It rather involved being on the other side of this airtight hatchway: Code injection via QueueUserAPC

devblogs.microsoft.com/oldnewthing/20150519-00

May 19, 2015

Raymond Chen

A security vulnerability report arrived that took the following form:

> The `QueueUserAPC` function can be used to effect an elevation of privilege, as follows:
>
> 1. Identify a process you wish to attack.
> 2. Obtain access to a thread with `THREAD_SET_CONTEXT` access.
> 3. Make some educated guesses as to what DLLs are loaded in that process. Start with `kernel32.dll`, since you're going to need it in step 5.
> 4. From the attacking process, scan the memory of those DLLs looking for a backslash, followed by something that can pass for a path and file name. Such strings are relatively abundant because there are a lot of registry paths hard-coded into those binaries. Suppose you found the string `\Windows NT\CurrentVersion\AppCompatFlags`. Even though ASLR randomizes DLL placement, the placement is consistent among all processes, so an address calculated in one process is highliy likely to be valid in all processes.
> 5. Create a DLL called `C:\Windows NT\CurrentVersion\AppCompatFlags.dll`. Put your payload in this DLL.
> 6. From the attacking thread, call `QueueUserAPC` with the address of `LoadLibraryW` as the function pointer, the victim thread as the thread handle, and a pointer to the fixed string identified in part 4 as the last parameter.
> 7. The next time the victim thread processes APCs, it will pass `\Windows NT\Current-Version\AppCompatFlags` to the `LoadLibraryW` function, which will load the payload DLL, thereby effecting code injection and consequent elevation of privilege.

Note that this attack fails if the victim thread never waits alertably, which is true of most threads.

If you have been paying attention, the alarm bells probably went off at step 2. If you have `THREAD_SET_CONTEXT` access to a thread, then you pwn that thread. There's no need to use `QueueUserAPC` to make the thread do your bidding. You already have enough to make the thread dance to your music. In other words, you are already on the other side of the airtight hatchway.

Here’s how: Look for a code sequence that goes

```
push someregister
call LoadLibraryW
```

Use the `SetThreadContext` function to set the pushed register equal to the address of the string you found in step 4, and set the instruction pointer to the code fragment. The thread will then resume execution at the specified instruction pointer: It pushes the address of the string, and then it calls `LoadLibraryW`. Bingo, your DLL loads, and you didn’t even have to wait for the thread to wait alertably.

On non-x86 platforms, this is even easier: Since all other platforms use register-based calling conventions, you merely have to load the address of the string into the “first parameter” register (`rcx` for x64) and set the instruction pointer to the beginning of `LoadLibaryW`.

By default, `THREAD_SET_CONTEXT` access is granted only to the user, and never to lower integrity levels. In other words, a low IL process cannot get `THREAD_SET_CONTEXT` access to a medium or high integrity thread, and a medium IL process cannot get access to a high integrity thread. This means that, by default, you can only get `THREAD_SET_CONTEXT` access to threads that have equivalent permissions to what you already have, so there is no elevation.

Raymond Chen

Follow