On the confusing names for the Windows service SID types

devblogs.microsoft.com/oldnewthing/20231004-00

October 4, 2023

Raymond Chen

When you configure a Windows Service process, you can specify what identity the service runs as (*e.g.*, Local Service), and you can also configure how the service SID appears in the service's token. Programmatically, you do this by specifying a `SERVICE_CONFIG_SERVICE_SID_INFO`, with one of the following SID types:[1]

`SERVICE_SID_TYPE_NONE`	Do not add the service SID.
`SERVICE_SID_TYPE_UNRESTRICTED`	Add the service SID as an unrestricted SID.
`SERVICE_SID_TYPE_RESTRICTED`	Add the service SID as a restricted SID.

The names of these values is a little confusing, because the terms "restricted" and "unrestricted" apply not to the token but to the service SID *inside* the token.

Even more confusing is that the documentation for `RESTRICTED` says

> This type includes `SERVICE_SID_TYPE_UNRESTRICTED`.

Huh? "Restricted includes unrestricted"? Shouldn't it be the other way, so that unrestricted is a superset of restricted? You'd think that an unrestricted service should be able to do all the things that a restricted service can do, and more!

All of this becomes less confusing if I rename the flags.

`SERVICE_SID_PRESENCE_NONE`	The service SID is not present.
`SERVICE_SID_PRESENCE_PRESENT`	The service SID is present.
`SERVICE_SID_PRESENCE_RESTRICTED`	The service SID is restricted.

The first step is to decide whether or not the service SID is present at all. If not, then pass `NONE` and you're done.

If you do want the service SID to be present, then pass `PRESENT`. And then you get to decide whether you want it restricted. If so, then also pass `RESTRICTED`. Otherwise, pass only `PRESENT`, and it will be present and unrestricted.

Adding the service SID to the token allows the service to access resources protected by an ACL that grants access to the service SID. Restricting the service SID after adding it further constraints what the process can access.

The remark in the documentation about "Restricted include unrestricted" is a remark about the flags themselves, and not what they mean. The `RESTRICTED` flag also includes the `UNRESTRICTED` (a.k.a. `PRESENT`) bit as a convenience, because the service SID must be present in order for it to be restricted.

Okay, now let's translate from the alternate names back to the official names:

If you don't want the service SID to be present, then pass `NONE`.

If you do want the service SID to be present, then pass `UNRESTRICTED`. If you want it to be present and restricted, then pass `RESTRICTED`. (You don't have to pass `UNRESTRICTED | RESTRICTED` because `RESTRICTED` includes `UNRESTRICTED`.)

Exercise: Now that you understand what these SID types mean, you can address this security vulnerability report:

> I have found a way for one service to access the files created by another service.
>
> First, write a service that create a file in a well-known location. Install the service and configure it to run as Local Service. Use the `sc sidtype` command to set the service SID to unrestricted. Start the service, and observe that the file is created.
>
> Now create a second service which tries to access the file in that well-known location. Install the service service and also configure it to run as Local Service. Start the second service, and observe that it can successfully access the file, even though the first service is unrestricted.
>
> When a service is unrestricted, the files it creates should be owned by the service SID, so that other services cannot access them.

[1] There are other side effects of adding the service SID, which are spelled out in the documentation, but I'm going to focus on whether and how the service SID is added to the SID list.

Answer to exercise: Whether the service SID is present has no effect on the account the service runs as. Both services are running as Local Service, and files created when running as Local Service are owned by Local Service. The service SID in the first service's token does not alter the owner of newly-created files. It merely says that the first service, in addition to accessing everything that Local Service can access, can also access anything which grants access to the service SID.

Therefore, it is not surprising and indeed expected that the second service, also running as Local Service, can access any files created by Local Service.

If you want to customize the owner assigned to objects created by a process, use `SetToken-Information` with `TokenOwner` to specify the SID that should be recorded as the owner of newly-created objects.