Solving the problem rather than answering the question: Why does somebody want to write an unkillable process?

devblogs.microsoft.com/oldnewthing/20130620-00

June 20, 2013

Raymond Chen

Via their customer liaison, a customer wanted to know how to create a process that runs with the context of the user, but which the user cannot terminate without elevating to administrator.

The customer is engaging in the futile arms race between programs and users (which is more properly a walls and ladders scenario). And we saw that Windows has decided to keep users in control of their own programs and data and let them kill any process that belongs to them. (For one thing, allowing a process running in a user's context to protect itself from termination would mean that malware could make itself unkillable without requiring elevation.)

We asked the customer liaison why their program is so important that they don't want the user to terminate it.

The customer liaison explained, "The program is launched when the user logs in, and the customer doesn't want the user to be able to terminate the process from Task Manager."

Observe that the customer (through the liaison) is not answering the question. They keep saying what they want to do (looking for an answer) without describing the problem they are trying to solve (developing a solution). We had to ask them a second time, adding that even if they managed to protect the program from being terminated via Task Manager, that doesn't mean the user can't get the program to *crash*, which is equivalent to terminating it. Now, the typical user won't use these techniques, but the typical user also won't go to the Processes tab of Task Manager and click *End Task* and then accept the scary warning dialog. We're talking about a determined user at this point.

Finally, the customer coughed up the actual scenario. They have a service that monitors some central database. Based on the information in the central database, the service may decide that the user needs to log off. (Say, because the user's profile server is going offline, or because the service needs to reconfigure the system. The exact reason isn't important.) The

application that launches at login listens for a notification from the service that a forced-logoff is about to occur and displays a balloon notification to the user warning them to save all their work because they are going to be logged off in *N* minutes.

At this point, I realized that they didn't have a problem in the first place: The "horrible dire consequences" of the user terminating the application is simply that they don't get balloon notifications? *Who cares?* For one thing, balloon notifications do not have guaranteed timely delivery. For example, Explorer won't show a balloon if the user is in a fullscreen application or if the user is simply not at the computer. Furthermore, the user already has a way of not getting balloon notifications: By ignoring them when they appear on the screen!

If the user kills the "Please save your work and log off because the system is shutting down in *N* minutes for maintenance" notification application, and then they get booted off the computer without warning, well, they get what they deserve. It's like somebody who intentionally pries the safety cover off the emergency power cutoff switch. If they bump into the switch and accidentally kill power to their computer, well, they went out of their way to disable the safety cover—they deserve what they get.

The customer appears not to have been listening very closely, because they summarized the situation as follows: "Right. There are currently two ways to meet the requirements. First, launch the UI process with service privileges, so that the user does not have permission to kill it. Or launch the UI process with user privileges, but marked as unkillable. It looks like your recommendation is to log off immediately if the user terminates the UI process. This will likely be satisfactory, but how can a process force the user to log off if it is terminated?"

I felt like I was stuck in one of those sitcom tropes where the comically obtuse character refuses to process any new information:

"I brought you the boxes from the storeroom."

— *Thanks. Please put them against the wall.*

"Should I put them on the shelf?"

— *No. Please put them against the wall.*

"And then move them to the shelf?"

— *No. Just put them against the wall.*

"Oh, sorry. I'll take them back to the storeroom."

— *No. Leave them here. Just put them against the wall.*

"You want them here?"

— *Yes. Against the wall, please.*

"Okay, I'll put them on the shelf."

The customer took my explanation of why the process doesn't need to be unkillable and reinterpreted it as offering one of three options:

- Launch a UI process with service privileges, which is a well-known security vulnerability.
- Creating a process that cannot be killed by its owner, which we already noted was not possible. (Even if you deny *Terminate* permission, the user can simply edit the ACL to restore *Terminate* permission, because users always have `WRITE_DAC` permission on objects they own.)
- Creating a process that logs the user off if it is terminated, which is possible with the help of the service, but is not something I ever mentioned at all.

I had to issue a clarification to the customer liaison: "My recommendation was not to log the user off immediately if the user kills the UI process. My recommendation is not to worry. Let the users kill the process if they want. They are only harming themselves. By killing the UI process, the users went in and disabled their smoke detector. If they die in a fire, that's their problem."

Unfortunately, this did not settle the issue with the customer, and we had another round of the scene with the comically obtuse character.

"Good news. We found a way to hide the process from the Applications tab, but it still shows up in the Processes tab. We would like to know if there is a way to hide the UI process from the Processes tab as well."

Aaaaarrrrgggghhh.

The customer is totally fixated on putting the boxes on the shelf, no matter how many times we say, "Putting the boxes on the shelf will not solve your problem."

Eventually we settled on some sort of compromise: Have the service monitor the UI process, and if it terminates, log the user off immediately or alternatively relaunch the UI process.

Extra wrinkle: Actually, the original design of the system was that the UI process would show the balloon, then wait *N* minutes, and then call `ExitWindowsEx` to log the user off. That's why they didn't want the user to be able to kill the process: If they killed the process, then nobody was going to perform the logoff. The solution to this was to move the forced logoff into the service. Note that preventing the user from terminating the UI process wouldn't solve their problem anyway, because the user could simply patch out the `Exit-`

`WindowsEx` call or suspend the process so it never got a chance to call `ExitWindowsEx` in the first place. As with Web programming, you need to design on the assumption that the client has been completely compromised.

Raymond Chen

Follow