Pulling sleight of hand tricks in a security vulnerability report, episode 2

devblogs.microsoft.com/oldnewthing/20230607-00

June 7, 2023

Raymond Chen

A security vulnerability report came in that claimed to have “found a way for privileged accounts to force the system to crash, and for non-privileged accounts to force the termination of any process.” They claim that they were exploiting a vulnerability in `MessageBox`.

They included a proof of concept, which went something like this.

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

public class Program
{
  [DllImport("kernel32.dll")]
  static extern IntPtr OpenProcess(int access, bool inherit, int id);

  [DllImport("kernel32.dll")]
  static extern IntPtr VirtualAllocEx(IntPtr process, IntPtr address,
   int size, uint type, uint protection);

  [DllImport("kernel32.dll")]
  static extern bool WriteProcessMemory(IntPtr process, IntPtr address,
    IntPtr source, int size, out uint written);

  [DllImport("kernel32.dll")]
  static extern IntPtr CreateRemoteThread(IntPtr process,
   IntPtr attributes, uint stackSize, IntPtr address,
   IntPtr parameter, uint flags, out uint threadId);

  [DllImport("user32.dll")]
  static extern int MessageBox(IntPtr window, string text, string caption, uint
type);

  [DllImport("user32.dll")]
  public static extern int EnableMenuItem(IntPtr menu, uint id, uint enable);

  public static void Main(string[] args)
  {
    Func<IntPtr, uint, uint, int> MessageBox = EnableMenuItem;
    GCHandle gcHandle = GCHandle.Alloc(MessageBox);
    IntPtr inject = GCHandle.ToIntPtr(gcHandle);
    int size = MessageBox.ToString().Length;

    int id = Process.GetProcessesByName(args[0])[0].Id;

    IntPtr process = OpenProcess(0x1F0FFF, false, id);

    IntPtr memory = VirtualAllocEx(process, IntPtr.Zero, size, 0x00001000, 0x40);
    uint written;
    WriteProcessMemory(process, memory, inject, size, out written);

    uint threadId;
    CreateRemoteThread(process, IntPtr.Zero, 0, memory, IntPtr.Zero, 0, out
threadId);

  }
}
```

The instructions for running the proof of concept were very simple:

> As administrator:
> attack.exe svchost
>
> As normal user:
> attack.exe notepad (or any other process name)

As is customary of low-quality reports, the finder provides no explanation of what they're attacking or how the attack works. They just attach a program and say "Good luck figuring out what I did!"[1]

I mean, I like puzzles. But this is not the place for puzzles.

Here's what's going on.

First, the code gets the native address and size of the `MessageBox` function.

Next, they take the program name from the command line and find the process with that ID. (If there's more than one, then they take the first one.)

Once they have the process ID, they use `OpenProcess` to get a handle.

With the process handle, they use `VirtualAllocEx` to allocate (`MEM_COMMIT` = `0x00001000`) read-write-execute data (`PAGE_EXECUTE_READWRITE` = `0x40`) in the victim process, and then copy the `MessageBox` function into the process.

Finally, they inject a thread to execute the injected code.

Is this a security vulnerability?

Notice the first parameter to `OpenProcess`: It is `0x1F0FFF` = `PROCESS_ALL_ACCESS`.[2] If you can get "all access" rights to a process, then you pwn the process, and it's therefore not suprising that you can inject code into it to make it crash.

In fact, if your goal is to crash the process, you don't need to do all this nonsense. `PROCESS_ALL_ACCESS` includes `PROCESS_TERMINATE`, so this entire program could be simplified to

```
public class Program
{
  public static void Main(string[] args)
  {
    System.Diagnostics.Process.
        GetProcessesByName(args[0])[0].Kill();
  }
}
```

or a C# 9 one-liner,

```
System.Diagnostics.Process.GetProcessesByName(args[0])[0].Kill();
```

or avoid having to write any code at all: Run Task Manager, find the `svchost.exe` or `notepad.exe` you want to terminate, and click “End Task”.

Oh, and did you see the sleight of hand?

The report first says that an administrator can terminate any process, and they picked `svchost`. But then when they said that a non-administrator can also terminate any process, and somehow they switch from `svchost` to a lowly `notepad`.

That’s because when they tried having a non-administrator attack `svchost`, it didn’t work.

Somehow conveniently forgot to mention that.

Remember, you’re a researcher, not a student turning in a homework assignment. If you find evidence that runs counter to your hypothesis, you need to take it into consideration, not hide it and hope that nobody notices.

Bonus chatter: There are plenty of other wrong things about this vulnerability report. I’ll leave them as Easter Eggs for you to discover.

[1] I suspect that one of the reasons they don’t explain what their code does, or how the code accomplishes what it claims to do, is that *they don’t know themselves*. They just wrote some code, gosh it acts funny, must be a security vulnerability, send it to Microsoft!

[2] Specifically, it is the value of `PROCESS_ALL_ACCESS` from Windows XP. The value was upgraded in Windows Vista to `0x001FFFFF` to include the “limited information” access bits, but this finder is apparently working from a very old worksheet.