VB Solutions
Building a Network Security Monitor

Harness the power of the Win32 APIs
to build a useful VB app without complex coding

BY MICHAEL OTEY

VB Solutions is a new department in *Windows NT Magazine* that shows you how to use Visual Basic (VB) to solve business problems. In this space, we'll build a variety of solutions for specific business problems ranging from network administration to integrating Microsoft Office/BackOffice applications with Object Linking and Embedding (OLE).

This column doesn't teach you how to write VB--instead, it focuses on using VB to provide quick and easy-to-implement solutions. Although a working knowledge of VB is important to understand how the utilities in this column work, you don't have to know VB to benefit from them. You can download the source and executable code for all VB Solutions utilities from *Windows NT Magazine*'s Web site at www.winntmag.com.

Network Security Monitor

This month's solution is a network administration utility--Network Security Monitor--that uses VB to collect and report security violations for your networked Windows NT systems and lets you perform a quick security check on those systems. Network Security Monitor warns you about attempted network security violations by displaying all login failures for each networked NT system. Repeated login failures are a telltale sign of unauthorized network access attempts.

Requirements

You can use Network Security Monitor if you are running the NetBEUI protocol or NetBIOS over TCP/IP on your network. User and password definitions must match across all NT systems you want to monitor. For example, user MIKEO must have the same password on all systems.

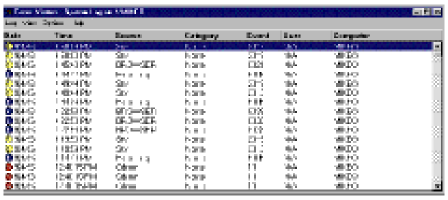
How It Works

Both NT Server and NT Workstation use an Event Log to track security-related events and other system- and application-related events. If you aren't familiar with NT's event logs, see "Windows NT Event Logs," page 153, for a brief explanation of this NT feature. Mike Reilly shows you how to audit your NT security in "Find Holes in Your NT Security," October 1996.

NT also has a built-in Event Viewer that lets you select and view event logs for local and remote systems. However, Event Viewer lets you view only one system at a time, so it's too cumbersome

SCREEN 1:

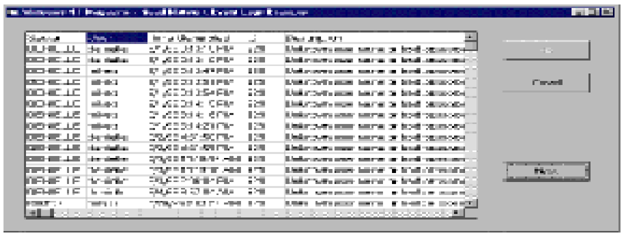
for checking several systems regularly. Network Security Monitor solves this problem by reading event logs from multiple networked NT systems. **Screen 1** shows the Network Security Monitor program's main window.



Reading event logs from multiple networked NT systems with Network Security Monitor main window

Using Network Security Monitor is easy. When the program starts, it retrieves a list of networked NT systems and displays them in the main window's left list box. You can monitor any or all systems in the list that appear in the list box on the right. After choosing the systems, you simply click OK to begin reading those systems' security event logs. The Network Security Monitor highlights each system in the list box on the right as it begins reading that system's log. The program displays a progress bar as it reads through the logs. After Network Security Monitor finishes, the progress bar disappears and a View Results button appears. Clicking *View Results* displays the Network Security Monitor Results window, as shown in **Screen 2**.

SCREEN 2:



Displaying information in the Network Security Monitor Results window

The Network Security Monitor Results window displays a grid that contains the server name, the user, the time stamp the Event Log generated, the event ID, and a brief description of the event. Each system appears in the order you selected it, and events appear chronologically (newest to oldest). Collecting login security errors with Network Security Monitor is a snap.

Inside Network Security Monitor

Considering Network Security Monitor's functionality, you might think that building it requires complicated communications coding and a knowledge of system internals--but that's not the case. NT provides a rich set of more than 800 APIs, most of which you can call from VB. They let you access a variety of system functions. Network Security Monitor takes advantage of a small set of NT APIs to handle the trickiest parts of the program.

The first API Network Security Monitor uses, the Win32 NetBIOS API, lets the program browse for the available networked NT systems. NT supplies this API in the DLL netapi32.dll. It contains many functions; Network Security Monitor uses one, netserverenum, that returns a list of networked systems.

LISTING 1:
VB declaration for NetBIOS API function netserverenum
Declare Function NetServerEnum Lib "Netapi32" _ (vComputerName As Any, ByVal lLevel As Long, vBuffer As Any, lPreferedMaxLen As Long, lEntriesRead As Long, lTotalEntries As Long, vServerType As Any, ByVal sDomain As String, vResume As Any) As Long

To use NetBIOS or any API functions in VB, you must declare them. Listing 1 shows the VB declaration for function netserverenum.

When Network Security Monitor first starts, it calls subroutine getserverinfo in the main form's form_load event, which in turn calls function net serverenum. Listing 2 shows a partial listing of the getserverinfo subroutine. (You can obtain the complete code for getserverinfo and the other VB programs for Network Security Monitor from www.winntmag.com.)

At A in Listing 2, you can see the call to netserverenum. The first parameter identifies the name of the NT system that will run function netserverenum. A null value causes this function to execute on the local system. Specifying a remote server name causes the function to run on a remote system. The second parameter controls the layout of the returned information. This parameter accepts a constant value of 100 or 101; 101 returns only the most basic remote system information. The third parameter is a pointer to a buffer that holds the returned list of networked computers. The fourth parameter specifies the preferred length of data to read. The fifth and sixth parameters return the number of systems read and the total number of systems, respectively. The seventh parameter is a filter for the type of system to include in the list. This parameter accepts several constant values. This example needs the returned list of system names to

LISTING 2:	
Subroutine getserverinfo (partial listing)	
	’Clear all sComputerName. sComputerName = "" vServerType = SV_TYPE_NT
A	’Call NetServerEnum to get a list of servers. lReturnCode = NetServerEnum(ByVal 0&, 101, lSeverInfo101, lPreferedMaxLen, lEntriesRead, lTotalEntries, vServerType, ByVal sDomain, vResume)
	’ NetServerEnum index is 1 based. x = 1 lSeverInfo101StructPtr = lSeverInfo101

include just NT workstations and servers because only NT systems maintain security event logs. To get this information, you enter the constant &H1000 in this parameter, which sets the filter to include only NT systems. (At the top of Listing 2, the constant SV_TYPE_NT represents the value &H1000. It is set in another part of the code.) The eighth parameter is a filter for the domain to search. If this parameter is null as it is here, the primary domain is searched. You can specify another domain name (typically for another network or a different workgroup). The last parameter is not used, but it must be null. (The fact that these parameters are set to null is not obvious based on the variable name, *vResume*, but *vResume* is never given a value; it's simply declared and not used.)

Once the call to netserverenum succeeds, an array of structures is in the vbuffer parameter. Each element in the structure contains information about a particular server. Table 1 shows the layout of the server structure that function netserverenum returns.

The code at B in Listing 2 traverses the returned array of structures, extracting the computer names and adding them to the list box on Network Security Monitor's main form. After subroutine getserverinfo is complete, Network Security Monitor displays the main window with the system list.

B

```
Do While x <= lTotalEntries

RtlMoveMemory tSeverInfo101, ByVal
lSeverInfo101StructPtr, Len(tSeverInfo101)

lstrcpyW bBuffer(0),
tSeverInfo101.wki101_servername
'Get every other byte from Unicode string.
i = 0
Do While bBuffer(i) <> 0
sComputerName = sComputerName &
Chr(bBuffer(i))
i = i + 2
Loop

List_AllServers.AddItem sComputerName

'Debug.Print "Found server: " &
sComputerName
sComputerName = ""
x = x + 1

lSeverInfo101StructPtr =
lSeverInfo101StructPtr + Len(tSeverInfo101)

Loop
```

```
lReturnCode =
NetApiBufferFree(lSeverInfo101)
```

After you select the networked systems to monitor and you click OK, Network Security Monitor loops through the list of selected NT systems. For each selected system, the program calls subroutine readservereventlog to retrieve that system's security Event Log.

Luckily, you don’t need in-depth system internals information to access NT event logs. As you do to obtain the system names, you use a set of system API functions to open and read each system’s Event Log. These functions, which are in advapi32.dll (the Registry API), include openeventlog, which opens an Event Log on a remote or local system; getnumberofeventlogrecords, which returns the number of entries in the log; and readeventlog, which reads the Event Log entries. Listing 3 shows the three functions’ declarations. Subroutine readservereventlog contains calls to each of the three functions; Listing 4 is a partial listing of this subroutine.

For each selected NT system, Network Security Monitor opens the Event Log by calling function openeventlog (A in Listing 4) and reading the security events. The first parameter of openeventlog contains the server name from the list box of selected servers on the main form. The second parameter is a string that specifies the type of Event Log to open. Because Network Security Monitor reads security event logs only, the stringsEventLogType contains the value *security*. Although Network Security Monitor reads only security event logs, you can open and read the application and system event logs by specifying the value *application* or *system,* respectively. When function openeventlog completes successfully, it returns a handle, a unique identifier for the open Event Log.

LISTING 3:
Declarations for functions openeventlog, getnumberofeventlogrecords, and readeventlog
Declare Function OpenEventLog Lib "advapi32" Alias "OpenEventLogA" (ByVal lpUNCServerName As String, ByVal lpSourceName As String) As Long
Declare Function GetNumberOfEventLogRecords Lib "advapi32.dll" (ByVal hEventLog As Long, NumberOfRecords As Long) As Long
Declare Function ReadEventLog Lib "advapi32.dll" Alias "ReadEventLogA" (ByVal hEventLog As Long, ByVal dwReadFlags As Long, ByVal dwRecordOffset As Long, lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, pnBytesRead As Long, pnMinNumberOfBytes Needed As Long) As Long

LISTING 4:	
Subroutine readservereventlog (partial listing)	
’ Open the event log and get the number of records. lEventLogHwd = OpenEventLog(sServerName, sEventLogType)	**A**
If GetNumberOfEventLogRecords(lEventLogHwd, lRecordNumber) <> False Then ’Set up the progress bar. ProgressBar1.Max = lRecordNumber ProgressBar1.Value = 1	**B**

Next, Network Security Monitor calls function getnumberofeventlogrecords to retrieve the number of entries in the log. The code that accomplishes this call is at B. The handle returned by openeventlog is passed as the first parameter. The second parameter returns the number of records in the open Event Log. The program uses this number to set up a progress bar that shows relatively how much of the Event Log you have read. If no records appear in the Event Log, an error occurs, and the program displays a message box.

```
DoEvents
Else
MsgBox "Error getting number of Event Log
Records: " & Err.LastDllError
Exit Function
End If

If lRecordNumber <= 0 Then
MsgBox "No records in event log"
Exit Function
End If
```

The code that reads through the event logs is at C. A loop reads the security Event Log for the number of entries retrieved previously with function getnumberofeventlogrecords. The program increments the progress bar each time the read loop is executed. The program calls function readeventlog, which retrieves data from the security Event Log. readeventlog's first parameter takes the Event Log handle that the call to openeventlog obtained earlier. The second parameter controls whether the program reads the Event Log forward or backward. To display the most current event

```
' Now set up the read flags.
lReadFlags =
EVENTLOG_BACKWARDS_READ Or
EVENTLOG_SEEK_READ

' Loop through the record numbers.
For x = 1 To lRecordNumber
' Increment the progress bar as we read the event
log.
ProgressBar1.Value = x
lReadRecordOffset = x

lNumBytesToRead = 56
ReDim bBuffer(0 To lNumBytesToRead - 1)

TryAgain:
lReturnCode = ReadEventLog(lEventLogHwd,
lReadFlags, lReadRecordOffset, _bBuffer(0),
lNumBytesToRead, lBytesRead,
lMinNumBytesNeeded)

If lReturnCode = False Then
If Err.LastDllError =
ERROR_INSUFFICIENT_BUFFER Then
lNumBytesToRead = lMinNumBytesNeeded
ReDim bBuffer(0 To lNumBytesToRead - 1)
GoTo TryAgain
ElseIf Err.LastDllError =
ERROR_HANDLE_EOF Then
Exit For
Else
MsgBox "Error reading the Event log on: " &
sServerName
Exit For
End If
End If
```

C

```
' Copy the data from the pointer to a string
buffer.
sBuffer = Space$(lBytesRead)
RtlMoveMemory ByVal sBuffer, bBuffer(0),
lBytesRead
```

backward. To display the most current event first, the program must read the security Event Log backward (i.e., from most recent to least recent), so you use the eventlog_backwards_read flag with the eventlog_seek_read flag, which shows where the read will begin. In this case, the flag is at the beginning of the file because of the value of the lreadrecordoffset parameter. The program uses the third parameter with the eventlog_seek_read flag, and the third parameter specifies where in the security Event Log the readeventlog function starts reading. (If you set this parameter to 0, readeventlog starts reading from the beginning of the log.) The fourth parameter is a pointer to a buffer that contains data that readeventlog returns.

readeventlog's fifth parameter sets the number of bytes to read, and the sixth parameter returns the number of bytes that the program actually read. You can read more than one security Event Log entry per call, but for simplicity, this example reads only one record at a time. However, there's a gotcha in reading one entry at a time. Event Log records vary in length, so you can't use a fixed value for the number of bytes. Therefore, the program attempts the first read using an event entry's minimum length. If the return code shows that the call failed due to insufficient buffer space, the program resizes the buffer according to the value returned in the sixth parameter and reattempts the call to function readeventlog. If the call is successful, subroutine readservereventlog calls the Win32 rtlmovememory function to copy the Event Log data from the buffer to a VB string. This API is part of the NT kernel. Its purpose is to copy data from one memory location to another. For convenience, I've declared the rtlmovememory API in both the netapi.bas file and the eventlog.bas file. This configuration lets you easily apply the rtlmovememory function in other projects that use either the NetBIOS APIs or the Event Log APIs.

The data returned by readeventlog follows a predefined layout or structure, as shown in Table 2. The first part of the returned data uses a fixed format. In contrast, the second portion of the returned data varies in length. The offset data in the beginning of the structure identifies the location of the variable data. The returned data includes the time stamp, Event ID, username, computer name, and string data that may contain an informational message.

One point is important to know about reading the Event Log string data that function readeventlog returns: The function doesn't return the same string data you see in NT's Event Viewer. Instead, the string data returned by the API contains substitution parameters that you must convert and reformat before they are meaningful. The substitution parameters don't affect Network Security Monitor because it looks for a specific type of event with known string data. However, if you want to write a more sophisticated application that interprets and displays more types of Event Log data than Network Security Monitor, your program must read the Services key of the NT Registry to retrieve the name of the eventmessages library. It contains the formatmessage function. Then your program must load that library and call the formatmessage function to fill in the substitution parameters for each string entry. (This extra step accommodates internationalization of the string data.)

Powerful APIs

As you've seen, creating a useful VB utility is easy with just a few Win32 APIs. I didn't have to master complex system internals or network coding to develop Network Security Monitor--only call some Win32 API functions. Although using the APIs isn't hard, I suggest you use the Microsoft Developer Network CD to learn about the API functions--as I did.

Although I wrote Network Security Monitor to search for one type of Event Log entry, you can easily modify this program to look for different Event Log entries or to read through the other event logs. Enjoy using Network Security Monitor--and stay tuned for more VB Solutions!

Editor's Note: For the complete source and executable code for this solution, go to the *Windows NT Magazine* site, www.winntmag.com. *Windows NT Magazine* wants to publish your VB solutions. If you've created some interesting and useful VB solutions for your business' problems, send them to us! If we agree that your VB solutions are interesting and useful to our readers, we'll publish your code and pay you $100. You can send contributions or ideas for VB solutions to me at mikeo@teca.com.

ABOUT THE AUTHOR

Michael Otey *is a technical editor for* NEWS/400 *and president of TECA, in Portland, Oregon. You can reach him at mikeo@teca.com.*

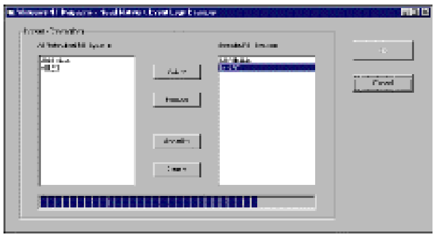
VB SOLUTIONS Building a Network Security Monitor

Windows NT Event Logs

If you aren't familiar with Windows NT's event logs, take a quick look at NT's built-in Event Log viewer. You can run Event Viewer on NT 4.0 from the Start menu by selecting the Program option and then the Administrative Tools (Common) option. The Event Viewer option on the Administrative Tools submenu starts NT's Event Viewer. Screen A shows a sample system Event Log through Event Viewer.

Event Viewer lets you view the NT Event Log for either a local or remote NT system. In Screen A, you can see each event. A color signals its priority: Yellow is for a warning event, blue signals an informational event, and red signifies a warning event. Event Viewer also shows the date and time the event was generated and the event ID, the user, and the computer on which the event was generated.

SCREEN A:



Viewing the NT system log with Event Viewer

The three types of NT event logs are:

* **System log,** which tracks miscellaneous system events (for instance, the system log tracks events during system startup and hardware and controller failures)

* **Application log,** which tracks application-related events (for instance, some applications generate informational messages that appear as entries in the Application Event Log; application errors such as failing to load a DLL can also appear in the application log)

* **Security log,** which tracks events such as logon, logoff, changes to access rights, and system startup and shutdown. However, by default, the security log is turned off. To track security events with Network Security Monitor, you must start NT event logging on the target remote systems. To enable NT security logging, you must sign on with a user ID that has administration rights. Then from the Start Menu, choose Program and then Administrative Tools (Common). From the Administrative Tools submenu, choose User Manager, which displays the User Manager window. Select Audit from User Manager's Policies menu to display the audit dialog

TABLE 1: Server Structure Returned by Function netserverenum

Field	**Type**	**Description**
Platform ID	Long	Information level to use for platform-specific information
Server Name	Long	Computer name
Version Major	Long	The major release version number of the LAN Manager
Version Minor	Long	The minor release version number of the LAN Manager
Server Type	Long	The type of software the computer is running
Server Comment	Long	A comment describing the server

TABLE 2: Event Log Record Structure

Fixed-Format Data	**Type**	**Description**

lLength	Long	Specifies the length, in bytes, of this event record
lReserved	Long	Reserved
lRecordNumber	Long	Contains a record number that can be used with the eventlog_seek_read
lTimeGenerated	Long	The time at which this entry was submitted (seconds since 1-1-1970)
lTimeWritten	Long	Specifies the time at which this entry was received (seconds since 1-1-1970)
lEventID	Long	Identifies the event
iEventType	Integer	Specifies the type of event
iNumStrings	Integer	Specifies the number of strings present in the log (at the position indicated by StringOffset)
iEventCategory	Integer	Specifies a subcategory for this event
iReservedFlags	Integer	Reserved
lClosingRecordNumber	Long	Reserved
lStringOffset	Long	Specifies the offset of the strings within this Event Log entry
lUserSidLength	Long	Specifies the length, in bytes, of the UserSid member
lUserSidOffset	Long	Specifies the offset of the security identifier
lDataLength	Long	Specifies the length, in bytes, of the event-specific data (at the position indicated by DataOffset)
lDataOffset	Long	Specifies the offset of the event-specific information within this log

Variable-Length Data	**Type**	**Description**
vSourceName	Variant	Contains the variable-length null-terminated string specifying the name of the source that generated the entry
vComputername	Variant	Contains the variable-length null-terminated string specifying the name of the computer that generated this event
vSID	Variant	Specifies the security identifier of the active user
vTChar	Variant	Contains the message string