PREV NEXT

Integrity Levels

Windows Vista is the first version of Windows to include support for integrity protection. Integrity protection adds a layer of defense to help reduce the chance that malicious software will damage the operating system. Integrity levels are often referred to in the various Windows development kits as mandatory labels. It is important to point out that integrity levels don't prevent data disclosure; a privacy leak is still a privacy leak in the face of integrity protections. Integrity protections allow or disallow only write operations, no read operations.

Important The goal of integrity levels is to protect the operating system from damage, not to protect user data from disclosure.

The tenet of integrity levels is pretty simple. A process of a lower integrity level can't write to an object of a higher integrity level. Integrity controls in Windows Vista provide assurance that processes of lower trustworthiness (that is, lower integrity) cannot modify files or system objects of higher trustworthiness (see Figure 2-4). This is often expressed as "write-down, no write-up."

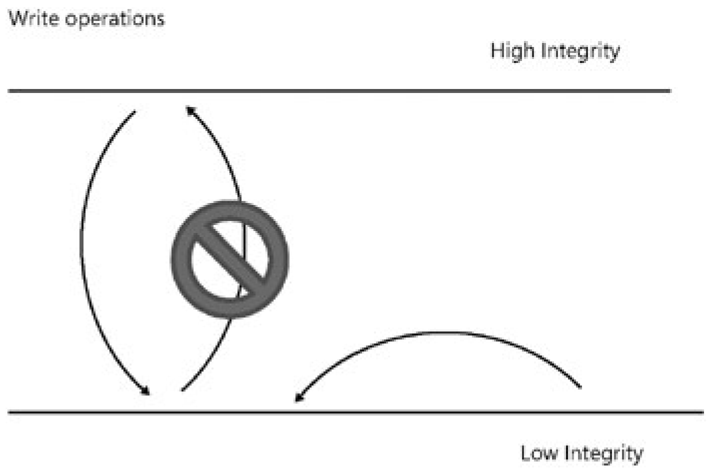


Figure 2-4: Integrity model in Windows Vista. Lower-integrity subjects cannot write to higher-integrity objects.

In Windows Vista, all protected objects are labeled with an integrity level. Most user and system files and registry keys on the system have a default label of "medium" integrity. The primary exception is a set of specific folders and files writeable by Internet Explorer 7 at Low integrity. Most processes run by standard users are labeled with medium integrity, and most services are labeled with System integrity. The root directory is protected by a high-integrity label.

When a process attempts to open an object for write access, the integrity level is checked first. If that check succeeds, then a normal DACL check is performed. The flowchart in Figure 2-5 shows the process.

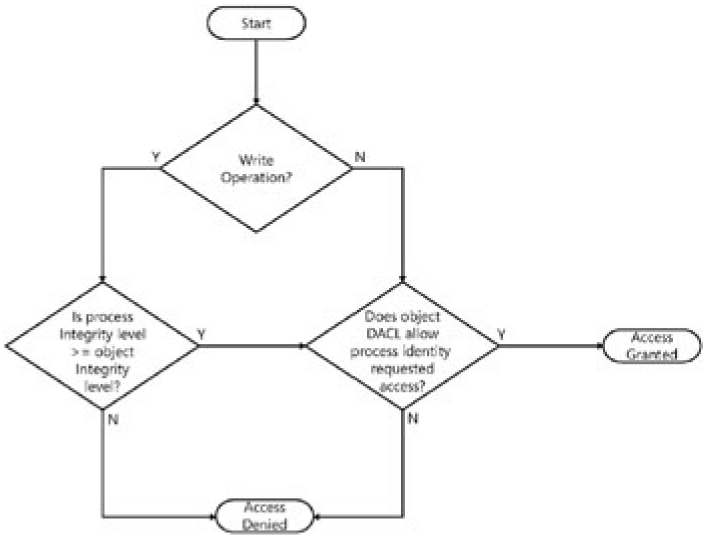

Start
Write Operation?
Y
N
Is process Integrity level >= object Integrity level?
Does object DACL allow process identity requested access?
Access Granted
Access Denied

Figure 2-5: Flowchart showing the process of determining object access in Windows Vista.

The integrity levels in Windows Vista are represented in classic, well-known SID nomenclature, as shown in Table 2-3.

Table 2-3: Integrity SIDs
Open table as spreadsheet

Integrity Levels	SID
Low integrity (LW)	S-1-16-4096
Medium integrity (ME)	S-1-16-8192
High integrity (HI)	S-1-16-12288
System integrity (SI)	S-1-16-16384

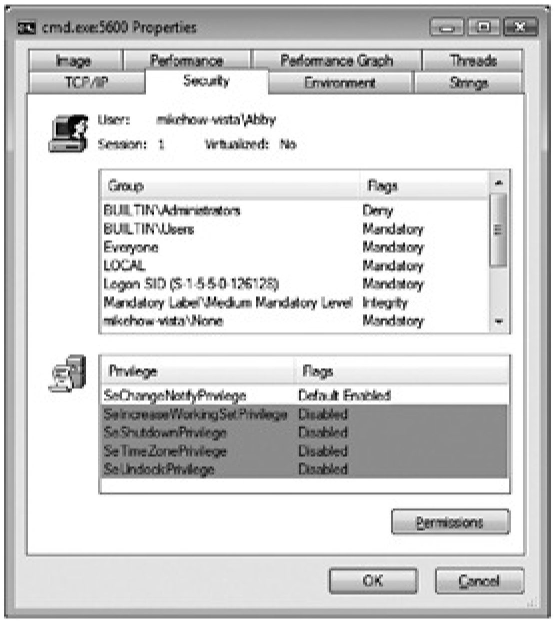
Note It is possible to define custom integrity levels; for example, S-1-16-8200 is between medium and high integrity. But to keep this chapter simple and sane, we focus only on the four standard integrity levels. In fact, you should just stick to these basic four integrity levels!

If you look at the output from `whoami` earlier in this chapter you will see the following entry:

```
Group Name: Mandatory Label\Medium Mandatory Level
Type:       Unknown SID type
SID:        S-1-16-8192
Attributes: Mandatory group, Enabled by default, Enabled group
```

As you can see, the SID in the token shows that this process is running at medium integrity. And yes, “Unknown SID type” is a bug; it should read “Mandatory SID type.”

Process Explorer also shows the integrity level for the application as shown in Figure 2-6. When a user elevates to administrator, the integrity label included in the full-privilege token is a high-integrity level label.



User: mikehow-vista\Abby

Session: 1 Virtualized: No

Group	Flags
BUILTIN\Administrators	Deny
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-126128)	Mandatory
Mandatory Label\Medium Mandatory Level	Integrity
mikehow-vista\None	Mandatory

Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

Figure 2-6: By default, all processes run at a medium-integrity level; this screen shot shows a process running at medium.

You can experiment with integrity levels with the following code, which will run a process at low-integrity level:

```
DWORD err = 0;
HANDLE hToken = NULL;
HANDLE hNewToken = NULL;
PROCESS_INFORMATION ProcInfo = {0};
PSID pIntegritySid = NULL;

try {
      if (!OpenProcessToken(
          GetCurrentProcess(),
          TOKEN_QUERY | TOKEN_DUPLICATE,
          &hToken)) {
      wprintf(L"OpenProcessToken failed (%d)\n", GetLastError());
      throw GetLastError();
}

TOKEN_ELEVATION_TYPE ElevationType;
DWORD cbSize = sizeof TOKEN_ELEVATION_TYPE;
       if (GetTokenInformation(hToken,
          TokenElevationType,
          &ElevationType,
          sizeof(ElevationType),
          &cbSize)) {
       if (ElevationType == TokenElevationTypeFull) {
         throw ERROR_ACCESS_DENIED;
       }
} else {
       wprintf(L"GetTokenInformation failed (%d)\n", GetLastError());
       throw GetLastError();
}

    if (!DuplicateTokenEx(
       hToken,
       TOKEN_ALL_ACCESS,
       NULL,
       SecurityImpersonation,
       TokenPrimary,
       &hNewToken)) {
    wprintf(L"DuplicateTokenEx failed (%d)\n", GetLastError());
    CloseHandle(hToken);
    hToken = NULL;
    throw GetLastError();
}
    if (!ConvertStringSidToSid(
         L"S-1-16-4096",// Low integrity SID
         &pIntegritySid)) {
    wprintf(L"ConvertStringSidToSid failed (%d)\n", GetLastError());
    throw GetLastError();
}

TOKEN_MANDATORY_LABEL til = {0};
til.Label.Attributes = SE_GROUP_INTEGRITY;
til.Label.Sid = pIntegritySid;
   if (!SetTokenInformation(
      hNewToken,
      TokenIntegrityLevel,
      &til,
      sizeof(TOKEN_MANDATORY_LABEL) + GetLengthSid(pIntegritySid))) {
   wprintf(L"SetTokenInformation failed (%d)\n", GetLastError());
   throw GetLastError();
}

STARTUPINFO StartupInfo = {0};
StartupInfo.cb = sizeof(STARTUPINFO);
   if (!CreateProcessAsUser(
     hNewToken,NULL,
     wszProcessName,
     NULL,NULL,
     FALSE,
     CREATE_NEW_CONSOLE,
     NULL,NULL,
     &StartupInfo,&ProcInfo)) {
   wprintf(L"CreateProcessAsUser failed (%d)\n", GetLastError());
   throw GetLastError();
 }
} catch (DWORD dwErr) {
   err = dwErr;
}

if (hToken) CloseHandle(hToken);
if (hNewToken) CloseHandle(hNewToken);
if (ProcInfo.hProcess) CloseHandle(ProcInfo.hProcess);
if (ProcInfo.hThread) CloseHandle(ProcInfo.hThread);
if (pIntegritySid) LocalFree(pIntegritySid);
return err;
```

You can create a low-integrity SID using *ConvertStringSidToSid* as shown in the sample code, or use the following code:

```
SID LowIntegritySid = {
   SID_REVISION, 1,
   {SECURITY_MANDATORY_LABEL_AUTHORITY},
   SECURITY_MANDATORY_LOW_RID;
```

Companion Content This book's companion Web site includes a sample application named SetMIC in the Ch02 folder to start a process at an equal or lower integrity level than the user.

Important Note that in the sample code we reject the request to launch an application if the user has a full admistrative token. Doing so can lead to a potential elevation of privilege vulnerability because the spawned process would be a low-integrity process running as an administrator. You can spawn a low-integrity process from a high-integrity process, so long as the high-integrity process is not executing with a full token.

As a test, log on as an administrator, start a command shell at low integrity with the SetMIC tool, and then try writing to the root directory with the following command:

```
echo "Hello, World!" > c:\test.txt
```

It will fail because the process performing the write operation is of a lower integrity level than the object being written to (low versus high). Note that an account cannot create a process at a higher integrity level than its own integrity level. If you attempt to do so, the call to *SetTokenInformation* will fail, and *GetLastError* will return 1314 (privilege not held) unless the process identity has the *SE_RELABEL_NAME* privilege enabled.

Where Can a Low-Integrity Process Write?

Of course, a low-integrity process can successfully write to very few locations. You can store user-specific data in %userprofile%\AppData\LocalLow; the following code shows how to get this directory:

```
#include "shlobj.h"
#include "knownfolders.h"
wchar_t *wszPath;
if (SHGetKnownFolderPath(
  FOLDERID_LocalAppDataLow,
  KF_FLAG_DEFAULT_PATH,
  NULL,
  &wszPath) == S_OK) {
      // path is in wszPath
      CoTaskMemFree(wszPath);
}
```

Setting Integrity Labels on Objects

You can also create a new object, such as a file, directory, registry key, named pipe, or shared memory at a specified integrity level. The integrity level for an object is set using a new label format for a system ACL (SACL).

If you are familiar with the Security Descriptor Definition Language (SDDL) (Microsoft 2006c), then you already know how to compose an integrity ACE. The SDDL format for an integrity label is:

```
S:flags(ML;inheritance;mask;;;level)
```

- "S" means a SACL.
- The *flags* option describes optional control flags that indicate various protection and inheritance options.
- "ML" means mandatory label ACE type.
- The *inheritance* option describes optional flags that control inheritance and inheritance propogation.
- The *mask* value is one of the following: NW (No-Write up), NR (No-Read up), or NX (No-Execute up). The most common is NW. These settings are explained a little later.
- The level value is the actual integrity level: LW, ME, HI or SI for low, medium, high, and system, respectively.

The following code shows how to set a No-Write up (NW), medium integrity level (ME) for an object, in this case a file.

```
SECURITY_ATTRIBUTES sa = {0};
sa.nLength = sizeof(SECURITY_ATTRIBUTES);
wchar_t *wszSacl = L"S:(ML;;NW;;;ME)";

if (ConvertStringSecurityDescriptorToSecurityDescriptor(
     wszSacl,
     SDDL_REVISION_1,
     &(sa.lpSecurityDescriptor),
     NULL)) {
           wchar_t *wszFilename = L"c:\\files\\foo.txt";
           HANDLE h = CreateFile(wszFilename,
                        GENERIC_WRITE,0, &sa,
                        CREATE_ALWAYS, 0,NULL);
           LocalFree(sa.lpSecurityDescriptor);
           if (INVALID_HANDLE_VALUE == h) {
                 wprintf(L"CreateFile failed (%d)", GetLastError());
           } else {
                 CloseHandle(h);
                 h = NULL;
           }
} else {
        wprintf(L"SDDL Conversion failed (%d)", GetLastError());
}
```

You can look at or set the integrity level of a directory or file on an NTFS volume in Windows Vista with the icacls command. For example, the following command shows that the C:\Users\username\AppData\LowLocal folder used by Internet Explorer in protected mode is set to low integrity:

```
C:\Users\michael\AppData>icacls LocalLow
LocalLow CONTOSO\Michael:(I)(OI)(CI)(F)
       NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
       BUILTIN\Administrators:(I)(OI)(CI)(F)
       Mandatory Label\Low Mandatory Level:(OI)(CI)(NW)
```

You can set the integrity level of a file or directory using the */setintegritylevel* argument.

Note Remember, if no explicit integrity label SID exists in a process token, or on an object, then that object is medium integrity.

Note that objects created by a high-integrity-level subject are medium-integrity-level by default. This was done as an application compatibility mechanism. For instance, to enable file sharing between medium and high; if a high-integrity-level notepad saves a text file in the user's profile, a medium-integrity-level process should be able to modify it. To keep some protection, objects created by a high-integrity-level subject are owned by the Administrators group.

Determining an Object's Integrity Level

You can determine the integrity level of a process with the following code. There really should be no reason to do this other than for troubleshooting, but we added it for completeness.

```
DWORD GetProcessIntegrityLevel(wchar_t __out_ecount_z(cbIl) *wszIl,
                               size_t cbIl) {
   if (!wszIl) return 0xffffffff;
   memset(wszIl,0,cbIl);
   DWORD err = 0;
   try {

        HANDLE hToken = NULL;
        if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken))
             throw GetLastError();
        DWORD cbBuf = 0;
        if (GetTokenInformation(hToken,TokenIntegrityLevel,NULL,0,&cbBuf) != 0)
             throw GetLastError();
        TOKEN_MANDATORY_LABEL * pTml =
             reinterpret_cast<TOKEN_MANDATORY_LABEL*> (new char[cbBuf]);
        if (pTml &&
             GetTokenInformation(
                       hToken,
                       TokenIntegrityLevel,
                       pTml,
                       cbBuf,
                       &cbBuf)) {
             CloseHandle(hToken);
             hToken = NULL;
             DWORD ridIl = *GetSidSubAuthority(pTml->Label.Sid, 0);
             if (ridIl < SECURITY_MANDATORY_LOW_RID)
                  wcscpy_s(wszIl,cbIl,L"?");
             else if (ridIl >= SECURITY_MANDATORY_LOW_RID &&
                  ridIl < SECURITY_MANDATORY_MEDIUM_RID)
                 wcscpy_s(wszIl,cbIl,L"Low");
             else if (ridIl >= SECURITY_MANDATORY_MEDIUM_RID &&
                  ridIl < SECURITY_MANDATORY_HIGH_RID)
                 wcscpy_s(wszIl,cbIl,L"Medium");
             else if (ridIl >= SECURITY_MANDATORY_HIGH_RID &&
                  ridIl < SECURITY_MANDATORY_SYSTEM_RID)
                 wcscpy_s(wszIl,cbIl,L"High");
             else if (ridIl >= SECURITY_MANDATORY_SYSTEM_RID)
                 wcscpy_s(wszIl,cbIl,L"System");
             if (ridIl > SECURITY_MANDATORY_LOW_RID &&
                  ridIl != SECURITY_MANDATORY_MEDIUM_RID &&
                  ridIl != SECURITY_MANDATORY_HIGH_RID &&
                  ridIl != SECURITY_MANDATORY_SYSTEM_RID)
                  wcscat_s(wszIl,cbIl,L"+");
             delete [] reinterpret_cast<char*>(pTml);
             pTml = NULL;
        } else {
             throw GetLastError();
        }
    } catch(DWORD dwErr) {
           err = dwErr;
           wprintf(L"Error %d",GetLastError());
    } catch(std::bad_alloc e) {
           err = ERROR_OUTOFMEMORY;
           wprintf(L"Error %d",err);
    }
      retur n err;
    }
```

Rules for Integrity Settings

The rules for setting object integrity levels are as follows:

- A process (subject) integrity level must be equal to or higher than the integrity level of an object to modify the object, assuming the normal NW mask is used.
- The *SE_RELABEL_NAME* privilege is required to raise the integrity level of an object.
- A process can lower the integrity level of an object for which it can obtain modify access.
- When a server impersonates a client on the same machine, the integrity level in the impersonation token

is the same as the integrity level of the client process.

The NW, NR, and NX Masks

If you look at the SDDL representation of an integrity ACE, you will notice that ACE often includes NW, which means “No-Write up” in the integrity mask. Two other integrity label access masks exist: NR (No-Read up) and NX (No-Execute up). These are broader access policies. For example, a low-integrity process cannot read an object, such as a file, labeled S:(ML;;NR;;;ME), and a low-integrity process cannot read or execute a file labeled S(ML;;NXNR;;;ME).

The security experts reading this are probably thinking, “Isn’t this the Biba integrity model or Bell-Lapadula?” The answer is no. We explored various security models a few years ago, but found they were not appropriate as defined to the highly interactive and general-purpose environment in which Windows Vista is used. So we decided develop a model–which certainly owes a debt to the Biba model–focused on the integrity of the system. That’s what we have today: a technology that has helped form the basis of UAC and Protected Mode in Internet Explorer.

Tip Note that the icacls tool will only set the NW integrity mask.

A Defensive Model Using Integrity Levels

In Chapter 6, “Internet Explorer 7 Defenses,” we explain how Internet Explorer 7 in Windows Vista uses integrity levels to reduce potential damage from malicious Web software. If you dig a little deeper into the integrity model in Windows Vista, you’ll see a very useful defensive model that can be applied to any software that is one socket away from the Internet. At a very high level, the simplified Internet Explorer 7 Protected Mode model looks like the screen shot in Figure 2-7.

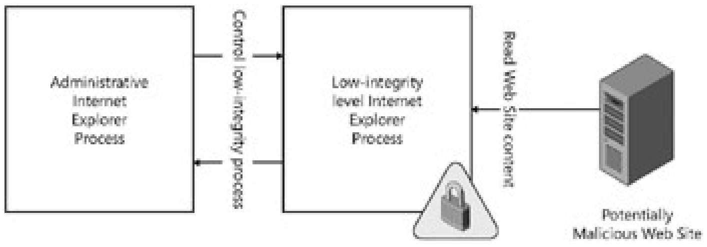

Administrative Internet Explorer Process
Control low-integrity process
Low-integrity level Internet Explorer Process
Read Web Site content
Potentially Malicious Web Site

Figure 2-7: A high-level view of Internet Explorer 7.0 Protected Mode in Windows Vista. The process that parses potentially malicious content runs as a low-integrity process.

Potentially, this model can be replicated to your Internet-facing application. Admittedly, it’s not simple, but it’s a good defense. If your application is currently monolithic, you should start by splitting the process into two logical chunks. The first process performs all the administrative tasks as well as opening the administrative communication channels, perhaps an authenticated named pipe or TCP socket.

The administrative process then spins up the low-integrity process and opens the communication and control channel between the two processes. In some cases, you could start this new process as a low-privilege user account, or you could strip unneeded privileges from the process as it starts up. Note that if the two processes communicate using an interprocess communication protocol (IPC) that supports ACLs (such as anonymous pipes, named pipes, or shared memory and in Windows Vista, sockets), then you must set a low-integrity label on the IPC object; otherwise, the low-integrity process will not be able to write to the IPC.

Companion Content The sample code Ch02\LowIntegrityPipe shows how to open a pipe between a medium-integrity parent process and a low-integrity child process.

Note that a low-integrity process can send data to a higher-privilege process listening on an RPC or LRPC endpoint. Security-aware processes should use client impersonation to determine if the caller has sufficient access to read or modify server managed resources. Also, because sockets don’t use ACLs (other than to open the port in the first place), you can send data from a low-integrity process to a high-integrity process using sockets.

Important It is imperative that the higher-integrity administrative process does not render or parse any low-integrity data that comes from the untrusted, low-integrity process. However, if a high-integrity process must parse low-integrity, then the high-integrity process must verify the data exhaustively.

Note Integrity controls in Windows Vista are a very useful and easy-to-use defense. If your application is one socket away from the Internet, you should consider adding an option to enable your application to run at low integrity. This will help isolate any potential damage an exploit could cause. You should limit which applications run at low integrity to only critical applications that are likely to come under attack, otherwise in a pathological case, if all applications run at low-integrity and access low-integrity resources, then all applications can compromise all data!

PREV NEXT