

Analysis of CVE-2016-4203 - Adobe Acrobat and Reader CoolType Handling Heap Overflow Vulnerability

(/2016/07/20/analysis-of-cve-2016-4203-adobe-acrobat-and-reader-cooltype-handling-heap-overflow-vulnerability)

by  **Kai Lu** | Jul 20, 2016 | Filed in: Security Research (/category/security-research)

Summary

Recently, Adobe patched some security vulnerabilities (<https://helpx.adobe.com/security/products/acrobat/apsb16-26.html>) **in Adobe Acrobat and Reader. One of them is a heap buffer overflow vulnerability (CVE-2016-4203) I recently discovered. In this blog, we want to share our analysis of this vulnerability.**

Proof of Concept

This vulnerability can be reproduced by opening the PoC file "poc_minimized.pdf" with Adobe Reader DC. When opened, AcroRd32.exe crashes, and the crash information is shown below:

(8de0,6bc4): Access violation - code c0000005 (first chance)

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

eax=097eeee ebx=00000015 ecx=097ef6e0 edx=0000cc6c esi=2952d000 edi=00000024

eip=0959a23c esp=2911e5f8 ebp=2911e608 iopl=0 nv up ei pl nz na po nc

cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010202

CoolType!CTInit+0x45ef1:

0959a23c 0fb606 movzx eax,byte ptr [esi] ds:002b:2952d000=??

0:022> kb

ChildEBP RetAddr Args to Child

WARNING: Stack unwind information not available. Following frames may be wrong.

00 2911e608 09598db1 097eee15 097ef564 097ef44c CoolType!CTInit+0x45ef1

01 2911e674 095939a0 17e9cc38 17e9cd98 17e9cd58 CoolType!CTInit+0x44a66

02 2911e6d0 095935c3 17e9cc38 17e9cd98 17e9cd58 CoolType!CTInit+0x3f655

03 2911e720 0958d9a3 17e9cc38 17e9cd98 17e9cd58 CoolType!CTInit+0x3f278

04 2911e784 0958d79c 00000001 2911e7d8 00000001 CoolType!CTInit+0x39658

05 2911e79c 095a6cb0 2911e8a8 2911e7d8 1a99cf68 CoolType!CTInit+0x39451

06 2911e8fc 095a6996 1a99cf68 097f61a8 2911ea88 CoolType!CTInit+0x52965

07 2911eac4 095a614e 2911ecb8 00000000 097f6480 CoolType!CTInit+0x5264b

08 2911eb90 095a506f 773dfa00 00000000 00000001 CoolType!CTInit+0x51e03

09 2911ef58 095a468a 00000025 20f02fec 00001088 CoolType!CTInit+0x50d24

0a 2911ef98 095a3691 20f02fe0 00000002 2911f028 CoolType!CTInit+0x5033f

0b 2911f100 095a32c7 2911f518 2911f8ac 0000044a CoolType!CTInit+0x4f346

0c 2911f150 0908a44c 145aae2c 2911f518 2911f8ac CoolType!CTInit+0x4ef7c

0d 2911f258 0906bab0 2911f34c 00000000 21d6629c AGM!AGMInitialize+0x51bf0

0e 2911f268 0906b98f 00000000 f7510c27 20d6cf70 AGM!AGMInitialize+0x33254

```

0f 2911f280 0906ba9c 00000081 21cb6d00 21cb6d00 AGM!AGMInitialize+0x33133

10 2911f9ac 0906182e 090004ca 00001fa0 2911fa01 AGM!AGMInitialize+0x33240

11 2911f9bc 09080a4d 21d76fe8 00000053 00000000 AGM!AGMInitialize+0x28fd2

12 2911fa01 00000000 c8000000 5021ca6f 0027c3d2 AGM!AGMInitialize+0x481f1


0:022> !heap -p -a esi

address 2952d000 found in

_DPH_HEAP_ROOT @ 3d01000

in busy allocation ( DPH_HEAP_BLOCK:      UserAddr      UserSize -      VirtAddr      VirtSize)

                        293c3924:      2952cf40      c0 -      2952c000      2000

6a749abc verifier!AVrfDebugPageHeapAllocate+0x0000023c

7749d836 ntdll!RtlDebugAllocateHeap+0x0000003c

773ffb40 ntdll!RtlpAllocateHeap+0x000000f0

773fdecb ntdll!RtlpAllocateHeapInternal+0x0000027b

773fdc2e ntdll!RtlAllocateHeap+0x0000002e

6854ed63 MSVCR120!malloc+0x00000049 [f:\dd\vctools\crt\crtw32\heap\malloc.c @ 92]

095550fc CoolType!CTInit+0x00000db1

095589db CoolType!CTInit+0x00004690

...

```

Analysis

This vulnerability is an instance of a heap buffer overflow vulnerability. Let's look into this specially crafted PDF file first. The comparison between the normal PDF file and the minimized PoC file is shown below.

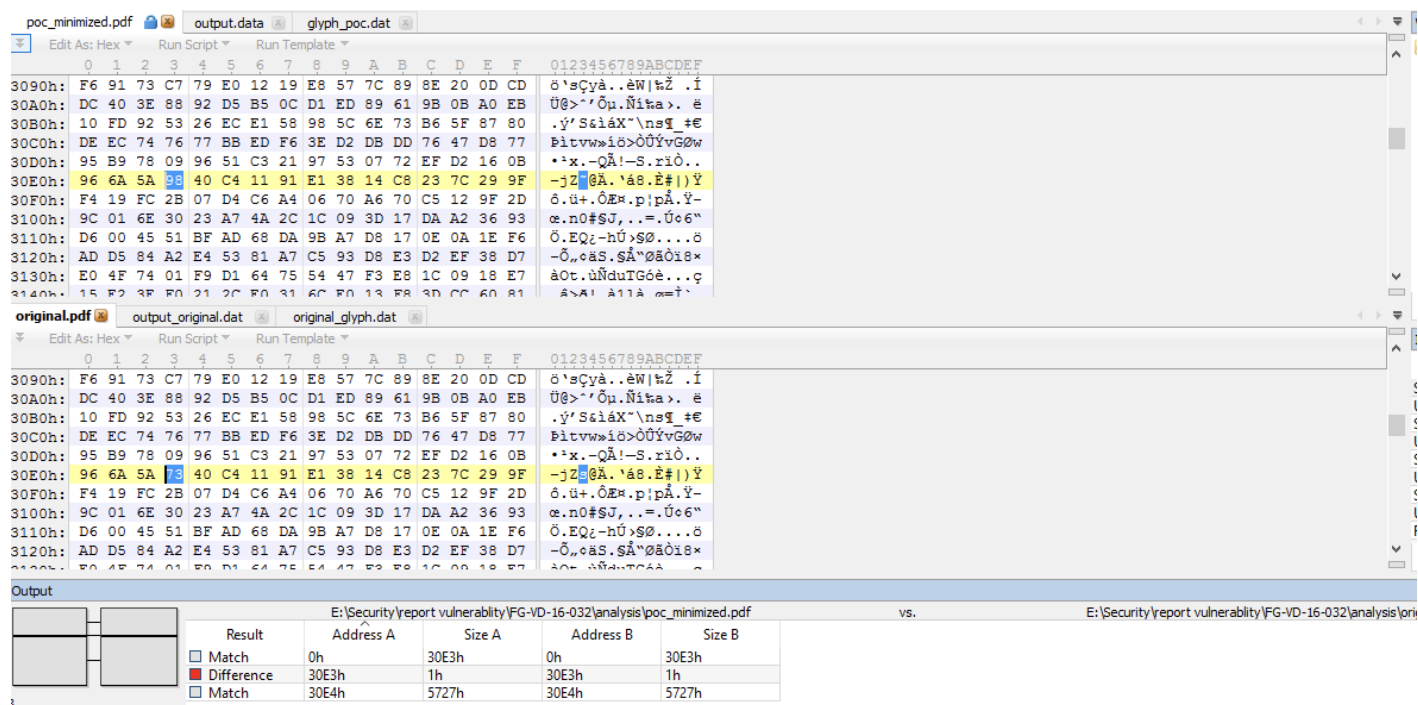


Figure 1. The PoC File vs The Original PDF File

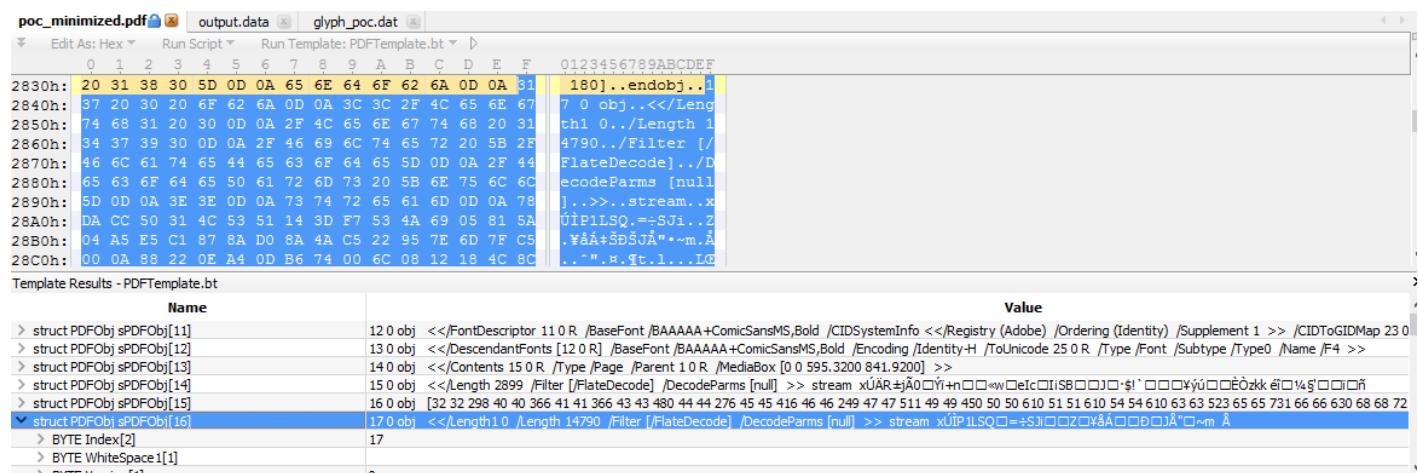


Figure 2. Parsing of The PoC File with 010 Editor

From Figures 1 and 2 we can see the only difference between the original PDF file and the PoC file is a single byte at offset 0x30e3, located in obj 17. The object structure is shown below.

```
17 0 obj
<
/Length 14790
/Filter [/FlateDecode]
/DecodeParms [null]
>>
stream
...
endstream
endobj
```

The question is, what type of data does this object store? We found that the following object in the PDF file references obj 17:

```
8 0 obj
<
/FontFile2 17 0 R
/FontName /AAAAAA+ComicSansMS
/Flags 32
/ItalicAngle 0
/Ascent 1102
/Descent -312
/CapHeight 0
/StemV 0
/Type /FontDescriptor
/CIDSet 19 0 R
>>
endobj
```

From the field “/FontFile2 17 0 R”, we know that obj 17 stores a font file. The data of the font file is encoded with FlateDecode in obj 17. For more information on deflate, please refer to <https://tools.ietf.org/html/rfc1951>. (<https://tools.ietf.org/html/rfc1951>.)

Zlib is a C library which implements the deflate algorithm. We can use it to decompress the data in obj 17. The decompressed data of obj 17 in the PoC file is stored in the file output.dat, which has a length of 0x4650. The decompressed data of obj 17 in the original PDF file is stored in the file output_original.dat which also has a length of 0x4650. The following is the comparison between output.dat and output_original.dat:

The screenshot shows a hex editor with two panes: 'output_original.dat' and 'output.dat'. Both panes display hex and ASCII data for a font file. The comparison table at the bottom shows the following results:

Result	Address A	Size A	Address B	Size B
Match	0h	CF5h	0h	CF5h
Difference	CF5h	11h	CF5h	11h
Match	D06h	88h	D06h	88h
Difference	D8Eh	5h	D8Eh	5h
Match	D93h	AC	D93h	AC
Difference	E3Fh	4h	E3Fh	4h
Match	E43h	116h	E43h	116h
Difference	F59h	3h	F59h	3h
Match	F5Ch	2h	F5Ch	2h

Selected: 3317 [CF5h] bytes (Range: 0 [0h] to 3316 [CF4h]) Start: 0 [0h] Sel: 3317 [CF5h] Size: 18000

Figure 3. The PoC Font File (output.dat) vs The Original Font File (output_original.dat)

We can see that about one hundred bytes are different. The next question, then, is to figure out what data triggers this vulnerability. Some debugging skills are needed. Let's go back to Windbg first.

```

0:022> r

eax=097eeee ebx=00000015 ecx=097ef6e0 edx=0000cc6c esi=2952d000 edi=00000024

eip=0959a23c esp=2911e5f8 ebp=2911e608 iopl=0         nv up ei pl nz na po nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202

CoolType!CTInit+0x45ef1:

0959a23c 0fb606          movzx  eax,byte ptr [esi]      ds:002b:2952d000=??


0:022> !heap -p -a esi

address 2952d000 found in

_DPH_HEAP_ROOT @ 3d01000

in busy allocation ( DPH_HEAP_BLOCK:      UserAddr      UserSize -      VirtAddr      VirtSize)

                        293c3924:      2952cf40          c0 -      2952c000          2000

6a749abc verified!AVrfDebugPageHeapAllocate+0x0000023c

7749d836 ntdll!RtlDebugAllocateHeap+0x0000003c

773ffb40 ntdll!RtlpAllocateHeap+0x000000f0

773fdecb ntdll!RtlpAllocateHeapInternal+0x0000027b

773fdc2e ntdll!RtlAllocateHeap+0x0000002e

6854ed63 MSVCR120!malloc+0x00000049 [f:\dd\vctools\src\crtw32\heap\malloc.c @ 92]

095550fc CoolType!CTInit+0x00000db1

095589db CoolType!CTInit+0x00004690

...


0:022> db 2952cf40 Lc0

2952cf40  00 02 00 b3 ff fb 01 bf-05 da 00 0b 00 23 00 55 .....#.U

2952cf50  40 36 06 38 00 1e 12 1e-06 20 25 30 25 50 25 c0 @6.8..... %0%P%.

2952cf60  25 04 80 25 90 25 a0 25-03 09 38 03 18 15 1b 18 %..%..%..8.....

2952cf70  38 0f 21 1f 25 2f 25 3f-25 7f 25 04 1f 0c bf 0c 8.!.%/%?%.%.....

2952cf80  02 50 0c 7f 0c 02 0c 25-10 d6 5d 71 5d 7d c4 c4 .P.....%.jq]]..

2952cf90  18 fd 7d c4 c4 18 10 7d-d4 18 ed 5d 71 00 3f 2f ..}....}...jq.?!

```

```

2952cfa0 10 d6 ed 31 30 01 22 26-35 34 36 33 32 1f 00 ed ...10."&54632...

2952cfb0 02 03 14 16 00 ed 06 23-22 26 35 34 26 35 34 12 .....#"&54&54.

2952cfc0 35 34 36 33 32 1f 00 ed-02 01 4a 30 46 46 30 30 54632.....J0FF00

2952cfd0 45 45 03 07 36 2c 2b 37-07 14 37 2b 2b 37 14 04 EE..6,+7..7++7..

2952cfe0 f2 44 30 30 44 44 30 30-44 fc d4 3c ef 3c 2c 38 .D00DD00D..<.<,8

2952cff0 38 2c 3c ef 3c 5e 01 19-5e 2d 38 38 2d 5e fe e7 8,<.<^..^~88-^..

```

From the above output, an out of bounds memory access is triggered when handling a heap buffer with size 0xc0 and starting at 0x2952cf40. Next, we search some data from this heap buffer, like [00 02 00 b3] in output.dat.

output.dat original_glyph.dat output_original.dat																
Edit As: Hex Run Script Run Template																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2CD0h:	14	0E	14	04	3C	3E	4C	52	97	91	5D	1A	6A	1A	BC	73
2CE0h:	3B	A6	FE	83	46	3E	38	28	0E	1E	19	79	94	02	DF	7C
2CF0h:	2B	33	5B	31	2A	37	58	61	4E	69	72	6D	6E	58	6E	6D
2D00h:	6C	4C	B4	BF	D0	75	59	11	0C	29	37	00	02	00	B3	FF
2D10h:	FB	01	BF	05	DA	00	0B	00	23	00	55	40	36	06	38	00
2D20h:	1E	12	1E	06	20	25	30	25	50	25	C0	25	04	80	25	90
2D30h:	25	A0	25	03	09	38	03	18	15	1B	18	38	0F	21	1F	25
2D40h:	2F	25	3F	25	7F	25	04	1F	0C	BF	0C	02	50	0C	7F	0C
2D50h:	02	0C	25	10	D6	5D	71	5D	7D	C4	C4	18	FD	7D	C4	C4
2D60h:	18	10	7D	D4	18	ED	5D	71	00	3F	2F	10	D6	ED	31	30
2D70h:	01	22	26	35	34	36	33	32	1F	00	ED	02	03	14	16	00
2D80h:	ED	06	23	22	26	35	34	26	35	34	12	35	34	36	33	32
2D90h:	1F	00	ED	02	01	4A	30	46	46	30	30	45	45	03	07	36
2DA0h:	2C	2B	37	07	14	37	2B	2B	37	14	04	F2	44	30	30	44
2DB0h:	44	30	30	44	FC	D4	3C	EF	3C	2C	38	38	2C	3C	EF	3C
2DC0h:	5E	01	19	5E	2D	38	38	2D	5E	FE	E7	00	02	FF	EE	FD
2DD0h:	AB	02	90	05	D8	00	0B	00	2B	00	53	40	38	50	24	60
0123456789ABCDEF																
...<>LR~\].j..4s																
; pfF>8(...y".B																
+3[1*7XaNirmnXnm																
1L'zDuY..)7...Y																
û.ç.Û...#.U@6.8.																
... %0\$P\$A\$.e%.																
% %..8.....8.!.%																
/%?%.%...ç..P...																
..%.Ô]q]}AA.ý)AA																
..)Ô.i]q.?.Ôi10																
..%54632..i.....																
i.#"%54654.54632																
..i...J0FF00EE..6																
,+7..7++7..ôD00D																
D00DûÔ<i<,88,<i<																
^..^~88-^pç..ýiý																
«...ø...+.S@8P\$`																

Copied: 4 bytes

Start: 11531 [2D0Bh]

Sel: 4 [4h]

Figure 4. The Result of Data Searching in output.dat

We find it at offset 0x2d0b, and it only appears once. We then use TTFTemplate.bt in 010 Editor to parse the font file, and find [00 02 00 B3] is in the 'glyph' table. Next, we extract the data with size 0xc0 in the 'glyph' table from output.dat and output_original.dat respectively, and save them as glyph_poc.dat and glyph_original.dat. The following is the comparison between them.

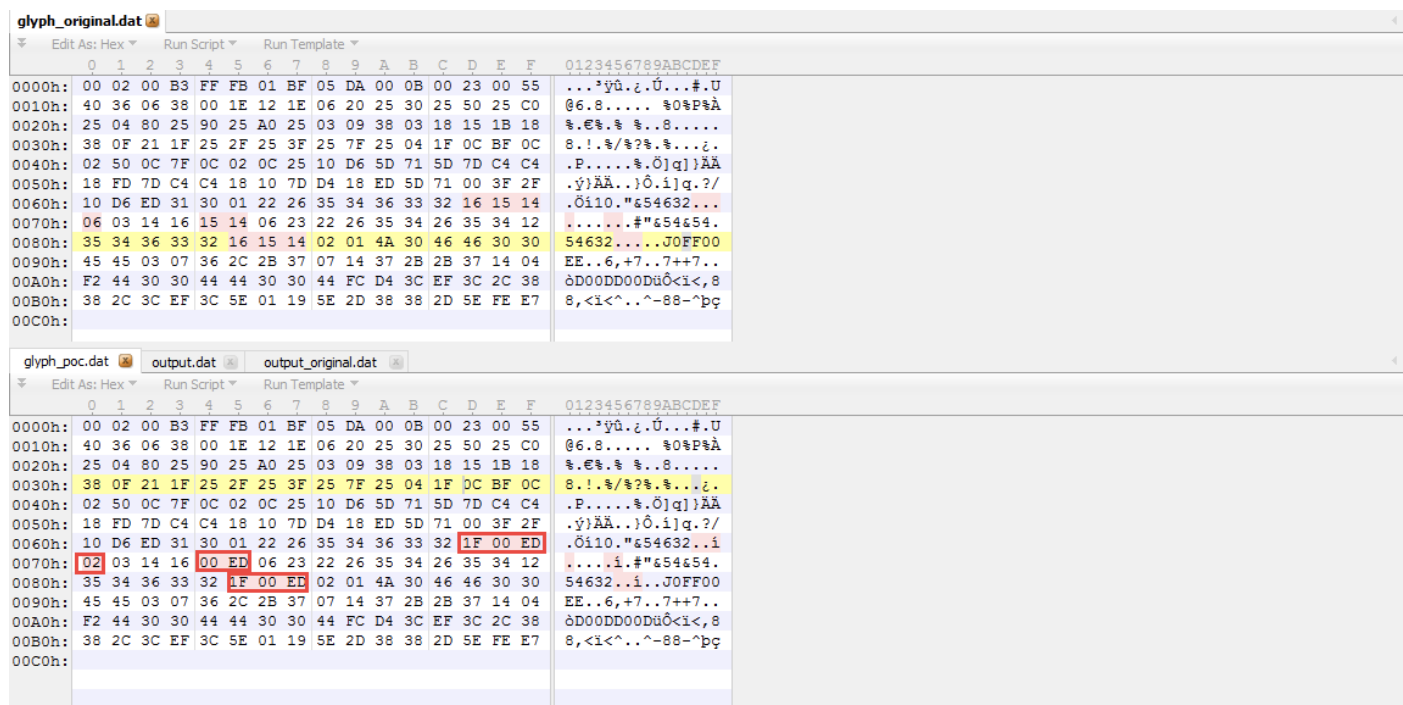


Figure 5. The Comparison Between glyph_poc.dat And glyph_original.dat

As you can see, there are three differences between them. The 'glyph' table contains the data that defines the appearance of the glyphs in the font. This includes specification of the points that describe the contours that make up a glyph outline, and the instructions that grid-fit that glyph. The 'glyph' table supports the definition of simple glyphs and compound glyphs, that is, glyphs that are made up of other glyphs. The number of glyphs in the font is restricted only by the value stated in the 'head' table. The order in which glyphs are placed in a font is arbitrary.

Each glyph begins with the following header:

Type	Name	Description
SHORT	numberOfContours	If the number of contours is greater than or equal to zero, this is a single glyph; if negative, this is a composite glyph.
SHORT	xMin	Minimum x for coordinate data.
SHORT	yMin	Minimum y for coordinate data.
SHORT	xMax	Maximum x for coordinate data.
SHORT	yMax	Maximum y for coordinate data.

Obviously, glyph_poc.dat stores a simple glyph because the first two bytes 0x0002 is greater than zero.

The structure of a simple glyph is shown below:

Type	Name	Description
USHORT	endPtsOfContours[n]	Array of last points of each contour; n is the number of contours.
USHORT	instructionLength	Total number of bytes for instructions.
BYTE	instructions[n]	Array of instructions for each glyph; n is the number of instructions.
BYTE	flags[n]	Array of flags for each coordinate in outline; n is the number of flags.
BYTE or SHORT	xCoordinates[]	First coordinates relative to (0,0); others are relative to previous point.
BYTE or SHORT	yCoordinates[]	First coordinates relative to (0,0); others are relative to previous point.

Based on the specification of the 'glyph' table above, we try to parse it by marking different colors as follows.

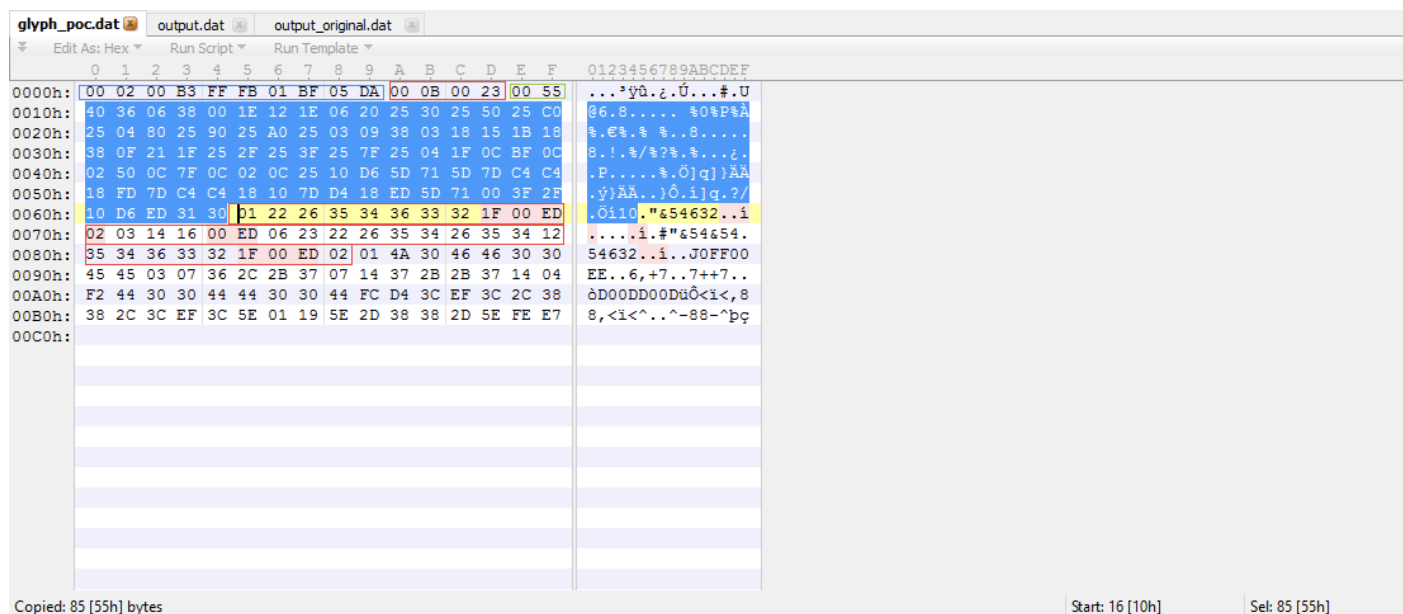


Figure 6. Parsing of The Simple glyph

Then we trace how Adobe Reader handles the simple glyph. We set the following breakpoint in Windbg.

```
bu CoolTypeCTInit+0x44a61 " .if((poi(poi(poi(esp+0xc))) & 0x0fffffff) = 0x0'b3000200) { .printf "\"hit:\\n\\\"; } .else {gc}"
```

When the breakpoint is hit, we get the following debug information:

```
0:021> g

(8de0.6bc4): C++ EH exception - code e06d7363 (first chance)

hit:

eax=097ef6e0 ebx=097ef6d0 ecx=097eeda0 edx=2911e6fc esi=097ef758 edi=097ef756

eip=09598dac esp=2911e610 ebp=2911e674 iopl=0         nv up ei pl zr na pe nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246

CoolType!CTInit+0x44a61:

09598dac e8f1110000    call    CoolType!CTInit+0x45c57 (09599fa2)


0:022> dds esp

2911e610  097eee00 CoolType!CTGetVersion+0x1dd1c8  -->param1

2911e614  097ef510 CoolType!CTGetVersion+0x1dd8d8  -->param2

2911e618  097ef3bc CoolType!CTGetVersion+0x1dd784  -->param3

2911e61c  097ef6e0 CoolType!CTGetVersion+0x1ddaa8  -->param4

2911e620  17e9cd98                                -->param5

2911e624  00000001                                -->param6

2911e628  00000002                                -->param7

2911e62c  097eee58 CoolType!CTGetVersion+0x1dd220

2911e630  097eee60 CoolType!CTGetVersion+0x1dd228

2911e634  097ef756 CoolType!CTGetVersion+0x1ddb1e

2911e638  097ef758 CoolType!CTGetVersion+0x1ddb20

2911e63c  2911e6c4

2911e640  2911e6c8

2911e644  097ef6d0 CoolType!CTGetVersion+0x1dda98

2911e648  17e9cdc4

2911e64c  17e9cf84

2911e650  00000015

...
```

```

0:022> dd 097ef6e0 L4

097ef6e0 2952cf40 2952cf4a 2952d000 00000000


0:022> db 2952cf40 lc0

2952cf40 00 02 00 b3 ff fb 01 bf-05 da 00 0b 00 23 00 55 .....#.U
2952cf50 40 36 06 38 00 1e 12 1e-06 20 25 30 25 50 25 c0 @6.8.....%0%P%.
2952cf60 25 04 80 25 90 25 a0 25-03 09 38 03 18 15 1b 18 %..%..%..8....
2952cf70 38 0f 21 1f 25 2f 25 3f-25 7f 25 04 1f 0c bf 0c 8.!.%/%?%.%....
2952cf80 02 50 0c 7f 0c 02 0c 25-10 d6 5d 71 5d 7d c4 c4 .P.....%.]q]]..
2952cf90 18 fd 7d c4 c4 18 10 7d-d4 18 ed 5d 71 00 3f 2f ..}....}...]q.?!
2952cfa0 10 d6 ed 31 30 01 22 26-35 34 36 33 32 1f 00 ed ...10."&54632...
2952cfb0 02 03 14 16 00 ed 06 23-22 26 35 34 26 35 34 12 .....#*&54&54.
2952cfc0 35 34 36 33 32 1f 00 ed-02 01 4a 30 46 46 30 30 54632.....J0FF00
2952cfd0 45 45 03 07 36 2c 2b 37-07 14 37 2b 2b 37 14 04 EE..6,+7..7++7..
2952cfe0 f2 44 30 30 44 44 30 30-44 fc d4 3c ef 3c 2c 38 .D00DD00D..<.<.,8
2952cff0 38 2c 3c ef 3c 5e 01 19-5e 2d 38 38 2d 5e fe e7 8,<.<^..^88-^..

0:022> !heap -p -a 2952cf40

address 2952cf40 found in

_DPH_HEAP_ROOT @ 3d01000

in busy allocation ( DPH_HEAP_BLOCK:      UserAddr      UserSize -      VirtAddr      VirtSize)

                293c3924:      2952cf40          c0 -      2952c000          2000

6a749abc verifier!AVrfDebugPageHeapAllocate+0x00000023c
7749d836 ntdll!RtlDebugAllocateHeap+0x0000003c
773ffb40 ntdll!RtlpAllocateHeap+0x000000f0
773fdecb ntdll!RtlpAllocateHeapInternal+0x0000027b
773fdc2e ntdll!RtlAllocateHeap+0x0000002e
6854ed63 MSVCR120!malloc+0x00000049 [f:\dd\vctools\crt\crtw32\heap\malloc.c @ 92]

```

```

095550fc CoolType!CTInit+0x00000db1
095589db CoolType!CTInit+0x00004690
...

```

From the above output, the heap buffer at address 0x2952cf40 with size 0xc0 stores the simple glyph. It matches the data in glyph_poc.dat. The address 0x2952cf4a points to endPtsOfContours[2], and the address 0x2952d000 points to the end of the simple glyph.

The following are the values of some fields in the simple glyph.

```

endPtsOfContours [0]: 00 0b

endPtsOfContours [1]: 00 23

instructionLength: 00 55

instructions[n]: 40 36 06 38 00 1e 12 1e-06 20 25 30 25 50 25 c0

                25 04 80 25 90 25 a0 25-03 09 38 03 18 15 1b 18

                38 0f 21 1f 25 2f 25 3f-25 7f 25 04 1f 0c bf 0c

                02 50 0c 7f 0c 02 0c 25-10 d6 5d 71 5d 7d c4 c4

                18 fd 7d c4 c4 18 10 7d-d4 18 ed 5d 71 00 3f 2f

                10 d6 ed 31 30

Flags[]: .....
xCoordinates[]: .....
yCoordinates[]: .....

```

We continued to trace how the 'flags' field in the simple glyph are handled. In IDA Pro, the following is the code branch of handling the 'flags' field. Its length is 0x24(endPtsOfContours [1] + 1=0x23+0x1=0x24).

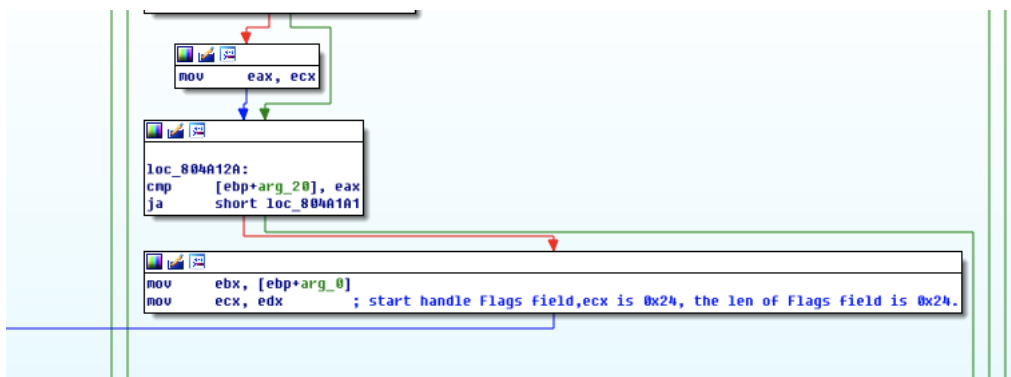


Figure 7. The Branch of Handling the 'flags' Field

In Windbg, we get the following debug info:

0:022> bu 0959a132

0:022> g

Breakpoint 1 hit

eax=0000004d ebx=097eee00 ecx=00000000 edx=00000024 esi=2952cfa5 edi=00000000

eip=0959a132 esp=2911e5f8 ebp=2911e608 iopl=0 nv up ei ng nz ac pe cy

cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000297

CoolType!CTInit+0x45de7:

0959a132 8bca mov ecx,edx

0:022> db esi

2952cfa5 01 22 26 35 34 36 33 32-1f 00 ed 02 03 14 16 00 .*&54632.....

2952cfb5 ed 06 23 22 26 35 34 26-35 34 12 35 34 36 33 32 ..#*&54&54.54632

2952cfc5 1f 00 ed 02 01 4a 30 46-46 30 30 45 45 03 07 36 J0FF00EE..6

2952cfd5 2c 2b 37 07 14 37 2b 2b-37 14 04 f2 44 30 30 44 ,++..7++7...D00D

2952cfe5 44 30 30 44 fc d4 3c ef-3c 2c 38 38 2c 3c ef 3c D00D..<.<,88,<.<

2952cff5 5e 01 19 5e 2d 38 38 2d-5e fe e7 ?? ?? ?? ?? ^..^88-^..?????

2952d005 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????????????????

2952d015 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????????????????

0:022> db ebx

097eee00 01 00 00 00 00 00 00 00-00 00 00 00 00 00 00

097eee10 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00

097eee20 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00

097eee30 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00

097eee40 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00

097eee50 00 00 00 00 00 00 00 00-00 00 0c 00 00 00 00

097eee60 0b 00 23 00 00 00 00 00-00 00 00 00 00 00 00 ..#.....

097eee70 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00



The buffer at address 0x097eee00 is used to store the result of parsing the 'flags' field.

The following C code in function sub_8049FA2 from IDA Pro is used to handle the 'flags' field:

//parse the Flags field which length is 0x24, the result is stored in param a1. v20 points to the offset of data in the glyph object.

do

{

if (v14)

{

v28 -= v14;

v54 = v28;

if (v28 < 0)

return 5133;

if (v14)

{

LOBYTE(v50) = *(v27 - 1);

memset(v27, v50, v14);

v27 += v14;

do

--v14;

while (v14);

v28 = v54;

}

}

else

{

v29 = *(_BYTE *)v20;

*v27 = *(_BYTE *)v20;

if (v29 & 8)

{

if (++v20 > *(_DWORD *) (a4 + 8))

return 5133;

v14 = *(_BYTE *)v20;

```
    }

    ++v20;

    ++v27;

    --v28;

    if ( v20 > *(_DWORD *) (a4 + 8) )

        return 5133;

    }

}

while ( v28 > 0 );

if ( v14 )

    return 5121;

v30 = v55;

v31 = 0;

v32 = (char *)a1; // a1 stores the result of parsing Flags field. |01 22 26 35 34 36 33 32 1f ed ed ed 03 14 16 00 ed ed ed ed ed ed 23 22 26 35 34 26 35 34
12 35 34 36 33|

v33 = 0;

v51 = a1;

v48 = 0;
```

After the parsing of the 'flags' field is finished, we see the following debug info:

```

0:022> t

eax=097ef6e0 ebx=097eee24 ecx=00000000 edx=00000006 esi=2952cfc4 edi=00060000

eip=0959a1ab esp=2911e5f8 ebp=2911e608 iopl=0         nv up ei pl zr na pe nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246

CoolType!CTInit+0x45e60:

0959a1ab 8b7d30      mov     edi,dword ptr [ebp+30h] ss:002b:2911e638=00000024


0:022> db ebx-24 L30

097eee00 01 22 26 35 34 36 33 32-1f ed ed ed 03 14 16 00  ."&54632.....

097eee10 ed ed ed ed ed ed 23-22 26 35 34 26 35 34 12  .....#"&54&54.

097eee20 35 34 36 33 00 00 00 00-00 00 00 00 00 00 00 00 5463.....


0:022> db esi

2952cfc4 32 1f 00 ed 02 01 4a 30-46 46 30 30 45 45 03 07 2.....J0FF00EE..

2952cfd4 36 2c 2b 37 07 14 37 2b-2b 37 14 04 f2 44 30 30 6,+7..7++7...D00

2952cfe4 44 44 30 30 44 fc d4 3c-ef 3c 2c 38 38 2c 3c ef DD00D..<.<,88,<.

2952cff4 3c 5e 01 19 5e 2d 38 38-2d 5e fe e7 ?? ?? ?? ?? <^..^~88-^..????

2952d004 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????

2952d014 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????

2952d024 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????

2952d034 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ????

```

From Figure 5, we can see that three different bytes are located in the 'flags' field. This leads to a very different result.

The result is stored in a buffer at address 0x097eee00, and its size is 0x24. Then the buffer is used to parse the `xCoordinates[]` and `yCoordinates[]` fields. The following C code in function `sub_8049FA2` from IDA Pro is used to handle the `xCoordinates[]` field:

```
v48 = 0;

if ( v55 > 0 ) //handling xCoordinates, the result is stored in param a3.

{

v34 = 0;

do

{

v35 = *v32;

if ( v35 & 2 )

{

v36 = (v35 & 0x10) == 0;

v37 = *(_BYTE *)v20;

if ( v36 )

v34 -= v37;

else

v34 += v37;

++v20;

}

else if ( !(v35 & 0x10) )

{

v38 = *(_BYTE *)v20 + 1;

v34 += _byteswap_ushort(*(_WORD *)v20);

v33 = v48;

v20 += 2;

}

*_DWORD *a3 = v34;

a3 += 4;

v32 = (char *)v51 + 1;

v51 = (char *)v51 + 1;

v30 = v55;
```

```
if ( v20 > *(_DWORD *) (a4 + 8) )  
  
    return 5133;  
  
v48 = ++v33;  
  
}  
  
while ( v33 < v55 );  
  
}
```

After handling 'xCoordinates' is finished, we find the following debug info:

```
0:022> t

eax=097eee24 ebx=00000000 ecx=00000024 edx=0000efeb esi=2952cfef edi=00000024

eip=0959a224 esp=2911e5f8 ebp=2911e608 iopl=0         nv up ei pl zr na pe nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246

CoolType!CTInit+0x45ed9:

0959a224 895d28      mov     dword ptr [ebp+28h],ebx ss:002b:2911e630=097eee24


0:022> db esi

2952cfef 3c ef 3c 2c 38 38 2c 3c ef 3c 5e 01 19 5e 2d 38  <.<,88,<.<^..^~8

2952cffb 38 2d 5e fe e7 ?? ?? ??-?? ?? ?? ?? ?? ?? ?? 8-^..????????????

2952d00b ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????????????

2952d01b ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????????????


0:022> dd 097ef3bc L40

097ef3bc 0000321f 0000321f 00003132 00003132

097ef3cc 00003132 00003134 00003135 0000317f

097ef3dc 000031af 000077f5 fffa825 fffed6a

097ef3ec fffed67 fffed67 fffed6e 0000239a

097ef3fc 00004ed1 000055e5 fff8d10 fffb847

097ef40c fffcc4b fffbe8f fffeebf fffee7b

097ef41c fffee37 fffee07 fffee07 fffee07

097ef42c fffedd7 fffedd7 fffedd7 fffee1b

097ef43c fffee1b fffee1b fffef17 fffefeb

097ef44c 00000000 00000000 00000000 00000000

097ef45c 00000000 00000000 00000000 00000000

097ef46c 00000000 00000000 00000000 00000000

097ef47c 00000000 00000000 00000000 00000000

097ef48c 00000000 00000000 00000000 00000000

097ef49c 00000000 00000000 00000000 00000000
```

```
097ef4ac 00000000 00000000 00000000 00000000
```

Next, the following C code in function `sub_8049FA2` from IDA Pro is used to handle the `yCoordinates[]` field:

```
if ( v30 > 0 ) // handle yCoordinates, the result is stored in param a2.
```

```
{ //v20 points to the following buffer.
```

```
2952cfef 3c ef 3c 2c 38 38 2c 3c ef 3c 5e 01 19 5e 2d 38 <.<,88,<.<^..^~8
```

```
2952cffb 38 2d 5e fe e7 ?? ?? ??-?? ?? ?? ?? ?? ?? ?? 8-^..????????????
```

```
v39 = (char *)a1; // a1 stores the result of parsing the Flags field. |01 22 26 35 34 36 33 32 1f ed ed ed 03 14 16 00 ed ed ed ed ed ed 23 22 26 35 34 26 35 34 12 35 34 36 33|
```

```
v40 = 0;
```

```
v41 = a4;
```

```
do
```

```
{
```

```
v42 = *v39;
```

```
if ( v42 & 4 )
```

```
{
```

```
v36 = (v42 & 0x20) == 0;
```

```
v43 = *(_BYTE *)v20; //crash here!
```

```
if ( v36 )
```

```
v40 -= v43;
```

```
else
```

```
v40 += v43;
```

```
++v20;
```

```
}
```

```
else if ( !(v42 & 0x20) )
```

```
{
```

```
v44 = *(_BYTE *)v20 + 1;
```

```
v40 += _byteswap_ushort(*(_WORD *)v20);
```

```
v41 = a4;
```

```
v20 += 2;
```



```
}  
  
v45 = (_DWORD *)a2;  
  
a2 += 4;  
  
*v45 = v40;  
  
*(_BYTE *)a1 &= 1u;  
  
v39 = (char *)a1 + 1;  
  
a1 = (char *)a1 + 1;  
  
if ( v20 > *(_DWORD *) (v41 + 8) )  
  
    return 5133;  
  
}  
  
while ( ++v31 < v55 ); //v55 is 0x24  
  
}
```

When the crash occurs, we see the following debug info, shown below:

```

0:022> t

eax=097eeee ebx=00000015 ecx=097ef6e0 edx=0000cc6c esi=2952d000 edi=00000024

eip=0959a23c esp=2911e5f8 ebp=2911e608 iopl=0         nv up ei pl nz na po nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202

CoolType!CTInit+0x45ef1:

0959a23c 0fb606          movzx  eax,byte ptr [esi]      ds:002b:2952d000=??

```

```

0:022> t

(8de0.6bc4): Access violation - code c0000005 (first chance)

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

eax=097eeee ebx=00000015 ecx=097ef6e0 edx=0000cc6c esi=2952d000 edi=00000024

eip=0959a23c esp=2911e5f8 ebp=2911e608 iopl=0         nv up ei pl nz na po nc

cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202

CoolType!CTInit+0x45ef1:

0959a23c 0fb606          movzx  eax,byte ptr [esi]      ds:002b:2952d000=??

```

```

0:022> dd 097ef510

097ef510  00003cef 00003cef 00003d2b 00003d57

097ef520  00003d8f 00003dc7 00003dc7 00003dc7

097ef530  00003d9b 00003dd7 00003ec6 00003f02

097ef540  ffff9d03 ffff9cea ffff9c8c ffff9c4

097ef550  fffc9fc fffc9ca29 fffc9ca87 fffc9cb85

097ef560  fffc6c 00000000 00000000 00000000

097ef570  00000000 00000000 00000000 00000000

097ef580  00000000 00000000 00000000 00000000

```

```

0:022> db esi L10

2952d000  ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ??  ?????????????????

```

Because ebx is smaller than edi, the loop condition is still true and the program continues to parse the yCoordinates[] field. While esi points to the end of the simple glyph, it causes an out of bounds memory access.

In summary, this vulnerability is a typical heap buffer overflow vulnerability. In particular, the vulnerability is caused by a crafted PDF file which includes an invalid font file. It causes an out of bounds memory access, due to improper bounds checking when manipulating an array pointer. Attackers can exploit the vulnerability by using the out of bounds access for unintended reads, writes, or frees – potentially leading to code corruption, control-flow hijack, or information leak attack.

Mitigation

All users of Adobe Acrobat and Reader are encouraged to upgrade to the latest version of these software. Additionally, organizations that have deployed Fortinet IPS solutions are already protected from this vulnerability with the signature **Adobe.Acrobat.Reader.CoolType.TTF.Memory.Corruption**.

by  Kai Lu | Jul 20, 2016 | Filed in: Security Research (/category/security-research)

Tags: [adobe \(/tag/adobe\)](#) | [vuln \(/tag/vuln\)](#) | [vulnerability \(/tag/vulnerability\)](#) | [cve \(/tag/cve\)](#)

↑ Next Post: A Peek into BlackMoon's Sustained Attacks against South Korea (/2016/07/21/a-peek-into-blackmoon-s-sustained-attacks-against-south-korea)

↓ Previous Post: Finding the Right Balance Between Security and Patient Care (/2016/07/15/finding-the-right-balance-between-security-and-patient-care)

0 Comments Fortinet Blog

 Login ▾




♥ Recommend 2  Share

Sort by Best ▾



Start the discussion...

Be the first to comment.

 Subscribe  Add Disqus to your site [Add Disqus](#) [Add](#)  Privacy

Corporate

About Fortinet (<http://fortinet.com/aboutus/aboutus.html>)

Investor Relations (<http://investor.fortinet.com>)

Careers (<http://jobs.fortinet.com>)

Partners (<http://fortinet.com/partners/index.html>)

Global Offices (<http://fortinet.com/aboutus/locations.html>)

Fortinet in the News (<http://fortinet.com/aboutus/media/news.html>)

Contact Us (http://fortinet.com/contact_us/index.html)

How to Buy

Find a Reseller (http://fortinet.com/partners/reseller_locator/locator.html)

FortiPartner Program (http://fortinet.com/partners/partner_program/fpp.html)

Fortinet Store (<https://store.fortinet.com>)

Products

Product Family (<http://fortinet.com/products/index.html>)

Certifications (http://fortinet.com/aboutus/fortinet_advantages/certifications.html)

Awards (http://fortinet.com/aboutus/fortinet_advantages/awards.html)

Video Library (<http://video.fortinet.com>)

Service & Support

FortiCare Support (http://fortinet.com/support/forticare_support/index.html)

Support Helpdesk (<https://support.fortinet.com>)

FortiGuard Center (<http://fortiguard.com>)

 (<http://www.facebook.com/fortinet>)  (<http://www.twitter.com/fortinet>)  (<http://www.youtube.com/user/SecureNetworks>)  (<http://www.linkedin.com/company/fortinet>)
 (/feed)

Copyright © 2000 - 2017 Fortinet, Inc. All Rights Reserved. | Terms of Service (<http://fortinet.com/aboutus/legal.html>) | Privacy (<http://fortinet.com/aboutus/privacy.html>)