Why does CryptDestroyHash crash, but only sometimes?

devblogs.microsoft.com/oldnewthing/20160127-00

January 27, 2016

Raymond Chen

A customer was having a problem with the cryptographic hashing functions. They reported that their function ran successfully most of the time, but once in a while, it crashed at the call to `CryptDestroyHash` :

```
bool SomethingSomething(BYTE *buffer, int bufferSize)
{
    bool succeeded = true;
    HCRYPTPROV provider = 0;
    HCRYPTHASH hash = 0;

    if (!CryptAcquireContext(&provider, NULL, NULL,
                             PROV_RSA_FULL, CRYPT_VERIFYCONTEXT) ||
        !CryptCreateHash(provider, CALG_MD5, 0, 0, &hash))
    {
        succeeded = false;
        goto Exit;
    }

    BYTE hashResult[16]; // MD5 hash is 16 bytes
    DWORD hashResultSize = sizeof(hashResult);

    if (!CryptHashData(hash, buffer, bufferSize, 0) ||
        !CryptGetHashParam(hash, HP_HASHVAL, hashResult,
                                          &hashResultSize, 0)) {
        succeeded = false;
        goto Exit;
    }

    DoSomethingWith(hashResult); // some business logic

    if (provider) {
        CryptReleaseContext(provider, 0);
    }

    if (hash) {
        CryptDestroyHash(hash);
    }

Exit:

    return succeeded;
}
```

The reason for the crash is straightforward. As noted in the documentation, you must call `CryptDestroyHash` before `CryptReleaseContext`. (The technical reason for this is that each hash has a reference back to the context, so if you destroy the context, you leave the hash with a dangling pointer.)

This was a relatively straightforward consult. A simple programming error. The customer thanked us for identifying the problem, but then followed up with "But why is it happening only rarely? Shouldn't it crash all the time?"

Remember that when you break the rules, the behavior is undefined, and one valid manifestion of undefined behavior is "Everything seems to work okay."

You may have noticed some other problems with the code provided.

- If anything goes wrong, the calls to `CryptDestroyHash` and `CryptReleaseContext` are skipped, which means that the code leaks a hash and a context. The `Exit` label should be moved to just in front of the `if (provider)`.
- Setting `succeeded = true` and then manually setting it to `false` when something goes wrong strikes me as a high-risk proposition. If somebody adds code to the function and does a `goto Exit;` without also setting `succeeded = false;`, the function will falsely report success. I prefer to fail safe and initialize `succeeded = false;`, and set it to `true` only after I am sure that the function succeeded.

Using RAII would have solved both the order-of-destruction problem and the memory leaks.

```
bool SomethingSomething(BYTE *buffer, int bufferSize)
{
    // assuming suitable definitions for CryptProv and CryptHash
    CryptProv provider(NULL, NULL, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT);
    if (!provider) return false;
    CryptHash hash(provider.get(), CALG_MD5, 0, 0);
    if (!hash) return false;

    BYTE hashResult[16]; // MD5 hash is 16 bytes
    DWORD hashResultSize = sizeof(hashResult);

    if (!CryptHashData(hash.get(), buffer, bufferSize, 0) ||
        !CryptGetHashParam(hash.get(), HP_HASHVAL, hashResult,
                                          &hashResultSize, 0)) {
        return false;
    }

    DoSomethingWith(hashResult); // some business logic

    return true;
}
```

Raymond Chen

Follow