How can I check the integrity level of my process?

devblogs.microsoft.com/oldnewthing/20221017-00

October 17, 2022

Raymond Chen

Integrity levels capture the sense of "running as a regular Win32 process", "running elevated", "running in a sandbox process", that sort of thing. They describe what degree of security enforcement is applied to the process and how protected the process is from other processes.

You can inspect a process's integrity level by calling `GetTokenInformation` and asking for `TokenIntegrityLevel`. For this trick, I'm going to use the magic `GetCurrentProcessToken()` function to save me the trouble of hunting down the process token (and then closing it when done). And I'll use the `wil::get_token_information` helper function from the Windows Implementation Library to do the grunt work of calling `GetTokenInformation` twice, once to get the buffer size, and again to fill it.

```
#include <wil/token_helpers.h>

DWORD GetCurrentProcessIntegrityLevel()
{
  auto info = wil::get_token_information<TokenIntegrityLevel>(
    GetCurrentProcessToken());
  auto sid = info->Label.Sid;
  return *GetSidSubAuthority(sid,
    *GetSidSubAuthorityCount(sid)-1);
}
```

To get the integrity level, we obtain the `TokenIntegrityLevel` information, which takes the form of a `TOKEN_MANDATORY_LABEL`. That structure consists of a `Label`, and in the `Label` is a `Sid`. That's where the integrity level is.

The integrity level is encoded in the SID as the relative identifier (the final subauthority). So we ask how many subauthorities there are and ask for the last one.

All that's left is mapping that integer to a semantic range.

```
auto integrityLevel = GetCurrentProcessIntegrityLevel();
if (integrityLevel >= SECURITY_MANDATORY_SYSTEM_RID) {
  print("System integrity");
} else if (integrityLevel >= SECURITY_MANDATORY_HIGH_RID) {
  print("High integrity");
} else if (integrityLevel >= SECURITY_MANDATORY_MEDIUM_RID) {
  print("Medium integrity");
} else if (integrityLevel >= SECURITY_MANDATORY_LOW_RID) {
  print("Low integrity");
} else {
  print("Below low integrity?");
}
```

Alternatively, we can check from low to high, but the tests look weird because we're testing the upper boundary of the range, which is named after the *next* range.

```
auto integrityLevel = GetCurrentProcessIntegrityLevel();
if (integrityLevel < SECURITY_MANDATORY_LOW_RID) {
  print("Below low integrity?");
} else if (integrityLevel < SECURITY_MANDATORY_MEDIUM_RID) {
  print("Low integrity");
} else if (integrityLevel < SECURITY_MANDATORY_HIGH_RID) {
  print("Medium integrity");
} else if (integrityLevel < SECURITY_MANDATORY_SYSTEM_RID) {
  print("High integrity");
} else {
  print("System integrity");
}
```

Note the importance of using range checks rather than direct equality checks. That way, you will successfully handle new integrity levels that are created inside an existing range, such as `SECURITY_MANDATORY_MEDIUM_PLUS_RID`, which is an integrity level inserted into the "Medium" range, above the regular `SECURITY_MANDATORY_MEDIUM_RID`. There's also an unnamed integrity level that is `SECURITY_MANDATORY_MEDIUM_RID + 0x10` which is assigned to medium integrity applications with UIAccess rights.

The sample code in the archived content almost gets it right, but it forgets to handle the case of an integrity level less than `SECURITY_MANDATORY_LOW_RID`.

Raymond Chen

Follow