A function for creating an absolute security descriptor from a self-relative one

devblogs.microsoft.com/oldnewthing/20241002-00

October 2, 2024

We saw a little while ago that the `CoInitializeSecurity` function demands an absolute security descriptor, even though the security descriptor you have in your hand is usually a self-relative one.[1]

So let's write a helper function for converting from self-relative to absolute format.

The `MakeAbsoluteSD` function does the bulk of the work. We just have to allocate the memory for the absolute security descriptor. And we may as well use one giant memory allocation for the whole thing.

```
HRESULT
    make_hlocal_absolute_sd_from_self_relative_nothrow(
        PSECURITY_DESCRIPTOR relative,
        PSECURITY_DESCRIPTOR* result)
{
    DWORD bodySize, daclSize, saclSize, ownerSize, groupSize;
    RETURN_HR_IF(E_UNEXPECTED,
        MakeAbsoluteSD(relative,
            nullptr, &bodySize,
            nullptr, &daclSize,
            nullptr, &saclSize,
            nullptr, &ownerSize,
            nullptr, &groupSize));

    auto error = GetLastError();
    RETURN_HR_IF(HRESULT_FROM_WIN32(error), error != ERROR_INSUFFICIENT_BUFFER);

    auto totalSize = bodySize + daclSize + saclSize + ownerSize + groupSize;
    RETURN_HR_IF(E_UNEXPECTED, totalSize < bodySize);
    auto absolute = wil::make_unique_hlocal_nothrow<BYTE[]>(totalSize);
    RETURN_IF_NULL_ALLOC(absolute);

    auto dacl = absolute.get() + bodySize;
    auto sacl = dacl + daclSize;
    auto owner = sacl + saclSize;
    auto group = owner + ownerSize;
    RETURN_IF_WIN32_BOOL_FALSE(
        MakeAbsoluteSD(relative,
            absolute.get(), &bodySize,
            reinterpret_cast<PACL>(dacl), &daclSize,
            reinterpret_cast<PACL>(sacl), &saclSize,
            reinterpret_cast<PSID>(owner), &ownerSize,
            reinterpret_cast<PSID>(group), &groupSize));

    *result = absolute.release();
    return S_OK;
}

wil::unique_hlocal_security_descriptor
    make_hlocal_absolute_sd_from_self_relative(
        PSECURITY_DESCRIPTOR relative)
{
    wil::unique_hlocal_security_descriptor result;
    THROW_IF_FAILED(make_hlocal_absolute_sd_from_self_relative_nothrow(
        relative, result.put()));
    return result;
}
```

It’s a standard two-step. First ask how much memory you need, then allocate it. The trick here is that instead of allocating separate memory blocks for the body, DACL, SACL, owner, and group, we instead allocate one big block and put everything in it. This allows us to return a single pointer as the `PSECURITY_DESCRIPTOR`, and freeing that pointer frees everything.

The calculation of the total size risks overflow only when we add the body size to the sizes of the other pieces. The maximum sizes of SIDs and ACLs are well below the point of overflow: ACLs have a maximum size of 65535 bytes (since the size is stored in a 16-bit unsigned integer), and SIDs have a maximum size of `SECURITY_MAX_SID_SIZE` = 68 bytes. The only thing whose size we do not have any insight into is the body, so we can add up the other pieces without worrying about overflow, and then check for overflow when adding the body size.

You can use this helper function to create an absolute version of a self-relative SID for the `CoInitializeSecurity` function.

```
wil::unique_hlocal_security_descriptor rel;
ULONG size;
// 3 = COM_RIGHTS_EXECUTE | COM_RIGHTS_EXECUTE_LOCAL
THROW_IF_FAILED(ConvertStringSecurityDescriptorToSecurityDescriptorW(
    L"O:PSG:BUD:(A;;3;;;WD)", SDDL_REVISION_1, psd.put(), &size));
THROW_IF_FAILED(CoInitializeSecurity(
    make_hlocal_absolute_sd_from_self_relative(
        rel.get()).get(),
    -1, nullptr, nullptr, RPC_C_AUTHN_LEVEL_DEFAULT,
    RPC_C_IMP_LEVEL_IDENTIFY, nullptr, EOAC_NONE, 0));
```

[1] Even more ironically, the `CoInitializeSecurity` function itself wants a self-relative security descriptor! The failure occurs because it’s trying to convert to self-relative format and can’t because the security descriptor is *already* self-relative.