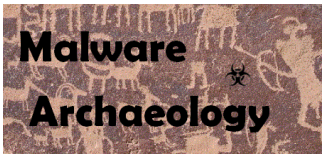
WINDOWS LOGSCALE LOGGING CHEAT SHEET - Win 7 – Win2022

This "**Windows Crowdstrike Logscale (formerly Humio) Logging Cheat Sheet**" is intended to help you get started setting up Logscale queries and alerts for the many critical Windows security related events. By no means is this list extensive; but it does include some very common items that are a must for any Information Security and Log Management Security Program. Start with these samples and add to it as you understand better what is in your logs and what you need to monitor and alert on.



DEFINITIONS:

WINDOWS LOGGING CONFIGURATION: Before you can gather anything meaningful with Logscale, or any other log management solution, the Windows logging and auditing must be properly Enabled and Configured before you can Gather and Harvest the logs into Logscale. The Center for Internet Security (CIS) Benchmarks will give you some guidance on what to configure; but does not go far enough to log and audit what is really needed for a proper Information Security, Incident Response and Threat Hunting program. The "***Windows Logging Cheat Sheet***" contains the details needed for proper and complete security logging to understand how to Enable and Configure Windows audit log settings so you can capture meaningful and actionable security related data. You can get the "***Windows Logging Cheat Sheet***" and other logging cheat sheets here:

- ***MalwareArchaeology.com/cheat-sheets***

REPORTS: Saved Queries that are saved for reference and can be used as needed.

ALERTS: Queries you want to be emailed, sent to a chat app or sent to your smartphone to alert you that something is outside the norm and needs to be investigated immediately. Do not get alert heavy or your staff will ignore them, make your alerts actionable, meaning someone must look into the alert as it is unusual and/or suspicious. This takes tuning to remove normal noisy items.

DASHBOARDS: A collection of reports or alerts that are saved into a dashboard view for quick reference. Often used for NOC's and SOC's to monitor critical activity. Dashboards are left up to each user as organization's have different needs and preferences on what they want to see.

ABOUT THE SAMPLE QUERIES: The sample queries found in this cheat sheet are based on using Elastics WinLogBeat agents. The field names are from the WinLogBeat version 7 format. You can use many log collection agents and adjust the queries to the field names the agent labels the fields as they will be similar as you see in the log data.

- WInLogBeat and Filebeat agents mentioned in this cheat sheet:
  - WinLogBeat - https://www.elastic.co/downloads/beats/winlogbeat
  - FileBeat - https://www.elastic.co/downloads/beats/filebeat

AUDIT YOUR WINDOWS ADVANCED AUDIT POLICIES TO THE CHEAT SHEETS:

MEASURE YOUR AUDIT SCORE: If you are contemplating if your Windows auditing is set well enough to utilize this cheat sheet or use it for any incident response, threat hunting or Log Management/SIEM solution, you should first measure your settings. It just so happens we have a tool for you, ***LOG-MD*** can be used to quickly measure your audit policy settings against several industry standards including the, The Center for Internet Security (CIS) Windows Benchmarks and our "***Windows Logging Cheat Sheet***". You can get a ***FREE*** copy of ***LOG-MD*** below and feel free to support our efforts by purchasing a Pro or Premium version for your Windows incident response and threat hunting toolkits:

- logmd.com/download

Open an administrator command prompt or PowerShell session and navigate to where you have ***LOG-MD***. Run the following command and look at the two reports to see how you score. Run this on a sample system based on how your AD GPO is setup on where you apply audit policy settings. Adjust your settings until you pass the audit score.

- LOG-MD -a > Report_Audit_Score.txt

REPORTS: Review the two reports:

- Report_Audit_Score.txt – The score of your system in a Pass/Fail format
- Report_Audit_Settings.txt – Detailed report of settings compared to other standards

RESOURCES: Places to get more information.

You can sign up for Logscale and use it for free on a limited use of data, great to use for creating detections for your home or lab.

- https://www.crowdstrike.com/products/next-gen-siem/falcon-logscale – More information and documentation for Logscale – ***FREE TRIAL AND USE !!!***

- MalwareArchaeology.com/cheat-sheets – More Windows Logging Cheat Sheets and resources
- MalwareArchaeology.com/logging – Scripts and sample agent configs to set and collect your logging
- LOGMD.com – LOG-MD Free, Pro and Premium – Windows incident response and threat hunting tool. Also, FILE-MD – Comes with LOG-MD-Premium to perform static analysis on files, folders or entire drives
- Better descriptions of Event ID's
    - www.ultimatewindowssecurity.com/securitylog/encyclopedia/Default.aspx
- https://www.myeventlog.com – Lookup a Windows Event ID
- Microsoft list of Event IDs for Win 10 and Server 2016 (works for any version)
    - https://www.microsoft.com/en-us/download/confirmation.aspx?id=52630
- Google – Of course
- JPCERT/CC Research Report on Detecting Lateral Movement – Great for creating queries and alerts on detecting malicious lateral movement:
    - https://blogs.jpcert.or.jp/en/2017/12/research-report-released-detecting-lateral-movement-through-tracking-event-logs-version-2.html

CRITICAL EVENTS TO MONITOR:

The following is a short list of events to collect, create queries and alert on various conditions covered in this cheat sheet;

1. NEW PROCESS STARTING: Event Code 4688 will capture when a process or executable starts.
2. CLEAR THE LOGS: Event Codes 104 and 1102 will catch when logs are cleared.
3. USER LOGONS: Event Codes 4624 and 4625 will capture user logons to the system.
4. SHARE ACCESSED: Event Code 5140 & 5145 will capture when a user connects to a file share.
5. NEW SERVICE INSTALLED: Event Code 7045 will capture when a new service is installed.
6. NETWORK CONNECTION MADE: Event Code 5156 will capture when a network connection is made from the source to the destination including the ports used and the process used to initiate the connection. Requires the use of the Windows Firewall auditing being enabled
7. FILE AUDITING: Event Code 4663 will capture when a new file is added, modified or deleted.
8. REGISTRY AUDITING: Event Code 4657 will capture when a new registry item is added, modified or deleted
9. WINDOWS FIREWALL CHANGES: Event Code 2004 will capture when new firewall rules are added.
10. SCHEDULE TASKS ADDED: Event Code 106 will capture when a new scheduled task is added.
11. WINDOWS POWERSHELL COMMANDS: Event Code 200, 400, 500, 600, 800 (PSv2), and 4100, 4103, 4104 (PSv5) will capture what PowerShell is executing, large script blocks, BASE64, and obfuscation involved.

FILTERING EVENTS:

1. Filtering Events: It is common to blacklist or exclude event codes and events that are noisy, excessive, or trusted normal items that impact storage and licensing. Your top events in your Logscale dataspace, if enabled will be Process Creation Success (4688), Process Terminate (4689), and Windows Firewall Filtering Platform Connection Success (5156 & 5158) for workstations, and logins (4624) for domain controllers. Filtering events on the client side reduces the number of events you send and collect into Logscale. This helps by collecting the right things and not all the things, just the relevant security events. Once you understand what normal noise is, what has minimal risk to be exploited, or what events are not important to security monitoring, then filter these out at the client or server. For Windows, Logscale recommends the Elastic Winlogbeat and Filebeat event collectors for Windows to collect events. You can get the Beats collection agents here:
   - **https://www.elastic.co/downloads/beats/winlogbeat**
   - **https://www.elastic.co/downloads/beats/filebeat**

   You can get a copy of a sample winlogbeat.yml file that is tweaked for the events in the cheat sheet and includes many exclusions on our website:
   - **https://www.malwarearchaeology.com/logging**

   You can get information on what to collect and filter from the cheat sheets as well which may be found on our website:
   - **https://www.malwarearchaeology.com/cheat-sheets**

COLLECTING SECURITY RELEVANT EVENTS AND EXCLUDING NOISY EVENTS:

2. Winlogbeat examples:

The following are some examples of what you can collect and exclude from sending data to Logscale using the winlogbeat.yml file. You can exclude by event ID, exact name, contains name, or a combo of two items. It is best to be specific on exclusions and not get too general or overly broad when you exclude a lot of items. Trimming unnecessary events is more work, but the results are better with less chances of an exclusion path being used to hide malware for example. Get a full Winlogbeat.yml file from ***MalwareArchaeology.com***

```
winlogbeat.event_logs:
 - name: Security
   event_id: -4689, -5158, -5440, -5444
 - name: Application
   ignore_older: 720h
 - name: System
 - name: "Microsoft-Windows-TaskScheduler/Operational"
 - name: "Microsoft-Windows-PowerShell/Operational"
   event_id: -4105, -4106
 - name: Microsoft-Windows-Windows Defender/Operational
 - name: Microsoft-Windows-Windows Firewall With Advanced Security/Firewall
   include_xml: true
 - name: "Windows PowerShell"
   event_id: -501
 # - name: "Microsoft-Windows-Sysmon/Operational" (optional in case you are using the Sysmon service)

processors:
#
# Exclude 4688 Process Create items
#
- drop_event.when.or:
 - equals.event_data.NewProcessName: 'C:\Users\<username>\AppData\Local\Temp\SkypeSetup.exe'
 - contains.event_data.NewProcessName: 'C:\Users\<username>\AppData\Local\GoToMeeting\'
 - equals.event_data.NewProcessName: 'C:\Program Files\winlogbeat\winlogbeat.exe'     (no need to collect forwarding agent)
#
# Exclude 4688 Process Create by Command Line items
#
- drop_event.when.or:
 - equals.event_data.CommandLine: 'C:\WINDOWS\system32\SearchFilterHost.exe" 0 540 544 552 65536 548'
 - equals.event_data.CommandLine: 'cmd ver'
 - contains.event_data.CommandLine: 'C:\WINDOWS\system32\SearchFilterHost.exe'
#
# Exclude 5156 Win Firewall item by Application and Source address
#
- drop_event.when.and:
 - contains.event_data.Application: '\program files\programX\server.exe'
 - equals.event_data.SourceAddress: '192.168.1.100'
```

COLLECTING SECURITY RELEVANT EVENTS FROM ONE OR MORE FLAT LOG FILES:

3. Filebeat example:

The following are some examples of what you can collect from a file or files on your system. The example below collects LOG-MD and FILE-MD reports that are in CSV format to forward to Logscale. Get a full Filebeat.yml file from ***MalwareArchaeology.com***

filebeat.inputs:

```
- type: log
  paths:
    - C:\ProgramData\filebeat\logs\filebeat
  enabled: true
  exclude_lines: ['.*Harvester started for file.*', '.*Starting input of type.*', '.*pipeline.*', '.*crawler.*']
  encoding: utf-8
  close_eof: true
  scan_frequency: 5m
  fields:
    type: Filebeat
    level: debug
    review: 1

- type: log
  paths:
    - C:\ProgramData\Elastic\Beats\winlogbeat\Logs\winlogbeat
  enabled: true
  exclude_lines: ['.*successfully published*', '.*log/log.*']
  encoding: utf-8
  close_eof: true
  scan_frequency: 5m
  fields:
    type: WinLogBeat
    level: debug
    review: 1

#
#  Audit Score
#
- type: log
  enabled: true
  paths:
    - C:/Program Files/LOG-MD/_Logs/Audit_Score.txt
# NOT Unicode
  encoding: utf-8
  close_eof: true
  scan_interval: 5m
  fields:
    "type": LOG-MD-Audit
    "type_tool": LOG-MD
    "type_category": Audit
```

The following Logscale Queries should be both a Query and an Alert. Alerts should be actionable, meaning when they go off something new and/or odd has occurred and you should respond and investigate. If the alert has false positives, edit them to reduce the normal noise.

MONITOR FOR PROCESSES STARTING – SECURITY LOG - 4688:

1. Monitor for Suspicious/Administrative Processes: This search is based on built-in Windows administrative utilities and known hacking utilities that are used in exploitation. Expand this list as needed to add utilities used in attacks. Some administrative tools are very noisy and normally used or automatically executed regularly and should NOT be included in order to make your alert more actionable and accurate that something suspicious has occurred. An extensive list of administrative utilities to monitor for may be found on:
   - http://www.MalwareArchaeology.com\config

   SAMPLE QUERY: Trigger alert on 5th command executed, by a system which exceeds what a normal administrator would do. It is typical to see a threat actor execute many admin recon utilities across multiple systems which is a great indicator of reconnaissance being performed. Adjust for your environment, but this is a great query to catch recon and lateral movement activity.

```
winlog.channel=Security and event.code=4688 // List of Admin Utils
(/arp.exe/i OR /at.exe/i OR /bcdedit.exe/i OR /bcp.exe/i OR /chcp.exe/i OR /cmd.exe/i OR /cscript.exe/i OR /csvde/i OR
/dsquery.exe/i OR /ipconfig.exe/i OR /mimikatz.exe/i OR /nbtstat.exe/i OR /\\nc.exe/i OR /netcat.exe/i OR /netstat.exe/i OR
/nmap.exe/i OR /nslookup.exe/i OR /netsh.exe/i OR /OSQL.exe/i OR /ping.exe/i OR /powershell.exe/i OR /powercat.ps1/i OR
/psexec.exe/i OR /psexecsvc.exe/i OR /psLoggedOn.exe/i OR /procdump.exe/i OR /qprocess.exe/i OR /query.exe/i OR /rar.exe/i OR
/route.exe/i OR /runas.exe/i OR /rundll32.exe/i OR /schtasks.exe/i OR /sethc.exe/i OR /sqlcmd.exe/i OR /sc.exe/i OR /ssh.exe/i OR
/sysprep.exe/i OR /systeminfo.exe/i OR /tasklist.exe/i OR /tracert.exe/i OR /vssadmin.exe/i OR /whoami.exe/i OR /winrar.exe/i OR
/wscript.exe/i OR /"winrm.*"/i OR /"winrs.*"/i OR /wmic.exe/i OR /wsmprovhost.exe/i OR /wusa.exe/i)
| SubjectUserName:=winlog.event_data.SubjectUserName | TargetUserName:=winlog.event_data.TargetUserName |
TargetDomainName:=winlog.event_data.TargetDomainName | CommandLine:=winlog.event_data.CommandLine
| NewProcessName:=winlog.event_data.NewProcessName | ParentProcessName:=winlog.event_data.ParentProcessName
| NewProcessName =~ /(?<ShortName>[^\\]+$)/
| groupby([host.name, SubjectUserName],
  function=[Process_Cnt:=count(distinct=false, "NewProcessName"),
        { groupby(ShortName) | sort()
        | format("%d:%s", field=[_count,ShortName], as=Processes)
        | collect([Processes])},
        { collect(CommandLine) | table(CommandLine) }])
| table([host.name, SubjectUserName, Process_Cnt, Distinct_Cnt, Processes, CommandLine]) | Process_Cnt > 4
```

MONITOR FOR PROCESSES STARTING – SECURITY LOG - 4688:

2. Monitor for Suspicious/Administrative Processes: This query is the same as above but using the File input and match() feature of Logscale.

SAMPLE QUERY: Trigger alert on excessive admin commands executed using a File and match() feature. Add the count feature from the previous query sample to count how many commands and by system are executed.

```
winlog.channel=Security and event.code=4688 // List of Admin Utils from File
| Token:= winlog.event_data.TokenElevationType | User_Name:= winlog.event_data.SubjectUserName |
Domain:=winlog.event_data.SubjectDomainName | New_Process_Name:= winlog.event_data.NewProcessName | PPID:=
winlog.event_data.ProcessId | PID:= winlog.event_data.NewProcessId | Command_Line:= winlog.event_data.CommandLine |
Parent_Process:=winlog.event_data.ParentProcessName
| match(file="Admin_Utilities.csv", field=New_Process_Name, ignoreCase=true)
| alt{ Token=%%1937 | Token := "Type_2_Admin" ; Token=%%1936 | Token := "Type_1_Normal" ; Token=%%1938 | Token :=
"Type_3_No_Admin"}
| table([event.code, @timestamp, host.name, User_Name, Domain, PID, New_Process_Name, Command_Line, PPID,
Parent_Process, Token])
```

3. Monitor for LOLBAS Usage: Threat actors will often use built-in utilities known as “Living-Off-The-Land Binaries and Scripts” (LOLBAS) that are trusted by whitelisting and security solutions that can be used for malicious activities. Watching for the use of these, filtering out known good, can detect a PenTester or malicious threat actor activity. A full list of LOLBAS files that can be used to build a list may be found here:
    - **https://github.com/LOLBAS-Project/LOLBAS**

4. Monitor for Whitelisting (AppLocker) bypass attempts: Threat actors will often use built-in utilities that are trusted by whitelisting solutions that can be used for malicious activities. Watching for the use of these, filtering out known good, can detect a PenTester or malicious hacking activity. A full list of older Applocker bypasses may be found here:
    - **https://github.com/api0cradle/UltimateAppLockerByPassList**

An extensive list of administrative utilities, including bypass utilities to monitor can also be found on:

- **https://www.malwarearchaeology.com/logging**

The “***match()***” command will benefit this query tremendously by having a list to match from a file for all the utilities versus having them all listed in the query. Keep in mind the files must be exact matches and does not support the wildcard ‘*’ so the entire path of the command or exact match must be added to the file column, for example;

- C:\windows\system32\ipconfig — Correct format, case does not matter if ***ignorecase=true*** used
- /ipconfig/i — Ignore case does not work (in queries only)
- *ipconfig.exe — Wildcards not supported in files (in queries only)

4. Monitor for execution in the Users directory: One of the most useful alerts for workstations and servers is monitoring of the c:\Users directory structure. Creating alerts for any odd executions under "***C:\Users***" can catch malicious activity and this is often where systems will first get infected. Exclude known good items as this will see all user launched processes. This can be crafted to monitor any important directory structure you might have.

SAMPLE QUERY: Monitor for process execution in the "C:\Users" directory structure

```
winlog.channel=Security "event.code"="4688" and /users/i // User Dir execution - Beats v7
| SubjectUserName:=winlog.event_data.SubjectUserName | TargetUserName:=winlog.event_data.TargetUserName |
TargetDomainName:=winlog.event_data.TargetDomainName | CommandLine:=winlog.event_data.CommandLine |
NewProcessName:=winlog.event_data.NewProcessName | ParentProcessName:=winlog.event_data.ParentProcessName
| NewProcessName=/c:\\users/i
| table([event.code, @timestamp, host.name, SubjectUserName, TargetUserName, TargetDomainName, CommandLine,
ParentProcessName, NewProcessName])
```

MONITOR FOR CLEARING OF LOGS – SECURITY & SYTEM LOGS:

1. Monitor for clearing of the event logs that can be performed by threat actors to clear their tracks.

SAMPLE QUERY: Monitor for clearing of the logs

```
(winlog.channel=System AND event.code="104") or (winlog.channel=Security AND event.code="1102")
| UserName:= winlog.user.name | UserName:=winlog.user_data.SubjectUserName | Domain:=winlog.user.domain |
Domain:=winlog.user_data.SubjectDomainName | Channel:=winlog.user_data.Channel | user.type:=winlog.user.type |
Action:=event.action | Action:=winlog.task
| table([event.code, @timestamp, host.name, UserName, Domain, user.type, Action, winlog.channel])
```

MONITOR FOR NETWORK CONNECTIONS – SECURITY LOG - 5156:

1. Monitor for Suspicious Network IP's: This does require the use of the Windows Firewall audit settings, but does not require any enforcement of the Windows Firewall. The traffic is captured in the logs and more importantly what process made the connection to an IP address. You can create exclusions by IP addresses (such as broadcast IP's) and by process names to reduce the output. The "***match()***" command will benefit this query tremendously by excluding items. This looks at outbound traffic

SAMPLE QUERY:

```
winlog.channel=Security AND event.code=5156 AND winlog.event_data.Direction="%%14593" // Win FW Non Std Ports v7
| Application:=winlog.event_data.Application | Dest_IP:=winlog.event_data.DestAddress | Dest_Port:=winlog.event_data.DestPort |
Protocol:=winlog.event_data.Protocol | Direction:=winlog.event_data.Direction | PID:=winlog.event_data.ProcessID |
Src_IP:=winlog.event_data.SourceAddress | Src_Port:=winlog.event_data.SourcePort
| alt{ Protocol=6 | Protocol := "TCP" ; Protocol=17 | Protocol := "UDP" }
| iplocation(Dest_IP) | Co:=Dest_IP.country | St:=Dest_IP.state | City:=Dest_IP.city
| alt { Direction="%%14592" | Direction := "Inbound" ; Direction="%%14593" | Direction := "Outbound" }
| table([event.code, @timestamp, host.name, Dest_IP, Dest_Port, Direction, Co, St, City, Protocol, Src_IP, Src_Port, Application])
| NOT Dest_Port=80 | NOT Dest_Port=137 | NOT Dest_Port=443 | NOT Dest_Port=514
| NOT Dest_IP=239.255.255.250 | NOT Dest_IP=255.255.255.255 | NOT Dest_IP="::1"
```

MONITOR FOR USER LOGONS – SECURITY LOG - 4624 & 4625:

1. Monitor for Logon Success: Logging for failed logons seems obvious, but when a user credential gets compromised and their credentials used for exploitation, successful logins will be a major indicator of malicious activity, recon, share enumeration and lateral movement. This alert looks for successful logons to detect when a rogue user account moves across systems in your network, also known as lateral movement. It is important to understand the type of account and how it is logging in, for example if a service account is logging in over the network this may indicate a compromised service account that has more than 'logon as a service' and being misused. A single user account logging into multiple systems with a network logon that normally does not may indicate lateral movement of a compromised account.

SAMPLE QUERY:

```
winlog.channel=Security AND event.code=4624 OR event.code=4625 // User Logins v7
| Auth_Type:=winlog.event_data.AuthenticationPackageName | Impersonation:=winlog.event_data.ImpersonationLevel |
IP_Address:=winlog.event_data.IpAddress | Port:=winlog.event_data.IpPort | LMPackage:=winlog.event_data.LmPackageName |
Logon_Process:=winlog.event_data.LogonProcessName | Type:=winlog.event_data.LogonType |
Logon_Type:=winlog.event_data.LogonType | PID:=winlog.event_data.ProcessId |
Process_Name:=winlog.event_data.ProcessName | Domain:=winlog.event_data.SubjectDomainName |
UserName:=winlog.event_data.SubjectUserName | Target_Dom:=winlog.event_data.TargetDomainName |
Targ_User:=winlog.event_data.TargetOutboundUserName
| alt{ Logon_Type=0 | Logon_Type := "System" ; Logon_Type=2 | Logon_Type := "Network" ; Logon_Type=3 | Logon_Type :=
"Interactive" ; Logon_Type=4 | Logon_Type := "Batch" ; Logon_Type=5 | Logon_Type := "Service" ; Logon_Type=7 | Logon_Type :=
"Unlock"; Logon_Type=8 | Logon_Type := "Net_Clear"; Logon_Type=9 | Logon_Type := "Run_As"; Logon_Type=10 | Logon_Type :=
"RDP"; Logon_Type=11 | Logon_Type := "Cache_Int"; Logon_Type=12 | Logon_Type := "Cache_Rem"; Logon_Type=13 |
Logon_Type := "Cache_Unlock"}
| table([event.code, @timestamp, host.name, UserName, Domain, Type, Logon_Type, Targ_Dom, Targ_User, Auth_Type,
IP_Address, Port, LMPackage, Logon_Process, PID, Process_Name]) | NOT IP_Address="-"
| NOT (Logon_Type=Service AND Process_Name=C:\Windows\System32\services.exe) | NOT Logon_Type=Batch | NOT
Logon_Type=Interactive | NOT Type=2 | NOT Type=11 | NOT Type=7 | NOT Type=0
```

2. Monitor Network (type 3) and RDP (type 10) for Logons: Threat actors will often laterally move across systems looking for information and data to harvest by using the command line 'Net Use' command. Monitoring lateral movement behavior by quantity of systems can indicate suspicious behavior. Adding a count can detect mass recon lateral movement as well or scripted attempts (>5 systems), some normal admin behaviors will need to filtered out. The same holds true for Remote Desktop Connections (RDP), but these would likely hit servers after hours so less chance of detection since they are interacting with the actual desktop should be monitored. The equivalent PowerShell commands as well should be monitored.

MONITOR FOR USER LOGONS – SECURITY LOG - 4624 & 4625 continued:

3. Monitor for Logon Failures: Watch for excessive logon failures, especially Internet facing systems and systems that contain confidential data. This can also detect brute force attempts and users who have failed to changed their passwords on additional devices such as smartphones. You can add "***count***" to watch for quantity, exclude certain accounts you know are good and normally fail. Avoid excluding administrative accounts as they are the ones the hackers are after. You can also populate a query with past compromised accounts that you have retired to monitor for attempts to use older compromised credentials known as Honey Accounts.

SAMPLE QUERY:

```
winlog.channel=Security | event.code=4625 // Failed Logons v7
| Auth_Type:=winlog.event_data.AuthenticationPackageName | Impersonation:=winlog.event_data.ImpersonationLevel |
IP_Address:=winlog.event_data.IpAddress | Port:=winlog.event_data.IpPort | LMPackage:=winlog.event_data.LmPackageName |
Logon_Process:=winlog.event_data.LogonProcessName | Type:=winlog.event_data.LogonType |
Logon_Type:=winlog.event_data.LogonType | PID:=winlog.event_data.ProcessId |
Process_Name:=winlog.event_data.ProcessName | Domain:=winlog.event_data.SubjectDomainName |
UserName:=winlog.event_data.SubjectUserName | Target_User:=winlog.event_data.TargetUserName |
Target_Dom:=winlog.event_data.TargetDomainName | Targ_User:=winlog.event_data.TargetOutboundUserName |
Reason:=winlog.event_data.FailureReason
| alt{ Logon_Type=0 | Logon_Type := "System" ; Logon_Type=2 | Logon_Type := "Network" ; Logon_Type=3 | Logon_Type :=
"Interactive" ; Logon_Type=4 | Logon_Type := "Batch" ; Logon_Type=5 | Logon_Type := "Service" ; Logon_Type=7 | Logon_Type :=
"Unlock"; Logon_Type=8 | Logon_Type := "Net_Clear"; Logon_Type=9 | Logon_Type := "Run_As"; Logon_Type=10 | Logon_Type :=
"RDP"; Logon_Type=11 | Logon_Type := "Cache_Int"; Logon_Type=12 | Logon_Type := "Cache_Rem"; Logon_Type=13 |
Logon_Type := "Cache_Unlock"}
| alt{ Reason=%%2313 | Reason := "Unknown user name or bad password"}
| table([winlog.event.code, @timestamp, host.name, UserName, Domain, Type, Logon_Type, Targ_Dom, Target_User, Targ_User,
Auth_Type, IP_Address, Port, LMPackage, Logon_Process, PID, Process_Name, Reason, error.message])
```

4. Monitor for Administrative and Guest Logon Failures: Threat actors and malware often script and try to brute force known accounts, such as Administrator and Guest. Add this to the previous query(s) to alert on just the administrator and guest accounts, or any Honey Accounts mentioned prior. Adding a count can detect brute force or script attempts (>5 what exceeds your lockout setting).

5. Local Administrator Password Solution (LAPS): Threat actors often try, harvest and crack local administrator accounts to use against other systems in the environment. By using LAPS you can create unique passwords for every administrator account which would then make a failed logon attempt for and administrator account more valuable. If logons to the administrator are attempted across multiple systems or towards an important server such as a Domain Controller or Data Server where confidential data is stored, then this can be a very valuable query.

WINDOWS LOGSCALE LOGGING CHEAT SHEET - Win 7 – Win2022

MONITOR FOR FILE SHARES – SECURITY LOG - 5140:

1. Monitor for File Shares being accessed: Once a system is compromised, threat actors will connect or jump to other systems to infect and/or to steal data. Watch for accounts crawling across file shares. Some management accounts will do this normally so exclude these to the systems they normally connect. Other activity from management accounts such as new processes launching will alert you to malicious behavior when excluded in this alert. JP CERT has a great document on how to detect lateral movement from log data here:
    - https://blogs.jpcert.or.jp/en/2017/12/research-report-released-detecting-lateral-movement-through-tracking-event-logs-version-2.html

SAMPLE QUERY: A network share was accessed

```
winlog.channel=Security event.code=5140 // Monitor file shares v7
| Src_IP:= winlog.event_data.IpAddress | Src_Port:=winlog.event_data.IpPort | PID:= winlog.process.id | UserName:=
winlog.event_data.SubjectUserName | Domain:=winlog.event_data.SubjectDomainName | Path:= winlog.log.event_data.ShareLocalPath
| Object:=winlog.event_data.ObjectType | Share:=winlog.event_data.ShareName | Share_Local_Path:=winlog.event_data.ShareLocalPath
| Target:=winlog.event_data.RelativeTargetName | Access_List:=winlog.event_data.AccessList | Mask:=winlog.event_data.AccessMask
| table([event.code, @timestamp, host.name, Src_IP, Src_Port, PID, UserName, Domain, Object, task, Share_Local_Path, Path, Share, Mask,
Access_List])
 | NOT Src_IP=::1
```

SAMPLE QUERY: A network share was accessed – shows details of folders and files accessed

```
winlog.channel=Security event.code=5145 // Monitor Detailed File Shares v7
| Src_IP:= winlog.event_data.IpAddress | PID:= winlog.process.pid | UserName:=winlog.event_data.SubjectUserName | Domain:=
winlog.event_data.SubjectDomainName | Target_File:= winlog.event_data.RelativeTargetName | Path:=
winlog.event_data.ShareLocalPath | Share:= winlog.event_data.ShareName | ObjectType:=winlog.event_data.ObjectType
| table([event.code, @timestamp, host.name, Status, Src_IP, PID, UserName, Domain, task, Path, Share, Target_File, ObjectType])
```

2. Folder/File Auditing vs Detailed Shares: Event ID 5140 will only show you the share that a user accessed, not the folders or files. Event ID 5145 shows the share, folders, and files a user accesses to provide much needed details of shares that contain confidential information. You may choose to collect 5140 on systems with systems of little value, such as end user laptops and workstations, maybe some servers. If you want to see who accesses folders and files where data that needs protecting and monitoring resides, then collecting 5145 is the better bet, but more noisy volume wise.

   You can also choose to enable folder auditing on a root folder and all sub-folders where data that needs monitoring. This would allow you to cherry pick folders that need monitoring versus an entire share which can result in less data, but you will need to set folder auditing on each folder you wish to monitor and will be found in Event ID 4663 events. You can see and example of folder auditing later in this cheat sheet. Refer to the "***Windows File Auditing Cheat Sheet***" for more details on File/Folder auditing which can be obtained here:
    - **https://www.malwarearchaeology.com/cheat-sheets**

MONITOR FOR SERVICE INSTALLS AND START/STOP – SYSTEM LOG – 7040 & 7045 & SECURITY LOG 4697:

1. Monitor for New Service Installs: Monitoring for a new service install is crucial. Threat actors often use a new service to gain elevated persistence for their malware when a system restarts. Services can load DLLs and Drivers which are not logged elsewhere. This is for the System Log.

SAMPLE QUERY:

```
winlog.channel=System event.code="7045" // New Service v7
| Service_FileName:=winlog.event_data.ServiceFileName | Service_Name:=winlog.event_data.ServiceName |
Serv_Type:=winlog.event_data.ServiceType | Domain:=winlog.user.domain | UserName:=winlog.user.name |
Start_Type:=winlog.event_data.ServiceStartType | Start_Type:=winlog.event_data.StartType | Service_FileName:=winlog.event_data.ImagePath
| Acct_Name:=winlog.event_data.AccountName
| table([event.code, @timestamp, host.name, UserName, Domain, Service_Name, Service_FileName, Start_Type, Serv_Type, Acct_Name])
```

2. Monitor for New Service Installs: Monitoring for a new service install is crucial. Threat actors often use a new service to gain elevated persistence for their malware when a system restarts. Services can load DLLs and Drivers which are not logged elsewhere. This is for the Security Log.

```
winlog.channel=Security event.code="4697" // New Service from Security Log v7
| Service_FileName:=winlog.event_data.ServiceFileName | Service_Name:=winlog.event_data.ServiceName |
Serv_Type:=winlog.event_data.ServiceType | Domain:=winlog.event_data.SubjectDomainName |
UserName:=winlog.event_data.SubjectUserName | Start_Type:=winlog.event_data.ServiceStartType | Start_Type:=winlog.event_data.StartType
| Service_FileName:=winlog.event_data.ImagePath | Acct_Name:=winlog.event_data.ServiceAccount | PID:=winlog.process.pid
| alt{ Start_Type=2 | Start_Type := "Automatic" ; Start_Type=3 | Start_Type := "Manual" ; Start_Type=1 | Start_Type := "System" ; Start_Type=4 |
Start_Type := "Disabled" }
| alt{ Serv_Type=0xe0 | Serv_Type := "User" ; Serv_Type=0x1 | Serv_Type := "Kernel" ; Serv_Type=0xd0 | Serv_Type := "Individual" ;
Serv_Type=0x10 | Serv_Type := "Win32" }
| table([event.code, @timestamp, host.name, UserName, Domain, Start_Type, Serv_Type, Acct_Name, PID, Service_Name, Service_FileName])
```

3. Monitor Service Starts and Stops: Monitoring for service state changes can show when a service is stopping and starting, maybe due to a value change. This normally happens a lot, so false positives will need to be filtered, or just make this a query and not an alert. Only Windows based services will be seen as 3rd party services will not show up in this log. You would need to implement logging from the "***Windows Advanced Logging Cheat Sheet***" to log 3rd party services starts and stops another way, and is more noisy as it produces more data.

SAMPLE QUERY:

```
winlog.channel=System AND (event.code=7036 OR event.code=7040) // Service Start and Stop v7
| PID:=winlog.process.pid | Service:=winlog.event_data.param1 | State:=winlog.event_data.param2 | Svc_Type:=winlog.keywords[0]
|UserName:=winlog.user.name | Domain:=winlog.user.domain
| table([event.code, @timestamp, host.name, UserName, Domain, PID, Service, State, Svc_Type, @rawstring])
| NOT Service="Background Intelligent Transfer Service"
```

4. Monitor for Service Keys for State Changes: Monitoring for service state changes can show when a service is altered. Threat actors often use an existing service to avoid new service detection and modify the ***ServiceDll*** to point to a malicious payload gaining persistence for their malware when a system restarts. Unfortunately, these details are not in the logs, but this alert can lead you to look into a service state change or enable auditing on keys that trigger seldom used services to watch for ***ServiceDll*** changes. There are a few services that will normally start and stop regularly and will need to be excluded. Use registry auditing (4657) to monitor for changes to the ***ServiceDll*** value.

Monitoring for Folders and Registry changes can be a powerful tool. Auditing for users directories and keys must be setup after the user is created, so your process would need to take this into account to set the auditing on folders and keys.

MONITOR FOR FILE CHANGES – SECURITY LOG - 4663:

1. Monitor for New files: This requires directories and/or files to have auditing set on each object. You want to audit directories that are well known for malware such as AppData\Local, AppData\LocalLow & AppData\Roaming as well as \Users\Public or ProgramData. Refer to the "***Windows File Auditing Cheat Sheet***" for more details on File/Folder auditing which can be obtained here:
    - **https://www.malwarearchaeology.com/cheat-sheets**

SAMPLE QUERY: Monitor for new files and folders being created

```
winlog.channel=Security event.code=4663 AND host.name=SURFER AND event.action="File System"// File Folder Auditing v7
| UserName:= winlog.event_data.SubjectUserName | Process_Name:= winlog.event_data.ProcessName | Object_Name:=
winlog.event_data.ObjectName | Access_Mask:= winlog.event_data.AccessMask | PID_Hex:= winlog.event_data.ProcessId
| alt{ Access_Mask=0x10000 | Accesses := "Delete" ; Access_Mask=0x2 | Accesses := "WriteData" ; Access_Mask=0x6 | Accesses :=
"AppendData - WriteData" ; Access_Mask=0x4 | Accesses := "AppendData" }
| table([event.code, @timestamp, host.name, UserName, Accesses, PID_Hex, Process_Name, Object_Name, Accesses])
```

MONITOR FOR REGISTRY CHANGES – 4657:

2. Monitor for Registry Changes: Adding auditing to known exploited registry keys is a great way to catch malicious activity. These registry keys should not change very often unless something is installed or updated. The goal is to look for NEW items and changes to known high risk items like the Run and RunOnce keys. Refer to the "***Windows Registry Auditing Cheat Sheet***" for more details on Registry key auditing which can be obtained here:
    - **https://www.malwarearchaeology.com/cheat-sheets**

SAMPLE QUERY: Monitor for new keys or values being added

```
winlog.channel=Security event.code=4657 AND winlog.event_data.ObjectName=*Run* // Registry Changes to audited keys v7
| Handle_ID:= winlog.event_data.HandleId | UserName:= winlog.event_data.SubjectUserName | Domain:=
winlog.event_data.SubjectDomainName | Process_Name:= winlog.event_data.ProcessName | Object_Name:= winlog.event_data.ObjectName |
Value_Name:= winlog.event_data.ObjectValueName
| New_Value:= winlog.event_data.NewValue | New_Value_Type:= winlog.event_data.NewValueType | Old_Value:= winlog.event_data.OldValue
| Old_Value_Type:= winlog.event_data.OldValueType | Object_Type:= winlog.event_data.ObjectType | Access_List:=
winlog.event_data.AccessList | Operation_Type:= winlog.event_data.OperationType | PID:= winlog.event_data.ProcessId
| alt{ Operation_Type=%%1904 | Operation_Type := "New Value Created" ; Operation_Type=%%1906 | Operation_Type := "Value Deleted" ;
Operation_Type=%%1905 | Operation_Type := "Existing Value Modified"}
| alt{ New_Value_Type=%%1872 | New_Value_Type := "Reg_None" ; New_Value_Type=%%1873 | New_Value_Type := "Reg_Sz" ;
New_Value_Type=%%1874 | New_Value_Type := "Reg_Expand_Sz" ; New_Value_Type=%%1875 | New_Value_Type := "Reg_Binary" ;
New_Value_Type=%%1876 | New_Value_Type := "Reg_DWord" ; New_Value_Type=%%1879 | New_Value_Type := "Reg_Multi_Sz" ;
New_Value_Type=%%1883 | New_Value_Type := "Reg_QWord"}
| table([event.code, @timestamp, host.name, UserName, Domain, PID, Handle_ID, Process_Name, Value_Name, Object_Name, Old_Value,
Old_Value_Type, New_Value, New_Value_Type, Object_Type, Operation_Type, Access_List])
| NOT Object_Name=*Services\W32Time\SecureTimeLimits\RunTime*
```

MONITOR FOR WINDOWS FIREWALL CHANGES – FIREWALL LOG - 2004 & 2005:

1. Monitor for Additions to Firewall Rules: Malware and threat actors will often add a firewall rule to allow access to some Windows service or application. This log is not in the standard Windows logs and will need to be added to your Winlogbeat.yml file in order to collect them. The Windows firewall logs may be found under:
    - Applications and Services Logs – Microsoft - Windows – Windows Firewall with Advanced Security - Firewall

SAMPLE QUERY: Monitor for new rules being added

```
winlog.channel="Microsoft-Windows-Windows Firewall With Advanced Security/Firewall" (event.code="2004" or event.code="2005" or event.code="2006") // Firewall Rule Changed v7
| Action:=winlog.event_data.Action | Active:=winlog.event_data.Active | Direction:=winlog.event_data.Direction | Traversal:=winlog.event_data.EdgeTraversal | PID:=winlog.process.pid | User_Name:=winlog.user.name | Domain:=winlog.user.domain | Context:=winlog.event_data.EmbeddedContext | Local_IP:=winlog.event_data.LocalAddresses | Mod_App:=winlog.event_data.ModifyingApplication | Rem_IP:=winlog.event_data.RemoteAddresses | Profiles:=winlog.event_data.Profiles | User_GUID:=winlog.user.identifier | Rule_Name:=winlog.event_data.RuleName
| table([event.code, @timestamp, host.name, User_Name, User_GUID, Domain, Action, Active, Direction, Traversal, Profiles, Rule_Name, PID, Context, Local_IP, Rem_IP, Mod_App, @rawstring ]) | sort(@timestamp,reverse=true)
| not Rule_Name="*Microsoft.BingWeather*" | not Rule_Name="*Microsoft.YourPhone*" | not Rule_Name="*Microsoft.GetHelp*"
```

MONITOR FOR SCHEDULED TASK CHANGES – TASKSCHEDULER LOG – (106 Add, 141 Delete, 110 Started):

1. Monitor for Additions to Scheduled Tasks: Malware and hackers will often add a scheduled task to elevate privileges and start their malware. This log is disabled by default, is not in the standard Windows logs, and will need to be enabled and added to your Winlogbeat.yml file in order to collect into Logscale. The TaskScheduler log may be found under:
    - Applications and Services Logs – Microsoft - Windows – TaskScheduler - Operational

SAMPLE QUERY: Monitor for scheduled task being registered and deleted

```
winlog.channel="Microsoft-Windows-TaskScheduler/Operational" and (event.code="106" or event.code="141") // Sched Task Registered and deleted v7
| UserContext:=winlog.event_data.UserContext | Task_Name:=winlog.event_data.TaskName | PID:=winlog.process.pid | User_Name:=winlog.user.name | Domain:=winlog.event_data.SubjectDomainName | Status:=winlog.keywords[0] | Process:=winlog.event_data.Path
| table([event.code, @timestamp, host.name, Status, event.action, User_Name, Domain, UserContext, Process, Task, event_data.ResultCode, PID, Task_Name, @rawstring ]) | sort(@timestamp,reverse=true)
| NOT Task_Name=\NahimicTask64 | NOT Task_Name=\NahimicTask32
```

2. Monitor for scheduled task being started: Once a task is created it will need to start. Threat actors will often create, start, then delete a task to infect a system, or use a task as persistence to infect a system every hour or on login. Some false positive updates will have this behavior, Chrome, OneDrive, etc. filter these out with a NOT statement or File match().

SAMPLE QUERY: Monitor for existing scheduled task being started

```
winlog.channel="Microsoft-Windows-TaskScheduler/Operational" and (event.code="110") // Sched Task Started v7
| UserContext:=winlog.event_data.UserContext | Task_Name:=winlog.event_data.TaskName | PID:=winlog.process.pid | User_Name:=winlog.user.name | Domain:=winlog.event_data.SubjectDomainName | Status:=winlog.keywords[0] | Process:=winlog.event_data.Path
| table([event.code, @timestamp, host.name, Status, event.action, User_Name, Domain, UserContext, Process, Task, event_data.ResultCode, PID, Task_Name, @rawstring ]) | sort(@timestamp,reverse=true)
| NOT Task_Name=\Microsoft\Windows\CertificateServicesClient* | NOT Task_Name=\NahimicTask32 | NOT Task_Name=\NahimicTask64
```

PowerShell is used more and more to download malicious payloads, hide malicious code, and obfuscate code to make malicious PowerShell activity harder to detect. The following examples are provided to help build queries and alerts to detect malicious PowerShell behavior.

MONITOR FOR POWERSHELL BYPASS – 4688, 400, 500, 600, 800, and 4104:

1. Monitor for PowerShell bypass attempts: Threat actors will often use PowerShell bypasses to exploit a system to avoid using built-in utilities and dropping additional malware files on disk. Watching for policy, hidden, profile and execution policy bypasses will allow you to detect this potential hacking activity. Though there are various ways to spell some of these bypasses the most common ones are effective to monitor for. Adjust your queries as new techniques are discovered. The following article discusses research that evaluated malicious PowerShell activity:
    - https://unit42.paloaltonetworks.com/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/

SAMPLE QUERY: Catch bypass attempt with Security Log Event ID 4688

```
winlog.channel=Security event.code=4688 AND (/powershell/i and /bypass/i) or (/powershell/i and /-ep/i or /-exec/i) or (/powershell/i and /-nop/i) or (/powershell/i and /hidden/i) or (/pwsh/i and /bypass/i) or (/pwsh/i and /-exp/i) or (/pwsh/i and /-nop/i) or (/pwsh/i and /hidden/i) // PowerShell bypasses v7
| UserName:=winlog.event_data.SubjectUserName | Command_Line:= winlog.event_data.CommandLine | New_Process_Name:= winlog.event_data.NewProcessName
| table([event.code, @timestamp, host.name, UserName, New_Process_Name, Command_Line])
```

SAMPLE QUERY: Catch bypass attempt with a Windows PowerShell (PS v2) Event ID 400, 500, 600, or 800 (Legacy)

```
event.provider=PowerShell AND event.code=800 AND (/powershell/i and /bypass/i) or (/powershell/i and /-ep/i or /-exec/i) or (/powershell/i and /-nop/i) or (/powershell/i and /hidden/i) or (/pwsh/i and /bypass/i) or (/pwsh/i and /-exp/i) or (/pwsh/i and /-nop/i) or (/pwsh/i and /hidden/i) //Legacy PowerShell bypasses v7
| regex(regex="UserId=(?<User>[^=]*)\\n\\tHostName", field="winlog.event_data.param2")
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="winlog.event_data.param2")
| Command = /(?<Spl_Chars>[^a-zA-Z0-9 ^:^\\^\/^_^\"[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect([event.code, @timestamp, host.name, User, Spl_Chars, SplCnt, Command])) | length(Spl_Chars, as=SplCnt)
| table([event.code, @timestamp, host.name, User, Spl_Chars, SplCnt, Command]) | SplCnt > 0
```

SAMPLE QUERY: Catch bypass attempt with a PowerShell/Operational (PS v5) Event ID 4104

```
winlog.channel="Microsoft-Windows-PowerShell/Operational" AND (event.code=4104 OR event.code=4103) AND (/powershell/i and /bypass/i) or (/powershell/i and /-ep/i or /-exec/i) or (/powershell/i and /-nop/i) or (/powershell/i and /hidden/i) or (/pwsh/i and /bypass/i) or (/pwsh/i and /-exp/i) or (/pwsh/i and /-nop/i) or (/pwsh/i and /hidden/i) // PowerShell bypasses v7
| regex(regex="Host.Application.=(?<HostApp>.*\n)", field=winlog.event_data.ContextInfo)
| UserName:=winlog.user.name | Domain:=winlog.user.domain | ScriptBlock:=winlog.event_data.ScriptBlockText
| table([event.code, @timestamp, host.name, UserName, Domain, HostApp])
```

MONITOR FOR WINDOWS POWERSHELL WEB DOWNLOAD:

1. Monitor for PowerShell for Web Downloads: Threat actors will use PowerShell to download malicious content. Monitor for PowerShell downloads and filter out known good events.

SAMPLE QUERY: Catch PowerShell downloading content - Security Log - Event ID 4688

```
winlog.channel="Security" event.code=4688 ((/powershell/i or /pwsh/i) and (/webclient/i or /download/i or /http/i)) not \\wevtutil.exe //PowerShell Downloads v7
| UserName:= winlog.event_data.SubjectUserName | Command_Line:= winlog.event_data.CommandLine |
New_Process_Name:=winlog.event_data.NewProcessName
| table([event.code, @timestamp, host.name, UserName, Block, Command_Line, New_Process_Name])
```

SAMPLE QUERY: Catch PowerShell downloading content - PowerShell/Operational (PS v5) Event ID 4104 & 4104

```
winlog.channel="Microsoft-Windows-PowerShell/Operational" AND (event.code=4104 OR event.code=4103) AND
(winlog.event_data.ScriptBlockText=/webclient/i or winlog.event_data.ScriptBlockText=/download/i or winlog.event_data.ScriptBlockText=/http/i)
//PowerShell Downloads v7
| UserName:=winlog.user.name | Domain:=winlog.user.domain | ScriptBlock:=winlog.event_data.ScriptBlockText
| table([event.code, @timestamp, host.name, UserName, Domain, ScriptBlock])
```

SAMPLE QUERY: Catch PowerShell downloading content – Windows PowerShell Log - Event ID 400, 500, 600 or 800

```
event.provider=PowerShell AND event.code=800 AND (/webclient/i or /download/i or /http/i)//Legacy PowerShell Downloads v7
| regex(regex="UserId=(?<User>[^=]*)\\n\\tHostName", field=winlog.event_data.param2)
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="winlog.event_data.param2")
| Command = /(?<Spl_Chars>[^a-zA-Z0-9 ^:^\\^\/^_^\"[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect([event.code, @timestamp, host.name, User, Spl_Chars, SplCnt, Command])) | length(Spl_Chars, as=SplCnt)
| table([event.code, @timestamp, host.name, level, Spl_Chars, SplCnt, Command])
```

MONITOR FOR WINDOWS POWERSHELL BASE64 EVENTS:

1. Monitor for PowerShell Base64 execution with Security Log Event ID 4688: Threat actors will often use obfuscation of PowerShell code to hide what they are doing. Monitoring the Process Command Line executions can catch potentially malicious obfuscated PowerShell. The query below looks for a series of characters that indicate Base64 being used with Process Command Line logging enabled.

```
winlog.channel="Security" AND event.code=4688 /powershell/i and /enc/I // PowerShell Base64 v7
| CMD_Line:=winlog.event_data.CommandLine | New_Process:=winlog.event_data.NewProcessName |
Parent_Process:=winlog.event_data.ParentProcessName
| regex(regex="(?<Block>[a-zA-Z0-9]{25,100})", field="CMD_Line")
| table([event.code, @timestamp, host.name, event_data.SubjectUserName, Block, CMD_Line, New_Process, Parent_Process,
event_data.NewProcessName])
```

2. Monitor for large PowerShell ScriptBlocks with 4104: Threat actors will often trigger large script blocks of PowerShell as they execute their fu. There is plenty of normal large blocks as well, but if you filter these out, you can catch large blobs of script executing. The query below looks for the count of the script block over 1000 characters using in the "PowerShell/Operational" log. The "***PowerShell/Operational***" log may be found under:
    - Applications and Services Logs – PowerShell/Operational

```
winlog.channel="Microsoft-Windows-PowerShell/Operational" AND event.code=4104 // PowerShell Base64 v7
| Block_Text:=winlog.event_data.ScriptBlockText | PID:=winlog.process.pid | Task:=winlog.task | User_Name:=winlog.user.name |
Blk_Cnt:=winlog.event_data.MessageTotal | Blk_No:=winlog.event_data.MessageNumber
| regex(regex="(?<Block>[a-zA-Z0-9]{100,1000})", field="Block_Text")
| length(Block, as=Blk_Cnt)
| table([event.code, @timestamp, host.name, User_Name, Blk_No, Blk_Cnt, PID, Blk_Cnt, Block, Block_Text, Task]) | Blk_Cnt > 10
```

MONITOR FOR WINDOWS POWERSHELL OBFUSCATION - TICKS AND SPECIAL CHARACTERS:

1. Monitor for PowerShell Obfuscation with 4688: Threat actors will often use obfuscation of PowerShell code to hide what they are doing. Monitoring the Process Command Line executions can catch potentially malicious obfuscated PowerShell. The query below looks for and counts the amount of ticks, semicolons, and dollar signs to detect the use of PowerShell obfuscation using the Security log and Process Execution 4688 events with Process Command Line logging enabled.

SAMPLE QUERY: PowerShell Obfuscation Security Logs

```
winlog.channel=Security AND event.code=4688 AND winlog.event_data.NewProcessName=*powershell* // Process Obfuscation v7
| User_Name:=winlog.event_data.SubjectUserName | Tar_User:=winlog.event_data.TargetUserName |
Tar_Dom:=winlog.event_data.TargetDomainName | CMD_Line_Obf:=winlog.event_data.CommandLine |
CMD_Line:=winlog.event_data.CommandLine | Process_Name:=winlog.event_data.NewProcessName
| CMD_Line_Obf = /(?<Spl_Chars>[^a-zA-Z0-9 ^:^\\^\/^_^\"[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect(fields=[event.code, @timestamp, host.name, User_Name, Tar_user, Tar_Dom, level, Process_Name, CMD_Line,
Spl_Chars, SplCnt])) | length(Spl_Chars, as=SplCnt)
| table([event.code, @timestamp, host.name, User_Name, Tar_User, Tar_Dom, Process_Name, level, SplCnt, CMD_Line]) | SplCnt > 10 |
sort(@timestamp,reverse=true)
```

2. Monitor for PowerShell Obfuscation with 200, 400, 500, 600 or 800: Start with ID 800 as it contains the username. Hackers will often use obfuscation of PowerShell code to hide what they are doing. Monitoring the PowerShell executions can catch potentially malicious obfuscated PowerShell. The query below looks for and counts the amount of ticks, carats, semicolons, and dollar signs to detect the use of PowerShell obfuscation using the "Windows PowerShell" log (v2-v5) events. The "***Windows PowerShell***" logs may be found under:
    - Applications and Services Logs - Windows PowerShell

SAMPLE QUERY: PowerShell Obfuscation Legacy Logs

```
event.provider=PowerShell AND event.code=800 //Legacy PowerShell Obfuscation v7
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="winlog.event_data.param2")
| Command = /(?<Spl_Chars>[^a-zA-Z0-9 ^:^\\^\/^_^\"[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect([event.code, @timestamp, host.name, level, Spl_Chars, SplCnt, Command])) | length(Spl_Chars, as=SplCnt)
| table([event.code, @timestamp, host.name, level, Spl_Chars, SplCnt, Command]) | SplCnt > 10
```

3. Monitor for PowerShell Obfuscation with 4104: Threat actors will often use obfuscation of PowerShell code to hide what they are doing. Monitoring the PowerShell executions can catch potentially malicious obfuscated PowerShell. The query below looks for and counts the amount of ticks, carats, semicolons, and dollar signs to detect the use of PowerShell obfuscation using the "PowerShell/Operational" log (v5) events. The ***PowerShell/Operational***" logs may be found under:
    - Applications and Services Logs - Windows PowerShell

SAMPLE QUERY: PowerShell obfuscation, noisier than other event IDs, excluding normal commands will be needed - PowerShell/Operational (PS v5) Event ID 4104

```
winlog.channel="Microsoft-Windows-PowerShell/Operational" AND event.code=4104 // Powershell Obfuscation Gr 20 AND ("*-
encodedcommand*" OR "*iEx*" OR "*nsadasd*") // PowerShell Obfuscation v7
| Block_Text:=winlog.event_data.ScriptBlockText | Block_Text_Obf:=winlog.event_data.ScriptBlockText | PID:=winlog.process.pid |
Task:=winlog.task | User_Name:=winlog.user.name | User_Type:=winlog.user.type | Blk_Cnt:=winlog.event_data.MessageTotal |
Blk_No:=winlog.event_data.MessageNumber
| Block_Text_Obf = /(?<Spl_Chars>[^a-zA-Z0-9 ^:^\\^\/^_^\"[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect(fields=[event.code, @timestamp, host.name, User_Name, User_Type, Blk_No, Blk_Cnt, PID, Task, Process_Name,
Block_Text, Spl_Chars, SplCnt])) | length(Spl_Chars, as=SplCnt) | length(Block_Text, as=Txt_Cnt)
| table([event.code, @timestamp, host.name, User_Name, User_Type, PID, Blk_No, Blk_Cnt, SplCnt, Txt_Cnt, Task, Block_Text]) | SplCnt > 20
| NOT Block_Text="*www.MalwareArchaeology.com*"
```