How come a duplicated token doesn't behave identically to the original?

devblogs.microsoft.com/oldnewthing/20160512-00

May 12, 2016

Raymond Chen

A customer was experimenting with tokens and discovered that things fall apart when they have a thread impersonate itself. Shouldn't that have no effect? Here's what they discovered. Error checking and cleanup have been elided for expository purposes.

```
// This call succeeds
CComPtr<IUnknown> something;
CoCreateInstance(CLSID_Something, nullptr,
                 CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&something));

// Get the current token for the thread.
// This call also succeeds. (Note that OpenThreadToken
// fails if the thread is not impersonating.)
HANDLE token;
OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, TRUE, &token);

// Duplicate the token. This call succeeds.
HANDLE dupToken;
DuplicateToken(token, SecurityImpersonation, &dupToken);

// Impersonate the duplicate. This call succeeds.
ImpersonateLoggedOnUser(dupToken);

// But now, CoCreateInstance fails with E_ACCESSDENIED!
CComPtr<IUnknown> something2;
CoCreateInstance(CLSID_Something, nullptr,
                 CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&something2));
```

The `DuplicateToken` function says that the new token duplicates the original, but it does not appear to be a true duplicate because when we swap out the original thread token for the duplicate, things stop working. What's going on?

There are a lot of things in a token. But there's something important that's not in the token.

One of my colleagues from the kernel team explains: When you duplicate a token with the `DuplicateToken` function, it creates a new kernel object, namely the token, and the new token is a duplicate of the original. But the new token has its own properties, and the important one here is the security descriptor.

When a new kernel object is created, and you don't provide an explicit security descriptor for the new object, then the object is given a default security descriptor. And that default security descriptor comes from the default DACL of the token that is in effect at the point of the call.

When you apply this rule to tokens, you find that, even though the behavior is consistent with other kernel objects, it also means that it is very easy to create a token that doesn't have access to itself. When you impersonate with that token, bad things happen.

It's like going to the FedEx Office store and giving them a DHL envelope with the instructions, "Please make a copy of this letter." They take the letter out of the envelope, make a copy, and then take the copy and give it to you *in a FedEx Office envelope*. They copied the letter, like you instructed, but it's in a different envelope.

If you also want to duplicate the security descriptor, you can get the original token's security descriptor with `GetKernelObjectSecurity` or `GetSecurityInfo`, and then pass that security descriptor to `DuplicateTokenEx`.

The customer confirmed that the recommendation worked.

```
// This call succeeds
CComPtr<IUnknown> something;
CoCreateInstance(CLSID_Something, nullptr,
                 CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&something));

// Get the current token for the thread.
// This call also succeeds. (Note that OpenThreadToken
// fails if the thread is not impersonating.)
HANDLE token;
OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, TRUE, &token);

//Get the security descriptor for the token.
// This call succeeds.
PACL dacl;
PSECURITY_DESCRIPTOR sd;
GetSecurityInfo(token, SE_KERNEL_OBJECT, DACL_SECURITY_INFORMATION,
    nullptr, &dacl, &sd);

// Duplicate the token with that security descriptor.
// This call succeeds.
SECURITY_ATTRIBUTES sa = { sizeof(sa), sd, TRUE };
HANDLE dupToken;
DuplicateTokenEx(token, MAXIMUM_ALLOWED, &sa, SecurityImpersonation,
    TokenImpersonation, &dupToken);

// Impersonate the duplicate. This call succeeds.
ImpersonateLoggedOnUser(dupToken);

// CoCreateInstance now succeeds.
CComPtr<IUnknown> something2;
CoCreateInstance(CLSID_Something, nullptr,
                 CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&something2));
```

Raymond Chen

Follow