SMASHING THE FONT SCALER ENGINE IN WINDOWS KERNEL

Ling Chuan Lee@F13 Labs

Lee Yee Chan@F13 Labs

About US

- Ling Chuan Lee (a.k.alclee_vx)

  Founder F13 Laboratory
- Lee Yee Chan (a.k.a lychan25)

  Founder of F13 Laboratory

Agenda

- Introduction
- Fuzzing Infra
- Bug Hunting with TrueType Font Fuzzer
- Windows Kernel Font Attack Vector
- TrueType Font Bugs
- Notes

Introduction

Introduction

- TrueType Font is a digital font includes many different kind of information used by rasterizer and operating system software to display characters on the computer screen or print out in other device, for instance: printer
- Two groups of categories are exist:

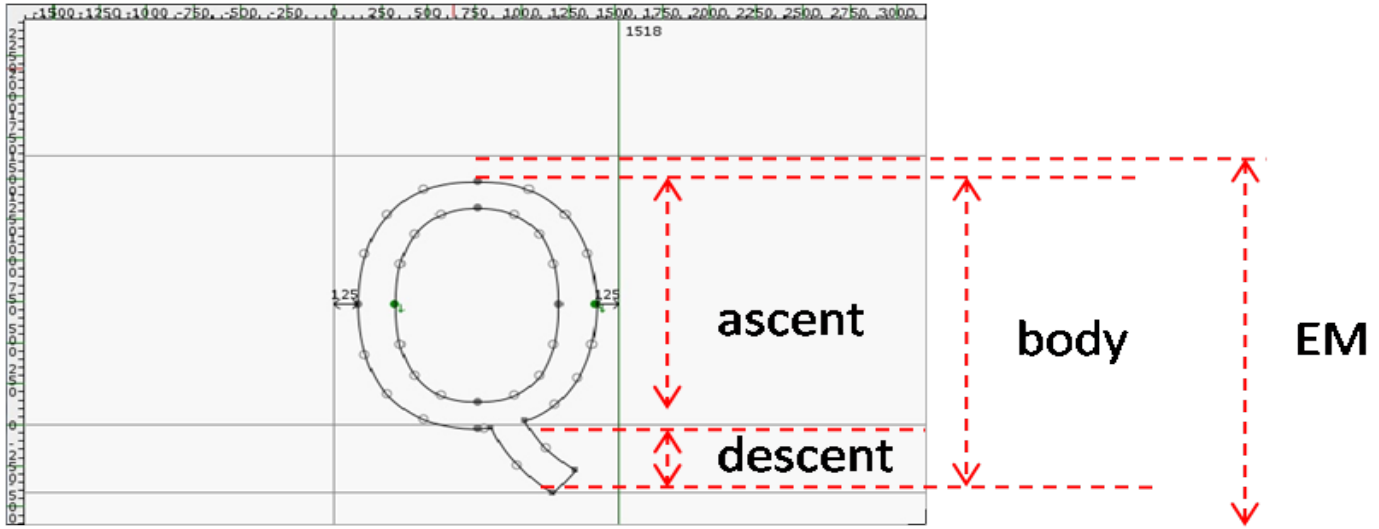
  - GDI Font: raster, vector, TrueType &OpenType

  - Device Font
- Foundation of TrueType font:
  - Outline : Glyph shapes are described by font outline, a glyph outline consists of a series of contours
  - FUnit : Describe the TrueType font file point location in the em square

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

Introduction

- Foundation of TrueType font:
  - Em Square : An imaginary square that is used to size and align glyphs
  - Grid : Two dimensional coordinate system; x-axis describes movement in a horizontal direction; y-axis describes movement in a vertical direction



Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

Introduction

- Process glyphs from a TrueType font file to be displayed on raster devices:
  - The outline stored in the font file is scale to the requested size.
  - Scaler converts FUnits to pixel coordinates and scales outline to the size requested by application.
  - Instructions associated with glyph are carried out by the interpreter. Interpreter executes instructions associated with glyph and grid fits.
  - The result is a grid-fitted outline for the requested glyph.
  - The outline is then scan converted to produce bitmap that can be render on the targeted device.

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

Font Scaler Engine

- TrueType Font Scaler creates the necessary bitmap at a particular resolution when a specific point size is requested by an application.
- The conversion of an outline into a bitmap is referred to as scan conversion.
- To solve the low resolution issue in outline to bitmap conversion, each glyph include a set of instructions that instruct the font scaler to modify the shape of the glyph before scan conversion.

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

Font Scaler Engine

- Font Scaler consists of a set of API functions. User can pass parameters to the Font Scaler through the fs_GlyphInputType data structure and receive information from the fs_GlyphInfoType record.
- Functions for Engine exported interface:
  - Win32k!fs_OpenFonts
  - Win32k! fs_NewGlyph
  - Win32k!fs_ContourScan
  - Win32k!fs_FindBitMapSize
  - Win32k!fs_SetUpKey
  - Win32k!fs_Initialize
  - Win32k!fs_NewSfnt
  - Win32k!fs_WinNTGetGlyphIDs
  - Win32k!fs_ConvertGrayLevels
  - Win32k!fs_NewControurGridFit
  - Win32k!fs_GetGlyphIDs

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995
Credit to : Understanding Windows Kernel Font Scaler Engine Vulnerability, Wang Yu SyScan 360 2012

Font Scaler Engine

- Functions for Engine internal interface:
  - Win32k!fs__Contour - Win32k!fs__NewTransformation
- Functions for Engine converter function:
  - Win32k!fsc_SetupScan - Win32k!fsc_FillGlyph
  - Win32k!fsc_FillBitMap - Win32k!fsc_CalcSpline
  - Win32k!fsc_MeasureGlyph - Win32k!fsc_CalcLine
  - …… ……
- Functions for Engine support interface:
  - Win32k!fsg_CreateGlyphData - Win32k!fsg_GridFit
  - Win32k!fsg_ExecuteGlyph - Win32k!fsg_PrivateFontSpaceSize
  - …… ……

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995
Credit to : Understanding Windows Kernel Font Scaler Engine Vulnerability, Wang Yu SyScan 360 2012

Font Scaler Engine

- Functions for Bitmap related:
  - Win32k!sbit_GetMetrics - Win32k!sbit_GetBitmap
  - Win32k!sbit_ValidateScaleY - Win32k!sbit_ValidateScaleX
  - …… ……
- Functions for Instruction Virtual Machine:
  - itrp_Execute - itrp_InnerExecute
  - itrp_CALL - itrp_FDEF
  - …… ……
- Functions for Font Structure Parser:
  - Win32k!sfac_GetSbitMetrics - Win32k!sfac_SearchForStrike
  - …… ……

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995
Credit to : Understanding Windows Kernel Font Scaler Engine Vulnerability, Wang Yu SyScan 360 2012

Fuzzing Infra

Fuzzing Infra

- We use 2 servers in our fuzz farm specific for TrueType Font fuzzing on Windows Platform.
- We build with:
  - Custom built server with Memory 32GB, Storage 1 TB, 1 wireless card
  - Ubuntu server 10.04 64-bits (we don't really care about the latest version)
  - Each Ubuntu server installed VMWare Workstation 9.0
  - Each Ubuntu server able to switch on 18 Windows 8 Pro operating system
  - Total fuzzing test case per day ~300,000
  - Total Hardware cost ~RM 10K

Fuzzing Infra

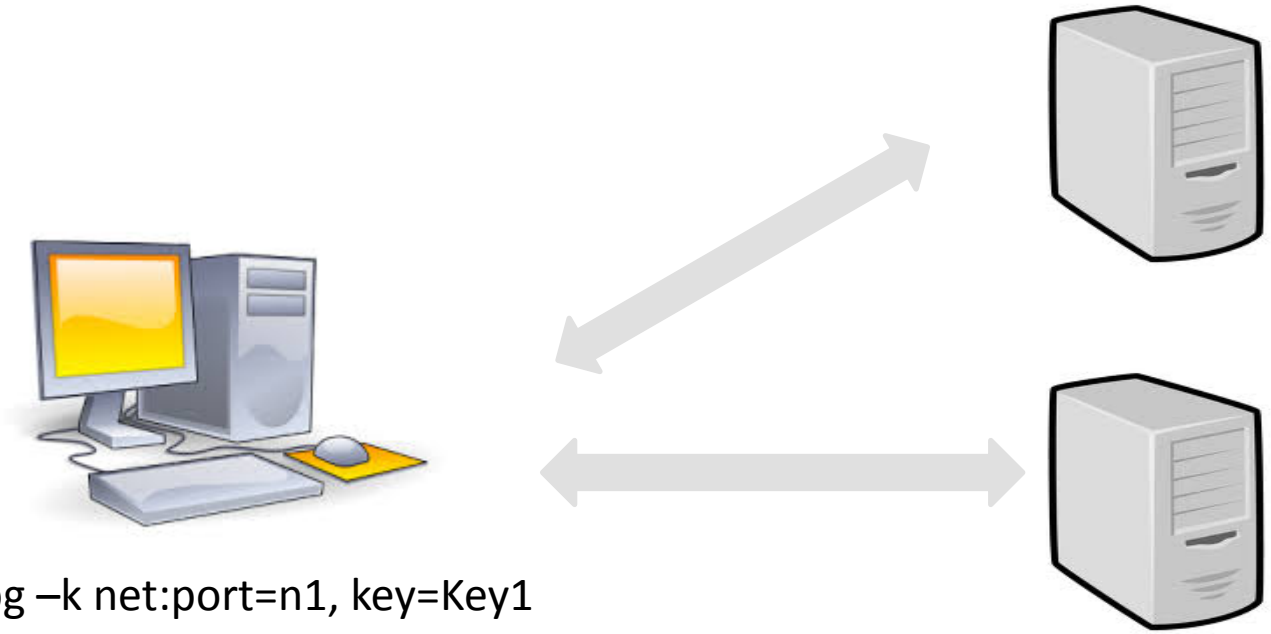
- Since Windows 8 Pro, kernel debugging over an Ethernet network is supported. The reverse engineer not need to suffer set up the kernel debugging via serial port, 1394 port etc.
- To set up WinDBG kernel debugging through the network:
  - Target computer (Debuggee):
    - bcdedit /debug on
    - bcdedit /dbgsettings net hostip:w.x.y.z port:n

    Host computer (Debugger):
    - windbg –k net:port=n, key=Key

Reference: http://msdn.microsoft.com/en-us/library/windows/hardware/hh439346(v=vs.85).aspx

Fuzzing Infra

-



Windbg –k net:port=n1, key=Key1
Windbg –k net:port=n2, key=Key2
……
……

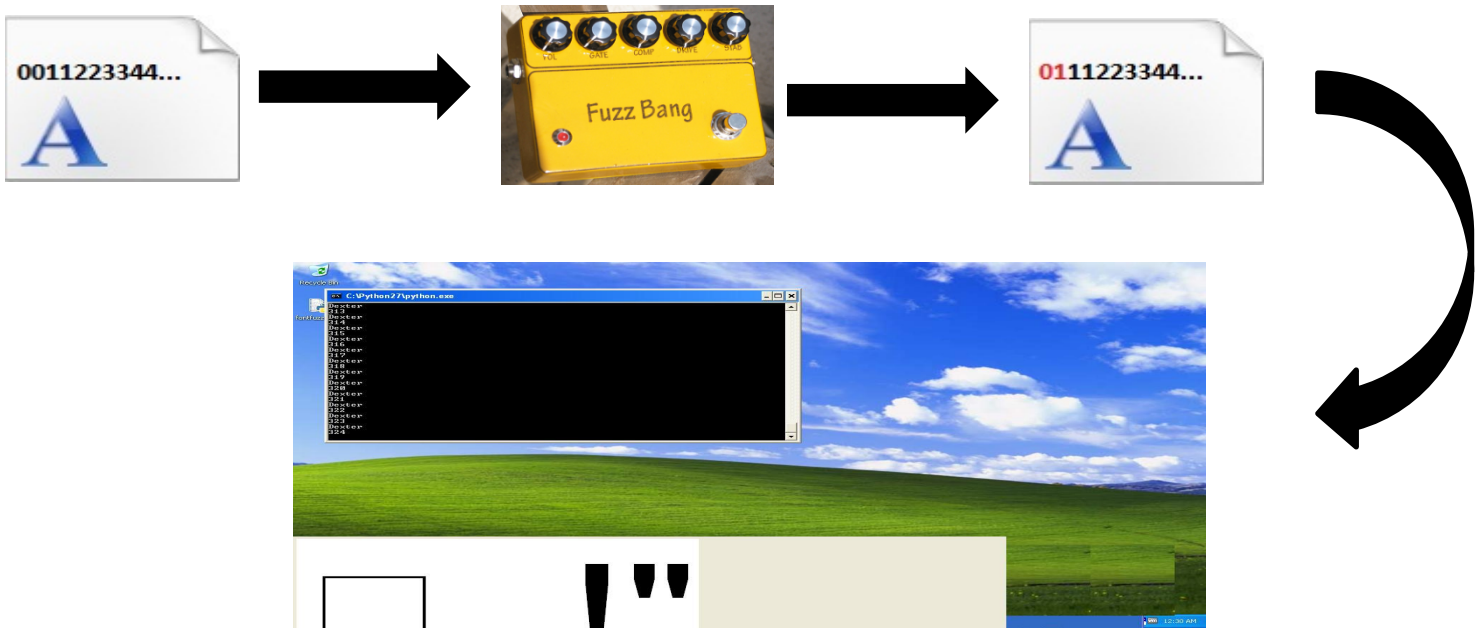
Each server running with ~18 Windows 8 Pro OS enable kernel mode debugging over an ethernet network

Fuzzing Infra

Bug Hunting with TrueType Font Fuzzer

TrueType Font Fuzzer

- Fuzzing is a software testing methodology, aims to provide invalid, mutated or malformed inputs of application in the hope that the application exhibits some security issue



TrueType Font Fuzzer

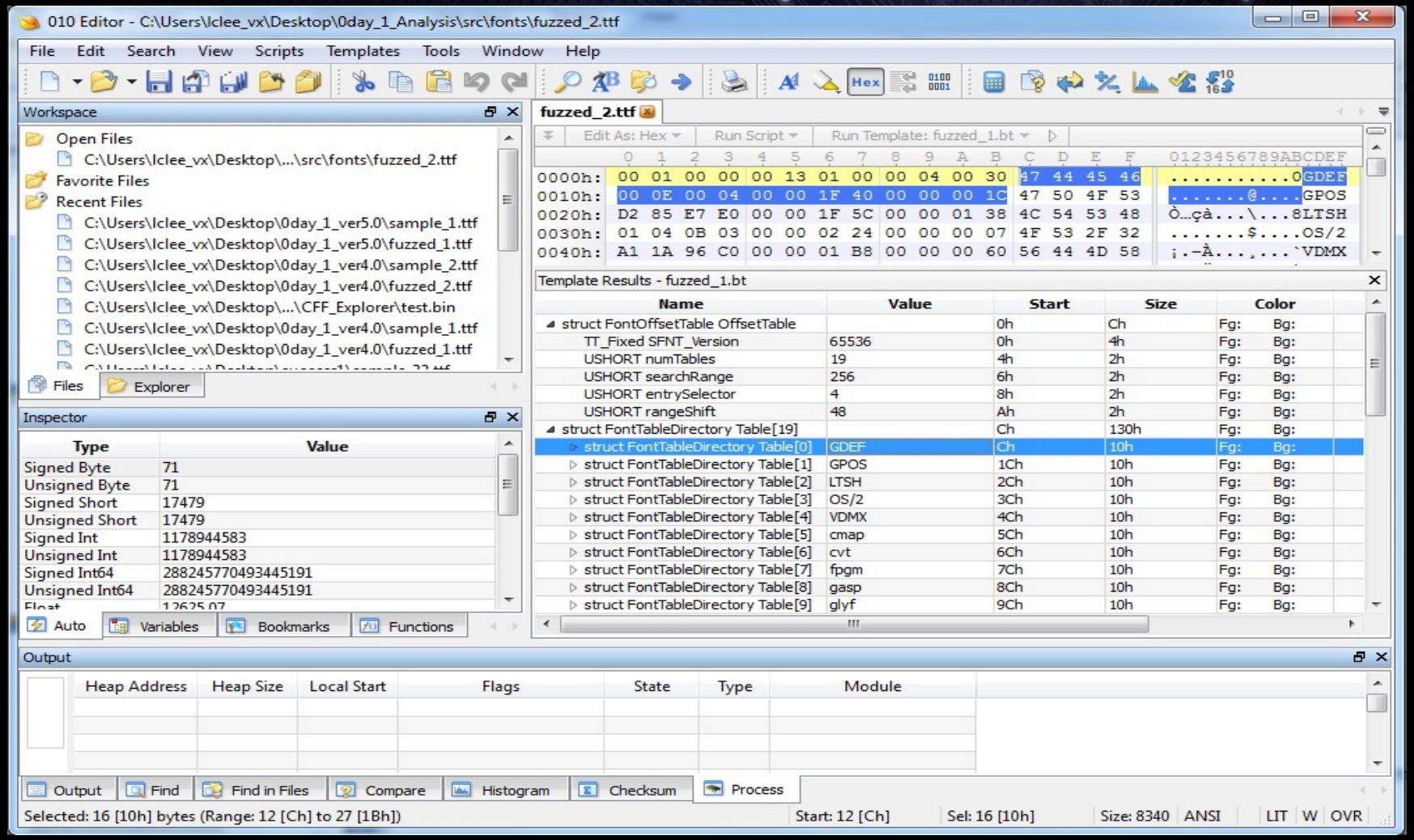
- Dumb fuzzing:
  - Simple modification of legitimate data feeding the targeted application without awareness of its data structure
  - Dumb fuzzing is not the good method in TrueType Font Fuzzing
- Smart fuzzing:
  - Generate inputs that are malformed but mostly compliant with the consideration of data structure such as ‘checksum’, ‘offset’, ‘relations’ and ‘encoding’

TrueType Font Format

- The 010 Binary Editor parses a variety of file into a hierarchical structure formats using a binary template
  - Free 30-day trial ☺!!
  - Support multiple platform: Windows, Mac OSX
  - Free binary templates (*.bmp, *.zip, *.wav)
- We developed the TrueType (*.ttf) and OpenType (*.otf) binary template for the internal usage

Reference: http://www.sweetscape.com/010editor/

TrueType Font Format



TrueType Font Format

- A TrueType font file contains data, in table format that comprises an outline font.
- The TrueType font file begins at byte 0 with the Font Offset Table
- Offset Table is divided into 5 subtable:

sfnt version : 65536 (0x0001 0000) for version 1.0

numTables : Number of tables

searchRange : (Maximum power of 2 ≤ numTables)x16

entrySelector : Log2(Maximum power of 2≤ numTables)

rangeShift : numTablesx 16 - searchRange

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

TrueType Font Format

fuzzed_2.ttf

Edit As: Hex | Run Script | Run Template: fuzzed_1.bt

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 00 01 00 00 00 13 01 00 00 04 00 30 47 44 45 46  ...........0GDEF
0010h: 00 0E 00 04 00 00 1F 40 00 00 00 1C 47 50 4F 53  .......@....GPOS
0020h: D2 85 E7 E0 00 00 1F 5C 00 00 01 38 4C 54 53 48  Ò…çà...\...8LTSH
0030h: 01 04 0B 03 00 00 02 24 00 00 00 07 4F 53 2F 32  .......$....OS/2
0040h: A1 1A 96 C0 00 00 01 B8 00 00 00 60 56 44 4D 58  ¡.–À...¸...`VDMX
```

Template Results - fuzzed_1.bt

Name	Value	Start	Size	Color
struct FontOffsetTable OffsetTable		0h	Ch	Fg: Bg:
TT_Fixed SFNT_Version	65536	0h	4h	Fg: Bg:
USHORT numTables	19	4h	2h	Fg: Bg:
USHORT searchRange	256	6h	2h	Fg: Bg:
USHORT entrySelector	4	8h	2h	Fg: Bg:
USHORT rangeShift	48	Ah	2h	Fg: Bg:
struct FontTableDirectory Table[19]		Ch	130h	Fg: Bg:
struct FontTableDirectory Table[0]	GDEF	Ch	10h	Fg: Bg:
struct FontTableDirectory Table[1]	GPOS	1Ch	10h	Fg: Bg:
struct FontTableDirectory Table[2]	LTSH	2Ch	10h	Fg: Bg:
struct FontTableDirectory Table[3]	OS/2	3Ch	10h	Fg: Bg:
struct FontTableDirectory Table[4]	VDMX	4Ch	10h	Fg: Bg:
struct FontTableDirectory Table[5]	cmap	5Ch	10h	Fg: Bg:
struct FontTableDirectory Table[6]	cvt	6Ch	10h	Fg: Bg:
struct FontTableDirectory Table[7]	fpgm	7Ch	10h	Fg: Bg:
struct FontTableDirectory Table[8]	gasp	8Ch	10h	Fg: Bg:
struct FontTableDirectory Table[9]	glyf	9Ch	10h	Fg: Bg:

TrueType Font Format

- The Font Table Directory entries is followed after the Font Offset Table, begins at byte 12.
- Entries in the Table Directory must be sorted in ascending order by 'tag' name
- Font Table Directory Header:

tag *: 4 byte identifier*

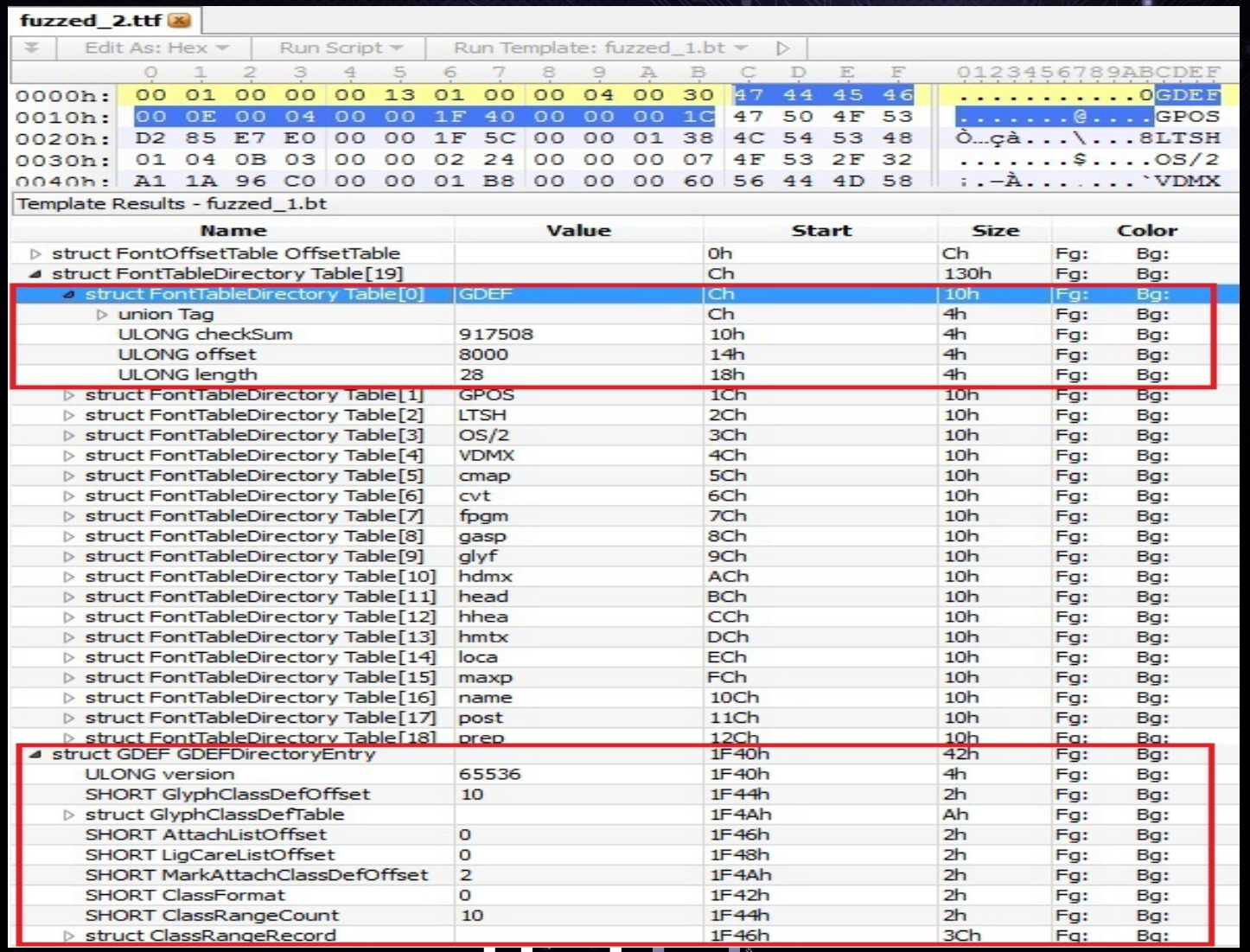
checkSum *: Checksum of the table*

offset *: Beginning offset of the font table entry*

length *: Length of the table*

Reference: TrueType 1.0 Font File, Technical Specification Revision 1.66 August 1995

TrueType Font Format



TrueType Font Format

- Required Tables in Font Offset Table:

  cmap : character to glyph mapping

  glyf : glyph data

  head : font header

  hhea : horizontal header

  hmtx : horizontal metrics

  loca : index to location

  maxp : maximum profile

  name : naming table

  post : PostScript information

  OS/2 : OS/2 and Windows specific metrics

TrueType Font Format

- Optional Tables in Font Offset Table:

  cvt : Control Value Table
  EBDT : Embedded bitmap data
  EBLC : Embedded bitmap location data
  EBSC : Embedded bitmap scaling data
  fpgm : font program
  gasp : grid-fitting and scan conversion procedure
  hdmx : horizontal device metrics
  kern : kerning
  LTSH : Linear threshold table
  prep : CVT Program
  PCLT : PCL5

TrueType Font Format

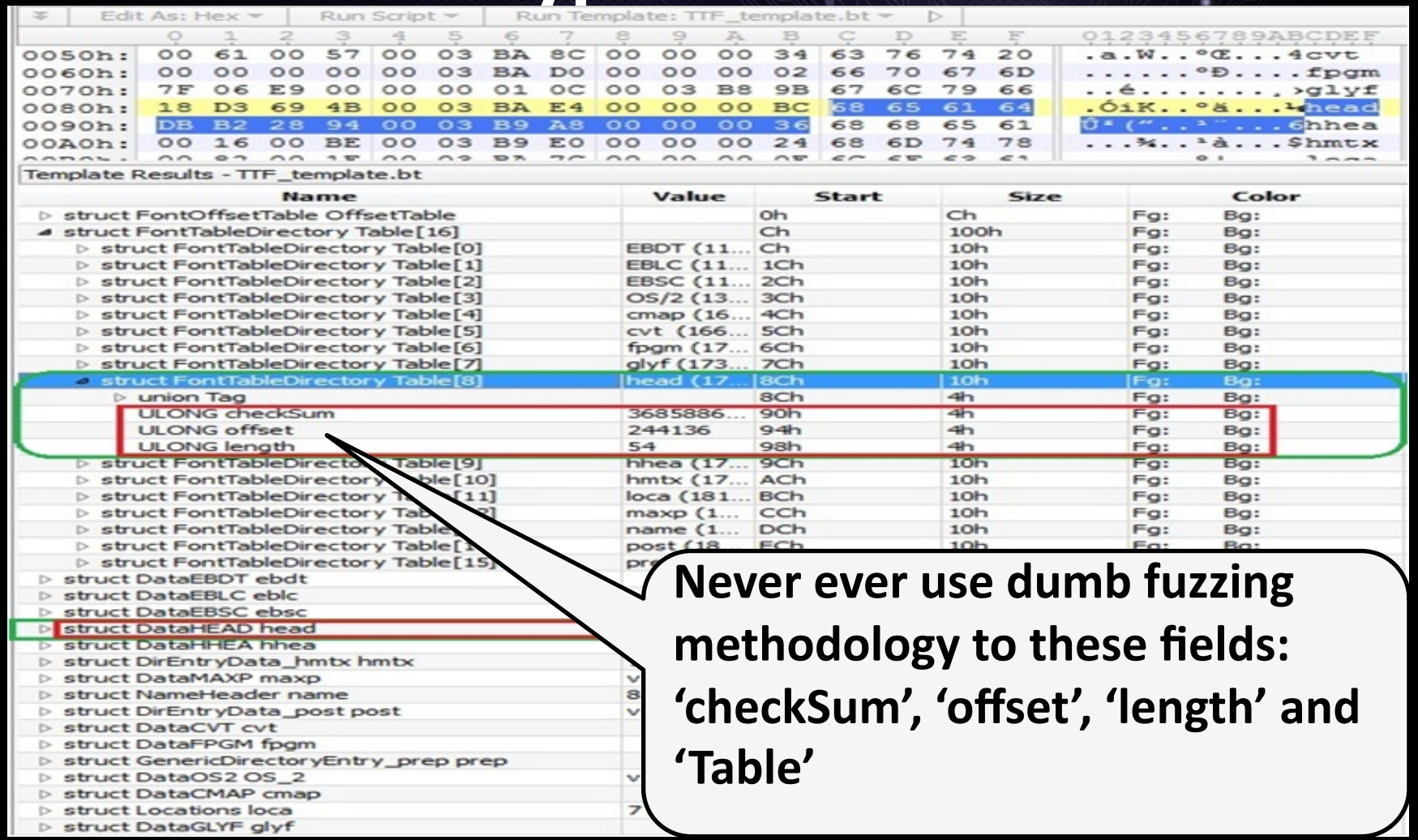
- Optional Tables in Font Offset Table:

  VDMX : Vertical Device Metrics table

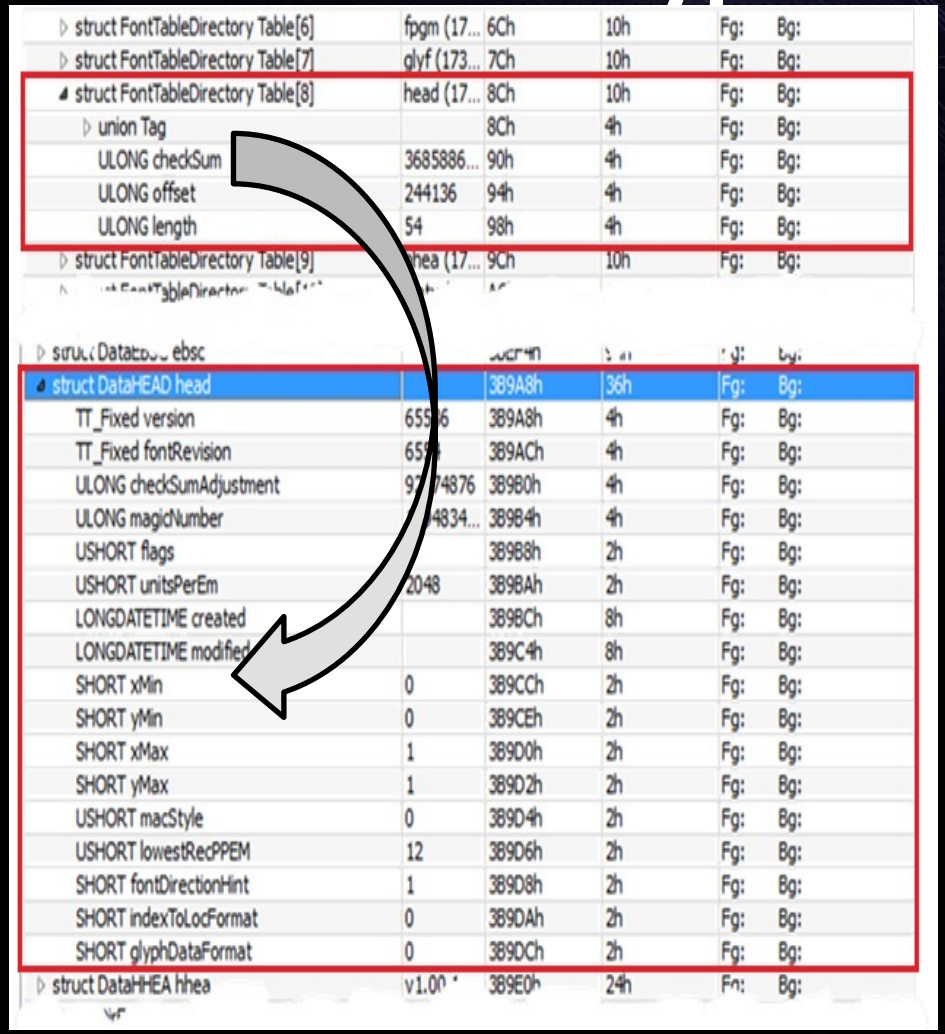
  vhea : Vertical Metrics header

  vmtx : Vertical Metrics

Bug Hunting with TrueType Font Fuzzer



Bug Hunting with TrueType Font Fuzzer

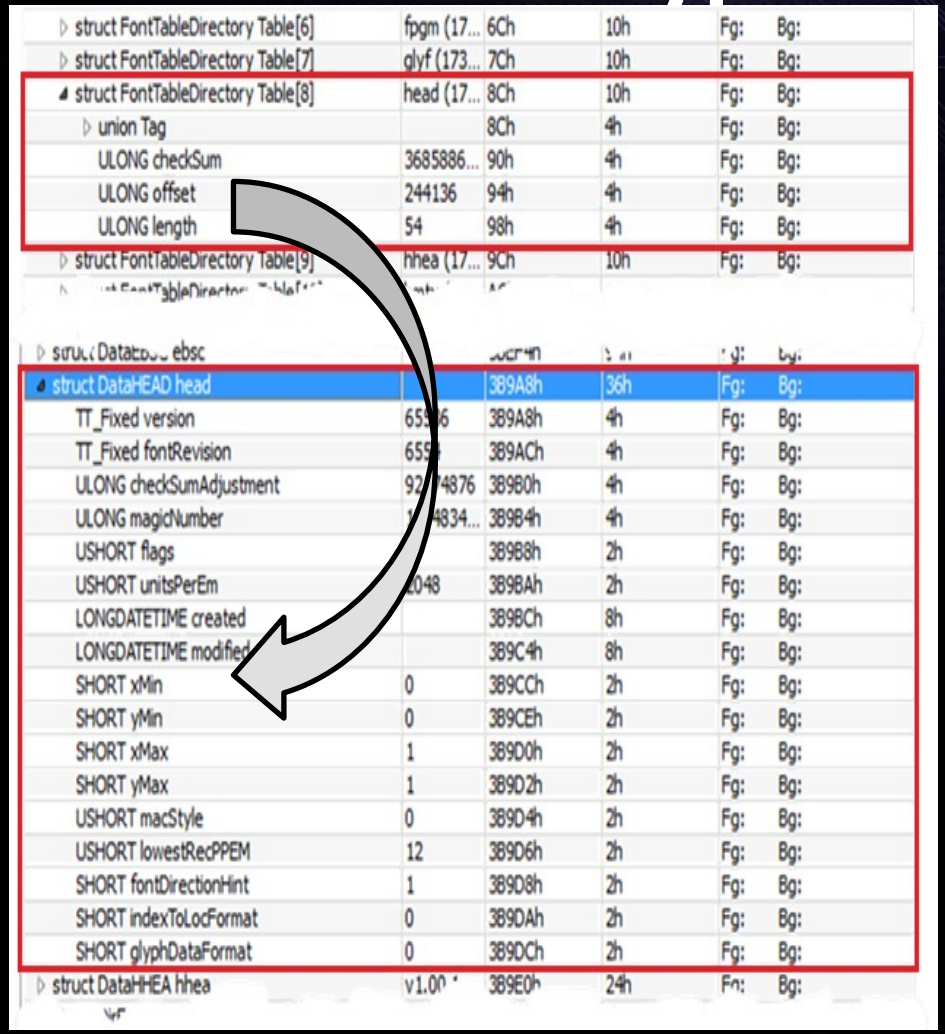


1. python code is used to determine the checksum of 'head' table

```
def chk(tab):
    total_data=0
    for i in range(0, len(tab), 4):
        data=unpack(">I", tab[i:i+4]) [0]
        total_data += data
    final_data=0xFFFFFFFF &total_data
    return final_data
```

2. The checksum calculation implies 4 byte boundaries for the entire table, and pad the remaining space with zeros

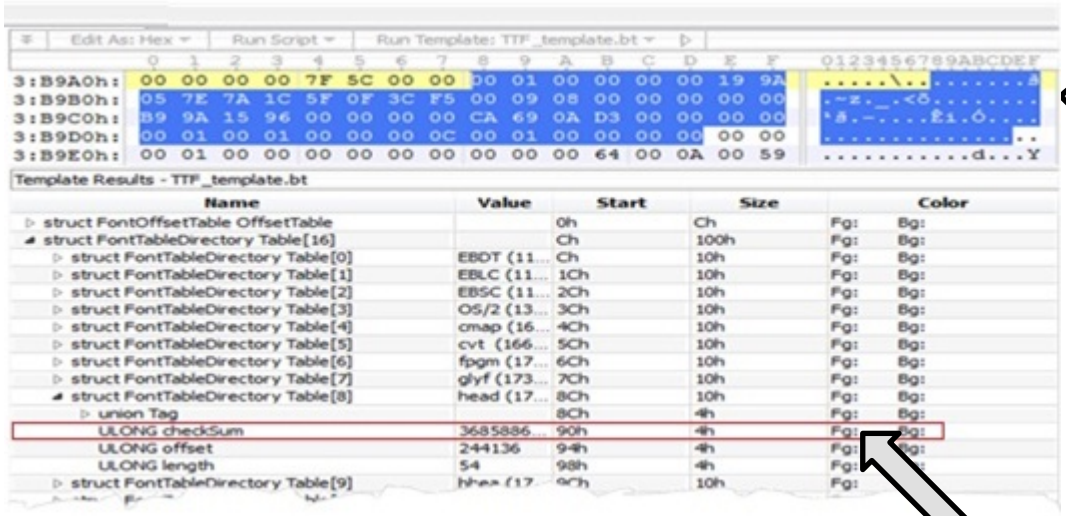
Bug Hunting with TrueType Font Fuzzer

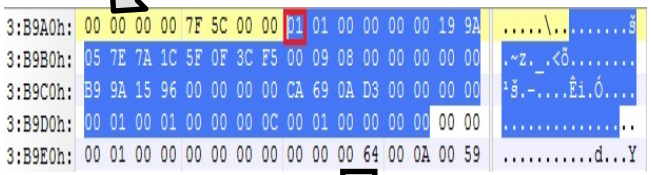


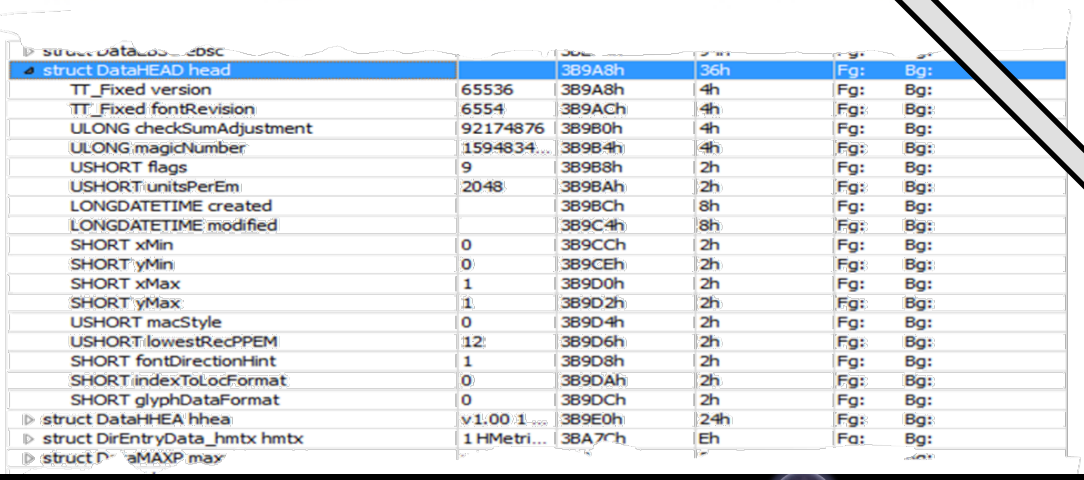
3. ‘Offset’ points to the beginning of ‘DataHEAD’ structure

4. ‘Length’ defines the size of ‘DataHEAD’ table

Bug Hunting with TrueType Font Fuzzer








```
def chk(tab):
    total_data=0
    for i in range(0, len(tab), 4):
        data=unpack(“>I”, tab[i:i+4]) [0]
        total_data += data
    final_data=0xFFFFFFFF & total_data
    return final_data
```

Bug Hunting with TrueType Font Fuzzer

- We can fuzz a font by:

  a. Byte flipping of the entire TrueType font table
  (1 byte, 2 bytes, 4 bytes…)

  b. During fuzzing, every flipping test requires to fix the checksum value

  c. Filling in a LOGFONT structure

  d. Calling 'CreateFontIndirect' to return a font handle (HFONT)

  e. Work with fonts at a lower level through font APIs:
  GetFontData, GetGlyphIndices, ExtTextOut with ETO_GLYPH_INDEX flag

Bug Hunting with TrueType Font Fuzzer

- Every fuzzed TrueType font will:

  a. automatically install the crafted font in 'C:\WINDOWS\Fonts' folder

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

  b. Register a window class and creating a new window to automate the font text display in the range of font size

  c. Remove the fonts in 'C:\WINDOWS\Fonts' folder

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

- Part of the fuzzer's source code is shared in github:

  https://github.com/lingchuanlee/FontFuzzer

Bug Hunting with TrueType Font Fuzzer

- Create the window class and define LOGFONT structure

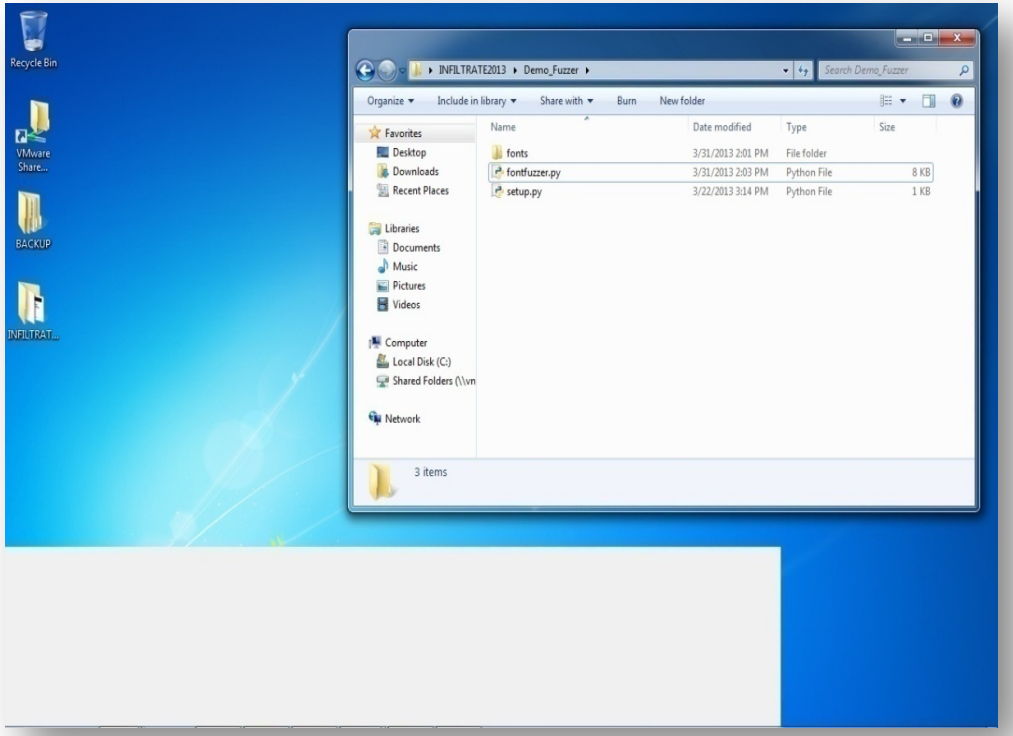
lf =win32gui.LOGFONT()

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

w=mainWindow()

hwnd=w.CreateWindow()

hdc=windll.user32.GetDC(hwnd)



Bug Hunting with TrueType Font Fuzzer

- Defined a character map of a TrueType font

```
z=[
    chr(0),chr(1),chr(2),chr(3),chr(4),chr(5),chr(6),chr(7),chr(8),chr(9),
    chr(10),chr(11),chr(12),chr(13),chr(14),chr(15),chr(16),chr(17),chr(18),chr(19),
     [......]
    chr(250),chr(251),chr(252),chr(253),chr(254),chr(255)
       ]
array_types=c_wchar*256
var1=array_types()
for y in range(1, 256, 1):
    var1[y]=z[y]
    [……]
```

Bug Hunting with TrueType Font Fuzzer

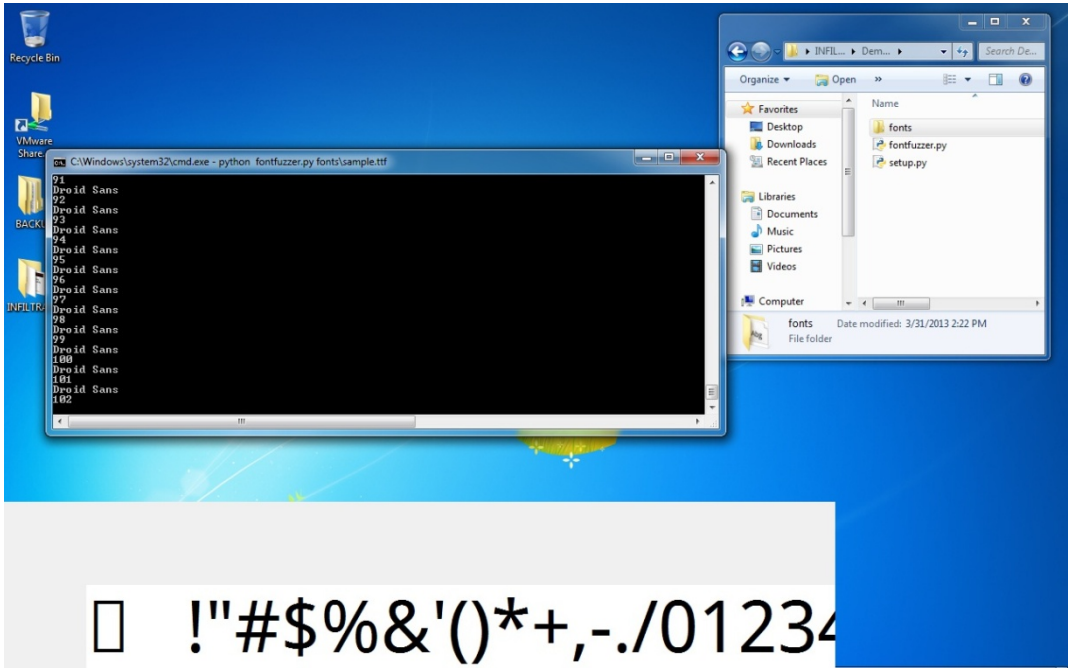
- A range of font size

```
for fontsize in range (1, 100, 1):
        lf.lfHeight=fontsize
        lf.lfFaceName="Droid"
        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
```

Bug Hunting with TrueType Font Fuzzer

- Draws text using the selected font

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



Demonstration

Windows Kernel Font Attack Vector

GDI Font Kernel Attack

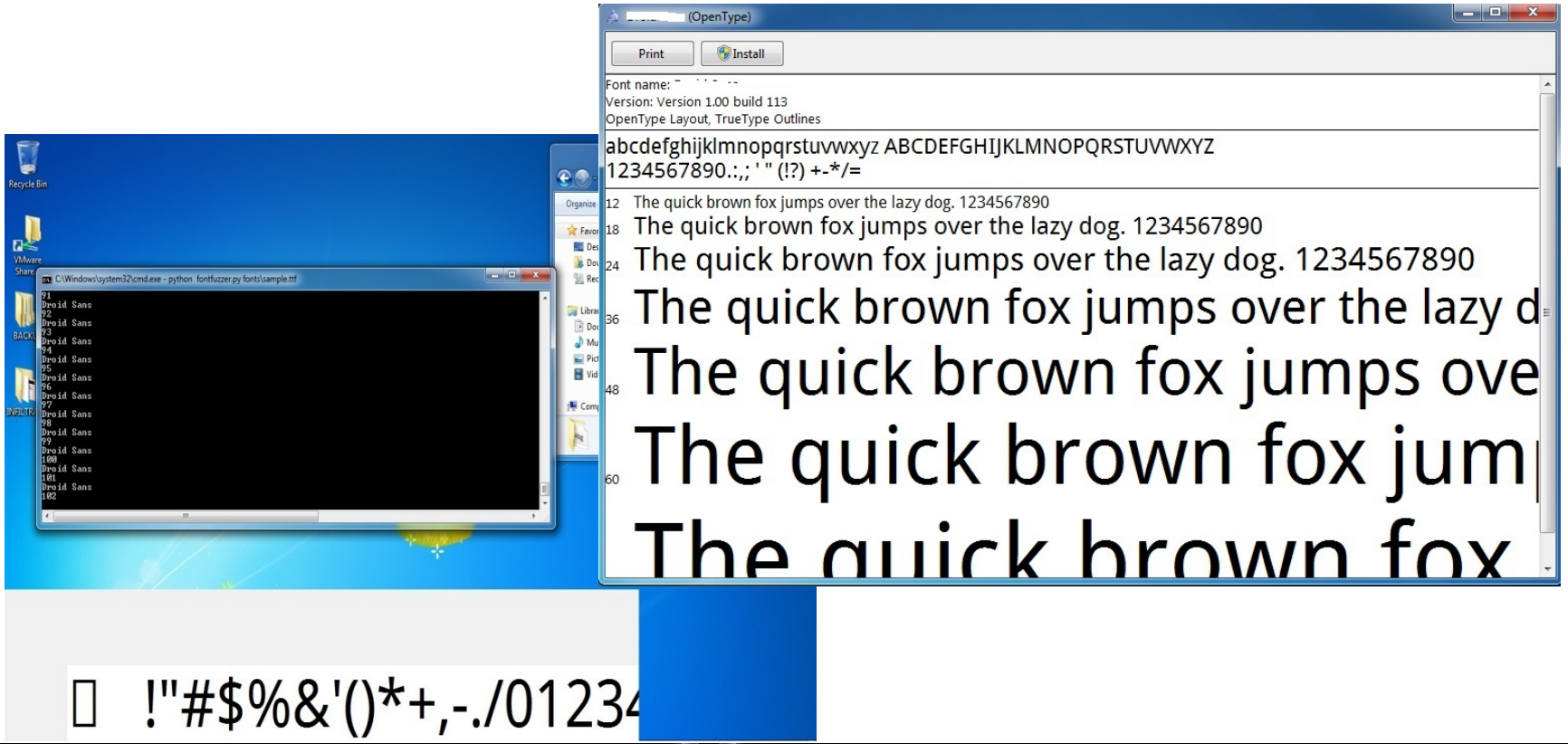
- The Graphics Device Interface (GDI) is part of the core OS component. Responsible for graphical object and transmitting output to devices such as video displays as well as printers
- There are different types of font available on Windows. Two groups of categories: GDI fonts and device fonts
- GDI fonts, based in Windows consists of three types: raster, stroke and true type

Font Attack Vector

- We identify font vulnerability as one of the likely weak points and accessible via browser (Firefox, Chrome), Microsoft Office Documents (*.docx, *.pptx) and other application Adobe Portable Document format (.pdf)
- Local Windows Kernel Exploit - copy and execute a crafted font on Windows system to raise the attacker's privilege as super user
- Remote Windows Kernel Exploit - included social engineering and requires the target to open the crafted Microsoft Word (*.docx) or website

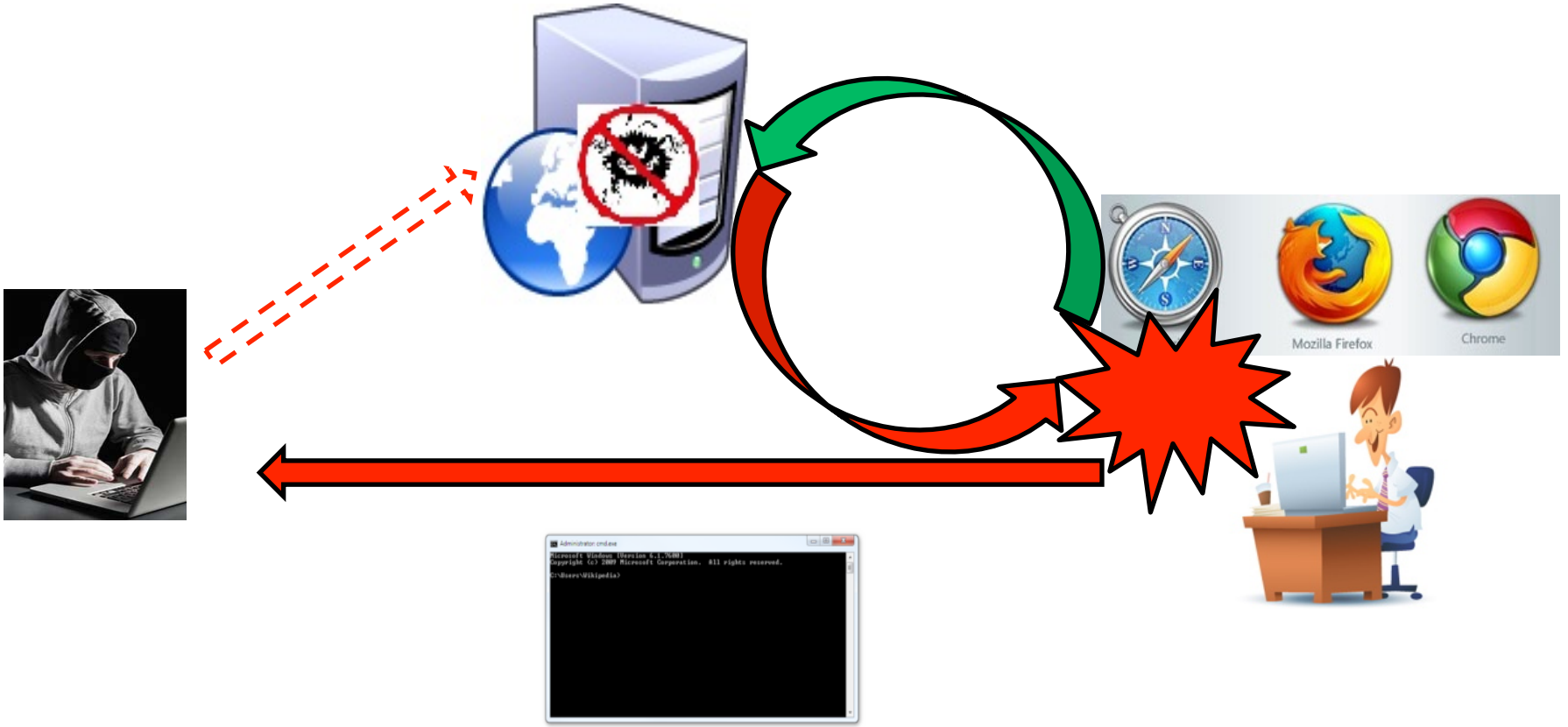
Local Font Attack Vector

- The attacker copy and execute a crafted font in Windows system to raise the user privilege as super user



Remote Font Attack Vector

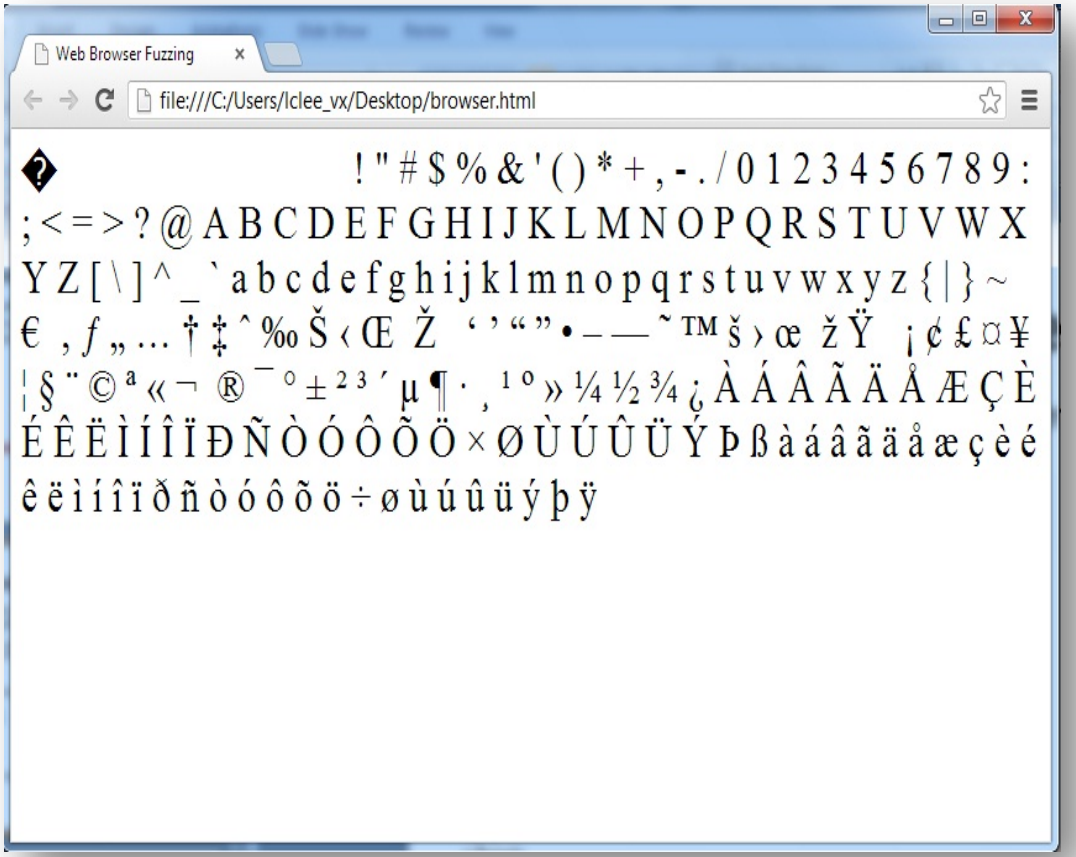
- The font vulnerability could allow remote code execution if the victim opens the crafted web page embedded with TrueType font



Remote Font Attack Vector

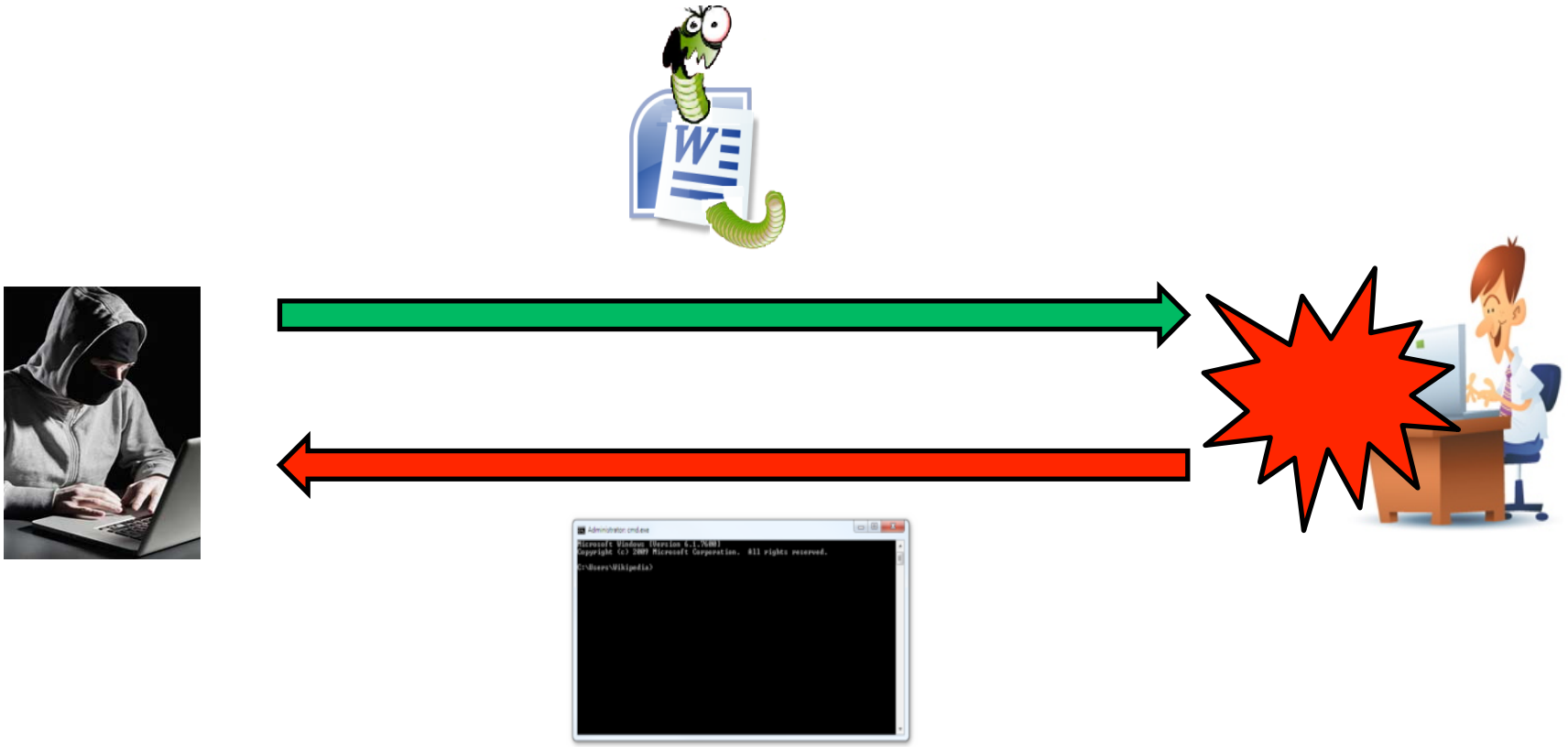
- The attacker can use CSS @font-face property to embed crafted TrueType font onto the web page

```
@font-face{
        font-family:“Crafted Font";
        src: url("sample.ttf") format('truetype');
        }
body{
    font-family:‘Crafted Font';
     font-size:30;
    font-style:normal;
    font-weight=bold;
    font-stretch=0;
   }
</style>
</head>
<body>
    &#0; &#1; &#3; &#4; &#5; &#6;
    [……]
</body>
```



Remote Font Attack Vector

- The font vulnerability could allow remote code execution if the victim opens the crafted Microsoft Office Word file(*.docx)

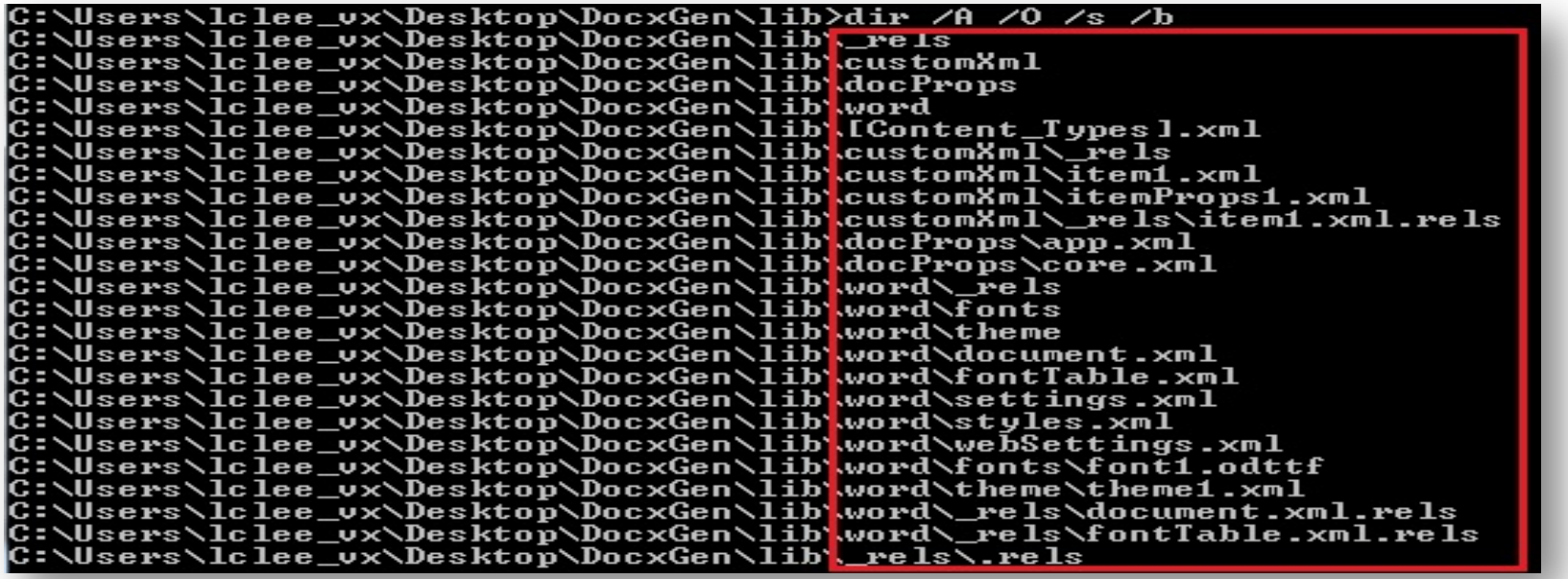


Remote Font Attack Vector

- ODTTF is an embedded font file type used in Microsoft Office XML format and Microsoft's XML Paper Specification Format(XPS)
- Embedded font obfuscation prevents end-users from using standard ZIP utilities to extract fonts from OpenXPS or Office document files and install them on the systems
- To perform embedded font obfuscation, a 128 bit (16 bytes) GUID (Globally Unique Identifier) is generated for the font to be obfuscated

Remote Font Attack Vector

- Microsoft Office 2007 is adopted an XML-based file format for Excel 2007, Word 2007 and PowerPoint 2007
- The new file format called Office Open XML Format improve file and data management, data recovery and extend the support with the earlier versions



Remote Font Attack Vector

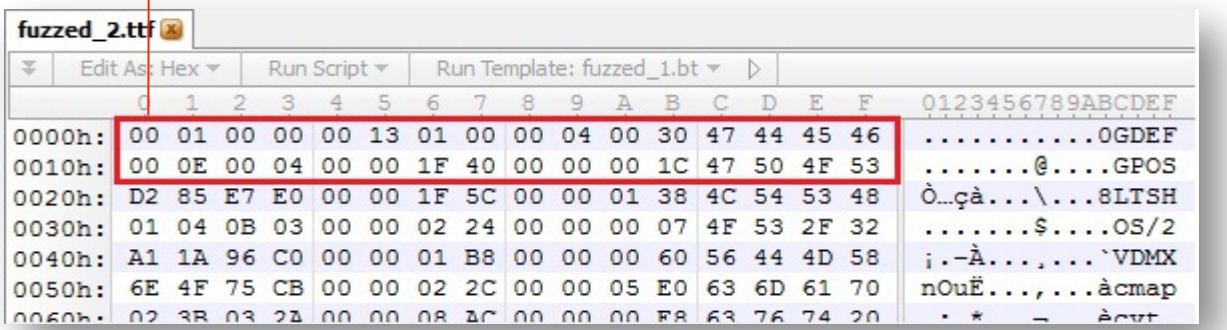
- Editing components of a document in the XML to force the Office Word use the crafted obfuscated TrueType Font (*.odttf)
- /~path~/Office Word/word/fontTable.xml, we set the Globally Unique Identifier (GUID) value in "w:fontKey"
- Perform an XOR operation on the first 32 bytes of the binary data of the font with the generated GUID Key

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:fonts xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
    <w:font w:name="Droid">
        <w:panose1 w:val="00000000000000000000"/>
        <w:charset w:val="00"/>
        <w:family w:val="auto"/>
        <w:pitch w:val="variable"/>
        <w:sig w:usb0="00000003" w:usb1="00000000" w:usb2="00000000" w:usb3="00000000" w:csb0="00020001" w:csb1="00000000"/>
        <w:embedRegular r:id="rId2" w:fontKey="{F81ED817-36AD-4C71-88D3-E5A62FBAF834}"/>
    </w:font>
</w:fonts>
```

Remote Font Attack Vector

. Key[0] fontKey[i] ;i=0,1,2,……15 Key[15]

F8	1E	D8	17	36	AD	4C	71	88	D3	E5	A6	2F	BA	F8	34



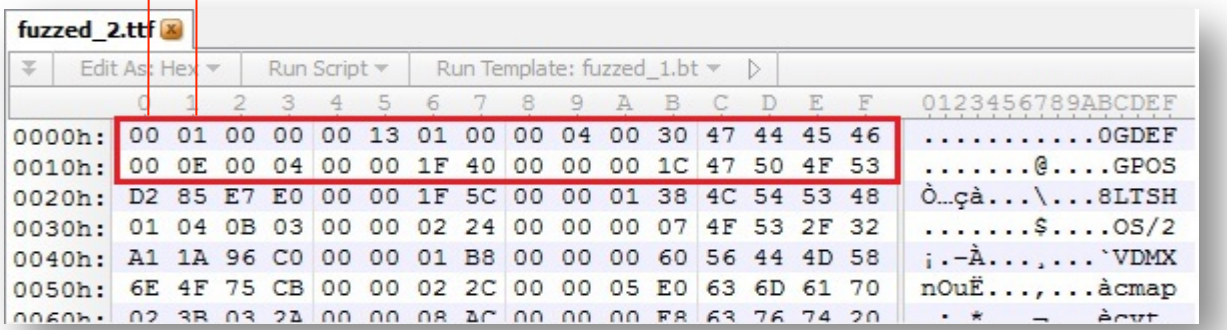
TrueType Font

```
fontKey = keys.decode("hex")
obfFontString = open(ttfFontFile, 'rb').read()
fontString = [ord(x) for x in obfFontString]
for i in range(16):
        fontString[i] = ord(obfFontString[i]) ^ ord(fontKey[15-i])
        fontString[i+16] = ord(obfFontString[i+16]) ^ ord(fontKey[15-i])
```

Remote Font Attack Vector

Key[0] fontKey[i] ;i=0,1,2,……15 Key[15]

F8	1E	D8	17	36	AD	4C	71	88	D3	E5	A6	2F	BA	F8	34



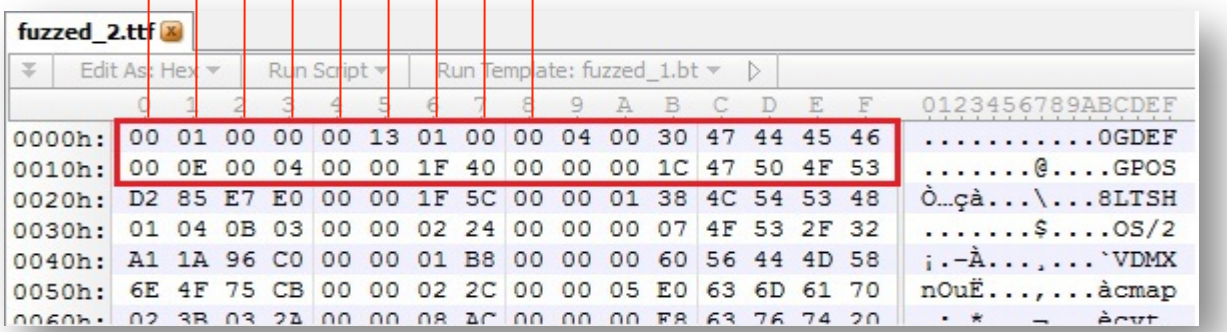
TrueType Font

```
fontKey = keys.decode("hex")
obfFontString = open(ttfFontFile, 'rb').read()
fontString = [ord(x) for x in obfFontString]
for i in range(16):
        fontString[i] = ord(obfFontString[i]) ^ ord(fontKey[15-i])
        fontString[i+16] = ord(obfFontString[i+16]) ^ ord(fontKey[15-i])
```

Remote Font Attack Vector

Key[0] fontKey[i] ;i=0,1,2,……15 Key[15]

F8	1E	D8	17	36	AD	4C	71	88	D3	E5	A6	2F	BA	F8	34



TrueType Font

```
fontKey = keys.decode("hex")
obfFontString = open(ttfFontFile, 'rb').read()
fontString = [ord(x) for x in obfFontString]
for i in range(16):
        fontString[i] = ord(obfFontString[i]) ^ ord(fontKey[15-i])
        fontString[i+16] = ord(obfFontString[i+16]) ^ ord(fontKey[15-i])
```

Remote Font Attack Vector

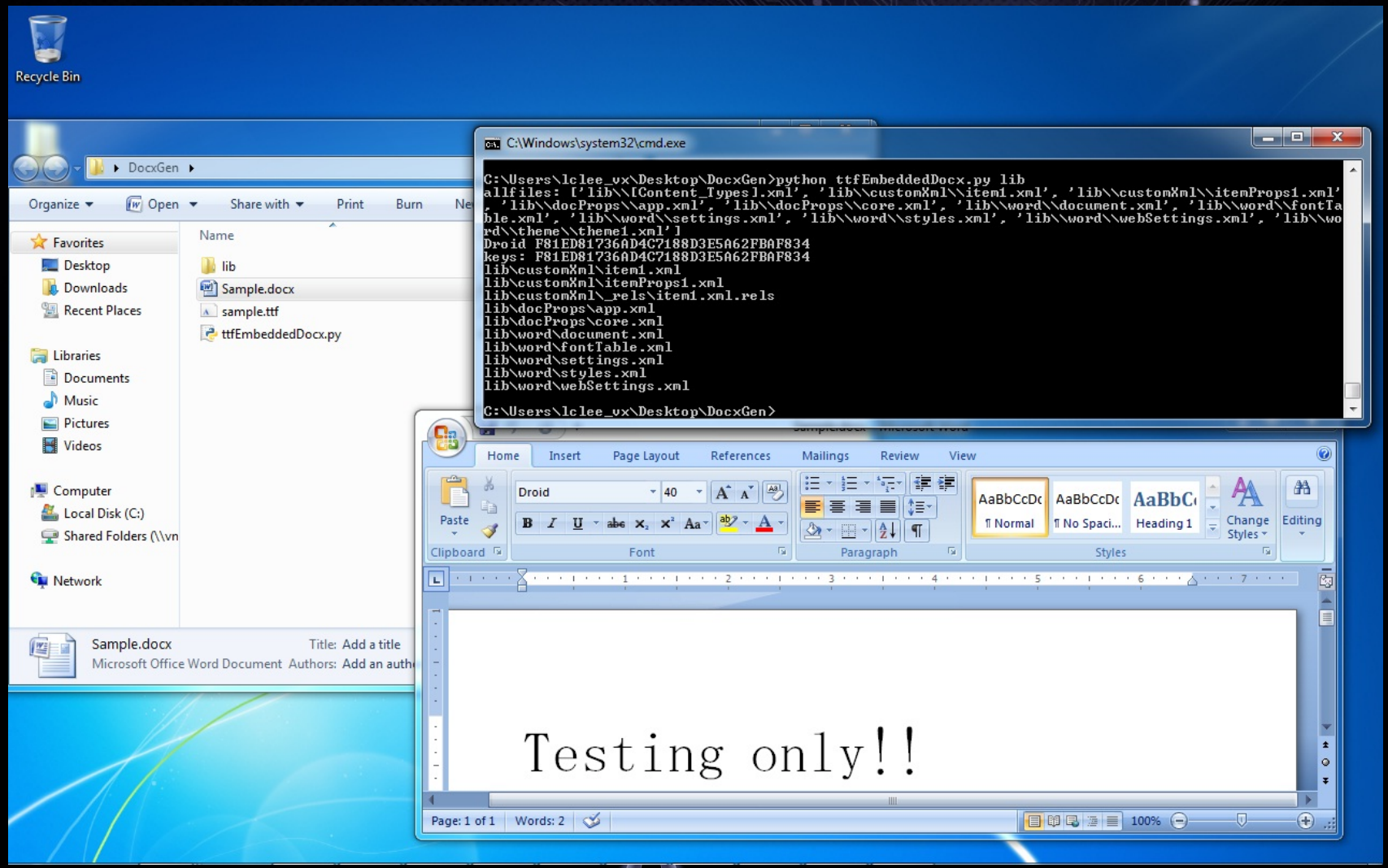
- /~path~/Office Word/word/document.xml, we defined the value of “w:rFonts w:ascii”, “w:hAnsi”, “w:sz w:val”, “w:szCs w:val”

```
<w:body>
    <w:p w:rsidR="000B1ED5" w:rsidRPr="00E75B86" w:rsidRDefault="00E75B86">
        <w:pPr>
            <w:rPr>
                <w:rFonts w:ascii="MyFont" w:hAnsi="MyFont"/>
            </w:rPr>
        </w:pPr>
        <w:r w:rsidRPr="00E75B86">
            <w:rPr>
                <w:rFonts w:ascii="MyFont" w:hAnsi="MyFont"/>
                <w:sz w:val="80" />
                <w:szCs w:val="80" />
            </w:rPr>
            <w:t>
            &#112;&#113;
            </w:t>
        </w:r>
    </w:p>
    <w:sectPr w:rsidR="000B1ED5" w:rsidRPr="00E75B86" w:rsidRSect="000B1ED5">
        <w:pgSz w:w="12240" w:h="15840"/>
        <w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440" w:header="720" w:footer="720" w:gutter="0"/>
        <w:cols w:space="720"/>
        <w:docGrid w:linePitch="360"/>
    </w:sectPr>
</w:body>
```

This element specifies a font which shall be used to format all characters

This element specifies the font size. The element’s val attribute are expressed as half-point values

Remote Font Attack Vector



TTF Bugs

TrueType Font Bugs

- Microsoft has provided “exploitable” Windows debugging extension for WinDBG, a powerful automated crash analysis and security risk assessment tool
- The tool was created by the MSEC Security Team and the latest version 1.6.0 supported ARM crash dump
- Based on our experience, the tool doesn’t much help in Windows kernel crash dump
- Use “!analyze –v “ and manual tracing kungfu “ph”, “th”
- VirtualKD is cool and improves your kernel debugging performance with VMWare / VirtualBox , plus it is supported Windows 8 now!!

Bug #1 (CVE-2013-3129)

```
.......
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 50, {9079d004, 1, 8f4a453e, 0}

Probably caused by : win32k.sys ( win32k!AddVertSmartScan+5b )

Followup: MachineOwner
---------

nt!RtlpBreakWithStatusInstruction:
81145304 cc              int     3
kd> !load winext\msec.dll
kd> !exploitable

!exploitable 1.6.0.0
Warning: Unable to read from the TEB in the current thread.
Warning: Unable to read from the TEB in the current thread.
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - Write Access Violation in Kernel Memory starting at nt!RtlpBreakWithStatusInstruction+0x0000000

All kernel mode write access violations in kernel memory are exploitable.
```

???

Bug #1 (CVE-2013-3129)

```
kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced.  This cannot be protected by try-except,
it must be protected by a Probe.  Typically the address is just plain bad or it
is pointing at freed memory.
Arguments:
Arg1: 9079d004, memory referenced.
Arg2: 00000001, value 0 = read operation, 1 = write operation.
Arg3: 8f4a453e, If non-zero, the instruction address which referenced the bad memory
      address.
Arg4: 00000000, (reserved)

Debugging Details:
------------------


WRITE_ADDRESS:  9079d004 Paged session pool

FAULTING_IP:
win32k!AddVertSmartScan+5b
8f4a453e 668930          mov     word ptr [eax],si

MM_INTERNAL_CODE:  0

IMAGE_NAME:  win32k.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  51633932

MODULE_NAME: win32k

FAULTING_MODULE: 8f42a000 win32k

DEFAULT_BUCKET_ID:  WIN8_DRIVER_FAULT

BUGCHECK_STR:  AV

PROCESS_NAME:  csrss.exe
```

Bug #2

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 7F, {8, 80f94000, 0, 0}

Probably caused by : win32k.sys ( win32k!itrp_CALL+8 )

Followup: MachineOwner
---------

nt!RtlpBreakWithStatusInstruction:
812f1304 cc              int     3
kd> !load winext\msec.dll
kd> !exploitable

!exploitable 1.6.0.0
Warning: Unable to read from the TEB in the current thread.
Exploitability Classification: UNKNOWN
Recommended Bug Title: BugCheck starting at nt!RtlpBreakWithStatusInstruction+0x0000000000000000 (Hash=0xd12ccb1a.0x712c

A BugCheck was detected, but no further information about the severity could be determined.
```

Instruction virtual machine??

???

Bug #2

```
*                                                                              *
*                        Bugcheck Analysis                                     *
*                                                                              *
********************************************************************************

UNEXPECTED_KERNEL_MODE_TRAP (7f)
This means a trap occurred in kernel mode, and it's a trap of a kind
that the kernel isn't allowed to have/catch (bound trap) or that
is always instant death (double fault).  The first number in the
bugcheck params is the number of the trap (8 = double fault, etc)
Consult an Intel x86 family manual to learn more about what these
traps are. Here is a *portion* of those codes:
If kv shows a taskGate
        use .tss on the part before the colon, then kv.
Else if kv shows a trapframe
        use .trap on that value
Else
        .trap on the appropriate frame will show where the trap was taken
        (on x86, this will be the ebp that goes with the procedure KiTrap)
Endif
kb will then show the corrected stack.
Arguments:
Arg1: 00000008, EXCEPTION_DOUBLE_FAULT
Arg2: 80f94000
Arg3: 00000000
Arg4: 00000000

Debugging Details:
------------------


BUGCHECK_STR:  0x7f_8

TSS:  00000028 -- (.tss 0x28)
eax=8f8eb5b3 ebx=a0a87cf8 ecx=a0a8844e edx=0000002b esi=a0a8844e edi=a0a870c0
eip=8f8eb5bb esp=8b728ff4 ebp=8b72901c iopl=0         nv up ei ng nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010282
win32k!itrp_CALL+0x8:
8f8eb5bb 56              push    esi
Resetting default scope

DEFAULT_BUCKET_ID:  WIN8_DRIVER_FAULT

PROCESS_NAME:  csrss.exe
```

Bug #3

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 7F, {8, 81136000, 0, 0}

Probably caused by : win32k.sys ( win32k!itrp_MDRP+31 )

Followup: MachineOwner
---------

nt!RtlpBreakWithStatusInstruction:
8134a304 cc              int     3
kd> !load winext\msec.dll
kd> !exploitable

!exploitable 1.6.0.0
Warning: Unable to read from the TEB in the current thread.
Exploitability Classification: UNKNOWN
Recommended Bug Title: BugCheck starting at nt!RtlpBreakWithStatusInstruction+0x0000000000000000 (Hash=0xd12ccb1a.0x5d95

A BugCheck was detected, but no further information about the severity could be determined.
```

Instruction virtual machine??

???

Bug #4

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 7F, {8, 812d6000, 0, 0}

Probably caused by : win32k.sys ( win32k!itrp_DIV+6e )

Followup: MachineOwner
---------

nt!RtlpBreakWithStatusInstruction:
81557304 cc              int     3
kd> !load winext\msec.dll
kd> !exploitable

!exploitable 1.6.0.0
Warning: Unable to read from the TEB in the current thread.
Exploitability Classification: UNKNOWN
Recommended Bug Title: BugCheck starting at nt!RtlpBreakWithStatusInstruction+0x0000000000000000 (Hash=0xd12ccb1a.0x24fa

A BugCheck was detected, but no further information about the severity could be determined.
```

Instruction virtual machine??

So What??!!

- Many crash dump happened in our fuzzing process. Now what we concern is, exploitable? Or able to control EIP?? Or just BSOD??
- In bug #1, we fuzzed on GLYPH table in TrueType Font files.
- For bug#2, bug#3, bug#4, we fuzzed on FPGM table in TrueType Font files
- Some bugs very interesting and able to embed into Microsoft Office Word or browser to launch the remote Windows kernel attack
- Some bugs just didn't work once the font is embed into Office Word or browser

Demonstration

Some Notes…

- Delete the glyph from the TrueType Font sample and concentrate on a few glyph before you start to fuzz, or else the reverse engineering or root cause finding process going to kill you!!
- Two TrueType font tests need to include:
  - Open the TrueType Font file using FontView.exe
  - Calling the glyph index from character map and display the text in different size
- Do not fuzz and display the text start at font size 0, Microsoft Office does not accept font size 0

Some Notes…

- Many open source tool help you on TrueType Font fuzzing:
  - FontForge
  - Microsoft Typography Tools – FontTools.exe
  - 010 Binary Editor
  - VirtualKD
- Many commercial ($$) tool help you on TrueType Font editing:
  - FontLab Studio 5
  - Fontographer
  - BitFonter
- Happy TTF Fuzzing!!

Thanks