Misdirected security vulnerability: Malformed file results in memory corruption

devblogs.microsoft.com/oldnewthing/20180518-00

May 18, 2018

Raymond Chen

A security vulnerability report arrived that went something like this:

Subject: An Exploitable Vunerability in dwrite.dll (stack hash 0x6b6f6f4c.0x4654d7441)

The file `dwrite.dll` allows an attacker to execute arbitrary code or trigger a denial of service by a crafted corrupted PNG file. This vulnerability can be reproduced by using LitWare Publisher 2.0, clicking *Insert Image* and then selecting the attached PNG file. Debugger output follows:

```
ModLoad: 00060000 0009f000   litware.exe
ModLoad: 77df0000 77f47000   ntdll.dll
ModLoad: 76dd0000 76ea0000   C:\WINDOWS\SysWOW64\KERNEL32.DLL
...
ModLoad: 71e40000 71e59000   C:\Windows\SysWOW64\dwrite.dll

(1788.17ec): Access violation - code c0000005 (first chance)
eax=00000000 ebx=0039d82c ecx=0000000a edx=00000000 esi=006c6608 edi=36363636
eip=767b997e esp=003afb44 ebp=003afb54 iopl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010246
msvcrt!_VEC_memzero+0x6a:
767b997e f3aa rep stos byte ptr es:[edi]

0:004> k
ChildEBP RetAddr
003afb54 74db8b92 msvcrt!_VEC_memzero+0x6a
003afb8c 68f41cbf COMCTL32!Progress_Paint+0x4c
003afba8 68ff6fbb COMCTL32!Progress_UpdatePosition+0x6b
003afbbc 690c3dea COMCTL32!Progress_Update+0x46
003afc48 756377d8 COMCTL32!Progress_WndProc+0x14bfdc
003afc74 756378cb USER32!InternalCallWinProc+0x23
003afcf0 7563f139 USER32!UserCallWinProcCheckWow+0x100
003afd58 75648fce USER32!SendMessageWorker+0x656
003afd80 0006e7db USER32!SendMessageA+0x8b
003afdbc 00080688 litware+0xe7db
003afde0 00092061 litware+0x20688
003affa8 0007ff75 litware+0x32061
003affc4 0008fe12 litware+0x1ff75
003b00ec 00074066 litware+0x2fe12
003b1760 756377d8 litware+0x14066
003b2d40 756378cb USER32!InternalCallWinProc+0x23
003b2d6c 7563f139 USER32!UserCallWinProcCheckWow+0x100
003b2de8 75648fce USER32!SendMessageWorker+0x656
003b2e50 00e949db USER32!SendMessageA+0x8b
003b2e78 00e9546f litware+0x1849db
003b2ec8 0108a800 litware+0x18546f
003b307c 77a7850d litware+0x37a800
003b30c8 77e4bf39 KERNEL32!BaseThreadInitThunk+0xe
003b30d4 77e4bf0c ntdll!__RtlUserThreadStart+0x72
003b3118 00000000 ntdll!_RtlUserThreadStart+0x1b

0:004> !exploitable

Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - User Mode Write AV
    starting at msvcrt!_VEC_memzero+0x000000000000006a
    (stack hash 0x6b6f6f4c.0x4654d7441)

User mode write access violations that are not near NULL are exploitable.
```

Okay, let's look at what we were given. We have a stack trace, where the code is trying to zero out a block of memory, but the address of that block of memory is `0x36363636`, which is invalid and awfully suspicious. This was classified by the `!exploitable` debugger extension as exploitable because the bad address is used control a write.

Great, now let's look at the analysis by the submitter.

The submitter says that the vulnerability is in `dwrite.dll`. It's not clear from the stack trace why `dwrite.dll` is getting the blame. After all, there is no `dwrite` code anywhere on the stack.

The only appearance of `dwrite` in the debugger output is the fact that `dwrite.dll` was the DLL most recently loaded by the process. I guess the submitter decided that this was enough to pin the blame on `dwrite.dll`. "The most recent person to enter the room is responsible for everything bad that happens."

(Note that they quite pointedly took the bug title that the `!exploitable` debugger extension recommended and changed the name of the DLL, while keeping the rest the same, including the stack hash!)

What more likely happened is that loading the malformed PNG file caused LitWare to corrupt some memory that `Progress_Paint` was using, and then when LitWare finally got around to updating the progress control, the `Progress_Paint` function crashed because its pointer to some internal data structure got corrupted.

This is a case of somebody knowing just enough to be dangerous. They found the `!exploitable` debugger extension, but they didn't understand how to use it.

Yes, they found a vulnerability. But they didn't understand how to assign the blame, so they blamed Microsoft!

This issue needs to go to the publishers of LitWare. Their program is the one that is corrupting memory in response to the malformed PNG file.

Bonus chatter: This conclusion took some time to verify, because it's possible that LitWare Publisher uses a component provided by Microsoft (such as the Windows Imaging Component) to decode PNG files, in which case the vulnerability could be in that component rather than in LitWare itself. But we were able to confirm that the memory corruption was coming from LitWare's own custom PNG decoder.

This last step is important, because if you had simply redirected the report without investigating it, and the problem really was in, say, the Windows Imaging Component, then you are in the unenviable position of having been informed of a security vulnerability and

blown it off. And then when the vulnerability makes the front page of the news, everybody's going to say, "What a bunch of idiots over there at Microsoft. They received the vulnerability report months ago and rejected it as not their bug!"

Bonus chatter: Turns out that the finder was using an old version of LitWare. The problem doesn't exist in the latest version of LitWare, so presumbly LitWare already found and fixed the problem on their side.

Raymond Chen

Follow