It rather involved being on the other side of this airtight hatchway: Attacking another program by modifying its memory

devblogs.microsoft.com/oldnewthing/20240102-00

January 2, 2024

Raymond Chen

A security vulnerability report arrived that took the following form:

> There is a security vulnerability in XYZ.DLL. This DLL contains a function pointer stored in memory. An attacker can modify this function pointer and gain arbitrary code execution.

The proof of concept went like this:

```
auto processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, victimProcessId);
void* value = new_value_for_function_pointer;
void* victimAddress = address_of_variable_in_victim_address_space;
WriteProcessMemory(processHandle, victimAddress, &value, sizeof(value), &actual);
```

It is evident from this proof of concept that we are already on the other side of this airtight hatchway: `PROCESS_ALL_ACCESS` gives you total control over the victim process. If you wanted to gain control over it, just inject a thread and go to town! No need to hunt around for a function pointer you can overwrite to point to some other function, and the presumably arrange for that other function to do something unexpected when it is called.

The finder explained that the `WriteProcessMemory` was just a way to simulate a write-what-where vulnerability. The real attack would be to find a write-what-where vulnerability somewhere in the victim process, and then leverage that to overwrite the function pointer in XYZ.DLL.

That still assumes that you're on the other side of the airtight hatchway. If you have gained write-what-where control over the process, then you can overwrite a return address onto the stack or overwrite an object's vtable pointer to point to a constructed fake vtable. There's nothing specific about XYZ.DLL. Any component that uses function pointers will do.[1]

The real vulnerability is the component that has a write-what-where vulnerability. That's the guy that let you into the airtight hatchway. But once you get there, you've already won.

Bonus chatter: Address Space Layout Randomization (ASLR) and Control Flow Guard (CFG) make these types of attacks harder to carry out remotely. In particular, this finder quietly disabled Call Flow Guard protection as part of their proof of concept so that their overwritten function pointer would be used without validation.

[1] Unless your point is that all function pointers are a security vulnerability?