Free Centralized Event Log Collection and Analysis Solution for Windows

GIAC Certified Windows Security Administrator (GCWN)
Practical Assignment Version 4.0

Option 1; Identify a Windows Security Challenge, Assess Risk, Determine Mitigation, and Script a Solution

Submitted May 2, 2004
Barron Mertens

Table of Contents

ABSTRACT

The lack of built-in centralized event log collection and analysis in a standard Windows network is addressed. An approach utilizing VBScript, WMI, and a SQL back-end is fully explored, described, and implemented. The performance and functionality of this solution is then explored in a production environment. All code required to implement this solution is provided and the solution is implemented using only free tools or those already included with Windows. The architecture is such that an exchange or publishing of analysis logic between Administrators should be relatively easy.

1.1 Identification and Description of a Windows Security Challenge

It is impractical to expect a System Administrator to be actively monitoring the event logs on all servers at all times in their infrastructure. As the number of machines grow this becomes an increasing problem. Even an Administrator who some how finds the time to check event logs regularly, will still likely only be checking once a day or some similar period. In today's security climate where viruses spread around the globe in a matter of hours, this type of interval is not adequate.

One of the distinct differences between large IT organizations and small IT organizations is often the emphasis placed on processes by the larger organizations. In a large operation there are often operators dedicated to monitoring the current logs of running servers and equipment. In a large infrastructure with many servers there will likely be the means, people, and resources to justify an add-on tool to assist with this task. Unfortunately, many Windows infrastructures are small, less than 20 servers, and often have only one Administrator. It is difficult to justify the cost of these "Enterprise Grade" tools with their "Enterprise Grade" price tags. In some industries, such as Non-Profit Organizations or Educational Institutions, there may be a large number of servers being maintained with a very small budget. These "Enterprise Grade" tools all have one thing in common, they cost significant amounts of money. Some examples would be ~$700 per server and this is one of the root causes of the security challenge/issue being discussed. If a small Windows network can buy modest servers for $2000 - $3000, the cost of monitoring events represents a major additional cost component of 20% - 35%. The combination of cash strapped organizations and the relatively high price of current solutions tends to generate a situation where "Enterprise Grade" solutions are seen as too expensive and typically many organizations end up with no solution put in place at all.

The very verbose nature and lengthy process of sifting through the current log system manually using the Event Viewer in Windows is part of the reason most Administrators do not have time to do it. A means to describe a pattern or footprint that can then be searched for and alerted on would be a huge improvement. This would also allow the work/wisdom of a single Administrator to be utilized by others in their group and potentially shared within On-line Communities such as SANS, NTBUGTRAQ, TechNet etc.

In a modern complicated infrastructure it is not sufficient to simply monitor the "Security" log and think that you have an adequate solution. With security threats taking on so many forms (DoS, Trojans, Brute Force) a good "Security" solution will need to be able to collect and analyze not just "Security" logs but all the Windows Event logs, and if possible, a number of other Application specific logs (IIS, FTP, SMTP, ICF).   An intelligent analysis involving multiple logs at once will then be required to provide good results without generating a high number of “False Positives”. A condensed view without the “False Positives” and high levels of noise is a crucial factor for success.

Without the ability to centrally collect and analyse security and event logs there is a high chance that a network is essentially being driven “hands-off” and security issues will only be detected after the fact (if at all) and this will lead to security incidents becoming larger and more expensive due to the undetected or slowly detected nature. Early or pro-active detection of problems will always be the most cost-effective and most secure approach when you look at the big picture.

In many infrastructures the real-time monitoring is only for faults, not for activity. For example, if my network we very aggressively monitor for the availability of services and servers from multiple points inside and outside of our network. What we do not do, is actively monitor the current correct functioning of our network and try to determine when there is an abnormality. If someone was trying brute force password cracking on a single non-essential server, how long could it go unnoticed? If they are working on the Administrator account most networks do not have the tweak (for network logins) in place to allow this account to lockout due to failed password attempts so they could work on this account for as long as it takes. It they are trying thousands of attempts in a short time, and are locking out accounts, the activity will likely be noticed relatively quickly because it generates a fault (the locked out account). However if they are willing to work slowly and not cause many entries in the security log, this type of activity will likely not be detected even by a quick manual inspection of the security logs. A hacker wanting to try a null session exploit generates a footprint in the event logs, but how many Administrators are reading their logs carefully enough or often enough to notice? If a hacker has obtained a legitimate set of account credentials (through social engineering for example) and is illegally accessing your systems, a casual and manual inspection of event logs is unlikely to highlight this activity. If an Administrator has made an error in configuration and users (or hackers) are accessing resources that they should not, most monitoring systems will not highlight this as it is not a broken service, it is simply un-expected activity. The ability to sort out the noise in the Windows Event Logs would allow you to obtain a condensed view where unusual activity could be detected without manually reading thousands of log entries.

Essentially, without a central log collection/analysis system, the Admin will always be stuck looking at a partial, isolated, post-incident view of what is happening in their infrastructure. To have an increased level of security it is necessary to move into a nearly real-time, centrally collected and analyzed method of watching their infrastructure. To make the solution truly effective it needs to pro-actively seek notification of the system Administrator through whatever means are available. This not a very exciting or impressive security challenge, but it is nonetheless a crucial area for improvement that most if not all organizations could pursue. It will be of primary importance to any organizations not currently running an add-on package with these capabilities such as Microsoft Operations Manager. This will likely mean most small Windows networks are in need of a solution such as this.

1.2 Assess Risk and Identify Mitigation

The chosen security challenge is the lack of an affordable central pro-active log monitoring and analysis program in most Windows networks.

By resolving this challenge we will mitigate a major flaw in process that allows a great deal of activity to occur on Windows hosts without a pro-active method of noticing unusual or inappropriate activity. Much like an IDS (Intrusion Detection System) at the gateway, a central pro-active log monitoring and analysis program will comb through the wealth of information available looking for interesting activity at the host level. As an improvement in process, addressing this challenge has the potential to mitigate multiple current and future vulnerabilities and provide a means to detect inappropriate/interesting activity in general. This type of activity could take on a multitude of forms, everything from “password grinding” to un-authorized use of Administrator credentials. It would include hardware/software issues that could have relevance to security such as the failure of host-based firewall or anti-virus software to startup or even an unexpected shutdown of such a service.

Resolving this challenge introduces only a few new vulnerabilities into an existing Windows network. Having a rich central data store with scattered bits of information about all your servers means that a hacker wishing to learn about your network or hide illegal access would want access to your data store and would gain a great deal of useful info very quickly if they were to succeed. For example, you can generally tell which accounts are privileged and what privileges they hold from the security logs. Of course, a hacker would already need to have broken through a networks security layers somewhat if they are able to access a data store. I think the best approach here would be to keep a copy of the logs in their normal location and simply transfer a copy of the information to the central data store. This would actually make it less likely for a hacker who has some success at penetrating security being able to cover their tracks as there would be two copies of logs that would need to be deleted and they may be unaware of the second copy’s existence or may not have gained access to it. The transfer of log info across the network from all the monitored servers back to the central data store is a potential issue, a hacker would not need to gain access to the data store itself if they can successfully packet capture the network conversion between servers and the central data store. Once again the ability of a hacker to packet capture this conversation would in most cases represent a network that has already been compromised. It would be a good practice here to enable some encryption such as IPSec to encrypt the conversation between server and central data store so that even someone doing packet capture within your network would still only get encrypted packets. A properly implemented IPSec solution would be very effective at protecting the data in transit and could be done entirely with tools included with Windows.

8

The systems threatened by this challenge would range from the relatively unimportant to potentially mission-critical systems; this will vary from site to site. In general though, all Windows systems not being properly monitored are threatened by this challenge.

This challenge is generally a process issue and is not specific to a single vulnerability, it really represents the inability to detect or notice problems as they are occurring. Potentially it encompasses very lethal threats through to relatively benign threats and general informational events.

Management needs to understand that all those servers that they paid for and pay to maintain are generating a real-time stream of essential information. This information stream is a potential gold-mine that can be used to avoid and contain security/system problems all across the network. The challenge though is that it is cost-prohibitive to have staff manually monitor all the available information and it is even harder to intelligently analyses and combine information from multiple machines together to diagnose system-wide issues. By having a central data store with intelligent analysis and pro-active monitoring/alerting this wealth of information can be tapped into in a cost-effective manner and the objectives of tighter security and better uptime can be achieved. Management should also understand that large existing solutions exist such as Microsoft Operations Manager that addresses these issues quite well. The cost of these solutions though tends to be a major hurdle, Microsoft Operations Manager pricing is approximately $350 US per processor at retail and there are additional modules to pay for in order to monitor some apps such as Exchange Server. There are other commercial solutions available (see list elsewhere) and while they may or may not be slightly cheaper, there does not seem to be any commercial solutions available that come with a very low per server price. This causes problems for smaller operations in that capital is rarely available to purchase such software tools. The short take-home message for management is to deal with better monitoring up front and you will reap the rewards in reduced costs and improved operations down the road.

The behind the scenes nature of logs gathering will have little direct effect on users. The servers sending events to the central data store will bear the burden of some additional workload to submit events. If these servers are heavily loaded this additional workload could be noticed by end users as slower response times. The central data store will have an additional workload (if it is not put in place specifically for this task) and once again end users could notice a slowdown in response times. It is unlikely that any communication with end-users would be required unless the additional workload is causing response issues on overloaded servers. Hopefully the only difference end users would notice is an increased awareness on the part of the network administrators of the functioning of the network and a decrease in incidents that affect the network.

There are a number of commercial applications available that address this challenge, some more completely than others, but in general this is not a new or secret issue and vendors have responded with a number of tools to choose from.

Table 1 Examples of commercial solutions

Event Archiver	www.doriansoft.com/totalsolution/index.htm
Event Archiver Enterprise	www.eventarchiver.com/download.asp
Event Log Monitor	www.tntsoftware.com/Products/ELM/
EventReader	www.strongsoftware.net/eventrd/
EventReporter version 4.0	www.eventreporter.com/en/
GFI LANguard Security Event Monitor	www.gfi.com/lanselm/
LogCaster	www.rippletech.com/main.php
Microsoft Operations Manager	www.microsoft.com/mom/
NTLast	www.ntobjectives.com/ntlastv2.htm

Table 1 is not intended to be a complete listing of products of this type and they are simply listed alphabetically. It is outside the scope of this paper to review and critique these products. For a discussion of auditing and some of the third party software mentioned here see http://www.giac.org/practical/Steven_Toy.doc

These commercial tools all address this challenge with varying degrees of success but all cost significant amounts when you look at monitoring 2-20 servers and become very expensive if you were to consider monitoring workstations as well.

There are some tools available from Microsoft that can address some of the issues within the challenge but do not address the whole problem.

Dumpel.exe from Windows 2000 Resource Kit:

Dump Event Log is a command-line tool that dumps event logs from local/remote systems into tab-delimitated text files. This tool can also be used to filter certain events. The Windows Scheduler service could be used with batch files calling Dumpel.exe to try to automate the collection of event logs but there are several problems. The program cannot filter just portions of the event logs based on timestamps or event numbers forcing you to take the whole event log each time. To maintain this approach over time it usually becomes necessary to clear the event log after dumping it using Dumpel.exe so that you do not have redundant data on the next dump. This causes several issues, it means that the actual server no longer has a redundant copy of the events. As well, local/manual inspection of the event logs becomes troublesome. The wholesale nature of this process also means that each dump cycle must be run less frequently than would be ideal, perhaps on the order of hours or days instead of minutes or seconds.

Eventquery.pl from The Windows 2000 Resource Kit, Supplement One:
This Perl script displays events from the Event Viewer logs on local/remote computers running Windows 2000 and can also be used to filter for specific events.

EventCombMT from Microsoft Windows Server 2003 Resource Kit Tools
EventCombMT will parse event logs from multiple local/remote servers at the same time, and can also be used to filter for certain events.

None of these tools provide a complete solution to the challenge. However they could be utilized to address parts of the challenge if a custom scripting solution was pursued. Dumpel.exe could be used for the initial event collection from the servers and some custom method could be used to move the event data from the tab-delimitated text files into a central data store where the rest of the custom solution could work with the data. The functionality of Data Transformation Services in SQL Server 2000 might be able to handle this type of task.

Eventquery.pl could also be used for the initial event collection from the servers and then some custom Perl code could transfer this info to a central data store. Eventquery.pl requires ActiveState ActivePerl Build 521 be installed on the central computer pulling events from the remote servers.

EventCombMT is a very capable tool and includes a useful collection of intelligent filters right out of the box but can only generate text files as output and you would still need to move the results into a data store for further analysis. EventCombMT is also designed to simply "comb" through data residing on multiple remote computers and does not actually move the event log data to a central data store, it only creates a text file of its own output. This limitation would severely limit the use of EventCombMT for any type of audit or archival objectives.

There are also some similar components available from non-Microsoft sources

Dumpevt.exe Somarsoft (www.somarsoft.com/)
Command line utility to dump local or remote event logs to delimited text files suitable for import to database

ELDump.exe Jesper Lauritsen (www.ibt.ku.dk/jesper/ELDump/default.html)
Command line utility to dump local or remote event logs

Win32::EventLog Jesse Dougherty (search.cpan.org/search?dist=libwin32)
Perl module for processing of Windows event logs from Perl scripts

The Dumpevt.exe and ELDump.exe products allow more control over what is dumped into text files alleviating some of the drawbacks of Dumpel.exe (having to clear whole log to achieve an effective dump each time). Dumpevt.exe is a commercial product while ELDump.exe is free. The Win32::EventLog module provides similar functionality to Eventquery.pl. All three of these tools have the same general uses and limitations as the tools from Microsoft, they are only for the collection of event log data, and they do not directly transfer it to a central data store or provide a means for analysis. They are all however possible components to be used as part of a custom batch/script solution.

Another approach would be to modify the Windows servers to support the pushing of their event log data to a central Syslog server. Syslog is a widely used approach in the world of Unix and network devices to push all event log type of data to a centralized logging system. This approach is the most comprehensive available (as it could include non-Windows devices) and has many benefits and much merit. However, there are a couple of issues that made it less desirable for our network and will likely apply to many other networks as well. In my reading (*www.sans.org/rr/papers/index.php?id=713*) it was discovered that HP ProCurve Switches do not support Syslog, our network consists solely of a dozen HP ProCurve Switches. Our network is a “departmental” type of network (actually a faculty at a University) and we do not really have a security perimeter and certainly do not maintain our own, meaning we have no firewalls, routers or other networking equipment to gather Syslog data from. All these types of devices are managed by the IT group at the core of the University’s network. Our network is also a homogenous collection of a dozen Windows 2003 and Windows 2000 servers. My reading also suggested that while at least some of a Syslog on Windows solution could be done for free there seemed to be little available for free to do analysis and alerting. In fact after reading *www.sans.org/rr/papers/index.php?id=713* I discovered that even the commercial Syslog package used in the paper was unable to analyze log data from any device other than ones running its Syslog provider, meaning that while they had central log collection from all devices (HP excluded) they couldn’t analyze the data from all those network devices anyway, only the servers running the commercial Syslog provider. When I translated these shortcomings into our network I realized that the Syslog approach would simply be a cross-platform approach that would cost a significant amount of money and have no advantages over a dedicated Windows package for our network.

The approach chosen was to develop a custom solution that would copy event log data from all our Windows servers to a central MS SQL Server data store where it could be analyzed and used for alerting/reporting/auditing. In the case of our network and our IT group, I worked through our choice of technologies from the back-end to the front. We had existing expertise in SQL Server driven ASP web development using VBScript, so we had a high degree of comfort and confidence using SQL Server and VBScript to deliver the analysis/alerting/reporting/auditing aspects. The challenge for me would be how to get the data into SQL Server from all those servers. My experience at SANS 2003 in New Orleans (Track 5 Securing Windows) gave me a great exposure to the use of VBScript for administration of Windows machines and demonstrated the incredible power available with WMI. One of the scripts presented there

(WMI_Events_Consumer.vbs, Version: 1.1, Jason Fossen) demonstrated the crucial core functionality that I would need, how to get event log data loaded into VBScript where I knew I would be able to handle it appropriately. It also offered me the realization that I could make a solution that was nearly real-time and non-invasive/non-destructive to the existing Event logs. While the same basic functionality was available using Perl it was decided that VBScript had several advantages; I was comfortable with it, Microsoft promotes it as the preferred language for Windows administration, it interacts directly and gracefully with WMI and I preferred the approach where my collection code and any analysis/alerting code would be done all in the same programming language. Another benefit to using VBScript was that basically the same code could be used in an "on-demand" manner in ASP web pages or the command line with .vbs files and then the same code could also be used for "scheduled" processes using .vbs files and the Windows Scheduler.

I also soon discovered that having the option to run VBScript event consumers on each of our servers would give me a chance to implement a push style approach to getting all the data into our central SQL data store. This push approach would make it much easier to integrate my solution into our hardened servers, in particular our Web Farm machines that supported no standard types of remote management. The push approach had some additional merit in my opinion as it would mean that our production SQL Server would still only be doing SQL work and not running lots of VBScript components to pull data into SQL. A push approach also means the event log data only travels across the network once, not once from the source to a monitoring station and once more from there to the data store. The workload of collecting event data and submitting it to a central data store would also be distributed to each of the servers in our network more evenly with a push approach. We would also have greater resiliency with this approach as the code would be running an independent copy on each server so that the only single point of failure would be our big clustered SQL Server production system that we were very comfortable relying on (three years of >99.9% uptime).

The last factor looked at, although perhaps the most pivotal, was cost; all the tools required are included with Windows or available for free downloads. The only additional component required that would cost money would be SQL Server 2000 and I felt comfortable that the availability of MSDE and MySQL would mean that I could develop a custom solution using free tools that just about any Windows administrator should be able to plug into their existing Windows network and take advantage of without spending any money. This lack of cost would appropriately address one of the factors (high cost of "Enterprise Solutions" like Microsoft Operations Manager) previously discussed that leads many smaller Windows networks to operate without the appropriate degree of log monitoring and analysis to properly secure their infrastructures.

To address the portability of the solution to any R.D.M.S. I decided to attempt to write the solution using no MS SQL Server specific features if possible. This approach would also help in the feasibility of analysis logic (mostly SQL) being sharable and portable from one network to another. To also help this issue it was decided to try to leave the data as close to its original format as possible in the data store so that the data structure would hopefully remain identical in everyone's implementation of the solution. I felt that any optimizations I would make for my network would likely be wrong for someone else's network and they would likely start changing data structures leading to a loss of the complete portability.

To summarize my approach was chosen based on the following criteria;

- Cost of the entire solution
- Technologies used
- Minimal impact on existing infrastructure and practices
- Ability to integrate into varied and complex security configurations
- Ability to have an open system that would support sharing of work
- Support a secured or hardened configuration

1.3 Solve a Windows Security Challenge

My chosen solution was based on using VBScript/WMI to move event data into a central data store where it could be analyzed/audited/alerted using VBScript. This approach had a software cost of zero; VBScript and WMI are free as part of Windows. While I would use SQL Server 2000 (we already had this licensed and running), I decided to make sure the solution would be portable to free database packages like MSDE and MySQL so this would be a zero software cost approach for everyone. To further my goals of making this a free solution I decided to make sure the architecture was optimized for the efficient collection of the event data so this new workload would have a minimal effect (no new servers required) on an existing infrastructure. The analysis, reporting, auditing, and alerting on functions would all be handled by SQL and VBScript so that automatic processes could be controlled using Windows Scheduler (free), any web interface would use IIS (free), and the command line functionality would of course, also be free.

The technologies chosen (VBScript, WMI, SQL and built-in Windows components like IIS and Scheduler) were all chosen not just because they are free but because they are commonly used and well supported technologies that I was familiar with and knew that I could find support for and would leave my solution accessible to thousands of other Windows Administrators who are also familiar with the same products.

When I thought about all the different Windows networks out there that might consider using my solution, I realized that there would be a large range of sizes, capacities, security and level of spare server horsepower available for new projects. Realizing that if I wanted to provide a free solution that would be embraced by many networks not currently running something like it, I had to make it scalable, easy to install/integrate, and as efficient as possible. By choosing SQL Server as the backend data store (or MySQL), I knew I was using the right architecture to allow the analysis/reporting/auditing functionality as this type of work on a large collection of text file based data would be very inefficient. The choice also meant that I should be able to support the collection of large amounts of log data as SQL Server and MySQL regularly support databases larger than 1 TB in size and are capable of bulk load rates of over 100,000 rows per second (*http://www.microsoft.com/sql/techinfo/administration/2000/scalabilityfaq.asp* and *http://www.mysql.com/*).

Choosing VBScript/WMI/Scheduler helped me to achieve my goal of making my solution easy to install/integrate. The changes required on each server were very minor; simply place a .vbs file on a network share, create a scheduled task pointed at that file and configured to run on system startup, and create a DSN pointing at the central data store. This setup was required on each server that I wanted to push events from. An alternative approach was to have a single machine collect events from multiple machines and submit them to the central data store. My testing showed that on machines with very high rates of event logging there was a noticeable load generated by the VBScript code that was collecting events and sending them to the data store. By using machines that were not heavily loaded to collect events from those that were heavily loaded I could significantly decrease the incurred load.

The ability to support both push and pull models not only helped the solution integrate easily, it also helped to support unusual configurations including our hardened load-balanced Web Farm (runs e-commerce apps) that would not respond to a push approach for a number of reasons. Since we did not want to make any significant changes to this audited setup, having the choice of installing code onto the machine and pushing data out was critical. Having these two choices was enough for me to get my solution working with all my servers (with minimal changes) but I could foresee that some setups (such as a fire-walled data store machine and fire-walled source servers) would likely require a user to write firewall rules to allow the data to flow.

Several changes were required for the security of the process itself. I created a new Domain Administrator account specifically for this task with a long and complex password, shared it with no one (we keep a sealed copy in a safe) and used it only for these scheduled tasks. In our network only Domain Administrator accounts have the logon as batch job and logon as a service rights as Admin rights are also need to audit security logs. Finally I created a unique login for that account in SQL Server and assigned it datawriter privileges in that database and denied access to all other databases. By using built-in Windows Authentication for access to the SQL server I had no need to store username and password information in a connection string in my .vbs scripts (see *http://www.sans.org/rr/papers/index.php?id=1371* for background) where it could be easily compromised on any of the remote servers. By using SQL Server and a System Data Source Name (ODBC link) to connect my VBScript to my data store I was able to only store a chosen name for the DSN in my script, there was no server name and no username and password (see script appendix). This way if someone did gain access to the script file they could only inspect the process, not gain any valuable account or server location information. Just in case someone did get the username and password, that account only had rights to input data into the central data store, they could not read any data, so they could not further their penetration by sifting through event logs for useful information like accounts with elevated privileges. This approach seemed to generate the least amount of risk and offered multiple layers of security with good containment in place.

1.4 Scripting / Automation of the Solution

To see the actual solution in production I will present several scripts showing examples of what is running at each stage of the solution. The first script will be the VBScript that loads the event log data and sends it to the database. I will then show the SQL script to generate **tblInBound** the database table that stores all the events. Now that we have data loaded into the database I will show an example script (**mail-errors.vbs**) that uses this data store and sends email to an Administrator when ever a error occurs.

Code Block 1 EventSink.vbs

What follows is the actual VBScript code (**EventSink.vbs**) that runs on each machine doing the uploading of event data to the data store. A discussion of what is occurring will follow each block of data. I will attach the complete and un-interrupted code at the end of the paper.

```
OPTION Explicit
DIM strEventSinkDBConn, strComputers, objEventSinkDB, strDBQuery, objWMIconn, objSink
DIM intRecordNumber, strLogFile, intEventIdentifier, intEventCode, strSourceName
DIM strType, strCategory, strCategoryType, strUser, strComputerName, strMessage
DIM dtTimeGenerated, dtTimeWritten
```

This block simply sets the VBScript Option to Option Explicit forcing me to declare all variables before using them and then DIMs all the variables used in the script.

```
strEventSinkDBConn="DSN=EventSink"   'Set the connection string to the name of the DSN
                                     'IP address or computer name, comma separated
strComputers=Array("127.0.0.1")      'names within quotes eg ("machine1","127.0.0.1")
```

The above block first defines the connection string used to connect to the data store. Notice the use of a system DSN (called EventSink) and Windows Authentication to connect to the SQL Server so that the server name and no account credentials appear in the connection string. The variable strComputers is of the subtype variant and has an array assigned to it using the Array() function. This is a rather odd approach but VBScript allows you to assign an array to a variant data type and treat it like an array without going through the effort of determining the dimensions of the array. The real advantage though is the use of the Array() function which automatically assigns comma separated values to array elements. When this script is modified to monitor more than one machine you enter a comma separated list of machine names on this line with each machine name enclosed in quotes.

strComputers=Array("127.0.0.1","DC1", “Server2”, “192.168.1.1”)

It doesn’t matter if there are spaces after the comma; a TRIM function is used later to remove them.

```
Set objEventSinkDB=CreateObject("ADODB.Connection") 'Create an ADODB connection object
objEventSinkDB.Open strEventSinkDBConn              'Open that connection
CreateEventSink(strComputers)                       'Sets up EventSink
```

The three lines above accomplish two major tasks. First, they create an ADODB connection and then open that connection to the database specified in the connection string defined earlier. The third line calls the user subroutine CreateEventSink (discussed late this page) and sends the array of computers to be monitored along as a parameter.

```
Do While True                 'this loop is designed to run forever
        WScript.Sleep(5000)   'checking for events at interval
Loop                          'low impact delay between checks, in milliseconds
```

This three line loop with no exit is actually the main block of executing code, it simply goes to sleep for 5 seconds at a time and then wakes up long enough to handle any events. The use of the WScript.Sleep function allows for an efficient script that does not create a big draw on system resources while the script is doing no work. The only way to exit this script when run from the command line is to hit ‘control c’. When run as a scheduled task under another users credentials you must use the Task Manager to kill the process.

```
SUB CreateEventSink(strComputers)
   DIM intCounter, strComputerName
   On Error Resume Next
   FOR intcounter = 0 TO UBound(strComputers)
strComputerName = TRIM(strComputers(intCounter))
Set objWMIconn = GetObject("WinMgmts:{impersonationLevel=impersonate,”&_
                   “authenticationLevel=PktPrivacy, (security)}!\\"&strComputerName)
        If Err <> 0 Then
               WScript.Echo Err.Number & VbCr
               WScript.Echo Err.Description
               Err.Clear
               Exit Sub
        End If
        Set objSink = WScript.CreateObject("WbemScripting.SWbemSink","objSink_")
        objWMIconn.ExecNotificationQueryAsync objSink, "select * from “&_
           “__instancecreationevent where targetinstance isa 'Win32_NTLogEvent'"
        If Err <> 0 Then
               WScript.Echo Err.Number & VbCr
               WScript.Echo Err.Description
               Err.Clear
               Exit Sub
        End If
   NEXT
   On Error Goto 0
END SUB
```

Here is the previously called user subroutine CreateEventSink that is passed the array of computers to monitor. First, I DIM two local variables. Then I turn off standard error handling so that I can trap any errors in this area manually. Next I create a standard For Next loop that goes from 0 (all arrays in VBScript start at 0) to the upper boundary of the array determined by the UBound function, this will be 0 when the script is only monitoring one machine. This will result in the For Next loop only going through one iteration. I then TRIM any spaces left around the computer name in this array element and assign the name to the variable strComputerName. The following line is the WMI connection object and it carries the Security option that allows only appropriate users to access critical components like the Security Event Log. This statement also has the authentication parameter set to **PktPrivacy** and the impersonationLevel parameter set to **impersonate**. Please refer to Appendix B for a listing and explanation of these two settings.

These two settings were put in place to ensure that the data sent to the event sink came only from the proper WMI provider and that no-one could intercept the data in transmission in an easily interpreted form. Microsoft's DCOM is using standard SSPI services to protect the RPC and it is using 40 or 128 bit encryption depending on which version of Windows you are running. Since I am running Windows 2003, I took it to mean that I was running 128 bit encryption and therefore just about anyone would have that level on encryption available to them if they had at minimum a fully patched Windows 2000 system. One of the reasons I used the PktPrivacy setting here is that we did not have IPSec encryption in place in our network, only IPSec packet signing.

I then have a little error trapping block that would simply write the error out on the screen (if it is running at the command prompt) and clear the error. I then create the Sink object and perform an asynchronous query for all the Windows Event Logs. This approach is slightly different and I believe more efficient than the approach demonstrated in **WMI_Events_Consumer.vbs** v1.1 by Jason Fossen. My approach utilizes an asynch call that essentially batches the work and goes to sleep in between batches. I believe this is a more efficient approach than used by Jason who handles the events in a continuous synchronous manner using the ExecNotificationQuery call instead of the ExecNotificationQueryAsync call I used. I then go into another error trapping section before turning back on normal error handling and exiting the For Next loop and then the subroutine.

```
SUB objSink_OnObjectReady(objWMIObject, objWMIAsyncContext)
        strComputerName =(objWMIObject.TargetInstance.ComputerName)
        strLogFile =(objWMIObject.TargetInstance.LogFile)
        IF LEN(strLogFile)=0 THEN strLogFile=""
        intRecordNumber =(objWMIObject.TargetInstance.RecordNumber)
        IF NOT ISNUMERIC(intRecordNumber) THEN intRecordNumber=0
        strSourceName =(objWMIObject.TargetInstance.SourceName)
        intEventIdentifier =(objWMIObject.TargetInstance.EventIdentifier)
        intEventCode =(objWMIObject.TargetInstance.EventCode)
        strType =(objWMIObject.TargetInstance.Type)
        strCategory =(objWMIObject.TargetInstance.Category)
        strCategoryType =(objWMIObject.TargetInstance.CategoryString)
        dtTimeGenerated =FixDateFormat(objWMIObject.TargetInstance.TimeGenerated)
        dtTimeWritten =FixDateFormat(objWMIObject.TargetInstance.TimeWritten)
        strUser =(objWMIObject.TargetInstance.User)
        strMessage =CleanString(objWMIObject.TargetInstance.Message)
        WriteToDB objEventSinkDB
END SUB
```

This subroutine is automatically called whenever the event sink has events and it simply assigns each of the values available from the event log to a variable. I do some checking for empty fields and such here but only on fields that have generated errors in my testing. I then call the user sub WriteToDB and pass along which connection object to use. All the data from the event is in variables accessible outside this sub so I do not need to pass those values directly.

```
SUB WriteToDB(ByRef objEventSinkDB)
     strDBQuery ="INSERT INTO tblInbound (EventDateTime, ComputerName, LogFile, “&_
       “RecordNumber, SourceName, EventIdentifier, EventCode, Type, " &_
        "Category, CategoryString, TimeGenerated, TimeWritten, [User], Message)”&_
       “ VALUES (GETDATE(), '"&strComputerName&"', '"&strLogFile&_
       "', "&intRecordNumber&", '"&strSourceName&"',"&intEventIdentifier&", "&_
        intEventCode &", '"&strType&"', '"&strCategory &"', '"&strCategoryType&_
       "', '"&dtTimeGenerated&"', '"&dtTimeWritten&"', '"&strUser&_
       "', '"&strMessage&"');"
     'WScript.Echo "Writing Event to DB at:" & NOW() & VbCr 'uncomment for debug
     On Error Resume Next
     objEventSinkDB.Execute(strDBQuery)
       If Err <> 0 Then
              WScript.Echo "Error# " & Err.Number & VbCr
              WScript.Echo "Error: " & Err.Description & VbCr
              WScript.Echo "SQL Query: " & strDBQuery
              Err.Clear
              Exit Sub
       End If
      On Error Goto 0
END SUB
```

The user sub WriteToDB while messy looking is very simple, it concatenates together a long string containing a complete SQL insert statement and it then executes the query. An example of the finished string would be;

```
“INSERT INTO tblInbound (EventDateTime, ComputerName, LogFile, RecordNumber, SourceName,
       EventIdentifier, EventCode, Type, Category, CategoryString, TimeGenerated,
       TimeWritten, [User], Message) VALUES (GETDATE(), 'ComputerName', 'System', 1729,
       'Service Control Manager',1073748860, 7036, 'information', '0', '', '5/1/2004
       12:31:41 AM', '5/1/2004 12:31:41 AM', '', 'The Alerter service entered the stopped
       state.');”.
```

The statement or Query is always the same, just the values of each field change with each event.

```
FUNCTION FixDateFormat(objEventDateFormat)          'converts date/time 4 VBScipt/SQL
       FixDateFormat = CDate(DateSerial(Left(objEventDateFormat,4),
          Mid(objEventDateFormat,5,2),
          Mid(objEventDateFormat,7,2)) +
          TimeSerial(Mid(objEventDateFormat,9,2),
          Mid(objEventDateFormat,11,2),
          Mid(objEventDateFormat,13,2)))
END FUNCTION

FUNCTION CleanString(strForCleaning)
       CleanString = REPLACE(strForCleaning,"'","`")       'swaps ` for '
END FUNCTION
```

The two user functions at the end here simply correct an incompatible data/time format and remove any single quotes from the event data as it would cause an error when submitting the query. This is all the script code involved in getting all your event log data to a central data store.

Code Block 2 create-tblInBound.sql

The SQL code used to create the table in the database that receives all the Windows Event Logs is listed below. Some of the field sizes are very generous and some fine-tuning may be possible to conserve storage space. The actual syntax here may have some SQL Server specific content but given this script it would be easy to clean up for another database engine.

```
BEGIN
CREATE TABLE [dbo].[tblInBound] (
        [id] [bigint] IDENTITY (1, 1) NOT NULL ,
        [EventDateTime] [datetime] NULL ,
        [ComputerName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [LogFile] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [RecordNumber] [bigint] NULL ,
        [SourceName] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [EventIdentifier] [bigint] NULL ,
        [EventCode] [bigint] NULL ,
        [Type] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Category] [bigint] NULL ,
        [CategoryString] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [TimeGenerated] [datetime] NULL ,
        [TimeWritten] [datetime] NULL ,
        [User] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Message] [varchar] (4000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
END
```

The following code is an example of a small .vbs script called **mail-errors.vbs** that is designed to parse all the event log data since it was last run and send an email to an Administrator if it finds any events marked as an Error. I use this as a scheduled task that runs once a minute (very small load generated by this job) and it checks to see if there have been any errors in the last minute. To setup my processes like this I followed the principles of least privilege and created another Domain Account and gave it read-only access to the table of event data in SQL Server 2000 since the job only needed to read data, not write any.

Code Block 3 mail-errors.vbs

```
OPTION EXPLICIT
DIM strMailServer, strMailTo, strMailFrom, strMailSubject, strMailBody
DIM strEventSinkDBConn, objEventSinkDB, rsCheck4Errors, intCheckInterval
```

Once again here I use the Explicit option to force all variable to be named and then I DIM all the variables.

```
intCheckInterval=1     'Interval is in minutes, same as scheduled task interval
strMailServer=  "smtp.yourdomain.com"                ‘set to your SMTP server
strMailTo=      “theAdmin@yourdomain.com”            ‘set to destination
strMailFrom=    “theCode@yourdomain.com”             ‘set to source
strMailSubject= "Test of Alert on Error"             'This is just here for testing
strMailBody=    "This is a test message body."       'This is just here for testing
```

The block above sets the time windows that the SQL query looks back at. It is in minutes and this number should match the interval you set in the scheduled task setup. This use of a sliding window is so that the query only looks at the most recent data and that you do not generate an alert for events more than once. There are more robust ways to address these challenges but they would all involve writing data to a file or changing data in the database and I was looking for the lowest possible impact here so I stuck with only doing a read operation. I then set a bunch of values for the mail server used; the destination of the email, and who the email should say it is from.

```
strEventSinkDBConn="DSN=EventSink"                   ‘connection string = system DSN
Set objEventSinkDB=CreateObject("ADODB.Connection")  'Create ADODB connection object
objEventSinkDB.Open strEventSinkDBConn               'Open that connection
```

These three lines are exactly the same as the **EventSink.vbs** file and they simply define a connection to the database and then open that connection up.

```
SET rsCheck4Errors=objEventSinkDB.Execute("SELECT * FROM tblInBound WHERE ”&_
        “EventDateTime >= DATEADD(n,"& intCheckInterval*-1 &",GETDATE()) AND ”&_
        “Type='Error';")
```

This line fills a recordset with the results of the SQL query. Note the use of DateAdd with the intCheckInterval*-1 to get SQL to restrict the results to entries from the last minute and having a Type of Error. The small line of SQL is really the smarts of the whole page and this is the part that I would envision being shared. The two following blocks of code simply create the email and then send it to the user. They know nothing about the query except that if they got data in the recordset to make and send the messages. These blocks could then be re-used over and over without modification. If someone else wrote an SQL query that return complete event log entries that we of interest, all I would have to do is paste it into this line that creates the recordset, make sure it is fed any parameters it needs and save the file under a new name. This would then add a completely new function to the solution. There is really no limit to the possible querys that could be constructed but I will only be discussing two examples of the many I am using in our complete solution.

```
While NOT rsCheck4Errors.EOF
        strMailSubject = "Error: " & rsCheck4Errors("ComputerName") & " " &_
           rsCheck4Errors("SourceName")
        strMailBody = rsCheck4Errors("id") &VbCrLf& rsCheck4Errors("EventDateTime")&_
           VbCrLf& rsCheck4Errors("ComputerName") &VbCrLf& rsCheck4Errors("LogFile")&_
           VbCrLf& rsCheck4Errors("RecordNumber") & VbCrLf& rsCheck4Errors("SourceName")&_
           VbCrLf& rsCheck4Errors("EventIdentifier") & VbCrLf&_
           rsCheck4Errors("EventCode") & VbCrLf& rsCheck4Errors("Type") & VbCrLf&_
           rsCheck4Errors("Category") & VbCrLf&  rsCheck4Errors("CategoryString")&_
           VbCrLf&  rsCheck4Errors("TimeGenerated") & VbCrLf&_
           rsCheck4Errors("TimeWritten") & VbCrLf& rsCheck4Errors("User") & VbCrLf&_
           rsCheck4Errors("Message") & VbCrLf
        funSendMail strMailServer, strMailTo, strMailFrom, strMailSubject, strMailBody &_
           rsCheck4Errors.MoveNext
WEND
```

The above block is simply a loop that will only fire if there are events in the recordset and it will loop for as many times as there are events. All that happens inside the loop is we concatenate a mail subject and mail body together, the body contains all the data from the event log entry. I then call the function that actually sends the mail, passing along the data it needs.

```
FUNCTION funSendMail(strMailServer,strMailTo,strMailFrom,strMailSubject,strMailBody)
       DIM objMail
       SET objMail = WScript.CreateObject("CDO.Message")
       objMail.From = strMailFrom
       objMail.To = strMailTo
       objMail.Subject = strMailSubject
       objMail.TextBody = strMailBody
       objMail.Configuration.Fields("http://schemas.microsoft.com/cdo/configuration/”&_
              “smtpserver") = strMailServer
       objMail.Configuration.Fields("http://schemas.microsoft.com/cdo/configuration/”&_
              “sendusing") = 2
       objMail.Configuration.Fields.Update
       objMail.Send
       SET objMail=Nothing
END FUNCTION
```

This last chunk of code simply sends an email to whomever it is told to, with whatever subject and body is passed into it.

The next program is called **display-all-comp-events.vbs** and it is designed to query the data store and return all events involving a particular computer for a given interval. This script has some additional power as it brings back events not just from the computer you specified but also events from other computers that contained the computer name in the event. This is an example where some simple scripting can accomplish something that would be nearly impossible to do manually using the Event Viewer. The Event Viewer will not let you search the actual message contents for a specific string such as computer name so the only way to find events involving computer B when you are looking at the event logs on computer A is to manually read every event. By centrally collecting the logs we are able to blur the distinctions between machines and view our systems as much more of an intergrated whole.

Code Block 4 display-all-comp-events.vbs

```
OPTION EXPLICIT
DIM strEventSinkDBConn, objEventSinkDB, rsCheck4Errors
DIM strComputerName, intCheckInterval

If WScript.Arguments.Count <> 2 Then
       WScript.Echo "Usage: display-all-comp-events.vbs <computername> <time window>"
       WScript.Quit
End If

strComputerName = TRIM(WScript.Arguments(0))      'Computername like DC1 or 192.168.0.0
intCheckInterval = CINT(WScript.Arguments(1))     'Interval is in minutes
```

The block of code above is pretty standard. I DIM my variables, perform a check that the user supplied the right number of arguments at the command line, and then assign those arguments to variables.

```
strEventSinkDBConn = "DSN=EventSink"                    'connection string
Set objEventSinkDB = CreateObject("ADODB.Connection")  'ADODB connection object
objEventSinkDB.Open strEventSinkDBConn                  'Open that connection

SET rsCheck4Errors = objEventSinkDB.Execute("SELECT * FROM tblInBound WHERE “&_
        ″(ComputerName = '"& strComputerName&"' OR Message like '%"&strComputerName&_
        "%') AND   EventDateTime >= DATEADD(n,"&     intCheckInterval*-1 &",GETDATE()) ;")
```

Once again we see the standard three lines to handle the connection to the database and then the actual query. I used the same name for the recordset here so that I could use either the email functions previously shown or command line output functions that I will show here. Notice that the query looks not just in the ComputerName column but also checks for occurrences of the computer name in the Message column of all messages. It then restricts the results to those that occurred within the interval.

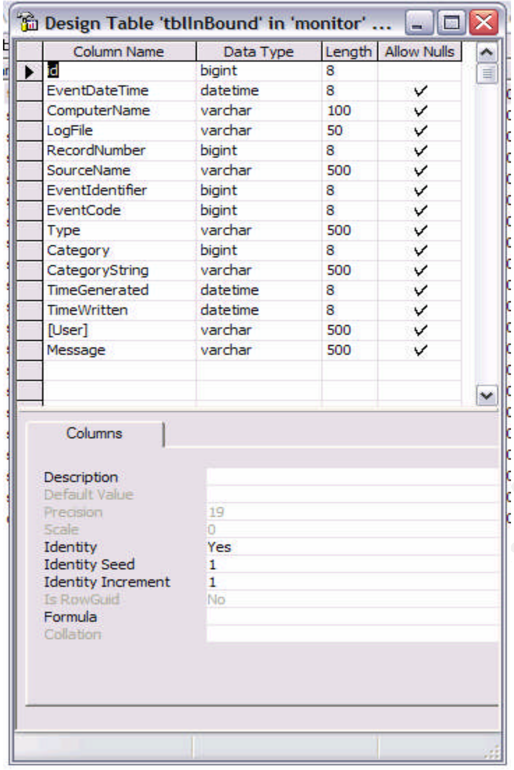
```
While NOT rsCheck4Errors.EOF
        funDisplayEvent()
        rsCheck4Errors.MoveNext
WEND


FUNCTION funDisplayEvent()
        WScript.Echo "#########################################################"&VbCr
        WScript.Echo "ID: " &rsCheck4Errors("id") &VbCr
        WScript.Echo "DB Time: " &rsCheck4Errors("EventDateTime") &VbCr
        WScript.Echo "ComputerName: " &rsCheck4Errors("ComputerName") &VbCr
        WScript.Echo "LogFile: " &rsCheck4Errors("LogFile") &VbCr
        WScript.Echo "RecordNumber: " &rsCheck4Errors("RecordNumber") &VbCr
        WScript.Echo "SourceName: " &rsCheck4Errors("SourceName") &VbCr
        WScript.Echo "EventIdentifier: " &rsCheck4Errors("EventIdentifier") &VbCr
        WScript.Echo "EventCode: " &rsCheck4Errors("EventCode") &VbCr
        WScript.Echo "Type: " &rsCheck4Errors("Type") &VbCr
        WScript.Echo "Category: " &rsCheck4Errors("Category") &VbCr
        WScript.Echo "CategoryString: " &rsCheck4Errors("CategoryString") &VbCr
        WScript.Echo "TimeGenerated: " &rsCheck4Errors("TimeGenerated") &VbCr
        WScript.Echo "TimeWritten: " &rsCheck4Errors("TimeWritten") &VbCr
        WScript.Echo "User: " &rsCheck4Errors("User") &VbCr
        WScript.Echo "Message: " &VbCrLf &rsCheck4Errors("Message") &VbCr
        WScript.Echo "##################################################"&VbCrLf&VbCrLf
END FUNCTION
```

Here we see the same type of While loop that will iterate through the recordset calling the display function each time. The function funDisplayEvent simply writes to the command line all the data contained with the event.

1.5 Validate Tool Functionality

The “installation” of the various components is very quick and quite painless. The first stage is to get the database ready by creating a fresh database and then the tblInBound using the supplied script. For quick reference and those that are used to working with a GUI in SQL Server here is the design table panel. Here it is very easy to see all the column names and datatypes.

Figure 1 Enterprise Manager view of tblInBound



Design Table 'tblInBound' in 'monitor' ...

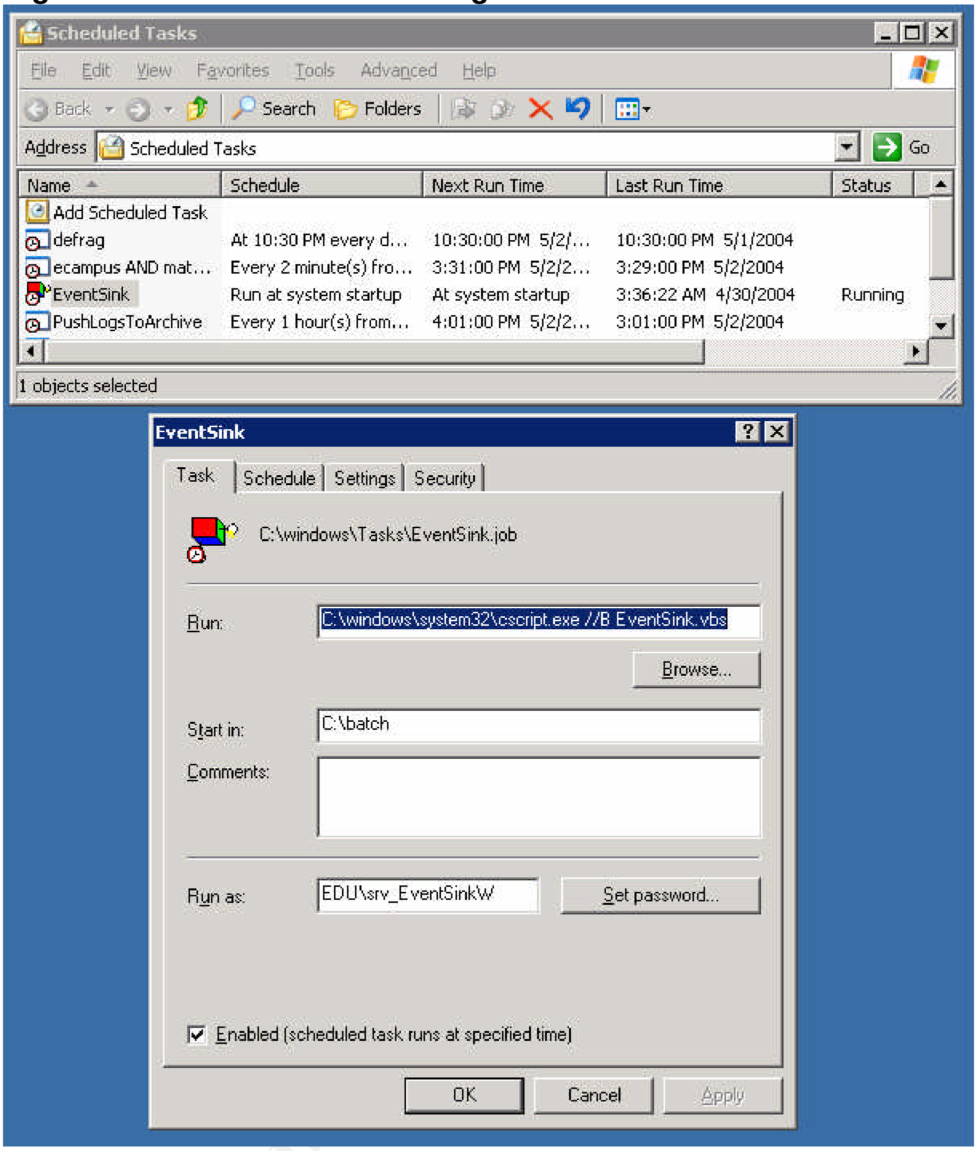
Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
EventDateTime	datetime	8	✓
ComputerName	varchar	100	✓
LogFile	varchar	50	✓
RecordNumber	bigint	8	✓
SourceName	varchar	500	✓
EventIdentifier	bigint	8	✓
EventCode	bigint	8	✓
Type	varchar	500	✓
Category	bigint	8	✓
CategoryString	varchar	500	✓
TimeGenerated	datetime	8	✓
TimeWritten	datetime	8	✓
[User]	varchar	500	✓
Message	varchar	500	✓

Columns

Description	
Default Value	
Precision	19
Scale	0
Identity	Yes
Identity Seed	1
Identity Increment	1
Is RowGuid	No
Formula	
Collation	

The next step is creating the required AD accounts, I created srv_EventSinkW for the processes that populate the database and srv_EventSinkR for any process that needs to read the database. Using Enterprise Manager I assigned appropriate access permissions within SQL Server so these accounts could do what they needed and nothing more. Next I needed to create a scheduled task to run **EventSink.vbs,** I configured this task to run on system startup and assigned the appropriate user account for the task to run as. It is necessary to make sure that these accounts have the “logon as a batch job” user account rights on all the required machines. I handled this using a Group Policy in Active Directory.

Figure 2 Scheduled Tasks configuration



Scheduled Tasks
File Edit View Favorites Tools Advanced Help
Back Search Folders
Address Scheduled Tasks
Go

Name	Schedule	Next Run Time	Last Run Time	Status
Add Scheduled Task				
defrag	At 10:30 PM every d...	10:30:00 PM 5/2/...	10:30:00 PM 5/1/2004	
ecampus AND mat...	Every 2 minute(s) fro...	3:31:00 PM 5/2/2...	3:29:00 PM 5/2/2004	
EventSink	Run at system startup	At system startup	3:36:22 AM 4/30/2004	Running
PushLogsToArchive	Every 1 hour(s) from...	4:01:00 PM 5/2/2...	3:01:00 PM 5/2/2004	

1 objects selected

EventSink

Task Schedule Settings Security

C:\windows\Tasks\EventSink.job

Run: C:\windows\system32\cscript.exe //B EventSink.vbs

Start in: C:\batch

Comments:

Run as: EDU\srv_EventSinkW

☑ Enabled (scheduled task runs at specified time)

This task should start manually if you right-click on the task and choose “run”. If this is successful you should now be collecting events in your database. When I needed to test, I would restart a non-essential service (Alerter) and look for the event entry to appear in the database. You can also start the same script from the command line and uncomment the line that echos a line to the screen each time it writes to the database.

An example of a single event log entry as captured would be:

Code Block 5 Sample of captured events

```
299     5/1/2004 12:31:42 AM   ComputerName   System  1729    Service Control Manager
        1073748860      7036    information     0       5/1/2004 12:31:41 AM 5/1/2004
        12:31:41 AM     "The Alerter service entered the stopped state.”
```

The first field (299) is an automatically created unique ID column created by SQL Server. The next field is also created by SQL Server and it is a timestamp generated on the SQL Server itself when the query was received. This is followed by the ComputerName, LogFile, RecordNumber, SourceName, EventIdentifier, EventCode, Type, Category, CategoryString, TimeGenerated, TimeWritten, User and finally the Message itself.

I then created another scheduled task for **mail-errors.vbs,** this time using the srv_EventSinkR account and configured to run once a minute for 24 hours a day. Whenever this process detects an error event in in its query it sends an email to the configured account that looks like this.

Code Block 6 Example alert email generated

```
From: "theAdmin" < theAdmin@yourdomain.com >
Date: Sun, 2 May 2004 15:57:01
To:< theAdmin@yourdomain.com >
Subject: Error: MachineWithError Service Control Manager
621215
5/2/2004 3:56:27 PM
MachineWithError
System
63477
Service Control Manager
-1073734790
7034
Error
0

5/2/2004 3:56:26 PM
5/2/2004 3:56:26 PM

The BrowserHawk BDF service terminated unexpectedly.  It has done this 1 time(s).
```

In production, in our network this script typically deliverers a less than two minute lag between the original error generation and successful notification of the on-call Administrator who carries a Blackberry PDA that will receive any email within about 30 seconds of us creating the SMTP message. This seems to have achieved the design goal of nearly real-time proactive notification of the Administrator.

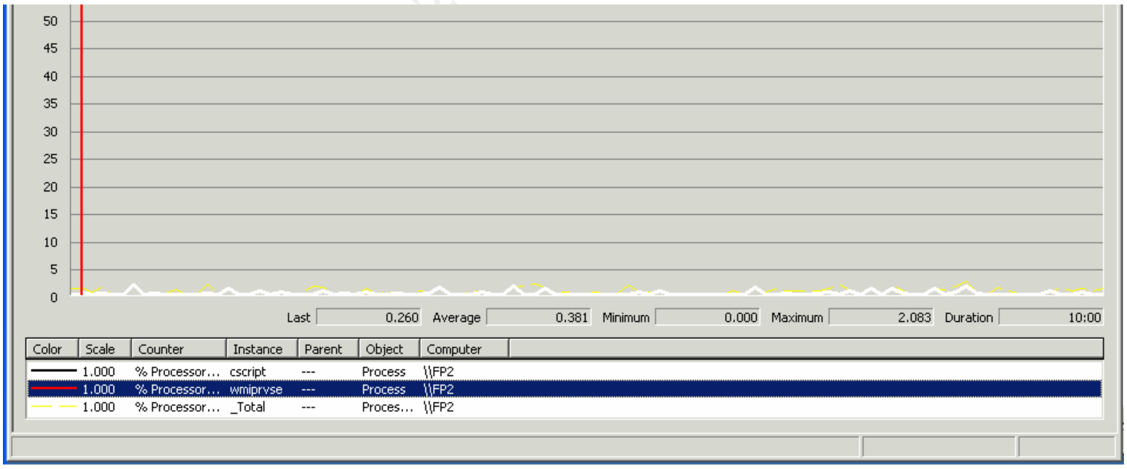
Here is an example of the output generated by the **display-all-comp-events.vbs** script at the command line, you would see one block like this for each event that matches the query.

Code Block 7 Example output from display-all-comp-events.vbs

```
##########################################################
ID: 298
DB Time: 5/1/2004 12:00:37 AM
ComputerName: TABLET-PC1
LogFile: Application
RecordNumber: 647
SourceName: MSSQLSERVER
EventIdentifier: 1073759001
EventCode: 17177
Type: information
Category: 2
CategoryString: Server
TimeGenerated: 5/1/2004 12:00:37 AM
TimeWritten: 5/1/2004 12:00:37 AM
User:
Message:
This instance of SQL Server has been using a process id of 1892 since 4/29/2004
11:19:14 AM (local) 4/29/2004 3:19:14 PM (UTC).

##########################################################
```

The performance impact of this solution was a very important issue in our network as we have very few spare resources or processing power on our servers for this task. I checked the performance impact first with a remote pull configuration then with local push configuration and lastly with a combination of both running. I also briefly checked for the impact of the solution on our database server but soon realized that this load was overwhelmed by a huge factor by the current load on our system to the point where it was difficult to identify.

Figure 3 Performance with remote pull architecture

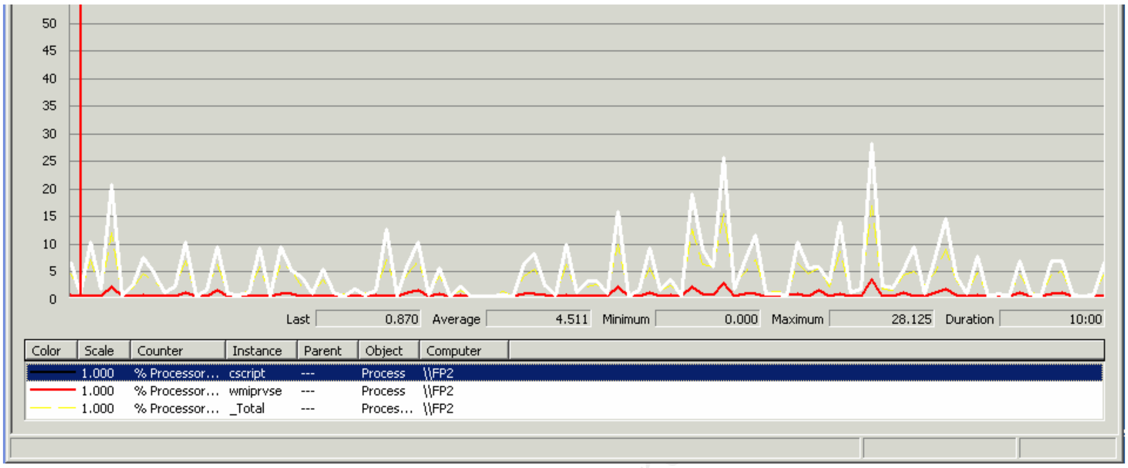


Last 0.260 Average 0.381 Minimum 0.000 Maximum 2.083 Duration 10:00

Color	Scale	Counter	Instance	Parent	Object	Computer
—	1.000	% Processor...	cscript	---	Process	\\FP2
—	1.000	% Processor...	wmiprvse	---	Process	\\FP2
— —	1.000	% Processor...	_Total	---	Proces...	\\FP2

In Figure 1 examine the summary numbers for the highlighted (white line) process (wmiprvse) from a Performance Monitor trace, this process averaged 0.381% of CPU time in the 10 minute trace period and generated 2283 events. This configuration is a remote server (trace shown) having events harvested remotely by another machine. In this setup the traced machine doesn't really have any VBScript running, the application is calling on the WMI provider service directly.

Figure 4 Performance with local push architecture

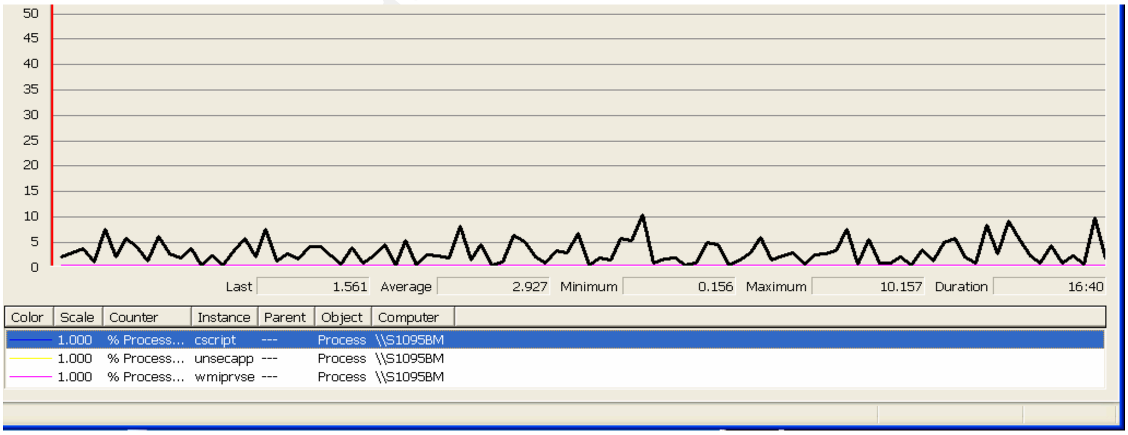


Last 0.870 Average 4.511 Minimum 0.000 Maximum 28.125 Duration 10:00

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	% Processor...	cscript	---	Process	\\FP2
▬	1.000	% Processor...	wmiprvse	---	Process	\\FP2
- -	1.000	% Processor...	_Total	---	Proces...	\\FP2

In Figure 2 examine the highlighted (white line) process (cscript) from a Performance Monitor trace, this process averaged 4.511% of CPU time in the 10 minute trace period and generated 2357 events. This push approach also generated peaks of up to 28% CPU time while the pull approach from Figure 1 produced peaks of only 2% CPU time. This configuration is the original approach and it involves a machine harvesting its own event log data and submitting it directly to the database server.

Figure 5 Single machine doing pull and push



Last 1.561 Average 2.927 Minimum 0.156 Maximum 10.157 Duration 16:40

Color	Scale	Counter	Instance	Parent	Object	Computer
▬	1.000	% Process...	cscript	---	Process	\\S1095BM
▬	1.000	% Process...	unsecapp	---	Process	\\S1095BM
▬	1.000	% Process...	wmiprvse	---	Process	\\S1095BM

In Figure 3 we see a slightly different setup, this machine is monitoring itself and a remote machine (remote machine generated 99% of events) and submitting both to a remote data store. The time period traced here was based on a 10 second sample giving a period of 16:40 and that period contained 5817 events. The average load generated by the script process was 3% with a peak of

10% so you can see that this approach of pulling events from the busiest machines seems to be the most efficient and effective.

Based on these results we have implemented the solution using a mix of approaches; machines with a very high volume of event generation and high loads are being collected remotely by a designated monitoring station while the rest of our infrastructure is submitting their own events to the central database.

Conclusions

The goal of this project was to create a tool that met as many of the following objectives as possible;

- Free
- Centralized collection of as many Event Logs as possible
- Simple to install
- Easily integrated into any infrastructure
- Minimal processing power required
- Easy to secure the tool
- Worked in real-time or near real-time
- Proactively attempt to contact the Administrator
- Facilitated easy sharing of work with peers
- Able to analyse data across all computers and logs
- Intelligent and flexible analysis
- Utilized common technologies

The solution implemented was free for myself and anyone without SQL Server 2000 can easily replace it with a free database server such as MySQL or MSDE. I was able to collect from all the Event Logs that appear in Event Viewer, including the Security, Application, System, DNS Server, Active Directory, and File Replication Service. With further work more logs could be integrated but this will be somewhat challenging to make other logs conform to the existing data schema. The install process certainly was simple with only a couple of steps and no true install program, just some simple configuration steps like creating a scheduled task. If the machine was having its events collected remotely there was no direct contact with the machine at all. The solution fit into my infrastructure by utilizing multiple approaches; push, pull, and a combination of both. If an organization makes heavy use of host firewalls there could be some problems but I feel they are relatively easily overcome. The combination of push and pull approaches seems to be able to deliver a solution with a minimal impact on server resources. Accumulating all these logs centrally though could tax your storage capacity on your database server over time and some type of flushing or archiving capability would need to be scripted. Securing the solution itself was straight-forward using the built-in mechanisms in DCOM and IPSec could also be added or used as an alternative. Securing the Active Directory accounts and database was also quite straight-forward for any experienced Administrator. The rate of log collection through to generating alerts received by an Administrator in such a short time period (less than 2 minutes) exceeded my expectations. Using SMTP mail to try to proactively contact the Administrator was a perfect solution for our organization but some alternate methods might need to be added for others. The exchanging of SQL queries that return recordset of interest seems like it will be reasonably effective at facilitating sharing of work between fellow Administrators. The single table data structure makes it transparent to analyse data across all logs and computers. The intelligence and flexibility of the analysis will only be as good as the queries the Administrator can write or obtain from others. I believe the choice of VBScript/WMI/SQL was appropriate and that many others share experience with these common and free tools. I believe the project as a whole was a great success but will need ongoing effort to develop more queries to fully realize the potential of this approach.

References

Dessiatnikov, Dmitry. Securing SQL Connection String. URL: http://www.sans.org/rr/papers/index.php?id=1371 (8/1/2004).

Lavy, Matthew. Meggitt, Ashley. Windows Management Instrumentation (WMI). New Riders. ISBN: 1578702607. URL: http://www.scribblin.gs/computing/wmibook.html (17/10/2001)

Maples, Wayne. SysLog Servers for NT/2000/XP. URL: http://is-it-true.org/nt/nt2000/atips/atips105.shtml (21/11/2003)

Microsoft. Microsoft Solution for Securing Windows 2000 Server, Chapter 9 - Auditing and Intrusion Detection v1.4. URL: http://www.microsoft.com/technet/Security/prodtech/win2000/secwin2k/09detect.mspx (5/8/2003).

Microsoft. Platform SDK, Windows Management Instrumentation. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_start_page.asp

Roger R. McLaren. Central Auditing of Windows NT Using Windows Script Host. URL: http://www.giac.org/practical/Roger_McLaren_GCNT.zip (3/30/2001).

Spencer, Ken. WMI 101: Event Logging. URL: http://www.winnetmag.com/Article/ArticleID/20581/20581.html (06/*2001*)

Appendix A Software Code Listings

EventSink.vbs

```
OPTION Explicit
DIM strEventSinkDBConn, strComputers, objEventSinkDB, strDBQuery, objWMIconn, objSink
DIM intRecordNumber, strLogFile, intEventIdentifier, intEventCode, strSourceName
DIM strType, strCategory, strCategoryType, strUser, strComputerName, strMessage
DIM dtTimeGenerated, dtTimeWritten

strEventSinkDBConn="DSN=EventSink"    'Set the connection string to the name of the DSN
                                      'IP address or computer name, comma separated
strComputers=Array("127.0.0.1")       'names within quotes eg ("machine1","127.0.0.1")
Set objEventSinkDB=CreateObject("ADODB.Connection") 'Create an ADODB connection object
objEventSinkDB.Open strEventSinkDBConn              'Open that connection
CreateEventSink(strComputers)                       'Sets up EventSink

'***************************************************************************************
Do While True                 'this loop is designed to run forever
        WScript.Sleep(5000)   'checking for events at interval
Loop                          'low impact delay between checks, in milliseconds
'***************************************************************************************

'***************************************************************************************
SUB CreateEventSink(strComputers)
   DIM intCounter, strComputerName
   On Error Resume Next
   FOR intcounter = 0 TO UBound(strComputers)
strComputerName = TRIM(strComputers(intCounter))
Set objWMIconn = GetObject("WinMgmts:{impersonationLevel=impersonate,”&_
                   “authenticationLevel=PktPrivacy, (security)}!\\"&strComputerName)
        If Err <> 0 Then
               WScript.Echo Err.Number & VbCr
               WScript.Echo Err.Description
               Err.Clear
               Exit Sub
        End If
        Set objSink = WScript.CreateObject("WbemScripting.SWbemSink","objSink_")
        objWMIconn.ExecNotificationQueryAsync objSink, "select * from “&_
           “__instancecreationevent where targetinstance isa 'Win32_NTLogEvent'"
        If Err <> 0 Then
               WScript.Echo Err.Number & VbCr
               WScript.Echo Err.Description
               Err.Clear
               Exit Sub
        End If
   NEXT
   On Error Goto 0
END SUB
'***************************************************************************************

'***************************************************************************************
SUB objSink_OnObjectReady(objWMIObject, objWMIAsyncContext)
        strComputerName =(objWMIObject.TargetInstance.ComputerName)
        strLogFile =(objWMIObject.TargetInstance.LogFile)
        IF LEN(strLogFile)=0 THEN strLogFile=""
        intRecordNumber =(objWMIObject.TargetInstance.RecordNumber)
        IF NOT ISNUMERIC(intRecordNumber) THEN intRecordNumber=0
        strSourceName =(objWMIObject.TargetInstance.SourceName)
        intEventIdentifier =(objWMIObject.TargetInstance.EventIdentifier)
        intEventCode =(objWMIObject.TargetInstance.EventCode)
        strType =(objWMIObject.TargetInstance.Type)
        strCategory =(objWMIObject.TargetInstance.Category)
        strCategoryType =(objWMIObject.TargetInstance.CategoryString)
        dtTimeGenerated =FixDateFormat(objWMIObject.TargetInstance.TimeGenerated)
        dtTimeWritten =FixDateFormat(objWMIObject.TargetInstance.TimeWritten)
        strUser =(objWMIObject.TargetInstance.User)
        strMessage =CleanString(objWMIObject.TargetInstance.Message)
        WriteToDB objEventSinkDB
END SUB
'*****************************************************************************************

'*****************************************************************************************
SUB WriteToDB(ByRef objEventSinkDB)
     strDBQuery ="INSERT INTO tblInbound (EventDateTime, ComputerName, LogFile, “&_
       “RecordNumber, SourceName, EventIdentifier, EventCode, Type, " &_
        "Category, CategoryString, TimeGenerated, TimeWritten, [User], Message)”&_
       “ VALUES (GETDATE(), '"&strComputerName&"', '"&strLogFile&_
       "', "&intRecordNumber&", '"&strSourceName&"',"&intEventIdentifier&", "&_
        intEventCode &", '"&strType&"', '"&strCategory &"', '"&strCategoryType&_
       "', '"&dtTimeGenerated&"', '"&dtTimeWritten&"', '"&strUser&_
       "', '"&strMessage&"');"
     'WScript.Echo "Writing Event to DB at:" & NOW() & VbCr 'uncomment for debug
     On Error Resume Next
     objEventSinkDB.Execute(strDBQuery)
        If Err <> 0 Then
                WScript.Echo "Error# " & Err.Number & VbCr
                WScript.Echo "Error: " & Err.Description & VbCr
                WScript.Echo "SQL Query: " & strDBQuery
                Err.Clear
                Exit Sub
        End If
      On Error Goto 0
END SUB
'*****************************************************************************************
```

create-tblInBound.sql

```
BEGIN
CREATE TABLE [dbo].[tblInBound] (
        [id] [bigint] IDENTITY (1, 1) NOT NULL ,
        [EventDateTime] [datetime] NULL ,
        [ComputerName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [LogFile] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [RecordNumber] [bigint] NULL ,
        [SourceName] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [EventIdentifier] [bigint] NULL ,
        [EventCode] [bigint] NULL ,
        [Type] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Category] [bigint] NULL ,
        [CategoryString] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [TimeGenerated] [datetime] NULL ,
        [TimeWritten] [datetime] NULL ,
        [User] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Message] [varchar] (4000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
END
```

mail-errors.vbs

```
OPTION EXPLICIT
DIM strMailServer, strMailTo, strMailFrom, strMailSubject, strMailBody
DIM strEventSinkDBConn, objEventSinkDB, rsCheck4Errors, intCheckInterval
intCheckInterval=1     'Interval is in minutes, same as scheduled task interval
strMailServer=  "smtp.yourdomain.com"               ‘set to your SMTP server
strMailTo=      “theAdmin@yourdomain.com”            ‘set to destination
strMailFrom=    “theCode@yourdomain.com”             ‘set to source
strMailSubject= "Test of Alert on Error"            'This is just here for testing
```

```
strMailBody=    "This is a test message body."       'This is just here for testing
strEventSinkDBConn="DSN=EventSink"                   ‘connection string = system DSN
Set objEventSinkDB=CreateObject("ADODB.Connection")  'Create ADODB connection object
objEventSinkDB.Open strEventSinkDBConn               'Open that connection

SET rsCheck4Errors=objEventSinkDB.Execute("SELECT * FROM tblInBound WHERE ”&_
        “EventDateTime >= DATEADD(n,"& intCheckInterval*-1 &",GETDATE()) AND ”&_
        “Type='Error';")
While NOT rsCheck4Errors.EOF
        strMailSubject = "Error: " & rsCheck4Errors("ComputerName") & " " &_
           rsCheck4Errors("SourceName")
        strMailBody = rsCheck4Errors("id") &VbCrLf& rsCheck4Errors("EventDateTime")&_
           VbCrLf& rsCheck4Errors("ComputerName") &VbCrLf& rsCheck4Errors("LogFile")&_
           VbCrLf& rsCheck4Errors("RecordNumber") & VbCrLf& rsCheck4Errors("SourceName")&_
           VbCrLf& rsCheck4Errors("EventIdentifier") & VbCrLf&_
           rsCheck4Errors("EventCode") & VbCrLf& rsCheck4Errors("Type") & VbCrLf&_
           rsCheck4Errors("Category") & VbCrLf&  rsCheck4Errors("CategoryString")&_
           VbCrLf&  rsCheck4Errors("TimeGenerated") & VbCrLf&_
           rsCheck4Errors("TimeWritten") & VbCrLf& rsCheck4Errors("User") & VbCrLf&_
           rsCheck4Errors("Message") & VbCrLf
        funSendMail strMailServer, strMailTo, strMailFrom, strMailSubject, strMailBody &_
           rsCheck4Errors.MoveNext
WEND
FUNCTION funSendMail(strMailServer,strMailTo,strMailFrom,strMailSubject,strMailBody)
        DIM objMail
        SET objMail = WScript.CreateObject("CDO.Message")
        objMail.From = strMailFrom
        objMail.To = strMailTo
        objMail.Subject = strMailSubject
        objMail.TextBody = strMailBody
        objMail.Configuration.Fields("http://schemas.microsoft.com/cdo/configuration/”&_
                “smtpserver") = strMailServer
        objMail.Configuration.Fields("http://schemas.microsoft.com/cdo/configuration/”&_
                “sendusing") = 2
        objMail.Configuration.Fields.Update
        objMail.Send
        SET objMail=Nothing
END FUNCTION
```

display-all-comp-events.vbs

```
OPTION EXPLICIT
DIM strEventSinkDBConn, objEventSinkDB, rsCheck4Errors
DIM strComputerName, intCheckInterval

If WScript.Arguments.Count <> 2 Then
        WScript.Echo "Usage: display-all-comp-events.vbs <computername> <time window>"
        WScript.Quit
End If

strComputerName = TRIM(WScript.Arguments(0))      'Computername like DC1 or 192.168.0.0
intCheckInterval = CINT(WScript.Arguments(1))     'Interval is in minutes
strEventSinkDBConn = "DSN=EventSink"                   'connection string
Set objEventSinkDB = CreateObject("ADODB.Connection")  'ADODB connection object
objEventSinkDB.Open strEventSinkDBConn                 'Open that connection

SET rsCheck4Errors = objEventSinkDB.Execute("SELECT * FROM tblInBound WHERE “&_
        ”(ComputerName = '"& strComputerName&"' OR Message like '%"&strComputerName&_
        "%') AND   EventDateTime >= DATEADD(n,"&     intCheckInterval*-1 &",GETDATE()) ;")
While NOT rsCheck4Errors.EOF
        funDisplayEvent()
        rsCheck4Errors.MoveNext
WEND


FUNCTION funDisplayEvent()
```

```
        WScript.Echo "#########################################################"&VbCr
        WScript.Echo "ID: " &rsCheck4Errors("id") &VbCr
        WScript.Echo "DB Time: " &rsCheck4Errors("EventDateTime") &VbCr
        WScript.Echo "ComputerName: " &rsCheck4Errors("ComputerName") &VbCr
        WScript.Echo "LogFile: " &rsCheck4Errors("LogFile") &VbCr
        WScript.Echo "RecordNumber: " &rsCheck4Errors("RecordNumber") &VbCr
        WScript.Echo "SourceName: " &rsCheck4Errors("SourceName") &VbCr
        WScript.Echo "EventIdentifier: " &rsCheck4Errors("EventIdentifier") &VbCr
        WScript.Echo "EventCode: " &rsCheck4Errors("EventCode") &VbCr
        WScript.Echo "Type: " &rsCheck4Errors("Type") &VbCr
        WScript.Echo "Category: " &rsCheck4Errors("Category") &VbCr
        WScript.Echo "CategoryString: " &rsCheck4Errors("CategoryString") &VbCr
        WScript.Echo "TimeGenerated: " &rsCheck4Errors("TimeGenerated") &VbCr
        WScript.Echo "TimeWritten: " &rsCheck4Errors("TimeWritten") &VbCr
        WScript.Echo "User: " &rsCheck4Errors("User") &VbCr
        WScript.Echo "Message: " &VbCrLf &rsCheck4Errors("Message") &VbCr
        WScript.Echo "##################################################"&VbCrLf&VbCrLf
END FUNCTION
```

Appendix B WMI connection setttings

Authentication Settings for WMI connection

From: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/setting_client_application_process_security.asp

Moniker name	Description
Default	WMI uses the default Windows Authentication setting.
None	Uses no authentication.
Connect	Authenticates the credentials of the client only when the client establishes a relationship with the server.
Call	Authenticates only at the beginning of each call when the server receives the request.
Pkt	Authenticates that all data received is from the expected client.
PktIntegrity	Authenticates and verifies that none of the data transferred between client and server has been modified.
PktPrivacy	Authenticates all previous impersonation levels and encrypts the argument value of each remote procedure call.

Impersonation Setting for WMI connection

From: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/setting_client_application_process_security.asp

Moniker name	Description
Anonymous	Hides the credentials of the caller. Calls to WMI may fail with this impersonation level.
Identify	Allows objects to query the credentials of the caller. Calls to WMI may fail with this impersonation level.
Impersonate	Allows objects to use the credentials of the caller. This is the recommended impersonation level for Scripting API for WMI calls.
Delegate	**Windows 2000 and later:** Allows objects to permit other objects to use the credentials of the caller. This impersonation will work with Scripting API for WMI calls but may constitute an unnecessary security risk.

Appendix C Other Software Mentioned

Event Archiver; *http://www.doriansoft.com/totalsolution/index.htm*
Event Archiver Enterprise; *http://www.eventarchiver.com/download.asp*
Event Log Monitor; *http://www.tntsoftware.com/Products/ELM/*
EventReader; *http://www.strongsoftware.net/eventrd/*
EventReporter version 4.0; *http://www.eventreporter.com/en/*
GFI LANguard Security Event Monitor; *http://www.gfi.com/lanselm/*
LogCaster; *http://www.rippletech.com/main.php*
Microsoft Operations Manager; *http://www.microsoft.com/mom/*
NTLast; *http://www.ntobjectives.com/ntlastv2.htm*
Dumpel.exe; Windows 2000 Resource Kit
Eventquery.pl; Windows 2000 Resource Kit, Supplement One
EventCombMT; Windows Server 2003 Resource Kit Tools
Dumpevt.exe; Somarsoft; *http://www.somarsoft.com/*
ELDump.exe; Jesper Lauritsen; *http://www.ibt.ku.dk/jesper/ELDump/default.html*
Win32::EventLog; Jesse Dougherty; *http://search.cpan.org/search?dist=libwin32*
WMI_Events_Consumer.vbs, Version: 1.1, Jason Fossen
MySQL; *http://www.mysql.com/*
MSDE; *http://www.microsoft.com/sql/msde/*
SQL Server; *http://www.microsoft.com/sql/*

Appendix D Listing of event codes

This whole appendix comes from http://www.microsoft.com/technet/Security/prodtech/win2000/secwin2k/09detect.mspx and is included here as a reference for anyone trying to extract meaningful information from the event logs.

Table 9.1 Logon Events That Appear in the Security Event Log

Event ID	Description
528	A user successfully logged on to a computer.
529	The logon attempt was made with an unknown user name or a known user name with a bad password.
530	An attempt was made to log on with the user account outside of the allowed time.
531	A logon attempt was made using a disabled account.
532	A logon attempt was made using an expired account.
533	The user is not allowed to log on at this computer.
534	The user attempted to log on with a logon type that is not allowed, such as network, interactive, batch, service, or remote interactive.
535	The password for the specified account has expired.
536	The Net Logon service is not active.
537	The logon attempt failed for other reasons.
538	A user logged off.
539	The account was locked out at the time the logon attempt was made. This event can indicate that a password attack was launched unsuccessfully resulting in the account being locked out.
540	Successful Network Logon. This event indicates that a remote user has successfully connected from the network to a local resource on the server, generating a token for the network user.
682	A user has reconnected to a disconnected Terminal Services session. This event indicates that a previous Terminal Services session was connected to.
683	A user disconnected a Terminal Services session without logging off. This event is generated when a user is connected to a Terminal Services session over the network. It appears on the terminal server.

Table 9.2 Account Logon Events That Appear in the Event Log

Event ID	Description
672	An authentication service (AS) ticket was successfully issued and validated.
673	A ticket granting service (TGS) ticket was granted.
674	A security principal renewed an AS ticket or TGS ticket.
675	Pre-authentication failed.
676	Authentication Ticket Request failed.
677	A TGS ticket was not granted.
678	An account was successfully mapped to a domain account.
680	Identifies the account used for the successful logon attempt. This event also indicates the authentication package used to authenticate the account.
681	A domain account logon was attempted.
682	A user has reconnected to a disconnected Terminal Services session.
683	A user disconnected a Terminal Services session without logging off.

Table 9.3 Account Management Events That Appear in the Event Log

Event ID	Description
624	User Account Created
625	User Account Type Change
626	User Account Enabled
627	Password Change Attempted
628	User Account Password Set
629	User Account Disabled
630	User Account Deleted
631	Security Enabled Global Group Created
632	Security Enabled Global Group Member Added
633	Security Enabled Global Group Member Removed
634	Security Enabled Global Group Deleted
635	Security Disabled Local Group Created
636	Security Enabled Local Group Member Added
637	Security Enabled Local Group Member Removed
638	Security Enabled Local Group Deleted

Event ID	Description
639	Security Enabled Local Group Changed
641	Security Enabled Global Group Changed
642	User Account Changed
643	Domain Policy Changed
644	User Account Locked Out

Table 9.4 Object Access Events That Appear in the Event Log

Event ID	Description
560	Access was granted to an already existing object.
562	A handle to an object was closed.
563	An attempt was made to open an object with the intent to delete it. (This is used by file systems when the FILE_DELETE_ON_CLOSE flag is specified.)
564	A protected object was deleted.
565	Access was granted to an already existing object type.

Table 9.5 How to Perform Key Auditing Actions for Object Access Event 560

Auditing Action	How It Is Achieved
Find a specific file, folder or object	In the Event 560 details, search for the full path of the file or folder you wish to review actions for.
Determine actions by a specific user	Define a filter that identifies the specific user in a 560 event.
Determine actions performed at a specific computer	Define a filter that identifies the specific computer account where the task was performed in a 560 event.

Table 9.6 Privilege Use Events That Appear in the Event Log

Event ID	Description
576	Specified privileges were added to a user's access token. (This event is generated when the user logs on.)
577	A user attempted to perform a privileged system service operation.

Event ID	Description
578	Privileges were used on an already open handle to a protected object.

Here are examples of some of the event log entries that can exist when specific user rights are used:

- Act as part of the operating system. Look for Event ID 577 or 578 with the SeTcbPrivilege access privilege indicated. The user account that made use of the user right is identified in the event details. This event can indicate a user's attempt to elevate security privileges by acting as part of the operating system. For example, the GetAdmin attack, where a user attempts to add their account to the Administrators group uses this privilege. The only entries for this event should be for the System account, and any service accounts assigned this user right.
- Change the system time. Look for Event ID 577 or 578 with the SeSystemtimePrivilege access privilege indicated. The user account that used the user right is identified in the event details. This event can indicate a user's attempt to change the system time to hide the true time that an event takes place.
- Force shutdown from a remote system. Look for Event IDs 577 and 578 with user right SeRemoteShutdownPrivilege access privilege indicated. The specific security identifier (SID) the user right is assigned to and the user name of the security principal that assigned the right are included in the event details.
- Load and unload device drivers. Look for Event ID 577 or 578 with the SeLoadDriverPrivilege access privilege indicated. The user account that made use of this user right is identified in the event details. This event can indicate a user's attempt to load an unauthorized or Trojan horse (a type of malicious code) version of a device driver.
- Manage auditing and security log. Look for Event ID 577 or 578 with the SeSecurityPrivilege access privilege indicated. The user account that made use of this user right is identified in the event details. This event will occur both when the event log is cleared and when events for privilege use are written to the security log.
- Shut down the system. Look for Event ID 577 with the SeShutdownPrivilege access privilege indicated. The user account that made use of this user right is identified in the event details. This event will occur when an attempt to shut down the computer takes place.
- Take ownership of files or other objects. Look for Event ID 577 or 578 with the SeTakeOwnershipPrivilege access privilege indicated. The user account that used the user right is identified in the event details. This event can indicate that an attacker is attempting to bypass current security settings by taking ownership of an object.

Table 9.7 Process Tracking Events That Appear in the Event Log

Event ID	Description
592	A new process was created.
593	A process exited.
594	A handle to an object was duplicated.
595	Indirect access to an object was obtained.

Table 9.8 System Events That Appear in the Event Log

Event ID	Description
512	Windows is starting up.
513	Windows is shutting down.
514	An authentication package was loaded by the Local Security Authority.
515	A trusted logon process has registered with the Local Security Authority.
516	Internal resources allocated for the queuing of security event messages have been exhausted, leading to the loss of some security event messages.
517	The security log was cleared.
518	A notification package was loaded by the Security Accounts Manager.

You can use these event IDs to capture a number of security issues:

- Computer Shutdown/Restart. Event ID 513 shows each instance of when Windows was shut down. It is important to know when servers have been shut down or restarted. There are a number of legitimate reasons, such as a driver or application was installed requiring a restart, or the server was shut down or restarted for maintenance. However, an attacker may also force a restart of a server in order to gain access to the system during startup. All cases where the computer is shut down should be noted for comparison with the event log.

  Many attacks involve the restart of a computer. By investigating the event logs, you can determine when a server has been restarted, and whether the restart was a planned restart, or an unplanned restart. Event ID 513 shows Windows starting up, as will a series of other events which are automatically generated in the system log. These would include Event ID 6005, which indicates that the Event Log service has started.

  In addition to this entry, look for the existence of one of two different event log entries in the system log. If the previous shutdown was clean, such as when an administrator restarts the computer, then Event ID 6006, the Event Log service was stopped, is recorded in the system log. By examining the details of the entry, you can determine which user initiated the shutdown.

If the restart was due to an unexpected restart, an Event ID 6008, the previous system shutdown at <time>on <date> was unexpected is recorded in the system log. This can be indicative of a denial of service (DoS) that caused a shutdown of the computer. But remember, it also can be due to a power failure, or device driver failure as well.

If the restart was made because of a stop error that resulted in a blue screen, then Event ID 1001, with a source of Save Dump, is recorded in the system log. The actual stop error message can be reviewed in the event details.

Note: To include the recording of Event ID 1001 entries, the check box option Write an event to the system log must be selected to enable the recovery settings section of the System Control Panel applet.

- Modifying or Clearing of the Security Log. An attacker may try to modify the security logs, disable auditing during an attack, or clear the security log to prevent detection. If you notice large blocks of time with no entries in the security log, you should look for Event IDs 612 and 517 to determine which user modified the audit policy. All occurrences of Event ID 517 should be compared to a physical log indicating all times that the security log was cleared. An unauthorized clearing of the security log can be an attempt to hide events that existed in the previous security log. The name of the user that cleared the log is included in the event details.

Table 9.9 Policy Change Events That Appear in the Event Log

Event ID	Description
608	A user right was assigned.
609	A user right was removed.
610	A trust relationship with another domain was created.
611	A trust relationship with another domain was removed.
612	An audit policy was changed.
768	A collision was detected between a namespace element in one forest and a namespace element in another forest. (Occurs when a namespace element in one forest overlaps a namespace element in another forest.)

The two most important events to look for here are Event IDs 608 and 609. A number of attempted attacks may result in these events being recorded. The following examples will all generate Event ID 608 if the user right is assigned or 609 if it is removed. In each case the specific SID that the user right is assigned to, along with the user name of the security principal that assigned the right, is included in the event details:

- Act as part of the operating system. Look for Event IDs 608 and 609 with user right seTcbPrivilege in the event details.
- Add workstations to the domain. Look for the events with user right SeMachineAccountPrivilege in the event details.
- Back up files and directories. Look for the events with user right SeBackupPrivilege in the event details.
- Bypass traverse checking. Look for events with user right SeChangeNotifyPrivilege in the event details. This user right allows users to traverse a directory tree even if the user has no other permissions to access that directory.
- Change the system time. Look for events with user right SeSystemtimePrivilege in the event details. This user right allows a security principal to change the system time, potentially masking when an event takes place.
- Create permanent shared objects. Look for events with user right SeCreatePermanentPrivilege in the event details. The holder of this user right can create file and print shares.
- Debug Programs. Look for events with user right SeDebugPrivilege in the event details. A holder of this user right can attach to any process. This right is, by default, only assigned to administrators.
- Force shutdown from a remote system. Look for events with user right SeRemoteShutdownPrivilege in the event details.
- Increase scheduling priority. Look for events with user right SeIncreaseBasePriorityPrivilege in the event details. A user with this right can modify process priorities.
- Load and unload device drivers. Look for events with user right SeLoadDriverPrivilege in the event details. A user with this user right could load a Trojan horse version of a device driver.
- Manage auditing and security log. Look for events with user right SeSecurityPrivilege in the event details. A user with this user right can view and clear the security log.
- Replace a process level token. Look for events with user right SeAssignPrimaryTokenPrivilege in the event details. A user with this user right can change the default token associated with a started subprocess.
- Restore files and directories. Look for events with user right SeRestorePrivilege in the event details.
- Shut down the system. Look for events with user right SeShutdownPrivilege in the event details. A user with this user right could shut down the system to initialize the installation of a new device driver.
- Take ownership of files or other objects. Look for events with user right SeTakeOwnershipPrivilege in the event details. A user with this user right can access any object or file on an NTFS file system disk by taking ownership of the object or file.