I accidentally performed an operation on INVALID_HANDLE_VALUE, and it worked: What just happened?

devblogs.microsoft.com/oldnewthing/20230914-00

September 14, 2023

Raymond Chen

Suppose you have some code that wants to open a file and share it with another process.

```
HANDLE file = CreateFile(...);

HANDLE dup;
if (DuplicateHandle(
        GetCurrentProcess(), /* source process */
        file, /* source handle */
        targetProcess,
        &dup,
        0,
        FALSE, /* no inherit */
        DUPLICATE_SAME_ACCESS)) {
    ... tell the other process to use "dup" ...
}

CloseHandle(file);
```

This code is missing some critical error handling: What if the `CreateFile` fails?

If `CreateFile` fails, it returns `INVALID_HANDLE_VALUE`. This code then passes `INVALID_HANDLE_VALUE` as the source handle to duplicate into the other process. But instead of failing, the `DuplicateHandle` succeeds and produces a handle. What handle just got duplicated?

Some time ago, we studied why `HANDLE` return values are so inconsistent, and I mentioned that

> By coincidence, the value `INVALID_HANDLE_VALUE` happens to be numerically equal to the pseudohandle returned by `GetCurrentProcess()`.

Therefore, what happened is that we gave a copy of our own process handle to the other process.

Oops.

But wait, there's more. The handle returned by `GetCurrentProcess()` has `PROCESS_ALL_ACCESS` permission. Not only did you give the other process the wrong thing, you gave it possibly the worst possible thing: You gave it *full control over your own process*.

Double oops.

Why does `GetCurrentProcess()` return a pseudo-handle value that matches a common error case that people could overlook? I don't know, but I have an idea.

Developer 1: "Hey, what fake handle value should `GetCurrentProcess()` return?"

Developer 2: "I dunno. We need to pick something that is guaranteed never to accidentally match a real handle value."

Developer 1: "Look, there's this special value `INVALID_HANDLE_VALUE` that is returned to indicate that an error occurred. This is provably a handle value that can never match a real handle, since it is used to indicate a problem."

Developer 2: "Great, let's use that!"

It seemed like a good idea at the time. For a special value, use something that couldn't possibly conflict with a normal value.

Unfortunately, people are fallible, and bugs can occur like

```
HANDLE file = INVALID_HANDLE_VALUE;

if (want_file) {
    file = CreateFile(...);
    if (file == INVALID_HANDLE_VALUE) {
        error();
        return;
    }

/* other intervening code */

// Okay, give the file to the other process.
HANDLE dup;
if (DuplicateHandle(
        GetCurrentProcess(), /* source process */
        file, /* source handle */
        targetProcess,
        &dup,
        0,
        FALSE, /* no inherit */
        DUPLICATE_SAME_ACCESS)) {
    ... tell the other process to use "dup" ...
}

CloseHandle(file);
```

The code that calls `DuplicateHandle` forgot to check whether `want_file` is set and inadvertently passed `INVALID_HANDLE_VALUE`. Oops, duplicated a full-access process handle.

This pattern of “using an invalid value to carry a special alternate meaning” is actually quite common. For example, `SetWindowsHookEx` uses a thread ID of zero to mean “global hook”, since zero is not a valid thread ID. And many functions imbue the special value `nullptr` with all sorts of special meaning.

In retrospect, the choice to use `INVALID_HANDLE_VALUE` as a pseudohandle was unfortunate. But what happened happened. You can’t change the past, and in computer software, you have to live with your mistakes.

Bonus chatter: RPC has the `system_handle` attribute which lets you pass kernel handles via RPC, and you also have to say what kind of kernel handle you intend to pass. For files, you say `system_handle(sh_file)`. If you pass the wrong kind of handle, the operation fails. There’s a sample in the Windows Classic Samples repo.