It rather involved being on the other side of this airtight hatchway: Posting completions to somebody else’s I/O completion port

devblogs.microsoft.com/oldnewthing/20240917-00

September 17, 2024

A security vulnerability report claimed that you can accomplish code execution into another process by attacking its thread pool I/O completion port: The completion information for the thread pool’s I/O completion port contains (among other things) a pointer to a structure that contains a pointer to a structure that contains a function pointer, and an attacker can achieve code injection by creating fake data structures in the target process, and then posting a fake completion to the target process’s thread pool I/O completion port. The thread pool code will blindly trust the pointers it receives, process them as if they were genuine thread pool IO completion notifications, and eventually call the function pointer. The finder included a proof of concept and said that they planned to present a paper on this vulnerability at a conference in a few months.

Okay, let’s see. The first thing you have to do is create some fake data structures in the victim process, so you need to do some `VirtualAllocEx` and `WriteProcessMemory` to get the memory in the process to have the necessary structure.

The next thing you have to do is get access to the target process’s thread pool I/O completion port, so that’s probably some `ReadProcessMemory` calls to find its value, and then a `DuplicateHandle` to extract it. (Or you could just brute force it by calling `DuplicateHandle` repeatedly until you find something that’s an I/O completion port.)

So what permissions did we require here?

Creating the fake data structures required `PROCESS_VM_OPERATION` to allocate the memory (though you may be able to avoid this if you can just find some existing memory to repurpose) and `PROCESS_VM_WRITE` to build the fake data structures (though you can avoid this if you can leverage an existing a write-what-where vulnerability somewhere else in the process).

Gaining access to the I/O completion port handle requires `PROCESS_DUP_HANDLE` access.

But if you already have that much power over the victim process, then you don’t need to go through all this trouble!

If you have `PROCESS_VM_WRITE` access or an existing write-what-where vulnerability, you could just find an existing *real* data structure in the victim’s memory and overwrite the callback pointer directly!

In other words, `PROCESS_VM_WRITE` / write-what-where access is itself sufficient to take over a process. All the other stuff is just style points, accomplishing the same task but in a more roundabout and impressive-looking way.[1]

It turns out that `PROCESS_DUP_HANDLE` is also extremely powerful. You can use it to steal any handle from a process, or to replace any handle with another handle. (Replacement is a bit tricky: You have to close the handle, and then duplicate a new handle to replace it. You don’t get to choose the value of the new handle, so you just have to keep calling it until the system gives you the value you want by reusing the slot of the old handle.)

We asked the finder what conference it was at which they intended to present their findings. We never heard back.

[1] It’s like when circus acrobats “mess up” a stunt the first time, then reset and “succeed” the second time. This gets a better crowd reaction than just succeeding on the first try.