



**SIGNALSEC**  
BEYOND INTELLIGENCE

# BREAKING FONT PARSERS

CELIL UNUVER – BEKIR KARUL

- Celil UNUVER @celilunuver
  - Founder and Researcher @SignalSEC Ltd.
  - Organizer of NOPcon Hacker Conference
  - Vuln Research, RE, Fuzzing and SCADA
  - Speaker at CODE BLUE, CONFidence 2010, Swiss Cyber Storm, DefCamp
- Bekir KARUL @bek\_phys
  - Researcher @SignalSEC Ltd.
  - Windows Internals, Malware, RE
  - bekirkarul.com

- I. Introduction to Fonts
- II. TTF Structure
- III. Kirlangıç TTF Fuzzer
- IV. Generating test cases with fuzzed TTF
- V. Fuzzing Results
- VI. Questions



- Applications can use four different kinds of GDI font technologies to display and print text
  1. Raster
  2. Vector
  - 3. TrueType**
  4. Microsoft OpenType
- The GDI(Graphis Device Interface ) is part of the core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers.
- The differences between these fonts reflect the way that the glyph for each character or symbol is stored in the respective font-resource file

**Raster** : a glyph is a bitmap that the system uses to draw a single character in the font.

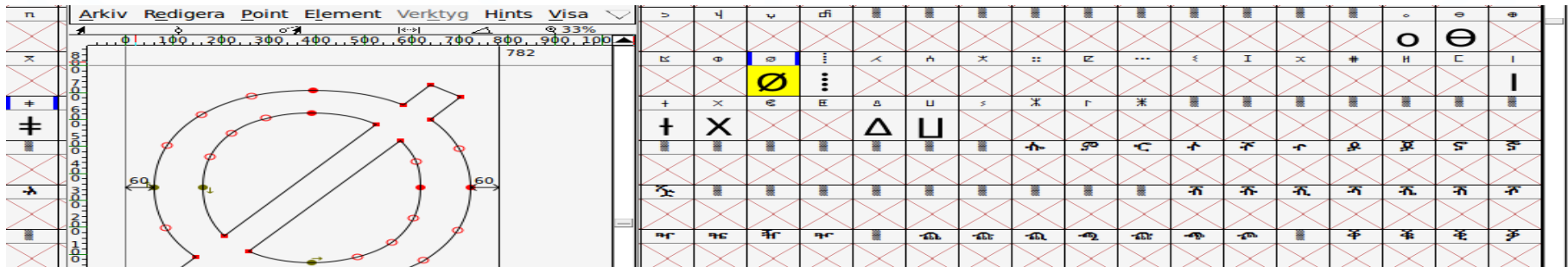
**Vector** : a glyph is a collection of line endpoints that define the line segments that the system uses to draw a character in the font.

**TrueType & OpenType** : a glyph is a collection of line and curve commands as well as a collection of hints.

[http://msdn.microsoft.com/en-us/library/dd162893\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/dd162893(v=vs.85).aspx)

# TrueType Font

- A TrueType font file contains data in table format, that comprises an outline font
- And the raster device uses combinations of data from different tables to render the glyph data in the font.
- You can use TrueTypeViewer, FontForge and TTFDump as font editor/viewer



# TrueType Font Structure

- 010 Editor is probably best choice for viewing/editing files
- There are many file format templates for 010 Editor including TTF!
- We modified 010 Editor's TTF template a bit for readability, here is example

**YourHandwriting.ttf** [icon]

Edit As: Hex    Run Script    Run Template: TTFTemplate.bt    ▶

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	00	01	00	00	00	0B	00	80	00	03	00	30	4F	53	2F	32	.....€...0OS/2
0010h:	3D	ED	7C	58	00	00	01	38	00	00	00	4E	63	6D	61	70	=i X...8...Ncmap
0020h:	E3	B3	43	8B	00	00	03	10	00	00	00	CE	67	61	73	70	ã³C<.....îgasp
0030h:	FF	FF	00	03	00	00	45	C4	00	00	00	08	67	6C	79	66	ÿÿ...EÄ....glyf
0040h:	10	14	1E	DC	00	00	04	A8	00	00	3C	A0	68	65	61	64	...Ü....~...< head
0050h:	DE	EB	9A	75	00	00	00	BC	00	00	00	36	68	68	65	61	Đěšů...¼....6hhea
0060h:	0D	46	04	CE	00	00	00	F4	00	00	00	24	68	6D	74	78	.F.Î...ô...\$hmtx
0070h:	3D	0D	11	BB	00	00	01	88	00	00	01	88	6C	6F	63	61	=..»....^....^loca
0080h:	07	10	F7	DA	00	00	03	E0	00	00	00	C6	6D	61	78	70	..÷Ú...à...Æmaxp
0090h:	00	CC	00	D8	00	00	01	18	00	00	00	20	6E	61	6D	65	.İ.Ø..... name

Template Results - TTFTemplate.bt

Name	Value	Start
▼ struct FontOffsetTable OffsetTable		0h
TT_Fixed SFNT_Ver	65536	0h
USHORT numTables	11	4h
USHORT searchRange	128	6h
USHORT entrySelector	3	8h
USHORT rangeShift	48	Ah
▼ struct FontTableDirectory Table[11]		Ch
> struct FontTableDirectory Table[0]	OS/2 (1330851634) at 312 for 78	Ch
> struct FontTableDirectory Table[1]	cmap (1668112752) at 784 for 206	1Ch

<http://www.sweetscape.com/010editor/templates/>

## TrueType Font Structure

- TTF font begins at byte 0 with the Font Offset Table
- Font Offset Table have 5 subtables:

SFNT\_Ver : Version information 65536 for version 1.0  
numTables : Number of tables  
searchRange : (Maximum power of 2  $\leq$  numTables) x 16  
entrySelector : Log2(maximum power of 2  $\leq$  numTables)  
rangeShift : numTables x 16-searchRange

▼ struct FontOffsetTable OffsetTable	
TT_Fixed SFNT_Ver	65536
USHORT numTables	11
USHORT searchRange	128
USHORT entrySelector	3
USHORT rangeShift	48



# TrueType Font Structure

- This table followed at byte 12 by the Table Directory entries
- Each entry has 4 members
- And Table Directory entries must be sorted in ascending order by tag

tag : 4 byte identifier  
checkSum : 4 byte checksum value of this table  
offset : Offset from beginning  
length : Length of this table

▼ struct FontTableDirectory Table[7]	loca (1819239265) at 992 for 198	7Ch	10h
> union Tag		7Ch	4h
ULONG checkSum	118552538	80h	4h
ULONG offset	992	84h	4h
ULONG length	198	88h	4h

- Important to say that we should NEVER fuzz these fields : checkSum, offset, length

- What is table checksum?

- Table checksums are the unsigned sum of a given table.
- We can calculate it by this little python code

```
totalData = 0
for i in range(0, len(tableD), 4):
    data = unpack(">I", tableD[i:i+4] ) [0]
    totalData += data
finalData = 0xffffffff & totalData
```

- This function implies that the length of a table must be multiple of four bytes.  
If not, you need to fill remaining with zeros

```
#Check if we need to adjust table lenght (When len(table) % 4 != 0)
fourFix = False
if len(tableData) % 4 != 0:
    fourFix = True
    fixedSize = ((len(tableData) + 3) & ~3) - len(tableData)
    tempTableData = tableData + (fixedSize * "0")
```

- Example

0030h:	FF FF 00 03	00 00 45 C4	00 00 00 08	67 6C 79 66	ÿÿ....EÄ....glyf
0040h:	10 14 1E DC	00 00 04 A8	00 00 3C A0	68 65 61 64	...Ü....~...< head
0050h:	DE EB 9A 75	00 00 00 BC	00 00 00 36	68 68 65 61	þěšü...4...6hhea
0060h:	0D 46 04 CE	00 00 00 F4	00 00 00 24	68 6D 74 78	.F.Î...ô...\$hmtx
0070h:	3D 0D 11 BB	00 00 01 88	00 00 01 88	6C 6F 63 61	=...»...^...^loca
0080h:	07 10 F7 DA	00 00 03 E0	00 00 00 C6	6D 61 78 70	..÷Ú...à...\$maxp

Template Results - TTTemplate.bt

Name	Value	Start
> struct FontTableDirectory Table[2]	gasp (1734439792) at 17860 for 8	2Ch
> struct FontTableDirectory Table[3]	glyf (1735162214) at 1192 for 15520	3Ch
▼ struct FontTableDirectory Table[4]	head (1751474532) at 188 for 54	4Ch
> union Tag		4Ch
ULONG checksum	3739982453	50h
ULONG offset	188	54h
ULONG length	54	58h
> struct FontTableDirectory Table[5]	hhea (1751672161) at 244 for 36	5Ch
> struct FontTableDirectory Table[6]	hmtx (1752003704) at 392 for 392	6Ch
> struct FontTableDirectory Table[7]	loca (1819239265) at 992 for 198	7Ch
> struct FontTableDirectory Table[8]	maxp (1835104368) at 280 for 32	8Ch
> struct FontTableDirectory Table[9]	name (1851878757) at 16712 for 909	9Ch
> struct FontTableDirectory Table[10]	post (1886352244) at 17624 for 236	ACh
> struct thead head		BCh
> struct thhea hhea	v1.00 98 hmtx records	F4h
> struct thmtx hmtx	98 HMetrics 0 leftSideBearing	188h

- Also we have another checksum for entire font
- To compute it:
  1. Set **checksumAdjustment** to 0 in **head** table
  2. Calculate the checksum for all the tables and enter that value into the table directory
  3. Calculate the checksum for the entire font
  4. Calculate  $0xB1B0AFBA - \text{sum}$
  5. Store result in **checksumAdjustment**

```
for i in range(0, len(tempFont), 4):
    data = unpack(">I", tempFont[i:i+4]) [0]
    totalData += data

finalData = totalData & 0xffffffff
finalData = (0xb1b0afba - finalData)
```

# TrueType Font Structure

- There are 2 types of tables

## 1. Required

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 Structure

- There are 2 types of tables

## 2. Optional

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 (grayscale)
hdmx	horizontal device metrics
kern	kerning
LTSH	Linear threshold table
prep	CVT Program
PCLT	PCL5
VDMX	Vertical Device Metrics table
vhea	Vertical Metrics header
vmtx	Vertical Metrics

Note that the number 0 is never a valid tag name.

- TTF structure is designed to keep the entire glyph data in various table
- Three new tables are used to embed bitmaps in TrueType fonts
  - I. EBDT – for embedded bitmap data
  - II. EBLC – for embedded bitmap locators
  - III. EBSC – for embedded bitmap scaling information
- The raster device which we mentioned previously generally uses these tables to render the glyph data

- What is name table?

- The naming table allows multilingual strings to be associated with the TrueType font file. These strings can represent copyright notices, font names, family names, style names, and so on.

The format of Naming Table is organized as follows:

Type	Name	Description
USHORT	format	Format selector (=0).
USHORT	count	Number of name records.
USHORT	stringOffset	Offset to start of string storage (from start of table).
NameRecord	nameRecord[count]	The name records where <i>count</i> is the number of records.
(Variable)	Storage for the actual string data.	

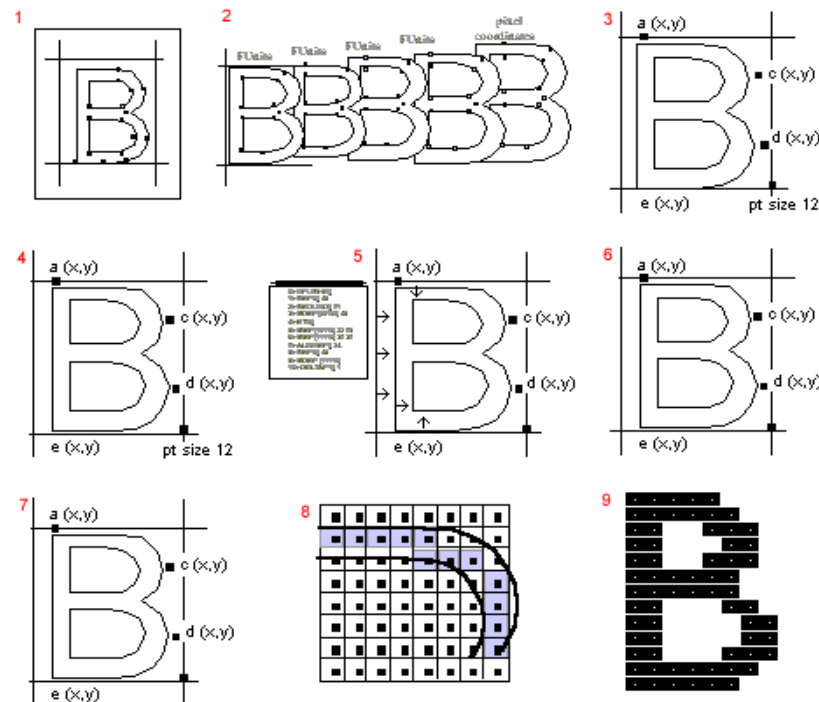
## Name Records

Each *NameRecord* looks like this:

Type	Name	Description
USHORT	platformID	Platform ID.
USHORT	encodingID	Platform-specific encoding ID.
USHORT	languageID	Language ID.
USHORT	nameID	Name ID.
USHORT	length	String length (in bytes).
USHORT	offset	String offset from start of storage area (in bytes).



- Anybody font designer here?
- So we do not need further examination of boring font structure



More information about TTF : TrueType 1.0 Font Files Technical Specification (1995)

```
E:\BOCUK\Projects\Kirlangic>kirlangic.py -h
usage: kirlangic.py [-h] [-i] [-fuzzvalue ff] [-o fuzzedfiles]
                  [-od fuzzeddocx] [-op fuzzedpdf] [-m 10000] [-docx] [-pdf]

Kirlangic TTF Fuzzer - v1.2
SIGNALSEC

optional arguments:
  -h, --help            show this help message and exit
  -i                    ttf input
  -fuzzvalue ff         fuzz value
  -o fuzzedfiles        fuzzed files output dir
  -od fuzzeddocx        docx output dir
  -op fuzzedpdf         pdf output dir
  -m 10000              max lenght of table
  -docx                 create docx files
  -pdf                  create pdf files

Usage:
./kirlangic.py -i ttfFile.ttf
```

# KIRLANGIC TTF FUZZER

- Created to fuzz TTF font and generate test cases with fuzzed TTF files
- Byte-flipping method
- It's structure aware (can fix table checksums etc.)
- It can generate Doc, Java, Silverlight, PDF test cases
- Will be public after the talk (<https://github.com/signalsec>)



```
[!] Number of tables: 11
```

No	Table	Checksum	Offset	Lenght
00	OS/2	3ded7c58	00000138	0000004e
01	cmap	e3b3438b	00000310	000000ce
02	gasp	ffff0003	000045c4	00000008
03	glyf	10141edc	000004a8	00003ca0
04	head	deeb9a75	000000bc	00000036
05	hhea	0d4604ce	000000f4	00000024
06	hmtx	3d0d11bb	00000188	00000188
07	loca	0710f7da	000003e0	000000c6
08	maxp	00cc00d8	00000118	00000020
09	name	8cacc3aa	00004148	0000038d
10	post	78977afa	000044d8	000000ec

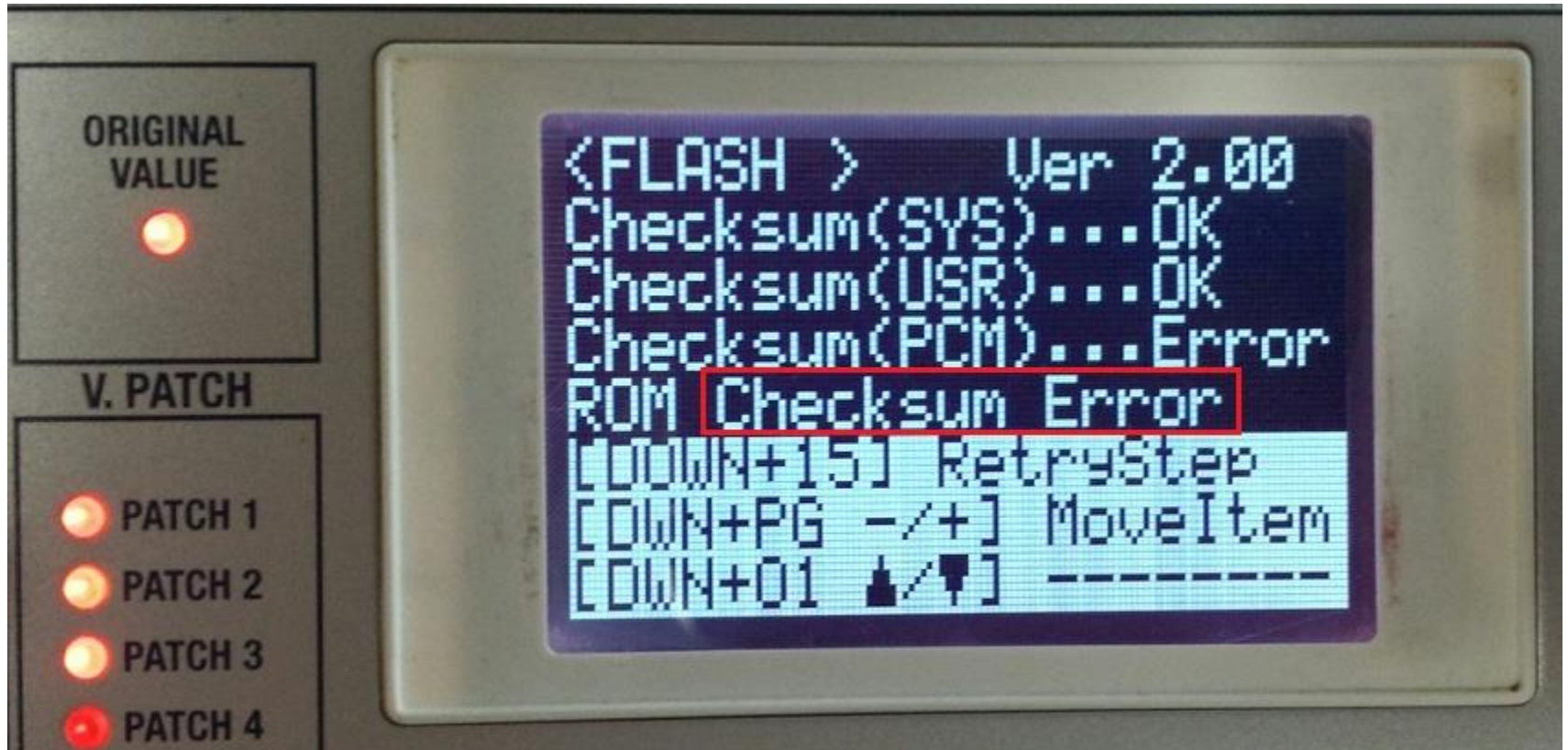
- We can use 2 types of fuzzing in TTF files

### 1. Dumb Fuzzing

\*Simply alter TTF file without awareness of its data structure



- Dumb Fuzzing Sucks in TTF



## 2. Smart Fuzzing

\*Simply alter TTF file with awareness of its data structure. In this type of fuzzing, we will not alter structure members like **checksum**, table name, offset etc..

- Fuzz process
  1. Get table information for each table(Table name, Checksum, Offset, Offset length)
  2. Fuzz it and generate fuzzed TTF files
  3. Do it again for other tables
  4. Generate test cases
  5. Send it to God damn font parsers



To sum up here is our pseudo code of fuzz process:

```
for i=1; i<= Number of tables; i++:  
    get table information here  
    .....  
    fuzzit(TTF File, Table Offset, Table length, Table no)  
  
fuzzit:  
    table Data = TTF File[Table Offset:Table Offset+Table length]  
    table Data Hex = table Data.encode("hex")  
  
    while start < table length:  
        table Data Hex[start] = `FUZZ DATA`  
        table Fuzzed Data = Table Data.decode("hex")  
        create TTF File(TTF File, tableFuzzedData, start, Table no, table offset)  
        start++
```

# Kırlangıç TTF Fuzzer

- Here is an example fuzzed table; «naming» table

Startup		table16offset103.ttf			
Edit As: Hex		Run Script		Run Template: TTFTemplate.bt	
		0 1 2 3 4 5 6 7 8 9 A B C D E F		0123456789ABCDEF	
1:2050h:		00 00 00 00 00 08 00 0C 02 27 00 01 00 00 00 00		.....'.....	
1:2060h:		00 09 00 0C 02 4E 00 01 00 00 00 00 00 0A 00 38		.....N.....8	
1:2070h:		02 CD 00 01 00 00 00 00 00 12 00 08 03 18 00 03		.í.....	
1:2080h:		00 01 04 09 00 00 00 70 00 00 00 03 00 01 04 09		.....p.....	
1:2090h:		00 01 00 10 00 AB 00 03 00 01 04 09 00 02 00 0E		.....«.....	
1:20A0h:		00 C6 00 03 00 01 04 09 00 03 00 36 00 DE 00 03		.Æ.....6.ß..	
1:20B0h:		00 01 04 09 00 04 FF FF 01 32 00 03 00 01 04 09		.....ÿÿ.2.....	
1:20C0h:		00 05 00 1A 01 4D 00 03 00 01 04 09 00 06 00 10		.....M.....	
1:20D0h:		01 77 00 03 00 01 04 09 00 07 00 50 01 92 00 03		.w.....P.'..	
1:20E0h:		00 01 04 09 00 08 00 18 02 0D 00 03 00 01 04 09		.....	
1:20F0h:		00 09 00 18 02 34 00 03 00 01 04 09 00 0A 00 70		.....4.....p	
1:2100h:		02 5B 00 03 00 01 04 09 00 12 00 10 03 06 00 43		.[.....C	
1:2110h:		00 6F 00 70 00 79 00 72 00 69 00 67 00 68 00 74		.o.p.y.r.i.g.h.t	
1:2120h:		00 20 00 28 00 63 00 29 00 20 00 32 00 30 00 31		. .(c.) . .2.0.1	
1:2130h:		00 31 00 20 00 62 00 79 00 20 00 76 00 65 00 72		.1. .b.y. .v.e.r	
1:2140h:		00 6E 00 6F 00 6E 00 20 00 61 00 64 00 61 00 6D		.n.o.n. .a.d.a.m	
1:2150h:		00 73 00 2E 00 20 00 41 00 6C 00 6C 00 20 00 72		.s... .A.l.l. .r	
Template Results - TTFTemplate.bt					
Name		Value		Start	
struct tNameRecord tNameRecord[16]				120AEh	
USHORT platformID		3		120AEh	
USHORT encodingID		1		120B0h	
USHORT languageID		1033		120B2h	
USHORT nameID		4		120B4h	
USHORT length		65535		120B6h	
USHORT offset		306		120B8h	

## Target ?

- Windows Kernel is not the only target
- Java, Adobe Flash, Office, Silverlight etc.

Oracle Java TrueType LookupCount Buffer Overflow Remote Code Execution Vulnerability

**ZDI-14-038:** April 3rd, 2014

Vulnerability Alert

Oracle Java most TTF Table Remote Code Execution

Adobe Flash Player 11.3 Kern Table Parsing Integer Overflow

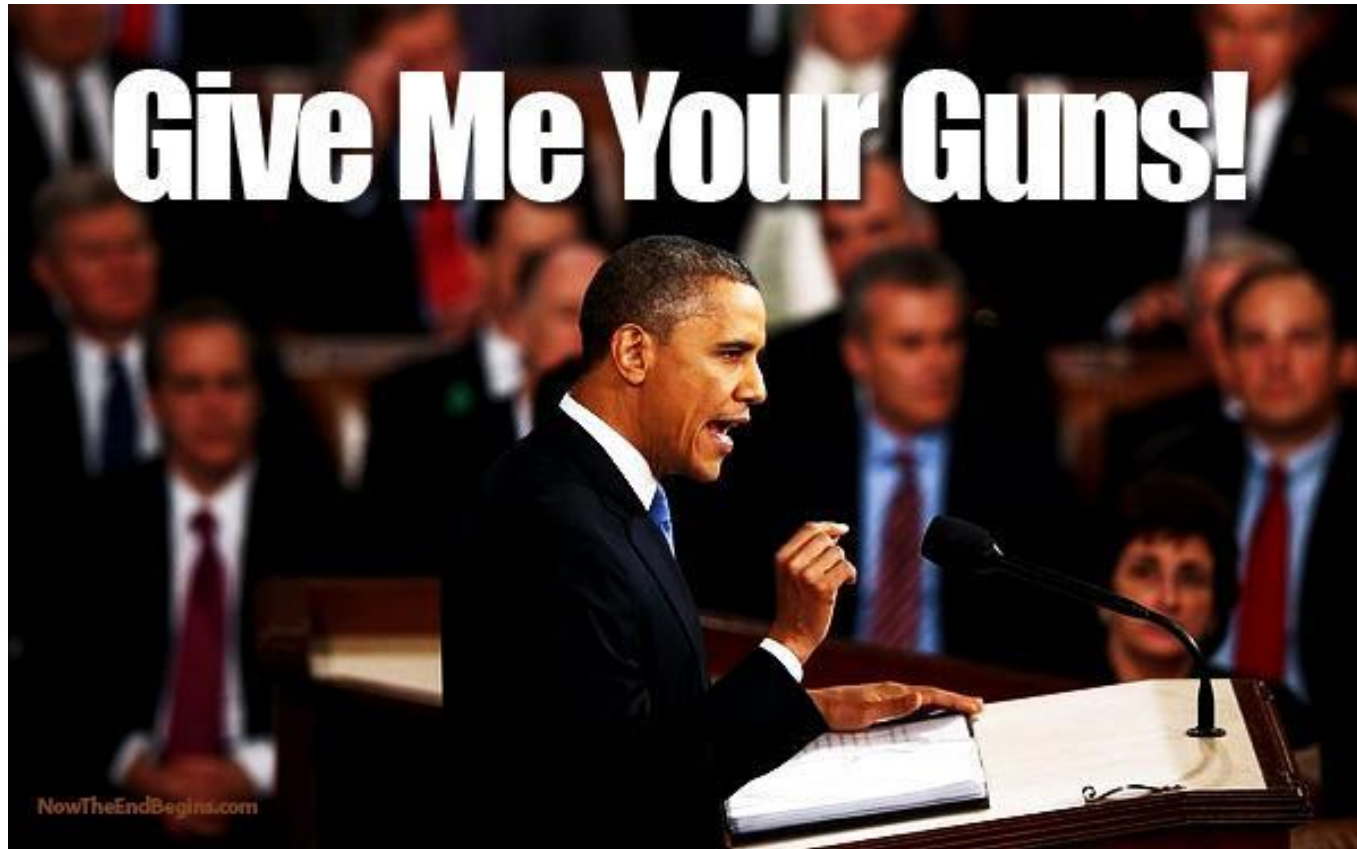


table10offset219.docx  
table10offset220.docx  
table10offset221.docx  
table10offset222.docx  
table10offset223.docx  
table10offset224.docx  
table10offset225.docx  
table10offset226.docx  
table10offset227.docx

```
----- Docx generating -----  
Font info:  
    Font: Dexter  
    Key : F81ED81736AD4C7188D3E5A62FBAF834  
  
[*] Fuzzed docx generating...  
[/] Total 335 fuzzed docx generated.  
  
[!] Terminated by the user...
```

How to create Office Documents with fuzzed TTF file?

## GENERATING TEST CASES WITH FUZZED TTF

- Generating office document



- Microsoft Office using a file format called Office Open XML Format since Office 2007.
- We can use any zip utility to open and modify its components.

# Generating Microsoft Office Documents

- How to generate ODTTF ?

- **ODTTF** is an embedded font file type used in Microsoft Office XML format.
- An obfuscation against font file is used to prevent extracting it from Office document and using it.
- To perform obfuscation, 16 byte GUID is generated for the each used font( in fontTable.xml )

```
<w:sig w:usb0="00000003" w:usb1="00000000" w:usb2="00000000" w:usb3="00000000" w:c
<w:embedRegular r:id="rId2" w:fontKey="{F81ED817-36AD-4C71-88D3-E5A62FBAF834}"/>
```

- Then, a XOR operation is performed on the first 32 bytes of the font with the generated GUID;

```
#Create new ODTTF
for i in range(16):
    origFontL[i] = ord(origFontR[i]) ^ ord(fontKey[15-i])
    origFontL[i+16] = ord(origFontR[i+16]) ^ ord(fontKey[15-i])
```

# Generating Microsoft Office Documents

- Then, defined the font and font size that will be used in the document ( in document.xml )

```
<w:pPr>
  <w:rPr>
    <w:rFonts w:ascii="Dexter" w:hAnsi="Dexter"/>
  </w:rPr>
</w:pPr>
<w:r w:rsidRPr="00E75B86">
  <w:rPr>
    <w:rFonts w:ascii="Dexter" w:hAnsi="Dexter"/>
    <w:sz w:val="8" />
    <w:szCs w:val="8" />
  </w:rPr>
  <w:t>:):):)</w:t>
</w:r>
</w:p>
```

- Notice w:sz and w:rFonts fields



## Generating Microsoft Office Documents

- And finally compress these files with *make\_archive* function (make\_archive function is from shutil module) and rename it document.zip to document.docx

```
#Move fuzzed odttf to docx template
move("font1.odttf", "docTemplate/word/fonts")

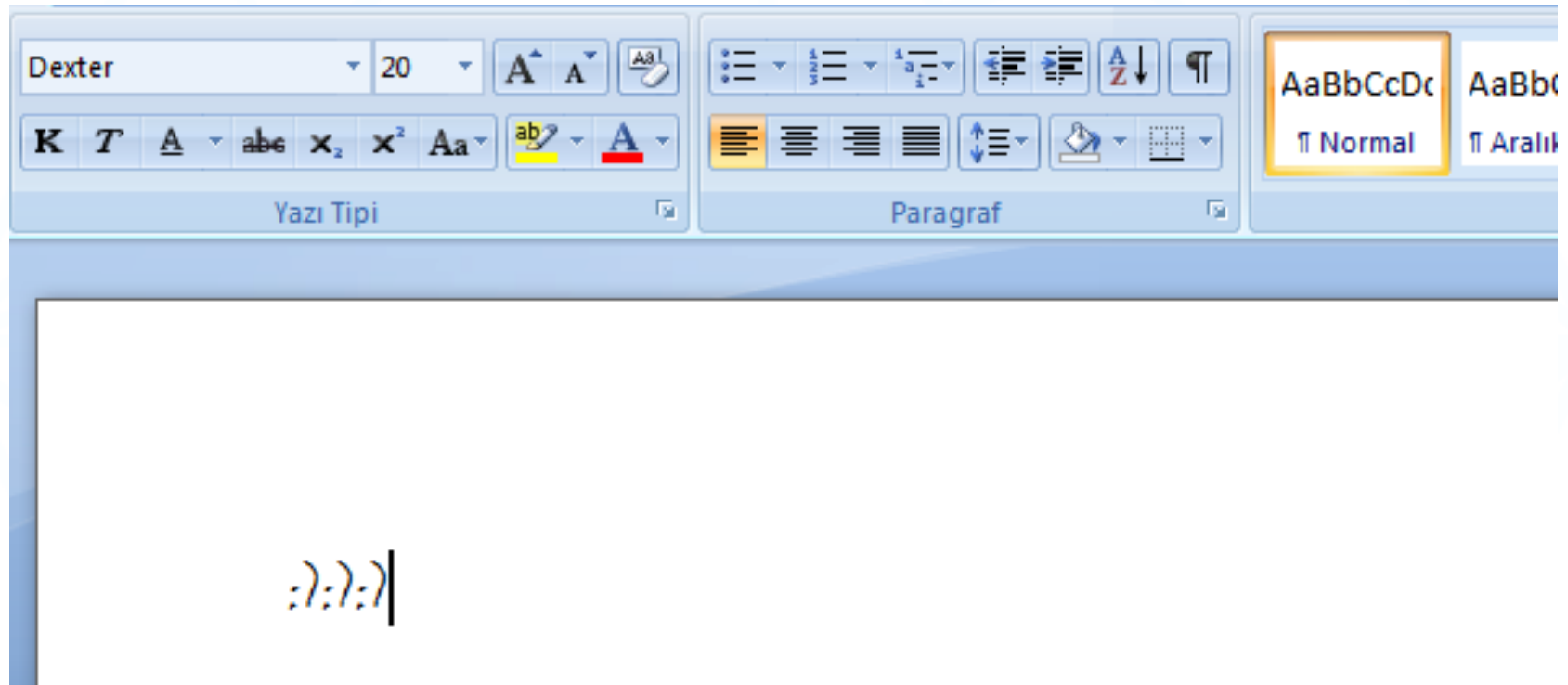
#Set fuzzed docx name
sDfile = tFile.split(".ttf")[0]

#Create Docx under fuzzeddocx
make_archive(args.od + "/" + sDfile, "zip", "docTemplate")
try:
    rename(args.od + sDfile + ".zip", args.od + sDfile + ".docx")
except WindowsError:
    #Already exist, remove it first
    remove(args.od + sDfile + ".docx")
    rename(args.od + sDfile + ".zip", args.od + sDfile + ".docx")
```

**P.S:** Do not forget to create new GUID for each new font you will use.

# Generating Microsoft Office Documents

- Result:



## Generating PDF

- We can use some modules to generate PDF testcases
  - **fpdf** module to create PDF files in a few line
  - Here you can see how we generate a pdf, add our fuzzed TTF into it, create a blank page and fill it with some data

```
def createPDF(tFile):
    #Set fuzzed pdf name
    sPfile = tFile.split(".ttf")[0]
    tempPDF = FPDF()
    #Add our fuzzed ttf into PDF
    try: tempPDF.add_font(sPfile, "", args.o + "/" + tFile, uni=True)
    except: return
    tempPDF.set_font(sPfile, "", 16)

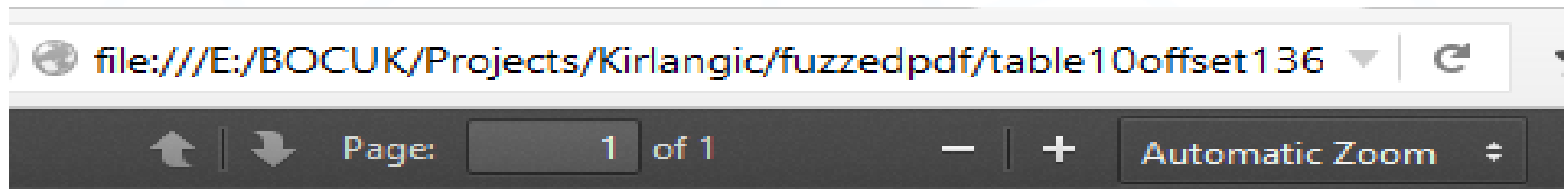
    #Create blank page and fill it with data
    tempPDF.add_page()
    tempPDF.cell(40, 10, "PDF TEST FILE")

    #Create fuzzed PDF
    try: tempPDF.output(args.op + sPfile + ".pdf", "F")
    except: return

    tempPDF.close()
```

**\*\*** Need to do a simple modification on FPDF core as it doesn't embed some corrupted fonts

- Result:



PDF TEST FILE

<https://pypi.python.org/pypi/fpdf>

# Generating Java Application

- We have 2 file to generate a Java application

## 1. FuzzJava.java

Our example java code. It uses **loadFont** to add our fuzzed TTF file into the application and use it with *Vbox->setStyle*.

```
Font.loadFont(  
    FuzzJavaTTF.class.getResource("__Dexter|.ttf__").toExternalForm(),  
    10  
);  
  
Label caption = new Label("kirlangiclar acayip guzel gulerler");  
caption.getStyleClass().add("caption");  
..  
..  
VBox layout = new VBox(10);  
layout.setStyle("-fx-padding: 20px; -fx-font-family: Dexter; -fx-background-color: white");  
..  
..  
final Scene scene = new Scene(layout);  
stage.setScene(scene);  
stage.show();
```

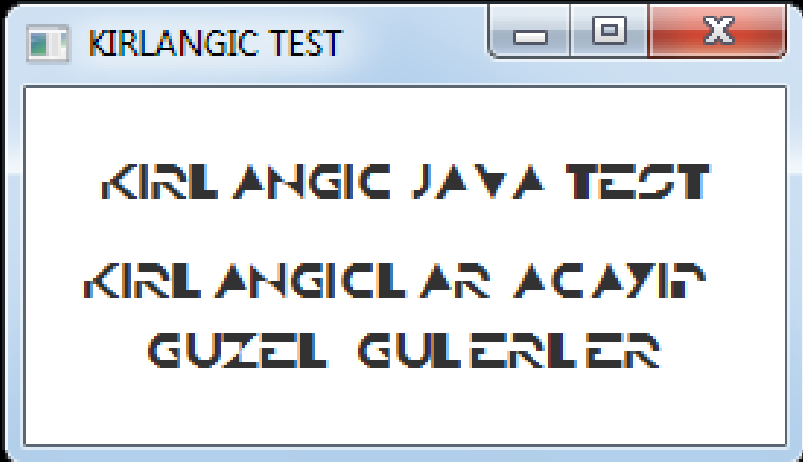
## 2. FuzzJava.py

Briefly it opens FuzzJava.java and update its content to change current used font to our fuzzed font. And finally compile it with java compiler.

```
casename = fuzzedTtf.split(".")[0]
newttf = "/fuzzedttf/" + fuzzedTtf
newdata = filedata.replace("dexter.ttf", newttf)
newdata = newdata.replace("FuzzJavaTTF", casename)
filename = "fuzzedjava/" + casename + ".java"
f = open(filename, 'w')
f.write(newdata)
f.close()
os.system("javac " + filename )
```

- Result:

```
Users\Bekir\Desktop\crashlog>javac FuzzJavaITF.java
Users\Bekir\Desktop\crashlog>java FuzzJavaITF
```



The screenshot shows a Windows-style window titled "KIRLANGIC TEST". The window contains three lines of text in a bold, stylized font: "KIRLANGIC JAVA TEST", "KIRLANGICLAR ACAYIP", and "GUZEL GULERLER". The text is displayed in a light blue color on a white background.

*for (i=0, i<ttf-amount, i++) {*

- Embed mutated TTFs to VS Silverlight Project
- Build the project with «MSBuild»
- Generate HTML files to call builded «XAP» silverlight applications

*}*



# Generating Silverlight Application

```
def MainPageXaml(fontFileName, fontFamilyName):
    with open("SilverlightApplication1\Master.xaml", 'r+') as rFile:
        readData = rFile.read()

        readData = readData.replace("""FontFamily="dexter.ttf#DexterC\""",
                                     """FontFamily="%s#%s\"""" % (fontFileName, fontFamilyName))
    with open("SilverlightApplication1\MainPage.xaml", 'w') as wFile:
        wFile.write(readData)
        wFile.close()

def CSProj(fileName):
    document = ET.parse('SilverlightApplication1\SilverlightApplication1.csproj')
    root = document.getroot()
    ET.register_namespace('', "http://schemas.microsoft.com/developer/msbuild/2003")

    for i in root.getchildren():
        for z in i.getchildren():
            if z.tag == '{http://schemas.microsoft.com/developer/msbuild/2003}Resource':
                z.attrib['Include'] = str(fileName)
                document.write('SilverlightApplication1\SilverlightApplication1.csproj')

def MSBuild():
    os.system("MSBuild.exe SilverlightApplication1.sln /p:Configuration=Release")
```

- No fuzzing farm! We performed fuzzing on our personal computers (4-6gb ram, i7 1.90 ghz etc.)

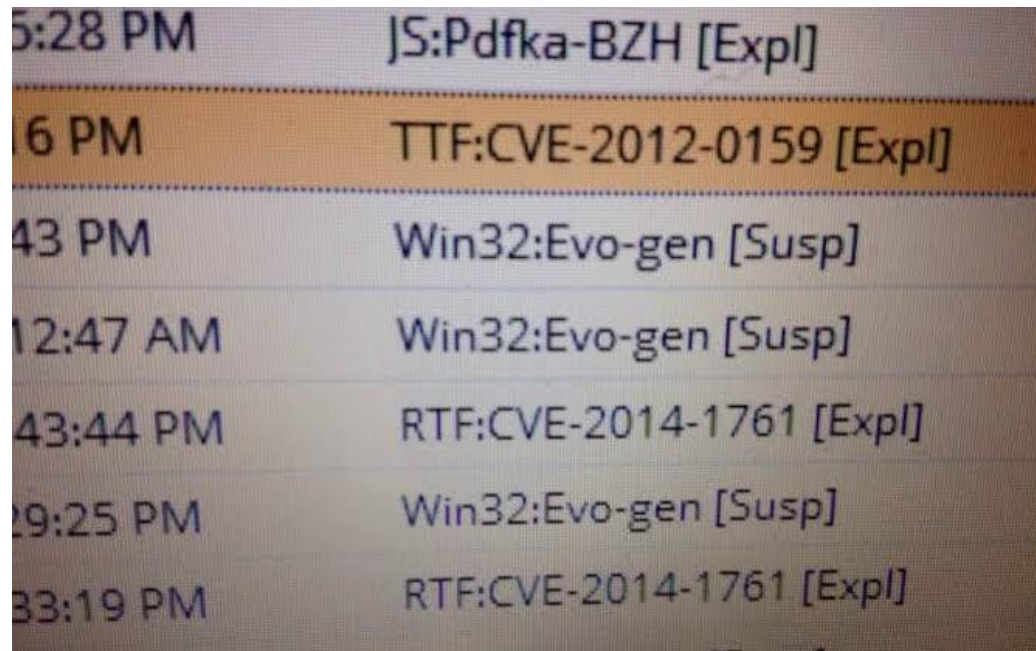


- Part-time fuzzing in our spare time (total fuzzing time = less than 48 hours)



- Two different small TTF samples
- Targeted MS Office 2013 & Oracle Java

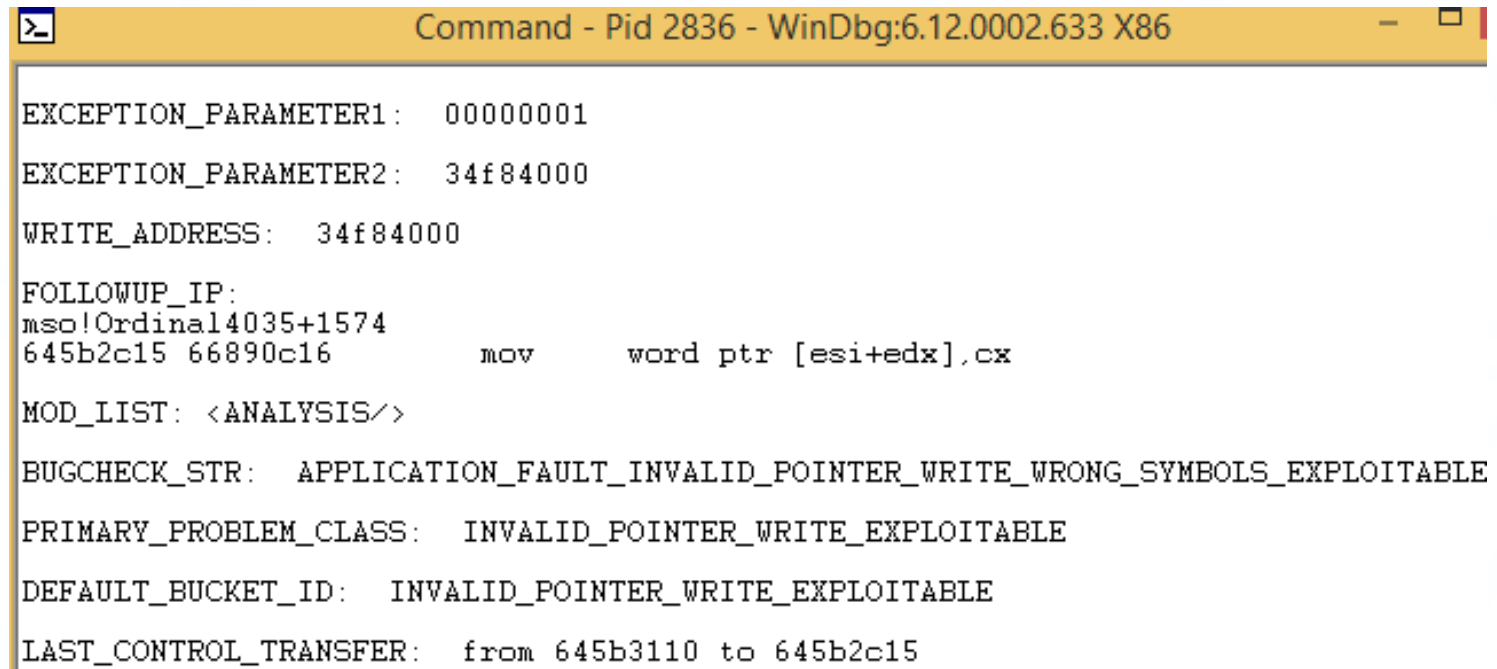
- Disable your antivirus during fuzzing
- It can re-discover some old vulnerabilities 😊



5:28 PM	JS:Pdfka-BZH [Expl]
6 PM	TTF:CVE-2012-0159 [Expl]
43 PM	Win32:Evo-gen [Susp]
12:47 AM	Win32:Evo-gen [Susp]
43:44 PM	RTF:CVE-2014-1761 [Expl]
29:25 PM	Win32:Evo-gen [Susp]
33:19 PM	RTF:CVE-2014-1761 [Expl]

- 4 bugs in Office 2013:
  - Two of them looks exploitable.
  - Reported to the vendor through ZDI (ZDI-CAN-3102), waiting for patch
  - Heap corruption through MSO.DLL
- 2 bugs in Java

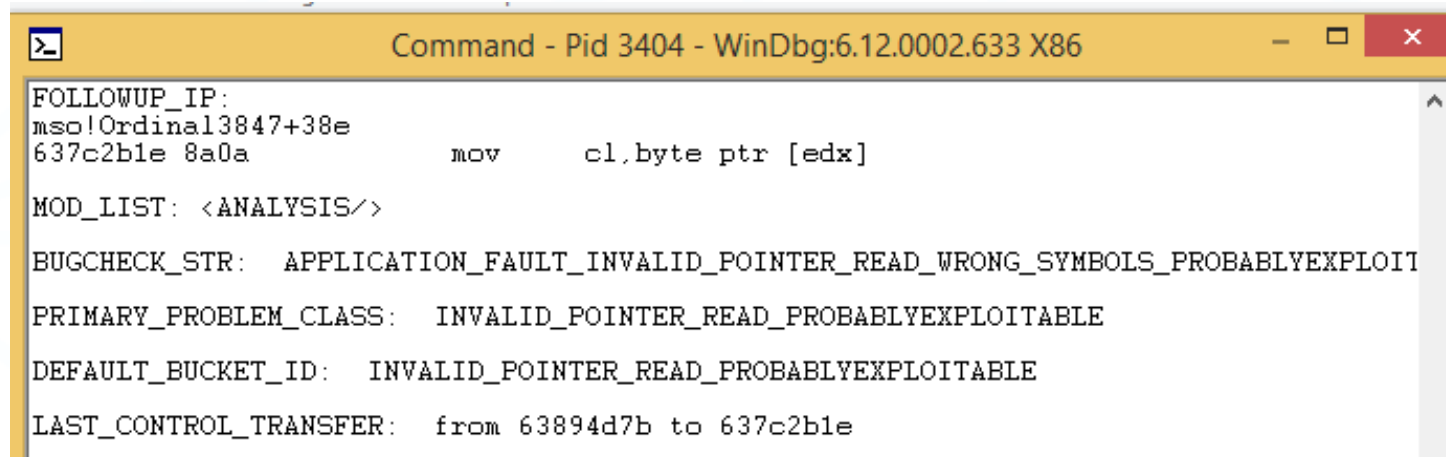
- POC#1 – MS Office 2013



```
Command - Pid 2836 - WinDbg:6.12.0002.633 X86

EXCEPTION_PARAMETER1:  00000001
EXCEPTION_PARAMETER2:  34f84000
WRITE_ADDRESS:  34f84000
FOLLOWUP_IP:
mso!Ordinal4035+1574
645b2c15 66890c16      mov     word ptr [esi+edx],cx
MOD_LIST: <ANALYSIS/>
BUGCHECK_STR:  APPLICATION_FAULT_INVALID_POINTER_WRITE_WRONG_SYMBOLS_EXPLOITABLE
PRIMARY_PROBLEM_CLASS:  INVALID_POINTER_WRITE_EXPLOITABLE
DEFAULT_BUCKET_ID:  INVALID_POINTER_WRITE_EXPLOITABLE
LAST_CONTROL_TRANSFER:  from 645b3110 to 645b2c15
```

- POC#2 – MS Office 2013



```
Command - Pid 3404 - WinDbg:6.12.0002.633 X86

FOLLOWUP_IP:
mso!Ordinal3847+38e
637c2b1e 8a0a          mov     cl,byte ptr [edx]

MOD_LIST: <ANALYSIS/>

BUGCHECK_STR:  APPLICATION_FAULT_INVALID_POINTER_READ_WRONG_SYMBOLS_PROBABLYEXPLOITABLE

PRIMARY_PROBLEM_CLASS:  INVALID_POINTER_READ_PROBABLYEXPLOITABLE

DEFAULT_BUCKET_ID:  INVALID_POINTER_READ_PROBABLYEXPLOITABLE

LAST_CONTROL_TRANSFER:  from 63894d7b to 637c2b1e
```



- Future work:
  - Add structure of some tables in TTF for more smart fuzzing
  - Develop scripts to embed TTF into Flash etc.
  - Try fuzzing TTF parsers on Mobile (iOS).. Apple Safari supports to view PDF/DOCX

- Thanks to:
  - lcle\_vx for bug fix & previous research
  - AbdulAziz Hariri & Moti Joseph for feedbacks
  - vangelis for amazing conf



**SIGNALSEC**  
BEYOND INTELLIGENCE

# Questions?

[cunuver@signalsec.com](mailto:cunuver@signalsec.com)

[bek@signalsec.com](mailto:bek@signalsec.com)