Visual Studio Magazine Online
Classic VB Corner

Creating Admin Tools in a Least Privileged World

Administrators get no respect, either, in this world of Least Privileged Users. Here's how you can set aside those precautions, and still get at the information your users need.

July 27, 2010 · by Karl E. Peterson

Administrators need system information. That's a given, right? Most developers are administrators and need this information as well, if for nothing more than to make sure their wares are functioning correctly. But something as seemingly innocent as querying process information on system processes is now considered highly suspect, and forbidden to most non-system processes.

I found this out the hard way (how else?) with a little utility I wrote to determine how long a system had been running since last restart. My Uptime tool works by extracting the date of last proper shutdown from the registry, as well as checking how long certain system processes have been running. You can read those details in April 2009 and May 2009 Classic VB Corner columns.

I was still happily running XP at the time, though. Upon moving to Windows 7, I was in for a bit of a shock. My calls to OpenProcess were now failing. I thought at first this was due to my querying 64-bit processes, as that was the operating system flavor I'd selected. But a little Googling showed the problem was wider spread than that. It turns out that to successfully open system processes, these days, you need to elevate your process's operating privileges to that of a debugger.

Sounds messy, and it can be a bit as you need to track your privilege state so you can just use this elevated state as needed and then restore back to the previous state. I wrote a little function that toggles the SE_DEBUG_NAME privilege on and off, leaving you the simpler task of keeping track of its current state.

```
Private Function DebugPrivs(ByVal Enable As Boolean) As Boolean
   Dim hProcess As Long
   Dim DesiredAccess As Long
   Dim hToken As Long
   Dim tkp As TOKEN_PRIVILEGES
   Dim nRet As Long

   ' Cache a copy of priviliges as we found them.
   Static bup As TOKEN_PRIVILEGES

   ' Get psuedohandle to current process.
   hProcess = GetCurrentProcess()
   ' Ask for handle to query and adjust process tokens.
   DesiredAccess = TOKEN_QUERY Or TOKEN_ADJUST_PRIVILEGES
   If OpenProcessToken(hProcess, DesiredAccess, hToken) Then
```

```
      ' Get LUID for backup privilege name.
      If LookupPrivilegeValue( _
         vbNullString, SE_DEBUG_NAME, tkp.LUID) Then

         If Enable Then
            ' Enable the debug priviledge.
            tkp.PrivilegeCount = 1
            tkp.Attributes = SE_PRIVILEGE_ENABLED
            If AdjustTokenPrivileges( _
               hToken, False, tkp, Len(bup), bup, nRet) Then
                  DebugPrivs = True
            End If
         Else
            ' Restore prior debug privilege setting.
            If AdjustTokenPrivileges( _
               hToken, False, bup, 0&, ByVal 0&, nRet) Then
                  DebugPrivs = True
            End If
         End If
      End If
      ' Clean up token handle.
      Call CloseHandle(hToken)
   End If
End Function
```

The DebugPrivs function takes a single Boolean parameter which indicates whether to toggle this privilege on or off. Hopefully the comments will give you the general gist of what's going on there. The meat of DebugPrivs lies in the AdjustTokenPrivileges calls, so that'd be a good API to read up on.

AdjustTokenPrivileges requires you pass it a locally unique identifier (LUID) for the privilege you'd like to obtain or release. These privileges have all been assigned constant String values, which are used as look-up references. In this case, we will pass SE_DEBUG_NAME ("SeDebugPrivilege") to the LookupPrivilegeValue API to get the LUID we need. All the security constants are listed on MSDN, and can also be found in the winnt.h file that comes with the downloadable SDK.

With that step out of the way, I just needed to incorporate this privilege elevation into my OpenProcess scheme. I tend to find that the most defensive posture one can adopt is to not ask for more than what's needed. In this case, I need to get progressively more aggressive as the simplest requests fail.

```
Private Sub ProcessOpen()
   Const opFlags As Long = PROCESS_QUERY_INFORMATION
   Const ERROR_ACCESS_DENIED As Long = 5&
   ' Attempt to open by any means possible.
   If (m_hProcess = 0) And (m_PID <> 0) Then
      m_hProcess = OpenProcess(opFlags, False, m_PID)
      ' Try again, with elevated priviledges?
      If Err.LastDllError = ERROR_ACCESS_DENIED Then
         m_Debugging = DebugPrivs(True)
         m_hProcess = OpenProcess(opFlags, False, m_PID)
      End If
      If m_hProcess = 0 Then
         Debug.Print "OpenProcess failed:"; Err.LastDllError
```

```
      End If
   End If
End Sub
```

If the first call to OpenProcess fails, I attempt to elevate my process' privileges to include debugging. This will generally allow me to open most system processes, assuming the user has sufficient privileges (eg, administrator) in the first place. Note that I also set a module-level flag indicating that I'm now running in this debugging state. That allows me to withdraw the elevation when I no longer need the process handle.

```
Private Sub ProcessClose()
   If m_hProcess Then
      Call CloseHandle(m_hProcess)
      If m_Debugging Then Call DebugPrivs(False)
      m_hProcess = 0
   End If
End Sub
```

Now my Uptime tool is working again! If you download the sample on my Web site, it also includes a compiled console implementation, which is where I find most administrators prefer to work with tools like this.

Even though I hardcoded my DebugPrivs routine to specifically toggle a single privilege elevation type, you can easily modify it to work with any of the several other privileges if you find the need to elevate them. Be careful what you ask for, though. Your users will not abide with recklessness in most of these (rightly so!) privileged areas.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.