Global Information Assurance Certification Paper

JAMES DEBARTOLO, JR

MARCH 30, 2004

GSEC PRACTICAL ASSIGNMENT

VERSION 1.4B

Detecting Modifications to your Servers

Table of Contents

Abstract

Monitoring servers is time consuming task. One way to reduce the load is to automate as much of the process as possible. There are many tasks that can be automated including, but not limited to virus definition updating, scanning for vulnerable systems, auditing event logs, and scanning for changes. This paper will document how you can use scripting to automate scanning systems for changes to processes, services, and monitoring event logs for errors and logon failures. The script discussed in this paper is designed for Windows operating systems. In order to scan systems and collect information the script will use Windows Management Instrumentation (WMI). WMI is Microsoft's initiative to managing systems and will be explained in more detail later in this paper. After reading this paper you will be on your way to automating the monitoring of your servers.

Why monitor your systems?

In today's world, people's use of the Internet continues to grow. The quantity of people online has leveled off, but what they do has continued to increase. Activity includes e-mail, research for work, purchasing, banking… At the same time the amount of malware of various types continues to grow. Workers will continue to use the Internet to do their job and other online activities. That means security professionals will need to continue to defend their systems. Many times this is difficult with the number of servers and workstations administrators and security professionals need to protect.

The amount of W32 viruses and worms documented has gone up dramatically in every 6 month reporting period since 2001. At the same time the time delay from vulnerability discovery to outbreak has gone down. So not only do we have more attacks to protect our network from we have less time to mitigate the vulnerabilities. Once servers are deployed the services and processes should remain relatively constant. Anti-virus services should remain running to keep your system protected, and new services or processes should not appear unless the system has been reconfigured to add a new capability. A new service or process could indicate the presence of a backdoor or some other malware. This change could also indicate that a new application is running. In either case a new vulnerability could be present on the system. One can argue that having virus protection, firewalls, and intrusion detection systems are enough protection. These systems generally detect known attack methods and will not detect new ones until signature files are updated. It is always good to practice defense in-depth. This helps protect your network in case one of your defenses fails. Also for those with limited funds the defense method in this paper is available on current Microsoft server products without any additional expense.

The Tools Used

The script described in this paper will use Windows Scripting Host (WSH), Microsoft® Visual Basic® Scripting Edition (VBScript), and Microsoft's Windows Management Instrumentation (WMI). WSH is the controller of Windows based script engines. It would be comparable to the MS-DOS command language and batch files. There are 2 versions of WSH Wscript.exe and Cscript.exe. Wscript.exe is the Windows-based version and Cscript.exe is the command-prompt-based version. VBScript is one of the scripting languages supported by WSH. WMI is the piece that allows us to manage and monitor Windows systems locally and remotely. It is a powerful component of VBScript technology. An explanation of what WMI is and the benefits of WMI provided by Microsoft are:

> Windows Management Instrumentation (WMI) makes managing Windows-based computers much more convenient than it has been in the past. WMI provides you with a consistent way to access comprehensive system management information and was designed, from the beginning, to work across networks. For system administrators managing Windows-based computers, understanding WMI is as important and useful as understanding the Active Directory® directory service.[1]

So as you can see Microsoft considers WMI to be very beneficial for management of Windows-based computers. WMI is very powerful and the scripts in this paper just take advantage of a small portion of WMI's capabilities. Many books on these subjects are also available at your favorite bookstore. Learning what you can do with WMI and scripts does take time, but the benefits can be huge. Microsoft does provide some help with their TechNet Scripting center and VBScript documentation online. The URLs for both of these are provided at the end of this paper. Some of the examples from the TechNet Scripting center were usefully in creating the script described in this paper. You can manage users accounts, services, query processes, map drives, monitor event logs, … The script I will present in this paper will focus on monitoring services, processes, finding errors in the event log, and extracting failed logons from the security logs.

[1] http://www.microsoft.com/resources/documentation/windows/2000/server/scriptguide/en-us/sas_wmi_dieu.mspx

What is WMI?

OK we now know WMI is important and it can help us but what is it. WMI was originally release in 1998 with Windows NT Service Pack 4. It was an add-on component. WMI is now included with Windows 2000, Windows XP, and Windows 2003. A download is also available for Windows 95 and Windows 98. Prior to WMI the method to manage servers in your enterprise was through the graphical tools provided by Microsoft and other third party tools. Your other option was programmatically using Win32 APIs. This did not give administrators many options to automate management of servers. WMI solves this problem by exposing most Windows resources. Administrators now have a framework that exposes these resources and can use scripts to manage windows servers and workstations. As long as you have administrative rights to a windows system you can retrieve information such as free disk space, running services, event log entries, installed memory, and many other system resources. Many of these resources can also be managed. Other scripting and programming languages supporting COM automation can also access resources through WMI. Other management applications are also able to interface with WMI including third party applications. WMI is Microsoft's implementation of Web-Based Enterprise Management (WBEM). "WBEM is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments."[2]

What about SNMP. SNMP is Simple Network Management Protocol. As the name implies SNMP is a protocol and is simple. SNMP was originally developed to manage routers and other network related equipment. SNMP made its debut in 1988 when routers and other network equipment were less powerful than the devices of today. In 1988 these devices usually did not require nor could they handle a complex management protocol, so SNMP was a perfect fit. SNMP's simple nature and close tie to its protocol make it more difficult for it to work in today's complex network. WBEM on the other hand is an initiative and is not tied to a protocol. If in the future a different network protocol is introduced a specification for WBEM transmitted over this protocol can be introduced. This is a flexibility SNMP does not have. Microsoft has provided a SNMP provider so that SNMP devices can be managed using WMI.

What we will audit and why?

As system administrators concerned about security we need to know what is going on with the servers we manage. Any changes need to be noted and we

[2] http://www.dmtf.org/standards/wbem

need to determine what caused the change. We also need to determine what the effect on the server and network will be. One area to look for changes is in processes and services that are running. New processes or services could indicate the presence of malware. Services or processes no longer running could indicate a security issue. Would your security concerns increase if your anti-virus was no longer running? If your Host Base Intrusion Detection system quit running would you be concerned? New processes or services could indicate a new application has been added to a server. This could add new vulnerabilities to a server. Would you be concerned if someone added IIS server to your Window 2003 based file server? Errors in the application or system event logs could also be a cause for concern. Do you suddenly have an application logging errors in the event log. Could these errors indicate an availability issue with a network application? Do you have failed logons coming from a remote system? This could indicate a hacker attempting to gain access. It could also indicate an infected machine looking for other machines on the network to infect. Perhaps a machine that ran disconnected from the network is now on the network. In the past since this machine was not connected to the network the administrator may have determined there was no need to install anti-virus software. When the machine is connected to the network if anti-virus software is not installed the machine may become infected. This machine does not need to be local it could be at a remote site on your WAN or on a partner site over an extranet. You may not have control over this machine, but should be concerned because it could attempt to infect the PCs and servers under your responsibility.

One of the first tasks you will need to do is determine a baseline for each machine you will audit. Establishing a baseline will also help you determine what is running on a machine and how it should be configured. Do you need IIS on a file server? To limit the vulnerabilities on a machine you should only run the services required so the machine can do its job. Similar to the principle of least privilege when assigning rights to users servers should only run the programs necessary for them to function as intended. This might be the most time consuming step, but it is important to be thorough. Good documentation is necessary so that what is normal or authorized for each server is known and you have something to compare to the current state. The script discussed in the paper will help you set up your baseline files. The first time the script is run you can use the output files for each server to set up baseline files. This will be discussed later on in this paper. You still need to take this extra step; the script will just help you with documenting your servers and getting the files in the correct format.

The script

Before you start working on the script you will need to setup your directory structure to work with the script, log files, baseline files, and list of servers to audit. Following is the structure I use.

- UNCPath\Scripts\audit
- UNCPath\Logs\ServerAudit

Based on the number and type of servers managed it may make sense to structure the directories differently. I have kept it fairly simple by having just two directories one for the script and one for any reports or logs. I do have the directory structure set up so that I can divide my scripts into multiple categories and do the same for my logs. I can create user management scripts and put them in a different Script subdirectory.

The script uses several input files. The first is a list of servers to audit. The rest of the input files are baseline files for each server represented in the list of servers to audit. The file names and path are as follows:

- UNCPath\Scripts\audit\ServerList.txt
- UNCPath\Logs\ServerAudit\baseline_servernameProcess.tsv
- UNCPath\Logs\ServerAudit\baseline_servernameServices.tsv

The serverlist.txt file contains server IP addresses and names separated by tabs similar to a hosts file. An example is provided in Figure 1.

```
10.1.1.3        ServerA
10.1.1.4        ServerB
10.1.1.5        ServerC
```

Figure 1 A sample ServcerList.txt file

The baseline_servernameProcess.tsv and baseline_servernameServices.tsv list the authorized processes and services for servername. The servername portion of these file names represent the name of the server in the ServerList.txt file. When the script runs the baseline files are used to compare against the current state. For example baseline_ServerAProcess.tsv would be the required input file when the script audits ServerA. In the rest of the paper I will use the text servername with the understanding it represents the name of the server audited. A few lines from baseline_servername.Process.tsv are in Figure 2 and a few lines from baseline_servername.Services.tsv are in Figure 3.

```
C:\WINNT\System32\smss.exe
C:\WINNT\system32\csrss.exe
C:\WINNT\system32\winlogon.exe
C:\WINNT\system32\services.exe
C:\WINNT\system32\lsass.exe
```

Figure 2 A few lines from a baseline file for running processes.

```
C:\WINNT\System32\services.exe          Auto   LocalSystem Running
C:\WINNT\system32\CIMNtfy\CIMNtfy.EXE       Auto   LocalSystem Running
C:\WINNT\System32\cisvc.exe     Manual         LocalSystem Stopped
C:\WINNT\system32\clipsrv.exe   Manual         LocalSystem Stopped
C:\WINNT\System32\services.exe          Auto   LocalSystem Running
C:\WINNT\System32\services.exe          Auto   LocalSystem Running
C:\WINNT\System32\llssrv.exe    Manual         LocalSystem Stopped
C:\WINNT\System32\services.exe          Auto   LocalSystem Running
```

Figure 3 A few lines from baseline file for services configuration.

As you can tell from Figure 2 a baseline_servernameProcess.tsv would list all of the processes that were running when the baseline for servername was established. These should be all of the processes that are authorized to run on the server. In a similar fashion a baseline_servernameServices.tsv would list the startup state and current state of all services when the baseline on servername was established. These are the authorized services on servername. The baseline files are the ones you will want to compare the current list of processes and services to each time the script runs. Special care will need to be taken to make sure there are no unauthorized modifications to the baseline files. If someone had access to the baseline files, modifications could be made that would hide or mask malicious changes to a server. The files generated when the script runs, are:

- UNCPath\Logs\ServerAudit\servernameProcess.tsv
- UNCPath\Logs\ServerAudit\servernameServices.tsv
- UNCPath\Logs\ServerAudit\Changes.log
- UNCPath\Logs\ServerAudit\ErrorReport.log
- UNCPath\Logs\ServerAudit\EventErrors.log
- UNCPath\Logs\ServerAudit\SecurityFailures.log

ServernameProcess.tsv contains the list of Processes that were running when the script ran. ServernameServices.tsv contains the list of services and their startup state and current state when the script ran. For each server that is audited there is a set of files where servername is the name of the server in the server list file. These are the files compared to the baseline files for the respective server. Any differences are then logged in the changes.log file that can be used for a quick inspection for any discrepancies. If no baseline file exists when the script runs an error will be logged in ErrorReport.log and the script will continue processing. You can then use the newly created servernameProcess.tsv and servernameServices.tsv to establish baseline files. All running processes will need to be authorized. In a similar fashion the current state and startup state of any service will need to be authorized. Once

authorized, you only need to change the name of the files by inserting 'baseline_' before the current name of the file.

How the script works

I will discuss key subroutines of the script in the following sections. At the end of the paper the script will be included in its entirety in Listing 1. The first subroutine I will discuss is CheckProcesses

```
sub CheckProcesses()

1     Set objWMIProcess = GetObject("winmgmts:" _
2     & "{impersonationLevel=impersonate}!\\" & strComputerIP & "\root\cimv2")
3     Set colProcess = objWMIProcess.ExecQuery _
4     ("Select * from Win32_Process where ExecutablePath <> null")
5     For Each objProcess in colProcess
6             objTextFile.WriteLine objProcess.ExecutablePath
7     Next
8     objTextFile.Close
9     strServerLog = "c:\Project\Logs\ServerAudit\" _
10    & strComputerName & "Process.tsv"
11    strServerBaseline = "c:\Project\Logs\ServerAudit\BaseLine_" _
12    & strComputerName & "Process.tsv"
13    objTextFileChanges.Writeline "Checking for processes for changes."
14    call CheckFiles(strServerLog, strServerBaseline)
```

Line numbers were added to the code for clarity. In lines 1 – 2 we connect to the server represented by strComputerIP and open the WMI namespace (\root\cimv2). Next in lines 3 – 4 we select from the CIM Class, Win32_Process, all processes running on the system. Once we have the list of processes, we write them to file for later comparison in lines 5 – 7. In lines 9 – 10 we set the string strServerLog to the name and path of the file just created. Since the established naming convention for a baseline file is baseline pre-pended to the file name we now know the name of the baseline file. This is done in lines 11 – 12. Now we can call subroutine CheckFiles with the names of the current file and the baseline file. The result of this subroutine is a list of changes between the current file and the baseline. How this subroutine works will be explained later. Sample data from CheckProcesses subroutine would look like the data in Figure 4.

C:\WINDOWS\System32\smss.exe
C:\WINDOWS\system32\winlogon.exe
C:\WINDOWS\system32\services.exe
C:\WINDOWS\system32\lsass.exe
C:\WINDOWS\system32\svchost.exe

Figure 4 Some sample output from CheckProcesses subroutine

```
sub CheckServices()

1       Set objWMIService = GetObject("winmgmts:" _
2       & "{impersonationLevel=impersonate}!\\" & strComputerIP & "\root\cimv2")
3       Set colService = objWMIService.ExecQuery _
4       ("Select * from Win32_Service")
5       For Each objService in colService
6                       objTextFile.WriteLine objService.PathName & vbtab & _
7                       objService.StartMode & vbtab & _
8                       objService.StartName & vbtab & _
9                       objService.State
10      Next
11      objTextFile.Close
12
13      strServerLog = "c:\Project\Logs\ServerAudit\" & strComputerName _
14      & "Service.tsv"
15      strServerBaseline = "c:\Project\Logs\ServerAudit\Baseline_" _
16      & strComputerName & "Service.tsv"
17      objTextFileChanges.Writeline "Checking for services for changes."
18      call CheckFiles(strServerLog, strServerBaseline)
```

A word of caution on checking processes. On systems that support multiple users logging on such as Windows Terminal Server it would not be a good idea to run this check. On these systems since they support remote logons from many users they would most likely have different process running based on what the users are doing when they logon. It may also not be a good idea to run this check on servers that start and stop processes on a frequent basis. You would not be able to set a reliable baseline under either of these conditions.

Subroutine CheckServices works nearly identical to CheckProcesses. In lines 1 – 2 we connect to the server represented by strComputerIP and open the WMI namespace (\root\cimv2). Next in lines 3 – 4 we select from the CIM Class, Win32_Services, all services configured on the system. For each item in the selection we write the path to the service, startup mode, startup name or account name under which the service runs, and current state information to a text file in lines 5 – 10 for later comparison. Then we set variables to represent the paths to

the baseline file and the current file and call CheckFiles in lines 13 – 18. Output from the CheckServices subroutine would look similar to the sample data in Figure 5 below. Looking at line 1 in more depth we see that Application Layer Gateway Service path is “c:\ WINDOWS\System32\alg.exe”. It is set to manual start, runs under “NT Authority\LocalServices” and is currently running. If this is the authorized state for this service we would expect it to normally report the same information every time we checked.

C:\WINDOWS\System32\alg.exe Manual NT AUTHORITY\LocalService Running
C:\WINDOWS\system32\svchost.exe -k netsvcs Manual LocalSystem Stopped
C:\WINDOWS\System32\svchost.exe -k netsvcs Auto LocalSystem Running

Figure 5 Some sample output from the CheckServices subroutine.

Sub CheckEventLog(ComputerName) Part 1

```
1     Const CONVERT_TO_LOCAL_TIME = False
2
3     Set dtmStartDate = CreateObject("WbemScripting.SWbemDateTime")
4
5     DateToCheck = CDate(date -1 & " 05:30:00 AM")
6     dtmStartDate.SetVarDate DateToCheck, CONVERT_TO_LOCAL_TIME
7
8     Set objWMIEvent = GetObject("winmgmts:" _
9       & "{impersonationLevel=impersonate}!\\" & ComputerName & "\root\cimv2")
10    Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
11      & "Win32_NTLogEvent Where Logfile = 'System'" _
12      & " and Type = 'Error' and TimeWritten >= '" & dtmStartDate & "'")
13    For Each objEvent in colLoggedEvents
14              objTextFileEventErrors.writeline ComputerName
15              objTextFileEventErrors.writeline "Event date: " & objEvent.TimeWritten
16              objTextFileEventErrors.writeline "Type: " & objEvent.Type
17              objTextFileEventErrors.writeline "Description: " & objEvent.Message
18    Next
19    Set objWMIEvent = GetObject("winmgmts:" _
20      & "{impersonationLevel=impersonate}!\\" & ComputerName & "\root\cimv2")
21    Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
22      & "Win32_NTLogEvent Where Logfile = 'Application'" _
23      & " and Type = 'Error' and TimeWritten >= '" & dtmStartDate & "'")
24    For Each objEvent in colLoggedEvents
25              objTextFileEventErrors.writeline ComputerName
26              objTextFileEventErrors.writeline "Event date: " & objEvent.TimeWritten
27              objTextFileEventErrors.writeline "Type: " & objEvent.Type
28              objTextFileEventErrors.writeline "Description: " & objEvent.Message
29    Next
```

In the first part of subroutine CheckEventLog the script sets the date and time, so that events after a certain time are selected. This is done in lines 3 – 6, in the code included here the date to check is set for 5:30 AM yesterday. The date would need to be set according to the time the script will be run and the frequency. If this script was run multiple times per day you would want to set the dtmStartDate variable to the last time the script was run. This way you can search for errors that have happened since the last time the script was run. In lines 8 – 9 we connect to the server represented by strComputerIP and open the WMI namespace (\root\cimv2). Next in lines 3 – 4 we select from the CIM Class, Win32_NTLogEvent, all events meeting our criteria. This select statement filters for events from the System EventLog with a type of error and those events that occurred after the date and time specified in dtmStartDate. The For Next loop in lines 13 – 18 loops through all of the events that meet the selection criteria and log the computer name, date and time of the event, event type, and a description of the event. Lines 19 – 29 repeat the process just described except events are selected from the application log. An example of output from events that meet the criteria is in Figure 6. This error happened on February 19, 2004 at 4:57:11 AM local time. The type of event was error. The error in this event was related to a trust relationship with another domain that was not available. My experience so far has been that this script has been very successful in reporting errors on the servers I manage. The errors are logged in one convenient report that is in my inbox when I arrive at work. I then go through the report and determine what servers need attention.

ServerA
Event date: 20040219045711.000000-300
Type: error
Description: No Windows NT Domain Controller is available for domain DomainA. (This event is expected and can be ignored when booting with the 'No Net' Hardware Profile.) The following error occurred:
There are currently no logon servers available to service the logon request.

Figure 6 Example log entry of an Event Log error

```
Sub CheckEventLog(ComputerName)  Part 2

1       Set objWMIEvent = GetObject("winmgmts:" _
2        & "{impersonationLevel=impersonate}!\\" & ComputerName _
3        & " \root\cimv2")
4       Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
5        & "Win32_NTLogEvent Where Logfile = 'Security'" _
6        & " and Type = 'audit failure' and TimeWritten >= '" & dtmStartDate & "'")
7       For Each objEvent in colLoggedEvents
8               If instr(objEvent.Message, "Logon Failure") > 0 then
9               objTextFileSecurityFailures.writeline ComputerName
10                      objTextFileSecurityFailures.writeline "Event date: " _
11                       & objEvent.TimeWritten
12                      objTextFileSecurityFailures.writeline "Type: " & objEvent.Type
13                      objTextFileSecurityFailures.writeline "Description: " _
14                       & objEvent.Message
15              End If
16      Next
```

The remaining part of subroutine CheckEventLog queries the security event log. In the select statement in lines 4 – 6 the notable difference from the prior queries is the type of event it selects. The query is looking for 'audit failure' events in the security log. Within the For Next loop in lines 7 – 16 the script filters all of the events that met the select criteria for those with "Logon Failure" in the description. The resulting report from this script has been the most beneficial. It has so far revealed several failed logon attempts that were not caused by what I expected. Instead of human intruders there have been a couple of cases of PCs infected with worms looking for other servers or PCs to infect. The attempts have been made from outside my area of responsibility. This proves no matter how good your security measures are you still have to worry about others that are perceived to come from trusted locations such as corporate WAN, corporate dialup/VPN, or partners connected via an extranet. We have multiple defenses in place so these attempts to spread the worm were stopped and I was able to report to individuals who could contact the administrators responsible for the infected PCs. In the future should our other defenses fail it would be advantageous to have these other situations cleaned up before that happens. It is to our benefit to practice defense in-depth.

On Windows 2000 and earlier systems you need to keep in mind that the default installation does not turn on auditing security events. You will need to turn this on so that these events are logged. For logon/logoff events you will want to log success and failure events. In the event that there are logon failures these will be logged when the script runs. You may want to manually check the logs and see if after a failure if there is a successful logon. This could indicate that your system was compromised. If a hacker does penetrate your system they may

attempt to cover their tracks by clearing the event log. This is logged as Event 517, “The Audit log was cleared”, in the security event log.

```
sub CheckFiles(ServerLog,ServerBaseline)

1       If objFSO.FileExists(ServerBaseline) Then
2               Set objTextFileLog = objFSO.OpenTextFile(ServerLog)
3               Set objTextFileBaseline = objFSO.OpenTextFile(ServerBaseline)
4               strFile = objTextFileLog.Readall
5               Do Until objTextFileBaseline.atEndOfStream
6                       strBaseline = objTextFileBaseLine.Readline
7                       If instr(StrFile, strBaseline) = 0 then
8                               objTextFileChanges.Writeline strComputerName _
9                                 & " *** " & strBaseline _
10                                & " *** baseline condition does not match current."
11                      End If
12              Loop
13              objTextFileLog.Close
14              objTextFileBaseline.Close
15              Set objTextFileLog = objFSO.OpenTextFile(ServerLog)
16              Set objTextFileBaseline = objFSO.OpenTextFile(ServerBaseline)
17              strFile = objTextFileBaseline.Readall
18              Do Until objTextFileLog.atEndOfStream
19                      strTextFileLog = objTextFileLog.Readline
20                      If instr(StrFile, strTextFileLog) = 0 then
21                              objTextFileChanges.Writeline strComputerName _
22                                & " *** " & strTextFileLog _
23                                & " *** current condition does not match baseline."
24                      End If
25              Loop
26              objTextFileLog.Close
27              objTextFileBaseline.Close
28      Else
29              objErrorReport.writeline ServerBaseline & " does not exist."
30      End If
```

Subroutine CheckFiles takes as input two files, compares them, and reports the differences. The two files are the baseline file and the current file. This subroutine is called from both CheckProcesses and CheckServices. CheckFiles does two different file checks. In the first the current log file is completely read into strFile in line 4. The baseline file is read one line at a time until end of file in the Do Loop in lines 5 – 12 as each line is read it is tested to see if it is contained in the current log file. If the string is not found, the instr command in line 7 returns 0, you now have a process or service that has been removed or changed state. An example of this is shown in Figure 7. In the second file check, lines 15 – 25 the reverse is tested. This time the complete baseline file is read in and then the Do Loop cycles through the current log file one line at a time. If it finds any line in the current file that it does not locate in the baseline file you now have a process or service that has been added or the state has changed. An example

of this is in Figure 8. At this point you would want to investigate why the process is now running but was not when the baseline was authorized. To some degree there is some duplication by doing both of these checks. If the only change is that a service is now stopped instead of running, two entries will be logged in the report. There will be one entry because the baseline does not match what is in the current log file. The second entry will be logged because the current entry is not contained in the baseline file. What you can do is compare the two lines and determine what has changed. In the example in Figure 9 you will notice that the service alg.exe has stopped.

ServerA *** C:\WINDOWS\system32\spoolsv.exe *** baseline condition does not match current.

Figure 7 A process that was running in the baseline that is now missing.

ServerA *** C:\WINDOWS\system32\clipsrv.exe *** current condition does not match baseline.

Figure 8 A process has been added.

ServerA *** C:\WINDOWS\System32\alg.exe Manual NT AUTHORITY\LocalServiceRunning *** baseline condition does not match current.
ServerA *** C:\WINDOWS\System32\alg.exe Manual NT AUTHORITY\LocalService Stopped *** current condition does not match baseline

Figure 9 Service is now stopped.

The CheckFiles subroutine does a fairly simple string comparison check that will help catch changes to our servers monitored. Any change in the string will be logged as a change. It could be the service or process is still running but the path has changed to a new location. Has a worm infected a system? If the current state of the service is stopped and the baseline condition is running will also be logged as a change. This is the example in figure 9. Did the service crash or has someone or something performed a denial of service? The logged changes will help you determine what needs a closer look. This should help you proactively administrate your servers before it becomes an availability issue for the users of the system. You will also be able to spend less time on servers that are running well.

One more report generated when running this script is ErrorReport.log. This report contains errors encountered when the script is run. There are two types of errors that might be generated when the script runs. The first type is “server IPAddress unavailable.” IPAddress is the IP address of the system that was unavailable when the script was executed. This error is logged if the script is unable to ping the computer. The second type of error that you could encounter is when baseline files for an audited system do not exist. This could indicate that you added a new system to monitor and did not create baseline files or that

something has happened to your baseline files. An example of ErrorReport.log is in Figure 10.

c:\Project\Logs\ServerAudit\BaseLine_ServerCProcess.tsv does not exist.

c:\Project\Logs\ServerAudit\Baseline_ServerCService.tsv does not exist.

server 10.3.8.6 unavailable.

Figure 10 Example ErrorReport.log.

Running the Script

The script is designed to run on a Windows 2003 Server or Windows XP Professional. Although the script is designed to run on Windows 2003 or Windows XP it is able to remotely audit Windows systems that support WMI. Ideally you may want to schedule them to run using Scheduled Tasks included in Microsoft's operating systems or any number of third party products. The script will need to run under an account that has administrative rights on the machines you intend to audit. More details on security will be in the following section on WMI Security. Another step you may consider is emailing these reports when the script completes so that they can be reviewed.

WMI and Security

Since WMI is very powerful and can control your enterprise you need to be concerned about security. Very similar to the way Windows NT, 2000, and 2003 maintain a list of who has access to what, WMI infrastructure maintains a list of who has access to what namespace. You can access and set the security by using the WMI control application. The WMI control application can be accessed from the Start menu, select Run, enter wmimgmt.msc, right-click WMI control, and select properties. This works for Windows 2000 and later. On Windows 95/98 and Windows NT from the Start menu, select Run, and enter wbemcntl. Vulnerabilities in DCOM can also affect WMI. Microsoft Security Bulletins MS03-026 and MS03-039 both dealt with buffer overflow vulnerabilities in DCOM. If your solution to mitigate these vulnerabilities was to shutdown DCOM you may have some problems with remote WMI queries. WMI is not the cause but is indirectly affected by vulnerability in DCOM. There has also been at least one worm, w32.HLLW.Cydog@mm written that attempts to use WMI to terminate processes running on a system. A WMI vulnerability is not exploited in this case but the power of WMI is used to further exploit the machine infected with the worm. I have found references to a couple of tools that will attempt cracking

Windows passwords using the WMI Service. A link to one of these tools, WMICracker.exe, is provided in the list of references.

Summary

Monitoring many servers is a time consuming task if done manually and it is bound to be skipped on days when other events take precedent. If you can automate the task you can increase the likelihood that the task will be completed on a regular basis. Then you can review the report and concentrate your efforts on the issues that need to be addressed. There are some simple tasks that can be done to monitor security breaches, configuration changes that can open up other vulnerabilities and system or application errors that can affect the availability of a system. The script discussed in this paper automates several tasks that a security professional or system administrator should do on a regular basis. After the script runs I have 4 reports ready for me to review:

- SecurityFailures.log
- EventErrors.log
- Changes.log
- ErrorReport.log

With these reports I can determine what course is necessary to make the proper correction. The Worldwide financial impact from major virus outbreaks has been over $10 billion every year since 1999[3]. The most recent outbreak of MyDoom alone is estimated at $4 billion Worldwide. MyDoom was also the fasting spreading worm ever released and was able to launch a successful denial-of-service attack against www.sco.com[4]. To me protecting our corporate networks is not an option but a requirement.

Not much work is needed to get this script working in your environment. I will summarize the steps you will need to take:

1) Run the script on Windows 2003 Server or Window XP Professional.
2) Audit machines running.
    a. Windows 2000 or higher.
    b. Windows NT 4 SP4 or higher with the WMI add-on installed.
    c. Windows 95/98 with the WMI installation package from Microsoft.
3) Setup some method of notification.
4) Run the script with administrative rights to the machines you want to audit.

Following these steps will get the script running in your environment and start saving you some time.

[3] http://www.computereconomics.com/article.cfm?id=936
[4] http://www.computereconomics.com/article.cfm?id=932

This script is just a start. There is more that can be done to enhance the automating of many of our administrative tasks. You may want to increase the frequency that the script runs to more quickly catch problems. Event Logs should also be collected and archived. Microsoft has a script in their TechNet Scripting Center that does the task. The link to this script is found in the list of references. There are many other scripts in the TechNet Scripting Center that can be used to make the life of an administrator a little easier. Commercial products are available as well that will monitor your event logs real time and archive the events. Another aspect of auditing you may want to consider is changes to the file system. Monitoring changes to a file system would be difficult to accomplish on your own. A product available from Tripwire Inc. is available to do this task. First you establish a snapshot of your system (baseline) and then at some interval you take a new snapshot of your system and compare it to the baseline snapshot. You would then need to setup some notification method that is triggered when a modification is detected. The principle is the same as the script discussed in this paper. Further details on this product are available at www.tripwire.com.

Listing 1 Complete audit script

```
Const ForReading = 1
Const FoobjTextFileChangesrWriting = 2
Const ErrorReport = "c:\Project\Logs\ServerAudit\ErrorReport.log"
Const ServerList = "c:\Project\Scripts\Audit\serverlist.txt"
Const PingComputer = "."
strLF = chr(10)

Set fsoServerList = CreateObject("Scripting.FileSystemObject")
Set tsoServerList = fsoServerList.OpenTextFile(ServerList)
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objErrorReport = objFSO.CreateTextFile(ErrorReport,True)
Set objTextFileChanges = objFSO.CreateTextFile("c:\Project\Logs\ServerAudit\Changes.log",
True)
Set objTextFileEventErrors =
objFSO.CreateTextFile("c:\Project\Logs\ServerAudit\EventErrors.log", True)
Set objTextFileSecurityFailures =
objFSO.CreateTextFile("c:\Project\Logs\ServerAudit\SecurityFailures.log", True)

Do Until tsoServerList.atEndOfStream
        strComputer = tsoServerList.readline
        DelimPos = Instr(strComputer, vbtab)
        strComputerIP = left(strComputer, DelimPos -1)
        strComputerName = right(strComputer, len(strComputer) - DelimPos)
        Set objWMIService = GetObject("winmgmts:\\" & PingComputer & "\root\cimv2")
        Set colComputers = objWMIService.ExecQuery _
          ("Select * from Win32_PingStatus where Address = '" & strComputerIP & "'")
        For Each objComputer in colComputers
                If objComputer.StatusCode = 0 Then
                        objTextFileChanges.Writeline "Checking " & strComputerName
                        Set objTextFile = objFSO.CreateTextFile("c:\Project\Logs\ServerAudit\" _
                          & strComputerName & "Process.tsv", True)
                        Call CheckProcesses
                        Set objTextFile = objFSO.CreateTextFile("c:\Project\Logs\ServerAudit\" _
                          & strComputerName & "Service.tsv", True)
                        Call CheckServices
                        Call CheckEventLog(strComputerName)
                        objTextFileChanges.Writeline "Done checking " & strComputerName
                        objTextFileChanges.Writeline
                Else
                        objErrorReport.writeline "server " & strComputerIP & " unavailable."
                End If
        Next
Loop
objTextFileChanges.Close
objTextFileEventErrors.Close
objTextFileSecurityFailures.Close

wscript.quit

sub CheckProcesses()

        Set objWMIProcess = GetObject("winmgmts:" _
```

```
        & "{impersonationLevel=impersonate}!\\" & strComputerIP & "\root\cimv2")
        Set colProcess = objWMIProcess.ExecQuery _
        ("Select * from Win32_Process where ExecutablePath <> null")
        For Each objProcess in colProcess
                objTextFile.WriteLine objProcess.ExecutablePath
        Next
        objTextFile.Close
        strServerLog = "c:\Project\Logs\ServerAudit\" _
        & strComputerName & "Process.tsv"
        strServerBaseline = "c:\Project\Logs\ServerAudit\BaseLine_" _
        & strComputerName & "Process.tsv"
        objTextFileChanges.Writeline "Checking for processes for changes."
        call CheckFiles(strServerLog, strServerBaseline)
End sub

sub CheckServices()

        Set objWMIService = GetObject("winmgmts:" _
        & "{impersonationLevel=impersonate}!\\" & strComputerIP & "\root\cimv2")
        Set colService = objWMIService.ExecQuery _
        ("Select * from Win32_Service")
        For Each objService in colService
                objTextFile.WriteLine objService.PathName & vbtab & _
                objService.StartMode & vbtab & _
                objService.StartName & vbtab & _
                objService.State
        Next
        objTextFile.Close

        strServerLog = "c:\Project\Logs\ServerAudit\" & strComputerName _
        & "Service.tsv"
        strServerBaseline = "c:\Project\Logs\ServerAudit\Baseline_" _
        & strComputerName & "Service.tsv"
        objTextFileChanges.Writeline "Checking for services for changes."
        call CheckFiles(strServerLog, strServerBaseline)

End Sub



sub CheckFiles(ServerLog,ServerBaseline)

        If objFSO.FileExists(ServerBaseline) Then
                Set objTextFileLog = objFSO.OpenTextFile(ServerLog)
                Set objTextFileBaseline = objFSO.OpenTextFile(ServerBaseline)
                strFile = objTextFileLog.Readall
                Do Until objTextFileBaseline.atEndOfStream
                        strBaseline = objTextFileBaseLine.Readline
                        If instr(StrFile, strBaseline) = 0 then
                                objTextFileChanges.Writeline strComputerName _
                                  & " *** " & strBaseline _
                                  & " *** baseline condition does not match current."
                        End If
                Loop
                objTextFileLog.Close
                objTextFileBaseline.Close
```

```
                Set objTextFileLog = objFSO.OpenTextFile(ServerLog)
                Set objTextFileBaseline = objFSO.OpenTextFile(ServerBaseline)
                strFile = objTextFileBaseline.Readall
                Do Until objTextFileLog.atEndOfStream
                        strTextFileLog = objTextFileLog.Readline
                        If instr(StrFile, strTextFileLog) = 0 then
                                objTextFileChanges.Writeline strComputerName _
                                  & " *** " & strTextFileLog _
                                  & " *** current condition does not match baseline."
                        End If
                Loop
                objTextFileLog.Close
                objTextFileBaseline.Close
        Else
                objErrorReport.writeline ServerBaseline & " does not exist."
        End If
End sub

Sub CheckEventLog(ComputerName)


        Const CONVERT_TO_LOCAL_TIME = False

        Set dtmStartDate = CreateObject("WbemScripting.SWbemDateTime")

        DateToCheck = CDate(date -1 & " 05:30:00 AM")
        dtmStartDate.SetVarDate DateToCheck, CONVERT_TO_LOCAL_TIME


        Set objWMIEvent = GetObject("winmgmts:" _
          & "{impersonationLevel=impersonate}!\\" & ComputerName & "\root\cimv2")
        Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
          & "Win32_NTLogEvent Where Logfile = 'System'" _
          & " and Type = 'Error' and TimeWritten >= '" & dtmStartDate & "'")
        For Each objEvent in colLoggedEvents
                objTextFileEventErrors.writeline ComputerName
                objTextFileEventErrors.writeline "Event date: " & objEvent.TimeWritten
                objTextFileEventErrors.writeline "Type: " & objEvent.Type
                objTextFileEventErrors.writeline "Description: " & objEvent.Message
        Next
        Set objWMIEvent = GetObject("winmgmts:" _
          & "{impersonationLevel=impersonate}!\\" & ComputerName & "\root\cimv2")
        Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
          & "Win32_NTLogEvent Where Logfile = 'Application'" _
          & " and Type = 'Error' and TimeWritten >= '" & dtmStartDate & "'")
        For Each objEvent in colLoggedEvents
                objTextFileEventErrors.writeline ComputerName
                objTextFileEventErrors.writeline "Event date: " & objEvent.TimeWritten
                objTextFileEventErrors.writeline "Type: " & objEvent.Type
                objTextFileEventErrors.writeline "Description: " & objEvent.Message
        Next

        Set objWMIEvent = GetObject("winmgmts:" _
          & "{impersonationLevel=impersonate}!\\" & ComputerName _
          & "\root\cimv2")
        Set colLoggedEvents = objWMIEvent.ExecQuery ("Select * from " _
```

```
        & "Win32_NTLogEvent Where Logfile = 'Security'" _
        & " and Type = 'audit failure' and TimeWritten >= '" & dtmStartDate & "'")
    For Each objEvent in colLoggedEvents
        If instr(objEvent.Message, "Logon Failure") > 0 then
            objTextFileSecurityFailures.writeline ComputerName
            objTextFileSecurityFailures.writeline "Event date: " _
              & objEvent.TimeWritten
            objTextFileSecurityFailures.writeline "Type: " & objEvent.Type
            objTextFileSecurityFailures.writeline "Description: " _
              & objEvent.Message
        End If
    Next
End Sub
```

References

Cole, Gwyn. “The Future of Systems Management.”
URL: http://www.wbem.co.uk/articles/SNMPvsWBEM.pdf (28 March 2004).

Madden, Mary. “America’s Online Pursuits.” Pew Internet & American Life Project. 22 December 2003. URL:http://www.pewinternet.org/reports/toc.asp?Report=106 (29 February, 2004).

McNab, Chris. “2004, year of the Chinese hacker.” 12 January 2004.
URL: http://www.oreillynet.com/pub/wlg/4148 (29 March 2004).

“My Doom Virus Update: Fastest Spreading Virus Ever.” February 2004.
URL: http://www.computereconomics.com/article.cfm?id=932 (29 February 2004).

Sevcenco, Serghei. “W32.HLLW.Cydog@mm.” 27 February 2003.
URL:
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.cydog@mm.html (28 March 2004).

“Symantec Internet Security Threat Report.” Malicious Code Trends. September 2003 URL:
https://enterprisesecurity.symantec.com/Content/displaypdf.cfm?SSL=YES&EID=0&PDFID=551&promocode=ITR (29 February 2004).

“The Cost Impact of Major Virus Attacks Since 1995.” February 2004.
URL: http://www.computereconomics.com/article.cfm?id=936 (29 February 2004).

“Using Tripwire for Servers for Damage Assessment and Remediation.” 2002
URL:
http://www.tripwire.com/files/literature/application_notes/appnote_damage_recovery.pdf (28 March 2004).

“Web-Based Enterprise Management (WBEM) Initiative.” 2004
URL: http://www.dmtf.org/standards/wbem (28 March 2004).

The following URLs are Copyright © 2004 Microsoft Corporation, One Microsoft Way, Redmond, Washington

Backup and Clear Event Logs
URL:
http://www.microsoft.com/technet/community/scriptcenter/logs/scrlog04.mspx (2 March 2004).

"Best Practices for Mitigating RPC and DCOM Vulnerabilities."
URL: http://www.microsoft.com/technet/security/topics/virus/bpdcom.mspx
(28 March 2004).

Stemp, Greg, et al. "WMI Scripting Primer: Part 1." 13 June 2002.
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting06112002.asp (28 March 2004).

Maintaining WMI Security
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/maintaining_wmi_security.asp (28 March 2004).

SNMP Provider
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/snmp_provider.asp?frame=true (28 March 2004).

TechNet Script Center
URL: http://www.microsoft.com/technet/community/scriptcenter/default.mspx (29 February 2004).

VBScript User's guide and Language reference
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtoriVBScript.asp (2 March 2004).

Windows Script Host Overview
URL:
http://www.microsoft.com/windowsxp/home/using/productdoc/en/default.asp?url=/WINDOWSXP/home/using/productdoc/en/wsh_overview.asp (2 March 2004).

Windows Management Intrumentation Overview
URL:
http://www.microsoft.com/resources/documentation/windows/2000/server/scriptguide/en-us/sas_wmi_dieu.mspx (1 March 2004).

Windows Management Instrumentation Scripting
URL:
http://www.microsoft.com/technet/prodtechnol/windows2000serv/maintain/featusability/wmiscrpt.mspx (29 February 2004).

Windows Management Instrumentation System Requirements
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/system_requirements.asp?frame=true (28 March 2004).