Pulling sleight of hand tricks in a security vulnerability report, or maybe it was a prank

devblogs.microsoft.com/oldnewthing/20200401-00

April 1, 2020

Raymond Chen

It can sometimes be hard to tell the difference between somebody who is playing an elaborate prank and somebody who has no idea what they're doing. (Just like how it can sometimes be hard to tell the difference between a very advanced question and a very naïve one.)

A security vulnerability report came in that claimed that an unprivileged user could extract certain types of information from an object despite not having access to it. The evidence for this alleged vulnerability did not take the form of a document that explained the nature of the problem, however. It came in the form of a video, a silent screen recording that purported to demonstrate the vulnerability.

The video contained no narration. You just had to sit there and watch a twelve-minute silent movie. There was no accompanying description, so it was like watching a magic trick performed by a silent magician, where you see the story unfold before your eyes without knowing where it's going to go next. You just have to sit back and wait for the surprise ending. For a magic trick, the suspense is part of the enjoyment. Not so much for a security investigation.

The video did seem to demonstrate that a user was indeed extracting information from the resource. But at the eight-minute mark, the video shows the user being added to the Administrators group. The user then signs on, opens an elevated command prompt, and runs the alleged exploit. The finder claimed that an unprivileged user could extract the information, but that's not what their demonstration video showed. We asked for clarification.

The finder came back with another video, this one even longer than the previous one. (At least they didn't reply with just a meme GIF.) The second video was still unnarrated, but at least it provided more details, showing each step in painstaking detail. Opening the security dialog to show the permissions on the resource and confirming that the user did not have access. Opening the source code in Notepad and scrolling through it.

```
C:\demo>notepad exploit.cpp
```

```
C:\demo>
```

And then it ran the proof of concept.

```
C:\demo>notepad exploit.cpp
C:\demo>exploit.exe
```

And then in another window, they made changes to the object, and the exploit was able to detect it:

```
C:\demo>notepad exploit.cpp
C:\demo>
2019/04/01 07:00:01: Value is now 3.
2019/04/01 07:00:05: Value is now 1.
2019/04/01 07:00:09: Value is now 4.
```

I tried playing along at home, but could not get the reported results.

So I studied the video more closely, and I observed a sleight-of-hand trick. Maybe you noticed it, too.

Look closely at the contents of the console.

Observe that the command line `exploit.exe` is not present any more.

In the video, the finder types `exploit.exe` into the command line, but instead of hitting `Enter`, they appear to hit `ESC` and erase the line. Despite the fact that `exploit.exe` was not run from the command line, it was nevertheless generating output.

What I suspect happened is that the finder had a copy of `exploit.exe` running *already*. That copy of `exploit.exe` was started while the standard user still had access to the resource. The security check happens at the acquisition of the handle, and once you have a handle, it's yours to keep. So the already-running copy of `exploit.exe` was able to gain access to the resource, and it was able to retain that access even after the ACLs on the resource were changed so that the standard user no longer has access.

It is that already-running copy of `exploit.exe` that was monitoring the resource. Just like how the magician has an identical red ball already hiding under the cup, but fools you into thinking it's the same red ball that vanished just moments ago.

So you tell me: Was this somebody carrying out an elaborate prank or somebody who simply didn't understand what they were doing?

Whether it was intended as such or not, this ended up being an effective denial-of-service attack against me personally, since I ended up spending quite a bit of time watching the videos closely, then reverse-engineering what the finder *believed* the vulnerability to be, and then studying the videos again to find out where they went wrong.

Raymond Chen

Follow