHNIQUES

Many types of malware have added fileless techniques to their arsenal. They are not truly fileless malware as they didn't make a full transition but have adopted fileless techniques in their attacks. We'll look at both these and more advanced cases of fully fileless malware.

Hancitor / Chanitor

Hancitor campaigns are social engineering based campaigns that have hit millions of homes and enterprises over the past 2 years. In most of the cases it uses a macro based document delivered to the target by email (usually an enterprise employee). Hancitor is considered one of the more sophisticated attack types available and used by cyber-criminal groups. It employs a number of fileless techniques:

- Local shellcode injection – as explained in this blog post, the shellcode is injected into the WinWord process using an application-defined callback function. Below is a possible list of such functions. Hancitor has used most of these, executed by the Visual Basic macro, but there are other methods that can be used for shellcode injection:

EnumTimeFormatsW	EnumResourceTypesExA
EnumResourceTypesA	EnumResourceTypesExW
EnumCalendarInfoW	EnumSystemCodePagesA
CallWindowProcA	EnumSystemCodePagesW
GraySprayA	EnumSystemLanguageGroupsA
CreateTimerQueueTimer	EnumSystemLanguageGroupsW
EnumChildWindows	EnumSystemLocalesA
CallWindowProcA	EnumSystemLocalesW
CallWindowProcW	EnumThreadWindows
CertEnumSystemStoreLocation	EnumTimeFormatsA
CreateTimerQueueTimer	EnumTimeFormatsW
EnumCalendarInfoA	EnumUILanguagesA
EnumCalendarInfoW	EnumUILanguagesW
EnumDateFormatsA	EnumWindowStationsA
EnumDateFormatsW	EnumWindowStationsW
EnumDesktopWindows	EnumWindows
EnumDesktopsA	GrayStringA
EnumDesktopsW	GrayStringW
EnumLanguageGroupLocalesA	SHCreateThread
EnumLanguageGroupLocalesW	SHCreateThreadWithHandle

EnumPropsExA	SendMessageCallbackA
EnumPropsExW	SendMessageCallbackW
EnumPwrSchemes	
EnumResourceTypesA	
EnumResourceTypesW	

- Process Hollowing – as described in this blog, the first appearances of Hancitor (starting in November 2016) created legitimate processes (svchost & explorer) and used a known technique of CreateProcess in suspend state, unmapping the executable memory and replacing it with the malicious code.

Kovter and Poweliks

Poweliks as **fully fileless** malware appeared first in 2014 and made headlines due to its innovative persistency method through registry. **Kovter** is the modern Poweliks variant. It first appeared in 2015 and it borrows some of Poweliks' fileless methods to stay stealthy and bypass today's detection techniques.

KovCoreG is one of the more sophisticated groups that distribute constantly evolving Kovter variants. Over the past 3 years, Kovter has been distributed through Exploit kits, Macro documents, Scripts (Chrome and Flash updates), and more.

There are multiple variants of Kovter, some are fully fileless executing from registry utilizing different legitimate Windows processes (WScript for JavaScript and VBScript, CMD for batch, MSHTA for HTA code, PowerShell, etc.). Some are partially fileless and execute interpreted scripts that are written to disk. This blog post analyzes a campaign using one of the numerous Kovter variants successfully prevented by Morphisec.

Main fileless techniques in use by Kovter:

- Windows registry manipulation for persistency
- Local injection – several stages of shellcode are injected into the process and executed by the use of VirtualAlloc, memset and CreateThread

Although in some cases Kovter may download a next stage payload to disk, in many cases it stays with a fully fileless, persistent click-fraud variant.

Sorebrect Ransomware

While most of the advanced ransomware today utilize at least one fileless technique (usually process hollowing on the same ransomware binary), Sorebrect was the first discovered ransomware to become fully fileless. Main fileless techniques used by the ransomware:

Process Hollowing + PsExec – Sorebrect utilized a stealthy PsExec technique for propagation and infection (as was done by NotPetya and Samsam). While staying fileless, it executed the process hollowing technique on a legitimate Windows process (svchost.exe), eventually encrypting the disk using a legitimate Windows process.

Dridex

The Dridex banking Trojan is an advanced malware family that rapidly adopts fileless techniques into its arsenal. A recent version of Dridex was one of the first to use a new code injection technique called Atombombing, which injects malicious code into any process by utilizing the Windows Atom Tables. This code injection method bypasses both whitelisting and static detection techniques.

SMB - EternalBlue – DoublePulsar / WannaCry

Although many are familiar with the SMB exploit that was used in the **WannaCry** ransomware breakout, only a few know about the fileless malware variants utilizing the same exploits.

SMB exploit is a perfect example of a fileless attack chain: starting from a network exploit, directly injecting a shellcode into the kernel (DoublePulsar) and then injection of code directly into the Usermode (through legitimate windows process – usually lsass.exe). Until this point, it is a fully fileless attack. In the case of WannaCry. a malicious executable was then downloaded by the injected code (lsass.exe) and installed as a service (lsass has system privileges). However, a few weeks before the WannaCry breakout, the same SMB exploit was used for credential theft directly from lsass.exe. In that case no executable was downloaded, and no file was scanned. The main fileless techniques used by this malware:

- CreateRemoteThread – Injecting of a new thread into the usermode lsass.exe process directly from kernel.
- APC – In some cases (following the implementation of some pentesting frameworks) new variants appeared which employed existing threads (threads in alertable state) for malicious shellcode execution.
- Network->Kernel code injection – The malicious shellcode is injected directly into the kernel through the SMB packets without a single file on disk.

DNSMessenger & Meterpreter Injection

DNSMessenger and Meterpreter are fully fileless attacks used by many advanced groups. DNSMessenger is basically a fileless method for delivering malicious code by utilizing the DNS network protocol. Malicious commands are delivered directly into the memory of the running process. Meterpreter is also a method of delivering malicious code through network directly into the memory of the running process, although usually utilizing TCP or HTTP protocols. While Meterpreter is widely used by many pentesting frameworks, both methods have been adopted by advanced groups like FIN7 and others. Methods utilized by those attacks are:

- Reflective loading – Meterpreter injection usually works by injecting a DLL code into the process directly from network and then remapping the DLL inside the process. The method requires the shellcode to identify all the required functions in the process and remap those addresses into the injected DLL.
- Local injection – In the case of the DNS messenger, a shellcode is injected into the running process by the same running process after receiving it from the DNS TXT records (using VirtualAlloc). This makes for stealthier execution and evasion from behavior detection solutions.

Packers

Although packing is a legitimate way to compress executable, essentially, it's in-memory self-modifying code that alters the memory state of the process. The same technique is utilized by many malware families for signature re-creation and more importantly – behavior detection evasion. Overall packing can also be used as a method for code injection by rewriting the existing executable and recreating its code after decryption and remapping of the new functionality. Malware likes to hide their real API and functionality with encryption of the functions and execution of a position independent code (shellcode). The same code does not use much of the declared API and performs reflective loading of new malicious DLLs. We identify this technique as fileless because the result is running malicious code that was created purely in memory, without writing to the disk. Many known malware heavily utilize packing and local code injection techniques to evade static analysis, including Locky, Cerber, LockyBot, Andromeda and others.

Conclusions

Fileless has gone mainstream, with fileless techniques being widely incorporated into existing malware and serving as the basis for new, highly advanced, fully fileless malware. Fileless-type attacks were one of the fastest growing threat vectors in 2017 and are predicted to grow even more in 2018. These attacks execute code in memory and use legitimate system resources to perform malicious actions. They leave no traces on disk, letting them completely bypass detection solutions. The majority of today's successful endpoint compromises are caused by fileless attacks; they have nearly a 10x greater success rate than file-based attacks.

Enterprises need to recognize this attack vector and look for new technologies dedicated to preventing fileless, in-memory threats. Morphisec's Moving Target Defense outmaneuvers fileless attacks by morphing the memory space so the system resources they target are inaccessible.