Using Sysmon to Enrich Security Onion's Host-Level Capabilities

GIAC (GCFA) Gold Certification

Author: Josh Brower, Josh@DefensiveDepth.com

Advisor: Richard Carbone

Accepted: March 19, 2015

Abstract

With more network traffic being encrypted, as well as the persistence of advanced adversaries, it is becoming increasingly imperative that there is greater visibility at the host-level. With this greater visibility comes the ability to more efficiently detect and respond to threats. This paper highlights the use of Sysmon to enrich existing Windows host visibility capabilities in Security Onion, as well as how to use this increased visibility in detection and incident response. In this paper, the author has developed custom parsers and rulesets for integrating host-based data into Security Onion, something which to date had not yet been done for this project.

1. Introduction

1.1. IDS & Network Security Monitoring

In 2003, Gartner declared Intrusion Detection Systems as a "market failure" primarily because of the high false positives and negatives, and the significant amount of time and resources needed to monitor and validate alerts. (Gartner, 2003) Within a year, Richard Bejtlich published the first major technical book outlining Network Security Monitoring (NSM), a methodology for gathering and analyzing network-centric data to help detect and respond to intrusions. (The Tao of Network Security Monitoring, 2004) One of the significant differentiators of NSM is that alerts generated by an IDS are only one of several types of data that is collected. Though there are various types of NSM data, practitioners would typically agree that NSM data is network-centric. The other types of data include:

- Full Content Data
- Session Data
- Statistical Data
- Extracted Content
- Metadata

Once an alert is generated, the analyst uses the other collected data to quickly and efficiently ascertain whether the alert needs to be escalated. The collection and use of data types other than alerts are one of the foundational elements of NSM.

In the eleven years since Bejtlich wrote his seminal book, practitioners have seen a number of issues in the last few years that have shown some of the limitations of network-centric monitoring: the rise of encrypted-by-default web traffic, which blinds defenders to most NSM data types, and the persistence of advanced adversaries, which has given rise to intelligence-driven computer network defense (CND). Intelligence-driven CND can be used in a network-centric situation; however, adversaries leave behind both network and host artifacts, meaning that a network-centric monitoring strategy will miss out on crucial opportunities to detect non-network adversarial activity.

1.2. Rise of the Encrypted Web

The collection of NSM data is typically through a TAP or SPAN on a strategic chokepoint in the network. If the network data between the client and server is encrypted, a number of types of NSM data will be useless to the analyst—full content, extracted content, and certain types of alerts. With the revelations of the past few years that a number of governments around the world have been intercepting their citizen's unencrypted communications, there has been significant interest in encrypting most, if not all of the web traffic around the world. In 2014, CloudFlare, which hosts a content delivery network (CDN) and security services for two million websites, enabled free SSL for all of their customers. They stated, "Having cutting-edge encryption may not seem important to a small blog, but it is critical to advancing the encrypted-by-default future of the Internet. Every byte, however seemingly mundane, that flows encrypted across the Internet makes it more difficult for those who wish to intercept, throttle, or censor the web." (Prince, 2014)

From a recent study, *The Cost of the "S" in HTTPS*, twenty-five thousand residential ADSL customers saw HTTPS usage in uploads accounting for 80% of traffic, compared to 45.7% in 2012. (Naylor, et al.) This trend is expected to continue for the foreseeable future.

This increase of encryption will typically be seen in north – south traffic, not necessarily east – west traffic, which means NSM sensors deployed to monitor internal traffic may not be so readily affected. However, sensors deployed at network egress points will certainly be affected unless some type of mitigation is put into place. These mitigations would include proxying the SSL traffic so that the network data could be read, though this solution is limited in practice due to performance, privacy, and liability concerns.

1.3. Rise of Intelligence-Driven Computer Network Defense

Unfortunately, it is not just encrypted traffic that harries NSM practitioners – the persistence of advanced adversaries continues unabated. This has given rise to intelligence-driven CND, which is a threat-centric risk management strategy. (Hutchins, Cloppert, & Amin) Simply put, as the defender gathers intelligence about intrusions and the adversary behind them, the defender is able to use this information in future detection

cycles against the adversary. Indicators are a key part of this intelligence. From the formative paper, *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains*: "By completely understanding an intrusion, and leveraging intelligence on these tools and infrastructure, defenders force an adversary to change every phase of their intrusion in order to successfully achieve their goals in subsequent intrusions. In this way, network defenders use the persistence of adversaries' intrusions against them to achieve a level of resilience." (Hutchins, Cloppert, & Amin)

A crucial part of this methodology is the ability to gather quality indicators. Quality indicators are extractable ("Can I find this indicator in my data?"), purposeful ("To what use will I put this indicator?"), and actionable ("If I find this indicator in my data, can I do something with that information?"). (Bianco, Enterprise Security Monitoring, 2013) Without these quality indicators, defenders will not be able to efficiently detect further intrusions by the same adversary. Various forms of indicators have differing values. Consider David Bianco's Pyramid of Pain:

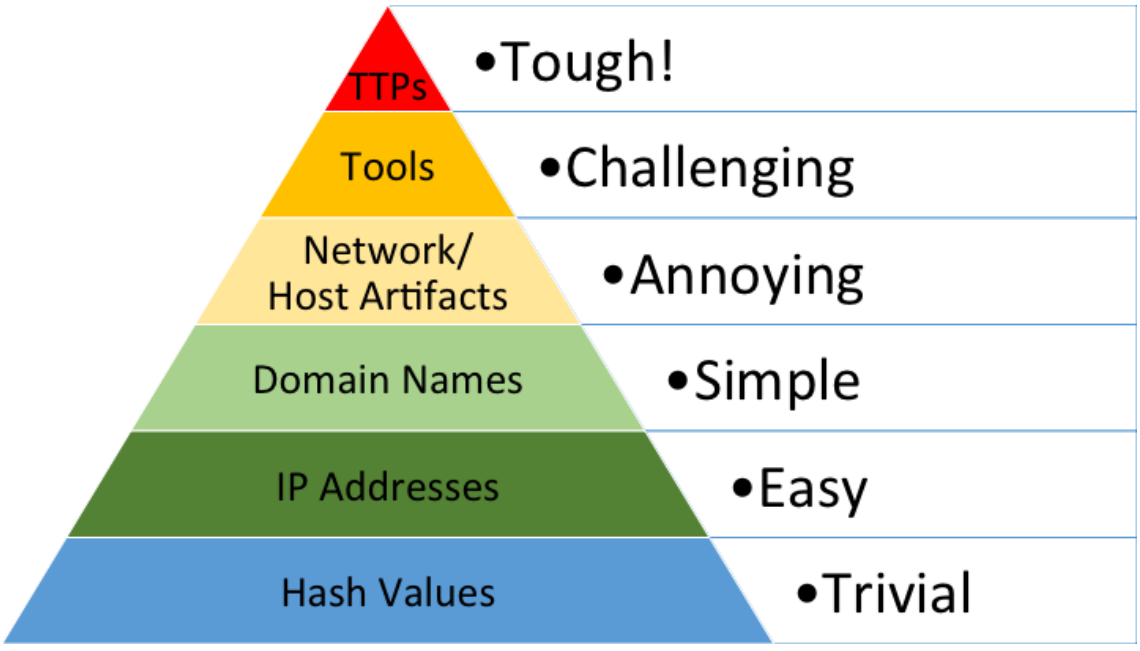

Figure 1. The Pyramid of Pain (Bianco, The Pyramid of Pain, 2014)

It can be seen that Hash Values and IP Addresses are on the bottom of the pyramid. This indicates that though these types of indicators can be useful, they are very easy for the adversary to cycle through, hence the probability of seeing the same indicator

used in multiple campaigns is much lower than tools that the adversary uses (which is much higher on the pyramid). The key point is that as the defender is able to build up their detection strategy around higher quality indicators, this will require the adversary to change their Tools, Tactics, and Procedures (TTPs), which is very costly in terms of time and resources. This does not negate the fact that the lower indicator types are still useful.

Though there are different types of indicators (Atomic, Computed and Behavioral), it is clear that the defender must have indicators that span the gamut of both network and host-level, as an adversary carries out operations in both spaces. (Hutchins, Cloppert, & Amin)

1.4. Enterprise Security Monitoring (ESM)

As NSM practitioners become blind to the majority of the traffic entering and exiting their networks and require the ability to generate quality indicators in both the network and host space, there needs to be a shift to include more than just network-centric data in detection and response strategies. Hosts on the network can be an extremely rich repository of data that can be extracted and used in detection and response in conjunction with NSM data. In essence, this is applying the same type of NSM mindset to host-level data. In fact, this concept has been coined "Enterprise Security Monitoring" by David Bianco. (Bianco, Enterprise Security Monitoring, 2013) ESM integrates intelligence-driven CND principles. As such, a notable point of ESM is the ability to locate relevant indicators pertinent to where an intrusion might be in relation to the kill chain. Because these indicators span both the network and host, there is a need to be able to have access to both categories of data.

Though many tools can generate both NSM and host data, the confounding issues typically revolve around how to efficiently collect the data and present it in a way that makes it usable for alerting, analysis and decision-making. This is where Security Onion brings it all together.

1.5. Security Onion

Security Onion is a NSM platform built on existing tools, maintained primarily by Doug Burks and Scott Runnels. It is based on Ubuntu, and integrates a number of tools for both network and host-level detection and analysis, including: (Burks, 2012)

- Snort – Open source network IDS from Sourcefire.
- Suricata – Open source network IDS from the Open Information Security Foundation.
- OSSEC – Open source host IDS.
- ELSA – Open source centralized log management application.
- Sguil – Open source analyst console for NSM practitioners.
- Bro – Open source network analysis framework.
- Squert – Open source web application used to query and view event data in Sguil.
- Snorby – Open source web application console for NSM practitioners.

Security Onion is built such that as these tools integrate and work together, the full range of NSM data and certain types of host data can be collected, viewed, analyzed and escalated efficiently. The host-level data is provided primarily through the use of OSSEC and ELSA. This paper will focus on enriching this capability through the integration of Sysinternal's Sysmon, so as to augment the detection and response blinded by encrypted traffic as well as gain access to additional host-level indicators.

2. Integrating Sysmon Data into Security Onion

2.1. Sysmon

Sysmon is a Sysinternals tool written by Mark Russinovich and Thomas Garnier, first made public August 2014. Currently at version 2.0 and 605 KB in size, it is installed as a Windows system service. From its official description: (Russinovich & Garnier, 2015)

-Logs process creation with full command line for both current and parent processes.

-Records the hash of process image files using SHA1 (the default), MD5, SHA256 or IMPHASH.

-Includes a process GUID in process create events to allow for correlation of events even when Windows reuses process IDs.

-Include a session GUID in each events [sic] to allow correlation of events on same logon session.

-Logs loading of drivers or DLLs with their signatures and hashes.

-Optionally logs network connections, including each connection's source process, IP addresses, port numbers, hostnames and port names.

-Detects changes in file creation time to understand when a file was really created.

-Rule filtering to include or exclude certain events dynamically.

-Generates events from early in the boot process to capture activity made by even sophisticated kernel-mode malware.

An example process creation event can be seen in Figure 2. All these events are logged to the local machine's Event Log.

```
Process Create:
UtcTime: 1/9/2015 10:38 AM
ProcessGuid: {00000000-afad-54af-0000-0010ba246e00}
ProcessId: 2760
Image: C:\Windows\system32\mstsc.exe
CommandLine: "C:\Windows\system32\mstsc.exe"
User: IT-JB\joshuab
LogonGuid: {00000000-99b7-54af-0000-0020eeb70400}
LogonId: 0x4B7EE
TerminalSessionId: 1
IntegrityLevel: Medium
HashType: SHA256
Hash: 9D5531DB8A212B4AC33B4CA16D914C468A48279236F5170CD389566E6CFA1BF7
ParentProcessGuid: {00000000-99b9-54af-0000-0010acb40500}
ParentProcessId: 4784
ParentImage: C:\Windows\Explorer.EXE
ParentCommandLine: C:\Windows\Explorer.EXE
```

Figure 2. An example of Sysmon Event ID 1: Process Creation

The type of host data that Sysmon covers is three-fourths of the data types from the Pyramid of Pain – Hash values (of all executables that are running), IP Addresses, Domain Names, and some Network/Host Artifacts. Finding another free, lightweight, and feature-rich tool that has the backing of a team like Sysinternals, is an almost impossible task. These reasons are what make Sysmon a good choice for enriching the host-level capabilities of Security Onion.

How will Sysmon data be integrated into Security Onion? For historical queries and manual hunting, Sysmon data will be accessible in ELSA. For generating alerts based on real-time incoming Sysmon events, OSSEC will be utilized.

2.2. Sysmon Event Collection from Servers & Workstations

Two different methods could be employed to collect the Sysmon events from the local client. Possible approaches use only OSSEC or a hybrid architecture where OSSEC and Windows Event Collection are utilized together.

2.2.1. OSSEC

One way to collect the Sysmon events from all installed clients would be to use the Host Intrusion Detection System (HIDS) that Security Onion includes, which is OSSEC. This architecture would include installing OSSEC on all servers and workstations, and configuring it through the <eventchannel> option to send Sysmon logs to Security Onion. (Windows Eventchannel Example) If this were the only function that OSSEC would be used for, most organizations would be reticent to deploy another client to their workstations and servers, especially when there are other, more efficient options to collect the Sysmon data.

2.2.2. Hybrid

The architecture that the author has used and recommends is that of a hybrid model. This would include installing OSSEC on only on servers, as there are typically other types of logs that need collection as well. For workstations, the use of the Windows Event Collector framework is recommended to collect all of the Sysmon logs onto a central Windows system. (Helweg) With the logs all in one location, an OSSEC client can be installed on the collection server, which would process all of the logs and ship

them off to the Security Onion sensor. For offsite users, events can still be collected by making the collector server publically available. Refer to the following diagram for what this particular architecture would look like:

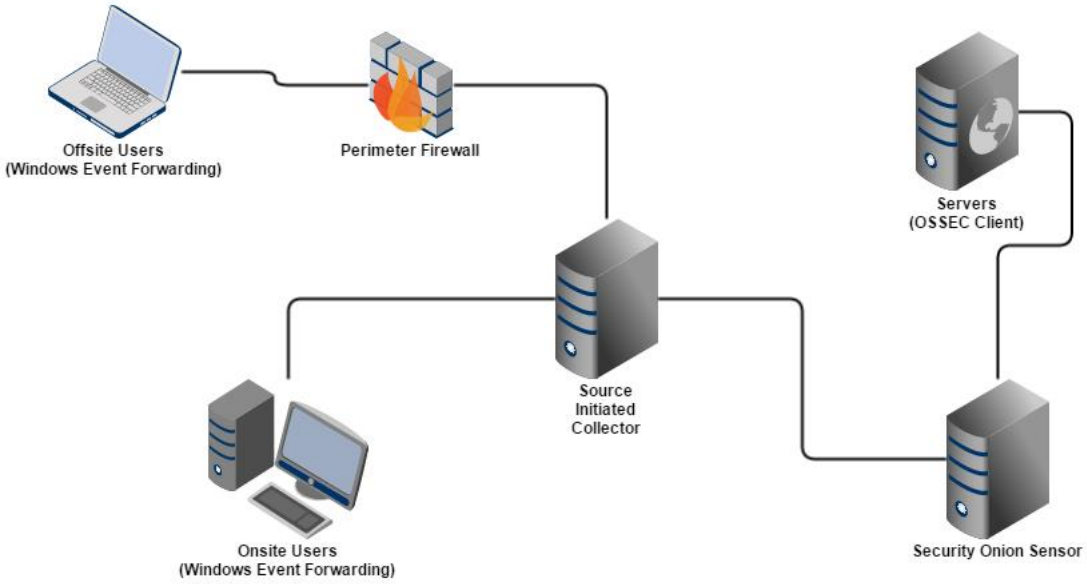

Figure 3. Diagram of hybrid collection model

Now that the logs have been collected and shipped to the Security Onion sensor, they must be processed by both OSSEC and ELSA before the data can be used by either of those tools. Because Sysmon is relatively new, the author of this paper was required to write his own parsers for both ELSA and OSSEC to be able to pull out the relevant data contained in Sysmon events. These parsers have been released by the author under the MIT License, and can be found in appendices C and D. In the near future, these parsers will most likely be integrated into the Security Onion core packages.

2.3. Host Data = "Big Data"

As with any log collection initiative, care must be taken to plan ahead on what type of data to log and how it will be used. It is very easy to turn on logging everywhere and then point the logging source to the Security Onion sensor, not realizing that the number of events will be utterly overwhelming. In Applied Network Security Monitoring, Chris Sanders and Jason Smith lay out a process to plan data collection based on a realistic understanding of organizational threats. (Sanders & Smith, 2014)

Without this mindset, detection and incident response will suffer, as analysts must sift through irrelevant data to find the context they need.

With this background in mind, a discussion around the types of data that Sysmon generates must be had. Though there are other types of Sysmon data that can be generated, it is outside the scope of this work. (Russinovich & Garnier, 2015) The data generated should be folded into the organizational data collection strategy so that only required information is collected.

Event ID 1 - Process Creation:

This can be a low to medium volume event and can be highly useful for both detection and forensic investigations. The recommendation would be to log these events, and send these logs to the Security Onion sensor for both detection capabilities and forensic investigations.

Event ID 2 - File creation time change:

This is a high volume event on both clients and servers and is useful primarily in a forensic investigation. Because of this, the recommendation would be to log these events, but keep them on the source system. If it is a critical asset, it may be appropriate to ship the logs off the system as they are generated.

Event ID 3 - Network Connection Detected:

This can be a high volume event on both clients and servers, based on the network traffic usage of the system. If event collection to the Security Onion sensor can be scoped to a certain key attribute (*connection initiated*, for instance), it would be more viable to log these events to the Security Onion sensor. Because of this, the recommendation would be to log these events, but keep them on the source system. If it is a critical asset, it may be appropriate to ship the logs off the system as they are generated.

Event ID 5 – Process Terminated:

This can be a low to medium volume event. For every event in this category, there should be a corresponding process created event (ID 1). This makes it useful in forensic

investigations. The recommendation would be to log these events and send those logs to the Security Onion sensor for forensic investigations.

Event ID 6 – Driver Loaded:

Depending on the system, this should be a low volume event. This event is highly useful for both detection and forensic investigations. The recommendation would be to log these events, and send those logs to the Security Onion sensor for both detection capabilities and forensic investigations.

With the Sysmon events collected and correctly parsed by OSSEC and ELSA, some practical usage for host data in detection and incident response will now be detailed.

3. Using Host Data in Detection

The following subsections introduce practical examples of using host-level data in detection, whether in relation to an opportunistic or structured threat.

3.1. Monitoring Key Windows Processes for Anomalies

So as to be able to maintain persistence, both targeted and opportunistic threats use certain techniques to attempt to blend into the background of a busy system. One of the primary ways of doing this is by emulating and/or abusing legitimate Windows processes. For instance, malware named *svhost.exe* instead of *svchost.exe*, which is a legitimate process. Another example would be the *Poweliks* class of malware, which hollows out a legitimate process and runs its malicious threads from there. In fact, in the case of *Poweliks*, there is no binary downloaded to the system itself, as it runs entirely in memory.

Using the host data generated by Sysmon, detection of these techniques can become commonplace. The crux of the idea is that it is well known how critical legitimate Windows processes should be running. Through Sysmon-generated data, monitor these processes for known legitimate behavior, and generate an alert when behavior is observed outside of the norm. Based on documentation from SANS and

personal experience, the author maintains publically available documentation on what is normal for critical Windows processes, which can be found in Appendix A. (SANS)

Let us take a closer look at this detection strategy. The current iteration of *Poweliks* hollows a legitimate Windows process, *dllhost.exe*, to perform its malicious tasks. (Harrell, 2014) When the author ran a copy of *Poweliks* on a system with Sysmon installed, the following pertinent data was generated:

Image: C:\Windows\syswow64\dllhost.exe
CommandLine: C:\Windows\syswow64\dllhost.exe

ParentImage: C:\Windows\syswow64\windowspowershell\v1.0\powershell.exe
ParentCommandLine: "C:\Windows\syswow64\windowspowershell\v1.0\powershell.exe" iex $env:a

Typically *dllhost.exe's* parent process would be *svchost.exe*, and at runtime, *dllhost.exe* would be passed the following parameter: */Processid:{}*. As can be seen, the *dllhost.exe* that is started by *Poweliks* falls outside the norm, and would have set off some alerts.

Based on this concept, the author has written more than ten OSSEC rules that cover normal behavior for a number of critical Windows processes. These rules can be found in Appendix F. Keep in mind that the rules were written with the corresponding OSSEC decoder for Sysmon logs, so they may need to be edited if used outside of that particular context. When writing the rules, there were a number of ways to alert on abnormal behavior: Image Location, User Context, Parent Process Image, and finally, how many instances should be running on the system. For simplicity, the ruleset was designed to alert on one abnormal attribute. The most immutable attribute would seem to be the parent image, which is why the ruleset only looks at the parent image for abnormalities. Within this attribute, two abnormalities are checked for. The first is whether the parent process image is known-good. For example, the parent image of *svchost.exe* should only ever be *C:\Windows\System32\services.exe*. The second abnormality is that there are a couple processes that should never spawn a child

process—*lsm.exe* and *lsass.exe*. With this being the case, there are a few rules that look for these particular images as the parent process image and alert if found.

As with any new rulesets, there will certainly be some false positives generated, and in turn, the rules will become better tuned. As the ruleset has been run in a production network, there appears to be a number of instances where the current community documentation on legitimate process behavior is inaccurate and/or incomplete. This can be attributed to the fact that there is no official documentation from Microsoft on what these process attributes should always be on all versions of Windows, hence the current body of knowledge (such as the *SANS Know Normal… Find Evil* poster), has been built up from years of observation.

3.2. Monitoring for Abnormal Application Usage

Rather than download another binary to the system and risk detection, there is a continued progression of adversaries using built-in applications to maintain persistence and/or pivot. This, coupled with the fact that most end-users will not be popping open *cmd.exe* or *powershell.exe*, could create a compelling indicator if this activity is found on an end-user's machine.

Monitoring the use of applications, like *cmd.exe* or *powershell.exe*, could be done through hash lookups, but that would require a large amount of overhead, as the binaries continually change due to Windows updates and upgrades. If the built-in application will be used, the binary will not typically be renamed, so as to not arouse more suspicion. With this being the case, generating alerts when *cmd.exe* or *powershell.exe* is run under the context of a non-system user could be a good indicator. An example OSSEC rule has been written to show how this monitoring could be accomplished—it can be found in Appendix G.

As should be clear, examples like this are context-specific—this particular example will not be applicable in all circumstances; however, it showcases another aspect of how host data can be used in detection.

3.3. Hash Lookups for Key Applications or Indicators

To consolidate and maintain their presence, intruders will need to pivot from their initial beachhead to other systems inside the compromised network. Beyond that, they

also need to connect to (potentially many) systems to carry out their objectives. Tools, such as Sysinternal's PsExec, have been a much-used capability in the past not only because it is a good quality tool, but also because it is legitimately utilized in a number of organizations for IT Operations. If an organization does not use PsExec, or there are certain systems that should never have PsExec run against them, it can be a good indicator if the presence of PsExec is found on them. Though OSSEC rules could be written to look for the name of 'PsExec' as in the previous section, it is highly unlikely that a malicious use of PsExec will use a binary named that way. As such, another method should be considered.

Part of the Sysmon-generated data includes the hash value of the running image. Using this data, and OSSEC's Constant Database (CDB) list lookup feature, all monitored processes can be checked to see if they are in fact PsExec. Based on this concept, an OSSEC rule and corresponding CDB that contains the SHA1 hashes for all PsExec binaries released between 2011 – 2014 was written. If interested, please refer to Appendix H.

As has already been discussed in Section 1.3, the use of indicators is a central tenet in intelligence-driven CND. Using the hash-based technique previously outlined, hashes related to a particular adversary could be detected. Though hashes are trivially easy to change (as can be seen by their position on the Pyramid of Pain), they can still be a useful indicator for detection.

3.4. Network Connections

Up until this point, process-created Sysmon events are the only host data that has been used. Sysmon also generates events when a network connection is initiated or received, notating which process is associated with it. This certainly provides a rich level of detail that could be the basis for a number of different detection alerts. Consider the following possibilities:

- Certain applications that should never be initiating network connections.
- Critical Windows processes initiating connections outbound on port 80/443.
- Critical Windows processes initiating connections outbound to non-internal address space.

After thorough online searches, there does not appear to be comprehensive publically available documentation that would give clear guidance on the previous scenarios. Using Sysmon, control data could be gathered from known-good systems that could provide the necessary information, at which point detailed detection rules could be written.

Having this type of network-to-process host data can also give much-needed context to validating if specific alerts need further follow up. For instance, by default, Skype generates network traffic that is typically classified as Peer-to-Peer (P2P), and a number of Emerging Threats P2P rules will trigger when they see this traffic. When trying to validate the alerts, pulling full content data of the traffic will not help, as it is encrypted. However, with Sysmon installed and monitoring on all endpoints, the analyst can easily pivot to ELSA and search for the 5-Tuple[1] associated with the alert. The analyst will clearly see that the traffic in question is associated with Skype.exe.

Because so many events can be generated for network connections, it may not be possible to use network connection events. However, if the log collection method allows filtering, then only certain types of events could be collected. For instance, if events are being collected through the Windows Event Collection method discussed previously, it is possible to use the XPath[2] syntax to filter out certain events to collect. An example of this would be if there was a desire to only collect network connection events if the network connection was initiated by the source host. This would dramatically reduce the number of events collected, and yet still yield some valuable data to parse through. The following XPath snippet would be of use:

```
<QueryList>
  <Query Id="0" Path="Microsoft-Windows-Sysmon/Operational">
    <Select Path="Microsoft-Windows-Sysmon/Operational">
      *[EventData[Data[@Name='Initiated'] and (Data='true')]]
    </Select>
  </Query>
</QueryList>
```

[1] 5-Tuple refers to the unique combination of Source IP and Port, Destination IP and Port, and finally, the protocol used for the connection.

[2] XPath, the XML Path Language, is used to query data from an XML document—in this case, the XPath query is being used to query data from a Windows Event log, an XML document.

Alternatively, Sysmon itself now has the ability to collect only events for an event type based on the logic applied to event type attributes. For instance, Sysmon could be configured to only collect network events if the associated process image is *svchost.exe*.

4. Using Host Data in Incident Response

The use of host data is not limited to just detection, but incident response as well. When a system has been chosen for a deep-dive forensic investigation, having access to historical logs of the system that provide details of process execution and network connections will be invaluable. During an investigation, searching Sysmon process creation and network logs around a pivot point timestamp will most likely yield process creation logs related to the incident. Unfortunately, Sysmon has not been written to conceal itself in any way – an adversary who has administrator privileges on the system can easily disable the Sysmon service, if they know it is there. However, if these logs are being shipped off to a Security Onion Sensor, there should still be some pertinent logs that make it off-box.

After a deep-dive forensic investigation, there will be a number of indicators that can then be used to search for other compromised systems throughout the organization. ELSA can be used to search for these indicators, as there is a historical body of logs that is maintained therein. Indicators come in all shapes and sizes; unfortunately, if Sysmon is the only host-data capability present, there will be major limitations on what kind of indicators can be searched for. See Appendix B for more details on the specific types of data that Sysmon generates.

4.1. Data Stacking

Data stacking is a process that is used in incident response to look for anomalous data that could be part of a compromise. As defined by Mandiant, data stacking is "the application of frequency analysis to large volumes of similar data in an effort to isolate and identify anomalies." (M-Labs, 2012) One of the core parts of data stacking is an iterative process that allows an investigator to pare down massive volumes of data into a manageable size. Using frequency analysis, anomalies are then identified. For instance, looking through the process-created Sysmon logs in the author's test install of ELSA,

there were about three-hundred events in a twenty-four hour period. One of the essential components of data stacking is finding an attribute of the data that should be the key that searches are based on, and on which the investigation pivots around. For this example, the Image name was used. With a sufficient amount of systems that have been collected from, frequency analysis should show that the vast majority of the executables that have been run have actually been found in the majority of the systems. With this being the case, attention would be drawn to the executables that are found on only a few of the systems. These particular executables are the anomalies that would be investigated further.

As has been discussed previously, a foundational part of data stacking is the ability to pare the data down to manageable slices to aid the analysis. This could be done directly through ELSA. From the scenario in the previous paragraph, it was clear that over seventy-five percent of the executables were run were from the *C:\Windows* directory or subdirectories. This would be a good place to use ELSA to show only the executables that were run in the *C:\Windows** directories. The query for this would be:

Class = SYSMON_PROCESS +SYSMON_PROCESS.IMAGE="C:\Windows" groupby:SYSMON_PROCESS.IMAGE

The query to exclude these results would be:

Class=SYSMON_PROCESS -SYSMON_PROCESS.IMAGE="C:\Windows" groupby:SYSMON_PROCESS.IMAGE)

Both sets of data can now be analyzed from a least frequency run perspective and any anomalies notated for further investigation. This can be done to a certain extent through ELSA, but most likely will done with an external tool. ELSA can export data from queries to XLSX or CSV format, which can be then imported into an appropriate tool.

5. Other Host Data

Data produced by Sysmon is just one example of host data that can be generated. If deploying Sysmon is not an option, enabling the built-in Process Tracking logging on

Windows might be an alternative. Though there is less useful data produced, (i.e. no hash of images) enabling it is as simple as writing a GPO and then collecting the events. The author has written an ELSA parser for these events for use in Security Onion installations, which can be found in appendices I and J.

6. Conclusion

Monitoring at the network-level will bring limited context, particularly with increasing amounts of traffic being encrypted and the persistence of adversaries. Increasing visibility at the host-level is the next logical step. As has been seen, bringing host data into the mix can significantly increase storage and compute requirements as well as the need for tools that can handle it all. Using a process, such as what Chris Sanders laid out for determining what types of host data is needed is foundational, so that analysts are not overwhelmed.

For Windows systems, one of the best free utilities that can generate interesting host data is Sysmon. Sysmon data, coupled with a few applications found in Security Onion (OSSEC and ELSA), can lay the groundwork for operationalizing host data. Sysmon data can be used to great effect during detection: looking for anomalous processes, certain image names, and/or hashes. During incident response, Sysmon data can be used to search for indicators of compromise or for data stacking activities.

Host data is an invaluable data set to have both in detection and incident response and with tools such as Sysmon and Security Onion, there is a much lower barrier of entry than ever before.

7. References

Bejtlich, R. (2004). *The Tao of Network Security Monitoring.* Addison-Wesley Professional.

Bejtlich, R. (2013). *The Practice of Network Security Monitoring: Understanding Incident Detection and Response* . No Starch Press.

Bianco, D. (2013, September 14). *Enterprise Security Monitoring.* Retrieved February 12, 2015, from speakerdeck.com: https://speakerdeck.com/davidjbianco/enterprise-security-monitoring

Bianco, D. (2014, January 17). *The Pyramid of Pain.* Retrieved from Enterprise Detection and Response: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

Burks, D. (2012, December 13). *A list of tools included in Security Onion...* Retrieved from Security Onion on Google Code: https://code.google.com/p/security-onion/wiki/Tools

Cloppert, M. (2009, July 22). *Security Intelligence: Introduction (pt 1).* Retrieved February 14, 2015, from SANS.org: http://digital-forensics.sans.org/blog/2009/07/22/security-intelligence-introduction-pt-1/

Gartner. (2003). *Hype Cycle for Information Security, 2003.* Gartner.

Harrell, C. (2014, December 17). *Prefetch File Meet Process Hollowing*. Retrieved from Journey Into Incident Response: http://journeyintoir.blogspot.com/2014/12/prefetch-file-meet-process-hollowing_17.html

Helweg, O. (2008, July 8). *Quick and Dirty Enterprise Eventing for Windows*. Retrieved from TechNet: http://blogs.technet.com/b/otto/archive/2008/07/08/quick-and-dirty-enterprise-eventing-for-windows.aspx

Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (n.d.). *Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains.* Retrieved February 12, 2015, from lockheedmartin.com: http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf

Know your Windows Processes or Die Trying. (2014, January 18). Retrieved February 14, 2015, from SysForensics: https://sysforensics.org/2014/01/know-your-windows-processes.html

M-Labs. (2012, November 7). *An In-Depth Look Into Data Stacking*. Retrieved February 12, 2015, from mandiant.com: https://www.mandiant.com/blog/indepth-data-stacking/

Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafò, M., . . . Steenkiste, P. (n.d.). *The Cost of the "S" in HTTPS.* Retrieved February 12, 2015, from cs.cmu.edu: http://www.cs.cmu.edu/~dnaylor/CostOfTheS.pdf

Prince, M. (2014, September 29). *Introducing Universal SSL*. Retrieved February 12, 2015, from Cloudflare.com: https://blog.cloudflare.com/introducing-universal-ssl/

Russinovich, M., & Garnier, T. (2015, January 19). *Sysmon 2.0*. Retrieved from Microsoft TechNet: https://technet.microsoft.com/en-us/sysinternals/dn798348

Sanders, C., & Smith, J. (2014). Chapter 2: Planning Data Collection. In C. Sanders, & J. Smith, *Applied Network Security Monitoring.* Waltham, MA: Syngress.

SANS. (n.d.). *Know Abnormal... Find Evil.* Retrieved February 12, 2015, from sans.org: http://digital-forensics.sans.org/media/poster_2014_find_evil.pdf

Security Intelligence: Attacking the Cyber Kill Chain. (2009, October 14). Retrieved February 14, 2015, from SANS: http://digital-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain

Tracking Processes/Malwares Using OSSEC. (2014, February 10). Retrieved February 14, 2015, from Rootshell.be: http://blog.rootshell.be/2014/02/10/tracking-processesmalwares-using-ossec/

Windows Eventchannel Example. (n.d.). Retrieved February 22, 2015, from OSSEC Docs: http://ossec-docs.readthedocs.org/en/latest/manual/monitoring/file-log-monitoring.html#windows-eventchannel-example

Appendix A

List of Resources

Resource	URL
ELSA Parsers for Sysmon logs	https://github.com/defensivedepth/Sysmon_ELSA_Parsers
OSSEC Decoder for Sysmon logs	https://github.com/defensivedepth/Sysmon_OSSEC/blob/master/Sysmon_OSSEC-Decoders.txt
Key Windows Processes Attributes	http://defensivedepth.com/windows-processes
OSSEC Rules: Process Anomalies	https://github.com/defensivedepth/Sysmon_OSSEC/blob/master/Process-Anomalies_OSSEC-Ruleset.txt
OSSEC Rules: Hash lookups	https://github.com/defensivedepth/Sysmon_OSSEC/blob/master/Alert-On-Hash_OSSEC-Ruleset.txt
OSSEC Rules: Powershell & cmd use	https://github.com/defensivedepth/Sysmon_OSSEC/blob/master/Alert-On-Image-Name_OSSEC-Ruleset.txt

Appendix B

Examples of Data Generated by Sysmon

Event ID 1: Process Create

Type	Example
Host	WIN-U93G48C7BOP
Timestamp	12/2/2014 8:26 PM
Image Name	taskhost.exe
Path to Image	C:\Windows\system32\
Process ID	1412
Process Guid	{00000000-205F-547E-0000-00100D090800}
Command line arguments	taskhost.exe U
User process is running as	WIN-U93G48C7BOP\Administrator
Logon Guid	{00000000-D448-547C-0000-0020C5460200}
Logon ID	0x246C5
Terminal Session ID	2
Integrity Level	High
Hash of binary	8570E08F5103FD0F496B1DE9ADEF6E49E237433F
Parent Process Guid	{00000000-D425-547C-0000-0010A1A40000}
Parent Process ID	736
Parent Image	svchost.exe
Parent Image Path	C:\Windows\system32\
Parent command line arguments	C:\Windows\system32\svchost.exe -k netsvcs

Event ID 3: Network Connection Detected

Type	Example
Host	WIN-U93G48C7BOP
Timestamp	12/2/2014 8:26 PM
Image Name	taskhost.exe
Path to Image	C:\Windows\system32\
Process ID	1412
Process Guid	{00000000-205F-547E-0000-00100D090800}
User process is running as	WIN-U93G48C7BOP\Administrator
Protocol	udp
Initiated (True or False)	True
Source is IPv6 (True or False)	False
Source IP	192.168.55.3
Source Hostname	WIN-U93G48C7BOP
Source Port	60352
Source Port Name	-
Destination is IPv6 (True or False)	False
Destination IP	192.168.55.101
Destination hostname	WIN-Y53G4757UET
Destination port	53
Destination port name	DNS

Appendix C

Sysmon_ELSA: Parser Patterns.xml

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<ruleset name="SYSMON" id='777'>
  <pattern>ossec_archive</pattern>
                <rules>
                        <rule provider="DefensiveDepth" class='10778' id='10778'>
                                <patterns>
            <pattern>@NUMBER::@@ESTRING::(@@ESTRING::)@ @IPv4::@->WinEvtLog
@ESTRING::(@@NUMBER::):@): @ESTRING:::@@ESTRING:::@@ESTRING:::@
@ESTRING:s0::@@ESTRING::{@@ESTRING:s1:}@@ESTRING::Image: @@ESTRING:s2:
CommandLine: @@ESTRING::User: @@ESTRING:s3: LogonGuid:@@ESTRING::Hash:
@@ESTRING:s4: @@ESTRING::ParentImage: @@ESTRING:s5: ParentCommandLine:@</pattern>
                </patterns>

                                        <examples>
                                                <example>
                                                        <test_message program="ossec_archive">2014 Dec
02 20:28:29 (10.0.15.14) 10.0.15.14->WinEvtLog 2014 Dec 02 15:26:07 WinEvtLog: Microsoft-Windows-
Sysmon/Operational: INFORMATION(1): Microsoft-Windows-Sysmon: SYSTEM: NT AUTHORITY:
WIN-U93G48C7BOP: Process Create: UtcTime: 12/2/2014 8:26 PM ProcessGuid: {00000000-205F-547E-
0000-00100D090800} ProcessId: 1412 Image: C:\Windows\system32\taskhost.exe CommandLine:
taskhost.exe U User: WIN-U93G48C7BOP\Administrator LogonGuid: {00000000-D448-547C-0000-
0020C5460200} LogonId: 0x246C5 TerminalSessionId: 2 IntegrityLevel: High HashType: SHA1 Hash:
8570E08F5103FD0F496B1DE9ADEF6E49E237433F ParentProcessGuid: {00000000-D425-547C-0000-
0010A1A40000} ParentProcessId: 736 ParentImage: C:\Windows\system32\svchost.exe
ParentCommandLine: C:\Windows\system32\svchost.exe -k netsvcs</test_message>
                                                        <!-- host-->
                                                        <test_value name="s0">WIN-
U93G48C7BOP</test_value>
                                                        <!-- processguid-->
                                                        <test_value name="s1">00000000-205F-547E-0000-
00100D090800</test_value>
                                                        <!-- image-->
                                                        <test_value
name="s2">C:\Windows\system32\taskhost.exe</test_value>
                                                        <!-- user-->
                                                        <test_value name="s3">WIN-
U93G48C7BOP\Administrator</test_value>
                                                        <!-- hash-->
                                                        <test_value
name="s4">8570E08F5103FD0F496B1DE9ADEF6E49E237433F </test_value>
                                                        <!-- parentimage-->
                                                        <test_value
name="s5">C:\Windows\system32\svchost.exe</test_value>
                                                </example>
                                        </examples>
                        </rule>

                        <rule provider="DefensiveDepth" class='10779' id='10779'>
```

```
                        <patterns>
                                <pattern>@NUMBER::@@ESTRING::(@@ESTRING::)@
@IPv4::@->WinEvtLog @ESTRING::(@@NUMBER::):@):
@ESTRING:::@@ESTRING:::@@ESTRING:::@
@ESTRING:s0::@@ESTRING::{@@ESTRING:s1:}@@ESTRING::Image: @@ESTRING:s2: User:
@@ESTRING:s3: Protocol: @@ESTRING::: @@ESTRING:s4: @@ESTRING::SourceIp:
@@ESTRING:: @@ESTRING::SourcePort: @@ESTRING:i0: @@ESTRING::DestinationIp:
@@ESTRING:s5: @@ESTRING::DestinationPort: @@ESTRING:i1: @</pattern>
                        </patterns>
                        <examples>
                                <example>
            <test_message program="ossec_archive"> 2014 Dec 02 20:28:31 (10.0.15.14) 10.0.15.14-
>WinEvtLog 2014 Dec 02 15:26:08 WinEvtLog: Microsoft-Windows-
Sysmon/Operational:INFORMATION(3): Microsoft-Windows-Sysmon: SYSTEM: NT AUTHORITY:
WIN-U93G48C7BOP: Network connection detected: UtcTime: 12/1/2014 9:03 PM ProcessGuid:
{00000000-D426-547C-0000-00103DB40000} ProcessId: 868 Image: C:\Windows\system32\svchost.exe
User: NT AUTHORITY\NETWORK SERVICE Protocol: udp Initiated: true SourceIsIpv6: true SourceIp:
a00:f0e:0:0:1822:ad8d:1e0:ffff SourceHostname: SourcePort: 60352 SourcePortName: DestinationIsIpv6:
true DestinationIp: a00:f01:7200:6500:6100:2000:4300:6f00 DestinationHostname: DestinationPort: 53
DestinationPortName: domain</test_message>
            <!-- hostname-->
            <test_value name="s0">WIN-U93G48C7BOP</test_value>
            <!-- processguid-->
            <test_value name="s1">00000000-D426-547C-0000-00103DB40000</test_value>
            <!-- image-->
            <test_value name="s2">C:\Windows\system32\svchost.exe</test_value>
            <!-- user-->
            <test_value name="s3">NT AUTHORITY\NETWORK SERVICE</test_value>
            <!-- initiated-->
            <test_value name="s4">true</test_value>
                                    <!-- sourceport-->
            <test_value name="i0">60352</test_value>
            <!-- destip-->
            <test_value name="s5"> a00:f01:7200:6500:6100:2000:4300:6f00</test_value>
                                    <!-- destport-->
            <test_value name="i1">53</test_value>
                                </example>
                        </examples>

                </rule>

        </rules>
</ruleset>
```

Appendix D

Sysmon_ELSA: Parser Schemas.sql

```
/*  v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com */

use syslog;

/*  Creates SYSMON_PROCESS Class & associated fields */
INSERT INTO classes (id, class) VALUES (10778, "SYSMON_PROCESS");

INSERT INTO fields (field, field_type, pattern_type) VALUES ("hostname","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("processguid","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("image","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("hash","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("parentimage","string", "QSTRING");

INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="hostname"), 11);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="processguid"), 12);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="image"), 13);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="user"), 14);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="hash"), 15);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_PROCESS"), (SELECT id FROM fields WHERE field="parentimage"), 16);

/*  Creates SYSMON_NETWORK Class & associated fields */
INSERT INTO classes (id, class) VALUES (10779, "SYSMON_NETWORK");

INSERT INTO fields (field, field_type, pattern_type) VALUES ("initiated","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("destip","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("sourceport","string", "QSTRING");
INSERT INTO fields (field, field_type, pattern_type) VALUES ("destport","string", "QSTRING");

INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="hostname"), 11);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="processguid"), 12);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="image"), 13);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="user"), 14);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="initiated"), 15);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="destip"), 16);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="sourceport"), 5);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id FROM classes
WHERE class="SYSMON_NETWORK"), (SELECT id FROM fields WHERE field="destport"), 6);
```

Appendix E

Sysmon_OSSEC – Decoder

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<!-- OSSEC Decoder for Sysmon Event ID 1: Process Create
  -
  - OSSEC to Sysmon Fields Mapping:
  - user = User
  - status = Image
  - url = Hash
  - extra_data = ParentImage
-->

<decoder name="sysmon-process">
<parent>windows</parent>
<type>windows</type>
<prematch>INFORMATION\(1\)</prematch>
<regex offset="after_prematch">Image: (\.*) \s*CommandLine: \.* \s*User: (\.*) \s*LogonGuid: \S*
\s*LogonId: \S* \s*TerminalSessionId: \S* \s*IntegrityLevel: \S* \s*HashType: \S* \s*Hash: (\S*)
\s*ParentProcessGuid: \S* \s*ParentProcessID: \S* \s*ParentImage: (\.*) \s*ParentCommandLine:</regex>
<order>status,user,url,data</order>
</decoder>
```

Appendix F

Sysmon_OSSEC: OSSEC Rules – Process Anomalies

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<!-- Ruleset to detect Windows Process Anomalies -
 - Uses Sysmon Event ID 1 logs & associated decoder.
 - Currently only looks at Parent Image Anomalies.
 - Windows Process Attributes documentation here: http://defensivedepth.com/windows-processes
 -
 - OSSEC to Sysmon (Event ID 1) Fields Mapping:
 - user = User
 - status = Image
 - url = Hash
 - extra_data = ParentImage
 -->

<rule id="184666" level="12">
  <if_sid>18100</if_sid>
  <status>svchost.exe</status>
  <description>Sysmon - Suspicious Process - svchost.exe</description>
</rule>

<rule id="184667" level="0">
  <if_sid>184666</if_sid>
  <extra_data>\services.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - svchost.exe</description>
</rule>

<rule id="184676" level="12">
  <if_sid>18100</if_sid>
  <status>lsm.exe</status>
  <description>Sysmon - Suspicious Process - lsm.exe</description>
</rule>

<rule id="184677" level="0">
  <if_sid>184676</if_sid>
  <extra_data>wininit.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - lsm.exe</description>
</rule>

<rule id="184678" level="12">
  <if_sid>18100</if_sid>
  <extra_data>lsm.exe</extra_data>
  <description>Sysmon - Suspicious Process - lsm.exe is a Parent Image</description>
</rule>

<rule id="184686" level="12">
  <if_sid>18100</if_sid>
  <status>csrss.exe</status>
  <description>Sysmon - Suspicious Process - csrss.exe</description>
```

```
</rule>

<rule id="184687" level="0">
  <if_sid>184686</if_sid>
  <extra_data>smss.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - csrss.exe</description>
</rule>

<rule id="184696" level="12">
  <if_sid>18100</if_sid>
  <status>lsass.exe</status>
  <description>Sysmon - Suspicious Process - lsass</description>
</rule>

<rule id="184697" level="0">
  <if_sid>184696</if_sid>
  <extra_data>wininit.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - lsass.exe</description>
</rule>

<rule id="184698" level="12">
  <if_sid>18100</if_sid>
  <extra_data>lsass.exe</extra_data>
  <description>Sysmon - Suspicious Process - lsass.exe is a Parent Image</description>
</rule>

<rule id="184706" level="12">
  <if_sid>18100</if_sid>
  <status>winlogon.exe</status>
  <description>Sysmon - Suspicious Process - winlogon.exe</description>
</rule>

<rule id="184707" level="0">
  <if_sid>184706</if_sid>
  <extra_data>smss.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - winlogon.exe</description>
</rule>

<rule id="184716" level="12">
  <if_sid>18100</if_sid>
  <status>wininit.exe</status>
  <description>Sysmon - Suspicious Process - wininit</description>
</rule>

<rule id="184717" level="0">
  <if_sid>184716</if_sid>
  <extra_data>smss.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - wininit.exe</description>
</rule>

<rule id="184726" level="12">
  <if_sid>18100</if_sid>
  <status>smss.exe</status>
  <description>Sysmon - Suspicious Process - smss.exe</description>
</rule>
```

```
<rule id="184727" level="0">
  <if_sid>184726</if_sid>
  <extra_data>system</extra_data>
  <description>Sysmon - Legitimate Parent Image - smss.exe</description>
</rule>

<rule id="184736" level="12">
  <if_sid>18100</if_sid>
  <status>taskhost.exe</status>
  <description>Sysmon - Suspicious Process - taskhost.exe</description>
</rule>

<rule id="184737" level="0">
  <if_sid>184736</if_sid>
  <extra_data>services.exe|svchost.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - taskhost.exe</description>
</rule>

<rule id="184746" level="12">
  <if_sid>18100</if_sid>
  <status>/services.exe</status>
  <description>Sysmon - Suspicious Process - services.exe</description>
</rule>

<rule id="184747" level="0">
  <if_sid>184746</if_sid>
  <extra_data>wininit.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - services.exe</description>
</rule>


<rule id="184766" level="12">
  <if_sid>18100</if_sid>
  <status>dllhost.exe</status>
  <description>Sysmon - Suspicious Process - dllhost.exe</description>
</rule>

<rule id="184767" level="0">
  <if_sid>184766</if_sid>
  <extra_data>svchost.exe|services.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - dllhost.exe</description>
</rule>

<rule id="184776" level="12">
  <if_sid>18100</if_sid>
  <status>\explorer.exe</status>
  <description>Sysmon - Suspicious Process - explorer.exe</description>
</rule>

<rule id="184777" level="0">
  <if_sid>184776</if_sid>
  <extra_data>userinit.exe</extra_data>
  <description>Sysmon - Legitimate Parent Image - explorer.exe</description>
</rule>
```

Appendix G

Sysmon_OSSEC: OSSEC Rules – Alert on Image Name

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<!-- Example Rules to detect (by image name) abnormal user behaviour -
 - Uses Sysmon Event ID 1 logs & associated decoder.
 -
 - OSSEC to Sysmon (Event ID 1) Fields Mapping:
 - user = User
 - status = Image
 - url = Hash
 - extra_data = ParentImage
 -->

<rule id="182667" level="12">
  <if_sid>18100</if_sid>
  <status>ipconfig.exe</status>
  <description>ipconfig usage</description>
</rule>


<rule id="182668" level="12">
  <if_sid>18100</if_sid>
  <status>powershell.exe</status>
  <description>powershell usage</description>
</rule>


<rule id="182669" level="12">
  <if_sid>18100</if_sid>
  <status>\cmd.exe</status>
  <description>cmd usage</description>
</rule>


<rule id="182670" level="12">
  <if_sid>18100</if_sid>
  <status>\at.exe</status>
  <description>at usage</description>
</rule>
```

Appendix H

Sysmon_OSSEC: OSSEC Rules – Alert on Hash

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<!-- Example Rule to detect (by hash) psexec usage -
  -  Uses Sysmon Event ID 1 logs & associated decoder.
  -
  -  OSSEC to Sysmon (Event ID 1) Fields Mapping:
  -  user = User
  -  status = Image
  -  url = Hash
  -  extra_data = ParentImage
  -->

<rule id="183668" level="12">
  <if_sid>18100</if_sid>
  <list field="url">lists/psexec</list>
  <description>psexec run!</description>
</rule>


<!-- SHA1 hashes for psexec releases 2011 - 2014 -
  -  Use in CDB file
  -->

cd23b7c9e0edef184930bc8e0ca2264f0608bcb3:psexec_v1.98
9a46e577206d306d9d2b2ab2f72689e4f5f38fb1:psexec_v2.00
2edeefb431663f20a36a63c853108e083f4da895:psexec_v2.10
b5c62d79eda4f7e4b60a9caa5736a3fdc2f1b27e:psexec_v2.11
```

Appendix I

Windows-Process-Tracking_ELSA: Parser Pattern.xml

```
<!-- v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com -->

<ruleset name="WIN-PROCESS-TRACKING" id='780'>
  <pattern>ossec_archive</pattern>
                <rules>
                        <rule provider="DefensiveDepth" class='10780' id='10780'>
          <patterns>
            <pattern>@NUMBER::@@ESTRING::(@@ESTRING::)@ @IPv4::@-
>WinEvtLog@ESTRING::domain: @@ESTRING:s0:: @@ESTRING::Name: @@ESTRING:s1:
Account Domain:@@ESTRING::Name: @@ESTRING:s2: Token Elevation Type: @@ESTRING:s3:
Creator@</pattern>
          </patterns>
          <examples>
                                        <example>
              <test_message program="ossec_archive"> 2015 Feb 04 22:05:18 (COLLECT02)
192.168.160.18->WinEvtLog 2015 Feb 04 17:05:14 WinEvtLog: Security: AUDIT_SUCCESS(4688):
Microsoft-Windows-Security-Auditing: (no user): no domain: DD-COLLECT-01.DD.US: A new process
has been created. Subject: Security ID: S-1-5-18 Account Name: DD-COLLECT-01$ Account Domain:
DD.US Logon ID: 0x3e7 Process Information: New Process ID: 0x68c New Process Name:
C:\Windows\System32\taskhost.exe Token Elevation Type: %%1936 Creator Process ID:
0x2d0</test_message>
              <!-- hostname-->
              <test_value name="s0">DD-COLLECT-01.DD.US</test_value>
              <!-- user-->
              <test_value name="s1">DD-COLLECT-01$</test_value>
              <!-- image-->
              <test_value name="s2">C:\Windows\System32\taskhost.exe</test_value>
              <!-- token_elevation-->
              <test_value name="s3">%%1936</test_value>
                                        </example>
                                </examples>
                        </rule>


                </rules>
</ruleset>
```

Appendix J

Windows-Process-Tracking_ELSA: Parser Schema.sql

```
/*  v. 2/8/15 --- Copyright (c) 2015 Josh Brower, Josh@DefensiveDepth.com */

use syslog;

/*  Creates WINDOWS_PROCESS Class & associated fields */
INSERT INTO classes (id, class) VALUES (10780, "WINDOWS_PROCESS");

INSERT INTO fields (field, field_type, pattern_type) VALUES ("token-elevation","string",
"QSTRING");

INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id
FROM classes WHERE class="WINDOWS_PROCESS"), (SELECT id FROM fields WHERE
field="hostname"), 11);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id
FROM classes WHERE class="WINDOWS_PROCESS"), (SELECT id FROM fields WHERE
field="user"), 12);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id
FROM classes WHERE class="WINDOWS_PROCESS"), (SELECT id FROM fields WHERE
field="image"), 13);
INSERT INTO fields_classes_map (class_id, field_id, field_order) VALUES ((SELECT id
FROM classes WHERE class="WINDOWS_PROCESS"), (SELECT id FROM fields WHERE
field="token-elevation"), 14);
```