WINDOWS HUMIO LOGGING CHEAT SHEET - Win 7 - Win2012

This "**Windows Humio Logging Cheat Sheet**" is intended to help you get started setting up Humio 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:

DEFINITIONS:

WINDOWS LOGGING CONFIGURATION: Before you can Gather anything meaningful with Humio, 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 Humio. 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 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: 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 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.

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
- MalwareArchaeology.com/logging – Scripts and sample agent configs to set and collect your logging
- Better descriptions of Event ID's
  - www.ultimatewindowssecurity.com/securitylog/encyclopedia/Default.aspx
- www.EventID.Net – Extensive list of Event ID's
- Microsoft list of Event IDs for Win 10 and Server 2016
  - https://www.microsoft.com/en-us/download/confirmation.aspx?id=52630
- Google – Of course
- Humio.com – More information and documentation for Humio

WINDOWS HUMIO LOGGING CHEAT SHEET - Win 7 - Win2012

CRITICAL EVENTS TO MONITOR:

The following is a short list of events to collect, create queries and alert on various conditions;

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 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
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 that impact storage and licensing. Your top events in your Humio 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 amount of events you send and collect into Humio. 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, Humio 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 Humio 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'
```

The following Humio 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 query 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 hacking 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

2. **SAMPLE QUERY:** Trigger alert on X command executed, what exceeds a normal administrator

```
"event_id"="4688" not event_data.SubjectUserName="*$"
(/arp.exe/i OR /at.exe/i OR /bcdedit.exe/i OR /bcp.exe/i OR /chcp.exe/i OR /cmd.exe/i OR /cscript.exe/i OR /csvde/i OR
/dsquery.exe/i OR /ipconfig.exe/i OR /mimikatz.exe/i OR /nbtstat.exe/i OR /\\nc.exe/i OR /netcat.exe/i OR /netstat.exe/i OR
/nmap.exe/i OR /nslookup.exe/i OR /netsh.exe/i OR /OSQL.exe/i OR /ping.exe/i OR /powershell.exe/i OR /powercat.ps1/i OR
/psexec.exe/i OR /psexecsvc.exe/i OR /psLoggedOn.exe/i OR /procdump.exe/i OR /qprocess.exe/i OR /query.exe/i OR /rar.exe/i OR
/reg.exe/i OR /route.exe/i OR /runas.exe/i OR /rundll32.exe/i OR /schtasks.exe/i OR /sethc.exe/i OR /sqlcmd.exe/i OR /sc.exe/i OR
/ssh.exe/i OR /sysprep.exe/i OR /systeminfo.exe/i OR /net.exe/i OR /reg.exe/i OR /tasklist.exe/i OR /tracert.exe/i OR
/vssadmin.exe/i OR /whoami.exe/i OR /winrar.exe/i OR /wscript.exe/i OR /"winrm.*"/i OR /"winrs.*"/i OR /wmic.exe/i OR
/wsmprovhost.exe/i OR /wusa.exe/i)
| New_Process_Name:= event_data.NewProcessName | UserName:= event_data.SubjectUserName | Command_Line :=
event_data.CommandLine
| New_Process_Name =~ /(?<ShortName>[^\\]+$)/
| groupby([@host, UserName],
  function=[Process_Cnt:=count(distinct=false, "New_Process_Name"),
        { groupby(ShortName) | sort()
        | format("%d:%s", field=[_count,ShortName], as=Processes)
        | collect([Processes])},
        { collect(Command_Line) | table(Command_Line) }])
| table([@host, UserName, Process_Cnt, Distinct_Cnt, Processes, Command_Line]) | Process_Cnt > 5
```

3. Monitor for Whitelisting bypass attempts: Hackers 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 these bypass applications may be found here:
   - **https://github.com/api0cradle/UltimateAppLockerByPassList**

An extensive list of administrative utilities, including whitelist bypass utilities to monitor for may be found on:

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

The "***Lookup***" command will benefit this query tremendously by having a list to look up all the utilities against.

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 users will first get infected. Exclude known good items.

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

```
"event_id"="4688"
and "event_data.NewProcessName"=/c:\\users/i
| UserName:= event_data.SubjectUserName | Command_Line:= event_data.CommandLine | New_Process_Name:=
event_data.NewProcessName
| table([event_id, @timestamp, @host, UserName, New_Process_Name, Command_Line])
```

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 and system crawling. This alert looks for successful logons to detect when a rogue user account crawls across systems in your network, also known as lateral movement.

SAMPLE QUERY:

```
"event_id"="4624"
not "event_data.TargetUserName"="SYSTEM" not "event_data.TargetUserName"="ANONYMOUS LOGON"
not "event_data.TargetUserName"="NETWORK SERVICE" not "event_data.TargetUserName"="LOCAL SERVICE"
| Target_User:= event_data.TargetUserName | Src_IP:= event_data.IpAddress | PID:= process_id | Src_WS_Name:=
event_data.WorkstationName | Logon_Type:= event_data.LogonType | User:= event_data.SubjectUserName | Domain:=
event_data.SubjectDomainName | Target_Domain:= event_data.TargetDomainName | Auth:=
event_data.AuthenticationPackageName | LM_Name:= event_data.Lm_PackageName | Logon_Process:=
event_data.LogonProcessName | Process_Name:= event_data.ProcessName
| alt{ event_id=4624 | Status := "Success" ; event_id=4625 | Status := "Failed" ; Logon_Type=2 | Logon_Type := "Interactive Logon"
; Logon_Type=3 | Logon_Type := "Network Logon" ; Logon_Type=5 | Logon_Type := "Service" ; Logon_Type=7 | Logon_Type :=
"Unlock" ; Logon_Type=10 | Logon_Type := "RDP" }
| alt{ Logon_Type=2 | Logon_Type := "Interactive Logon" ; Logon_Type=3 | Logon_Type := "Network Logon" ; Logon_Type=5 |
Logon_Type := "Service" ; Logon_Type=7 | Logon_Type := "Unlock" ; Logon_Type=10 | Logon_Type := "RDP" }
| table([Status, @timestamp, @host, Src_WS_Name, Src_IP, User, Domain, Target_User, Target_Domain, Logon_Type, Domain,
Auth, LM_Name, Logon_Process, PID, Process_Name])
```

2. Monitor for clearing of the event logs

SAMPLE QUERY: Monitor for clearing of the logs

```
"event_id"="104" or "event_id"="1102" | EventLog:= user_data.Channel | EventLog:= log_name | user.name:=
user_data.SubjectUserName
| table([event_id, @timestamp, @host, user.name, user.type, task, EventLog, message ])
```

3. 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 "***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.

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

SAMPLE QUERY:

```
"event_id"="4625"
not "event_data.TargetUserName"="SYSTEM" not "event_data.TargetUserName"="ANONYMOUS LOGON"
not "event_data.TargetUserName"="NETWORK SERVICE" not "event_data.TargetUserName"="LOCAL SERVICE"
| Target_User:= event_data.TargetUserName | Src_IP:= event_data.IpAddress | PID:= process_id | Src_WS_Name:=
event_data.WorkstationName | Logon_Type:= event_data.LogonType | User:= event_data.SubjectUserName | Domain:=
event_data.SubjectDomainName | Target_Domain:= event_data.TargetDomainName | Auth:=
event_data.AuthenticationPackageName | LM_Name:= event_data.Lm_PackageName | Logon_Process:=
event_data.LogonProcessName | Process_Name:= event_data.ProcessName
| alt{ event_id=4624 | Status := "Success" ; event_id=4625 | Status := "Failed" ; Logon_Type=2 | Logon_Type := "Interactive Logon"
; Logon_Type=3 | Logon_Type := "Network Logon" ; Logon_Type=5 | Logon_Type := "Service" ; Logon_Type=7 | Logon_Type :=
"Unlock" ; Logon_Type=10 | Logon_Type := "RDP" }
| table([Status, @timestamp, @host, Src_WS_Name, Src_IP, User, Domain, Target_User, Target_Domain, Logon_Type, Domain,
Auth, LM_Name, Logon_Process, PID, Process_Name])
```

4. Monitor for Administrative and Guest Logon Failures: Hackers 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 administrator and guest. Adding a count will detect brute force or script attempts (>5 what exceeds your lockout setting).

"event_id"="4625" (administrator or guest)

MONITOR FOR FILE SHARES – SECURITY LOG - 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: A network share was accessed

```
"event_id"="5140" not "event_data.ShareName"="\\*\IPC$"
| Status:= keywords[0] | Src_IP:= event_data.IpAddress | PID:= process_id | UserName:= event_data.SubjectUserName |
Domain:= event_data.SubjectDomainName | Path:= event_data.ShareLocalPath | Share:= event_data.ShareName
| table([event_id, @timestamp, @host, Status, Src_IP, PID, UserName, Domain, task, event_data.RelativeTargetName, Path,
Share])
```

SAMPLE QUERY: A network share was accessed – shows files accessed

```
"event_id"="5145"
not "event_data.ShareName"="\\*\IPC$"
| Status:= keywords[0] | Src_IP:= event_data.IpAddress | PID:= process_id | UserName:= event_data.SubjectUserName |
Domain:= event_data.SubjectDomainName | Target_File:= event_data.RelativeTargetName | Path:= event_data.ShareLocalPath |
Share:= event_data.ShareName
| table([event_id, @timestamp, @host, Status, Src_IP, PID, UserName, Domain, task, Target_File, Path, Share])
```

MONITOR FOR SERVICE CHANGES – SYSTEM LOG - 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. Services can load DLLs and Drivers which are not logged elsewhere.

SAMPLE QUERY:

```
"event_id"="7045"
| Service_Name:= event_data.ServiceName | Image_Path:= event_data.ImagePath | Service_Type:= event_data.ServiceType |
Start_Type:= event_data.StartType
| table([event_id, @timestamp, @host, computer_name, Service_Name, Image_Path, Service_Type, Start_Type])
```

2. Monitor for Service State Changes: Monitoring for 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:

```
"event_id"="7040" not "event_data.param1"="Background Intelligent Transfer Service"
| Service_Name:= event_data.param4 | Service_Desc:= event_data.param1 | Service_Type:= event_data.ServiceType |
Before_Start_Type:= event_data.param2 | After_Start_Type:= event_data.param3
| table([event_id, @timestamp, @host, user.name, user.type, Service_Name, Service_Desc, Before_Start_Type, After_Start_Type])
```

MONITOR FOR NETWORK CONNECTIONS – SECURITY LOG - 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. The "***Lookup***" command will benefit this query tremendously by excluding items.

SAMPLE QUERY:

```
"event_id"="5156"
not windows\\system32\\spoolsv.exe
| Application:= event_data.Application | Src_IP:= event_data.SourceAddress | Src_Port:= event_data.SourcePort |
Dest_IP:=event_data.DestAddress | Dest_port:= event_data.DestPort | Protocol:= event_data.Protocol
| alt{ Protocol=6 | Protocol := "TCP" ; Protocol=17 | Protocol := "UDP"} | iplocation(Dest_IP)
| table([event_id, @timestamp, @host, Src_IP, Src_Port, Dest_IP, Dest_IP.country, Dest_IP.state, Dest_IP.city, country, city,
Dest_Port, Protocol, Application]) | Dest_IP!="192.168.*.*" | Dest_IP!="52.109.*.*"
```

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. 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

```
"event_id"="4663"
not \\REGISTRY\\USER not "Services\\\\W32Time\\\\SecureTimeLimits"
| UserName:= event_data.SubjectUserName | Process_Name:= event_data.ProcessName | Object_Name:=
event_data.ObjectName | Access_Mask:= event_data.AccessMask | Handle_ID:= event_data.HandleId | PID:=
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_id, @timestamp, @host, UserName, Process_Name, Object_Name, Accesses, Access_Mask, Handle_ID, PID])
```

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

```
"event_id"="4657" not "Services\\\\W32Time\\\\SecureTimeLimits"
| Handle_ID:= event_data.HandleId | UserName:= event_data.SubjectUserName | Domain:= event_data.SubjectDomainName | Process_Name:=
event_data.ProcessName | Object_Name:= event_data.ObjectName | Value_Name:= event_data.ObjectValueName
| New_Value:= event_data.NewValue | New_Value_Type:= event_data.NewValueType | Old_Value:= event_data.OldValue | Old_Value_Type:=
event_data.OldValueType | Object_Type:= event_data.ObjectType | Access_List:= event_data.AccessList | Operation_Type:=
event_data.OperationType | PID:= 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"}
| alt{ Old_Value_Type=%%1872 | Old_Value_Type := "Reg_None" ; Old_Value_Type=%%1873 | Old_Value_Type := "Reg_Sz" ;
Old_Value_Type=%%1874 | Old_Value_Type := "Reg_Expand_Sz" ; Old_Value_Type=%%1875 | Old_Value_Type := "Reg_Binary" ;
Old_Value_Type=%%1876 | Old_Value_Type := "Reg_DWord" ; Old_Value_Type=%%1879 | Old_Value_Type := "Reg_Multi_Sz" ;
Old_Value_Type=%%1883 | Old_Value_Type := "Reg_QWord"}
| table([event_id, @timestamp, @host, UserName, Domain, PID, Handle_ID, Process_Name, Old_Value, Old_Value_Type, New_Value,
New_Value_Type, Object_Type, Operation_Type, Access_List, Value_Name, Object_Name])
```

WINDOWS HUMIO LOGGING CHEAT SHEET - Win 7 - Win2012

MONITOR FOR WINDOWS FIREWALL CHANGES – FIREWALL LOG - 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. 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

```
"event_id"="2004"
| Rule_Name:= event_data.RuleName | Application_Path:= event_data.ApplicationPath | Modifying_App:= event_data.ModifyingApplication
| table([event_id, @timestamp, @host, computer_name, user.name, user.type, Rule_Name, Modifying_App, Application_Path ])
```

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.

SAMPLE QUERY: Monitor for changes to existing rules

```
"event_id"="2005"
| Rule_Name:= event_data.RuleName | Application_Path:= event_data.ApplicationPath | Profiles:= event_data.Profiles | Modifying_App:=
event_data.ModifyingApplication | Direction:= event_data.Direction | Protocol:= event_data.Protocol
| table([event_id, @timestamp, @host, Rule_Name, Application_Path, Profiles, Modifying_App, Direction, Protocol])
```

MONITOR FOR WINDOWS FIREWALL CHANGES – TASKSCHEDULER LOG - 106:

1. Monitor for Additions to Scheduled Tasks: Malware and hackers will often add a scheduled task to 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 Humio. The TaskScheduler log may be found under:
    - Applications and Services Logs – Microsoft - Windows – TaskScheduler - Operational

SAMPLE QUERY: Monitor for new scheduled tasks being added

```
"event_id"="106"
| regex(regex="task \".(?<Task_Name>[\w\s]+?.*)", field="message") | UserContext:= event_data.UserContext
| table([event_id, @timestamp, @host, UserContext, task, Task_Name, message ])
```

2. Monitor for scheduled task being started: Once a task is created it will need to start. Hackers 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.

SAMPLE QUERY: Monitor for existing scheduled task being started

```
"event_id"="110"
| UserName:= event_data.UserContext | UserName:= event_data.UserName | Task_Name:= event_data.TaskName
| table([event_id, @timestamp, @host, UserName, UserContext, task, Task_Name, message ])
```

3. Monitor for scheduled task being started: Tasks generally get created, not deleted. Monitor for this condition to find a task that has been created, started and deleted.

SAMPLE QUERY: Monitor for scheduled task being registered and deleted

```
"event_id"="106" or "event_id"="141"
| regex(regex="task \".(?<Task_Name>[\w\s]+?.*)", field="message") | UserName:= event_data.UserContext | UserName:=
event_data.UserName
| table([event_id, @timestamp, @host, UserName, UserContext, task, Task_Name, message ])
```

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: Hackers 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 and 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://researchcenter.paloaltonetworks.com/2017/03/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/

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

```
"event_id"="4688"
(/powershell/i and /bypass/i) or (/powershell/i and (/-ep/i or /-exec/i)) or (/powershell/i and /-nop/i)
or (/pwsh/i and /bypass/i) or (/pwsh/i and /-exp/i) or (/pwsh/i and /-nop/i)
| UserName:= event_data.SubjectUserName | Command_Line:= event_data.CommandLine | New_Process_Name:=
event_data.NewProcessName
| table([@timestamp, @host, UserName, Command_Line, New_Process_Name])
```

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

```
"event_id"="400" or "event_id"="500" or "event_id"="600" or "event_id"="800"
(/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)
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="event_data.param3")
| table([event_id, @timestamp, @host, level, Command])
```

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

```
"event_id"="4104"
(/bypass/i or /-ep/i or /-exec/i or /-nop/i or /hidden/i)
| ScriptBlock:= event_data.ScriptBlockText
| table([event_id, @timestamp, @host, user.name, level, ScriptBlock, task])
```

WINDOWS HUMIO LOGGING CHEAT SHEET - Win 7 - Win2012

MONITOR FOR WINDOWS POWERSHELL WEB DOWNLOAD:

1. Monitor for PowerShell for Web Downloads: Hackers 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

```
"event_id"="4688" ((/powershell/i or /pwsh/i) and (/webclient/i or /download/i or /http/i))
| UserName:= event_data.SubjectUserName | Command_Line:= event_data.CommandLine | New_Process_Name:=
event_data.NewProcessName
| table([event_id, @timestamp, @host, UserName, Block, Command_Line, New_Process_Name])
```

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

```
"event_id"="4104" and (/webclient/i or /.download/i)
| table([event_id, @timestamp, @host, user.name, Block, message, task])
```

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

```
"event_id"="400" or "event_id"="500" or "event_id"="600" and (/webclient/i or /download/i or /http/i) not
"powershell download"
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="event_data.param3")
| table([event_id, @timestamp, @host, level, Command])
```

MONITOR FOR WINDOWS POWERSHELL BASE64 EVENTS:

1. Monitor for PowerShell Base64 execution with Security Log Event ID 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 a series of characters that indicate Base64 being used with Process Command Line logging enabled.

```
"event_id"="4688" /powershell/i and /enc/i
| regex(regex="(?<Block>[a-zA-Z0-9]{25,50})", field="event_data.CommandLine")
| UserName:= event_data.SubjectUserName | Command_Line:= event_data.CommandLine | New_Process_Name:=
event_data.NewProcessName
| table([event_id, @timestamp, @host, UserName, Block, Command_Line, New_Process_Name])
```

2. Monitor for large PowerShell ScriptBlocks with 4104: Hackers 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

```
"event_id"="4104"
| UserName:= event_data.SubjectUserName | ScriptBlock:= event_data.ScriptBlockText
| length(ScriptBlock, as=Block_Cnt) | Block_Cnt > 1000
| table([event_id, @timestamp, @host, event_data.SubjectUserName, user.name, task, Block_Cnt, ScriptBlock])
```

MONITOR FOR WINDOWS POWERSHELL OBFUSCATION - 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.

```
"event_id"="4688" /powershell/I or /pwsh.exe/i
| event_data.CommandLine = /(?<Spl_Chars>[^a-zA-Z0-9 ^\[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect()) | length(Spl_Chars, as=SplCnt)
| UserName:= event_data.SubjectUserName
| table([event_id, @timestamp, @host, UserName, level, Spl_Chars, SplCnt, event_data.CommandLine]) | SplCnt > 20
```

2. Monitor for PowerShell Obfuscation with 200, 400, 500, 600 or 800: 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

```
"event_id"="200" or "event_id"="400" or "event_id"="500" or "event_id"="600" or "event_id"="800" not "powershell download"
not "DropboxOEM"
| regex(regex="HostApplication=(?<Command>[^=]*)\\n\\tEngineVersion", field="event_data.param3")
| Command = /(?<Spl_Chars>[^a-zA-Z0-9 ^\[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect()) | length(Spl_Chars, as=SplCnt)
| table([event_id, @timestamp, @host, level, Spl_Chars, SplCnt, Command]) | SplCnt > "20"
```

3. Monitor for PowerShell Obfuscation with 4104: 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 "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

```
"event_id"="4104"
| ScriptBlock:= event_data.ScriptBlockText
| ScriptBlock = /(?<Spl_Chars>[^a-zA-Z0-9 ^@^|^#^\[^\]^\{^\}\)\(]+)/g
| groupby(@id, function=collect()) | length(Spl_Chars, as=SplCnt)
| table([event_id, @timestamp, @host, user.name, Spl_Chars, SplCnt, ScriptBlock, task]) | SplCnt > 20
```