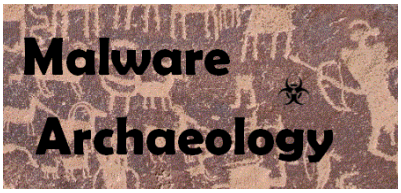
WINDOWS SPLUNK LOGGING CHEAT SHEET - Win 7 - Win2012

This "**Windows Splunk Logging Cheat Sheet**" is intended to help you get started setting up Splunk reports and alerts for the most 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 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.



Sponsored by:

LOG–MD
Discover it

DEFINITIONS:

WINDOWS LOGGING CONFIGURATION: Before you can Gather anything meaningful with Splunk, 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 Splunk. 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 program. The "***Windows Logging Cheat Sheet***" contains the details needed for proper and complete security logging to understand how to Enable and Configure Windows logging and auditing 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: Queries that are saved for reference and can be launched as needed.

ALERTS: Queries you want to be emailed on or sent to your smartphone to alert you that something is outside the norm and needs to be looked at immediately. Do not get alert heavy or your staff will ignore them as was the case in the Target and Neiman Marcus breaches.

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.

RESOURCES: Places to get more information.

- MalwareArchaeology.com/cheat-sheets – More Windows Logging Cheat Sheets and resources
- Better descriptions of Event ID's
  - www.ultimatewindowssecurity.com/securitylog/encyclopedia/Default.aspx
- www.EventID.Net – Extensive list of Event ID's
- www.CISecurity.org - Center for Internet Security Benchmarks
- Google – Of course
- Splunk.com – Endless information on Splunk
- Auditing the Registry with Splunk UF
  - https://docs.splunk.com/Documentation/Splunk/6.6.2/Data/MonitorWindowsregistrydata

WINDOWS SPLUNK LOGGING CHEAT SHEET - Win 7 - Win2012

CRITICAL EVENTS TO MONITOR:

1. NEW PROCESS STARTING: Event Code 4688 will capture when a process or executable starts.
2. USER LOGON SUCCESS: Event Code 4624 will capture when a user successfully logons to the system.
3. SHARE ACCESSED: Event Code 5140 will capture when a user connects to a file share.
4. NEW SERVICE INSTALLED: Event Code 7045 will capture when a new service is installed.
5. 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
6. FILE AUDITING: Event Code 4663 will capture when a new file is added, modified or deleted.
7. REGISTRY AUDITING: Event Code 4657 will capture when a new registry item is added, modified or deleted
8. WINDOWS POWERSHELL COMMAND LINE EXECUTION: Event Code 500 will capture when PowerShell is executed logging the command line used.
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.

FILTERING EVENTS:

1. Filter by Message, NOT by Event Code: It is common to blacklist event codes that are noisy or excessive that impacts storage and licensing. By enabling Process Creation Success (4688) Process Terminate (4689) and Windows Firewall Filtering Platform Connection Success (5156 & 5158) they will be the top four event codes in your Splunk index. Filtering by the content of the Message or Field name is the better way to go. Once you understand what normal noise is, has minimal risk to be exploited or important to security monitoring you can filter those out at the client or server. For Windows, Splunk limits the blacklist to only 10 entries, so you will need to chain similar events in one line. Here is an example of a proper exclusion:

```
[WinEventLog://Security]
disabled=0
current_only=1
blacklist = 4689,5158
blacklist1 = EventCode="4688" Message="(?:New Process Name:).+(?:SplunkUniversalForwarder\\bin\\splunk.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunkd.exe)|.+(?:SplunkUniversalForwarder\\bin\\btool.exe)"
blacklist2 = EventCode="4688" Message="(?:New Process Name:).+(?:SplunkUniversalForwarder\\bin\\splunk-winprintmon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-powershell.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-regmon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-netmon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-admon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-MonitorNoHandle.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-winevtlog.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-perfmon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-wmi.exe)"
blacklist3 = EventCode="4688" Message="(?:Process Command Line:).+(?:--scheme)|.+(?:--no-log)|.+(?:-Embedding)"
blacklist4 = EventCode="4688" Message="(?:Process Command Line:).+(?:system32\\SearchFilterHost.exe)|.+(?:find /i)|.+(?:Google\\Update\\GoogleUpdate.exe)|.+(?:WINDOWS\\system32\\conhost.exe)"
blacklist5 = EventCode="5156" Message="(?:Application Name:).+(?:splunkuniversalforwarder\\bin\\splunkd.exe)|.+(?:mcafee\\common framework\\masvc.exe) |.+(?:appdata\\local\\citrix\\gotomeeting\\????\\g2mcomm.exe)"
blacklist6 = EventCode="5156" Message="(?:Source Address:)(?s).+(?:224.0.0.251)|(?s).+(?:239.255.255.250)"
blacklist7 = EventCode="5156" Message="(?:Application Name:).+(?:google\\chrome\\application\\chrome.exe)"
blacklist8 = EventCode="4659" Message="(?:Object Name:)(?s).*(AppData\\Local\\Temp\\etilqs_)"
blacklist9 = EventCode="4663" Message="(?:Object Name:)(?s).*(\\REGISTRY\\)"
```

2. Indexes: It is normal to have an index named “windows” for the typical Application, Security, Setup and System logs, but separating other Windows logs into separate indexes is a good practice to reduce search times. Consider collecting the Windows PowerShell, TaskScheduler, Windows Firewall, AppLocker and other Applications and Services logs that you might want to collect into their own indexes if they get large in quantity of events. You can always use the “***Join***” command if you want to combine data from multiple indexes.

MONITOR FOR PROCESSES STARTING - 4688:

1. Monitor for Suspicious/Administrative Processes: This list is based on built-in Windows administrative utilities and known hacking utilities that are often seen used in exploitation. Expand this list as needed to add utilities used in hacking attacks. You do not need to alert on all processes launching, just suspicious ones or ones known to be used in hacking attacks. Some administrative tools are very noisy and normally used or automatically executed regularly and should NOT be included to make your alert more actionable and accurate that something suspicious has occurred.

SAMPLE QUERY:

```
index=windows LogName=Security EventCode=4688 NOT (Account_Name=*$) (arp.exe OR at.exe OR bcdedit.exe OR bcp.exe OR chcp.exe OR cmd.exe OR cscript.exe OR csvde OR dsquery.exe OR ipconfig.exe OR mimikatz.exe OR nbtstat.exe OR nc.exe OR netcat.exe OR netstat.exe OR nmap OR nslookup.exe OR netsh OR OSQL.exe OR ping.exe OR powershell.exe OR powercat.ps1 OR psexec.exe OR psexecsvc.exe OR psLoggedOn.exe OR procdump.exe OR qprocess.exe OR query.exe OR rar.exe OR reg.exe OR route.exe OR runas.exe OR rundll32 OR schtasks.exe OR sethc.exe OR sqlcmd.exe OR sc.exe OR ssh.exe OR sysprep.exe OR systeminfo.exe OR system32\\net.exe OR reg.exe OR tasklist.exe OR tracert.exe OR vssadmin.exe OR whoami.exe OR winrar.exe OR wscript.exe OR "winrm.*" OR "winrs.*" OR wmic.exe OR wsmprovhost.exe OR wusa.exe) | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host, Account_Name, Process_Name, Process_ID, Process_Command_Line, New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message
```

SAMPLE QUERY: Trigger alert on 4th command executed (Best alert to catch malwarians on your system)

```
(index=win_servers OR index=win_workstations) LogName=Security EventCode=4688 [ | inputlookup InfoSec_Admin_Utils.csv | fields New_Process_Name ] NOT (Some_server_name OR some_server_ip) NOT (Account_Name="-" OR Account_Name="*$") NOT (Process_Command_Line="some_command_you_trust" OR "some_other_cmd_you_trust") | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | replace Server_Name with Descriptive_Name in host | stats count values(host) AS Host values(Process_Command_Line) AS CMD_Line, values(New_Process_Name) AS New_Process_Name, values(Creator_Process_ID) AS Creator_Process_ID, values(New_Process_ID) AS New_Process_ID, values(Short_Message) AS Status by Account_Name | where NOT isnull(CMD_Line) | where count > 3
```

2. Monitor for Whitelisting bypass attempts: Hackers will often use PowerShell to exploit a system due to the capability of PowerShell to avoid using built-in utilities and dropping additional malware files on disk. Watching for policy and profile bypasses will allow you to detect this hacking activity.

SAMPLE QUERY:

```
index=windows LogName=Security (EventCode=4688) NOT (Account_Name="Something_good") (iexec.exe OR InstallUtil.exe OR Regsrv32.exe OR Regasm.exe OR Regsvcs.exe OR MSBuild.exe) | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host, Account_Name, Process_Name, Process_ID, Process_Command_Line, New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message
```

MONITOR FOR PROCESSES STARTING - 4688:

3. Monitor for PowerShell bypass attempts: Hackers will often use PowerShell to exploit a system due to the capability of PowerShell to avoid using built-in utilities and dropping additional malware files on disk. Watching for policy and profile bypasses will allow you to detect this hacking activity.

SAMPLE QUERY:

index=windows EventCode=4688 (powershell* AND (–ExecutionPolicy OR –Exp)) OR (powershell* AND bypass) OR (powershell* AND (-noprofile OR -nop)) | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host, Account_Name, Process_Name, Process_ID, Process_Command_Line, New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message

4. Monitor for all processes excluding trusted/known processes: You can create reports for any or all processes starting (4688) and filter out the known good ones to create a more actionable report and alert. For larger lists consider using the "***lookup***" command. Your .csv file has to be in a '***lookups***' directory in either the parent or a child local directory; /opt/splunk/etc/apps/search/lookups. The idea here is a typical system has a normal state, if you exclude all the normal processes, then if something new runs, say BlackPOS.exe as was the case in the retail breaches, you would be able to detect it.

SAMPLE QUERY:

index=windows LogName=Security EventCode=4688 NOT (Account_Name=*$) | NOT [inputlookup Trusted_processes.csv | fields Process_Name] | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host, Account_Name, Process_Name, Process_ID, Process_Command_Line, New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message

MONITOR FOR USER LOGONS – 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 and system crawling. This alert looks for successful logons > 2 and excludes domain controllers to detect when a rogue user account crawls across systems in your network.

SAMPLE QUERY:

```
index=windows LogName=Security EventCode=4624 NOT (host="DC1" OR host="DC2" OR host="DC…") NOT (Account_Name="*$" OR Account_Name="ANONYMOUS LOGON") NOT (Account_Name="Service_Account") | eval Account_Domain=(mvindex(Account_Domain,1)) | eval Account_Name=if(Account_Name="-",(mvindex(Account_Name,1)), Account_Name) | eval Account_Name=if(Account_Name="*$",(mvindex(Account_Name,1)), Account_Name) | eval Time=strftime(_time,"%Y/%m/%d %T") | stats count values(Account_Domain) AS Domain, values(host) AS Host, dc(host) AS Host_Count, values(Logon_Type) AS Logon_Type, values(Workstation_Name) AS WS_Name, values(Source_Network_Address) AS Source_IP, values(Process_Name) AS Process_Name by Account_Name | where Host_Count > 2
```

2. Monitor for Logon Failures: Watch for excessive logon failures, especially Internet facing systems and systems that contain confidential data. This will also detect brute force attempts and users who have failed to changed their passwords on additional devices such as smartphones. You can add "***stats 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.

SAMPLE QUERY:

```
index=windows LogName=Security EventCode=4625 | table _time, Workstation_Name, Source_Network_Address, host, Account_Name
```

3. Monitor for Administrative and Guest Logon Failures: Hackers and malware often try to brute force known accounts, such as Administrator and Guest. This alert will monitor and alert if configured for attempts > 5.

SAMPLE QUERY:

```
index=windows LogName=Security EventCode=4625 (Account_Name=administrator OR Account_Name=guest) | stats count values(Workstation_Name) AS Workstation_Name, Values(Source_Network_Address) AS Source_IP_Address, values(host) AS Host by Account_Name | where count > 5
```

MONITOR FOR FILE SHARES - 5140:

1. Monitor for File Shares being accessed: Once a system is compromised, hackers 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.

SAMPLE QUERY:

```
index=windows source="WinEventLog:Security" EventCode=5140 (Share_Name="*\\C$" OR Share_Name="*D$" OR Share_Name="*E$" OR Share_Name="*F$" OR Share_Name="*U$") NOT Source_Address="::1" | eval Destination_Sys1=trim(host,"1") | eval Destination_Sys2=trim(host,"2") | eval Dest_Sys1=lower(Destination_Sys1) | eval Dest_Sys2=lower(Destination_Sys2) | rename host AS Destination | rename Account_Domain AS Domain | where Account_Name!=Dest_Sys1 | where Account_Name!=Dest_Sys2 | stats count values(Domain) AS Domain, values(Source_Address) AS Source_IP, values(Destination) AS Destination, dc(Destination) AS Dest_Count, values(Share_Name) AS Share_Name, values(Share_Path) AS Share_Path by Account_Name
```

MONITOR FOR SERVICE CHANGES – 7045 & 7040:

1. Monitor for New Service Installs: Monitoring for a new service install is crucial. Hackers often use a new service to gain persistence for their malware when a system restarts. All the retail Point of Sale breaches included one or more new services that could have been easily detected with this alert alone.

SAMPLE QUERY:

```
index=windows LogName=System EventCode=7045 NOT (Service_Name=mgmt_service) | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time host Service_Name, Service_Type, Service_Start_Type, Service_Account, Short_Message
```

2. Monitor for Service State Changes: Monitoring for a service state changes can show when a service is altered. Hackers 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 the 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.

SAMPLE QUERY:

```
index=windows LogName=System EventCode=7040 NOT (“*Windows Modules Installer service*” OR “*Background Intelligent Transfer Service service*”) | table _time, host, User, Message
```

MONITOR FOR NETWORK CONNECTIONS - 5156:

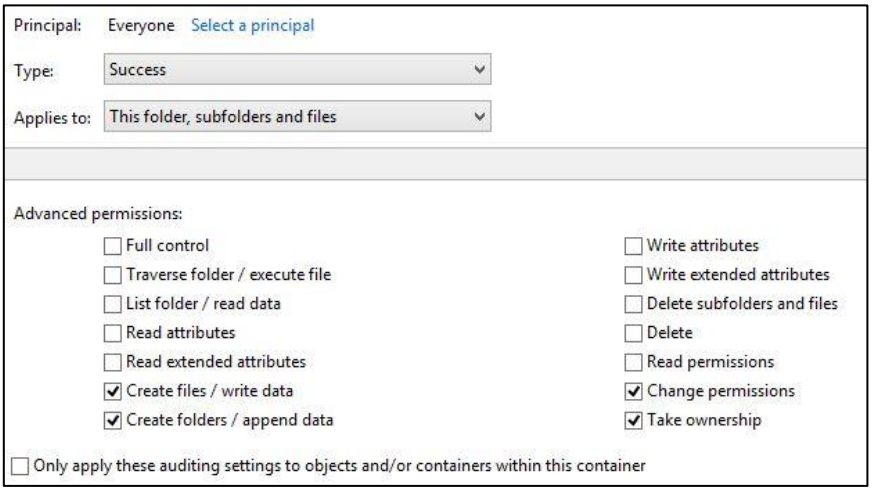
1. Monitor for Suspicious Network IP's: This does require the use of the Windows Firewall. In networks where this is normally not used, you can use Group Policy to set the Windows Firewall to an Any/Any configuration so no blocking occurs, yet the traffic is captured in the logs and more importantly what process made the connection. You can create exclusions by IP addresses (such as broadcast IP's) and by process names to reduce the output and make it more actionable. The "***Lookup***" command will benefit this query tremendously by excluding items.

SAMPLE QUERY:

```
index=windows LogName=Security EventCode=5156 NOT (Source_Address="239.255.255.250" OR Source_Address="224.0.0.*" OR Source_Address="::1" OR Source_Address="ff02::*" OR Source_Address="fe80::*" OR Source_Address="255.255.255.255" OR Source_Address=192.168.1.255) NOT (Destination_Address="127.0.0.1" OR Destination_Address="239.255.255.250" OR Destination_Address="*.*.*.255" OR Destination_Address="224.0.0.25*") NOT (Destination_Port="0") NOT (Application_Name="\\<some process name>\\" OR Application_Name="*\\bin\\splunkd.exe") | dedup Destination_Address Destination_Port | table _time, host, Application_Name, Direction, Source_Address, Source_Port, Destination_Address, Destination_Port | sort Direction Destination_Port
```

MONITOR FOR FILE CHANGES – 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, LocalLow & Roaming as well as \Users\Public for the following:

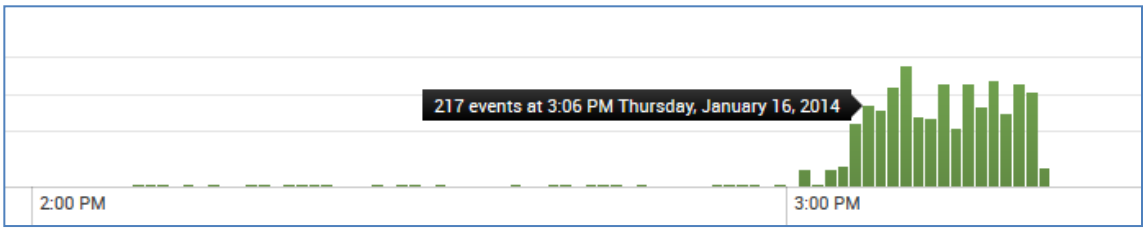


SAMPLE QUERY:

```
index=windows sourcetype=WinEventLog:Security EventCode=4663 NOT (Process_Name="*\\Windows\\servicing\\TrustedInstaller.exe" OR "*\\Windows\\System32\\poqexec.exe") NOT Object_Name="C:\\Users\\Surf\\AppData\\Local\\Google\\Chrome\\User Data*" NOT Object_Name="C:\\Users\\<special user>\\AppData\\Roaming\\Microsoft\\Windows\\Recent\\CustomDestinations") NOT (Object_Name="C:\\Windows\\System32\\LogFiles\\*" OR Object_Name="*ProgramData\\Microsoft\\RAC\\*" OR Object_Name="*\\Microsoft\\Windows\\Explorer\\thumbcache*" OR Object_Name="*.MAP" OR Object_Name="*counters.dat" OR Object_Name="*\\Windows\\Gatherlogs\\SystemIndex\\*") | rename Process_Name as Created_By | table _time, host, Security_ID, Handle_ID, Object_Type, Object_Name, Process_ID, Created_By, Accesses
```

MONITOR FOR FILE CHANGES – 4663 continued:

2. Monitor for Crypto events: Setting auditing on a File Server Share will allow large amounts of file changes from a crypto event to be detected. Look at a large quantity of changes > 1000 in 1 hour to detect the event. Use the same settings as above as you only need to monitor for NEW files. It is obvious when an event occurs!

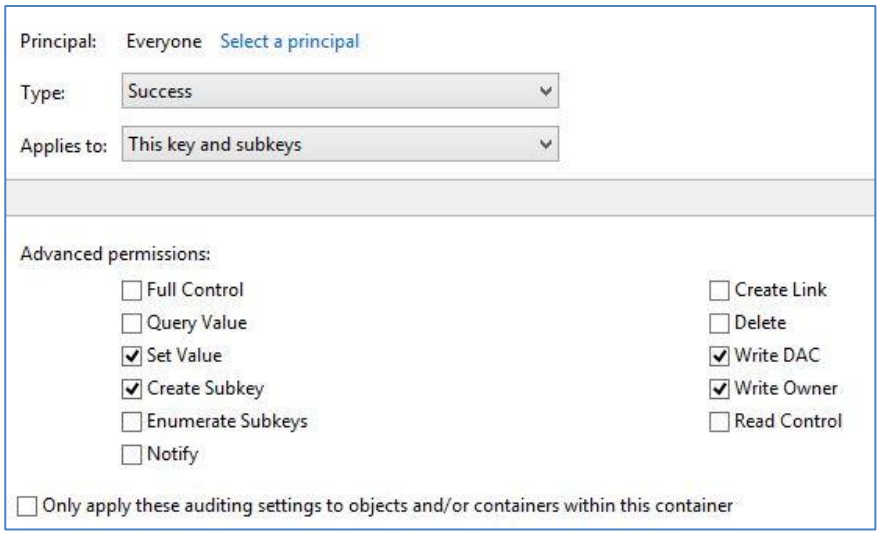


SAMPLE QUERY:

index=windows LogName=Security EventCode=4663 host=* (Accesses="WriteData (or AddFile)" AND Object_Name="*.*") NOT (Security_ID="NT AUTHORITY\\SYSTEM") NOT (Object_Name="*\\FireFoxProfile*" OR Object_Name="*.tmp*" OR Object_Name="*.xml" OR Object_Name="*Thumbs.db" OR Object_Name="\\Device\\HarddiskVolumeShadowCopy*") NOT (Object_Name="*:Zone.Identifier" OR Object_Name="*.part*") | stats count values(Object_Name), values(Accesses) by Security_ID | where count > 1000

MONITOR FOR REGISTRY CHANGES – 4657:

1. Monitor for Registry Changes: Adding auditing to known exploited registry keys is a great way to catch malicious activity. 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.



SAMPLE QUERY:

index=windows LogName=Security (EventCode=4657) Object_Name="*\\Run*" | table _time, host, Security_ID, Account_Name, Account_Domain, Operation_Type, Object_Name, Object_Value_Name, Process_Name, New_Value

MONITOR FOR WINDOWS POWERSHELL COMMAND LINE - 500:

1. Monitor for PowerShell Command Execution: Hackers will often use PowerShell to exploit a system due to the capability of PowerShell and to avoid using built-in utilities and drop additional malware on disk. Monitoring the PowerShell command lines that are executed can catching potentially malicious behavior. PowerShell logs have some odd formatting, the sample below shows a unique non-RegEx way to parse odd logs using the Splunk "***split***" command. PowerShell logs are the worst as far as using the "***split***" command. These logs are not in the standard Windows logs and will need to be added to your Splunk inputs.conf file in order to collect them. The "***Windows PowerShell***" logs may be found under:
    - Applications and Services Logs - Windows PowerShell

```
index=powershell LogName="Windows Powershell" (EventCode=500) | eval MessageA=split(Message,"Details:") | Eval Short_Message=mvindex(MessageA,0) | Eval MessageB=mvindex(MessageA,1) | eval MessageB = replace (MessageB,"[\n\r]","!") | eval MessageC=split(MessageB,"!!!!") | Eval Message1=mvindex(MessageC,0) | Eval Message2=mvindex(MessageC,1) | Eval Message3=mvindex(MessageC,2) | eval MessageD=split(Message3,"!!") | Eval Message4=mvindex(MessageD,3) | eval Message4=split(Message4,"=") | eval PS_Version=mvindex(Message4,1) | Eval Message5=mvindex(MessageD,4) | Eval Message6=mvindex(MessageD,5) | Eval Message7=mvindex(MessageD,6) | eval Message7=split(Message7,"=") | eval Command_Name=mvindex(Message7,1) | Eval Message8=mvindex(MessageD,7) | eval Message8=split(Message8,"=") | eval Command_Type=mvindex(Message8,1) | Eval Message9=mvindex(MessageD,8) | eval Message9=split(Message9,"=") | eval Script_Name=mvindex(Message9,1)| Eval Message10=mvindex(MessageD,9) | eval Message10=split(Message10,"=") | eval Command_Path=mvindex(Message10,1) | Eval Message11=mvindex(MessageD,10) | eval Message11=split(Message11,"=") | eval Command_Line=mvindex(Message11,1) | table _time EventCode, Short_Message, PS_Version, Command_Name, Command_Type, Script_Name, Command_Path, Command_Line
```

MONITOR FOR WINDOWS FIREWALL CHANGES – 2004 & 2005:

1. Monitor for Additions to Firewall Rules: Malware and hackers will often add a firewall rule to allow access to some Windows service or application. These logs are not in the standard Windows logs and will need to be added to your Splunk inputs.conf 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

```
index=windows LogName=Security EventCode=2004 | table _time, host, Rule_Name, Origin, Active, Direction, Profiles, Action, Application_Path, Service_Name, Protocol, Security_Options, Edge_Traversal, Modifying_User, Modifying_Application, Rule_ID
```

2. Monitor for Changes to Firewall Rules: Malware and hackers will often modify a firewall rule to allow access to some Windows service or application. These logs are not in the standard Windows logs and will need to be added to your Splunk inputs.conf file in order to collect them.

```
index=windows LogName=Security EventCode=2005 | table _time, host, Rule_Name, Origin, Active, Direction, Profiles, Action, Application_Path, Service_Name, Protocol, Security_Options, Edge_Traversal, Modifying_User, Modifying_Application, Rule_ID
```

MONITOR FOR WINDOWS POWERSHELL OBFUSCATIION - TICKS AND SPECIAL CHARACTERS:

1. Monitor for PowerShell Obfuscation with 4688: Hackers 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.

```
index=windows LogName="Security" EventCode=4688 NOT ("*ProgramData\\Some_Trusted\\Program*") | eval Orig_Command=Process_Command_Line | eval Clean_Command_Line=Process_Command_Line | eval Obfuscations=Process_Command_Line | rex field=Obfuscations mode=sed "s/[a-zA-Z0-9]//g" | rex field=Clean_Command_Line mode=sed "s/[']//g" | eval Tick_Count = mvcount(split(Obfuscations,"'"))-1 | eval Pct_Count = mvcount(split(Obfuscations,"%"))-1 | eval Dollar_Count = mvcount(split(Obfuscations,"$"))-1 | eval Plus_Count = mvcount(split(Obfuscations,"+"))-1 | eval SemiCol_Count = mvcount(split(Obfuscations,";"))-1 | table _time host, Orig_Command, Clean_Command_Line, Obfuscations, Tick_Count, Pct_Count, Dollar_Count, Plus_Count, SemiCol_Count | where Tick_Count > 2
```

2. Monitor for PowerShell Obfuscation with 400: Hackers 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 "Windows PowerShell" log (v2-v5) 400 events. The "***Windows PowerShell***" logs may be found under:
    - Applications and Services Logs - Windows PowerShell

```
index=powershell LogName="Windows Powershell" EventCode=400 | eval MessageA=split(Message,"Details:") | Eval Short_Message=mvindex(MessageA,1) | eval MessageA=split(Short_Message,"HostVersion=") | Eval MessageA=mvindex(MessageA,1) | eval MessageB=split(MessageA,"HostId=") | Eval PS_Version=mvindex(MessageB,0) | Eval MessageC=mvindex(MessageB,1) | eval MessageD=split(MessageC,"HostApplication=") | Eval Host_ID=mvindex(MessageD,0) | Eval MessageE=mvindex(MessageD,1) | eval MessageF=split(MessageE,"EngineVersion=") | Eval Host_Application=mvindex(MessageF,0) | Eval MessageG=mvindex(MessageF,1) | eval MessageH=split(MessageG,"RunspaceId=") | Eval Engine_Version=mvindex(MessageH,0) | Eval MessageJ=mvindex(MessageH,1) | eval MessageP=split(MessageJ,"CommandLine=") | Eval Command_Line=mvindex(MessageP,1) | eval Obfuscations=Host_Application | rex field=Obfuscations mode=sed "s/[a-zA-Z0-9]//g" | rex field=Clean_Host_Application mode=sed "s/ [^a-zA-Z0-9_]==/ /g" | eval Tick_Count = mvcount(split(Obfuscations,"'"))-1 | eval Pct_Count = mvcount(split(Obfuscations,"%"))-1  | table host, ComputerName, Host_Application, Clean_Host_Application, Obfuscations, Tick_Count, Pct_Count | where Tick_Count > 2
```

MONITOR FOR WINDOWS POWERSHELL BASE64 ENCODED OBFUSCATION:

3. Monitor for PowerShell Obfuscation with 4104 or 400: Hackers will often use obfuscation of PowerShell code to hide what they are doing. Monitoring the size of the PowerShell commands that hide things like Base64 encoded scripts can catch potentially malicious obfuscated PowerShell. The query below looks for the size of a scriptblock over 1000 characters using the "PowerShell/Operation" or "Windows PowerShell" log. The "***Windows PowerShell***" and "**PowerShell Operational**" logs may be found under:
    - Applications and Services Logs - Windows PowerShell
    - Applications and Services Logs – PowerShell/Operational

```
index=powershell (source="WinEventLog:Microsoft-Windows-PowerShell/Operational" OR source="WinEventLog:Windows PowerShell") (EventCode=4104 OR EventCode=400) NOT ("*Some_Trusted \\Program*") NOT ("*InvocationName*" OR "*InvocationInfo*") | eval Cmd_Length=len(Message) | where Cmd_Length > 1000 | table  _time, host, EventCode, Cmd_Length, Message
```

BLACKLIST UNWANTED ITEMS USING THE SPLUNK UNIVERSAL FORWARDER:

OVERVIEW:

With the enhanced logging the other Windows Cheat Sheets recommend, there will unfortunately be a lot more events being generated, and noise. Many Event IDs or the Message within an Event ID do not provide any security value and therefore can be dropped versus being sent to Splunk taking up valuable licensing. The idea here is to blacklist or exclude items at the Universal Forwarder (UF) before they are sent to Splunk and take up some of your valuable license. Whitelisting is the opposite where you tell the UF only to collect certain items which is another option and works identically. If you cannot blacklist enough items in the UF and want to do more, you will need to use the Splunk Heavy Forwarder, or use another syslog agent like nxlog or the "Windows Logging Service" (WLS).

LIMITS:

There can only be 10 blacklist or whitelist items per sourcetype. This can be limiting for logs like the Security log that have tons of events and messages, many of which we do not need to collect. However there is the ability to nest multiple items within one blacklist item.

FORMAT:

The format of the blacklist is RegEx, but not exactly the RegEx you may be used to. The following should provide enough information and detail to build what you need. The first is a straight blacklist by Event ID:

- blacklist = 4689,5158

The next option is to nest multiple messages or parts of a message into one blacklist entry. The following will drop some of the Splunk events from taking up space in Splunk, these are basically worthless events for security purposes and are very noisy. Notice it is by Event ID, Message, and Type within the message (4688, Message, New Process Name:)

- blacklist1 = EventCode="4688" Message="(?:New Process Name:).+(?:SplunkUniversalForwarder\\bin\\splunk.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunkd.exe)|.+(?:SplunkUniversalForwarder\\bin\\btool.exe)"

The next item is nesting many more similar items, there are no spaces between the |:

- blacklist2 = EventCode="4688" Message="(?:New Process Name:).+(?:SplunkUniversalForwarder\\bin\\splunk-winprintmon.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-powershell.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-regmon.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-netmon.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-admon.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-MonitorNoHandle.exe) |.+(?:SplunkUniversalForwarder\\bin\\splunk-winevtlog.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-perfmon.exe)|.+(?:SplunkUniversalForwarder\\bin\\splunk-wmi.exe)"

BLACKLIST UNWANTED ITEMS USING THE SPLUNK UNIVERSAL FORWARDER:

Each Event ID or Type of message must be separate blacklists. The following excludes by ***Process Command Line***, the one blacklist item I would recommend using most as it will be the most unique thing you can exclude.

- blacklist3 = EventCode="4688" Message="(?:Process Command Line:).+(?:--scheme)|.+(?:--no-log)|.+(?:-Embedding)"
- blacklist4 = EventCode="4688" Message="(?:Process Command Line:).+(?:system32\\SearchFilterHost.exe)|.+(?:find /i)|.+(?:Google\\Update\\GoogleUpdate.exe)|.+(?:WINDOWS\\system32\\conhost.exe)"

When you switch Event IDs, you will need a different blacklist. The following excludes Windows Firewall items by Application Name;

- blacklist5 = EventCode="5156" Message="(?:Application Name:).+(?:splunkuniversalforwarder\\bin\\splunkd.exe|.+(?:atlassian\\hipchat4\\hipchat.exe)"

The following item blacklists out Windows Firewall by Broadcast IPs:

- blacklist6 = EventCode="5156" Message="(?:Source Address:)(?s).+(?:224.0.0.251)|(?s).+(?:239.255.255.250)"

The following item blacklists the Windows Firewall by Application Name and by excluding the Chrome browser. This means all the noise from a user surfing will not be collected.

- blacklist7 = EventCode="5156" Message="(?:Application Name:).+(?:google\\chrome\\application\\chrome.exe)"

The following item blacklists a noisy object from collecting:

- blacklist8 = EventCode="4659" Message="(?:Object Name:)(?s).*(AppData\\Local\\Temp\\etilqs_)"

The following item blacklists all 4663 events that contain the “REGISTRY” since Event ID 4657 is the Event ID you will want to collect for Registry audit items, 4663 events for the registry are pretty much worthless and can be dropped.

- blacklist9 = EventCode="4663" Message="(?:Object Name:)(?s).*(\\REGISTRY\\)"

This should help you to create a detailed blacklist that can reduce the amount of events you send to Splunk in order to help reduce licensing impacts.

AUDIT THE REGISTRY USING THE SPLUNK UNIVERSAL FORWARDER (UF):

The Splunk UF allows you to monitor the Registry for Set, Create, Delete, and Renamed items to keys, values and data. There are also Open, Close, and Query, but that would be way too noisy to monitor for.

- https://docs.splunk.com/Documentation/Splunk/6.6.2/Data/MonitorWindowsregistrydata

Monitor the keys listed in the "Windows Registry Auditing Cheat Sheet" as a place to start and go from there.

```
[WinRegMon://HKCU]

    index = workstation_win
    sourcetype = "Win_Registry"
    source = HKCU
    disabled = 0
    hive = \\REGISTRY\\USER\\.*\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\\\?.*
    hive = \\REGISTRY\\USER\\.*\\Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce\\\\?.*
    hive = \\REGISTRY\\USER\\.*\\Software\\\\?.*
    proc = .*
    type = set|create|delete|rename
    baseline = 1
    baseline_interval = 120


[WinRegMon://HKLM]

    index = workstation_win
    sourcetype = "Win_Registry"
    source = HKLM
    disabled = 0
    hive = \\REGISTRY\\MACHINE\\SYSTEM\\CurrentControlSet\\services\\\\?.*
    hive = \\REGISTRY\\MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\\\?.*
    hive = \\REGISTRY\\MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce\\\\?.*
    proc = .*
    type = set|create|delete|rename
    baseline = 1
    baseline_interval = 120
```