Why doesn't RevertToSelf undo the most recent SetThreadToken?

devblogs.microsoft.com/oldnewthing/20160518-00

May 18, 2016

Raymond Chen

A customer was experiencing unexpected behavior in their Windows service process with respect to impersonation. The customer's question had two parts. Let's take them one at a time.

> Our service receives a request from a client and impersonates the client in order to satisfy the request.
>
> As part of satisfying the request, the service needs to impersonate a specific unrelated identity in order to get some information. That nested impersonation is done with `SetThreadToken`.
>
> When the nested impersonation is complete, we call `RevertToSelf`. But this does not restore the impersonation to the original client; instead, the thread loses all impersonation and becomes "Network Service", which is the token of the service process.
>
> Is this how the `RevertToSelf` function is supposed to work? MSDN doesn't explicitly mention this.

Here's what MSDN says about `RevertToSelf`:

> **RevertToSelf function**
>
> The **RevertToSelf** function terminates the impersonation of a client application.

It states right there that `RevertToSelf` ends impersonation. When it returns, impersonation has terminated. It is an ex-impersonation.

I guess that's why the function is called `RevertToSelf` and not `RevertToPreviousToken-PriorToMostRecentCallToSetThreadToken`.

The thread token is a single value. It's not a stack of values; `SetThreadToken` does not push a new value onto the top of the stack, and `RevertToSelf` does not pop the top value off the stack and reveal the previous value. For one thing, that model would make it hard to manage

impersonation if you wanted to change impersonation in a non-stack-like manner. Second, maintaining a stack of tokens would create problems if somebody destroyed a token while it was still in the token stack.

Nope, a thread token is just one token. When you call `SetThreadToken`, it replaces the token. When you call `RevertToSelf`, the token is cleared and the thread no longer has a token. Maybe `RevertToSelf` should have been named `ClearThreadToken`, since that would emphasize that the function erases any existing thread token, leaving the thread to inherit the identity of its host process.

If you want to change impersonation to some other identity, then call `SetThreadToken` with the token whose identity you want to impersonate.

Okay, that's part one. The customer's original question anticipated this answer and had a follow-up question.

> Presumably, if this is the expected behavior of the `RevertToSelf` function, then what the code needs to do in order to perform the nested impersonation is
>
> 1. Call `GetThreadToken` to get the current impersonation token.
> 2. Call `SetThreadToken` to set the nested impersonation token.
> 3. Do the necessary work.
> 4. To end nested impersonation, call `SetThreadToken` with the token obtained in step 1 to restore the thread token to the original impersonation token.
>
> Is that correct?

Close.

It's possible that step 1 will fail with `ERROR_NO_TOKEN`. That happens if the thread is not impersonating at all, which means that your code is operating from a flawed assumption. In that case, you have no nested impersonation; you just have impersonation. Step 4 needs to be adjusted as follows:

4. If step 1 failed with `ERROR_NO_TOKEN`, then call `RevertToSelf` to end impersonation. If step 1 succeeded, then the thread was previously impersonating, in which case call `SetThreadToken` with the token obtained in step 1 to restore the thread token to the original impersonation token.
5. Close the thread token obtained in step 1, if any.

The customer replied, "Thanks. It appears that we misunderstood the statement in MSDN."

Raymond Chen

Follow