Case Study: Automating Common InfoSec Auditing Tasks on a Windows 2000 Network

Clay Risenhoover

Case Study: Automating Common InfoSec Auditing Tasks on a Windows 2000 Network

Clay Risenhoover
GSEC: GIAC Security Essentials Certification
Version 1.4b
Option 2

Table of Contents

Abstract

Policies are only as good as the procedures used to implement them. When the procedures are too cumbersome or time-consuming, it is likely that policy compliance will suffer. Unrealistic procedures can lead to "implemented policies" that are weaker than the stated policies. Conversely, ensuring that procedures are easy to implement has the effect of making full policy compliance more likely.

In this case study, we will examine how automating information security audit procedures at a university had the effect of increasing security through increased policy compliance. We will discuss three stated policies, their associated procedures, and how poorly designed procedures led to weak "implemented policies." We will then discuss how the procedures were automated, and, finally, discuss the effects of the automation on the university's overall security stance.

Part 1 –Good Policies. Bad Procedures.

I am the director of the network operations department at a small state university. My department consists of a network administrator, a PC technician that we share with the help desk, and one or two part-time student employees. We manage 11 Windows 2000 servers, logon accounts for 500 faculty and staff and about 4,000 students. Like most IT professionals, we stay very busy, and have very little time to devote to "routine" matters, like audit policy compliance.

There are three particular daily audit policies that are very important to me, but that we did a pretty miserable job of complying with. These policies are listed below. The procedures associated with the policies were just too involved and time-consuming for us to get around to doing them every day. We had a *stated* policy that these audits would be conducted daily, but we ended up with an *implemented* policy that we would just do them whenever we could. This had the obvious effect of weakening our security stance significantly. In a university, where the network is necessarily pretty open (the default rule on our firewall is allow-all, and we turn off just the things that scare us), regular auditing is critical to overall network security.

The three policies and their associated procedures are:

Policy Number One

Stated Policy:
The network administrator will perform a daily audit of failed logon attempts on each Windows 2000 domain controller and the campus web server.

Procedures:
The network administrator will:
1. Use event viewer to connect to the Security event log on each domain controller (we have four domain controllers on two domains) and the campus web server.
2. Filter the events on each server to show events with Event ID of 529, described by Microsoft as “Logon Failure.” (Microsoft 299475, 2002)
3. For each server, make note of multiple failed logon attempts using the same username, or occurring from the same workstation, as these are potential signs of an attempt to brute-force a username and password.
4. Where there are a number of failed logon events in succession, research to see if these events were followed by a successful logon. This could indicate that a password was successfully guessed.
5. Compare the results from all servers and make notes of any trends.
6. Report any suspicious activity to the Director of Network Operations.

Implemented Policy:
Due to the amount of time consumed in performing these tasks, the implemented policy was that the network administrator would conduct this audit about once a month, obviously well below the required once per day.

Policy Number Two

Stated Policy:
The network administrator will perform a daily check of the amount of free space on each hard disk drive on each server. (The policy includes a list of eleven servers which must be monitored.)

Procedures:
The network administrator will:
1. Use Windows Explorer to connect to a special empty share set up on each disk drive for the purpose of checking available space.
2. Make note of the free space on the drive.
3. If the drive is below 15% free space, delete temporary files, clear or archive log files as possible to increase free space.
4. If the drive is below 10% free, report it to the Director of Network Operations for further analysis and consideration of a hardware upgrade.

Implemented Policy:
It was fairly simple to check the space on each drive, but there were at least 22 disk drives that must be checked on any given day – each of our servers had at least a boot disk and a data disk, some have more than that. The implemented policy was to "not worry about" checking the drives until we had some spare time, or until a server began to show signs of poor performance.

Policy Number Three

Stated Policy:
The network administrator will perform a daily check of each Windows 2000 server to determine whether new patches are available for the operating system or installed software packages. (The policy lists the eleven servers to be checked.)

Procedures:
The network administrator will:

1. Use HFNETCHK.EXE against each Windows 2000 server, using the command line "hfnetchk.exe –h serverName"
2. Make note of any patch whose status is listed as "Note" or "Patch not Found"
3. Research missing patches to see if they need to be applied.
4. Apply patches as needed.

Implemented Policy:
Because of the importance of server patches, especially in such an open computing environment, we did a much better job of complying with this one. We subscribed to email alerts from Microsoft that announced new patches and applied them as we could. We ran HFNETCHK at least once per week against every server, but this is still well short of the stated policy of running it against each server every day.

Observations/Conclusions
The stated policies were useful, and would provide a good foundation to our security efforts, but the implemented policies all left something to be desired. It was obvious that we needed to change our procedures in such a way that we would increase our ability to implement them.

I chose to use a combination of batch files, command-line utilities, and custom scripts written in PERL and VBSCRIPT to automate the procedures wherever possible. The scripts would be written to send daily email reports summarizing any important findings. We would then use the emails to determine when further research or action was warranted. The next section of the paper describes the scripts and programs used to accomplish the automation. An appendix is provided containing source code of each of the batch files and scripts used. Where third-party tools were used, the URLs of the associated web sites are given.

Part 2 –Keep the Policies. Fix the Procedures

The solution I chose to bridge the gap between our stated and implemented policies was to automate as many of the procedures as possible. In this section, we will explore the custom scripts and third-party programs that I used to accomplish the automation for each of the three policies discussed above. Filenames and program names are listed in **`this font.`**

The newly automated procedures for each of the three policies have a number of features in common:

- Each of these processes is run on a single Windows 2000 server, whose sole purpose is support and monitoring of the university network.
- BLAT, discussed under Policy Number One, is used any time an email is to be sent using an automated process.
- Scheduling of all tasks is accomplished using the Windows Task Scheduler utility.
- Each of the processes is run early in the morning, when network load is light. This also allows emailed reports to be waiting when the network administrator and I arrive at the school each morning.

Policy Number One – Audit Failed Logon Attempts

Remember that the first policy we discussed required the network administrator to audit failed logon events on various servers. The procedures called for this to be done manually, using the Microsoft Event Viewer software. I wrote two scripts and a batch file to automate the process and send the results to the network administrator and me via email. Sample output from each script is given in the Appendix, following the source code of the script.

The first component was a custom VBSCRIPT program called **`evtLog.vbs`** (Listing 1 in Appendix A), which would connect to each server whose name was given on the command line, extract the security events with event code 529, and write the "time written" and "message" fields from each of those events to the console. Because the script is run daily, it is configured to output only events that are less than 24 hours old. The VBSCRIPT programs required code to convert a local time to the time format used by the Windows Management Instrumentation (WMI) interface. I used functions written by Jeffery Hicks and published on CramSession.com to accomplish this conversion. (Hicks, 2002) The output of this program is captured to a file, called **`event.txt`** (Listing 2 in Appendix A), which was then processed by a custom PERL script that generated the final report text.

The PERL script, named **`event.pl`** (Listing 3 in Appendix A), reads **`event.txt`** and processes each entry to determine the username and the source workstation and domain name for each failed logon event. If the event is the first one seen for a particular workstation/username or domain/username pair, the date and time are also noted. Having a time stamp on the events aids the network administrator in conducting further research as it is needed. After reading every entry, the script generates a report detailing failed logons grouped by workstation/username pair, domain/username pair, and then giving counts for failed logons by source domain, source workstation, and by username. The output of this script is captured in a file called **`eventSummary.txt`**. (Listing 4 in Appendix A)

The contents of eventSummary.txt are sent via email to the network administrator and me using a third party utility called **`blat.exe`**. According to its author:

> Blat is a Public Domain Windows 95/NT console utility that sends the contents of a file in an e-mail message using the SMTP protocol. Blat is useful for creating scripts where mail has to be sent automatically (CGI, backups, etc.), or just as a quick way to send a file or message quickly from the command line. It will store relevant configuration details in the registry for ease of use. Optionally, blat can also attach multiple binary files to your message. (Charron, 2002)

The automation was completed by using a batch file called **`eventLog.bat`** (Listing 5 in Appendix A), containing the necessary commands to process the failed logons for one or more (up to nine) servers. For example, to check the events on two servers and generate a single report (useful for checking both domain controllers in a domain at once), the command line would be **`eventLog.bat server1 server2`**. Finally, a batch file called **`ServerEvents.bat`** (Listing 6 in Appendix A) was written to perform this process for the web server, both domain controllers in the student domain, and both domain controllers in the faculty/staff domain.

The batch file is run every day and the network administrator is now required to read the email report and research any trends of failed logons that seem unusual or disturbing.

Policy Number Two – Audit Free Space on Hard Disk Drives

The second policy required the network administrator to check the free space on each of the disk drives on various Windows 2000 servers. The process was simple, but there were a large number of drives to be checked. Automation for this task was accomplished using three VBSCRIPT programs, a PERL script with its associated configuration file, and a batch file that calls the PERL script, pipes its output into a text file and emails the file using BLAT.

The three VBSCRIPT programs, named **`freeSpace.vbs`**, **`totalSpace.vbs`** and **`pctSpace.vbs`** (listings 7-9 in Appendix A) are used to calculate the free space, total storage space, and percentage of space which is free on a disk drive, respectively. The drive to be checked is given on the command line as a physical path (e.g. C:) or a UNC path (e.g. \\servername\sharename).

The PERL script, named **`driveSpace.pl`** (Listing 10 in Appendix A), reads the description and path of each disk to be checked from a configuration file called **`driveSpecs.txt`**. It prints a short header, including the date run to the console. Then, for each drive, it calls the three VBSCRIPT programs to determine free space and overall size of the drive, and writes a one-line report to the console. The output of this script is captured in a file named **`drives.txt`** (Listing 11 in Appendix A), which is then sent out using BLAT.

The resulting email is reviewed daily, and drives are cleaned off or upgraded as needed. We were so happy with the results of the daily emails that I ported the VBSCRIPT code to an Active Server Page (asp) program that updates every five minutes. We now use this page as part of the “heads up display” in our office to keep constant tabs on the state of our drives.

I later modified the **`driveSpace.pl`** file to append to a comma-separated value (CSV) file, named **`space.csv`**, every time it is run. The CSV file contains the total and free space for each drive checked, and allows us to develop trend data on hard drive usage on each server.

Policy Number Three – Audit Server Patch Levels

The third policy we discussed required that the network administrator run HFNETCHK to check the patch level of each server. Researching HFNETCHK prior to automating the task revealed that HFNETCHK was deprecated, and its functionality was now exposed by the Microsoft Baseline Security Analyzer. According to Microsoft:

> The Hfnetchk tool is a command-line tool that administrators can use to centrally assess a computer or group of computers for the absence of security patches. As of the V1.1 release of the Microsoft Baseline Security Analyzer (MBSA), Hfnetchk is now exposed through the MBSA command line interface, **`mbsacli.exe /hf`**. (Microsoft 303215, 2003)

Consequently, I made the decision to replace HFNETCHK with the MBSA as I developed the automated procedures.

The MBSA uses an XML file named **`mssecure.xml`** downloaded from Microsoft's web site (Microsoft 305835, 2002), which contains information on current patches available for each of the supported products. By default, the file is downloaded automatically each time the program is run, but this behavior can be overridden at the command line using the "-x" flag (e.g. **`mbsacli /hf –x mssecure.xml`**) to force it to use a local XML file.

To save bandwidth, I use the command line utility **`WGET.exe`** (Syring, 2002) to download and save the XML file once, just prior to running the checks. **`WGET.exe`** is a port of a popular GNU utility to the Windows platform. It can be used to download web pages or files from a URL passed to it on the command line. I run the WGET command within a batch file named **`mssecure.bat`** (Listing 12 in Appendix A), which downloads a CAB file containing the XML file, and then extracts the XML from the CAB file.

After downloading the XML file containing current patch information, a PERL script named **`checkserver.pl`** (Listing 13 in Appendix A) is run, which for each server, runs the MBSA hot fix option, saves the results to a file, and sends the results to the network administrator and me in an email. Because the output for each server can be a bit long, the results for each server are sent in a separate message.

Part 3 – Good Procedures. Happier Days

Automating the bulk of the work required to implement these three policies has had a number of beneficial results:

- Our policy compliance is remarkably better than before the automation.
- Our security stance is greatly improved.
- Morale is better.
- We've added audit policies that we would never have considered before.

In this section, we will discuss each of these benefits, followed by a few final thoughts.

Better Policy Compliance

The advent of automated procedures has enabled us to comply more fully with our audit policies. Our implemented policies finally match up well with our stated policies.

Every morning when the network administrator comes to work, the hard part of each audit is already done for him, and a report is waiting in his inbox. There are still manual tasks to do related to the audit (for instance, if he sees 15 straight failed logon attempts for an account, and then the attempts just stop, he must research successful logons to see if the failures stopped because the subject successfully logged on, or because they gave up and went away), but they are much easier to do, given good information.

Improved Security Stance

The security of the university network has been improved in a number of ways. Servers are more available because they always have sufficient disk space to operate. Our logon accounts are more secure because logon failures are audited much more frequently than before. Our servers stay patched because we know within a day when new patches are released.

Improved Morale

While morale may not seem like an immediate security concern, it very well can be. A happy network administrator, one who is under less stress than ever before, is more likely to notice problems with his network. As a case in point, since I implemented the automated audits on our network, the network administrator now spends more time watching our Internet bandwidth and monitoring the intrusion detection logs. He has time to run the occasional packet analysis and has discovered new traffic that we should filter at our firewall.

Morale of our users is improved, as well. Server availability is much better than it was before. Our customer service is improved. The network administrator sometimes notices when a user is having logon problems, and is able to contact the user before they have to call the help desk.

Added Audit Policies/Future Work
The fact that we can easily audit our servers using scripts and batch files has led us to do even more auditing. For example, we are now working on procedures to report on usage on our Microsoft Exchange Server. By tracking successful and failed delivery and the quantity of messages handled, we will have a better view of how our network works on a "normal" day. Knowing what's normal just might help us out when things become suddenly abnormal.

Other ongoing projects include:

- An ASP page that allows us to track in near-real-time the number of email viruses stopped at our mail server each day. The script reads from the history database saved by Norton Anti Virus for Microsoft Exchange and give a total for the day, week and month.
- Scripts to provide a monthly report of what email viruses were found, in what quantity, at the mail server. My current prototype even builds a nifty graph, and outputs the report as an HTML document that we can post on our departmental website.
- An automated file system integrity checker that connects to each server and checks the MD5 hashes of important files against a database.
- A script to audit new account creation on each domain. This will let us view enrollment trends by watching as new student users are added, and provides accountability for the technicians who create user accounts.

Conclusion
Knowing your network and its operations in detail is one of the most important aspects of network security. Getting a "feel" for what is normal and what's not could easily mean the difference between detecting a subtle attack and not. Working through the process of automating some basic auditing tasks on my network has given me new insight into what *really* happens on that network on a daily basis. That just might be the most important improvement of all.

List of References

Microsoft Corp. "Windows 2000 Security Event Descriptions (Part 1 of 2)." Microsoft Knowledge Base Article #299475. October 11, 2002.
URL:http://support.microsoft.com/default.aspx?scid=kb;EN-US;299475
(Feb 26, 2003)

Charron, T. "BLAT for Windows. Easily mail any file from the command line." BLAT for Windows. December 12, 2002.
URL: http://www.interlog.com/~tcharron/blat.html (Feb 26, 2003)

Hicks, Jeffery "Event Tracker." CramSession.com. July, 2002.
URL: http://infocenter.cramsession.com/TechLibrary/GetHtml.asp?ID=1696
(Feb 26, 2003)

Microsoft Corp. "Microsoft Network Security Hotfix Checker (Hfnetchk.exe) Tool Is Available." Microsoft Knowledge Base Article # 303215. February 18, 2003.
URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;303215
(Feb 26, 2003)

Microsoft Corp. "Frequently Asked Questions about the Microsoft Network Security Hotfix Checker (Hfnetchk.exe) Tool."
Microsoft Knowledge Base Article #305385. October 11, 2002.
URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;305385"
(Feb 26, 2003)

Syring, Karl M. "Native Win32 Ports of Some GNU Utilities." GNU Utilities for Win32. August 11, 2002. URL: http://unxutils.sourceforge.net/ (Feb 27, 2003)

Appendix A – Source Code Listings

This appendix contains source code listings for each of the custom programs and scripts mentioned in the paper, in the order in which they are mentioned.

Listing 1 – evtLog.vbs

```
'The names of the servers to be checked are given as command-line arguments. Make sure
'at least one was given.

If WScript.Arguments.Count = 0 Then
    WScript.Echo "Usage: EvtLog.vbs server1 [server2] [server3] ..."
    WScript.Quit
End If

'Define the cutoff time for the script (how far back are we looking?). We run this
'script once a day, so we look at only the last 24 hours. The -360 parameter at the
'end indicates that we are in the central timezone. It's the offset from GMT (in minutes)
'for the local timezone.

cutoffTime = Convert2WMITime(DateAdd("h",-24,NOW),-360)

'Iterate thru each server given on the cmd line
For Each strComputer In WScript.Arguments

  'Connect to the server and grab all the Security event log entries with code 529
  Set objWMIService = GetObject("winmgmts:" _
      & "{impersonationLevel=impersonate,(Security)}!\\" & strComputer & "\root\cimv2")
  Set colLoggedEvents = objWMIService.ExecQuery _
      ("Select * from Win32_NTLogEvent where LogFile='Security' and EventCode='529'" ) _

  'The entries are returned in descending chronological order (latest event first),
  'so we can just go through them until we hit the cutoff time, and then quit...
  'We only care about the "Time Written" and "Message" fields for our purposes.

  For Each objEvent in colLoggedEvents
          if objEvent.TimeWritten < cutoffTime then exit for
      wscript.stdout.write "Time Written: " & objEvent.TimeWritten & vbcrlf
      wscript.stdout.write  "Message: " & objEvent.Message
      intCounter = intCounter + 1
  Next
Next

'**************************************************************************************
'These functions are taken from the script EventTracker.wsf
'v1.0 July 2002
'Jeffery Hicks
'jhicks@quilogy.com    http://www.quilogy.com
'They were downloaded from
'http://infocenter.cramsession.com/TechLibrary/GetHtml.asp?ID=1696
'
'They are used to convert time to and from the format used by the Windows Management
'Interface(WMI) calls in VBScript.
'**************************************************************************************

Function ConvWMITime(wmiTime)
  On Error Resume Next

  yr = left(wmiTime,4)
  mo = mid(wmiTime,5,2)
  dy = mid(wmiTime,7,2)
  tm = mid(wmiTime,9,6)

  ConvWMITime = mo & "/" & dy & "/" & yr & " " & FormatDateTime(left(tm,2) & _
    ":" & Mid(tm,3,2) & ":" & Right(tm,2),3)
End Function
```

```
Function Convert2WMITime(dDate,nTimeZone)
  Convert2WMITime=Year(dDate) & Pad(Month(dDate),2,"0") & Pad(Day(dDate),2,"0") & _
    Pad(Hour(dDate),2,"0") & Pad(Minute(dDate),2,"0") & "00.000000" & nTimeZone
End function

Function Pad(sPadding,nWidth,sChar)
  if Len(sPadding)<nWidth then
    Pad=String(nWidth-Len(sPadding),sChar) & sPadding
  else
    Pad=sPadding
  end if
End function
```

Listing 2 – Sample Output from evtLog.vbs

```
Time Written: 20030226094432.000000-360
Message: Logon Failure:

        Reason:         Unknown user name or bad password

        User Name:      admin

        Domain:         netoplaptop

        Logon Type:     3

        Logon Process:  NtLmSsp

        Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0

        Workstation Name:       NETOPLAPTOP
Time Written: 20030226094430.000000-360
Message: Logon Failure:

        Reason:         Unknown user name or bad password

        User Name:      admin

        Domain:         SOSU

        Logon Type:     3

        Logon Process:  NtLmSsp

        Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0

        Workstation Name:       NETOPLAPTOP
```

Listing 3 – Event.pl

```
#!/usr/bin/perl

# This script reads a text file containing the output of the evtLog.vbs script,
# which connects to the Security event log on a Windows server and dumps all the
# 'logon failure' (event code 529) events for a specified period. The file should
# be in the current directory and named 'event.txt'
#
# The script processes the file, line by line, looking for the 'User Name' 'Domain'
# and 'Workstation Name' fields from each event log entry. It builds associative arrays
# that include Domain:UserName pairs and Workstation:UserName pairs. It reports counts for
# each of these pairs, and then summaries for the number of failed logon attempts by
# domain, workstation name, and user name.

#get today's date
($sec,$min,$hour,$mday,$mon,$year,$wday,$ydat,$isdst) = localtime(time());

$year+=1900;    # Y2K all over again...
$mon+=1;        # Month seems to count starting with zero...

$mon = sprintf( "%02d", $mon );
$mday = sprintf( "%02d", $mday );
$hour = sprintf( "%02d", $hour );
$min = sprintf( "%02d", $min );
$sec = sprintf( "%02d", $sec );

print "Failed Logon Report\nRun $mon\/$mday\/$year $hour\:$min\:$sec\n";

open( INFILE, "event.txt" ) || die "Can't open event.txt.\n";
while( <INFILE> )
{
  s/\x0D//gi;

  if(/Time/)   # get the time for our nifty output format
  {
    /Time Written\: (\d\d\d\d)(\d\d)(\d\d)(\d\d)(\d\d)(\d\d)/; #yyyymmddhhmmss
        $year = $1;
        $month = $2;
        $day = $3;
        $hour = $4;
        $minute = $5;
        $second = $6;
  }
  if(/User/)   # get the user name used in this attempt
  {
    s/\t//gi;          # get rid of tab characters
        split/\:/;             # tokenize the line using ':' as the delimiter
        $user = @_[1]; # grab the user name
        chop( $user ); # get rid of the newline character at the end
       # put in some spaces for alignment and readability
        $user = sprintf( "%-16s", $user );
        ++$userCount{$user};   # increment the failed logon count for this user
  }

  if(/Domain/) # get the domain name for this attempt
  {
        s/\t//gi;      #process it like above...
        split/\:/;
        $domain = @_[1];
        chop( $domain );
        $domain = sprintf( "%-16s", $domain );
        ++$domainCount{$domain};

         # handle the array to count Domain:UserName pairs
        $DOMUN = $workStation . "\:" . $user;

# make note of time of the first attempt for this pair
if( $DOMUNCount{$DOMUN} == 0 )
        {
           $DOMUNtime{$DOMUN} = $month . "/" . $day . "/" . $year . " " . $hour .":" . $minute . ":" .
$second;
        }
        ++$DOMUNCount{$DOMUN};
  }

  if(/Workstation/) # same as above, but this time it's Workstation:UserName pairs
  {
    s/\t//gi;
        split/\:/;
        $workStation = @_[1];
        chop( $workStation );
        $workStation = sprintf( "%-16s", $workStation );
        ++$workStationCount{$workStation};

        $WSUN = $workStation . "\:" . $user;
        if( $WSUNCount{$WSUN} == 0 )
        {
           $WSUNtime{$WSUN} = $month . "/" . $day . "/" . $year . " " . $hour .":" . $minute . ":" .
$second;
        }
        ++$WSUNCount{$WSUN};
        }
}
close INFILE;

# Print the report. The output of this program is intended to be captured to a file and
# then sent as an email to concerned parties. This is where all of those associative
# arrays created above get used.

print "\n\n----------------------------------------------------\n";
print "Failed logon attempts by Workstation:Username pair:\n";
print "--------------------------------------------------\n";
foreach $key (sort(keys(%WSUNCount)))
{

  print "$WSUNtime{$key}\t$key\t$WSUNCount{$key}\n";
}

print "\n\n----------------------------------------------\n";
print "Failed logon attempts by Domain:Username pair:\n";
print "--------------------------------------------\n";
foreach $key (sort(keys(%DOMUNCount)))
{

  print "$DOMUNtime{$key}\t$key\t$DOMUNCount{$key}\n";
}

print "\n\n---------------------------------------\n";
print "Failed logon attempts by client domain:\n";
print "-------------------------------------\n";
foreach $key (sort(keys(%domainCount)))
{

  print "$key\t$domainCount{$key}\n";
}

print "\n\n--------------------------------------------\n";
print "Failed logon attempts by client workstation:\n";
print "------------------------------------------\n";
foreach $key (sort(keys(%workStationCount)))
{

  print "$key\t$workStationCount{$key}\n";
}
```

```
print "\n\n-----------------------------------\n";
print "Failed logon attempts by user name:\n";
print "-----------------------------------\n";
foreach $key (sort(keys(%userCount)))
{

  print "$key\t$userCount{$key}\n";
}
```

Listing 4 – Sample of eventSummary.txt file

```
Failed Logon Report
Run 02/26/2003 10:03:49


---------------------------------------------------
Failed logon attempts by Workstation:Username pair:
---------------------------------------------------
02/26/2003 09:44:32     NETOPLAPTOP     :admin                 2


----------------------------------------------
Failed logon attempts by Domain:Username pair:
----------------------------------------------
02/26/2003 09:44:32     :admin                  1
02/26/2003 09:44:30     NETOPLAPTOP     :admin                 1


---------------------------------------
Failed logon attempts by client domain:
---------------------------------------
DOMAIN1                 1
netoplaptop             1


--------------------------------------------
Failed logon attempts by client workstation:
--------------------------------------------
NETOPLAPTOP             2


-----------------------------------
Failed logon attempts by user name:
-----------------------------------
admin                   2
```

Listing 5 – eventLog.bat

```
cscript evtLog.vbs %1 %2 %3 %4 %5 %6 %7 %8 %9> event.txt
event.pl > eventSummary.txt
blat eventSummary.txt –t me@here.com -s "Failed Logon Events for %1 _
%2 %3 %4 %5 %6 %7 %8 %9"
```

Listing 6 – serverEvents.bat

```
call eventlog.bat webserver
call eventlog.bat domCtrl1 domCtrl2
call eventlog.bat domCtrl3 domCtrl3
```

Listing 7 – freeSpace.vbs

```
'freespace.vbs - a script to calculate free space on a drive
'Accepts the drive specifier as a command-line argument
'The drive name may be a local path (c:, d:, etc) or a UNC path
'(\\server\share). Results are printed in MB
dim fileSys, drive
dim objArgs,driveName
dim space, strSpace

set objArgs = wscript.Arguments 'get the command line args
Set fileSys = CreateObject( "Scripting.FileSystemObject" )

driveName = objArgs(0)

Set drive = fileSys.GetDrive(driveName) 'connect to the file system
space = int((drive.FreeSpace/1024)/1024)  'Get space in MB
strSpace=""
if space > 1000 then
  strSpace = int ( space / 1000 )
  strSpace = strSpace & ","
  space = space - (int( space/1000)*1000)
  if space < 100 then strSpace = strSpace & "0"
  if space < 10 then strSpace = strSpace & "0"
  strSpace=strSpace & space
else
  strSpace = space
end if

Wscript.echo strSpace
```

Listing 8 – totalSpace.vbs

```
'totalSpace.vbs - a script to report total space on a drive
'Accepts the drive specifier as a command-line argument
'The drive name may be a local path (c:, d:, etc) or a UNC path
'(\\server\share). Results are printed in MB
dim fileSys, drive
dim objArgs,driveName
dim space, strSpace

set objArgs = wscript.Arguments
Set fileSys = CreateObject( "Scripting.FileSystemObject" )

driveName = objArgs(0)

Set drive = fileSys.GetDrive(driveName)
space = int((drive.TotalSize/1024)/1024)  'Get space in MB
strSpace=""
if space > 1000 then
  strSpace = int ( space / 1000 )
  strSpace = strSpace & ","
  space = space - (int( space/1000)*1000)
  if space < 100 then strSpace = strSpace & "0"
  if space < 10 then strSpace = strSpace & "0"
  strSpace=strSpace & space
else
  strSpace = space
end if

Wscript.echo strSpace
```

Listing 9 – pctSpace.vbs

```
'pctSpace.vbs - a script to calculate free space on a drive as a percentage
'Accepts the drive specifier as a command-line argument
'The drive name may be a local path (c:, d:, etc) or a UNC path
'(\\server\share). Results are printed as percentages to one decimal place
dim fileSys, drive
dim objArgs,driveName
dim space, strSpace

set objArgs = wscript.Arguments
Set fileSys = CreateObject( "Scripting.FileSystemObject" )

driveName = objArgs(0)

Set drive = fileSys.GetDrive(driveName)
space = round( 1000 * drive.FreeSpace/drive.TotalSize ) 'calc percentage
space = space / 10 ' get to one significant decimal place
strSpace=""

strSpace = space & "%"
Wscript.echo strSpace
```

Listing 10 – driveSpace.pl

```
# drivespace.pl - a program to report free drive space on the
# drives specified in "DriveSpecs.txt"
# This program reads the description and location of the drives
# to check from its config file, checks each one using three VBSCRIPTs
# and writes two output files. drives.txt contains a report to be emailed
# to the system administrator, and space.csv contains a history of free
# space available on each drive, suitable for analysis in a spreadsheet.

#open the input file
open (INFILE, "DriveSpecs.txt" ) || die "Can't open input file!!!";
$count=0;

#get the date
($sec,$min,$hour,$mday,$mon,$year,$wday,$ydat,$isdst) = localtime();
$year+=1900;   # fix the date
$mon+=1;

#open the two output files
open (DRIVEFILE, ">drives.txt") || die "Can't open drives.txt!!\n\n";
print DRIVEFILE "Disk Usage Report for $mon-$mday-$year\n";
print DRIVEFILE "All results are in MB.\n\n\n";

open (OUTFILE, ">>space.csv" ) || die "Can't open output file.\n\n";
print OUTFILE "$mon-$mday-$year\,";

while( <INFILE> ) #this is mostly a collection of system calls that builds the output files.
{
  chomp;
  split /\t/;
  $drive=@_[0];
  $desc = sprintf( "%-20s", @_[1]);
  print DRIVEFILE "$desc\t\t";

  $temp = qx/cscript \/\/nologo c:\\drivespace\\pctspace.vbs $drive/;
  chop( $temp );
  print DRIVEFILE "$temp free\t\t";


  $temp = qx/cscript \/\/nologo c:\\drivespace\\freespace.vbs $drive/;
  chop( $temp );
  print DRIVEFILE $temp;

  $temp =~ s/\,//g;
  print OUTFILE "$temp\,";

  print DRIVEFILE " of ";
  $temp = qx/cscript \/\/nologo c:\\drivespace\\totalspace.vbs $drive/;
  chop( $temp );
  print DRIVEFILE $temp;

  $temp =~ s/\,//g;
  print OUTFILE "$temp\,";

  print DRIVEFILE " available.";

  print DRIVEFILE "\n";

  if( $count % 2 )
  {
    print DRIVEFILE"\n";
  }
  $count++;

}
print OUTFILE "\n"; #close the files and go away...
close OUTFILE;
close INFILE;
```

Listing 11 – Sample of drives.txt file

```
Disk Usage Report for 2-18-2003
All results are in MB.


Server1 C Drive              49.8% free            8,609 of 17,280 available.
Server1 E Drive              71.9% free            37,430 of 52,061 available.

SERVER2 C Drive              30.3% free            2,424 of 7,993 available.
SERVER2 D Drive              29.9% free            62,589 of 209,628 available.

SERVER3 C Drive              7.9% free             638 of 8,118 available.
SERVER3 D Drive              47.6% free            29,108 of 61,160 available.
```

Listing 12 – mssecure.bat

```
c:
cd \
cd netcheck
wget http://download.microsoft.com/download/xml/security/1.0/NT5/EN-US/mssecure.cab -O mssecure.cab
expand -r mssecure.cab -F:*.xml .
```

Listing 13 – checkServer.pl

```
@servers = ( "Server1", "Server2", "Server3", "Server4" );
foreach $server( @servers)
{
  print "Checking server: $server\n\n\n";
  system( "mbsacli /hf -x mssecure.xml -h $server > $server.txt" );
  print qx/blat $server\.txt -t me\@here.edu -s "NetCheck $server Report"/;
}
```