How can I launch an unelevated process from my elevated process, redux

devblogs.microsoft.com/oldnewthing/20190425-00

April 25, 2019

Raymond Chen

Some time ago, I showed how you can launch an unelevated process from an elevated process by asking Explorer to launch the program on your behalf.

There's another way which is a bit more direct, but it assumes that the thing you want to do can be done with a direct `CreateProcess` call. In other words, if you need the system to look up the user's file associations or default browser, then this technique is not for you.

The idea is to take advantage of `PROCESS_ CREATE_ PROCESS` access and the accompanying `PROC_THREAD_ATTRIBUTE_PARENT_PROCESS` process thread attribute:

> **PROC_THREAD_ATTRIBUTE_PARENT_PROCESS**
>
> The *lpValue* parameter is a pointer to a handle to a process to use instead of the calling process as the parent for the process being created. The process to use must have the **PROCESS_CREATE_PROCESS** access right.
>
> Attributes inherited from the specified process include handles, the device map, processor affinity, priority, quotas, the process token, and job object. (Note that some attributes such as the debug port will come from the creating process, not the process specified by this handle.)

Basically, this lets you tell the `CreateProcess` function, "Hey, like, um, pretend that other guy over there is creating the process."

Here's a Little Program to demonstrate. Remember that Little Programs do little to no error checking so that they can demonstrate the underlying technique without distractions.

```
int main(int, char**)
{
  HWND hwnd = GetShellWindow();

  DWORD pid;
  GetWindowThreadProcessId(hwnd, &pid);

  HANDLE process =
    OpenProcess(PROCESS_CREATE_PROCESS, FALSE, pid);

  SIZE_T size;
  InitializeProcThreadAttributeList(nullptr, 1, 0, &size);
  auto p = (PPROC_THREAD_ATTRIBUTE_LIST)new char[size];

  InitializeProcThreadAttributeList(p, 1, 0, &size);
  UpdateProcThreadAttribute(p, 0,
    PROC_THREAD_ATTRIBUTE_PARENT_PROCESS,
    &process, sizeof(process),
    nullptr, nullptr);

  wchar_t cmd[] = L"C:\\Windows\\System32\\cmd.exe";
  STARTUPINFOEX siex = {};
  siex.lpAttributeList = p;
  siex.StartupInfo.cb = sizeof(siex);
  PROCESS_INFORMATION pi;

  CreateProcessW(cmd, cmd, nullptr, nullptr, FALSE,
    CREATE_NEW_CONSOLE | EXTENDED_STARTUPINFO_PRESENT,
    nullptr, nullptr, &siex.StartupInfo, &pi);

  CloseHandle(pi.hProcess);
  CloseHandle(pi.hThread);
  delete[] (char*)p;
  CloseHandle(process);
  return 0;
}
```

We start by getting the shell window and using that to identify the process that is responsible for the shell.

We then use that process ID to open the process with the magic `PROCESS_CREATE_PROCESS` access.

We then ask the system how much memory is required to create a `PROC_THREAD_ATTRIBUTE_LIST` that holds one attribute, and allocate it.

We initialize the newly-allocated attribute list and update it with our process handle, saying that we want this to be the parent for the process we're about to create.

We set up a `STARTUPINFOEX` structure like usual, but we take the extra step of setting the `lpAttributeList` to point to the attribute list.

Finally, we call `CreateProcess`, but also set the `EXTENDED_ STARTUPINFO_ PRESENT` flag to tell it, "Hey, I gave you a `STARTUPINFOEX`, and if you look inside, you might find a surprise!"

A little bit of cleaning up, and we're done.

This program runs a copy of `cmd.exe` using the shell process (usually `explorer.exe`) as its parent, which means that if the shell process is unelevated, then so too will the `cmd.exe` process. Of course, if the user is an administrator and has disabled UAC, then Explorer will still be elevated, and so too will be the `cmd.exe`. But in that case, the user wants everything to run elevated, so you're just following the user's preferences.

Raymond Chen

Follow