

[Exploit Shop](#)

1-day vulnerability analysis using DarunGrim

- [About](#)

[MS11-038 : Vulnerability in OLE Automation Could Allow Remote Code Execution \(2476490\)](#) →

MS11-077: Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Remote Code Execution (2567053)

Posted: 2011/10/12 | Author: [lifeasageek](#) | Filed under: [Uncategorized](#) | [Leave a comment >](#)

Download MS11-077 .fon buffer overrun exploit : [my.fon.tar.gz](#)

Download very simple *.fon* fuzzer like tool : [ms11-077-fon-exploit.tar.gz](#)

Related CVEs

Font Library File Buffer Overrun Vulnerability – CVE-2011-2003

Diffing Binary Information

win32k.sys vs win32k.dll: 6.1.7601.21744 (win7sp1_ldr.110610-1504) VS 6.1.7601.21811 (win7sp1_ldr.110905-1505) (on Windows 7, 32bit)

Descriptions

This posting is no technical analysis, but it is driven by hardcore & intuition based analysis to make 1-day exploit.

MS11-077 was confusing at the first time. Because it involves 4 different vulnerabilities, we should try to match up these vulnerabilities whenever we reverse engineer the function. This time I will not show the DarunGrim diffing results cause it showed around 50 different functions! Don't get frustrated though. It's not going to take that long time to take a look all of them. Within 10 secs for each of the function, you might be able to decide whether the function is interesting or not.

Before getting to the details, you may also look into these. Three functions seem to be related to the null dereference bugs (`_NtUserfnINLBOXSTRING()`, `_NtUserfnSENTDDEMSG()`, `_InterQueueMsgCleanup()`). The function, `_ConvertToAndFromWideChar()`, seem to be related to "Win32k Use After Free Vulnerability – CVE-2011-2011". You must be able to understand what I am meaning by here as soon as you open up these functions with DarunGrim.

What I want to focus in this post is .FON buffer overrun bug (CVE-2011-2003). From Dar

diffing result, `_BmfdOpenFontContext()` showed the different point below.

<pre>[BF838EA9L] mov eax, [ebp+var_8] add eax, 7 shr eax, 3 mov [ebp+var_4], ecx cmp eax, 100h jbe short loc_BF838EC8</pre>	<pre>[BF838EE4L] mov eax, [ebp+var_8] add eax, 7 shr eax, 3 add eax, 5 mov [ebp+var_4], ecx cmp eax, 100h jbe short loc_BF838F06</pre>
---	--

What ??? Patched version only adds immediate value '5' to some value (`add eax, 5`), and that computed value is related to decide the size of allocation. Seems interesting but strange. It is time to see the details to understand the contexts. Here goes the disassembly around the changed BB of the old `win32k.sys`.

```
.text:90857F82 loc_90857F82: ; CODE XREF: BmfdOpenFontContext(x)+E2□j
.text:90857F82 mov eax, [ebp+numElement]
.text:90857F85 add eax, 7
.text:90857F88 shr eax, 3
.text:90857F8B mov [ebp+var_4], ecx
.text:90857F8E cmp eax, 100h
.text:90857F93 jbe short loc_90857FA1
.text:90857F95 add eax, 28h
.text:90857F98 mov [ebp+var_4], 3
.text:90857F9F jmp short loc_90857FA4
.text:90857FA1
.text:90857FA1 loc_90857FA1: ; CODE XREF: BmfdOpenFontContext(x)+E7□j
.text:90857FA1 ; BmfdOpenFontContext(x)+FA□j
.text:90857FA1 mov eax, [ebp+preDefinedSize]
.text:90857FA4
.text:90857FA4 loc_90857FA4: ; CODE XREF: BmfdOpenFontContext(x)+106□j
.text:90857FA4 push 64666D42h ; Tag
.text:90857FA9 push eax ; int
.text:90857FAA push 0 ; char
.text:90857FAC call _EngAllocMem@12 ; EngAllocMem(x,x,x)
```

This is the pseudo-code of these assembly. The variable naming was done at my convenience.

```
uint preDefinedSize = 0x28; // mov dword ptr [ebp-14h], 28h
```

Follow

```

sizeToAllocate = (numElement + 7) / 8;

if( sizeToAllocate <= 0x100)
sizeToAllocate = preDefinedSize;
else
sizeToAllocate += 0x28;

EngAllocMem(0, sizeToAllocate, 0x64666d42);

```

All right. In the patched version, the sizeToAllocate variable would be computed as “((numElement + 7) / 8) + 5”. After spending some time, we suspected some range of the values, which should have taken ‘else’ branch, mistakenly took ‘then’ branch. Because it took ‘then’ branch, the allocated size was too small and this small size of allocation would lead to buffer overrun later (We understand this interpretation is far from scientific or logical reverse engineering, but you should know that this sloppy logic is enough to write an 1-day exploit.)

More specifically, we suspected the numElement values, satisfying $0xaa \leq (\text{numElement} + 7) / 8 \leq 0x100$, would cause trouble (though we don’t know why and how !). We got this false fail idea in the patched binary from D. Brumeley et al.’s paper, “Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications” (<http://www.cs.berkeley.edu/~dawnsong/papers/apeg.pdf>).

Things are getting clear. Our goal should be to find the input which satisfies the above statement. From the MS technet description, “improper handling of a specially crafted .fon font file”, and the function name `_BmfdOpenFontContext()`, which implies bitmap font driver something, we decided to manipulate .fon file. To play with .fon files, we implemented very simple ‘.fon’ file format recognizing fuzzer like tool. Using this tool, we figured ‘width’ field is related (see our *fuzzer* for details) to control numElement variable, and it leads to ‘heap overflow’ when the variable satisfies the vulnerable condition. What’s the interesting is that you only need to visit the directly containing .fon file to trigger bitmap font driver routines 😊

I am attaching the .fon font file generated by our python codes (upon mkwinfont by Simon Tatham) and windbg crash dumps. We are not sure this bug can actually be used to execute the arbitrary codes, but we’d like to leave this question to you guys.

Download MS11-077 .fon buffer overrun exploit : my.fon.tar.gz

Download very simple *.fon* fuzzer like tool : ms11-077-fon-exploit.tar.gz

```

Breakpoint 1 hit
win32k!BmfdOpenFontContext+0xec:
90857f85 83c007 add eax,7
kd> r
eax=00000730 ebx=fe9aacf0 ecx=00000001 edx=00000001 esi=00000028 edi=fe7fc1f8
eip=90857f85 esp=8a2af8d0 ebp=8a2af904 iopl=0 nv up ei pl nz na pe nc
cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00000206
win32k!BmfdOpenFontContext+0xec:
90857f85 83c007 add eax,7
kd> r eax

```

Follow

```

eax=00000730
kd> p
win32k!BmfdOpenFontContext+0xef:
90857f88 c1e803 shr eax,3
kd> p
win32k!BmfdOpenFontContext+0xf2:
90857f8b 894dfc mov dword ptr [ebp-4],ecx
kd> r eax
eax=000000e6
kd> g

```

```

*** Fatal System Error: 0x00000019
(0x00000020,0xFE1ED440,0xFE1ED5A0,0x4A2C000C)

```

Break instruction exception – code 80000003 (first chance)

A fatal system error has occurred.
 Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

Connected to Windows 7 7600 x86 compatible target at (Wed Oct 12 18:38:42.012 2011 (UTC – 4:00)), ptr64 FALSE

Loading Kernel Symbols

```

.....
.....
.....

```

Loading User Symbols

```

.....

```

Loading unloaded module list

```

.....

```

```

*****

```

```

* *

```

* Bugcheck Analysis *

```

* *

```

```

*****

```

Use !analyze -v to get detailed debugging information.

BugCheck 19, {20, fe1ed440, fe1ed5a0, 4a2c000c}

Probably caused by : win32k.sys (win32k!EngFreeMem+1f)

Followup: MachineOwner

```

_ _ _

```

nt!RtlpBreakWithStatusInstruction:

Follow

828be394 cc int 3

kd> !analyze -v

* *

* Bugcheck Analysis *

* *

BAD_POOL_HEADER (19)

The pool is already corrupt at the time of the current request.

This may or may not be due to the caller.

The internal pool links must be walked to figure out a possible cause of the problem, and then special pool applied to the suspect tags or the driver verifier to a suspect driver.

Arguments:

Arg1: 00000020, a pool block header size is corrupt.

Arg2: fe1ed440, The pool entry we were looking for within the page.

Arg3: fe1ed5a0, The next pool entry.

Arg4: 4a2c000c, (reserved)

Debugging Details:

BUGCHECK_STR: 0x19_20

POOL_ADDRESS: fe1ed440 Paged session pool

DEFAULT_BUCKET_ID: VISTA_DRIVER_FAULT

PROCESS_NAME: csrss.exe

CURRENT_IRQL: 2

LAST_CONTROL_TRANSFER: from 8292fe71 to 828be394

STACK_TEXT:

8a2af3b4 8292fe71 00000003 dda0d4a7 00000065 nt!RtlpBreakWithStatusInstruction
8a2af404 8293096d 00000003 fe1ed440 000001ff nt!KiBugCheckDebugBreak+0x1c
8a2af7c8 829721b6 00000019 00000020 fe1ed440 nt!KeBugCheck2+0x68b
8a2af844 9088c189 fe1ed448 00000000 fe7fc1d8 nt!ExFreePoolWithTag+0x1b1
8a2af858 90950204 fe1ed458 90959cdf fe40f480 win32k!EngFreeMem+0x1f
8a2af86c 90959cf5 fe1ed458 8a2af8d8 8a2af8b4 win32k!BmfdCloseFontContext+0x41
8a2af87c 90965501 fe40f480 00000000 8a2af930 win32k!BmfdDestroyFont+0x16
8a2af8b4 90965554 fe40f480 00000000 8a2afc70 win32k!PDEVOBJ::DestroyFont+0x67
8a2af8e4 908d0d1e 00000000 8a2af910 00000001 win32k!RFONTOBJ::vDeleteRFONT+0x33
8a2af928 908d2d15 fe40f480 050a071e 8a2afc70 win32k!RFONTOBJ::bMakeInactiveHelper+0x25a
8a2af984 908fba77 00000000 8a2afc70 00000000 win32k!RFONTOBJ::vMakeInactive+0

Follow

```
8a2afa04 908fbd74 8a2afc3c 00000000 00000004 win32k!RFONTOBJ::bInit+0xe3
8a2afa1c 908a4b2b 8a2afc3c 00000000 00000004 win32k!RFONTOBJ::vInit+0x16
8a2afcb8 908a4a2f 69010742 00000340 00000040 win32k!GreGetCharABCWidthsW+0x86
8a2afd14 8289642a 69010742 00000340 00000040 win32k!NtGdiGetCharABCWidthsW+0xf8
8a2afd14 76f864f4 69010742 00000340 00000040 nt!KiFastCallEntry+0x12a
0435e9ac 00000000 00000000 00000000 00000000 ntdll!KiFastSystemCallRet
```

STACK_COMMAND: kb

FOLLOWUP_IP:

win32k!EngFreeMem+1f

9088c189 5e pop esi

SYMBOL_STACK_INDEX: 4

SYMBOL_NAME: win32k!EngFreeMem+1f

FOLLOWUP_NAME: MachineOwner

MODULE_NAME: win32k

IMAGE_NAME: win32k.sys

DEBUG_FLR_IMAGE_TIMESTAMP: 4a5bc2a2

FAILURE_BUCKET_ID: 0x19_20_win32k!EngFreeMem+1f

BUCKET_ID: 0x19_20_win32k!EngFreeMem+1f

Followup: MachineOwner

— — —

Share this:

[Twitter](#)

[Facebook](#)

Like this:

[Like](#)

Be the first to like this post.

Leave a Reply

[Follow](#)

Enter your comment here...

Guest

Log In

Log In

Log In



Email (required)

(Not published)

Name (required)

Website

Notify me of follow-up comments via email.

Notify me of new posts via email.

Post Comment

[MS11-038 : Vulnerability in OLE Automation Could Allow Remote Code Execution \(2476490\)](#) →

Recent Posts

- [MS11-077: Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Remote Code Execution \(2567053\)](#)
- [MS11-038 : Vulnerability in OLE Automation Could Allow Remote Code Execution \(2476490\)](#)
- [MS11-064 : Vulnerabilities in TCP/IP Stack Could Allow Denial of Service \(2563894\)](#)

Archives

- [October 2011](#)
- [September 2011](#)

Categories

- [Uncategorized](#)

Links

- [DarunGrim](#)

Blog at [WordPress.com](#). Theme: [Clean Home](#) by [Mid Mo Design](#).

Follow

☺

Follow