How to calculate the resulting security descriptor of a child object without creating it

devblogs.microsoft.com/oldnewthing/20170511-00

May 11, 2017

Raymond Chen

The `CreatePrivateObjectSecurity` function is part of a family of functions intended for programs that implement security descriptors for their own custom objects. Normally, you would let the kernel object manager deal with security descriptors, but if your object isn't a kernel object, then you have to do your own security management. These functions let you give your objects a security model that matches those of kernel objects.

The `CreatePrivateObjectSecurity` function is one of the functions for assigning security descriptors to sub-objects. It understands the rules for "container inherit" and "inherit only" as well as the magic SIDs like "creator owner". This is what your custom `CreateSubObject` function uses to generate the security descriptor for a new sub-object.

But we can use it here to calculate the security descriptor that would be applied to a subdirectory. We pretend that we are the file system, managing our custom "directory" object.

```
GENERIC_MAPPING fileGenericMapping = {
  FILE_GENERIC_READ,
  FILE_GENERIC_WRITE,
  FILE_GENERIC_EXECUTE,
  FILE_GENERIC_ALL,
};
PSECURITY_DESCRIPTOR childSd;
CreatePrivateObjectSecurity(
  parentSd,    // ParentDescriptor
  nullptr,     // CreatorDescriptor
  &childSd,    // NewDescriptor
  TRUE         // IsDirectoryObject
  nullptr,     // Token
  &fileGenericMapping); // GenericMapping
...
DestroyPrivateObjectSecurity(childSd);
```

The `CreatePrivateObjectSecurity` gives you the security descriptor which results from the information you pass in:

- The security descriptor of the parent object. In our case, it's the security descriptor of the parent directory.
- An optional custom security descriptor for the child object. In our case, we pass `NULL` to indicate that we want to inherit from the parent.
- Whether the sub-object is itself a container. The function uses this to decide whether "container inherit" ACEs apply to the new object.
- An optional token representing the user doing the creating. This is used to set the owner and group on the resulting security descriptor, as well as knowing what "creator owner" ACEs should be converted to. We pass `NULL` to say that the function should use the current default token.
- A `GENERIC_MAPPING` structure that specifies how generic access bits should be converted. Fortunately, the generic mapping for file system objects is documented, and it even has convenient names that we can use.

The theory is that this could be used to avoid the race condition when creating a folder that inherits its parent's ACL, and then overriding part of it. That race condition exists in the period of time after the subdirectory is created with default attributes and before the program can apply the new seurity attributes. During that time, somebody might gain access to the directory in a form that would have been disallowed by your override.

Using this technique, you can precalculate what the default attributes would be, then apply your custom override, and then pass those security attributes when you call `CreateDirectory`. This removes the race window where the subdirectory briefly has the wrong security attributes.

Unfortunately, you have other problems.

For one thing, you opened a different race condition: If the security attributes of the parent directory change, then you will apply those stale attributes to the subdirectory.

But the worse thing is that I glossed over the hard part: getting the security descriptor of the parent directory. The `CreatePrivateObjectSecurity` function is intended to be used by the code that is managing security of its custom objects, so it has full access to all the security descriptors. But in this case, we are an outside operator, and getting access to that security descriptor may not be possible. We may not have `READ_CONTROL` access to the DACL. We may not have `ACCESS_SYSTEM_SECURITY` rights to read the SACL.

So this idea sounded good on paper, but runs into problems in practice. Still, I mentioned it because it gives me an excuse to write about the `CreatePrivateObjectSecurity` function.

Raymond Chen

Follow