# Global Information Assurance Certification Paper

**Monitoring for Security Events Using Windows Management Instrumentation**

Stephen Seigler

January 7, 2002

## Summary

This paper was written to demonstrate a method of using Windows Management Instrumentation (WMI) and Windows Scripting Host (WSH) to interactively monitor a computer system for security events. It will cover monitoring the following aspects of security auditing:

Event Log
Registry
Processes
New User Creation
CD/Floppy Access
Scheduler Service

Conventional security auditing generally involves creating baselines of the aforementioned items, and periodically comparing the current status against the baseline. The WSH script included in the appendixes can be used to monitor each of these items interactively. This would have the benefit of reducing response time to a security event, by generating an alert when the event happens. Otherwise, the security event might not be discovered until the next manual audit.

## Example

Suppose a hacker is going to attempt to create a back door on your monitored system using the schedule service and netcat (nc.exe). This script would generate two alerts if they were successful. It would notify that a job has been scheduled, and then that an unapproved process is running. If they were unsuccessful, an alert would be generated about the failed attempt. If successful, without the script generating these alerts, the back door could potentially be accessible to the intruder until a manual audit showed the rogue process.

## Auditing

Network auditing is a very important but complex subject. There are many different opinions on determining what to audit, how to audit, and when to audit. The details of an audit policy will vary from one network to the next. Determining an audit policy is beyond the scope of this paper.

What is auditing? According to **Microsoft Windows NT 4.0 Security, Audit, and Control**,

> Auditing is an important component of the Effective Security Monitoring controls. Auditing means measuring the system against a predefined system setting to ensure no changes have occurred. Changes may indicate possible security breaches. If no auditing is being conducted, then the Effective Security Monitoring controls will not be satisfied and thus confidentiality, integrity, and availability of data is at risk.
>
> Auditing takes time and effort to implement and it uses a lot of resources; therefore, many corporations do not even turn on auditing on their Windows NT systems.

This script was written to help ease the burden of auditing, by reporting security events directly to a central location. In a large network, reviewing event logs alone is a monumental task.

**General Statements**

This script was written specifically to monitor Windows NT4, SP6a systems. Some customization may be necessary to run it on Windows 2000 or later.

There are three scripts that complete this monitoring solution: SystemMonitor.vbs, ProcessList.vbs, and AdministrativeWorkstation.vbs. We will discuss what each script accomplishes and then look the SystemMonitor.vbs script in detail.

Monitored systems and the workstation that receives the alerts must be running Windows Scripting Host (WSH) and Windows Management Instrumentation (WMI). WMI may be downloaded from http://www.microsoft.com/downloads/release.asp?ReleaseID=18490 WSH may be downloaded from http://msdn.microsoft.com/downloads/default.asp?URL=/downloads/sample.asp?url=/msdn-files/027/001/733/msdncompositedoc.xml These pages also contain the system requirements for these products.

To use this script, select the entire section Appendix A and paste into a text editor such as Notepad. Create a new folder for the script, and save the file as ASCII text with the name 'SystemMonitor.vbs' in the new folder.

If you intend to use this script to monitor processes, you must also create the ProcessList.vbs file in the same folder as the 'SystemMonitor.vbs' script. Select the entire section appendix B and paste into a text editor such as Notepad. Save the file as ASCII text with the name it as 'ProcessList.vbs'.

To receive pop-up alerts on the administrative workstation, you will need to run the 'Administrative.vbs' script. If this script is not running on the administrative workstation, events will be written to its event log without a pop-up notification. Select the entire section appendix C and paste into a text editor such as Notepad. Save the file as ASCII text with the name 'Administrative.vbs' in it.

If Notepad is used, make sure that word wrap is turned off, and it may be necessary to put quotation marks around the filename in the save as box to avoid the .txt extension being appended to the filename. If Wordpad is used, choose the save as text document option.

It is recommended that the SystemMonitor.vbs script be installed as a service on the system to be monitored with an automatic startup. This service must not run under the service account, or security events will not be sent to the administrative workstation. Please see Microsoft Knowledge Base article Q137890, HOWTO: Create a User-Defined Service. More information about this can be found in Appendix D.

The Administrative.vbs script may be installed as a service on the administrative workstation with an automatic startup. This service would need to be run under the system account, and allowed to interact with the desktop. Please see Microsoft Knowledge Base article Q137890, HOWTO: Create a User-Defined Service. More information about this can be found in Appendix D.

**SystemMonitor.vbs**

This is the main script and runs on the system to be monitored. It can be used for monitoring the following security events.

**Monitor the event log**
Harry Krimkowitz makes the following statements in his paper, **Mitigating Risks to the Insider Threat within your Organization**.

> Finally, you should require system administrators to perform comprehensive information security audits. You should require that audit logs be stored and reviewed at least weekly. Of course, you must allocate your system administrators sufficient time and resources to adequately review these logs.

While these statements are very correct, this script can help by monitoring all failed security events, and successful security events that would be of interest. This would certainly reduce the amount of time and resources required reviewing the logs. It would also eliminate the time between when a security event is written to the logs, and when the log is reviewed. The following table contains a list of successful events that this script monitors.

| | |
|---|---|
| Event ID: 517 | The audit log was cleared |
| Event ID: 608 | User Right Assigned |
| Event ID: 610 | New Trusted Domain |
| Event ID: 611 | Removing Trusted Domain |
| Event ID: 612 | Audit Policy Change |
| Event ID: 624 | User Account Created |
| Event ID: 625 | User Account Type Change |
| Event ID: 626 | User Account Enabled |
| Event ID: 631 | Global Group Created |
| Event ID: 632 | Global Group Member Added |
| Event ID: 635 | Local Group Created |
| Event ID: 636 | Local Group Member Added |
| Event ID: 639 | Local Group Changed |
| Event ID: 640 | General Account Database Change |
| Event ID: 641 | Global Group Changed |
| Event ID: 642 | User Account Changed |
| Event ID: 643 | Domain Policy Changed |
| Event ID: 644 | User Account Locked Out |

This list may be modified in the script. Please refer to **Microsoft Knowledge Base article Q17407** for a complete list of security log events. The script may be configured to monitor all warning and error events.

NOTE: It is important to note that depending on your audit policy, these events may not be recorded.

**Monitor specific registry keys for changes or new entries**
**Hacking Exposed, Third Edition**, p. 209 states:

> A Backdoor Favorite: Windows Startup Receptacles
>
> …attackers almost always place necessary Registry values under the standard Windows startup keys. These areas should be checked regularly for the presence of malicious or strange-looking commands.
>
> - HKLM\Software\Microsoft\Windows\CurrentVersion\Run and RunOnce, RunOnceEx, RunServices (Win 9x only)

CERT makes the following statements in their article titled **Selecting audit events for Windows NT 4.0 registry keys** regarding the following registry keys.

> These keys, their subkeys, and values determine logon and other authentication enforcement settings on the system, accessibility of

account information and the registry from other systems on the network, and dialup connections. In the case of domain controller systems, these settings apply to the entire NT domain and can affect related, trusted domains.

- SYSTEM\CurrentControlSet\Control\Lsa
- SYSTEM\CurrentControlSet\Control\SecurityProviders
- SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

This script monitors these keys for changes interactively. If the keys that are monitored are changed in any way, an alert will be generated.

**Monitor processes**
**Hacking Exposed, Third Edition**, p. 209 states:

> …regular analysis of the Process List can be useful. For example, you could schedule regular AT jobs to look for remote.exe or nc.exe in the Process List and kill them.

When the script starts, it checks each process that is already running, and compares it against the list of approved processes in the ProcessList.txt file. After that, it checks every process that starts up. The ProcessList.txt file that contains the list of approved processes can be generated by running the ProcessList.vbs script in appendix B. (Please see the detailed description of the ProcessList.vbs script.) It can also generate alerts when an approved process is created. For example, regedt32.exe may be an approved process, but you may want to know whenever it is used. Nc.exe however, is not an approved process, and you would certainly want to know when it is started or running.

**Monitor new user creation**
**Hacking Exposed, Third Edition**, p. 557-558 states the following regarding user accounts.

> **Creating Rogue User Accounts**
> Most every system administrator recognizes that superuser-equivalent accounts are critical resources to protect and audit. What is more difficult to track are inconspicuously named accounts that have superuser privileges. Malicious hackers will try to create such accounts without fail on conquered systems.

It is therefore a good practice to create a baseline list of all network user accounts and audit against it regularly. This script helps to get a head start on this by generating an alert when a new user account is created.

**Monitor the CD-ROM and/or the floppy drive**
Dave Ashcroft makes the following statements in his paper, **Physical Security: The Often Overlooked Weakness**.

> If your data is well protected from attack from the Internet, and your employees only have access to those parts of the databases that they need for their specific jobs, many would feel that they are adequately protected. However, one big problem is still often overlooked. This problem is a weakness in physical security.

He then proceeds to discuss controls related to physical security. By monitoring for CD-ROM and floppy access, this script will keep you better informed of access to your system. If someone is using the CD-ROM or floppy drive, physical access has been gained to the monitored system.

**Monitor the Scheduler service**
**Hacking Exposed, Third Edition**, p. 194-196 demonstrates a method of using the schedule service to allow remote command execution.

> When the scheduled command has executed, the job ID will vanish from the AT listing. If the command was entered correctly, the remote server is now running. Intruders can now gain a command shell on a remote system using the remote utility in client mode, as shown next.

This indicates that once the job has been run, there would now be no scheduled job, and no record of its existence. How would this scheduled job be detected? It is important that jobs be monitored as they are created, because a job scheduled by a hacker may only exist for a few minutes. This script can be configured to monitor scheduled job creation, and generate an alert when a job is scheduled.

**Here's a breakdown of the SystemMonitor.vbs script.**

Here we specify that all variables must be declared and declare them.

```
'This line forces declaration of all variables
Option Explicit

'Declare all variables
```

These are the variables used to control what the script monitors.

```
Dim AdministrativeAlert
Dim MonitorEventLog
Dim MonitorAllFailed
Dim MonitorAllErrors
```

```
Dim MonitorAllWarnings
Dim MonitorRegistry
Dim MonitorCDROM
Dim MonitorFloppy
Dim MonitorScheduler
Dim MonitorProcesses
Dim MonitorServices
Dim CDDriveLetter
Dim ScanSpeed
```

**Script Customization**

This section of the script sets the variables that the script uses to determine what to monitor.

This variable should be set to the name of the workstation that receives the alerts. This workstation must be running NT4, SP4 or later. The alerts will be written to the application log of this computer, and if the Administrative.vbs script is running on this workstation, it will generate a pop-up when an alert is received.

```
AdministrativeAlert = "computername"
```

These variables control whether the event log is monitored. If it is set to monitor the event log, the other variables control which types of events are monitored. This will analyze events as they are written to the event log. Unless MonitorAllFailed is set to 0, the script monitors all failed security events. It also monitors certain successful security events that would be of interest. The list of monitored successful security events may be modified in the script. Set MonnitorAllFailed = 1 to monitor all failed security events. Set MonitorAllErrors = 1 to monitor all error events. Set MonitorAllWarnings = 1 to monitor all warning events. If the script is used to monitor all error and warning events, this may generate significant amounts of alerts, depending on your environment.

This script monitors the event log for Event ID: 624, User Account Created to report when a new user is created. Auditing 'User and Group Management' must be enabled for this event to be logged. MonitorEventLog must not be set to 0 for this script to report all new users created.

```
MonitorEventLog = 1
MonitorAllFailed = 1
MonitorAllErrors = 0
MonitorAllWarnings = 0
```

This variable controls whether this script monitors the registry. This script will monitor all keys listed in the script below, which can be customized. It will monitor those keys for new entries, changes, and key or value deletion.

MonitorRegistry = 1

The keys that this script monitors for changes are listed below. This list can be customized in the script.

HKLM\SYSTEM\CurrentControlSet\Control\Lsa
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

This variable controls whether the script monitors the CD-ROM drive. This will log and generate an alert whenever a CD is put in the drive. Set CDDriveLetter to correspond with the drive letter of your CD-ROM drive. NOTE: This may cause the CD-ROM drive light to flash on and off, depending on the ScanSpeed chosen.

MonitorCDROM = 1
CDDriveLetter = "X:"

This variable controls whether the script monitors the floppy disk drive. This will log and generate an alert whenever a floppy disk is put in the floppy drive. NOTE: This may cause the floppy disk drive light to flash on and off, depending on the ScanSpeed chosen.

MonitorFloppy = 1

This variable controls whether the script monitors the scheduler service. This will log and generate an alert whenever a job is scheduled.

MonitorScheduler = 1

This variable controls whether the script monitors processes. This will monitor for unauthorized processes. It will log and generate an alert whenever an unauthorized process is detected. If processes are to be monitored, the ProcessList.vbs script must be run in the same folder as this script to generate the ProcessList.txt file, which will list all currently running processes. The ProcessList.txt should be modified as necessary to add authorized processes.

MonitorProcesses = 1

The rate of scanning affects the monitoring of the CD-ROM, floppy, scheduler, and new processes and is set by the ScanSpeed variable set below. Note that short scan rates may incur significant processor overhead. However, processes with a lifespan shorter than the chosen ScanSpeed might not be detected. Use the Task Manager to monitor the system overhead being used by wscript.exe and winmgmt.exe processes. This

number should be experimented with to determine the optimum number for each installation.

```
ScanSpeed = 5
```

This next line displays a message box for five seconds, so that the user knows that the script is starting.

```
result = WshShell.Popup("Monitor script is starting. Please wait...", 5)
```

Here we determine the location of this script, and set filenames for use later in the script.

```
strScriptPath = Left(WScript.ScriptFullName, InStrRev(WScript.ScriptFullName, "\"))
strLogFileName = strScriptPath & "monitor.log"
strProcessFilePath = "ProcessList.txt"
```

Here we make sure that there is a file for logging to. If there isn't, it creates it.

```
If Not objFSO.fileexists(strLogFileName) Then
   Set objFile = objFSO.CreateTextFile(strLogFileName, True)
   objFile.Close
End If
```

This section requests that events that have one of the event codes in the list be sent to the EventSink_OnObjectReady sub for processing. The codes listed here are the list of successful security events of interest listed above.

```
   ' Set up the event selection. EventSink_OnObjectReady will be called when
   ' one of the specified Win32_NTLogEvent occurs
   strQuery = "SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA
'Win32_NTLogEvent'" & _
         "and TargetInstance.EventCode = 517 or TargetInstance.EventCode = 608" & _
         " or TargetInstance.EventCode = 610 or TargetInstance.EventCode = 611" & _
         " or TargetInstance.EventCode = 612 or TargetInstance.EventCode = 624" & _
         " or TargetInstance.EventCode = 625 or TargetInstance.EventCode = 631" & _
         " or TargetInstance.EventCode = 632 or TargetInstance.EventCode = 635" & _
         " or TargetInstance.EventCode = 636 or TargetInstance.EventCode = 639" & _
         " or TargetInstance.EventCode = 640 or TargetInstance.EventCode = 641" & _
         " or TargetInstance.EventCode = 642 or TargetInstance.EventCode = 643" & _
         " or TargetInstance.EventCode = 644"
```

Here depending on how the event log variables have been configured, strings are appended onto the query string.

```
   If MonitorAllFailed = 1 Then
```

```
    strQuery = strQuery & " or TargetInstance.Type = 'audit failure'"
  End If

  If MonitorAllWarnings = 1 Then
    strQuery = strQuery & " or TargetInstance.Type = 'warning'"
  End If

  If MonitorAllErrors = 1 Then
    strQuery = strQuery & " or TargetInstance.Type = 'error'"
  End If
```

Here is where the actual list of registry keys to be monitored is declared. This list can be modified as necessary.

```
  ' Set up the registry key selection. RegistrySink_OnObjectReady will be called when
  ' a change is detected at any of the keys listed below
  Registry.ExecNotificationQueryAsync RegistrySink, _
  "SELECT * FROM RegistryKeyChangeEvent " & _
    "Where (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'Software\\Microsoft\\Windows\\CurrentVersion\\Run') " & _
    "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce') " & _
    "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath = 'Software\\Microsoft\\Windows
NT\\CurrentVersion\\Winlogon') " & _
    "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'System\\CurrentControlSet\\Control\\Lsa') " & _
    "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'System\CurrentControlSet\Control\SecurityProviders')"
```

This is the function that performs all logging and alerting. It writes the message that is passed to it to the log file, and if the alert variable is true, logs it to the event log of the administrative workstation.

```
Public Function LogEntry(blankline, msg, alert)
  Set Log = objFSO.OpenTextFile(strLogFileName, 8, False)

  If blankline Then
    Log.WriteBlankLines (1)
    Log.WriteLine (Date & " at " & Time)
    Log.WriteLine (msg)
    Log.Close
  Else
    Log.WriteLine (Date & " at " & Time)
    Log.WriteLine (msg)
    Log.Close
  End If
```

```
    If alert Then
       WshShell.LogEvent 0, msg, AdministrativeAlert
    End If

End Function
```

Here we log the fact that monitoring has started, and then display a message stating that system monitoring has started.

```
result = LogEntry(True, "System monitoring started.", False)

WScript.Echo "System monitoring in progress. To stop monitoring, click OK."
```

### Troubleshooting the scripts

Generally speaking, errors with the scripts will indicate that WMI and/or WSH are not installed, not installed correctly, or a WMI provider is not registered. To help determine where to start looking for the problem, use the customization section of the SystemMonitor.vbs script to disable monitoring everything except one thing. Then run the script. Keep enabling one item at a time, until you find what is causing the trouble. A WMI provider may need to be registered. For more information regarding WMI provider registration, please visit http://search.microsoft.com/us/dev/default.asp and search for 'register wmi provider'.

Error 2140: an internal Windows NT error occurred. If you receive this error message when you attempt to start the service, the application parameter is probably incorrect.

If you install the scripts as services, and the scripts do not appear to be working correctly, try executing them directly. Once you can run the scripts without errors and they work properly, then try them as services.

### Suggested Improvements

- Monitor ports. Write a routine that would create a baseline of all open ports, and then monitor for new ports that the system has open.
- Monitor new file creation. This would have to be implemented very carefully, or the alerts generated by this would be overwhelming. However, if a new executable was created in your inetpub\scripts folder, that would very likely be a security event.
- Create a user interface (UI). Depending on the volume of notifications that this script generates in an environment, it would possibly be advantageous to have a UI on the administrative workstation, instead of pop-up alerts.
- Log to ODBC database. Long term storage of security events reported to the administrative workstation would be best stored in an ODBC database. This would better facilitate queries, and generating reports.

- Add the ability to send alerts to multiple administrative workstations.

**References**


Joel Scambray, Stuart McClure, George Kurtz, Hacking Exposed (third edition) Osborne/McGraw-Hill, 2001

James G. Jumes, Neil F. Cooper, Paula Chamoun, Todd M. Feinman, Windows NT 4.0 Security, Audit, and Control, PricewaterhouseCoopers, 1999

Central Auditing of Windows NT Using Windows Script Host (WSH), March 30, 2001, Roger R. McLaren, http://rr.sans.org/win/WSH.php

Physical Security: The Often Overlooked Weakness, July 31, 2000, Dave Ashcroft, http://rr.sans.org/firewall/phys_sec.php

Mitigating Risks to the Insider Threat within your Organization, October 24, 2000, Harry Krimkowitz, http://rr.sans.org/securitybasics/insider_threat.php

Selecting audit events for Windows NT 4.0 registry keys, March 17, 1999, Carnegie Mellon University, http://www.cert.org/security-improvement/implementations/i028.04.html

Understanding WMI Eventing, September 2000, Alexander Nosov, http://www.winscriptingsolutions.com/Articles/Index.cfm?ArticleID=9805&pg=1

DetectNewProcesses, 2000, Unisys, http://estools.unisys.com/scriptlib/advanced/detectnewprocesses.html

Security Event Descriptions (Q174074), March 19, 1998, Microsoft, http://support.microsoft.com/default.aspx?scid=kb;EN-US;q174074

Windows Management Instrumentation (WMI) SDK, March 22, 2000, Microsoft, http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/MSDN-FILES/027/001/566/msdncompositedoc.xml

Appendix A

```
'This line forces declaration of all variables
Option Explicit

'Declare all variables
Dim objFSO
Dim objFile
Dim objTextStream
Dim WshShell

Dim strScriptPath
Dim strLogFileName
Dim strQuery
Dim result
Dim Log
Dim msg

Dim AdministrativeAlert
Dim MonitorEventLog
Dim MonitorAllFailed
Dim MonitorAllErrors
Dim MonitorAllWarnings
Dim MonitorRegistry
Dim MonitorCDROM
Dim MonitorFloppy
Dim MonitorScheduler
Dim MonitorProcesses
Dim MonitorServices
Dim CDDriveLetter
Dim ScanSpeed

Dim EventsSink
Dim RegistrySink
Dim FloppySink
Dim CDRomSink
Dim TaskSink
Dim ProcessSink

Dim Events
Dim Registry
Dim Service
Dim ServiceSet
Dim strProcessFilePath
Dim aryProcessName(1000)
Dim aryProcessExecutablePath(1000)
Dim aryProcessAlert(1000)
Dim varProcessApproved
Dim varProcessPathNull
Dim varCounter
Dim varCounterProcess
Dim strInputLine
```

```
Dim strSplitString
Dim ProcessList
Dim Process

' This is the section used to customize this script. There are other places in the script for customization.
' To change the workstation that receives alerts, change the value of the next line.
' Alerts will be written to the application log of this computer.
AdministrativeAlert = "computername"

' Monitor the event logs. This will analyze events as they are written to the event log.
' It will only alert on event codes listed in the script below, which can be customized.
' Set MonitorAllFailed = 1 to monitor all failed security events.
' Set MonitorAllErrors = 1 to monitor all error events.
' Set MonitorAllWarnings = 1 to monitor all warning events.
' NOTE: This script will report all new users created, but not if MonitorEventLog is set to 0.
MonitorEventLog = 1
MonitorAllFailed = 1
MonitorAllErrors = 0
MonitorAllWarnings = 0

' Monitor the registry. This will monitor all keys listed in the script below, which can be customized.
' It will monitor those keys for new entries, changes, and key or value deletion.
MonitorRegistry = 1

' Monitor the CD-ROM drive. This will log and generate an alert whenever a CD is put in the drive.
' Set CDDriveLetter to correspond with the drive letter of your CD-ROM drive.
' NOTE: This may cause the CD-ROM drive light to flash on and off, depending on the ScanSpeed chosen.
MonitorCDROM = 1
CDDriveLetter = "X:"

' Monitor the floppy drive. This will log and generate an alert whenever a floppy disk is put in the floppy
drive.
' NOTE: This may cause the floppy disk drive light to flash on and off, depending on the ScanSpeed
chosen.
MonitorFloppy = 1

' Monitor scheduler service. This will log and generate an alert whenever a job is scheduled.
MonitorScheduler = 1

' Monitor processes. This will monitor for unauthorized processes. It will log and generate an alert
whenever an
' unauthorized process is detected. If you monitor processes, be sure to run the processlist.vbs script in
the same
' folder as this script to generate the processlist.txt file, which will list all currently running processes.
' The processlist.txt may be modified as necessary to add authorized processes
MonitorProcesses = 1

' The rate of scanning affects the monitoring of the CD-ROM, floppy, scheduler, and new processes and is
' set by the ScanSpeed variable set below. Note that short scan rates may incur significant processor
' overhead. However, processes with a lifespan shorter than the chosen ScanSpeed might not be
detected.
' Use the Task Manager to monitor the system overhead being used by wscript.exe and winmgmt.exe
processes.
```

```
' This number should be experimented with to determine the optimum number for each installation.
ScanSpeed = 5

'Initialize variables
varCounter = 0
varCounterProcess = 0

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set WshShell = WScript.CreateObject("WScript.Shell")

result = WshShell.Popup("Monitor script is starting. Please wait...", 5)

' Get the path to this script
strScriptPath = Left(WScript.ScriptFullName, InStrRev(WScript.ScriptFullName, "\"))
strLogFileName = strScriptPath & "monitor.log"
strProcessFilePath = "ProcessList.txt"

' Check to determine if the log file exists
If Not objFSO.FileExists(strLogFileName) Then
   Set objFile = objFSO.CreateTextFile(strLogFileName, True)
   objFile.Close
End If

' Connect to WMI and obtain an SWbemServices objects, used by multiple monitors
Set Service = GetObject("WinMgmts:{impersonationLevel=impersonate, (security)}")

If MonitorEventLog = 1 Then

   ' Create the sink object that will receive the Events
   Set EventsSink = WScript.CreateObject("WbemScripting.SWbemSink", "EventsSink_")

   ' Connect to WMI and obtain an SWbemServices objects
   Set Events = GetObject("WinMgmts:{impersonationLevel=impersonate, (security)}" & _
            "!//./root/cimv2")

   ' Set up the event selection. EventSink_OnObjectReady will be called when
   ' one of the specified Win32_NTLogEvent occurs
   strQuery = "SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA
'Win32_NTLogEvent'" & _
          "and TargetInstance.EventCode = 517 or TargetInstance.EventCode = 608" & _
          " or TargetInstance.EventCode = 610 or TargetInstance.EventCode = 611" & _
          " or TargetInstance.EventCode = 612 or TargetInstance.EventCode = 624" & _
          " or TargetInstance.EventCode = 625 or TargetInstance.EventCode = 631" & _
          " or TargetInstance.EventCode = 632 or TargetInstance.EventCode = 635" & _
          " or TargetInstance.EventCode = 636 or TargetInstance.EventCode = 639" & _
          " or TargetInstance.EventCode = 640 or TargetInstance.EventCode = 641" & _
          " or TargetInstance.EventCode = 642 or TargetInstance.EventCode = 643" & _
          " or TargetInstance.EventCode = 644"

   If MonitorAllFailed = 1 Then
      strQuery = strQuery & " or TargetInstance.Type = 'audit failure'"
   End If
```

```
    If MonitorAllWarnings = 1 Then
        strQuery = strQuery & " or TargetInstance.Type = 'warning'"
    End If

    If MonitorAllErrors = 1 Then
        strQuery = strQuery & " or TargetInstance.Type = 'error'"
    End If

    Events.ExecNotificationQueryAsync EventsSink, strQuery

End If

If MonitorRegistry = 1 Then

    ' Create the sink object that will receive registry change events.
    Set RegistrySink = WScript.CreateObject("WbemScripting.SWbemSink", "RegistrySink_")

    ' Connect to WMI and obtain an SWbemServices object
    Set Registry = GetObject("WinMgmts:{impersonationLevel=impersonate, (security)}" & _
                "!//./root/default")

    ' Set up the registry key selection. RegistrySink_OnObjectReady will be called when
    ' a change is detected at any of the keys listed below
    Registry.ExecNotificationQueryAsync RegistrySink, _
    "SELECT * FROM RegistryKeyChangeEvent " & _
        "Where (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'Software\\Microsoft\\Windows\\CurrentVersion\\Run') " & _
        "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce') " & _
        "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath = 'Software\\Microsoft\\Windows
NT\\CurrentVersion\\Winlogon') " & _
        "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath = 'System\\CurrentControlSet\\Control\\Lsa') "
& _
        "OR (hive = 'HKEY_LOCAL_MACHINE' And Keypath =
'System\\CurrentControlSet\\Control\\SecurityProviders')"

End If

If MonitorCDROM = 1 Then

    ' Create the sink object that will receive the CD-ROM events.
    Set CDRomSink = WScript.CreateObject("WbemScripting.SWbemSink", "CDRomSink_")

    ' CDRomSink_OnObjectReady will be called when a CD-ROM is put in the drive.
    Service.ExecNotificationQueryAsync CDRomSink, _
        "SELECT * FROM __InstanceModificationEvent WITHIN " & ScanSpeed & " WHERE TargetInstance
ISA 'Win32_LogicalDisk' " & _
            "and TargetInstance.Name='" & CDDriveLetter & "' and PreviousInstance.Size !=
TargetInstance.Size and TargetInstance.Size > 0"

End If

If MonitorFloppy = 1 Then
```

```
    ' Create the sink object that will receive the floppy events.
    Set FloppySink = WScript.CreateObject("WbemScripting.SWbemSink", "FloppySink_")

    ' FloppySink_OnObjectReady will be called when a floppy disk is put in the drive.
    Service.ExecNotificationQueryAsync FloppySink, _
        "SELECT * FROM __InstanceModificationEvent WITHIN " & ScanSpeed & " WHERE TargetInstance
ISA 'Win32_LogicalDisk' " & _
            "and TargetInstance.Name='A:' and PreviousInstance.Size != TargetInstance.Size and
TargetInstance.Size > 0"

End If

If MonitorScheduler = 1 Then

    ' Create the sink object that will receive the task scheduler events.
    Set TaskSink = WScript.CreateObject("WbemScripting.SWbemSink", "TaskSink_")

    ' TaskSink_OnObjectReady will be called when a job is scheduled.
    Service.ExecNotificationQueryAsync TaskSink, _
        "select * from __InstanceCreationEvent WITHIN " & ScanSpeed & " where targetinstance isa
'Win32_ScheduledJob'"

End If

If MonitorProcesses = 1 Then
' This section monitors processes as they are created.
'Initialize Variables
varCounterProcess = 0
varProcessApproved = 0

    ' Create the sink object that will receive the new process events.
    Set ProcessSink = WScript.CreateObject("WbemScripting.SWbemSink", "ProcessSink_")

    ' Build known good process array from process list file
    If objFSO.FileExists(strScriptPath & strProcessFilePath) Then
        Set objTextStream = objFSO.OpenTextFile(strScriptPath & strProcessFilePath)

        Do While Not objTextStream.AtEndOfStream
            strInputLine = LCase(objTextStream.ReadLine)
            If Len(Trim(strInputLine)) > 0 And Left(strInputLine, 1) <> ";" Then
                strSplitString = Split(strInputLine, "#")
                aryProcessName(varCounter) = strSplitString(0)
                aryProcessExecutablePath(varCounter) = strSplitString(1)
                aryProcessAlert(varCounter) = strSplitString(2)
                varCounter = varCounter + 1
            End If
        Loop
        objTextStream.Close

        ' Connect to WMI and obtain an SWbemServices object
        Set ProcessList = GetObject("WinMgmts:{impersonationLevel=impersonate,
(security)}").ExecQuery("select * from Win32_Process")
```

```
        For Each Process In ProcessList
            Do Until varCounterProcess = varCounter
                If aryProcessName(varCounterProcess) = LCase(Process.Name) Then
                    varProcessApproved = 1
                    Exit Do
                End If
                varCounterProcess = varCounterProcess + 1
            Loop

        If varProcessApproved = 1 Then
            'Process is approved. Now check to see if an alert should still be generated
            If aryProcessAlert(varCounterProcess) = 1 Then
                msg = "An approved process has been created. " & Chr(10) & LCase(Process.Name) &
Chr(10) & Chr(13) & _
                      "This process is listed in the ProcessList.txt file, but an alert has been requested
because the line that lists this" & _
                      "  process ends with a 1. To avoid this alert in the future, edit the ProcessList.txt file
and change the 1 to a 0."
                result = LogEntry(True, msg, True)
            End If
        Else
            'Process is not approved. Log and generate alert.
            msg = "An un-approved process has been created. " & Chr(10) & LCase(Process.Name) &
Chr(10) & Chr(13) & _
                  "To avoid this alert in the future, add this process to the ProcessList.txt on the system
that generated this alert."
            result = LogEntry(True, msg, True)
        End If

        'Reinitialize Variables
        varCounterProcess = 0
        varProcessApproved = 0

        Next

        result = LogEntry(True, "All currently running processes have been checked.", False)

        Service.ExecNotificationQueryAsync ProcessSink, _
        "select * from __InstanceCreationEvent WITHIN " & ScanSpeed & " where targetinstance isa
'Win32_Process'"

        Else
            WScript.Echo ("ProcessList.txt does not exist. To monitor processes, please exit this script and
run the process.vbs file.")

    End If

End If

'This function performs all logging and alerting.
Public Function LogEntry(blankline, msg, alert)
    Set Log = objFSO.OpenTextFile(strLogFileName, 8, False)
```

```
    If blankline Then
        Log.WriteBlankLines (1)
        Log.WriteLine (Date & " at " & Time)
        Log.WriteLine (msg)
        Log.Close
    Else
        Log.WriteLine (Date & " at " & Time)
        Log.WriteLine (msg)
        Log.Close
    End If

    If alert Then
        WshShell.LogEvent 0, msg, AdministrativeAlert
    End If

End Function

' This is the subroutine executed when a Win32_NTLogEvent event matching the selection criteria occurs.
Sub EventsSink_OnObjectReady(objObject, objAsyncContext)
    msg = objObject.TargetInstance.Type & Chr(10) & Chr(13) & objObject.TargetInstance.Message
    result = LogEntry(True, msg, True)
End Sub

' This is the subroutine executed when a registry change is detected.
Sub RegistrySink_OnObjectReady(objObject, objAsyncContext)
    result = LogEntry(True, "Registry change detected at: " & objObject.hive & "\" & objObject.KeyPath,
True)
End Sub

'This is the subroutine executed when a floppy is inserted.
Sub FloppySink_OnObjectReady(objObject, objAsyncContext)
    result = LogEntry(True, "Floppy disk inserted!", True)
End Sub

'This is the subroutine executed when a CD-ROM is inserted.
Sub CDRomSink_OnObjectReady(objObject, objAsyncContext)
    result = LogEntry(True, "CD-ROM inserted!", True)
End Sub

' This is the subroutine executed when a new scheduled task is detected.
Sub TaskSink_OnObjectReady(objObject, objAsyncContext)
    result = LogEntry(True, "The command " & objObject.TargetInstance.Command & " has been
scheduled.", True)
End Sub

' This is the subroutine executed when a new Win32_Process is detected.
Sub ProcessSink_OnObjectReady(objObject, objAsyncContext)
'Initialize Variables
varCounterProcess = 0
varProcessApproved = 0

    varProcessPathNull = objObject.TargetInstance.ExecutablePath
```

```
    If IsNull(varProcessPathNull) Then
        Do Until varCounterProcess = varCounter
            If aryProcessName(varCounterProcess) = LCase(objObject.TargetInstance.Name) Then
                varProcessApproved = 1
                Exit Do
            End If
            varCounterProcess = varCounterProcess + 1
        Loop
    End If

    If Not IsNull(varProcessPathNull) Then
        Do Until varCounterProcess = varCounter
            If LCase(aryProcessName(varCounterProcess)) = LCase(objObject.TargetInstance.Name) And
aryProcessExecutablePath(varCounterProcess) = LCase(objObject.TargetInstance.ExecutablePath) Then
                varProcessApproved = 1
                Exit Do
            End If
            varCounterProcess = varCounterProcess + 1
        Loop
    End If

    If varProcessApproved = 1 Then
        'Process is approved. Now check to see if an alert should still be generated
        If aryProcessAlert(varCounterProcess) = 1 Then
            msg = "An approved process has been created. " & Chr(10) &
LCase(objObject.TargetInstance.Name) & " " & _
                    LCase(objObject.TargetInstance.ExecutablePath) & Chr(10) & Chr(13) & _
                    "This process is listed in the ProcessList.txt file, but an alert has been requested because
the line that lists this" & _
                    "  process ends with a 1. To avoid this alert in the future, edit the ProcessList.txt file and
change the 1 to a 0."
            result = LogEntry(True, msg, True)
        End If
    Else
        'Process is not approved. Log and generate alert.
        msg = "An un-approved process has been created. " & Chr(10) &
LCase(objObject.TargetInstance.Name) & " "" & _
                LCase(objObject.TargetInstance.ExecutablePath) & """ & Chr(10) & Chr(13) & _
                "To avoid this alert in the future, add this process to the ProcessList.txt on the system that
generated this alert."
        result = LogEntry(True, msg, True)
    End If

End Sub

result = LogEntry(True, "System monitoring started.", False)

WScript.Echo "System monitoring in progress. To stop monitoring, click OK."
```

## Appendix B

```
'Every time that this script is run, it will create a new ProcessList.txt file.
'Customizations that have been made to an existing ProcessList.txt file will be overwritten.

'This line forces us to declare all variables
Option Explicit

'Declare all variables
Dim objTextStream
Dim objFSO
Dim objFile
Dim WshShell
Dim result
Dim strProcessFilePath
Dim strScriptPath
Dim strSplitString
Dim ProcessList
Dim process

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set WshShell = WScript.CreateObject("WScript.Shell")

result = WshShell.Popup("Process list script is starting. Please wait...", 3)

'Get the path to this script
strScriptPath = Left(WScript.ScriptFullName, InStrRev(WScript.ScriptFullName, "\"))
strProcessFilePath = "ProcessList.txt"

'Check to see if a ProcessList.txt file already exists
If objFSO.FileExists(strScriptPath & strProcessFilePath) Then
   WScript.Echo ("ProcessList.txt file exists. Please delete or rename before running this script again!")
Else
   Set objFile = objFSO.CreateTextFile(strScriptPath & strProcessFilePath, True)
   objFile.Close

If objFSO.FileExists(strScriptPath & strProcessFilePath) Then
   Set objTextStream = objFSO.OpenTextFile(strScriptPath & strProcessFilePath, 8, False)
      objTextStream.WriteLine (";This list has been generated by the processlist.vbs script.")
      objTextStream.WriteLine (";Other processes may be added to this list as necessary.")
      objTextStream.WriteLine (";The number is necessary at the end of each line.")
      objTextStream.WriteLine (";A 0 indicates that no alert should be generated.")
      objTextStream.WriteLine (";A 1 indicates that an alert should be generated.")
      objTextStream.WriteLine (";Use a semi-colon to comment out a line.")
      objTextStream.WriteBlankLines (1)
   objTextStream.Close
End If

' Connect to WMI and obtain an SWbemServices object
```

```
Set ProcessList = GetObject("WinMgmts:{impersonationLevel=impersonate,
(security)}").ExecQuery("select * from Win32_Process")

For Each process In ProcessList
   If objFSO.FileExists(strScriptPath & strProcessFilePath) Then
      Set objTextStream = objFSO.OpenTextFile(strScriptPath & strProcessFilePath, 8, False)
         objTextStream.WriteLine (LCase(process.Name) & "#" & LCase(process.ExecutablePath) & "#" &
"0")
      objTextStream.Close
   End If
Next

End If

WScript.Echo ("Process list script complete!")
```

## Appendix C

```
'This line forces declaration of all variables
Option Explicit

'Declare all variables
Dim WshShell
Dim strQuery
Dim NTEvent
Dim Events
Dim msg

Set WshShell = WScript.CreateObject("WScript.Shell")

' Set up the event selection
strQuery = "SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA 'Win32_NTLogEvent'
and TargetInstance.SourceName = 'WSH'"

' Connect to WMI and obtain an SWbemServices objects
Set Events = GetObject("WinMgmts:{impersonationLevel=impersonate,
(security)}").ExecNotificationQuery(strQuery)

Do
   Set NTEvent = Events.nextevent
   If Err <> 0 Then
      WScript.Echo Err.Number, Err.Description, Err.Source
      Exit Do
   ElseIf NTEvent.TargetInstance.Message <> Empty Then
         msg = "Alert from " & NTEvent.TargetInstance.ComputerName & Chr(10) &
NTEvent.TargetInstance.Message
      WScript.Echo msg
   Else
      WScript.Echo "Event received, but it did not contain a message."
   End If
Loop
```

```
WScript.Echo "finished"
```

## Appendix D

Microsoft Knowledge Base article Q137890, HOWTO: Create a User-Defined Service describes one value under the Parameters key called Application. You will need to create an additional value called AppParameters under the Parameters key. Both values need to be string values. You will need to customize the path information of each value.

SystemMonitor.vbs service parameters

Value Name: Application
Data Type: REG_SZ
String: C:\winnt\system32\wscript.exe

Value Name: AppParameters
Data Type: REG_SZ
String: C:\Monitor\SystemMonitor.vbs

Be sure to set the startup type to automatic, and assign a valid network user account and password for the service.

Administrative.vbs service parameters

Value Name: Application
Data Type: REG_SZ
String: C:\winnt\system32\wscript.exe

Value Name: AppParameters
Data Type: REG_SZ
String: C:\Monitor\administrative.vbs

Be sure to set the startup type to automatic, and select the checkbox to allow the service to interact with the desktop.