Practical Process Analysis Automating Process Log Analysis with PowerShell

Matthew Moore

Practical Process Analysis – Automating Process Log Analysis with PowerShell

GIAC (GCFA) Gold Certification

Author: Matthew T Moore, matthew.t.moore1@gmail.com
Advisor: Robert Vandenbrink
Accepted: November 13, 2020

Abstract

Windows event log analysis is an important and often time-consuming part of endpoint forensics. Deep diving into user logins, process analysis, and PowerShell/WMI activity can take significant time, even with current tools. Additionally, while utilities exist to automatically parse out various Windows Logs, most of them do not include any native analytical functionality outside of the ability to manually filter on certain strings or event IDs. Window's native scripting solution, PowerShell, combined with Microsoft's Log Parser utility allowed for several scripts to be created with a focus on Process Creation and analysis. These scripts can detect processes spawning from unusual locations, processes that exist outside of a baseline 'Allow List', or processes that might otherwise appear to be normal, but are actually anomalous. These scripts complement other current tools such as Kape or Kansa, allowing for automated analysis of the data gathered.

1. Introduction

One of the most important elements in endpoint forensics is determining program execution-what ran, what spawned it, and what did it do? Many EDR solutions offer the capability to allow the analyst to quickly view the important details of process execution, sometimes even including a graphical representation of the process tree. However, these solutions are not foolproof, nor are they the end-all/be-all of forensics-they are a tool. A good approach to be sure, but if there is a failure of the sensor or some glitch on the server end that manages the data, then the picture it presents will not be accurate or complete. When that happens, it comes down to the fundamentals of forensics, including log analytics and artifact collection from the endpoint itself, to determine what happened.

Windows logging has changed significantly in the past 20 years. From the earliest logs on Windows NT and XP systems up to the detailed logging implemented in the modern Windows 10 and Server 2016 solutions, the Windows logs have been part of the backbone of administrative troubleshooting and forensic analysis techniques. With the volume of logs available and the space available for them, log analytics have become a time-consuming and expensive, albeit necessary, element in the forensic process. Unfortunately, when it comes to logging, the critical elements are often the tiny details that might otherwise escape notice. While some solutions offer options to automate collection and integration of these logs, the detailed examination still comes down to the analyst behind the computer and whatever tools they have available to dig through the forensic artifacts.

Enter the Windows Security logs. These logs, particularly for the forensic analyst, are critical-they detail events such as successful logins, failed logon attempts, privilege escalation, remote access, and process execution. Analysis of these logs is time and resource-intensive. Analysts can spend hours or even days going through various logs, cross-referencing findings, and pinning down leads. It can be mind-numbingly boring, tedious work, and is often compared to looking for a needle in a haystack. Various utilities can make viewing and parsing the logs easier, such as Event Log Explorer (FSPro Labs, 2020) or Microsoft's Log Parser utility. However, while they make going through the logs somewhat easier, and offer some filtering/searching capability, they do

not have any native functionality to analyze the data. There is nothing in them to pick up on discrepancies such as multiple LSASS instances, or a process named SCVHost.exe. There is no option to easily filter out the vast amount of normal activity, to make searching easier-baselining known processes to allow them to be filtered out, showing only the anomalous ones.

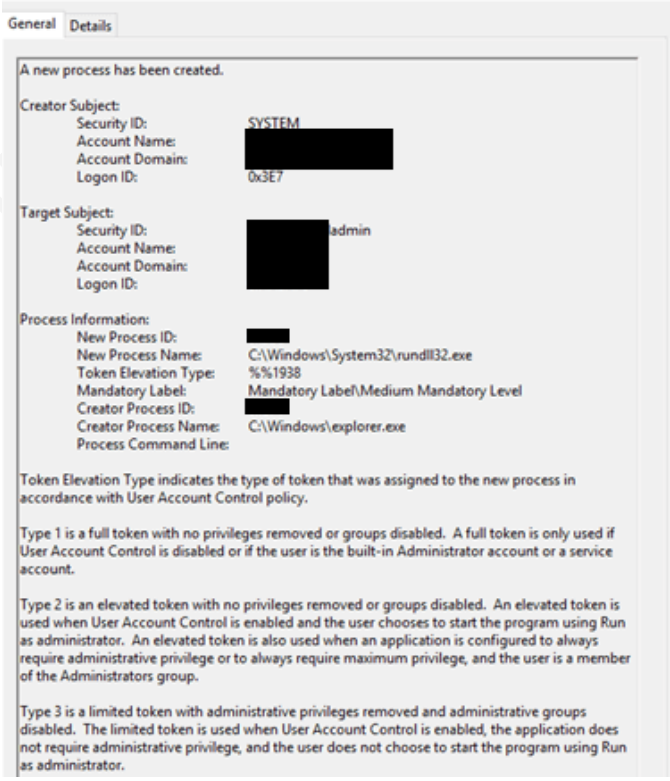
These drawbacks, and the time involved in manually filtering log events through various tools, led to the development of the analysis script provided in Appendix B. Due to the ease of use and native functionality of PowerShell within the Windows environment, the analysis scripts were constructed in PowerShell. This makes them portable, easily customizable, and relatively easy to understand and deconstruct if necessary. They are modular and allow the analyst the freedom of picking and choosing what to look for and what to throw away, along with providing several automated analysis capabilities. Lastly, they easily function with other utilities, such as Log Parser. This allows the analyst to pull out just what they need from the logs, saving time and processing power, as less data needs to be parsed for the analytical stages.

2. Process Analysis

Process Analysis is a multi-faceted field that encompasses several different areas of forensics. The study of malware analysis, for example, is in large part a study of process analysis, but in much greater depth and detail than is encountered by the average forensic examiner. Knowing what a process is capable of – elements such as network connections, sockets, creating other processes, or even injecting code into another process – is one of the key focuses of analyzing malicious software. Using those same discovered capabilities and data about the process itself allows the malware analyst to generate indicators of compromise (IOCs) for the malicious code, allowing it to be more easily detected and blocked elsewhere. Memory forensics investigates the processes as they were running at the time the memory sample was generated-giving a snapshot of the process at runtime, and an opportunity to look for anomalous activity that might not ordinarily be seen in logs. Network forensics can be combined with either memory forensics or malware analysis to track down the source of anomalous/malicious network activity. For endpoint forensics, knowing if the program ran and what other processes are

related to it are vital. While this information has some redundancy in other elements such as the Windows Registry and Prefetch, the Windows Event logs are a critical tool in determining evidence of execution and investigating how the process relates to the environment around it.

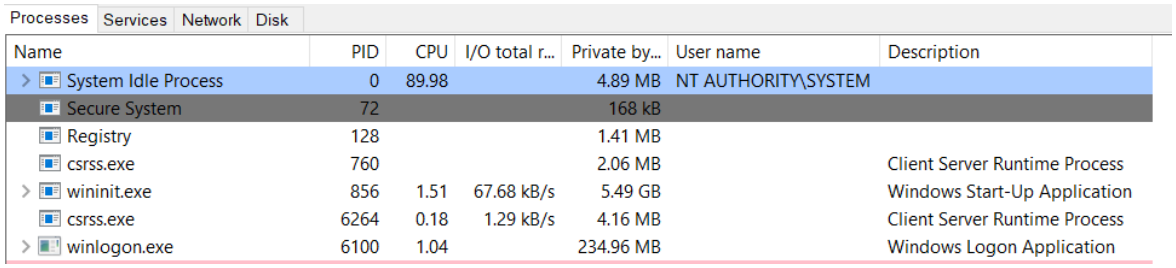
In endpoint forensics, process analysis starts by way of the Process Creation event, Event ID 4688 (EID 592 in Windows 2003 and below), found in the Windows Security log. This event, shown below, provides the analyst with detailed information about the process being executed: the path for the executable, who ran it, their domain, what process spawned it, when it happened, and if command-line auditing is enabled, what the command line was for the process when it ran. Additional details are also available denoting the UAC status of the process when run, as well as a new feature in Windows 10, the Mandatory Label. The Mandatory Label lists the integrity of the process based on the user integrity level & file integrity level for the process being run. (Smith, n.d.)



Time is always one of, if not the most, critical elements in a log. The time is listed as a primary field in the security log itself, rather than as a sub-field in the event description. By default, most event viewers will show the collected log in whichever time zone the analyst's machine is set to. For the native Windows Event Viewer, the time matches whatever the local machine is set to. Depending on the method of extracting the logs, the analyst may or may not have to take time-zone differences into account during analysis. Log Parser treats the logs the same as the native Windows Event Viewer, so the timestamps from there will be based on the local machine's time zone by default.

The fields in the event description start with the *Subject* info – their SID, account name, account domain/machine name, and the logon ID. All of these are good to know- the logon ID can be useful in tying the process creation event back to a particular logon session. Prior to Windows 10/2016, this field was simply 'Subject'. However, in Windows 10/Server2016, an additional field for *Target Subject* was added, which reflects the user information when a process is started under a different user account, such as when the application is started using RunAs. As such, the original *Subject* field was renamed to *Creator Subject*. By default, these two fields will show the same data, unless the process is run as a different account. (Dansimp, 2017)

The Process information section covers the details around the process execution itself-process ID, executable path, token elevation type, mandatory label, creator process ID, creator process path, and the process command line. The Process ID (PID) and Creator Process ID (CPID, sometimes called 'PPID' for Parent Process ID) are hex strings denoting the ID number of the process and the parent process that spawned it respectively. While they are encoded in the raw event as hex, they can easily be decoded to decimal, to compare against what may be seen in task manager or another utility like Process Hacker (see screenshot below).



Some EDR solutions, as well as various Microsoft & third-party tools, will include the process ID in the list of information provided to the analyst. The Process ID/Creator Process ID help form the links that create the chain of processes. This doubly-linked list is part of the Windows Executive Process data structure. Rootkits often hide by removing these links, leaving the process invisible to most analytical tools outside of memory analysis. (Yosifovich et al., 2017, p. 106) Tracing these links when available allows the analyst to track the activity back to an initial source-for example, Outlook spawning WinZip, which spawns WScript, which runs a malicious executable.

While some context can be gained by looking at the creator process path, more than one executable may be running with that path. The PID and CPID are critical in differentiating them. Additionally, the creator process path is not always visible in the event description depending on how the event logging is configured, so it is best not to become dependent on it.

The Token Elevation Type indicates what the UAC policy was in place on the process when it was run. Type 1 indicates that UAC was disabled, or the user was the built-in administrator account or a service account. Type 2 indicates that UAC is enabled, but the user runs the program as Administrator. This can also be if the program is configured to run as administrator by default. Type 3 indicates that UAC is enabled, but the user is not elevating privileges when running the program. (Smith, n.d.) The Mandatory Label, as previously mentioned, deals with the correlation between the user integrity level and the file integrity level, and is rated Low, Medium, or High, though newer versions include an SID reference to a more precise rating. (Holiu et al., 2018)

Lastly, the command line shows any arguments and switches that the process was created with. A note of caution – the command line only shows the *initial* command line from when the process was opened. For any process with shell capabilities, such as CMD.exe or PowerShell, any commands typed *after* the initial opening of the process will not show up in the command line. In some cases, as is often seen with CMD, the other commands will spawn their own sub-processes. These will show up as additional 4688 events, each with their own command lines. However, when analyzing other tools

such as PowerShell, often the only way to determine what happened is through other logs (such as PowerShell script block logs), or by memory analysis. (Dunwoody, 2016)

2.1. Process Baselining

Baselining known processes allows the analyst to filter out the bulk of 'known' process activity, allowing them to focus on unusual/out-of-place processes that might not raise an alarm at first glance. It can save hours of search and analysis time, letting the analyst put their time into where it's most useful. This is not to say baselining is a be-all/end-all. Comparing items against a baseline 'allow list' can cause the analyst to miss important details from 'known' processes, such as PowerShell, if that process is ignored due to being on the allow list. That is why the baseline section is a *module* in a larger tool, not the entirety of the tool itself. Items that might otherwise be more easily missed (scvhost running in system32, or exp1orer.exe, for example) are much easier to catch when a baseline of known common process locations exists to compare against.

Baselining is specific to one's own environment-a baseline for process activity in a corporate environment will look very different from a home user's environment. A baseline will vary greatly from company to company, sometimes even within companies. It is a good idea to ensure that a baseline is created for known processes & locations for every environment the analyst is expected to interact with. Generally, this is most easily done by getting a baseline from the various gold images used in the environment-ideally one for each gold image in use. This provides a variety of baselines to compare against, depending on which type of system is being analyzed.

There are multiple ways of creating this baseline. A PowerShell script can be run to recursively pull the names/locations of all executable files. A check can be run against the MFT, another reason why pulling a baseline forensic collection against a known good system is always a good idea-know what is normal in the environment. The baseline used with the script provided here was a CSV with the column header 'process'. This can be edited in the script if needed.

Once the baseline is established, the analyst can compare it against the list of processes that ran on the suspect system to pick out any anomalies. Tricks like slight mis-spellings of common processes or using the number 1 to replace the letter L, as was

shown above, which might normally be easily missed when perusing the process list, stand out distinctly when compared against a baseline.

2.2. 'Legitimate Malware'

Malicious software is generally defined as any software that is used for a nefarious purpose. This includes programs that are designed to alter or destroy data, network communications, or even whole systems to achieve the attacker's end goals. However, many otherwise 'legitimate' utilities can also fall under this category, allowing the attacker to 'live off the land'. Keeping track of these utilities and their actions is a good course of action, one that starts with knowing how and when they are used in your environment – *knowing what is normal helps find evil*. (Lee & Pilkington, 2018)

The same baseline that works to establish the allow list is also effective here-particularly regarding pattern-matching. Whereas the allow list defines what processes exist where, another useful utility of the baseline is that it establishes patterns in the process behavior. If it is normal in the analyst's environment to have PowerShell calling UpdateCompanyScript.ps1, then the analyst can check and confirm that it is legitimate activity, and not have cause for alarm when they see it on a suspect machine, or even tune the alerting in the script so that it is not as noisy. However, knowing that the organization has turned off a certain utility-the analyst seeing that utility in use should raise a red flag. Additionally, this ability to perform pattern matching allows the analyst to take intelligence on an adversary's activities post-exploitation, and search for them using the script.

In the provided script, several generic utilities have been added as alert flags. Be aware that the analyst will want to confirm what is and is not normal in their environment, and tune the flags accordingly, or even add more of their own. An additional consideration the analyst may wish to consider when tuning is that while an activity may be normal in their environment (for example, the use of Windows Remote Desktop), it can still be utilized by an adversary and so they may still wish to make note of any processes performing that activity.

The elements that this script currently flags are almost all native Windows utilities, due to the volume of third-party utilities available. The analyst is encouraged to

add utilities to this list as they see fit. The list includes shell utilities, such as CMD.exe and PowerShell; script interpreters Cscript, Wscript, and MShta; post exploit commands such as net, netsh, whoami, tasklist, and systeminfo; WMI related tools such as mofcomp, wmic, winmgmt, wmiprvse, and scrcons.exe; and remote access/execution utilities like mstsc, psexec, wsmprovhost, & ssh. (Satran, 2018)

2.3. Processes Running from Unusual Locations

One of the quickest ways to determine anomalous processes is to look at where they are running *from*. For the most part, executables in Windows systems are run from either the Windows System32 or SysWOW64 (occasionally also from the WinSXS folder), or from the two Program Files directories. Unless the organization has blocked it, processes can often be seen running from the user's AppData folder. These should be noted, however, as should programs run directly out of the user's Downloads folder. While not immediately suspicious, these locations give context to the actions performed and require more scrutiny. Additionally, applications running out of the drive root (C:\) or user's Documents folder should be noted as suspicious. The analysis script first looks for anything not running from System32/SysWOW64, or one of the Program Files directories. It then looks through those hits and notes any that are from the previously mentioned locations and adds them to a report for the analyst to investigate further.

2.4. Find Evil Poster

Through SANS, Rob Lee and Mike Pilkington have created an excellent resource to help analysts pick out abnormal activity from the most common Windows processes. The behaviors and patterns noted in the poster provided as part of the FOR508 course are common across most Windows platforms in use today and enable the analyst to recognize anomalous behavior from otherwise seemingly benign processes. The analysis script incorporates some of these patterns to automate detection for anomalies-for example, LSASS spawning from Explorer. (Lee & Pilkington, 2018)

2.5. Process Chain & Searching

One of the key elements in process analysis is knowing what process spawned the suspicious one, and what child processes it spawned in turn. This allows the analyst to

track activity back to an originating process and determine what other actions may have been taken by the process during/after execution. The ability to search for all processes created by a specific user, or all processes of a certain name or elevation type is also helpful to the investigator. It provides context around what actions a user took and what abilities those processes may have had through privilege escalation, in addition to being able to catalog all instances of a specific process.

The analysis script offers the option to search the ingested logs for a process by process ID, process name, username, and elevation token. The PID search can look for parent or child relationships, investigating what the process chain was that spawned it, as well as determining what child processes it spawned.

3. Automating the Process

3.1. Log Parser

Microsoft's Log Parser utility (https://www.microsoft.com/en-us/download/details.aspx?id=24659) is a command-line tool that allows the analyst to parse Windows Event logs, Registry data, Active Directory information, or multiple other data sources using SQL queries. It can also query various data formats, such as CSV and XML files. The utility can query for specific events and fields within the log files, extracting just the information required. (Allen, 2010) As the tool is command-line, it is perfectly suited to work with PowerShell to automate log parsing.

While the focus of this project has been on process analysis, it would be simple to adapt the Log Parser utility and PowerShell scripting to retrieve and analyze other events of interest from the Windows Event logs or any other compatible data source. The ability to search by fields also helps improve the analysis performance, allowing Log Parser to pre-filter the events collected before the in-depth analysis process begins. The command structure, including the various switches and options for interacting with LogParser, is included in Appendix D along with several examples. Additionally, Appendix E covers the dependencies for the script, including where to locate the LogParser executable so that the script can reference it.

3.2. PowerShell

PowerShell is Window's native scripting & automation utility. It comes installed by default on all current Windows platforms and has capabilities for interfacing with just about every element of the Windows environment. It can be used to perform remote configuration, general task automation, installation scripting, or just about any task the user can come up with. Rather than the structured command interface utilized by the Windows command prompt (cmd.exe), PowerShell utilizes cmdlets. These are either built-in or can be created and defined by the user. It utilizes a verb-noun command pattern which makes learning the scripting language intuitive and relatively straightforward. It also allows for interfacing with the .NET framework, allowing the analyst to create GUI interfaces for the scripts and making their tools more user-friendly.

Due to the ability to work with the command line as well, PowerShell allows for a much more flexible opportunity to incorporate external tools, such as MFTdump or LogParser. This can greatly speed up analysis, as rather than having to script everything for PowerShell to run, it can off-load some of that work to an external utility, saving the user valuable time. It has excellent flexibility in working with various data formats, including XML, HTML, and CSV, allowing the user to take in data from one utility, analyze it, and format it in an easy to read report. It can interface with Microsoft Office utilities as well, including formatting data within the document/spreadsheet, and running embedded tools such as pivot tables. (Bertram, 2020)

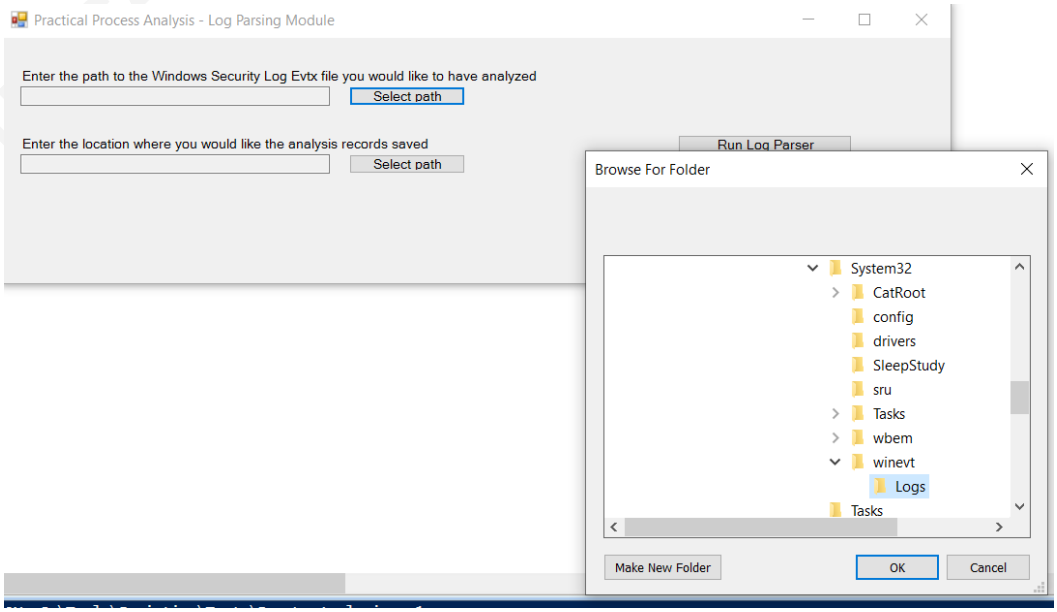
In terms of analysis capability, it is limited only by what the user can code. Any familiarity with basic coding allows the user to create modules and functions to perform complex and in-depth analysis, which when done manually, might take several hours or even days to perform, all trimmed down to minutes or seconds when automated in this manner. Collection scripts can be set to run across environments that feed into analysis scripts, which pear down to just the important data. The script provided is just scratching the surface of what can be done-it gets the job done, but there is always room to improve and expand the capabilities, and the analyst is encouraged to do so.

4. Usage

This script is designed to assist the user by automating some of the more tedious aspects of Windows log analysis. The focus here is on Process Log analysis, but the model can be extended to analyze any data covered by Windows Event Logs. User logins, anti-virus alerts, or even non-security related events such as application errors can all be collected and analyzed in this way.

4.1. Performing Analysis

The script GUI is broken down into two primary screens. The first screen, shown below, is where the user will enter the path to the Windows security log file, and the location where they would like the analysis documents saved. This allows the user to specify the location of the log file, such as when analyzing a collection of artifacts from another machine. Otherwise, PowerShell's native log analysis capabilities default to the machine's own event log files. The button to the right triggers the Log Parsing process and opens the analysis window.

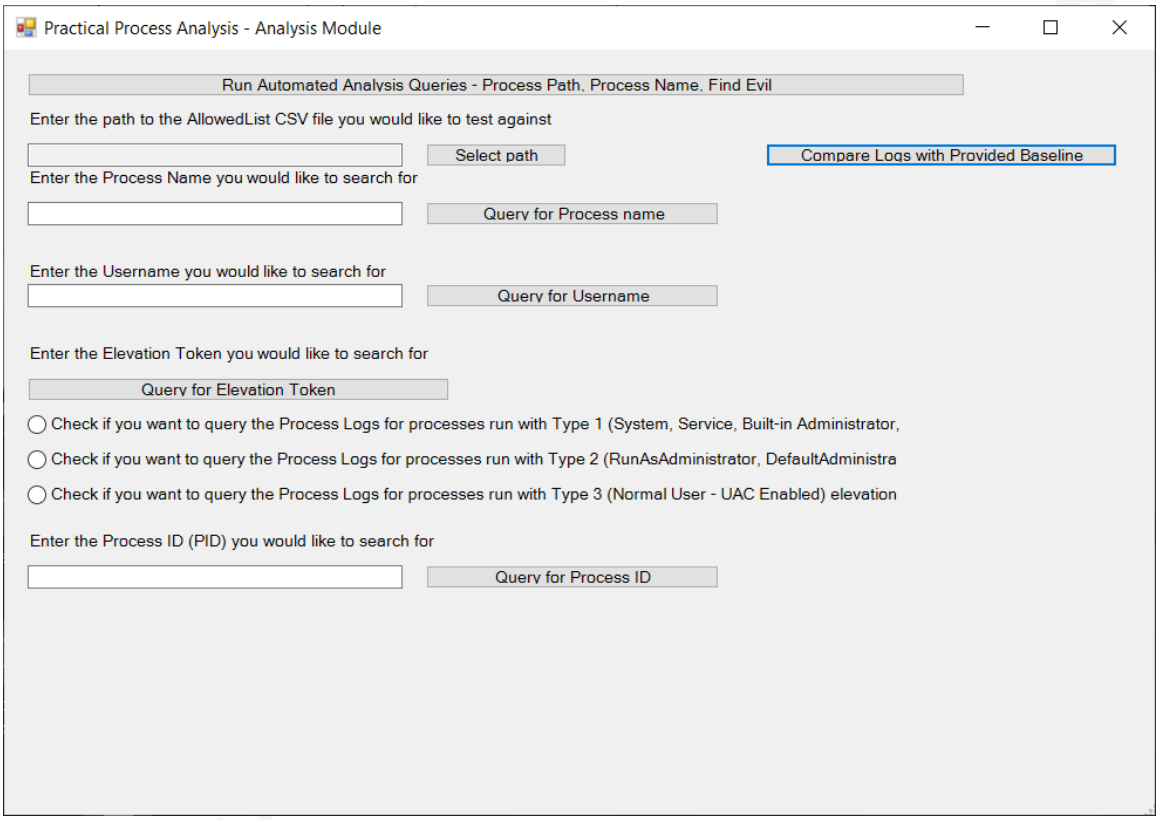


The parsing process begins by pulling the Windows Security log and searching for all events with Event ID (EID) 4688 – New Process Created. It then pulls out certain fields from the event logs and tabulates them all into a CSV file. The

process ID's are then converted from Hex to Decimal, and an index value for each line is added to allow for better search functionality. This final CSV, 'ProcLogs_Indexed.csv', is the one that is parsed and queried by the analysis module. The other CSV's are kept as well for error checking and redundancy but can easily be removed if desired.

The analysis window, shown below, offers several options to the user. First and foremost is the Automated Analysis queries. This runs the check for processes running from unusual locations, processes that could be used as 'legitimate malware', and runs the check for any process activity that would violate the 'known normal' activity in the SANS 'Find Evil' poster, such as LSASS running from Explorer.exe. This is where all the set, standard queries that do not require any user interaction occur. The output for these hits is recorded in a series of .txt files, which display the process name & location, the process ID, parent process ID, the username, and elevation rights. An example of these output files is shown in Appendix C. The TXT files are further broken down by what triggered the hit – for instance, while all anomalous location hits will be recorded under the AnomalousHits.txt file, there are also sub-files for programs running from Downloads, AppData, etc. This is true across all of the automated searches- processes such as whoami, netsh, or systeminfo would show up in the AnomalousHits-Post Exploit Commands.txt file.

Next is the Allow List option. This allows the user to make use of a baseline of known file locations on their systems to validate the process list against. This can detect some of the more elaborate attempts to bypass scrutiny - items such as scvhost.exe or 1sass.exe that might be missed on a first or second glance stand out like a sore thumb here. The files that don't match the allow list show up in AllowHits.txt.



Following the Allow list is the Process Queries – these allow the analyst to search the list of processes for process name, username, elevation token, and process ID. Each of these is saved as a CSV file with the search query in the filename - "searchquery_results.csv". They will include a list of all processes that match the search query, along with PID, CPID, username, and elevation rights.

4.2. Customizing the Modules

One of the most useful elements of using a script as opposed to a fully compiled executable is the ability to edit the script on the fly, without having to recompile all the code. In this case, it is particularly useful because the threat environment is always changing, and the focus of an investigation can vary from case to case. Rather than having to maintain multiple versions of the script or program for every conceivable instance, a broad-based script allows the analyst to quickly change what data they're returning, and what activity they're alerting on making for a much more robust tool.

Matthew T Moore, matthew.t.moore1@gmail.com

In the case of the script here, while knowing some PowerShell coding can be helpful, customizing the alerting sections and data that the user wants back is relatively straightforward. The functions for the automated analysis consist of 'Analyze_Proc_Path', 'Analyze_Proc_Name', and 'Analyze_Find_Evil'. These consist of a string search that looks for details in the process name and path, which can be edited by changing the string at the end of the 'If Process -like' statement. For example, the current query for wmic.exe looks like this:

```
If($_.Process -like "*wmic.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Wmic.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
```

The user could change "*wmic.exe" to "*evil.exe" and the query would then alert on all processes named evil.exe, or with that string at the end of their process name (eg: devil.exe would also alert), and provide the process name, date & time, PID, CPID, username, and elevation rights. If the user wanted to maintain the alert for WMIC, and add another alert for evil.exe, they could copy the line and make the edit to the copy, preserving the original as its own alert.

Additionally, if the user does not want the full listing of information currently provided with each query, they can remove or edit the relevant parts of the -Value string. Or, if they want additional details, they can be added based on the variable names from the original log parsing function. For example, if the analyst's organization has command-line auditing in place, they could include the following string inside the double quotation marks to add in the command line:

"Its command line is $($_.Cmdline)"

The user can also customize where they want these results stored. In the example above, the hits for wmic are stored in a file called 'AnomalousHits-Windows Management.txt'. This string could be altered to be whatever the user wants, and the user could combine alert hits however he or she desires, just by changing the string for the output file. All of the above also applies to the Search-

based functions: Search_Process, Search_User, Search_Elevation, and Search_PID_Child.

Another useful area where the user may want to edit is the path to the Allowed List. Rather than constantly selecting where this list is located every time the user runs the tool, they can place a static path to the list in the variable and remove the text box if they so prefer (this requires a bit more skill with PowerShell, but is not overly complicated).

```
$AllowList = $AllowPath_TextBox.Text
```

Changing this variable to a set path is simple. Remove the string after the equals (=) sign and replace it with the static path in double-quotes:

```
$AllowList = “C:\Path\to\allowed\list\list.csv”
```

The following section of code could then be commented out using a <# at the start of the code, and a #> at the end of it, as is seen in the title block for the code:

```
$AllowPath_label = [Label] @{
    Text = "Enter the path to the AllowedList CSV file you would like to test
against"
    Location = '20,50'
    Size = '600,20'
    }

$AllowPath_TextBox = [TextBox] @{
    Location = '20,80'
    Size = '320,20'
    Readonly = $true
    Text = $null
    }

$Allow_Select_Path_Button = [Button] @{
    Location = '360,80'
    Size = '120,20'
    Text = 'Select path'
    }
```

Adjustments like those mentioned above can save time and allow the analyst to focus on what data they need, and not be distracted by extraneous information.

5. Conclusion

With the ever-increasing amount of data that DFIR analysts need to sift through, any attempts at automation are to be grasped firmly and constantly improved upon. Those that can cut down the time on tedious and time-consuming tasks are particularly helpful. Every analyst should have a stake in automation and be willing to contribute in whatever manner they can. Being involved in the automation process in turn can help the analyst improve their skillsets and encourages them to constantly expand their knowledge.

Process Analysis is a time-intensive job, and with the utilization of Windows native scripting through PowerShell and a Microsoft log parsing utility, some of that time can be saved to allow the analyst to focus on the key elements. Knowing the environment they operate in and the patterns that comprise what is 'normal' is of critical importance to the analyst, and allows them to make more accurate, timely decisions. Additionally, while automation can be done in many ways, with multiple resources and programming /scripting languages, utilizing native tools such as PowerShell allows for greater ease of use and accessibility for a wider population than more specialized code and utilities. The flexibility inherent with PowerShell lends itself well to the analytical process across multiple domains, allowing for automation of not only the Windows Event logs, but of other forensic artifacts as well, potentially allowing multiple analysis tasks to be handled by a single script.

Combining the two elements together-analysts who are familiar with their environment with a focus on automation, allows them to improve accuracy and cut down on time needed for analysis. This in turn leads to a much stronger security posture overall for an organization, with sharper, more involved analysts who feel they can make a tangible difference. The script included in Appendix B is an excellent starting point for Process Analysis, but it should not be considered complete. The analyst reading this is encouraged to take this script, to customize it to match their environment, and add features & definitions to it to make it their own.

Matthew T Moore, matthew.t.moore1@gmail.com

References

Allen, J. (2010, May 24). *Using LogParser - Part 1 - Simple talk*. Redgate Hub. https://www.red-gate.com/simple-talk/blogs/using-logparser-part-1/

Bertram, A. (2020). *PowerShell for sysadmins: Workflow automation made easy*. No Starch Press.

Dansimp. (2017, April 19). *4688(S) a new process has been created. (Windows 10) - Windows security*. Technical documentation, API, and code examples | Microsoft Docs. https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4688

Download log parser 2.2 from official Microsoft download center. (2005, April 20). Microsoft. https://www.microsoft.com/en-us/download/details.aspx?id=24659

Dunwoody, M. (2016, February 11). *Greater visibility through PowerShell logging*. FireEye. https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html

FSPro Labs. (2020, May 2). *Event Log Explorer*. Windows event log analysis software, view and monitor system, application and security event logs — FSPro Labs. https://eventlogxp.com/

Holiu, Batchelor, D., Satran, M., & Jacobs, M. (2018, May 31). *Mandatory integrity control*. Technical documentation, API, and code examples | Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control?redirectedfrom=MSDN

Lee, R., & Pilkington, M. (2018). *SANS Hunt Evil Poster*. Digital Forensics Training | Incident Response Training | SANS. https://digital-forensics.sans.org/media/SANS_Poster_2018_Hunt_Evil_FINAL.pdf

Matthew T Moore, matthew.t.moore1@gmail.com

Mandia, K., Pepe, M., & Luttgens, J. (2014). *Incident response & computer forensics* (3rd ed.). McGraw-Hill Education.

Payette, B., & Siddaway, R. (2018). *Windows PowerShell in action* (3rd ed.). Manning Publications Co.

Satran, M. (2018, May 31). *WMI command-line tools*. Technical documentation, API, and code examples | Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-command-line-tools

Smith, R. F. (n.d.). *Windows security log event ID 4688 - A new process has been created*. Randy Franklin Smith's Ultimate Windows Security. https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4688

Yosifovich, P., Solomon, D. A., & Ionescu, A. (2017). *Windows internals, Part 1: System architecture, processes, threads, memory management, and more* (7th ed.). Microsoft Press.

Appendix A
Code Repository on Github

For up to date modules, features, and bug fixes, the analysis script can be found here:

https://github.com/malwarematt/PracticalProcessAnalysis

Feel free to contribute ideas, suggestions, or even code & modules to better support and improve forensics analysis capabilities.

Matthew T Moore, matthew.t.moore1@gmail.com

Appendix B
Process Analysis Modules

```
<# - Practical Process Analysis

ProcLogAnalysis.PS1
Created by: Matthew T Moore - GREM, GCFA, GCIA
Email: matthew.t.moore1@gmail.com

Description: This script is intended to accept Windows Security Event Logs
(Security.evtx) and parse them for Process Creation Events. It will then
analyze the logs for anomalous processes through several methods.
It also includes a utility to search through the process logs to find
particular strings or process ID's (PID).

The script relies on the presence of the log parser utility in a subfolder of
where the PowerShell script is located, called bin. The AllowList should be
formatted with the column header 'process', or can be edited in line 94.

Script Accompanies GIAC GCFA Gold Paper:
Practical Process Analysis - Automating Process Log Analysis with PowerShell

Advisor: Robert Vandenbrink
Submitted: November 13, 2020

#>


using assembly System.Windows.Forms
using namespace System.Windows.Forms

$Global:OutputPath=$null


Function Parse-Logs ($LogPath , $OutputPath) {

Write-host "Log path is $LogPath and Output Path is $OutputPath"

# - $secLogPath = Read-Host -Prompt "Enter the full path of the Windows
Security Log Evtx file you would like to have analyzed including filename"

$secLogPath = [System.String]::Concat($LogPath,"\Security.evtx")

# - $outputPath = Read-Host -Prompt "Enter the location where you would like
the analysis records saved"

$OutputPathFile = [System.String]::Concat($OutputPath,"\ProcLogs_Raw.csv")
$DecCsv = [System.String]::Concat($outputpath,"\ProcLogs_Dec.csv")
$IndexedCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")

# - Initial Log Parser Utility - pulls out event 4688 fields and passes them
into CSV file
$LPcmd= [System.String]::Concat($PSScriptRoot,"\bin\LogParser.exe")
$LPcmd_Proc__Select="Select TimeGenerated AS Date, EXTRACT_TOKEN(Strings,0,
'|') AS CS_SID, EXTRACT_TOKEN(Strings,1, '|') AS CS_Username,
EXTRACT_TOKEN(Strings,2, '|') AS CS_Domain, EXTRACT_TOKEN(Strings,3, '|') AS
CS_LogonID, EXTRACT_TOKEN(Strings,4, '|') AS PID, EXTRACT_TOKEN(Strings,5, '|')
AS Process, EXTRACT_TOKEN(Strings,6, '|') AS Elevation,
EXTRACT_TOKEN(Strings,7, '|') AS CPID, EXTRACT_TOKEN(Strings,8, '|') AS
CmdLine, EXTRACT_TOKEN(Strings,9, '|') AS TS_SID, EXTRACT_TOKEN(Strings,10,
'|') AS TS_Username, EXTRACT_TOKEN(Strings,11, '|') AS TS_Domain,
EXTRACT_TOKEN(Strings,12, '|') AS TS_LogonID, EXTRACT_TOKEN(Strings,13, '|') AS
CreatorProcess, EXTRACT_TOKEN(Strings,14, '|') AS MandatoryLabel FROM "
$LPcmd_Proc__Where=" WHERE EventID = 4688"
$LPcmd_Proc_=[System.String]::Concat($LPcmd_Proc__Select,"'",$secLogPath,"'",$L
Pcmd_Proc__Where)
```

```
& $LPcmd -stats:OFF -i:EVT -q:ON $LPcmd_Proc_  -o:csv -headers:ON | out-file
$OutputPathFile

# - Section to convert PID's and CPID's from Hex to Dec
$hexcsv = Import-Csv $OutputPathFile | ForEach-Object {

$hexpid = $($_.PID)
$hexCPID = $($_.CPID)

$decpid = [convert]::toint64($hexpid,16)
$decCpid = [convert]::toint64($hexCPID,16)

$_.PID = $decpid
$_.CPID = $decCpid

$_

} | ConvertTo-Csv -NoTypeInformation | % {$_.Replace('"','')} | Out-File
$DecCsv

# - Section to add index value to each row in CSV
$Incsv = Import-CSV $DecCsv | Select *,Index | ForEach-Object -Begin { $Line =
1 } {
    $_.Index = $Line++
    $_

} | ConvertTo-Csv -NoTypeInformation | % {$_.Replace('"','')} | Out-File
$IndexedCsv


$PA_form.ShowDialog()
}

Function Analyze-AllowedList($OutputPath , $AllowList){
Write-host "Output Path is $outputpath"
$AllowTXT = New-Item -Path $OutputPath -Name "AllowHits.txt" -ItemType "file" -
Value 'Hits for Processes Not on Allowed List' -Force
$allowtxtpath = "$OutputPath\AllowHits.txt"
# - Path to AllowedList:
$AllowCSV = import-csv $AllowList
# - Make sure Header in Allow CSV file is using the string 'process' for the
process column

# - Path to Testlist
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")
$proclist = [System.String]::Concat($outputpath,"\ProcList.csv")
$trimlist = [System.String]::Concat($outputpath,"\ProcTrim.csv")
import-csv $InputCsv | select -ExpandProperty Process | Out-File $proclist

Import-csv $proclist -Header "Process" | sort Process -Unique | ConvertTo-Csv -
NoTypeInformation | % {$_.Replace('"','')} | Out-File $trimlist

$testlist = import-csv $trimlist

foreach($i in $testlist){
    $Query = $($i.Process)
    if($AllowCSV.process -like $Query) {
        #Write-Host "$Query on Allow list"
        }Else {
            #Write-host "$query not on Allow List"
            Add-content -path $AllowTXTpath -Value "$Query not on Allowed List"
            }
        }
Write-Host "AllowedList Comparison Complete - Check AllowHits.txt for any
flagged processes"

# Clean-up - remove files for less clutter, left for testing purposes/initial
script validation
# To automatically remove files, delete the comment character (#) at the start
of the next 2 lines
# Remove-item -Path $proclist
# Remove-item -Path $trimlist
```

Matthew T Moore, matthew.t.moore1@gmail.com

```
}

Function Analyze_Proc_Path($OutputPath){
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")

Import-Csv $InputCsv | where {$_.Process -notlike "C:\program*" -and $_.Process
-notlike "C:\Windows\system32\*" -and $_.Process -notlike
"C:\Windows\SysWOW64\*" -and $_.Process -notlike "C:\Windows\explorer.exe"} |
foreach-object {

    Add-content -path "$OutputPath\AnomalousHits.txt" -Value "Anomalous
Location hit found - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and
CPID: $($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "

    If($_.Process -like "*appdata\*") { Add-content -path
"$OutputPath\AnomalousHits-AppData.txt" -Value "Process running from AppData -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
    If($_.Process -like "*downloads\*") { Add-content -path
"$OutputPath\AnomalousHits-Downloads.txt" -Value "Process running from
Downloads - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
    If($_.Process -like "*desktop\*") { Add-content -path
"$OutputPath\AnomalousHits-Desktop.txt" -Value "Process running from Desktop -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
    If($_.Process -like "*documents\*") { Add-content -path
"$OutputPath\AnomalousHits-Documents.txt" -Value "Process running from
Documents - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
    }
}

Function Analyze_Proc_Name($OutputPath){
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")

Import-Csv $InputCsv | foreach-object {
#Shells
If($_.Process -like "*PowerShell.exe") { Add-content -path
"$OutputPath\AnomalousHits-Shells.txt" -Value "PowerShell.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*cmd.exe") { Add-content -path "$OutputPath\AnomalousHits-
Shells.txt" -Value "CMD.exe running - $($_.Process) ran at $($_.Date) with PID:
$($_.PID) and CPID: $($_.CPID), created by user: $($_.CS_Username). Its
elevation rights are $($_.Elevation). Its index value is $($_.Index) "}

#Script Interpreters
If($_.Process -like "*cscript.exe") { Add-content -path
"$OutputPath\AnomalousHits-Script Interpreters.txt" -Value "Cscript.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*wscript.exe") { Add-content -path
"$OutputPath\AnomalousHits-Script Interpreters.txt" -Value "Wscript.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*mshta.exe") { Add-content -path
"$OutputPath\AnomalousHits-Script Interpreters.txt" -Value "MShta.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}

#Post Exploit Commands
```

```
If($_.Process -like "*net.exe") { Add-content -path "$OutputPath\AnomalousHits-
Post Exploit Commands.txt" -Value "Net.exe running - $($_.Process) ran at
$($_.Date) with PID: $($_.PID) and CPID: $($_.CPID), created by user:
$($_.CS_Username). Its elevation rights are $($_.Elevation). Its index value is
$($_.Index) "}
If($_.Process -like "*netsh.exe") { Add-content -path
"$OutputPath\AnomalousHits-Post Exploit Commands.txt" -Value "Netsh.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*whoami.exe") { Add-content -path
"$OutputPath\AnomalousHits-Post Exploit Commands.txt" -Value "Whoami.exe
running - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "*tasklist.exe") { Add-content -path
"$OutputPath\AnomalousHits-Post Exploit Commands.txt" -Value "Tasklist.exe
running - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "*systeminfo.exe") { Add-content -path
"$OutputPath\AnomalousHits-Post Exploit Commands.txt" -Value "Systeminfo.exe
running - $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}

#Windows Management/WMI Tools
If($_.Process -like "*mofcomp.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Mofcomp.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*winmgmt.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Winmgmt.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*wmic.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Wmic.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*wmiprvse.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Wmiprvse.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*scrcons.exe") { Add-content -path
"$OutputPath\AnomalousHits-Windows Management.txt" -Value "Scrcons.exe running
- $($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}

#Remote Access/Execution utilities
If($_.Process -like "*mstsc.exe") { Add-content -path
"$OutputPath\AnomalousHits-Remote Access.txt" -Value "Mstsc.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*wsmprovhost.exe") { Add-content -path
"$OutputPath\AnomalousHits-Remote Access.txt" -Value "Wsmprovhost.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*psexec.exe") { Add-content -path
"$OutputPath\AnomalousHits-Remote Access.txt" -Value "Psexec.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*putty.exe") { Add-content -path
"$OutputPath\AnomalousHits-Remote Access.txt" -Value "Putty.exe running -
$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
```

```
created by user: $($_.CS_Username). Its elevation rights are $($_.Elevation).
Its index value is $($_.Index) "}
If($_.Process -like "*ssh.exe") { Add-content -path "$OutputPath\AnomalousHits-
Remote Access.txt" -Value "Ssh.exe running - $($_.Process) ran at $($_.Date)
with PID: $($_.PID) and CPID: $($_.CPID), created by user: $($_.CS_Username).
Its elevation rights are $($_.Elevation). Its index value is $($_.Index) "}
}

}

Function Analyze_Find_Evil($OutputPath){
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")
$FindEvilTxt = New-Item -Path $OutputPath -Name "FindEvil_Hits.txt" -ItemType
"file" -Force

Import-Csv $InputCsv | foreach-object {
If($_.Process -like "C:\Windows\System32\smss.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\smss.exe" -and $_.CreatorProcess -notlike "") {
Add-content -path $FindEvilTxt -Value "SMSS running with unusual parent -
investigate further: $($_.Process) and PID: $($_.PID) ran at $($_.Date) with
Parent: $($_.CreatorProcess) and CPID: $($_.CPID), created by user:
$($_.CS_Username). Its elevation rights are $($_.Elevation). Its index value is
$($_.Index) "}
If($_.Process -like "C:\Windows\System32\wininit.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\Smss.exe") { Add-content -path $FindEvilTxt -Value
"Wininit running with unusual parent - investigate further: $($_.Process) and
PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\winlogon.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\Smss.exe") { Add-content -path $FindEvilTxt -Value
"Winlogon running with unusual parent - investigate further: $($_.Process) and
PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\csrss.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\Smss.exe") { Add-content -path $FindEvilTxt -Value
"Csrss running with unusual parent - investigate further: $($_.Process) and
PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and CPID:
$($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\lsaiso.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\wininit.exe") { Add-content -path $FindEvilTxt -
Value "LSAIso running with unusual parent - investigate further: $($_.Process)
and PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and
CPID: $($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\lsass.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\wininit.exe") { Add-content -path $FindEvilTxt -
Value "Lsass running with unusual parent - investigate further: $($_.Process)
and PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and
CPID: $($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.CreatorProcess -like "C:\Windows\System32\lsass.exe") { Add-content -path
$FindEvilTxt -Value "Lsass spawning child processes - investigate further:
$($_.Process) and PID: $($_.PID) ran at $($_.Date) with Parent:
$($_.CreatorProcess) and CPID: $($_.CPID), created by user: $($_.CS_Username).
Its elevation rights are $($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\services.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\wininit.exe") { Add-content -path $FindEvilTxt -
Value "Services running with unusual parent - investigate further:
$($_.Process) and PID: $($_.PID) ran at $($_.Date) with Parent:
$($_.CreatorProcess) and CPID: $($_.CPID), created by user: $($_.CS_Username).
Its elevation rights are $($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\svchost.exe" -and $_.CreatorProcess -
notlike "C:\Windows\System32\services.exe") { Add-content -path $FindEvilTxt -
Value "SVCHost running with unusual parent - investigate further: $($_.Process)
and PID: $($_.PID) ran at $($_.Date) with Parent: $($_.CreatorProcess) and
CPID: $($_.CPID), created by user: $($_.CS_Username). Its elevation rights are
$($_.Elevation). Its index value is $($_.Index) "}
If($_.Process -like "C:\Windows\System32\runtimebroker.exe" -and
$_.CreatorProcess -notlike "C:\Windows\System32\svchost.exe") { Add-content -
```

```
path $FindEvilTxt -Value "Runtimebroker running with unusual parent -
investigate further: $($_.Process) and PID: $($_.PID) ran at $($_.Date) with
Parent: $($_.CreatorProcess) and CPID: $($_.CPID), created by user:
$($_.CS_Username). Its elevation rights are $($_.Elevation). Its index value is
$($_.Index) "}
If($_.Process -like "C:\Windows\System32\taskhostw.exe" -and $_.CreatorProcess
-notlike "C:\Windows\System32\svchost.exe") { Add-content -path $FindEvilTxt -
Value "Taskhostw running with unusual parent - investigate further:
$($_.Process) and PID: $($_.PID) ran at $($_.Date) with Parent:
$($_.CreatorProcess) and CPID: $($_.CPID), created by user: $($_.CS_Username).
Its elevation rights are $($_.Elevation). Its index value is $($_.Index) "}
}

}

Function Search_Process($OutputPath,$ProcQuery){

#$ProcQuery = Read-Host -Prompt "Enter the Process Name you wish to search for"
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")
$PQueryFileName = [System.String]::Concat($ProcQuery,"_Results.csv")
$PQueryFile = New-Item -Path $OutputPath -Name $PQueryFileName -ItemType "file"
-Force

$Pquerycsv = Import-CSV $InputCsv | ForEach-Object {


if($_.Process -like "*$ProcQuery*") {

Write-Host "$ProcQuery hit found - $($_.Process) exists at $($_.Date) with PID:
$($_.PID) and CPID: $($_.CPID), created by user: $($_.CS_Username). Its
elevation rights are $($_.Elevation). Its index value is $($_.Index) "
$_
}


} | ConvertTo-Csv -NoTypeInformation | % {$_.Replace('"','')} | Out-File
$PQueryFile

}

Function Search_User($OutputPath,$UserQuery){
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")
$UQueryFile = [System.String]::Concat($OutputPath,$UserQuery,"_Results.csv")

$Uquerycsv = Import-CSV $InputCsv | ForEach-Object {


if($_.CS_Username -like "*$UserQuery*") {

Write-Host "$UserQuery hit found - $($_.Process) exists at $($_.Date) with PID:
$($_.PID) and CPID: $($_.CPID), created by user: $($_.CS_Username). Its
elevation rights are $($_.Elevation). Its index value is $($_.Index) "
$_
}

} | ConvertTo-Csv -NoTypeInformation | % {$_.Replace('"','')} | Out-File
$UQueryFile

}

Function Search_Elevation($OutputPath,$TokenQuery){
$InputCsv = [System.String]::Concat($outputpath,"\ProcLogs_Indexed.csv")

if($TokenQuery -eq "Type 1") {$TokenEquivalent = "%%1936"}
if($TokenQuery -eq "Type 2") {$TokenEquivalent = "%%1937"}
if($TokenQuery -eq "Type 3") {$TokenEquivalent = "%%1938"}


if($TokenQuery -eq "Type 1") {$TokenExplain = "System, Service, or built in
Administrator, or UAC Disabled"}
if($TokenQuery -eq "Type 2") {$TokenExplain = "Run as Administrator or program
runs as Administrator by default"}
```

Matthew T Moore, matthew.t.moore1@gmail.com

```
if($TokenQuery -eq "Type 3") {$TokenExplain = "Normal User"}

$TokenOutput =
[System.String]::Concat($outputpath,"\$TokenQuery","TokenElevationType.txt")

Import-Csv $InputCsv | foreach-object {
If($_.Elevation -like $TokenEquivalent){ Add-content -path $TokenOutput -Value
"$($_.Process) ran at $($_.Date) with PID: $($_.PID) and CPID: $($_.CPID),
created by user: $($_.CS_Username). Its elevation rights are $TokenExplain. Its
index value is $($_.Index) "}
}

}

# - removed Function Search_PID_Parent($OutputPath,$PIDQuery)


Function Search_PID_Child($OutputPath,$PIDQuery){

# - Child Process Query

$LogCSV = Import-Csv "$OutputPath\ProcLogs_Indexed.csv"
$LogCSV | where {$_.CPID -eq $PIDQuery} | % { Write-host "$($_.Process) (PID:
$($_.PID)) was written at $($_.date) by $($_.CPID) with Index $($_.index)"
}
}


# - Forms Section begins here - this allows for the GUI interface for this tool
$LP_form = [Form] @{
    Text = 'Practical Process Analysis - Log Parsing Module'
    Size = New-Object System.Drawing.Size(1000,300)
    }

$LP_Launch_button = [Button] @{
    Location = '700,100'
    Size = '180,20'
    Text = 'Run Log Parser'
    }

$LP_Launch_button.add_Click{
    $LogPath = $LogPath_TextBox.Text
    $OutputPath = $OutputPath_TextBox.Text

    Write-host "Log path is $LogPath and Output Path is $OutputPath"


    if ($LogPath -AND $OutputPath){

    Parse-Logs $LogPath $OutputPath
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
Output was not provided.","EventParser error",0)}

    }

$LogPath_label = [Label] @{
    Text = "Enter the path to the Windows Security Log Evtx file you would like
to have analyzed"
    Location = '20,30'
    Size = '600,20'
    }

$LogPath_TextBox = [TextBox] @{
    Location = '20,50'
    Size = '320,20'
    Readonly = $true
    Text = $null
    }

$Log_Select_Path_Button = [Button] @{
```

```
    Location = '360,50'
    Size = '120,20'
    Text = 'Select path'
    }

$OutputPath_label = [Label] @{
    Text = "Enter the location where you would like the analysis records saved"
    Location = '20,100'
    Size = '600,20'
    }

$OutputPath_TextBox = [TextBox] @{
    Location = '20,120'
    Size = '320,20'
    Readonly = $true
    Text = $null
    }

$Output_Select_Path_Button = [Button] @{
    Location = '360,120'
    Size = '120,20'
    Text = 'Select path'
    }

$PA_form = [Form] @{
    Text = 'Practical Process Analysis - Analysis Module'
    Size = New-Object System.Drawing.Size(1000,700)
    }

$AutomatedAnalysis_Launch_Button = [Button] @{
    Location = '20,20'
    Size = '800,20'
    Text = 'Run Automated Analysis Queries - Process Path, Process Name, Find
Evil'
    }

$Allow_Launch_button = [Button] @{
    Location = '650,80'
    Size = '300,20'
    Text = 'Compare Logs with Provided Baseline'
    }

$Allow_Launch_button.add_Click{
    $AllowList = $AllowPath_TextBox.Text
    Write-host "Output path is $outputpath in the button"
    if ($OutputPath -and $AllowList){

    Analyze-AllowedList $OutputPath $AllowList
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
Allowed List was not provided.","EventParser error",0)}

    }

$AllowPath_label = [Label] @{
    Text = "Select the AllowedList CSV file you would like to test against"
    Location = '20,50'
    Size = '600,20'
    }

$AllowPath_TextBox = [TextBox] @{
    Location = '20,80'
    Size = '320,20'
    Readonly = $true
    Text = $null
    }

$Allow_Select_Path_Button = [Button] @{
    Location = '360,80'
    Size = '120,20'
    Text = 'Select file'
    }
```

```
# Proc Query
$PQuery_Launch_button = [Button] @{
    Location = '360,130'
    Size = '250,20'
    Text = 'Query for Process name'
    }

$PQuery_Launch_button.add_Click{
    $ProcQuery = $PQuery_TextBox.text

    if ($OutputPath -and $ProcQuery){

    Search_Process $OutputPath $ProcQuery
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
search query was not provided.","EventParser error",0)}

    }

$PQuery_label = [Label] @{
    Text = "Enter the Process Name you would like to search for"
    Location = '20,100'
    Size = '600,20'
    }

$PQuery_TextBox = [TextBox] @{
    Location = '20,130'
    Size = '320,20'
    Text = $null
    }

# User Query
$UQuery_Launch_button = [Button] @{
    Location = '360,200'
    Size = '250,20'
    Text = 'Query for Username'
    }

$UQuery_Launch_button.add_Click{
    $UserQuery = $UQuery_TextBox.text

    if ($OutputPath -and $UserQuery){

    Search_User $OutputPath $UserQuery
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
search query was not provided.","EventParser error",0)}

    }

$UQuery_label = [Label] @{
    Text = "Enter the Username you would like to search for"
    Location = '20,180'
    Size = '600,20'
    }

$UQuery_TextBox = [TextBox] @{
    Location = '20,200'
    Size = '320,20'
    Text = $null
    }


# Token Query
$TQuery_Launch_button = [Button] @{
    Location = '20,280'
    Size = '360,20'
    Text = 'Query for Elevation Token'
    }

$TQuery_Launch_button.add_Click{
```

```
   If($TQType1_RadioButton.checked) {$TokenQuery = "Type 1"
write-host "Type 1 selected"}
    ElseIf($TQType2_RadioButton.checked) {$TokenQuery = "Type 2"
    write-host "Type 2 selected"}
        ElseIf($TQType3_RadioButton.checked) {$TokenQuery = "Type 3"
        write-host "Type 3 selected"}

    if ($OutputPath -and $TokenQuery){

    Search_Elevation $OutputPath $TokenQuery
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
search query was not provided.","EventParser error",0)}
    }

$TQuery_label = [Label] @{
    Text = "Enter the Elevation Token you would like to search for"
    Location = '20,250'
    Size = '600,20'
    }

$TQType1_RadioButton = [RadioButton] @{
    Location = '20,310'
    Size = '750,20'
    Checked = $false
    Text = "Check if you want to query the Process Logs for processes run with
Type 1 (System, Service, Built-in Administrator, or Disabled UAC) elevation
rights"
    }

$TQType2_RadioButton = [RadioButton] @{
    Location = '20,340'
    Size = '750,20'
    Checked = $false
    Text = "Check if you want to query the Process Logs for processes run with
Type 2 (RunAsAdministrator, DefaultAdministrator) elevation rights"
    }

$TQType3_RadioButton = [RadioButton] @{
    Location = '20,370'
    Size = '750,20'
    Checked = $false
    Text = "Check if you want to query the Process Logs for processes run with
Type 3 (Normal User - UAC Enabled) elevation rights"
    }

$PIDQuery_Launch_button = [Button] @{
    Location = '360,440'
    Size = '250,20'
    Text = 'Query for Process ID'
    }

$PIDQuery_Launch_button.add_Click{
    $PIDQuery = $PIDQuery_TextBox.text

    if ($OutputPath -and $PIDQuery){

    #Search_PID_Parent $OutputPath $PIDQuery
    Search_PID_Child $OutputPath $PIDQuery
    }
    else {[System.Windows.Forms.MessageBox]::Show("The path to the Logs or the
search query was not provided.","EventParser error",0)}

    }

$PIDQuery_label = [Label] @{
    Text = "Enter the Process ID (PID) you would like to search for"
    Location = '20,410'
    Size = '600,20'
    }

$PIDQuery_TextBox = [TextBox] @{
```

```
    Location = '20,440'
    Size = '320,20'
    Text = $null
    }

$AutomatedAnalysis_Launch_Button.add_Click{
    Analyze_Proc_Path $OutputPath
    Analyze_Proc_Name $OutputPath
    Analyze_Find_Evil $OutputPath
    }

$folderBrowser = New-Object System.Windows.Forms.FolderBrowserDialog
$folderBrowser.SelectedPath = "C:\"

#Open file
$fileBrowser = New-Object System.Windows.Forms.OpenFileDialog -Property @{
    InitialDirectory = "C:\"
    }

$Log_Select_Path_Button.Add_Click({
   $folderBrowser.ShowDialog()
   $LogPath_TextBox.Text = $folderBrowser.SelectedPath
   })

$Output_Select_Path_Button.Add_Click({
   $folderBrowser.ShowDialog()
   $OutputPath_TextBox.Text = $folderBrowser.SelectedPath
   })

$Allow_Select_Path_Button.Add_Click({
   $fileBrowser.ShowDialog()
   $AllowPath_TextBox.Text = $fileBrowser.Filename
   })

$LP_form.Controls.Add($LP_Launch_button)
$LP_form.Controls.Add($LogPath_label)
$LP_form.Controls.Add($LogPath_TextBox)
$LP_form.Controls.Add($Log_Select_Path_Button)

$LP_form.Controls.Add($OutputPath_label)
$LP_form.Controls.Add($OutputPath_TextBox)
$LP_form.Controls.Add($Output_Select_Path_Button)


$PA_form.Controls.Add($Allow_Launch_button)
$PA_form.Controls.Add($AutomatedAnalysis_Launch_button)

$PA_form.Controls.Add($Allow_Select_Path_Button)
$PA_form.Controls.Add($AllowPath_Textbox)
$PA_form.Controls.Add($AllowPath_label)

$PA_form.Controls.Add($PQuery_Launch_button)
$PA_form.Controls.Add($PQuery_label)
$PA_form.Controls.Add($PQuery_TextBox)

$PA_form.Controls.Add($UQuery_Launch_button)
$PA_form.Controls.Add($UQuery_label)
$PA_form.Controls.Add($UQuery_TextBox)

$PA_form.Controls.Add($TQuery_Launch_button)
$PA_form.Controls.Add($TQuery_label)
$PA_form.Controls.Add($TQType1_RadioButton)
$PA_form.Controls.Add($TQType2_RadioButton)
$PA_form.Controls.Add($TQType3_RadioButton)

$PA_form.Controls.Add($PIDQuery_Launch_button)
$PA_form.Controls.Add($PIDQuery_label)
$PA_form.Controls.Add($PIDQuery_TextBox)

$LP_form.ShowDialog()
```

Appendix C
Sample Output from Script

AllowHits.txt

Hits for Processes Not on Allowed List
C:\Unknown.exe not on Allowed List
C:\Users\test\AppData\Local\AppData.exe not on Allowed List
C:\Users\test\AppData\Local\Programs\ewrwxwk.exe not on Allowed List
C:\Users\test\Downloads\ewrwxwk.exe not on Allowed List
C:\Users\test\Downloads\good.exe not on Allowed List
C:\Users\test\Downloads\win_setup.exe not on Allowed List
C:\Windows\System32\CND.exe not on Allowed List
C:\Windows\System32\1sass.exe not on Allowed List

AnomalousHits.txt

Process running from AppData - C:\Users\test\AppData\Local\AppData.exe ran at 2020-08-03 06:07:23 with PID: 16696 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 9
Process running from AppData - C:\Users\test\AppData\Local\AppData.exe ran at 2020-08-03 06:09:25 with PID: 23372 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 14
Process running from AppData - C:\Users\test\AppData\Local\Programs\ewrwxwk.exe ran at 2020-08-03 06:11:28 with PID: 21756 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 18
SVCHost running with unusual parent - investigate further: C:\Windows\System32\svchost.exe and PID: 13588 ran at 2020-08-04 10:53:08 with Parent: C:\Windows\System32\svchost.exe and CPID: 17976, created by user: testvm$. Its elevation rights are %%1936. Its index value is 4013
Process running from Downloads - C:\Users\test\Downloads\good.exe ran at 2020-08-08 11:41:57 with PID: 20160 and CPID: 3220, created by user: test. Its elevation rights are %%1936. Its index value is 15268
Process running from Downloads - C:\Users\test\Downloads\win_setup.exe ran at 2020-08-08 11:42:23 with PID: 20024 and CPID: 10736, created by user: testvm$. Its elevation rights are %%1936. Its index value is 15271
Process running from Downloads - C:\Users\test\Downloads\ewrwxwk.exe ran at 2020-08-09 03:59:03 with PID: 21836 and CPID: 21396, created by user: test. Its elevation rights are %%1936. Its index value is 16815
SVCHost running with unusual parent - investigate further: C:\Windows\System32\svchost.exe and PID: 8976 ran at 2020-08-11 00:21:56 with Parent: C:\Windows\System32\svchost.exe and CPID: 4780, created by user: testvm$. Its elevation rights are %%1936. Its index value is 22213

AnomalousHits-AppData.txt

Matthew T Moore, matthew.t.moore1@gmail.com

Process running from AppData - C:\Users\test\AppData\Local\AppData.exe ran at 2020-08-03 06:07:23 with PID: 16696 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 9
Process running from AppData - C:\Users\test\AppData\Local\AppData.exe ran at 2020-08-03 06:09:25 with PID: 23372 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 14
Process running from AppData - C:\Users\test\AppData\Local\Programs\ewrwxwk.exe ran at 2020-08-03 06:11:28 with PID: 21756 and CPID: 3056, created by user: test. Its elevation rights are %%1936. Its index value is 18

AnomalousHits-Downloads.txt

Process running from Downloads - C:\Users\test\Downloads\good.exe ran at 2020-08-08 11:41:57 with PID: 20160 and CPID: 3220, created by user: test. Its elevation rights are %%1936. Its index value is 15268
Process running from Downloads - C:\Users\test\Downloads\win_setup.exe ran at 2020-08-08 11:42:23 with PID: 20024 and CPID: 10736, created by user: testvm$. Its elevation rights are %%1936. Its index value is 15271
Process running from Downloads - C:\Users\test\Downloads\ewrwxwk.exe ran at 2020-08-09 03:59:03 with PID: 21836 and CPID: 21396, created by user: test. Its elevation rights are %%1936. Its index value is 16815

AnomalousHits-Shells.txt

CMD.exe running - C:\Windows\SysWOW64\cmd.exe ran at 2020-08-03 19:48:46 with PID: 15080 and CPID: 8012, created by user: LOCAL SERVICE. Its elevation rights are %%1936. Its index value is 680
PowerShell.exe running - C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe ran at 2020-08-03 20:02:52 with PID: 15960 and CPID: 11248, created by user: testvm$. Its elevation rights are %%1936. Its index value is 760

FindEvil_Hits.txt

SVCHost running with unusual parent - investigate further: C:\Windows\System32\svchost.exe and PID: 13588 ran at 2020-08-04 10:53:08 with Parent: C:\Windows\System32\svchost.exe and CPID: 17976, created by user: testvm$. Its elevation rights are %%1936. Its index value is 4013
SVCHost running with unusual parent - investigate further: C:\Windows\System32\svchost.exe and PID: 8976 ran at 2020-08-11 00:21:56 with Parent: C:\Windows\System32\svchost.exe and CPID: 4780, created by user: testvm$. Its elevation rights are %%1936. Its index value is 22213

Wscript_Results.csv

Date,CS_SID,CS_Username,CS_Domain,CS_LogonID,PID,Process,Elevation,CPID,CmdLine,TS_SID,TS_Username,TS_Domain,TS_LogonID,CreatorProcess,MandatoryLabel,Index

Matthew T Moore, matthew.t.moore1@gmail.com

```
2020-08-03 19:44:35,S-1-5-
18,testvm$,VM,0x3e7,17496,C:\Windows\System32\wscript.exe,%%1936,17968,,S
-1-0-0,-,-,0x0,C:\Windows\System32\cscript.exe,S-1-16-16384,577
2020-08-03 19:44:40,S-1-5-21-2478493277-2462340594-4000629390-
166452,test,VMhost,0x13fb5f,19216,C:\Windows\SysWOW64\wscript.exe,%%193
6,18388,,S-1-0-0,-,-,0x0,C:\Windows\SysWOW64\runonce.exe,S-1-16-2342,585
2020-08-04 01:37:18,S-1-5-
18,testvm$,VM,0x3e7,11436,C:\Windows\System32\wscript.exe,%%1936,23332,,S
-1-0-0,-,-,0x0,C:\Windows\System32\cscript.exe,S-1-16-16384,2356
2020-08-05 04:30:09,S-1-5-
18,testvm$,VM,0x3e7,448,C:\Windows\System32\wscript.exe,%%1936,1552,, S-1-
5-21-2478493277-2462340594-4000629390-
166452,test,VMHost,0x10bba2,C:\Windows\System32\svchost.exe,S-1-16-
2342,6273
```

Appendix D
Log Parser Command Line Options & Help Documentation

```
C:\ >logparser.exe -h

Microsoft (R) Log Parser Version 2.2.10
Copyright (C) 2004 Microsoft Corporation. All rights reserved.

Usage:  LogParser [-i:<input_format>] [-o:<output_format>] <SQL query> |
            file:<query_filename>[?param1=value1+...]
            [<input_format_options>] [<output_format_options>]
            [-q[:ON|OFF]] [-e:<max_errors>] [-iw[:ON|OFF]]
            [-stats[:ON|OFF]] [-saveDefaults] [-queryInfo]

       LogParser -c -i:<input_format> -o:<output_format> <from_entity>
            <into_entity> [<where_clause>] [<input_format_options>]
            [<output_format_options>] [-multiSite[:ON|OFF]]
            [-q[:ON|OFF]] [-e:<max_errors>] [-iw[:ON|OFF]]
            [-stats[:ON|OFF]] [-queryInfo]

 -i:<input_format>  : one of IISW3C, NCSA, IIS, IISODBC, BIN, IISMSID,
                HTTPERR, URLSCAN, CSV, TSV, W3C, XML, EVT, ETW,
                NETMON, REG, ADS, TEXTLINE, TEXTWORD, FS, COM (if
                omitted, will guess from the FROM clause)
 -o:<output_format> : one of CSV, TSV, XML, DATAGRID, CHART, SYSLOG,
                NEUROVIEW, NAT, W3C, IIS, SQL, TPL, NULL (if omitted,
                will guess from the INTO clause)
 -q[:ON|OFF]       : quiet mode; default is OFF
 -e:<max_errors>   : max # of parse errors before aborting; default is -1
                (ignore all)
 -iw[:ON|OFF]      : ignore warnings; default is OFF
 -stats[:ON|OFF]   : display statistics after executing query; default is
                ON
 -c                : use built-in conversion query
 -multiSite[:ON|OFF] : send BIN conversion output to multiple files
                depending on the SiteID value; default is OFF
 -saveDefaults     : save specified options as default values
 -restoreDefaults  : restore factory defaults
 -queryInfo        : display query processing information (does not
                execute the query)


Examples:
 LogParser "SELECT date, REVERSEDNS(c-ip) AS Client, COUNT(*) FROM file.log
        WHERE sc-status<>200 GROUP BY date, Client" -e:10


Matthew T Moore, matthew.t.moore1@gmail.com
```

```
LogParser file:myQuery.sql?myInput=C:\temp\ex*.log+myOutput=results.csv
LogParser -c -i:BIN -o:W3C file1.log file2.log "ComputerName IS NOT NULL"

Help:
-h GRAMMAR              : SQL Language Grammar
-h FUNCTIONS [ <function> ] : Functions Syntax
-h EXAMPLES             : Example queries and commands
-h -i:<input_format>    : Help on <input_format>
-h -o:<output_format>   : Help on <output_format>
-h -c                   : Conversion help


C:\ >logparser.exe -h GRAMMAR

SQL Language Grammar
------------------------------------------------------------------------------

<query>          -> <select_clause> [ <using_clause>] [ <into_clause> ]
                 <from_clause> [ <where_clause> ] [ <group_by_clause> ]
                 [ <having_clause> ] [ <order_by_clause> ]

<select_clause>  -> SELECT [ TOP <integer> ] [ DISTINCT | ALL ]
                  <selection_list>
<selection_list> -> <selection_list_el> [ , <selection_list> ]
<selection_list_el> -> <field_expr> [ AS <alias> ] |
                 *

<using_clause>   -> USING <selection_list>

<into_clause>    -> INTO <into_entity>

<from_clause>    -> FROM <from_entity>

<where_clause>   -> WHERE <expression>
<expression>     -> <term1> [ OR <expression> ]
<term1>          -> <term2> [ AND <term1> ]
<term2>          -> <field_expr> <rel_op> <field_expr>            |
                 <field_expr> [ NOT ] LIKE <like_value>           |
                 <field_expr> [ NOT ] BETWEEN <field_expr> AND
                    <field_expr>                                  |
                 <field_expr> <unary_op>                          |
                 <field_expr> <incl_op> <content>                 |
                 <field_expr> <rel_op> [ALL|ANY] <content>        |
                 ( <field_expr_list> ) <incl_op> <content>        |
                 ( <field_expr_list> ) <rel_op> [ALL|ANY] <content> |
                 NOT <term2>                                      |
```

Matthew T Moore, matthew.t.moore1@gmail.com

```
                ( <expression> )
<content>           -> ( <value_list> ) |
                ( <query> )

<group_by_clause>    -> GROUP BY <field_expr_list> [ WITH ROLLUP ]

<having_clause>      -> HAVING <expression>

<order_by_clause>    -> ORDER BY <field_expr_list> [ ASC | DESC ] |
                ORDER BY * [ ASC | DESC ]

<field_expr_list>    -> <field_expr> [ , <field_expr_list> ]
<field_expr>         -> <sqlfunction_expr>    |
                <function_expr>       |
                <value>               |
                <alias>               |
                <field>
<sqlfunction_expr>   -> <sqlfunction> ( [ DISTINCT | ALL ] <field_expr> )   |
                <prop_sqlfunction> ( <field_expr> ) [ <on_fields> ] |
                COUNT ( [ DISTINCT | ALL ] * )                  |
                COUNT ( [ DISTINCT | ALL ] <field_expr_list> )     |
                PROPCOUNT ( * ) [ <on_fields> ]                 |
                PROPCOUNT ( <field_expr_list> ) [ <on_fields> ]
<function_expr>      -> <function> ( <field_expr_list> ) |
                <case_statement>
<value_list>         -> <value_list_row> [ ; <value_list> ]
<value_list_row>     -> <value> [ , <value_list_row> ]
<sqlfunction>        -> SUM | AVG | MAX | MIN | GROUPING
<prop_sqlfunction>   -> PROPSUM
<on_fields>          -> ON ( <field_expr_list> )
<function>           -> ADD | BIT_AND | BIT_NOT | BIT_OR | BIT_SHL        |
                BIT_SHR | BIT_XOR | COALESCE | COMPUTER_NAME | DIV  |
                EXP | EXP10 | EXTRACT_EXTENSION | EXTRACT_FILENAME  |
                EXTRACT_PATH | EXTRACT_PREFIX | EXTRACT_SUFFIX     |
                EXTRACT_TOKEN | EXTRACT_VALUE | FLOOR             |
                HASHMD5_FILE | HASHSEQ | HEX_TO_ASC | HEX_TO_HEX16  |
                HEX_TO_HEX32 | HEX_TO_HEX8 | HEX_TO_INT           |
                HEX_TO_PRINT | IN_ROW_NUMBER | INDEX_OF           |
                INT_TO_IPV4 | IPV4_TO_INT | LAST_INDEX_OF | LOG     |
                LOG10 | LTRIM | MOD | MUL | OUT_ROW_NUMBER         |
                QNTFLOOR_TO_DIGIT | QNTROUND_TO_DIGIT | QUANTIZE    |
                REPLACE_CHR | REPLACE_IF_NOT_NULL | REPLACE_STR     |
                RESOLVE_SID | REVERSEDNS | ROT13 | ROUND | RTRIM    |
                SEQUENCE | SQR | SQRROOT | STRCAT | STRCNT | STRLEN |
                STRREPEAT | STRREV | SUB | SUBSTR | SYSTEM_DATE     |
                SYSTEM_TIME | SYSTEM_TIMESTAMP | SYSTEM_UTCOFFSET   |
```

```
                TO_DATE | TO_HEX | TO_INT | TO_LOCALTIME          |
                TO_LOWERCASE | TO_REAL | TO_STRING | TO_TIME      |
                TO_TIMESTAMP | TO_UPPERCASE | TO_UTCTIME | TRIM    |
                URLESCAPE | URLUNESCAPE | WIN32_ERROR_DESCRIPTION
<case_statement>     -> CASE <field_expression> <when_statement_list>
                 [ <else_statement> ] END
<when_statement_list> -> <when_statement> [ , <when_statement_list> ]
<when_statement>     -> WHEN <field_expression> THEN <field_expression>
<else_statement>     -> ELSE <field_expression>
<value>            -> <string_value> |
                <real>       |
                <integer>    |
                <timestamp>  |
                NULL
<rel_op>           -> < | > | <> | = | <= | >=
<incl_op>          -> IN | NOT IN
<unary_op>         -> IS NULL | IS NOT NULL
<timestamp>        -> TIMESTAMP ( <string_value> , <timestamp_format> )
<timestamp_format>   -> ' *( <timestamp_separator> ) *( <timestamp_element>
                 *( <timestamp_separator> ) )'
<timestamp_element>  -> 1*4 y      |
                1*4 M      |
                MX | MP    |
                1*4 d      |
                dx | dp    |
                1*2 h      |
                hx | hp    |
                1*2 m      |
                mx | mp    |
                1*2 s      |
                sx | sp    |
                1*2 l      |
                lx | lp    |
                1*2 n      |
                nx | np    |
                tt
<timestamp_separator> -> <any_char_except_timestamp_element> |
                '?'
<field>          -> '[' <field_name> ']'   |
                <field_name>
<like_value>       -> ' *( <any_char> | % | _ ) '
<string_value>     -> ' *( <any_char> ) '
<comment>          -> '/*' <text> '*/'   |
                '//' <text> CRLF
```

```
C:\ >logparser.exe -h Functions

Functions Syntax
-------------------------------------------------------------------------------

ADD( addend1 <any type>, addend2 <any type> )
BIT_AND( arg1 <INTEGER>, arg2 <INTEGER> )
BIT_NOT( arg <INTEGER> )
BIT_OR( arg1 <INTEGER>, arg2 <INTEGER> )
BIT_SHL( arg1 <INTEGER>, arg2 <INTEGER> )
BIT_SHR( arg1 <INTEGER>, arg2 <INTEGER> )
BIT_XOR( arg1 <INTEGER>, arg2 <INTEGER> )
CASE <field_expression>
 WHEN <field_expression> THEN <field_expression>
 [ ... ]
 [ ELSE <field_expression> ]
END
COALESCE( arg1 <any type>, arg2 <any type> [, ....] )
COMPUTER_NAME()
DIV( dividend <INTEGER | REAL>, divisor <INTEGER | REAL> )
EXP( argument <INTEGER | REAL> )
EXP10( argument <INTEGER | REAL> )
EXTRACT_EXTENSION( filepath <STRING> )
EXTRACT_FILENAME( filepath <STRING> )
EXTRACT_PATH( filepath <STRING> )
EXTRACT_PREFIX( argument <STRING>, index <INTEGER>, separator <STRING> )
EXTRACT_SUFFIX( argument <STRING>, index <INTEGER>, separator <STRING> )
EXTRACT_TOKEN( argument <STRING>, index <INTEGER> [ , separator <STRING> ]
)
EXTRACT_VALUE( argument <STRING>, key <STRING> [ , separator <STRING> ] )
FLOOR( argument <REAL> )
HASHMD5_FILE( filePath <STRING> )
HASHSEQ( value <STRING> )
HEX_TO_ASC( hexString <STRING> )
HEX_TO_HEX16( hexString <STRING> [ , bigEndian <INTEGER> ] )
HEX_TO_HEX32( hexString <STRING> [ , bigEndian <INTEGER> ] )
HEX_TO_HEX8( hexString <STRING> )
HEX_TO_INT( hexString <STRING> )
HEX_TO_PRINT( hexString <STRING> )
IN_ROW_NUMBER()
INDEX_OF( string <STRING>, searchStr <STRING> )
INT_TO_IPV4( ipV4Address <INTEGER> )
IPV4_TO_INT( ipV4Address <STRING> )
LAST_INDEX_OF( string <STRING>, searchStr <STRING> )
LOG( argument <INTEGER | REAL> )
LOG10( argument <INTEGER | REAL> )
```

Matthew T Moore, matthew.t.moore1@gmail.com

```
LTRIM( string <STRING> )
MOD( dividend <INTEGER | REAL>, divisor <INTEGER | REAL> )
MUL( multiplicand <INTEGER | REAL>, multiplier <INTEGER | REAL> )
OUT_ROW_NUMBER()
QNTFLOOR_TO_DIGIT( value <INTEGER>, digits <INTEGER> )
QNTROUND_TO_DIGIT( value <INTEGER>, digits <INTEGER> )
QUANTIZE( argument <INTEGER | REAL | TIMESTAMP>,
      quantization <INTEGER | REAL> )
REPLACE_CHR( string <STRING>, searchCharacters <STRING>,
       replaceString <STRING> )
REPLACE_IF_NOT_NULL( argument <any type>, replaceValue <any type> )
REPLACE_STR( string <STRING>, searchString <STRING>, replaceString <STRING> )
RESOLVE_SID( sid <STRING> [ , computerName <STRING> ] )
REVERSEDNS( ipAddress <STRING> )
ROT13( string <STRING> )
ROUND( argument <REAL> )
RTRIM( string <STRING> )
SEQUENCE( [ startValue <INTEGER> ] )
SQR( argument <INTEGER | REAL> )
SQRROOT( argument <INTEGER | REAL> )
STRCAT( string1 <STRING>, string2 <STRING> )
STRCNT( string <STRING>, token <STRING> )
STRLEN( string <STRING> )
STRREPEAT( string <STRING>, count <INTEGER> )
STRREV( string <STRING> )
SUB( minuend <any type>, subtrahend <any type> )
SUBSTR( string <STRING>, start <INTEGER> [ , length <INTEGER> ])
SYSTEM_DATE()
SYSTEM_TIME()
SYSTEM_TIMESTAMP()
SYSTEM_UTCOFFSET()
TO_DATE( timestamp <TIMESTAMP> )
TO_HEX( argument <INTEGER | STRING> )
TO_INT( argument <any type> )
TO_LOCALTIME( timestamp <TIMESTAMP> )
TO_LOWERCASE( string <STRING> )
TO_REAL( argument <any type> )
TO_STRING( argument <INTEGER | REAL> ) |
      ( timestamp <TIMESTAMP>, format <STRING> )
TO_TIME( timestamp <TIMESTAMP> )
TO_TIMESTAMP( dateTime1 <TIMESTAMP>, dateTime2 <TIMESTAMP> ) |
       ( string <STRING>, format <STRING> )
       ( seconds <INTEGER | REAL> )
TO_UPPERCASE( string <STRING> )
TO_UTCTIME( timestamp <TIMESTAMP> )
TRIM( string <STRING> )
```

Matthew T Moore, matthew.t.moore1@gmail.com

```
URLESCAPE( url <STRING> [ , codepage <INTEGER> ] )
URLUNESCAPE( url <STRING> [ , codepage <INTEGER> ] )
WIN32_ERROR_DESCRIPTION( win32ErrorCode <INTEGER> )

C:\ >logparser.exe -h examples

Create a CSV file containing basic information from an ETW .etl binary log
 file:
 LogParser "SELECT * INTO Trace.csv FROM myFile.etl"

Create a chart containing the TOP 20 URL's in the "www.margiestravel.com" web
 site (assumed to be logging in the W3C log format):
 LogParser "SELECT TOP 20 cs-uri-stem, COUNT(*) AS Hits INTO MyChart.gif FROM
 <www.margiestravel.com> GROUP BY cs-uri-stem ORDER BY Hits DESC"
 -chartType:Column3D -groupSize:1024x768

Print the 10 largest files on the C: drive:
 LogParser "SELECT TOP 10 * FROM C:\*.* ORDER BY Size DESC" -i:FS

Create an XML report file containing logon account names and dates from the
 Security Event Log messages:
 LogParser "SELECT TimeGenerated AS LogonDate, EXTRACT_TOKEN(Strings, 0, '|')
 AS Account INTO Report.xml FROM Security WHERE EventID NOT IN
(541;542;543)
 AND EventType = 8 AND EventCategory = 2"

Load a portion of the registry into a SQL table:
 LogParser "SELECT * INTO MyTable FROM \HKLM" -i:REG -o:SQL -server:MyServer
 -database:MyDatabase -driver:"SQL Server" -username:TestSQLUser
 -password:TestSQLPassword -createTable:ON

Parse the output of a 'netstat' command:
 netstat | LogParser "SELECT * FROM STDIN" -i:TSV -iSeparator:space -nSep:2
 -fixedSep:off -nSkipLines:3

Display users' job title breakdown from Active Directory:
 LogParser "SELECT title, MUL(PROPCOUNT(*), 100.0) AS Percentage INTO
 DATAGRID FROM
'LDAP://myusername:mypassword@mydomain/CN=Users,DC=mydomain,DC
 =com' WHERE title IS NOT NULL GROUP BY title ORDER BY Percentage DESC"
 -objClass:user

Retrieve all the AccessFlags properties from IIS metabase objects:
 LogParser "SELECT ObjectPath, PropertyValue FROM IIS://localhost WHERE
 PropertyName = 'AccessFlags'"
```

Matthew T Moore, matthew.t.moore1@gmail.com

Send error entries in the IIS log to a SYSLOG server:
LogParser "SELECT TO_TIMESTAMP(date,time), CASE sc-status WHEN 500 THEN 'emerg' ELSE 'err' END AS MySeverity, s-computername AS MyHostname, cs-uri-stem INTO @myserver FROM <1> WHERE sc-status >= 400" -o:SYSLOG -severity:$MySeverity -hostName:$MyHostname

Create a pie chart with the total number of bytes generated by each extension:
LogParser "SELECT EXTRACT_EXTENSION(cs-uri-stem) AS Extension, MUL(PROPSUM(sc-bytes),100.0) AS Bytes INTO Pie.gif FROM <1> GROUP BY Extension ORDER BY Bytes DESC" -chartType:PieExploded -chartTitle:"Bytes per extension" -categories:off

Get the distribution of EventID values for each Source:
LogParser "SELECT SourceName, EventID, MUL(PROPCOUNT(*) ON (SourceName), 100.0) AS Percent FROM System GROUP BY SourceName, EventID ORDER BY SourceName, Percent DESC"

Parse a TSV file from a web URL:
LogParser "SELECT * FROM https://myusername:mypassword@www.margiestravel.com/MyUrl" -i:TSV

Create TSV files containing Event Messages for each Source in the Application Event Log:
LogParser "SELECT SourceName, Message INTO myFile_*.tsv FROM \\MYSERVER1\Application, \\MYSERVER2\Application"

Create a CSV file with information from a custom COM plugin:
LogParser "SELECT * INTO Report.csv FROM MyFromEntity" -i:COM -iProgID:MyCompany.MyComPlugin -iCOMParams:TargetMachine=localhost,ExtendedFields=on

List the fields extracted from a CSV file:
LogParser -h -i:CSV myfile.csv -headerRow:on

List the fields extracted from a TSV file:
LogParser -h -i:TSV myfile.tsv -headerRow:on

Display total network traffic bytes per second:
LogParser "SELECT QUANTIZE(DateTime, 1) AS Second, SUM(FrameBytes) INTO DATAGRID FROM myCapture.cap GROUP BY Second"

List from an ETW trace all the filepaths accessed by IIS:
LogParser "SELECT FileName, COUNT(*) AS Total INTO filenames.w3c FROM

Matthew T Moore, matthew.t.moore1@gmail.com

```
iistrace.etl GROUP BY FileName ORDER BY Total DESC" -providers:"IIS: WWW
Server" -fmode:full -o:W3C -encodeDelim:ON
```

Show all the IIS events from an ETW trace, grouped together by request:

```
LogParser "SELECT EventName, EventTypeName, Timestamp,
HASHSEQ(ContextId) AS
Id INTO DATAGRID FROM iistrace.etl WHERE ContextId IS NOT NULL ORDER BY Id,
Timestamp ASC" -providers:myfile.guid -fmode:full
```

Display the distribution of registry value types:

```
LogParser "SELECT ValueType, COUNT(*) INTO DATAGRID FROM \HKLM GROUP
BY
ValueType"
```

Display titles of current channels on MSDN BLogs:

```
LogParser "SELECT title INTO MyOutput.txt FROM http://blogs.msdn.com/MainFee
d.aspx#/rss/channel/item" -i:XML -fMode:Tree -o:tpl -tpl:mytemplate.tpl
```

Appendix E
Script Dependencies

The script presented in Appendix B relies on several variables, including some user-generated fields. However, two static dependencies need to be in place for the script to function correctly. First is that the LogParser binary needs to be in a subfolder 'bin' of where the script is located. Secondly, the Baselining module requires a list of known executables on the system and is read into the script by calling all instances of its 'process' column. This can be changed in the script if the analyst wishes to do so, in line 94 (shown below), by changing the field 'process' to whatever the analyst wishes to use.

```
if($AllowCSV.process -like $Query) {
```

	A	B	C
1	Process		
2	C:\Windows\System32\drvcfg.e		
3	C:\Windows\System32\drvinst.e		
4	C:\Windows\System32\dsac.exe		
5	C:\Windows\System32\dsacls.e		
6	C:\Windows\System32\dsadd.e		
7	C:\Windows\System32\dsdbutil.		
8	C:\Windows\System32\dsget.ex		
9	C:\Windows\System32\dsmgmt.		
10	C:\Windows\System32\dsmod.e		
11	C:\Windows\System32\dsmove.		
12	C:\Windows\System32\DsmUse		