Microsoft Money crashes during import of account transactions or when changing a payee of a downloaded transaction

devblogs.microsoft.com/oldnewthing/20121113-00

November 13, 2012

Raymond Chen

> **Update**: An official fix for this issue has been released to Windows Update, although I must say that I think my patch has more style than the official one. You do not need to patch your binary. Just keep your copy of Windows 8 up to date and you'll be fine.

For the five remaining Microsoft Money holdouts (meekly raises hand), here's a patch for a crashing bug during import of account transactions or when changing a payee of a downloaded transaction in Microsoft Money Sunset Deluxe. Patch the mnyob99.dll file as follows:

- File offset 003FACE8: Change 85 to 8D
- File offset 003FACED: Change 50 to 51
- File offset 003FACF0: Change FF to 85
- File offset 003FACF6: Change E8 to B9

Note that this patch is **completely unsupported**. If it makes your computer explode or transfers all your money to an account in the Cayman Islands, well, too bad for you.

If you are not one of the five remaining customers of Microsoft Money, this is a little exercise in application compatibility debugging. Why application compatibility debugging? Because the problem seems to be more prevalent on Windows 8 machines.

Note that I used no special knowledge about Microsoft Money. All this debugging was performed with information you also have access to. It's not like I have access to the Microsoft Money source code. And I did this debugging entirely on my own. It was not part of any official customer support case or anything like that. I was just debugging a crash that I kept hitting.

The crash occurs in the function `utlsrf08!DwStringLengthA`:

```
utlsrf08!DwStringLengthA:
        push    ebp
        mov     ebp,esp
        mov     eax,dword ptr [ebp+8]
        lea     edx,[eax+1]
again:
        mov     cl,byte ptr [eax]
        inc     eax
        test    cl,cl
        jne     again
        sub     eax,edx
        pop     ebp
        ret     4
```

The proximate cause is that the string pointer in `eax` is garbage. If you unwind the stack one step, you'll see that the pointer came from here:

```
        lea     eax,[ebp-20Ch]
        push    eax
        call    dword ptr [__imp__GetCurrentProcessId]
        push    eax
        push    offset "Global\TRIE@%d!%s"
        lea     eax,[ebp-108h]
        push    104h
        push    eax
        call    mnyob99!DwStringFormatA
        add     esp,14h
        lea     eax,[ebp-2E4h]
        push    eax
        push    5Ch
        push    dword ptr [ebp-2E4h] ; invalid pointer
        call    mnyob99!DwStringLengthA
        sub     eax,7
        push    eax
        lea     eax,[ebp-101h]
        push    eax
        jmp     l2
l1:
        mov     eax,dword ptr [ebp-2E4h]
        mov     byte ptr [eax],5Fh
        lea     eax,[ebp-2E4h]
        push    eax
        push    5Ch
        push    dword ptr [ebp-2E4h]
        call    mnyob99!DwStringLengthA
        push    eax
        push    dword ptr [ebp-2E4h]
l2:
        call    mnyob99!FStringFindCharacterA
        cmp     dword ptr [ebp-2E4h],edi
        jne     l1
```

I was lucky in that all the function calls here were to imported functions, so I could extract the names from the imported function table. For example, the call to `DwStringFormatA` was originally

```
        call    mnyob99!CBillContextMenu::SetHwndNotifyOnGoto+0x1e56a (243fc3cc)
```

But the target address is an import stub:

```
        jmp     dword ptr [mnyob99+0x1ec0 (24001ec0)]
```

And then I can walk the import table to see that this was the import table entry for `utlsrf08!DwStringFormatA`. From the function name, it's evident that this is some sort of `sprintf`-like function. (If you disassemble it, you'll see that it's basically a wrapper around `vsnprintf`.)

Reverse-compiling this code, we get

```
char name[...];
char buffer[MAX_PATH];
char *backslash;
...
DwStringFormatA(buffer, MAX_PATH, "Global\\TRIE@%d!%s",
                GetCurrentProcessId(), name);
// Change all backslashes (except for the first one) to underscores
if (FStringFindCharacterA(buffer + 7, DwStringLengthA(backslash) - 7,
                          '\\',&backslash))
{
  do {
    *backslash = '_'; // Change backslash to underscore
  } while (FStringFindCharacterA(backslash, DwStringLengthA(backslash),
                                 '\\',&backslash));
}
```

(Remember, all variable names are made-up since I don't have source code access. I'm just working from the disassembly.)

At this point, you can see the bug: It's an uninitialized variable at the first call to `String-FindCharacterA`. Whether we crash or survive is a matter of luck. If the uninitialized variable happens to be a pointer to readable data, then the `DwStringLengthA` will eventually find the null terminator, and since in practice the string does not contain any extra backslashes, the call to `FStringFindCharacterA` fails, and nobody gets hurt.

But it looks like their luck ran out, and now the uninitialized variable contains something that is not a valid pointer.

The `if` test should have been

```
if (FStringFindCharacterA(buffer + 7, DwStringLengthA(buffer) - 7,
                          '\\',&backslash))
```

This means changing the

```
        push    dword ptr [ebp-2E4h]
```

to

```
        lea     eax,[ebp-101h]
        push    eax
```

Unfortunately, the patch is one byte larger than the existing code, so we will need to get a little clever in order to get it to fit.

One trick is to rewrite the test as

```
if (FStringFindCharacterA(buffer + 7, DwStringLengthA(buffer + 7),
                          '\\',&backslash))
```

That lets us rewrite the assembly code as

```
        lea     eax,[ebp-2E4h]
        push    eax
        push    5Ch
        lea     eax,[ebp-101h]          ; \ was "push dword ptr [ebp-2E4h]"
        push    eax                     ; /
        call    mnyob99!DwStringLengthA ; unchanged but code moved down one byte
        nop                             ; \ was "sub eax,7" (3-byte instruction)
        nop                             ; /
        push    eax
        lea     eax,[ebp-101h]
        push    eax
```

The new instructions (`lea` and `push`) are one byte larger than the original `push` , but we got rid of the three-byte `sub eax, 7` , so it's a net savings of two bytes, which therefore fits.

However, I'm going to crank the nerd level up another notch and try to come up with a patch that involves modifying as few bytes as possible. In other words, I'm going for *style points*.

To do this, I'm going to take advantage of the fact that the string length is the return value of `DwStringFormatA` , so that lets me eliminate the call to `DwStringLengthA` altogether. However, this means that I have to be careful not to damage the value in `eax` before I get there.

```
lea     ecx,[ebp-2E4h] ; was "lea eax,[ebp-2E4h]"
push    ecx            ; was "push eax"
push    5Ch
nop                    ; \
nop                    ; |
nop                    ; |
nop                    ; | was "push dword ptr [ebp-2E4h]"
nop                    ; |
nop                    ; /
nop                    ; \
nop                    ; |
nop                    ; | was "call mnyob99!DwStringLengthA"
nop                    ; |
nop                    ; /
sub     eax,7
push    eax
lea     eax,[ebp-101h]
push    eax
```

Patching the `lea eax, …` to be `lea ecx, ...` can be done with a single byte, and the `push eax` is a single-byte instruction as well, so the first two patches can be done with one byte each. That leaves me with 11 bytes that need to be nop'd out.

The naïve way of nopping out eleven bytes is simply to patch in 11 `nop` instructions, but you can do better by taking advantage of the bytes that are already there.

```
ffb51cfdffff    push    dword ptr [ebp-2E4h]
85b51cfdffff    test    dword ptr [ebp-2E4h],esi
e8770a0000      call    mnyob99!DwStringLengthA
b9770a0000      mov     ecx,0A77h
```

By patching a single byte in each of the two instructions, I can turn them into effective nops by making them do nothing interesting. The first one tests the uninitialized variable against some garbage bits, and the second one loads a unused register with a constant. (Since the `ecx` register is going to be trashed by the call to `FStringFindCharacterA`, we are free to modify it all we want prior to the call. No code could have relied on it anyway.)

That second patch is a variation of one I called out some time ago, except that instead of patching out the call with a `mov eax, immed32`, we're using a `mov ecx, immed32`, because the value in the `eax` register is still important.

Here's the final result:

```
lea     ecx,[ebp-2E4h]           ; was "lea eax,[ebp-2E4h]"
push    ecx                      ; was "push eax"
push    5Ch
test    dword ptr [ebp-2E4h],esi ; was "push dword ptr [ebp-2E4h]"
mov     ecx,0a77h                ; was "call mnyob99!DwStringLengthA"
sub     eax,7
push    eax
lea     eax,[ebp-101h]
push    eax
```

Bonus chatter: When I shared this patch with my friends, I mentioned that this patch made me feel like my retired colleague Jeff, who had a reputation for accomplishing astonishing programming tasks in his spare time. You would pop into his office asking for some help, and he'd fire up some program you'd never seen before.

"What's that?" you'd ask.

"Oh, it's a debugger I wrote," he'd calmly reply.

Or you'd point him to a program and apologize, "Sorry, I only compiled it for x86. There isn't an Alpha version."

"That's okay, I'll run it in my emulator," he'd say, matter-of-factly.

(And retiring from Microsoft hasn't slowed him down. Here's an IBM PC Model 5150 emulator written in JavaScript.)

Specifically, I said, "I feel like Jeff, who does this sort of thing before his morning coffee."

Jeff corrected me. "If this was something I used to do before coffee, that probably meant I was up all night. Persistence >= talent."

Raymond Chen

Follow